Advanced Games Programming (AI)
Part 5: Imperfect Information Games

Michael Buro

December 23, 2024

[Under Construction]

Change Log

Outline

[AI-Lec 15 L24] Introduction

Thus far we have concentrated on single- and two-player perfect information games, in which players know the state of the game at all times

However, in real-world applications this condition is often violated. Consider for instance the stock-market, Poker, or RTS games. In those games, participants often have different information

The question becomes how to play imperfect information games well

This question is addressed by the mathematical field of game theory which studies strategic decision making and was founded by John von Neumann with his seminal 1928 paper “Zur Theorie der Gesellschaftsspiele” (Literally: On the theory of parlour games)

Introduction (continued)

Today, game theory is a thriving field with lots of applications ranging from building rational agents for complex tasks to trying to understand irrational human behaviour (see for instance “Predictably Irrational” by Dan Ariely)

As it turns out, computing good moves in arbitrary imperfect information games according to our current understanding is intractable, i.e., we only know exponential time algorithms, which are impracticable

An exception are two-player zero-sum games for which in each game episode the payoffs for both players always add up to 0. For such games, computing moves is tractable (i.e., in polynomial time)

Introduction (continued)

As we will see shortly, imperfect information games require us to consider randomized strategies

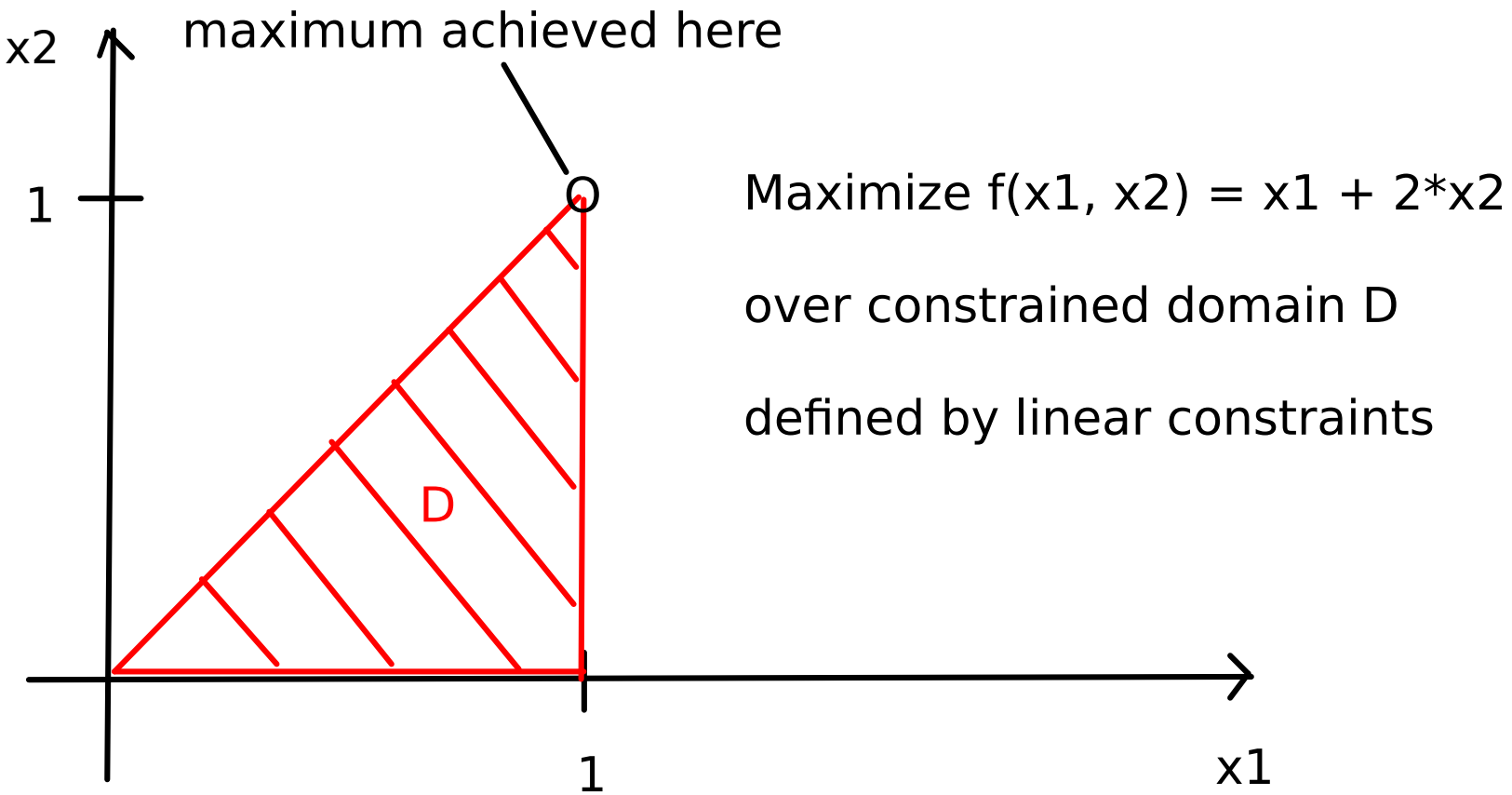
In the brief remaining time we will motivate a solution concept based on linear programming and give an example of how solving imperfect information games can be used to improve RTS game AI

For a more thorough description of the field of game theory I recommend visiting

`http://en.wikipedia.org/wiki/Game_theory`

Background: Linear Programming

The best-known category of constraint optimization problems is **linear programming** (LP), in which constraints must be linear inequalities which represent half-spaces whose intersection forms a **convex set**, and the objective function is also linear. For instance:



