Machine Learning in Multi-Agent Planning

Geoff Gordon

ggordon@cs.cmu.edu

ML problems in multi-agent planning

Structured prediction problems

Interference from other agents

Help from other agents

Outline

Interference from other agents

- No-regret algorithms

Help from other agents

- Auction algorithms

Examples and experiments

Poker

Poker as a machine learning problem

Prediction problem:

- From: background knowledge, observations of players
- Predict: how should I play in next hand

How should I play:

- From: history of observations and actions in current hand
- Predict: should I bet on this round?

Why is poker hard?

Big output space

Don’t get information about “what ifs”

Structured output space

Neither adversarial nor cooperative

Why is poker hard?

Big output space

Don’t get information about “what ifs”

Structured output space

Neither adversarial nor cooperative

Output space

Behavior strategy: information state $\mapsto P(\text{action})$

$[0,1]^{\#}$ info states

9 K pass pass bet call 8 7 K $\mapsto$ P(bet) = .72

Structured prediction problem

- $\mathcal{X}$: input space
- $\mathcal{Y}$: prediction space
- $\mathcal{H}$: hypothesis space
- ℓ_t: loss functions ($(y-5)^2$, $D_{\mathrm{KL}}(y \mid (.2, .3, .5))$)

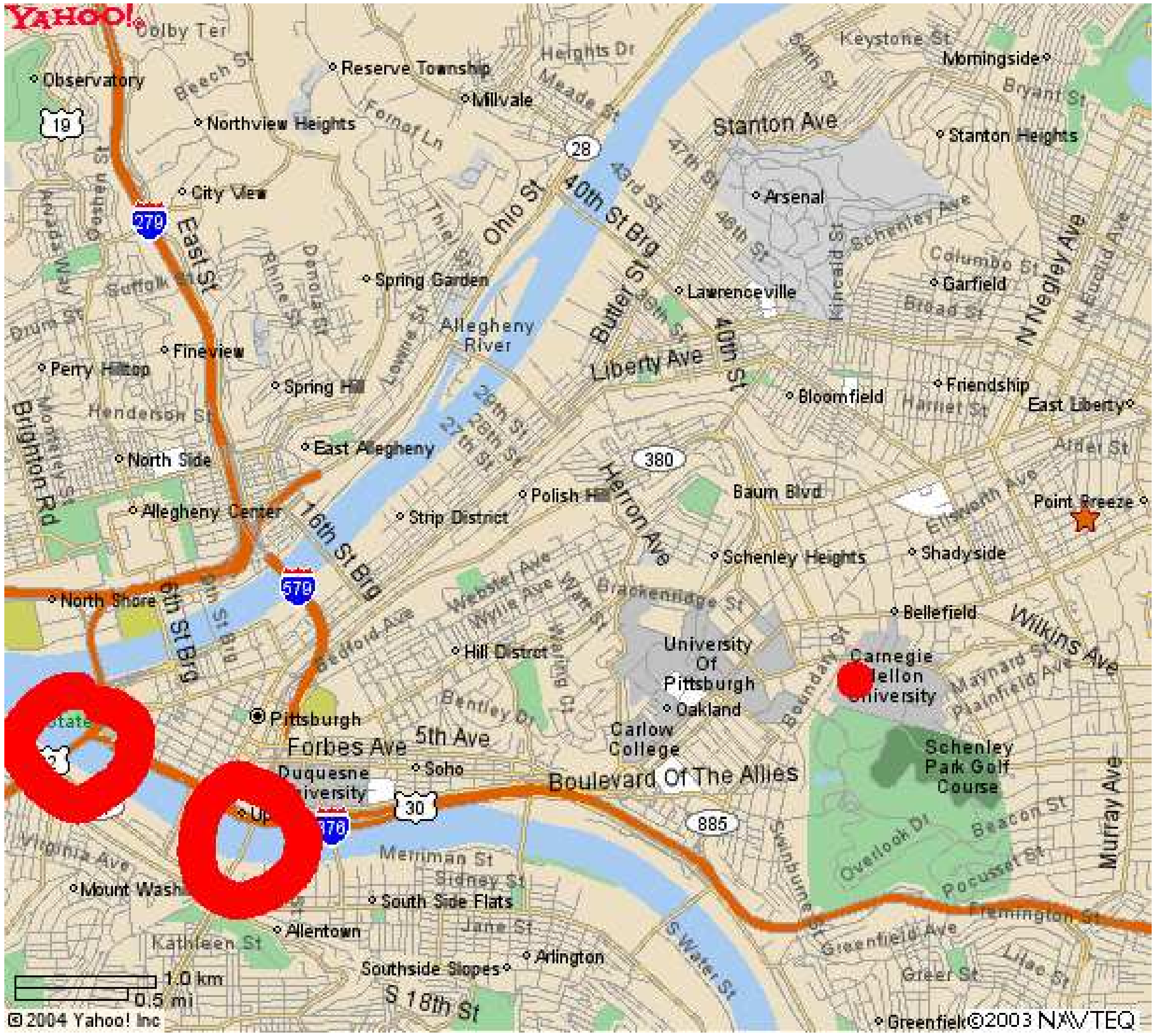
Will assume $\mathcal{H}$ is convex, $\subset \mathbb{R}^{d \times n}$

May have many vertices, many faces, or other complex features, but we have an efficient description

Poker as structured prediction

- $\mathcal{X}$: has Bill been drinking
- $\mathcal{Y}$: dist'n over behavior strategies
- $\mathcal{H} : \mathcal{X} \mapsto \mathcal{Y}$
- ℓ_t: money I lose in hand t

Path planning w/ unknown costs



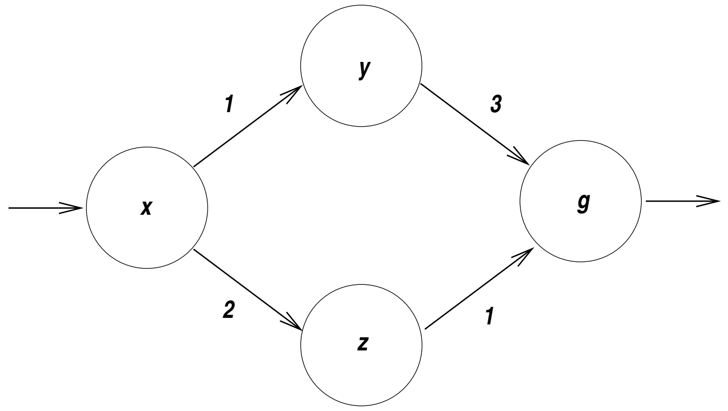
Path planning as structured prediction

$$\mathcal{X} = \begin{pmatrix} \text{day of week} \\ \text{phase of moon} \\ \text{recent eclipse} \end{pmatrix}$$

