Using Linear Programming and Divide and Conquer to Solve Large Games of Imperfect Information

Jon Parker
Department of Computer Science, Georgetown University | Department of Emergency Medicine, Johns Hopkins University
jon@cs.georgetown.edu | jparker5@jhmi.edu

Abstract
Solving games of imperfect information with linear programming took a significant leap forward when Koller, Megiddo, and Stengel (KMS) proposed an exponentially more compact way to represent two-player games of imperfect information as linear programs. Despite this substantial advancement, many recent works on solving these games rely on Counter Factual Regret Minimization (CFR) as opposed to linear programming. One reason CFR became a standard approach is that CFR is easily parallelizable whereas the linear program defined by KMS's technique is difficult to solve in parallel. Convenient parallelism made CFR more amenable to multicore computing environments and large games. This paper presents a method to parallelize the linear programming technique of KMS. The proposed iterative method divides a potentially intractable linear program representing a large game of imperfect information into many smaller linear programs. Each of the smaller LPs can be processed independently and in parallel. It is shown that the solutions to the smaller LPs interact together over multiple iterations of this algorithm to produce a strategy pair that converges to the Nash Equilibrium solution to the original undivided problem. This is the first work to propose a Dantzig-Wolfe style decomposition for solving two-player games of imperfect information.

Introduction
Algorithms specifically designed to operate in parallel are frequently used to solve dauntingly large problems. For example, the MapReduce framework (Dean & Ghemawat 2008) has parallelized the task of constructing Google's inverted index, the PFP-Growth algorithm (Li et al. 2008) quickly finds frequent patterns/itemsets for use in commercial market basket analysis and Counter Factual Regret Minimization (CFR) (Zinkevich et al. 2007) is used to approximate the Nash equilibrium solution to a form of poker with approximately $9 \cdot 10^{17}$ game states. Algorithms that can operate in parallel are useful because they allow multiple CPUs to operate on a problem at the same time and sometimes allow difficult, if not outright intractable, problems to be broken down into smaller pieces that are easily handled.

This paper presents a method to parallelize the computation of the Nash equilibrium solution to a large two-player game of imperfect information using linear programming. The proposed method:

- Decomposes a large, potentially intractable, linear program representing a two-player game of imperfect information into many smaller and more manageable linear programs.
- Iteratively updates a working feasible solution, i.e., a pair of playing strategies, to the global LP using results obtained by solving the smaller more manageable linear programs.
- Produces a strategy pair that converges to the Nash equilibrium solution of the large game despite never operating on the entire problem at any one time.
- Reduces the barrier to solving large games by enabling the use of off-the-shelf LP software and commodity computers to tackle games that otherwise would be too big for the current computing environment.
- Permits a speed-up by facilitating parallelism.
- Highlights a helpful new way to think about the game tree of a large game of imperfect information.

Related Work
Work by Koller, Megiddo, and Stengel (1994 and 1996) as well as Koller and Pfeffer (1997) made computing a Nash equilibrium solution to two-player games of imperfect information more tractable by introducing a superior way to represent those games as linear programs. They showed how to write a linear program that was exponentially more
compact than the best previously known method for expressing a two-player game of imperfect information as a linear program. Their insight was to redefine the game's independent decision constraints in terms of interconnected "realization probabilities". This reinterpretation added helpful structure to the problem by chaining probabilistic choices made at early decision nodes to probabilistic choices made at later decision node and eventually directly to the payoffs at each of the games terminal nodes. This reinterpretation converts many independent constraints (that ensure the actions taken at a single decision node reflect a correctly defined probability distribution) to a monolithic set of nested constraints that together define probabilistic weights for all possible terminal nodes in a game tree. These probabilistic weights are paired with the payoff at their corresponding terminal node to enable solving the game. An example of this reinterpretation of constraints is shown in Figure 1. The extremely compact linear program written using the chained constraints could then be solved with standard LP techniques. One downside of their approach is that a LP written this way is not easily broken into smaller problems using the Dantzig-Wolfe decomposition (Dantzig & Wolfe 1960) which requires a special problem structure.

The Dantzig-Wolfe decomposition is frequently introduced in a commercial context (Bradley, Hax, & Magnanti 1977 and Bazaraa, Jarvis, & Sherali 2011). There is usually a large company with many divisions and one global profit function. The divisions operate completely independently with the exception of a few constraints that restrict the entire company and its component divisions. The Dantzig-Wolfe decomposition is depicted as a mechanism that maximizes global profit by enabling the independent divisions to coordinate despite being completely independent. The coordination is achieved through prices the home-office charges its component divisions for the raw materials they require to produce outputs (sometimes referred to as proposals). Over multiple iterations of the algorithm the prices for raw materials stabilizes along with the outputs each division produces. This process solves the linear program reflecting the entire company without ever creating the LP that reflected that large, more difficult, problem.