Linear Program Example

Problem:

$$\text{maximize } x_1 + 2x_2 \text{ subject to } x_1 \leq 1,\ x_2 \leq x_1,\ x_2 \geq 0$$

Domain D is defined by 3 linear inequalities, each describing a half-plane. Their intersection forms a triangle

Solution:

$x_1 = x_2 = 1$ and the maximum value of the objective function is 3

Linear Programming (continued)

All linear programs can be written in the following normal form:

$$\begin{array}{ll} \text{maximize} & c^t x \\ \text{subject to} & Ax \leq b \\ \text{and} & x \geq 0 \end{array}$$

where b, c are constant column vectors, t denotes transposition, A is a constant matrix, x is the unknown column vector subject to optimization, and the inequalities hold component-wise, i.e., $x \geq 0$ means $x_1 \geq 0, ..., x_n \geq 0$

[Exercise: Determine A, b, c for the previous example]

Solving Linear Programs

It is easy to see that without loss of generality optimal values will be achieved in corners of domain D, of which there can be an exponential number depending on the number of variables

The **Simplex Algorithm** hops from corner to corner to find the optimal solution. It can take exponential time in the worst case, but runs much faster in many applications

In 1984 Kamarkar proved that LPs can be solved in polynomial time in the input size (number of variables plus constraints), using a so-called **interior-point method** which starts somewhere in the interior of the domain set and then incrementally walks towards the optimal constraint corner

Solving LPs (continued)

CPLEX is a fast commercial solver which uses an interior point method, which iteratively approaches the max point from the interior of the feasible region

GLPK (free software GNU Linear Programming Kit) works well for moderate input sizes

As we will see shortly, two-player zero-sum imperfect information games can be solved using LPs. More general payoff structures lead to Linear Complimentary Problems (LCPs) which are harder to solve

Convex Optimization

LPs are probably the most widely studied and broadly useful class of optimization problems

They are a special case of the more general problem of convex optimization, which allows the constraint region to be any convex region and the objective to be any function that is convex within the constraint region

Under certain conditions, convex optimization problems are solvable in polynomial time, scaling up to thousands of variables

Several important problems in machine learning and control theory can be formulated as convex optimization problems

Matrix Games

Recap:

- Two players
- Payoff matrices A, B (payoffs for players 1 and 2)
- Simultaneous moves: player 1 picks row i, player 2 picks column j
- Player 1 receives amount A_{ij}, player 2 receives B_{ij}

Example: Rock-Paper-Scissors

A: payoff for P1

P1 \ P2	R	P	S
R	0	−1	+1
P	+1	0	−1
S	−1	+1	0

B: payoff for P2

P1 \ P2	R	P	S
R	0	+1	−1
P	−1	0	+1
S	+1	−1	0

This is a zero-sum game (payoffs add up to 0, i.e., $B = -A$, so we only need to consider one matrix, say A)

How to Play RPS?

Deterministically?

Whenever someone announces to use a deterministic ("pure") strategy the opponent can win all the time

Try it!

What else can we do?

Randomize actions ("mix" strategies)

Prob(Rock) = $p_R \in [0, 1]$

Prob(Paper) = $p_P \in [0, 1]$

Prob(Scissors) = $p_S = 1 - p_R - p_P$

What if the opponent announces a strategy with one probability $\neq 1/3$? Can you beat that strategy on average?

How to Play RPS? (continued)

Yes. If one probability is $\neq 1/3$, consider a move m with the highest probability (must be $> 1/3$ – why?)

Then we just always play the move that defeats m

$\rightsquigarrow$ Our expected payoff > 0

So, what's left is $p_R = p_P = p_S = 1/3$

This is the only so-called MiniMax strategy for RPS

In what follows we will see how action probabilities for MiniMax strategies can be computed from a given payoff matrix of a 2-player zero-sum matrix game based on linear programming

MiniMax Theorem

Theorem (John von Neumann, 1928):

For every two-person, zero-sum game with finitely many pure strategies, there exists a value V and a mixed strategy for each player, such that

(a) Given MIN's strategy, the best payoff possible for MAX is V, and

(b) Given MAX's strategy, the best payoff possible for MIN is $-V$

Hence these mixed strategies are optimal strategies for both players in the paranoid sense when assuming the opponent always choses a best strategy

Generalization

More general:

Definition:

A tuple of strategies (also known as strategy profile) forms a Nash-equilibrium iff no unilateral strategy deviation beneficial to a single player exists

John Nash in his seminal 1950 paper proved that such equilibria exist for ALL finite *n*-player games, not just zero-sum, 2 player games

MiniMax strategies form a Nash-equilibrium for 2-player zero-sum games

MiniMax Strategies

MiniMax strategies in the zero-sum, 2 player case

A mixed strategy is a probability distribution over all players' choices (i.e., column vector of numbers ≥ 0 adding up to 1)

Suppose player MAX (picking row i) uses mixed strategy x and player MIN uses y (picking column j)

What is the expected payoff for MAX, assuming MAX's payoff is A_{ij}?