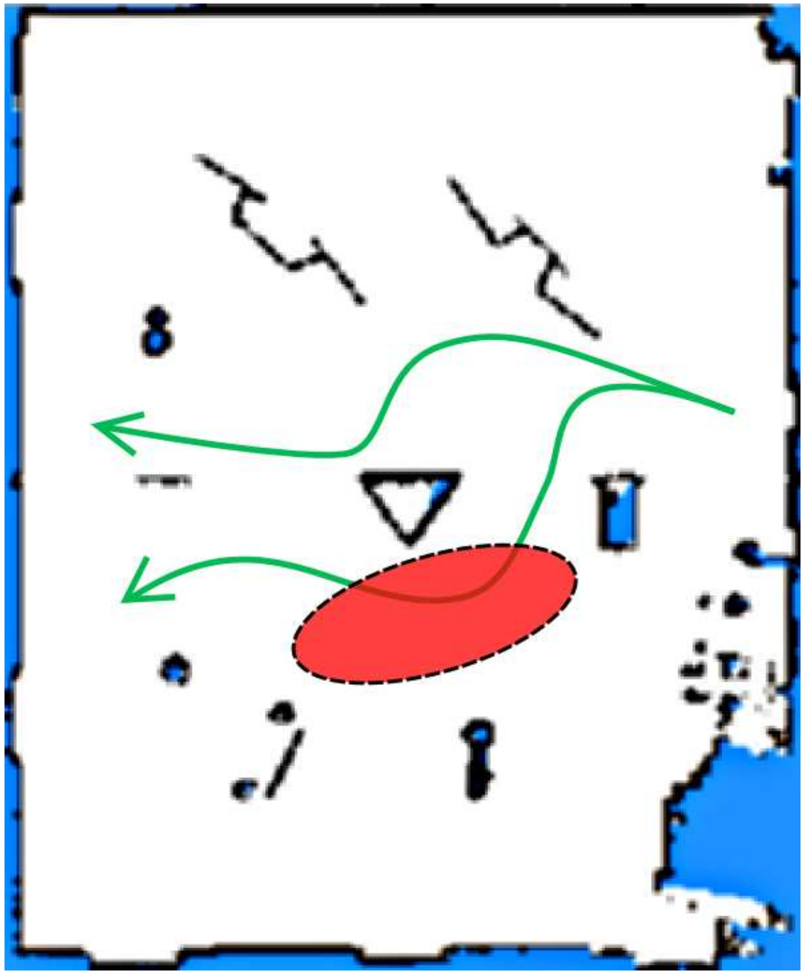
$\mathcal{Y} =$ paths in map

$$\mathcal{H} = \mathcal{X} \mapsto \mathcal{Y}$$

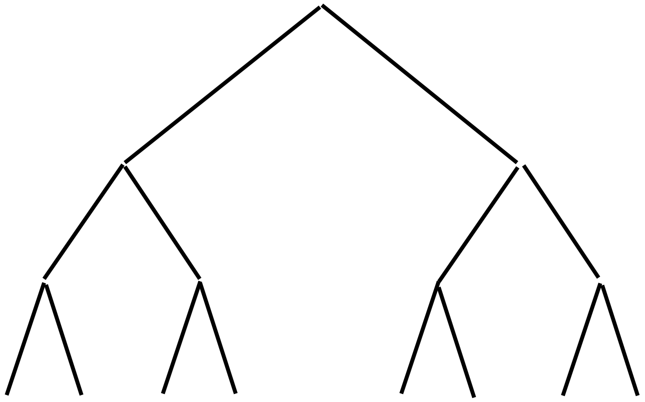
$$\ell(y) = (c + C) \cdot y$$

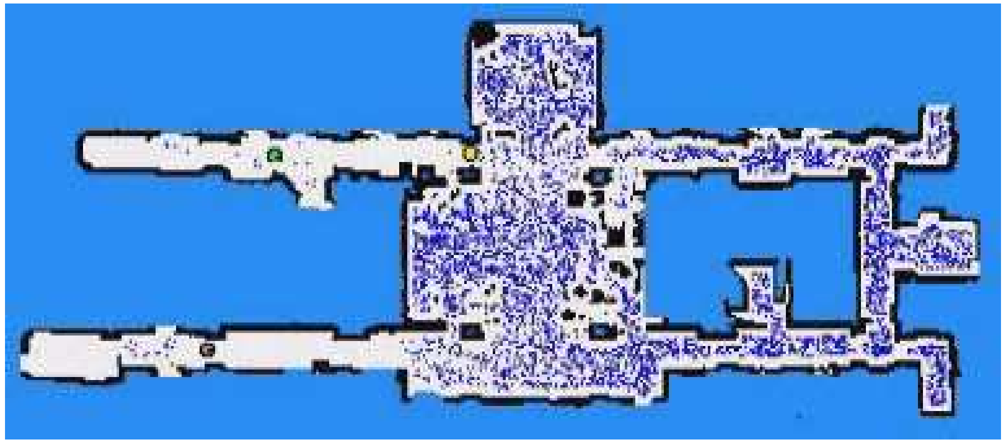


Adversarial path planning



Lookahead in multiagent planning





Lookahead, cont'd

$\mathcal{X} =$ features of current history

$\mathcal{Y} =$ behavior strategies

$\mathcal{H} = \mathcal{X} \mapsto \mathcal{Y}$

$\ell(y) = E(\text{heuristic}(\text{leaf}))$ (note: what ifs)