A helpful insight for solving games of imperfect information with a Dantzig-Wolfe style decomposition comes from the mindset of expert poker players. Many expert poker players think about positions in the game tree differently than the traditional definition of a game state would suggest. In a traditionally defined game state player 1 holds exactly hand X and player 2 holds exactly hand Y. There are also multiple information sets obfuscating the situation. However, an expert poker player is likely to a view a position in the game tree as a situation in which both players know that player 1 holds a distribution of hands while player 2 holds a second distribution of hands (see Figure 2). These

![Figure 1: An example of rewriting linear programming constraints that enforce probabilities (left) as nested constraints that enforce realization probabilities.](image1)

![Figure 2: An illustration of the "collapsed" game tree the expert poker player sees. In this collapsed view there is a single joint hand range in-play at each game state.](image2)
hand distributions are referred to as hand ranges in a poker player's vernacular and correspond to a weighted set of game states in the traditional game tree. It is as if the expert poker player ignores her/his actual hand and the various information sets because the real poker game is about deciding how to play a complete range of hands not just one hand in isolation. Thus, in practice, deciding how to play the hand held at that particular moment in time (i.e., hand X) only occurs after a good decision for all possible hands has been made in light of the joint hand range both players share. Evidence of this point of view can be found in a prominent essay by poker pro Phil Galfond (Galfond 2007) and the books *Let there be Range* (South & Nguyen 2008) and *The Mathematics of Poker* (Chen & Ankenman 2006). This higher level point of view is useful academically because it facilitates a Dantzig-Wolfe style decomposition of the linear program that represents the poker game tree. This decomposition exchanges joint hand ranges and payoff matrices as opposed to raw material prices and business division outputs.

One of the first teams to apply the techniques of Koller, Megiddo, and Stengel (1994 and 1996) to producing computer programs capable of playing heads-up Limit Texas Hold'em (a form of poker with 9·10^{17} game states) at a high level was Billings et al. (2003). This group even held a "Man vs. Machine" match (Moseman 2008 and Phys.org 2008) where the poker playing program Polaris defeated its human opponents. Eventually, this team abandoned LP based solutions in favor of the parallelizable Counterfactual Regret Minimization (CFR) algorithm (Zinkevich et al 2007). They used CFR to produce poker playing programs (Johanson et al. 2012) that handily defeated the programs from Billings et al. (2003). It is interesting to observe that CFR works by relying solely on local information (regret at a particular information set). This suggests a decomposition based approach is helpful.

A recent work by Pays (2014) pushed LP methods further in solving large two-player games of imperfect information by using a specialized interior point method to solve an abstracted version of 2 player Limit Texas Hold'em. One solved game tree contained 1.3·10^{9} game states (abstracted down from 9·10^{17} game states). This impressive feat was accomplished by fitting the entire game's LP, which had been abstracted and manipulated to fit the method, into the LP solver at once. It is just this type of method that can benefit from the observation that the LP systems proposed by KMS (which is Pays' starting point) can be decomposed. The method Pays used need not tackle the entire LP at once. Thus, it could be pushed further if desired.

**Method**

The method proposed in this paper uses divide and conquer and linear programming to find an approximate Nash equilibrium solution to a large, potentially intractable, two-player game of imperfect information. The algorithm begins by finding an initial feasible solution to the complete game. This step is straightforward and unencumbered by game size because a feasible solution can be found by requiring all probabilistic playing choices to be made uniformly among available options. Next, the variables in the global linear program are partitioned into n sets. If the partitioning is done appropriately it will enable the creation of n separate linear programs that are significantly smaller than the original LP because they treat the variables outside their assigned partition as constants (whose values are taken from the initial feasible solution). These smaller LPs are then solved independently and in parallel. The initial feasible solution is then improved by merging it with solutions to the n smaller LPs. The algorithm continues in this manner, repeatedly improving the working feasible solution (i.e., pair of playing strategies), until it has converged to the game's Nash equilibrium.