MiniMax Strategies (continued)

If MAX picks row i and MIN picks column j, the result is A_{ij}

This happens with probability $x_i \cdot y_j$

Thus for those two choices the expected result is

$$x_i \cdot A_{ij} \cdot y_j$$

Summing over all choice pairs leads to the expected payoff for MAX:

$$\sum_{i,j} x_i \; A_{ij} \; y_j \;=\; x^t A y \qquad (t = \text{transposed})$$

Best Response Strategies

Strategy x is called a best response strategy for MAX for a fixed y if it maximizes

$$x^t(Ay)$$

Likewise, a best response strategy y for MIN minimizes

$$(x^tA)y$$

for fixed x

The best response value for MAX given y can be written as follows:

$$\max_x \; x^t(Ay)$$

subject to $\sum_i x_i = 1$ and $x_i \geq 0$ for all i

Best Response Strategies (continued)

The constraints mean that x is a probability distribution

MIN tries to minimize MAX's value, and his optimization problem is analogous

Problem: this is a quadratic program (QP, maximizing a quadratric function subject to linear constraints), but not an LP

Recall: LPs have the following form:

$$\max_x \; c^t \cdot x \text{ subject to } Ax \leq b \text{ and } x \geq 0$$

where c, b, x are column vectors, A is a matrix, and $\leq, \geq$ apply to all vector components

I.e., we try to maximize a linear function over a set defined by linear constraints

Best Response Strategies (continued)

The MiniMax theorem says:

$$\min_y \max_x \; x^t(Ay) = \max_x \min_y \; (x^t A)y$$

Not quite an LP. But note that

$$\min_y \; (x^t A)y$$

for a given strategy x can be achieved by a pure strategy y (assigning probability 1 to one move). So we don't have to mix. Why?

$(x^t A)$ is a row vector and MIN simply chooses the single best counter move and assigns it probability 1. Mixtures can't be better

E.g., $(x^t A)y = (-1, 2, -5, 3)\; y$
$\rightsquigarrow$ pick $y_3 = 1$ and all others $= 0$
$\rightsquigarrow$ worst value for MAX is -5

Computing Nash-Strategy Profiles

With this insight we can rewrite the MinMax optimization as a linear program which optimizes a linear function subject to linear constraints:

$$\max_x \min_y \; x^t(Ay) = \max_x \min_j \; (x^t A)_j \quad (+)$$

subject to x being a distribution: $x \geq 0$ and $\sum_i x_i = 1$

Still not an LP yet, i.e., not of the form $(*)$:

$$\max_x \; c^t \cdot x \text{ subject to } Ax \leq b \text{ and } x \geq 0$$

Can you find an LP formulation for $\min_j \; (x^t A)_j$ for a fixed x?

Computing Nash-Strategy Profiles (continued)

The right-hand side of $(+)$ is equal to

$$\max_{z,x} z$$

$$\text{subject to } z \leq (x^t A)_j \text{ for all } j, x \geq 0, \text{and} \sum_i x_i = 1$$

Why? Because for a given x,

$$\min_j (x^t A)_j = \max_z z \text{ for } z \leq (x^t A)_j \text{ for all } j$$

This is an LP!

(Exercise: prove this by choosing corresponding values for c, A, and b in the LP normal form $(*)$)

Computing Nash-Strategy Profiles (continued)

The analogous LP for MIN is:

$$\min_{z,y} z$$

$$\text{subject to } z \geq (Ay)_i \text{ for all } i, y \geq 0, \text{and} \sum_j y_j = 1$$

This means that we can compute the game value z and move distributions x and y by solving above LPs

Example: RPS

```
      R    P    S    y (j)
   +----------
R  |   0   -1    1
P  |  +1    0   -1
S  |  -1   +1    0
x (i)
```

For MAX's strategy solve:

$\max_{z,x} z$ subject to
$x_1 + x_2 + x_3 = 1$
$x_1, x_2, x_3 \geq 0$
$z \leq x_2 - x_3$ value for move y_1 (col 1)
$z \leq -x_1 + x_3$ value for move y_2 (col 2)
$z \leq x_1 - x_2$ value for move y_3 (col 3)

Solving RPS

Each z constraint describes the payoff against a pure MIN strategy. MAX wants to maximize this value

Claim: $z = 0$ and $x_1 = x_2 = x_3 = 1/3$

Proof: Adding all constraints shows $z \leq 0$. Can $z = 0$ be reached?

Yes, by choosing $x_i = 1/3$

All other choices lead to $z < 0$

Online Solver

Try `https://www.math.ucla.edu/~tom/gamesolve.html` to see an LP based matrix game solver in action

E.g.,

```
   A: payoff for P1

        R    P    S - P2
     +-----------
R    |   0   -2   +1
P    |  +2    0   -1
S    |  -1   +1    0
|
P1
```

The Nash-equilibrium is $x = y = (\frac{1}{4}, \frac{1}{4}, \frac{1}{2})$ and the game value is 0

MiniMax Strategies in More Complex Games

Optional reading – material will not appear on any test

How to find MiniMax strategies in more complex imperfect information games?

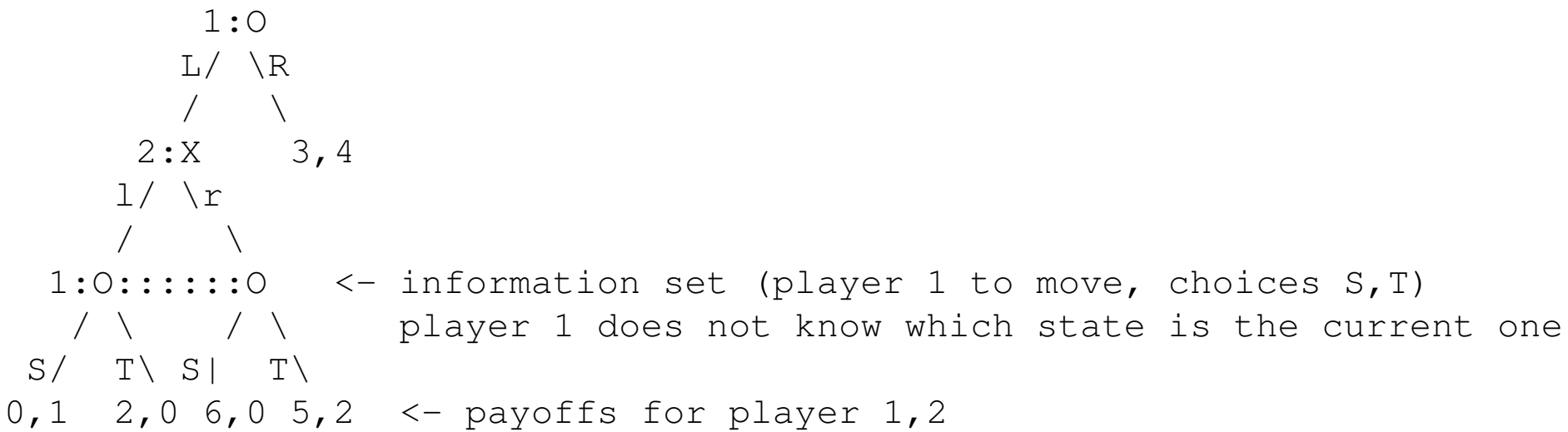
By generalizing the maxtrix game!