One-card poker

Bet $0 Bet $1 Deal

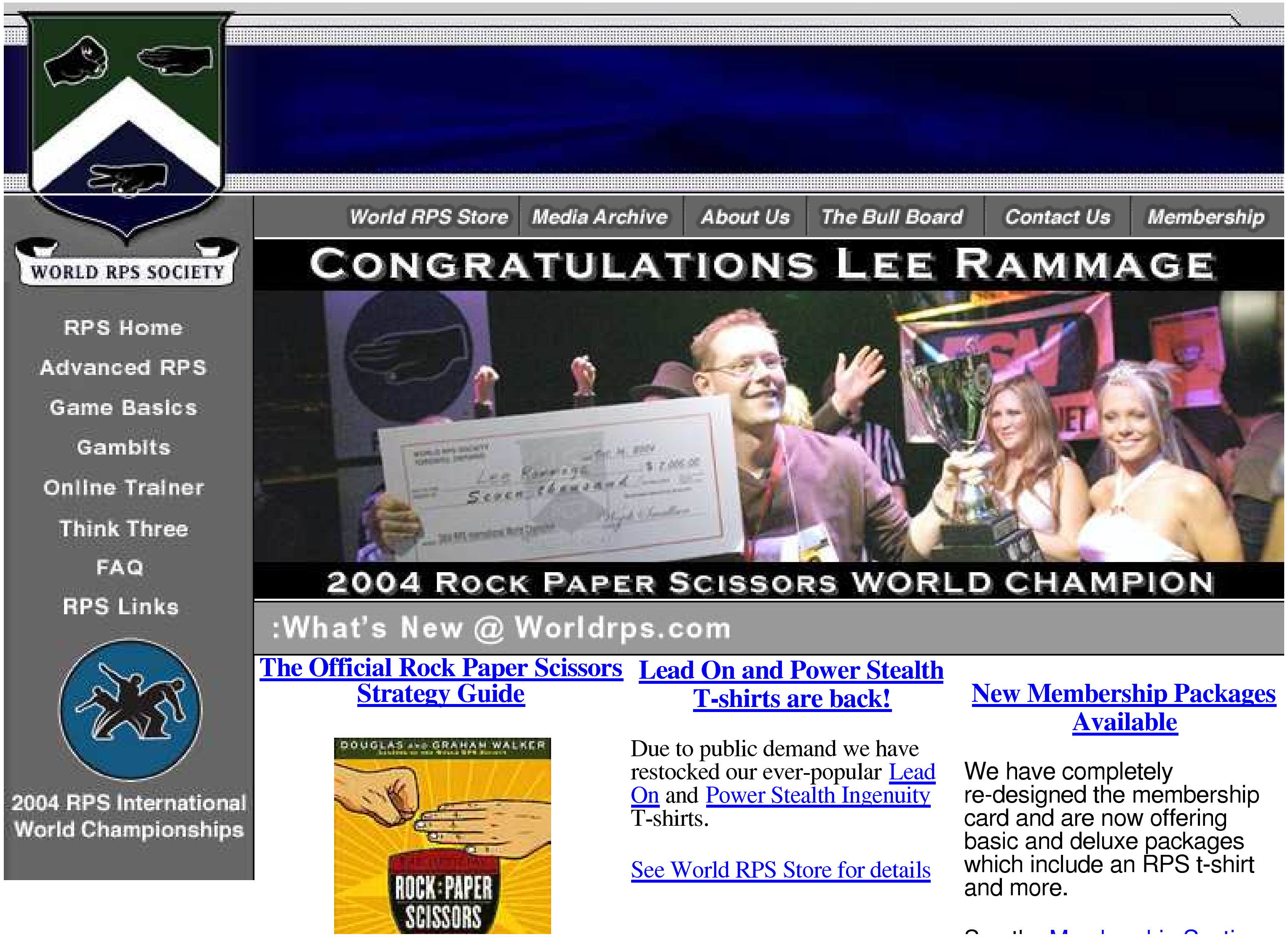
Reset

You have won $0 so far.
You and the computer each ante $1.
Your card is A. What do you bet?

Rock-paper-scissors

	R	P	S
R	0	1	-1
P	-1	0	1
S	1	-1	0

Rock-paper-scissors



The Official Rock Paper Scissors Strategy Guide

Lead On and Power Stealth T-shirts are back!

Due to public demand we have restocked our ever-popular Lead On and Power Stealth Ingenuity T-shirts.

See World RPS Store for details

New Membership Packages Available

We have completely re-designed the membership card and are now offering basic and deluxe packages which include an RPS t-shirt and more.

Neither statistical nor adversarial

Possible approaches:

- Learn model of environment (incl. other agents)
- Compute equilibrium (minimax, Bayes-Nash, . . .)

Problems:

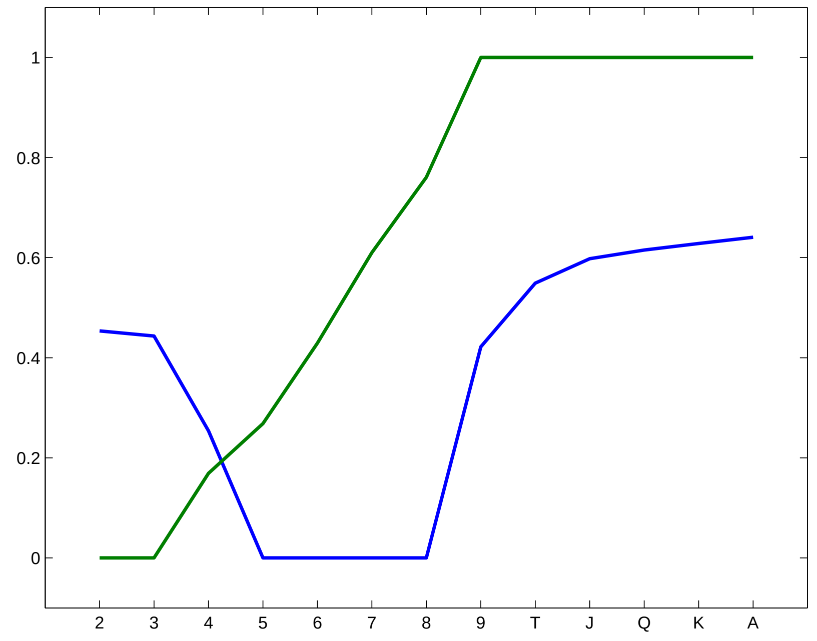
- Catastrophic failure when not i.i.d.
- Assumes you know other agents' motivations (inappropriate level of paranoia)

Building models

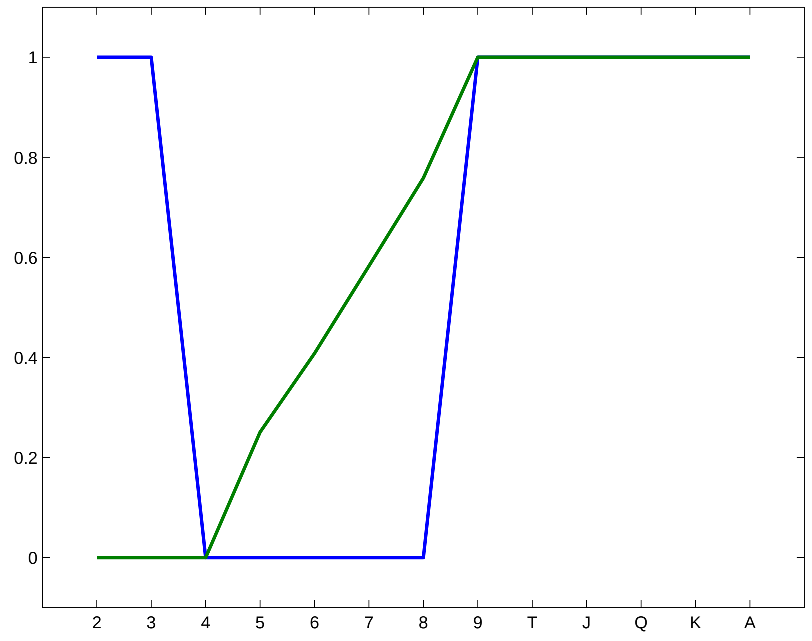
Sequence $HTHTHTHTHTHT\ldots$

Step t	1	2	3	4	5	$\ldots$
Predict	T	H	T	H	T	$\ldots$
Actual	H	T	H	T	H	$\ldots$

Paranoia



Gambler



Dealer

Tournament results

MaxPlayer	3.4856	Baazigar	2.3974
Victor	2.9968	Blitz	1.8700
Mouse	2.9552	CheatToLose	1.7122
TeamDiscoveryChannel	2.8924	OptPlayer (minimax)	1.5054
ActorCritic	2.8624	Corrado	0.7686
PatternRecognition	2.8572	KennyRogers	-0.7920
KillerPlayer	2.7906	RandomPlayer	-10.7776
YoavShohamAllStars	2.7594	SomeRegret	-22.4770

Total winnings in 10,000 games (5,000 each as Dealer and Gambler)

No regret

Regret = ρ = how much do I wish I had done something else?