**Partitioning the Problem**

The goal of partitioning is to create multiple smaller linear programs that answer the question: "What is the solution for this fraction of the game assuming play in the other fractions cannot change?" Of course, the feasible solution to an arbitrary linear program cannot usually be improved with an iterative "partition and solve separately" approach. A feasible solution can get stuck at a suboptimal point if variables in a tight constraint or group of constraints are allocated to different partitions. In such a situation no individual sub-LP will be able to improve the current feasible solution because no sub-LP can obtain the required slack by changing variables outside its partition. Fortunately, the LPs defining ex-

![Figure 3](image-url)
tensive form games can avoid this problem because the constraints of these programs govern actions taken at completely independent decision nodes. This independence allows us to unravel the artificial structure added by KMS's chaining technique that appears to prevent partitioning.

**Partitioning by Player**

We begin by discussing a modest two-way partitioning method where the first LP solves for player 1's best possible strategy assuming player 2's strategy is fixed and the second LP solves for player 2's best possible strategy assuming player 1's strategy is fixed. Figure 3 depicts an iterative algorithm for finding a Nash Equilibrium with these two LPs.

Koller, Megiddo, & Von Stengel (1994) give the pair of linear programs, shown in Figure 4, that solve for each player's best response strategy. Notice, the upper LP treats player 2's strategy (the \( y \) vector) as fixed while the lower LP treats player 1's strategy (the \( x \) vector) as fixed. The matrices \( A \) and \( B \) contain the game payoffs (from each player's point of view). While the matrix and vector pairs \((E, e)\) and \((F, f)\) ensure each player's overall strategy defines realization probabilities that weight each of the games terminal nodes. More information about these matrices and vectors can be found in (Koller, Megiddo, & Von Stengel 1994).

It should be noted that partitioning the game's variables by player and iteratively updating the current strategy pair is reminiscent of McMahan, Gordon, & Blum's Double Oracle Method (2003). Their method provably converges to a game's Nash Equilibrium but is intended for matrix games where there are relatively few pure strategies. The strategy updating rule for the iterative algorithm proposed here (see below) is more conservative than the updating rule used in the Double Oracle Method because preventing overfitting to the opponent's current playing strategy is a more important when solving large extensive form games as opposed to small matrix games.

**Partitioning by Time**

A more aggressive partitioning option is to build linear programs that reflect separate sub-trees of the game. Solving independent sub-trees has been successfully applied to large games of perfect information like checkers which is solved despite having \(5 \cdot 10^{20}\) game states (Schaeffer 2007). Unfortunately, solving sub-trees is less straightforward for games of imperfect information because the optimal strategy inside a sub-tree may change depending on the strategy used outside the sub-tree. In other words, sub-trees in imperfect information games may not be independent therefore computing a globally optimal strategy for sub-tree usually requires multiple iterations of an algorithm that interacts with the rest of the game tree.

In this context, it is useful to think of sub-trees as an expert poker player might (see Figure 2). In this frame of reference a sub-tree is rooted in a single non-traditional game state that contains a public joint probability distribution

![Figure 4: Two linear programs that solve for a player's best response assuming her/his opponent's strategy cannot change.](image1)

![Figure 5: Similarity between information exchange during a Dantzig-Wolfe LP decomposition (upper portion) and the proposed method (lower portion).](image2)
An alternative, and more traditional, viewpoint is given in (Burch et al. 2014) where a subgame is defined as "a forest of trees, closed under both the descendant relation and membership within augmented information sets for any player." Despite their equivalence, the expert poker player's "collapsed game tree" point of view is in some ways preferable to the "forest of trees" point of view because: (1) Decomposing the problem is easier to think about when the subgame is rooted in exactly one node and information set. (2) The joint probability distribution at a subgame's root node is naturally interpreted as the subgame's starting resources. (3) The joint probability distribution and payoff matrix at each subgame suggest a decomposition that closely parallels the economic example frequently used to explain the Dantzig-Wolfe decomposition.

Figure 5 illustrates the similarity between the Dantzig-Wolfe decomposition and the proposed algorithm. However, the similarity is not perfect. When the iterative Dantzig-Wolfe (DW) process determines proposed outputs (in the Figure's lower boxes) they are the direct result of solving a smaller sub-LP. The proposed outputs computed at any iteration of the DW algorithm reflect only the most recent input prices; they do not reflect the "history" of the process in any way. On the other hand, the proposed iterative algorithm exchanges joint probability distributions and payoff matrices that reflect the history of the system. The history of the algorithm is incorporated by setting the updated playing strategy pair to be a combination of the prior strategy pair and the most recently computed best possible strategy pair.

The purpose of integrating the old solution into the new solution is to inhibit cycling and overfitting. Consider the game paper-rock-scissors. The Nash equilibrium strategy for both players is to play each option with probability 1/3rd. However, if the opponent plays almost perfectly (i.e., by playing (.33, .33, .34)) the optimal response is wildly different from the Nash Equilibrium (i.e., the best response is (0, 1, 0)). Thus, to inhibit an endless cycle of overfitting the new strategy always mixes in a substantial amount of the old strategy.