But first, how do “game trees” for imperfect information games look?

Extensive Form Games

We have seen matrix game definitions earlier. This representation is called strategic form or normal form

Alternatively, the extensive form game representation consists of a game tree and an information set overlay:



Solving 2-Player Zero-Sum Extensive Form Games

Imagine all possible pure strategies in a finite 2-player zero-sum game, i.e., specify a move in every information set where the player is to move

Perfect information case:

Overlay both trees $\leadsto$ principal variation $\leadsto$ leaf value determines payoff

Imperfect information case:

Create payoff matrix, where a "move" is now a pure strategy

Payoff entry i, j is the value of pitting strategy i vs. j

Mixed MiniMax strategy = mixture of pure strategies

Solving Extensive Form Games (continued)

Distributions can be computed by solving above LPs

That's a huge matrix!

(# of pure strategies for MAX) · (# of pure strategies for MIN)

# of strategies usually exponential in the game tree size

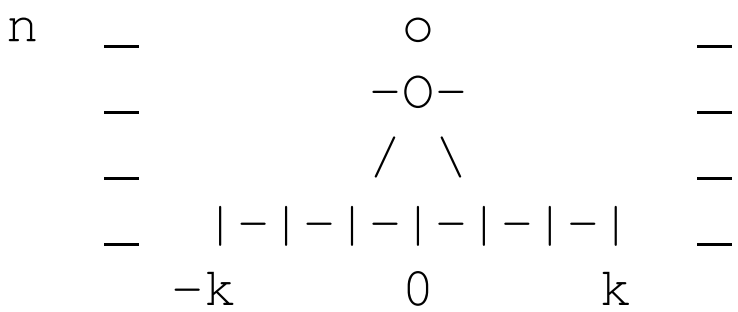
$\rightsquigarrow$ doubly exponential in game tree depth!

But there is a lot of redundancy in this LP (e.g., when moving left, all our choices in the right subtree are irrelevant)

There exists an algorithm [1,2] based on the so-called sequence form that computes MiniMax strategies in time polynomial in the size of the game tree!

BIG IMPROVEMENT, which led to the first competetive Poker bots

Applications: Oshi-Zumo



Two players, each start with n coins. Sumo wrestler in the middle of 2k+1 locations

In each round, players secretly bid any amount of coins they hold. The player who bid more pushes the Sumo one position towards the other player's side, both players pay in any case (in case of a tie the wrestler does not move)

The game is over when the wrestler is pushed over the edge, both players bid 0, or money runs out. In these cases the half the wrestler resides in determines the loser

Oshi-Zumo (continued)

The game defines a tree of matrix games which can be solved bottom-up by dynamic programming [3]:

- Start with base cases:
  (0 vs. 0 coins) x (any position -(k+1)..(k+1))
- Iterate until no more unsolved positions exist:
  - Look at all positions whose values haven't been computed yet
  - Determine whether all successor values computed
  - If so, compute value by solving LP

Example: k = 3 (7 Sumo locations)

Smallest position that requires strategy mixing:

```
      5 vs 2 coins, Sumo at -3  [5,2,-3]
      *
p1 5  |-|-|-|-|-|-| 2 p2
     -3     0     3

bids:   p1  prob p2
         1  0.5  0
         2  0.5  2

expected value for p1:
  (1,0) 25%   -> [4,2,-2] draw (p1 bids 1)
  (1,2) 25%   -> [4,0,-4] instant loss          => expected value -0.25
  (2,0) 25%   -> [3,2,-2] loss (p2 bids 1)
  (2,2) 25%   -> [3,0,-3] draw (bid 3x1)
```

All pure strategies for p1 lose more
E.g., bids (2,0) $\rightarrow$ (3,2,-2) $\rightarrow$ (*,1): p1 loses

Target Selection in RTS Games [5]

n vs. m units with attack value and hitpoints

Assume every unit can attack all others

Playing in rounds (simultaneous moves) until one side is defeated.

What is a good target ordering?

Matrix game formulation:

- rows/columns : which units do both players attack?
- matrix entry: expected value of reached state when attack actions i, j are chosen

Target Selection in RTS Games (continued)

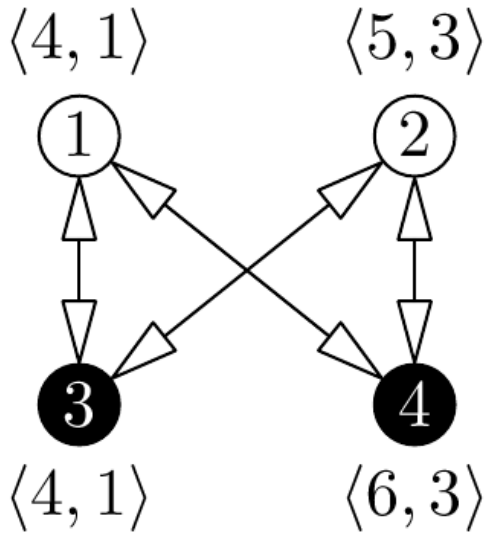
Tree of matrix games, can be solved bottom up by LP:

- Solve leaf state LPs and use their values to fill in the payoff matrix entries one level up
- Iterate, until root matrix game is solved

Large number of states, because in each level the number of move pairs is $n' \cdot m'$, where n', m' are the number of units left

Target Selection in RTS Games (continued)

Example that shows that mixed strategies are sometimes necessary [5]:



	0	1	0	1
	-1	1	1	1
	1	1	**0**	1
	0	1	**1**	1

Black: row player MAX `<hitpoints,attack>`

(entries = result of sub-games for MAX after playing (i,j), thin entries are dominated)

Target Selection in RTS Games (continued)

Playing deterministically (move 3 or 4): Black's score is 0

Play non-dominated actions with probability 0.5: Black's expected score is 0.5

We currently only know exponential time algorithms to solve this problem in general (generate tree of LPs)

High-Level Strategy Selection in RTS Games

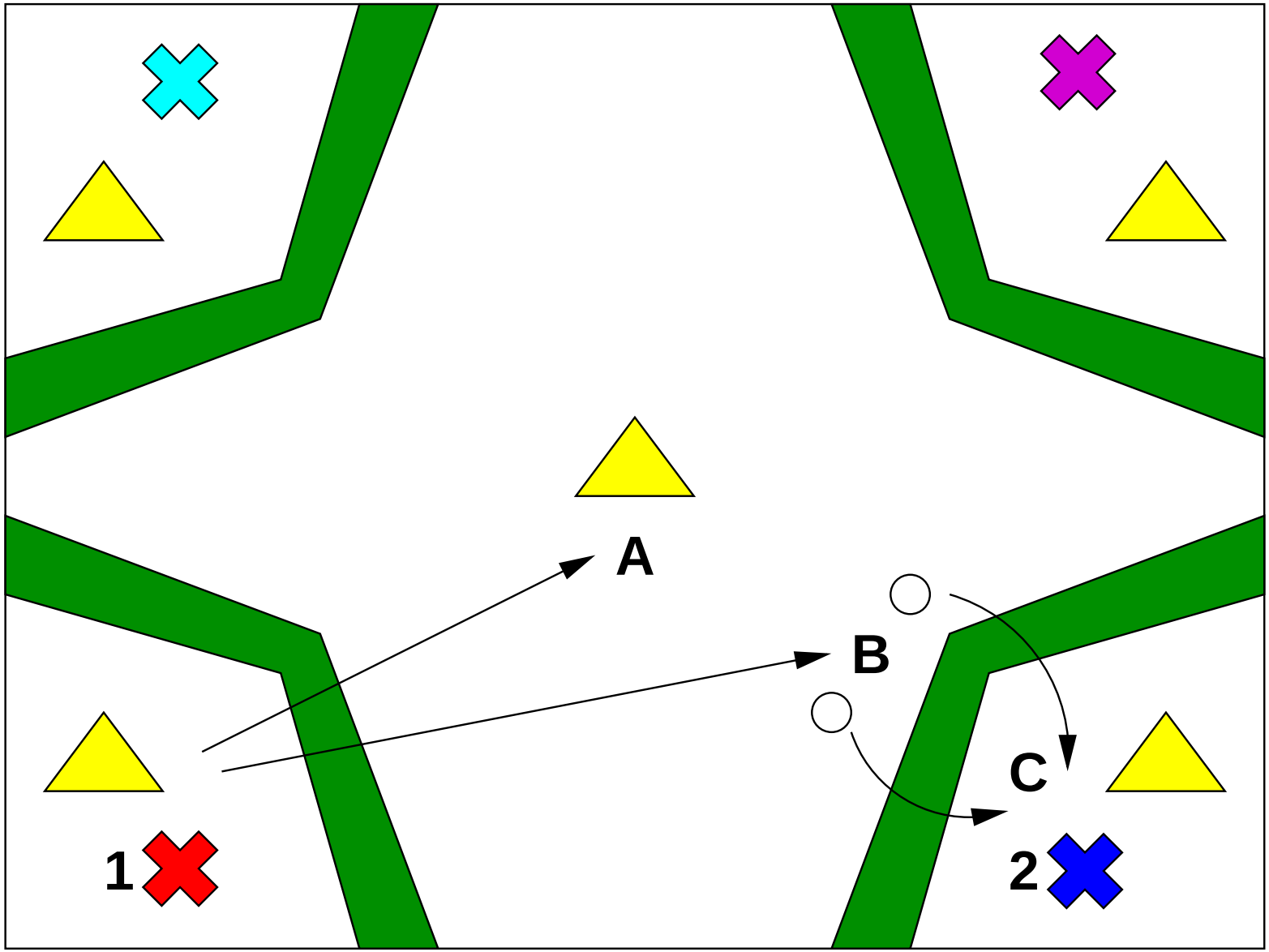
MiniMax search has been successful in chess, why not apply it to RTS games?