E.g., opp played $RRRRPRRRRSRRRR$, I played at random

Lots of regret for not playing “P all the time”

(Lots of negative regret for “S all the time”)

Comparison class

Allowable “something else” = comparison class

Little class: easy to get algorithms, but low regret isn’t impressive

Big class: hard to get algorithms, but low regret inspires confidence

Typical:

- all constant $h \in \mathcal{H}$ (e.g. “R all the time”)
- simple rules for modifying h (e.g. $R \mapsto P$)

No-regret algorithms

Guarantee ρ grows slower than $O(t)$, often $O(\sqrt{t})$

Average regret $\frac{\rho}{t} \to 0$ as $t \to \infty$ at rate $1/\sqrt{t}$

Guarantee is for *all* sequences of opp plays

$\Rightarrow$ approach equilibrium if opponent plays well, something like CLT if opponent plays obliviously

No-regret intuition

Choose actions with positive regret

Regret for chosen action can’t increase

When two actions have similar regrets, randomize

Algorithm recipe

Pick a *potential function*, $F(S)$

Keep track of *regret vector*, S_t

Compute $F'(S_t) = f(S_t) = \bar{H}$

Normalize to get $H = \alpha \bar{H} \in \mathcal{H}$

Play H

Regret vector and potential fn

$\rho(H) = S_t \cdot H$ = regret vs. H

safe set = $\{S \mid (\forall H \in \mathcal{H})\ S \cdot H \leq 0\}$

In safe set $\Rightarrow \rho \leq 0$

Potential: large outside safe set, zero inside, bounded curvature

Regret in RPS

$$x_t = \begin{pmatrix} \text{1 if I played R} \\ \text{1 if I played P} \\ \text{1 if I played S} \end{pmatrix} \qquad y_t \text{ = same for opponent}$$

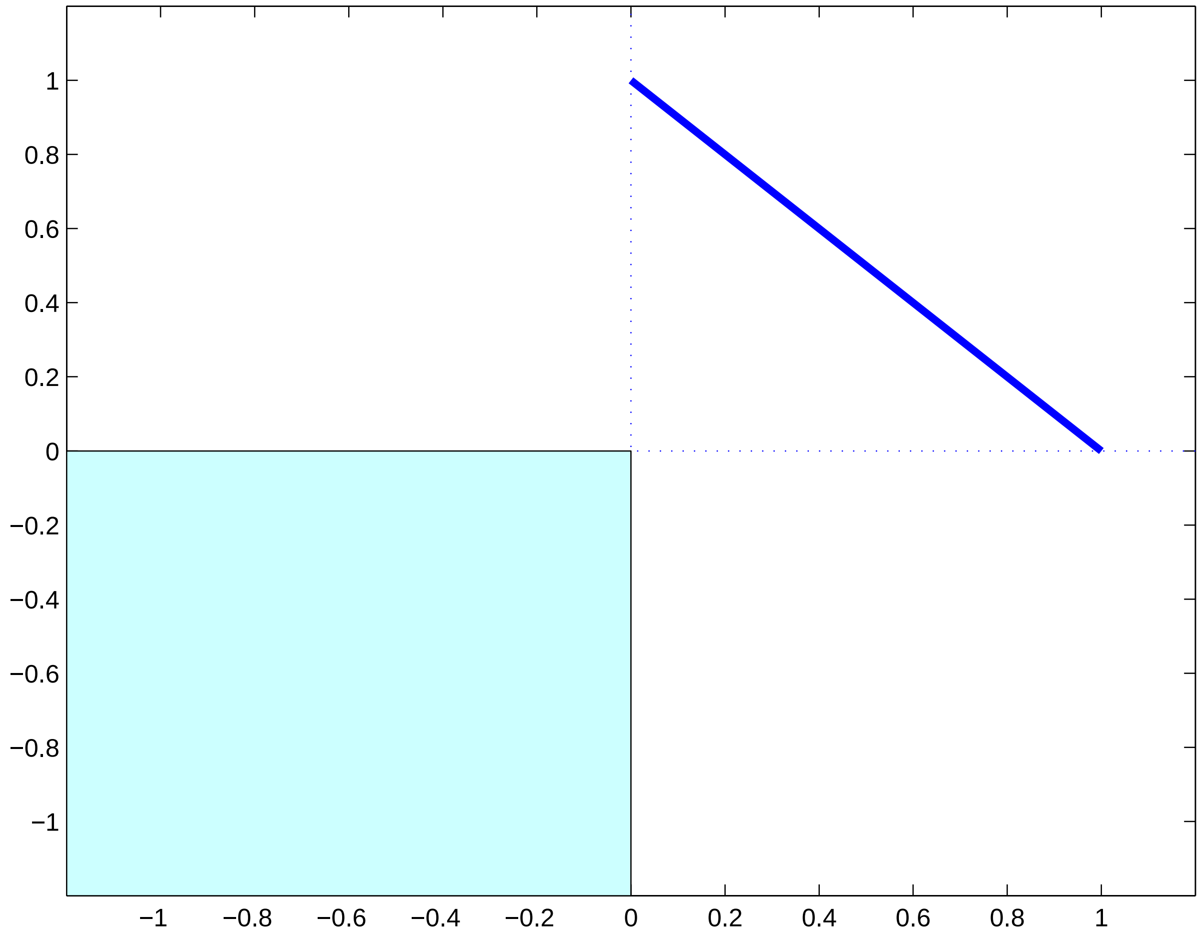
$My_t =$ my payoffs for each action at time t

$r_t = x_t \cdot My_t =$ my payoff

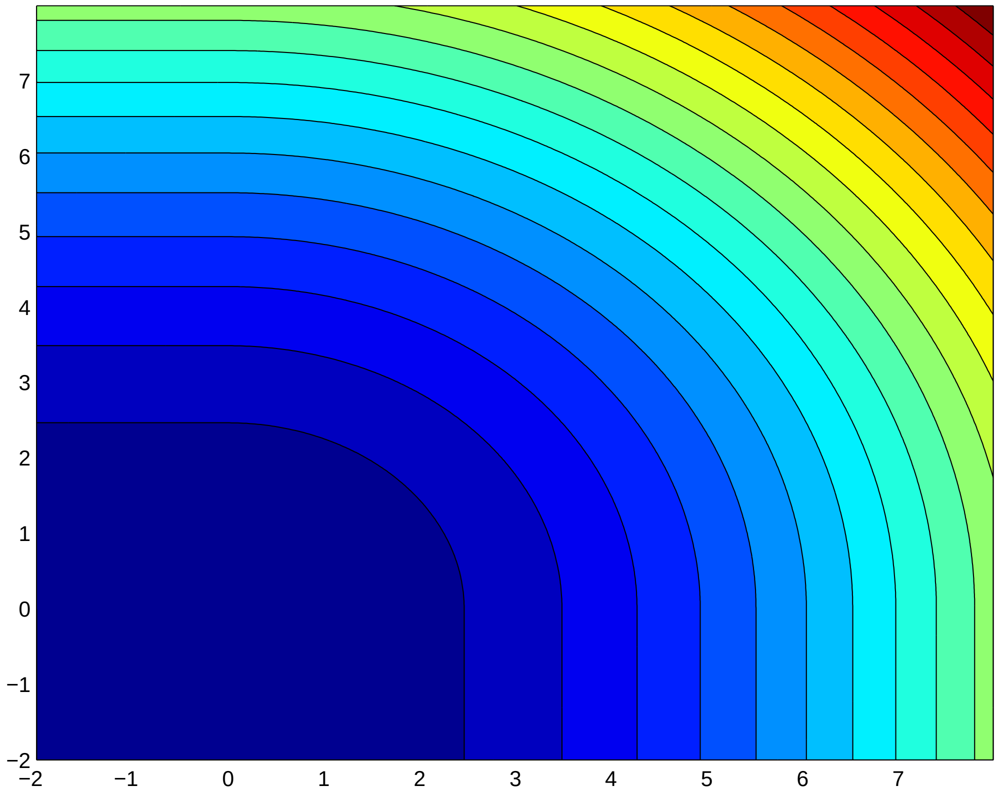
$s_t = My_t - r_t\mathbf{1} =$ my regret vector