**Updating the Playing Strategies/Feasible Solution**

The algorithm presented here iteratively updates playing strategies for both players, i.e., the working feasible solution to the full linear program, until the current strategy pair converges to an approximate Nash equilibrium of the game in question. This updating process never jettisons the old playing strategies entirely. Instead new playing strategies are obtained by mixing the last playing strategies with the locally optimal playing strategies found within each partition. Overwriting/forgetting the old playing strategies completely would allow the current strategy pair to oscillate around an equilibrium. The proposed updating rule is shown in Figure 7.

\[
Play(n + 1) = \lambda \cdot Play(n) + (1 - \lambda) \cdot BR
\]

where: \( \lambda = \frac{nc}{1 + nc}, \) \( 0 < c \leq 1, \) and \( n = \) iteration number

![Figure 6: An illustrated view of the proposed algorithm when the LP representing the complete game is partitioned by time.](image)

7. Varying the constant \( c \) changes how strongly the most recent optimal strategies effects the updated playing strategies. When \( c \) is set to 1 the new playing strategies are the average of all past optimal strategies as well as the initial feasible solution. This maximum possible \( c \) value strongly inhibits overfitting and cycling but also prevents faster convergence when possible (it also closely mimics traditional CFR's updating rule). A \( c \) value of .5 to .75 works well empirically by increasing the rate at which a strategy pair can converge to the globally optimal value while also dampening overfitting.
Validation
Small scale experiments (i.e., software unit tests) were performed to demonstrate that the partitioning methods introduced above (i.e., partitioning by player and partitioning by time) work as claimed. These experiments were performed only to test/debug software developed for use during the 2014 Annual Computer Poker Competition. These experiments were not performed to demonstrate the correctness of the approach and do not constitute a mathematical proof of any kind.

The "partitioning by player" approach was tested by confirming that solving the preflop phase of Limit Texas Hold'em using the "partitioning by player" approach generates the same solution as does traditional CFR. Similarly, the "partitioning by time" approach was tested by confirming that solving the preflop and flop phases of Limit Texas Hold'em by exchanging payoff matrices and joint hand-ranges between the preflop and flop layers (as the "partitioning by time" approach requires) generates the same solution as does traditional CFR.

Discussion
The decomposition presented here relies on two core insights. The first is a new way of looking at a game tree that suggests easily comprehensible objects to pass between upper and lower sub-LPs (i.e., joint hand ranges and payoff matrices in the poker example). The second insight is that the monolithic set of interconnect constraints proposed by KMS that appears to prevent decomposition can be "short circuited" by starting with an intermediate joint probability distribution or ending an intermediate payoff matrix.

This decomposition reduces the barrier to solving exceptionally large two-player games of imperfect information like full scale poker because repeatedly solving each of the significantly smaller sub-LPs is doable while solving the undivided LP is difficult due to memory constraints. It is important to point out that reducing the size of the LPs being solved may enable the use of off-the-self LP software and cloud computing environments like Amazon's EC2 and Microsoft's Azure. Consequently, this approach permits a speed-up by facilitating parallelism using a framework like MapReduce.

A downside to this decomposition is that the stopping rule is more involved than the single LP case because the solution asymptotes towards the Nash-Equilibrium it will not reach it exactly (assuming the initial feasible solution was not an equilibrium). Additionally, each of the sub-LPs must be resolved multiple times, each time reflecting a slightly different input joint probability distribution or set of downstream payoff matrices. It is obvious that the proposed iterative method will converge to a global equilibrium but the rate of convergence will depend on the nature of the game and the point at which the upper/trunk sub-LP is demarcated from lower sub-LPs.

Conclusion
This paper presents a way to decompose a linear program representing a two-player game of incomplete information into many smaller linear programs. The approach to decomposition closely parallels the Dantzig-Wolfe decomposition. However, the Dantzig-Wolfe decomposition itself cannot be applied due to the structure necessary to write the linear program compactly. The key to the proposed decomposition method is a simplifying way to think about games of imperfect information adopted by expert poker players that has not previously been leveraged to decompose an LP representing a two-player game of imperfect information. Once the smaller linear programs are constructed they can be processed in a highly parallel manner using a framework like MapReduce. Additionally, the smaller linear programs are easier to solve with off-the-shelf linear programming software/expertise because they do not push the boundary of computability.

References