- Huge branching factor (100s of units)
- Imperfect information (simultaneous moves + “fog of war”)

Minimax at lowest level won’t work

We need abstractions

An RTS Game Planning Problem



RTS-Plan Idea

Moves $\equiv$ Strategies

Computing Move $\equiv$ Select Strategy

Create Strategies (“scripts”)

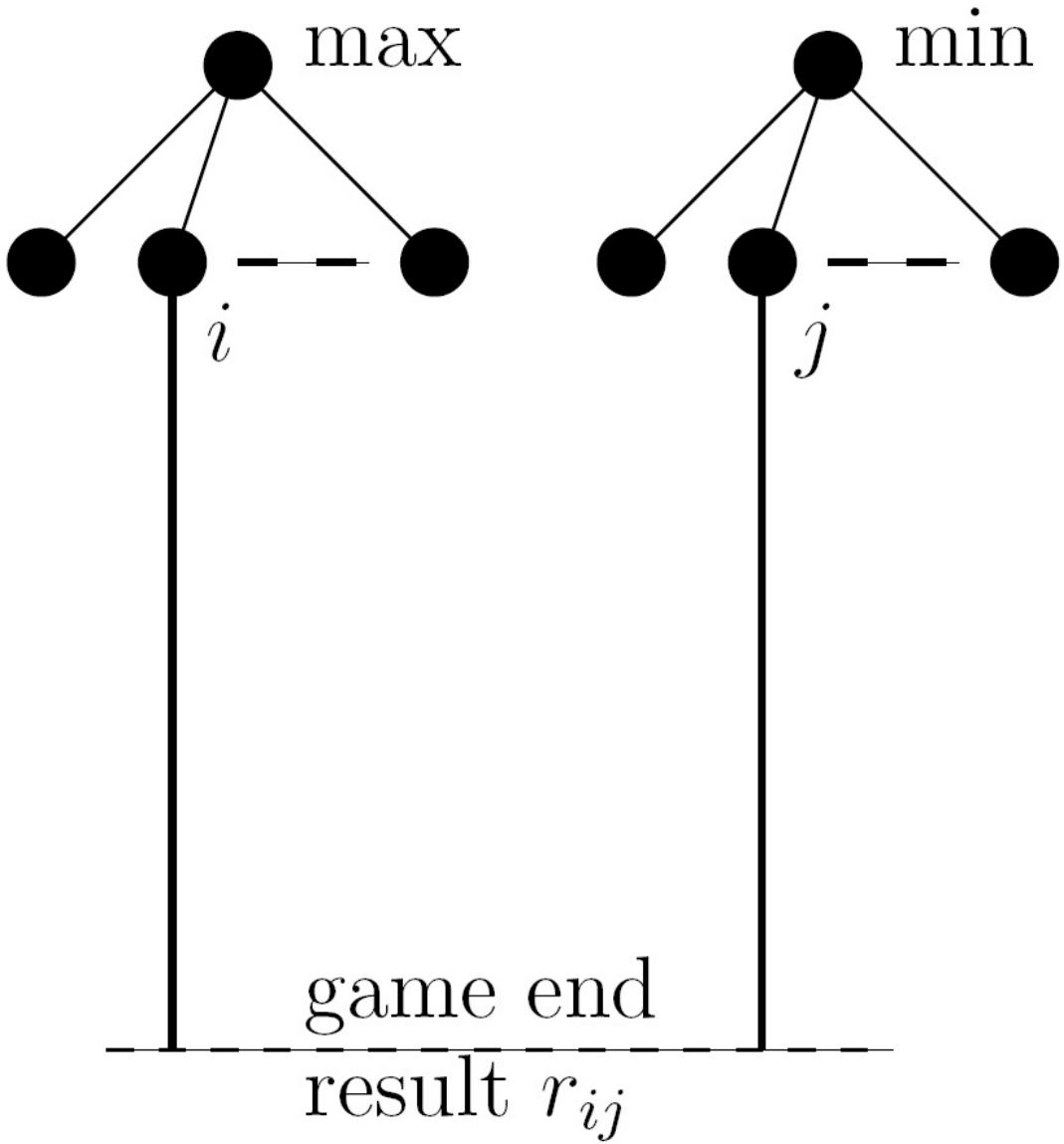
Apply traditional methods to select strategy to follow

RTS-Plan

During the game:

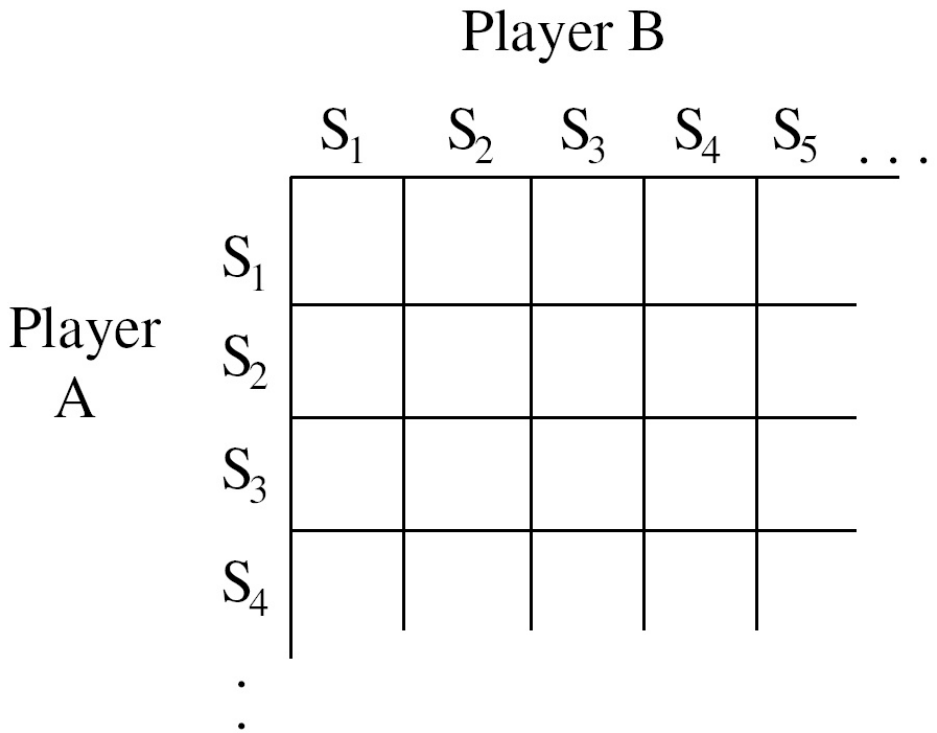
- Create payoff matrix with results of playing strategy *i* vs. *j*
- Solve LP $\rightsquigarrow$ strategy distribution
- Pick strategy accordingly, follow for a couple of seconds
- Then replan