$s = \sum_t s_t \qquad \rho = \max s$

Safe set in RPS



Potential



$$F(s) = \| \, [s]_+ \|_2^2$$

Algorithm for RPS

Given s

Compute s_+

Renormalize to get $q = \alpha s_+$

Randomize according to q

“Regret matching” [Hart & Mas-Colell]

One-card poker

Define $\mathcal{X}, \mathcal{Y}, \mathcal{H}$ and ℓ

Define regret vector S

Pick F (will be $F(S)$ = squared distance from safe set)

Show how to compute $F'(S)$

One-card poker

$\mathcal{X} = \{1\}$

$\mathcal{Y} =$ (randomized) poker strategies

$\mathcal{H} = \mathcal{Y}$

$\ell_t =$ money lost on round t

Regret vector

Poker strategies are $H = \{h \mid Ah + b = 0,\ h \geq 0\}$

Payoffs are $r \cdot h$

Regret is

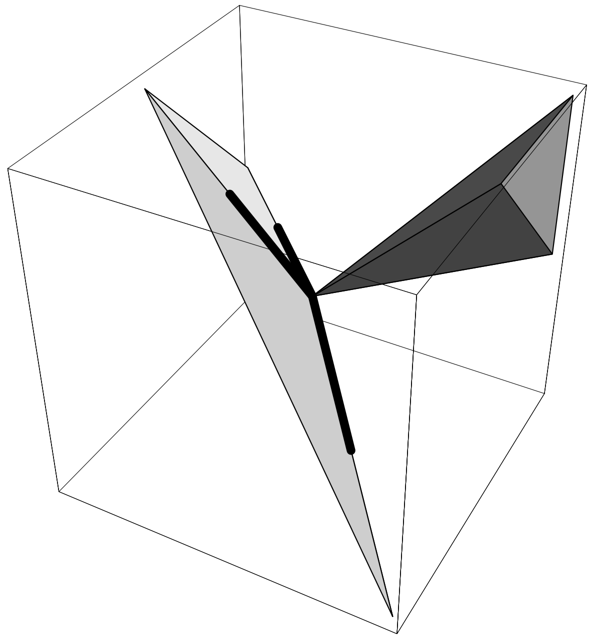
$$S_t = \begin{pmatrix} r_t \\ -r_t \cdot h_t \end{pmatrix}$$

Now $(h, 1) \cdot S = h \cdot r_t - r_t \cdot h_t$

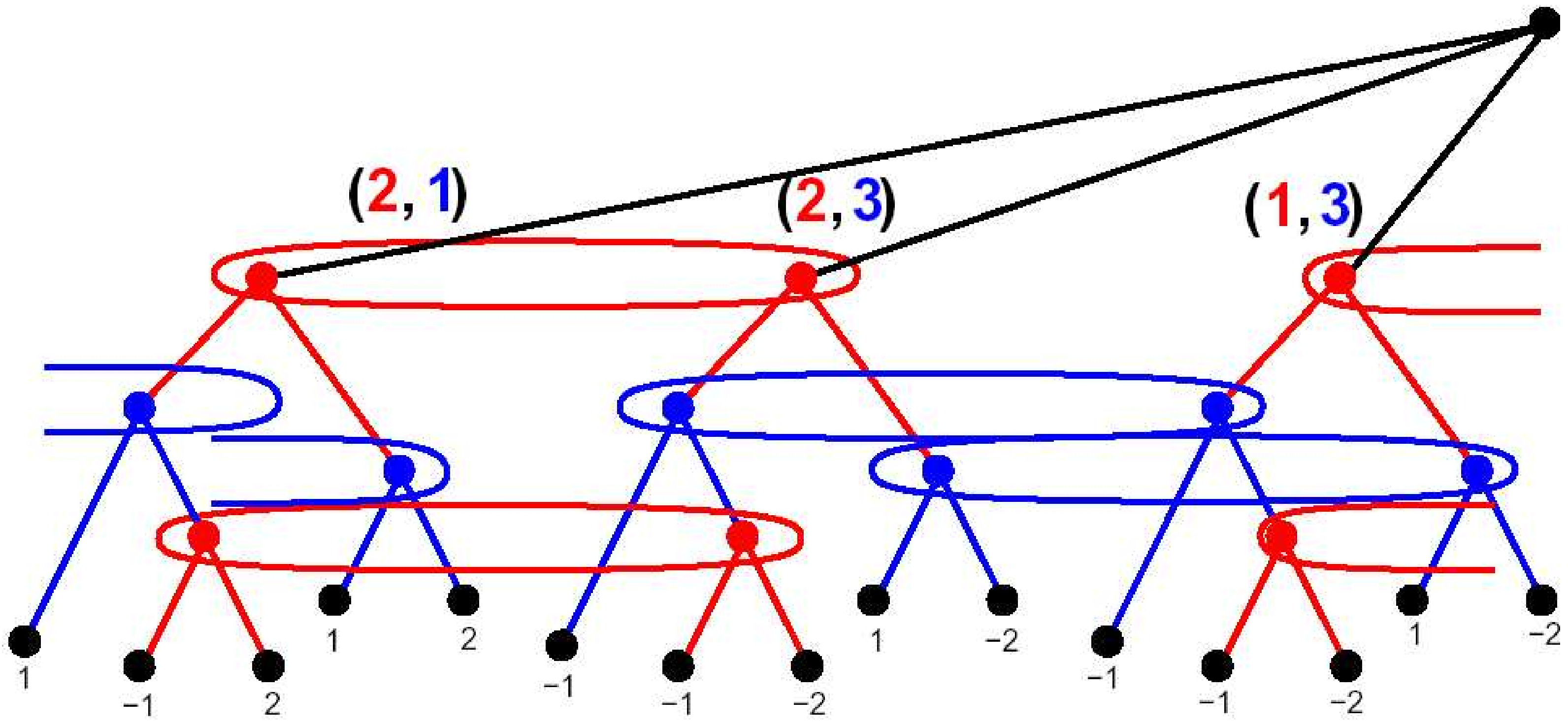
Safe set is $H^\perp = \{y \mid y \cdot h \leq 0\ \forall h \in H\}$