Strategy Simulation



Payoff Matrix

Player A \ Player B	R	P	S
R	0	−1	+1
P	+1	0	−1
S	−1	+1	0



Issues

- Simulations slow + matrix to be filled!
- Simulate till end?
- Subsequent choice points would be nice
- Partial observabilty?
- What if opponent strategy isn't covered?
- Nash-equilibria don't maximally exploit
- Payoff entries may be random variables
  $\leadsto$ need to solve LPs robustly

State/Action Abstraction

Consider unit groups

Strategies issue group commands

Spread Out Matrix Computation

```
for (int i=0; i < numOurStrategies; ++i) {
  for (int j=0; j < numTheirStrategies; ++j) {

    if (!nextSimulationAllowed())
      return notDone

    r[i][j] = simulate(ourStrat[i], theirStrat[j])
  }
}
return pickStrategy(r)
```

Fast Forwarding

Advance to next “interesting” point in time

```
nextTime = min(getNextCollideTime(),
               getNextOrderDoneTime(),
               getNextShootingTime(),
               getNextStrategyTimeoutTime()
           )
```

Basic Opponent Modeling

Monitor opponent actions

Only consider “nearby” opponent strategies

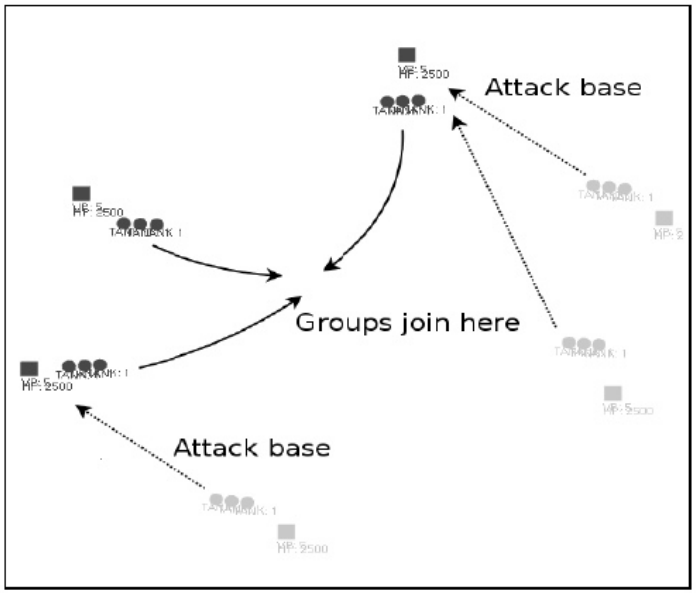
If no strategy matches, consider all strategies

Experiments

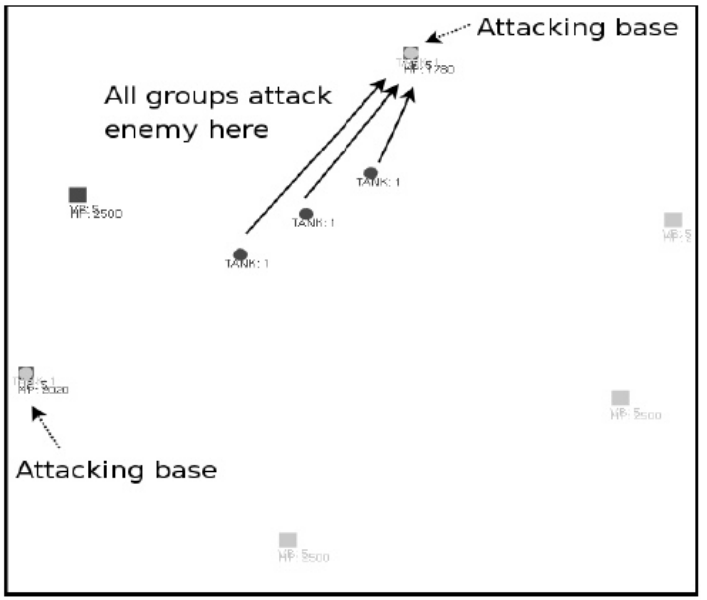
Army deployment scenarios

- 3 bases for both players
- Symmetric locations
- Flat terrain
- 3 groups of units located at each base

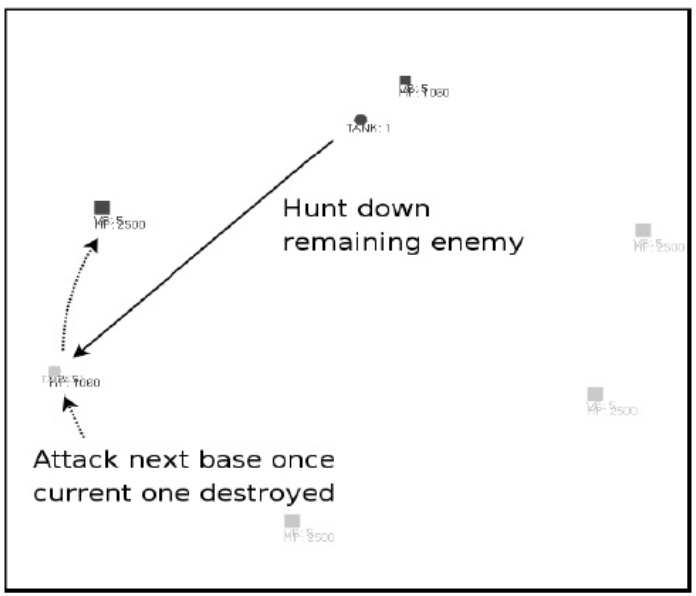
Example



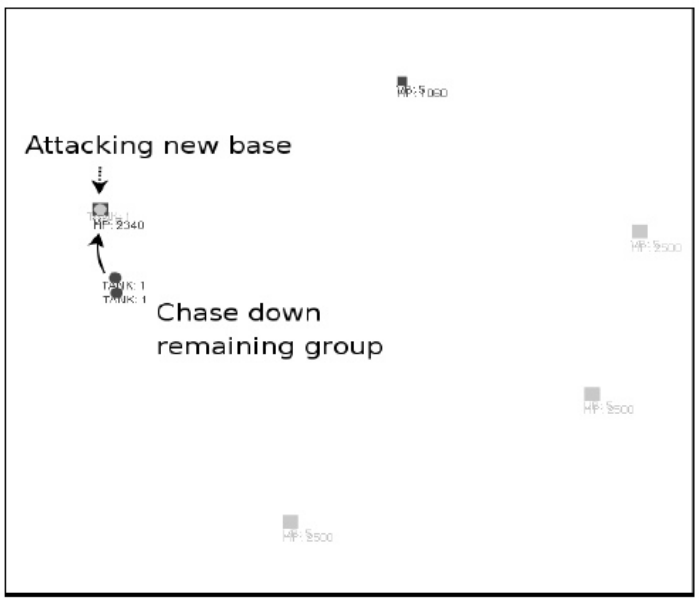
(a) Starting position and orders



(b) Opponents attack bases while we gather



(c) We eliminate part of the enemy force



(d) We eliminate the rest of the enemy

Results

RTS-Plan without opponent modeling vs. individual strategies

Strategy	Wins	Losses	Ties
Null	100	0	0
Join Defence	96	4	0
Mass Attack(base)	99	1	0
Mass Attack(units)	99	1	0
Spread Attack(base)	38	62	0
Spread Attack(units)	38	62	0
Half Defense-Mass Attack	99	1	0
Hunter	46	54	0
Attack Least Defended	99	1	0
Harass	70	28	2

Results

RTS-Plan with opponent modeling vs. individual strategies

Strategy	Wins	Losses	Ties
Null	100	0	0
Join Defence	100	0	0
Mass Attack(base)	99	1	0
Mass Attack(units)	99	1	0
Spread Attack(base)	83	17	0
Spread Attack(units)	83	17	0
Half Defense-Mass Attack	99	1	0
Hunter	63	37	0
Attack Least Defended	100	0	0
Harass	91	5	4

With opponent modelling, RTS-Plan is able to win against each single opponent strategy

Conclusion

RTS-Plan is first step towards adversarial planning in RTS games

Promising initial results

Computer Poker

Idea:

- Abstract full game (independent bidding rounds, hand strength buckets, etc.)
- Solve much smaller abstract game using LP approach
- Map solution back to full game

$\rightsquigarrow$ creates strong poker players, BUT ...

Nash-optimal players are not that interesting if weaker players need to be exploited – say for winning more tournaments

Exploitation

Nash-optimal player won't lose in the long run, but also will not win anything if the expected game value is 0

E.g., playing Rock all the time in RPS

Want to exploit the opponent for maximum profit

E.g., eventually play Paper all the time against the Rock player

Opponent modeling is non-trivial, but essential
  Assume that he knows that I know that he knows ...

Iocaine Powder [3] won the first computer RPS tournament

Dealing with Large Game Trees

Algorithms based on regret minimization have been shown to be effective in Poker, leading to world-class AI systems that can defeat professional players

The idea is to play repeatedly and monitor the performance of moves in information sets to adjust their probabilities according to the losses they incurred against the currently best counter strategy

See [6,8,9] for details on Counterfactual Regret (CFR) minimization

Strategies estimated by CFR converge to a Nash-equilibrium in 2-player, zero-sum games

CFR only requires memory linear in the number of information sets, not in the number of states

$\rightsquigarrow$ more fine-grained abstractions can be used

Future Work

- Look-ahead procedure similar to MiniMax search that approximates Nash-optimal strategies (see DeepStack paper [8])
- Stochastic programming: how to deal with payoff entries that are random variables? E.g., payoff estimates
- Model opponents effectively
- More than 2 players

We are working on a strong Skat program, for which we need to address 3-player aspects, opponent modelling, and abstraction to approximate Nash-equilibria (see [7])

References

[1] D. Koller et al., Fast Algorithms for finding randomized strategies in game trees, Proceedings of the 26th ACM Symposium on Theory of Computing, 750-759

[2] B. Stengel, Computing Equilibria for Two-Person Games, technical report, 1996

[3] `http://ofb.net/~egnor/iocaine.html`

[4] F. Sailer, M. Buro, and M. Lanctot, Adversarial Planning Through Strategy Simulation, CIG 2007, Hawaii USA, 2007

[5] T. Furtak and M. Buro, On the Complexity of Two-Player Attrition Games Played on Graphs, AIIDE, Stanford USA, 2010, on my webpage

References (continued)

[6] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret Minimization in Games with Incomplete Information. In Advances in Neural Information Processing Systems 20 (NIPS), 2008

[7] M. Buro et al., “Improving State Evaluation, Inference, and Search in Trick-Based Card Games”, IJCAI 2009 (on my webpage)

[8] M. Moravčík et al., DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker, https://arxiv.org/abs/1701.01724

[9] CFR Introduction: `.../material/cfr-intro.pdf`