Safe set

Safe set is $H^{\perp} = \{y \mid y \cdot h \leq 0 \ \forall h \in H\}$



One-card poker game tree



Information states and actions

Gambler:

- Holding 2-A in first round
- Holding 2-A in second round
- Pass or bet for each

Dealer:

- Holding 2-A after pass
- Holding 2-A after bet
- Pass or bet for each

Sequences

Sequence = history of my observations and actions, ending in my action

Ex: J pass bet bet

Special case: ϵ = empty sequence

Sequences for all players $\Rightarrow$ leaf of game tree

1CP: 780 leaves, 638 internals, 52 seqs/player

Sequence weights

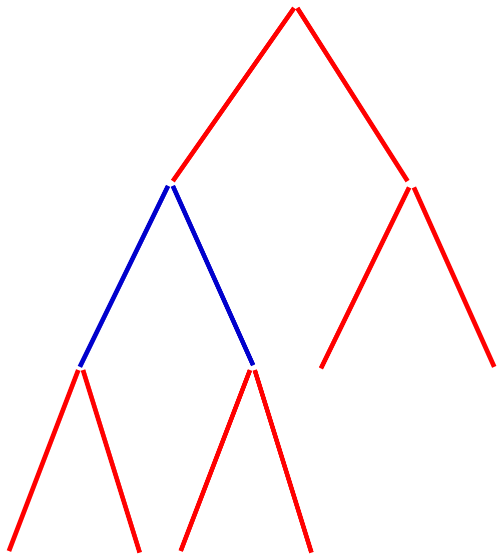
Let s, a be my info state, action

$$x_{sa} = \prod_{\substack{\text{ancestors} \\ (s', a') \text{ of } s}} P(a' \mid s')$$

s' is my info state, $s' \neq s$, a' leads towards s

Ex: J pass raise call
$\Rightarrow P(\text{pass} \mid J) P(\text{call} \mid \text{J pass raise})$

Convex sequence weight set



$$\left(\begin{array}{cccccccc|c} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{array}\right)$$

At any choice point, split flow from most recent ancestor I control

Linear payoffs

Strategies x(sequence), y(sequence)

Payoff = $\sum_{\text{leaves } i} r_i P(i \mid x, y)$

Ex: J K pass raise call $\Rightarrow$
$P(J)P(K)P(\text{pass} \mid J)P(\text{raise} \mid \text{K pass})$
$P(\text{call} \mid \text{J pass raise})$

$P(J)P(K)x(\text{J pass raise call})y(\text{K pass raise})$

The algorithm

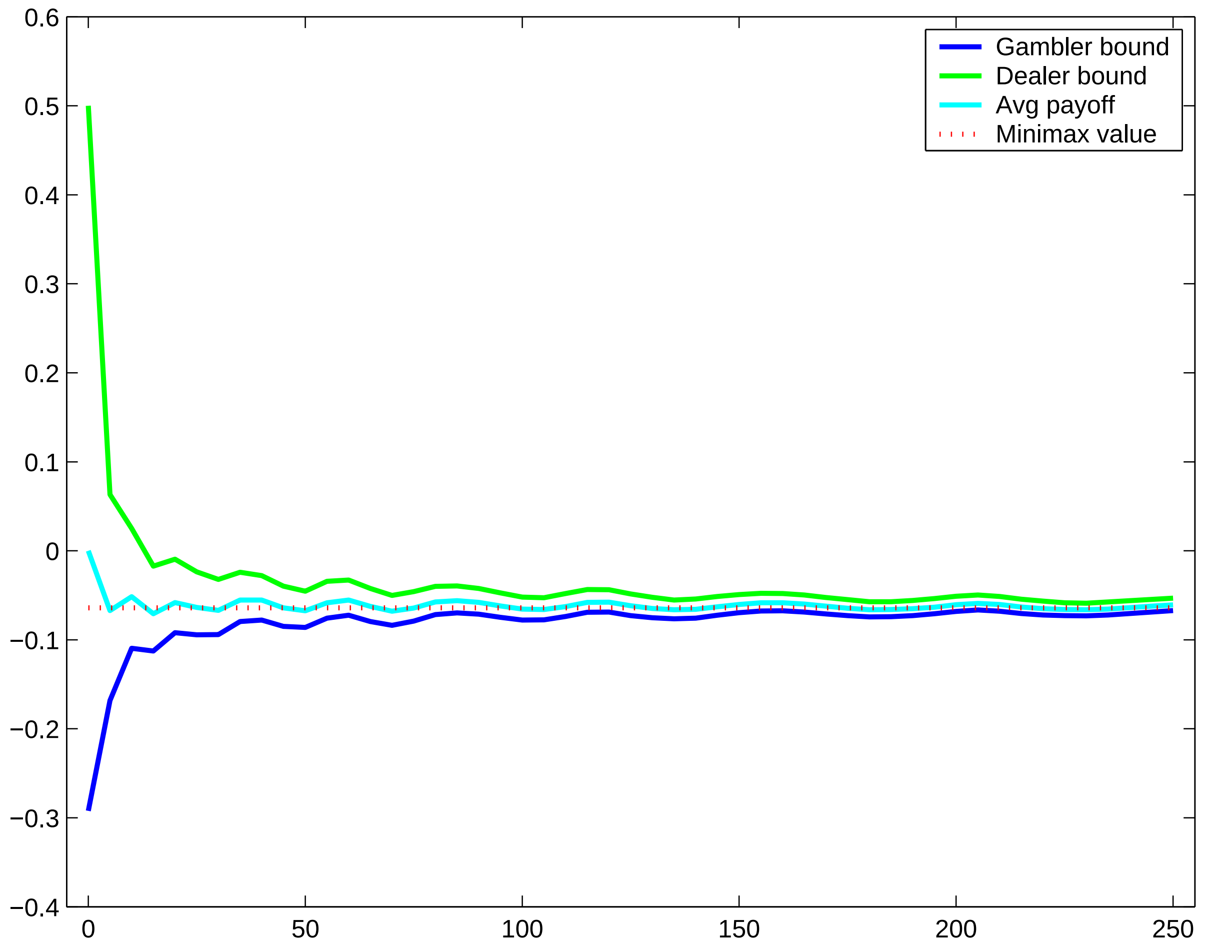
Given game A, a and regret vector s, solve:

$$\min_{x,\lambda} \; \left(\|x\|^2/2 - s \cdot x\right)$$
$$Ax = \lambda a \qquad x, \lambda \geq 0$$

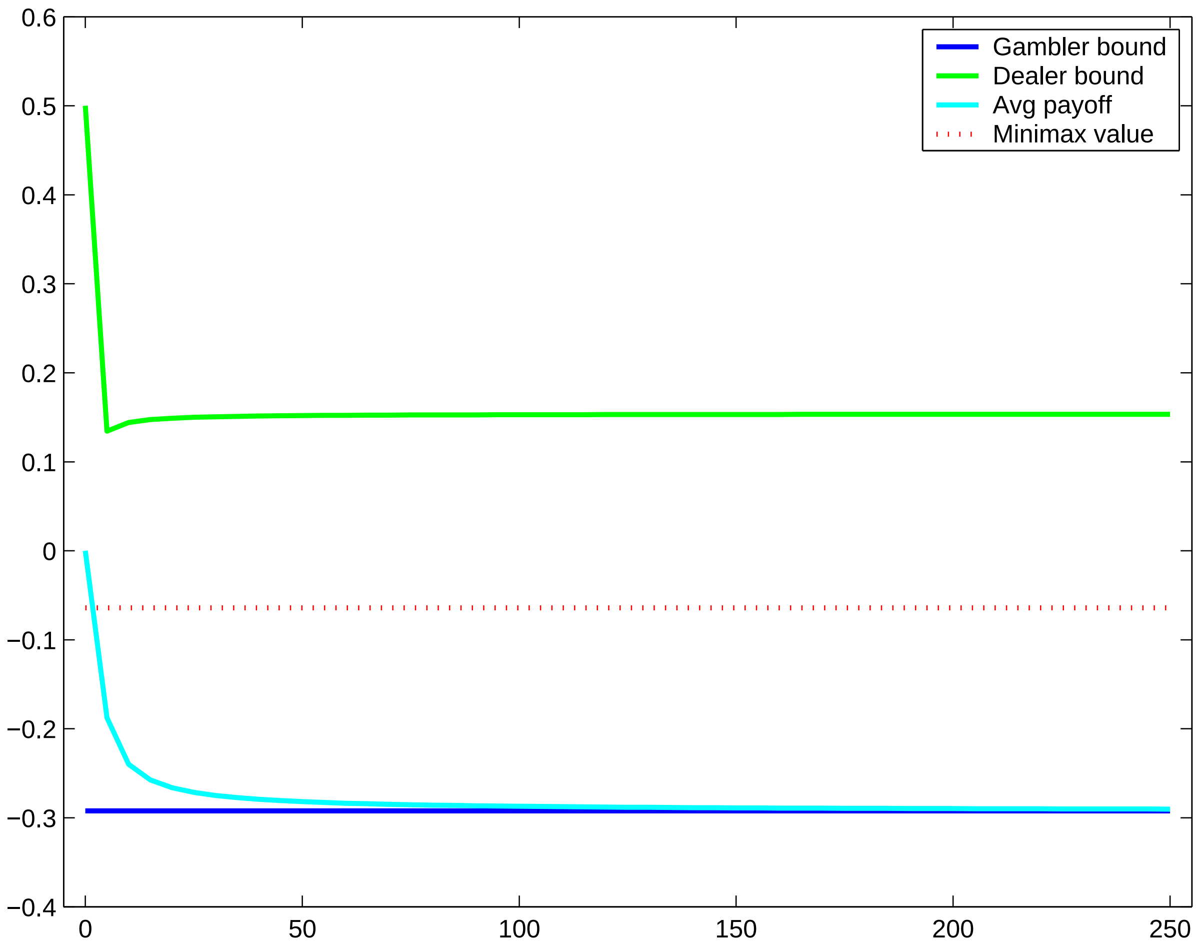
Then play x/λ (or arbitrarily if $\lambda = 0$)

Note: if $A = (1, 1, 1, \ldots)$ and $a = 1$, then $x = [s]_+$ (regret matching)

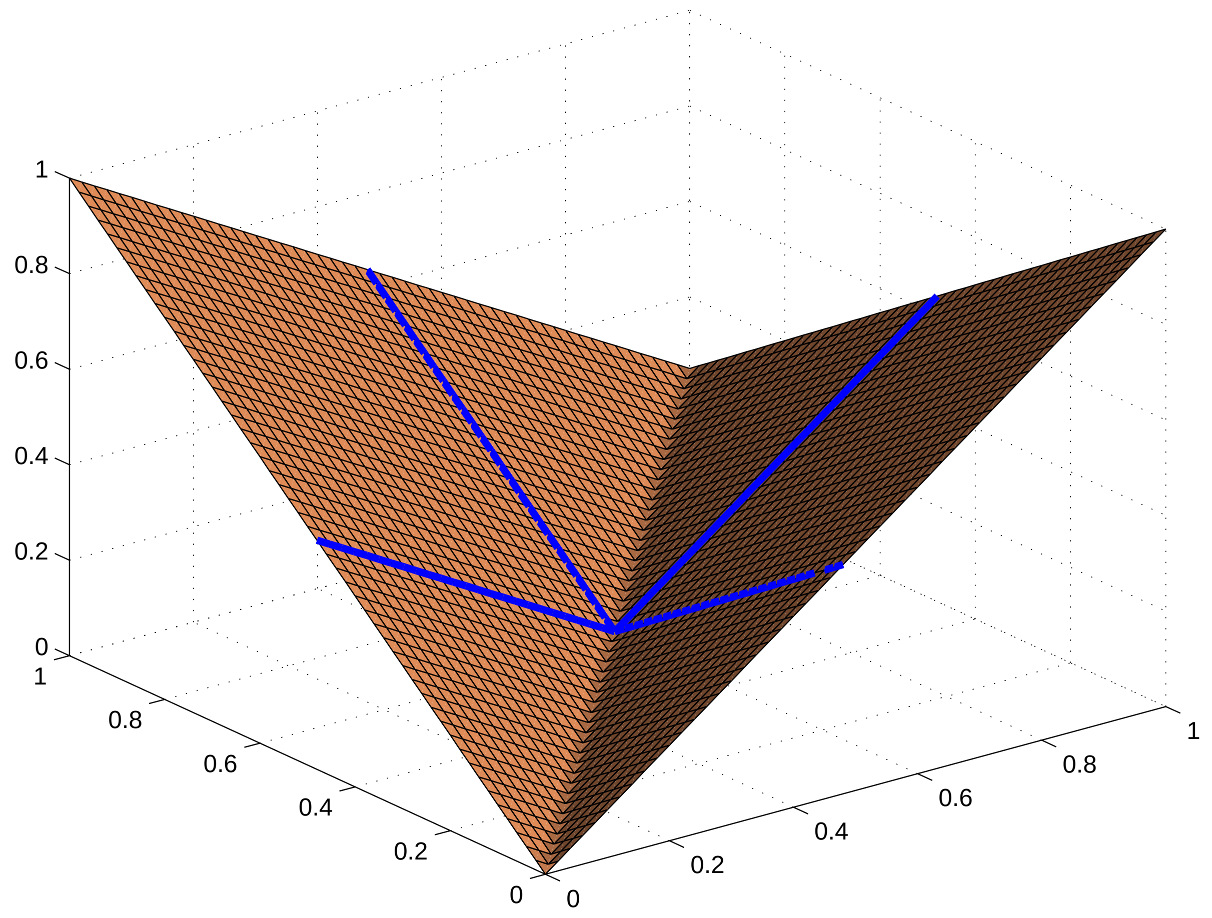
One-card poker self-play



Play against fixed Gambler



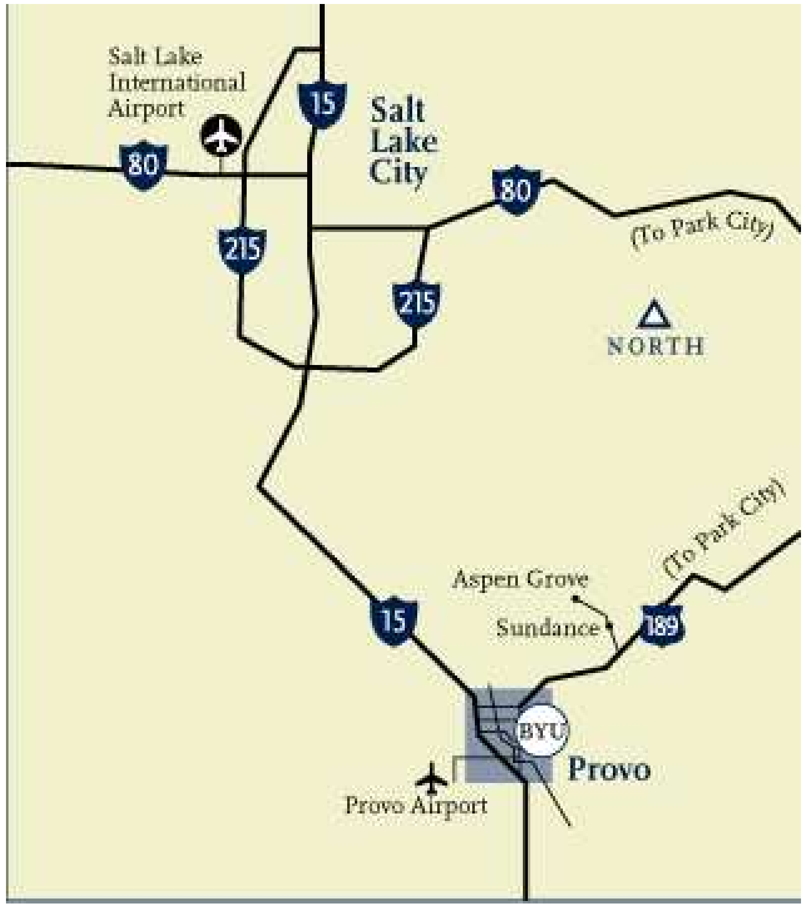
Help, I need somebody



Task allocation

[image credit: CMU’s FIRE project]

Traffic control



Basic auction algorithm

1. [everyone] Start with nominal plan
2. [auctioneer] Identify a point of interaction
3. [bidders] Calculate and submit bids
4. [auctioneer] Clear auction
5. Repeat from 2

[e.g., contract net protocol]

Auction applied to traffic control

1. Plan paths assuming we know edge costs, store cost-to-goal
2. Execute until conflict
3. Look ahead at alternate routes, calculate bids
4. Assign best feasible combination of paths
5. Repeat from 2

Chicken and egg

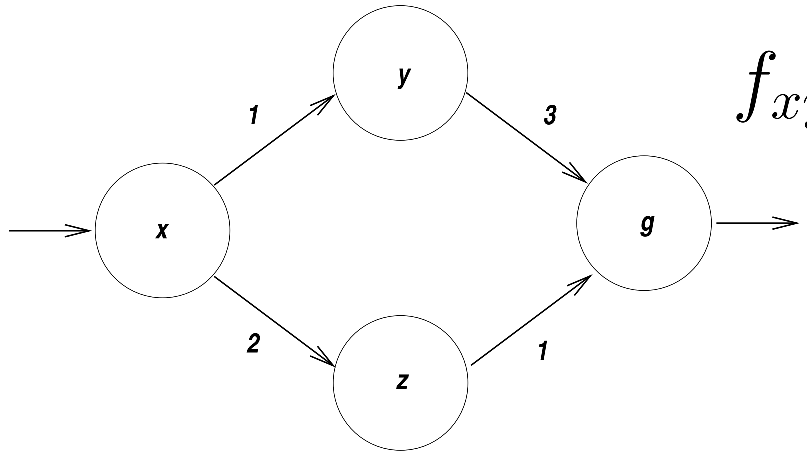
Look-ahead is inefficient w/o good cost-to-goal function

Cost-to-goal function requires good price estimates

Accurate price estimates require successful look-ahead

Path planning as LP

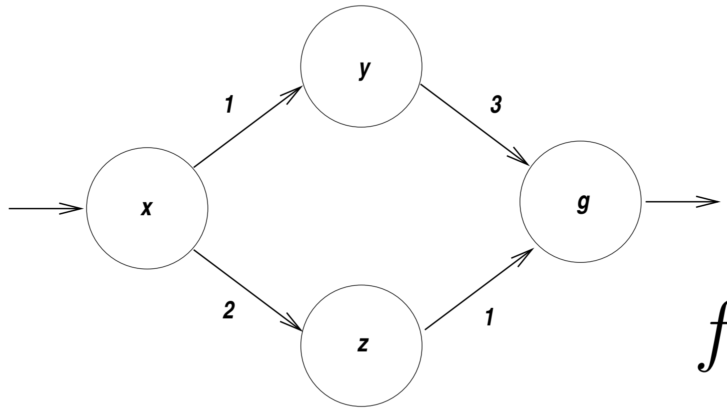
$$\begin{array}{rrrrrrcl}
-f_{xy} & -f_{xz} & & & & +1 & = & 0 \\
f_{xy} & & -f_{yg} & & & & = & 0 \\
& f_{xz} & & -f_{zg} & & & = & 0 \\
& & f_{yg} & +f_{zg} & -f_g & & = & 0
\end{array}$$



$$f_{xy}, f_{xz}, f_{yg}, f_{zg}, f_g \geq 0$$

Congestion, single-agent case

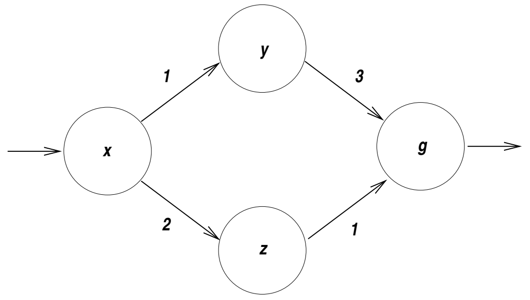
$$
\begin{array}{rrrrrrcr}
-f_{xy} & -f_{xz} & & & & +1 & = & 0 \\
f_{xy} & & -f_{yg} & & & & = & 0 \\
& f_{xz} & & -f_{zg} & & & = & 0 \\
& & f_{yg} & +f_{zg} & -f_g & & = & 0
\end{array}
$$

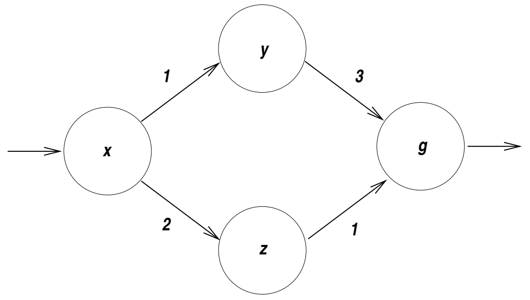


$$f_{zg} \leq 1/2$$

$$f_{xy}, f_{xz}, f_{yg}, f_{zg}, f_g \geq 0$$

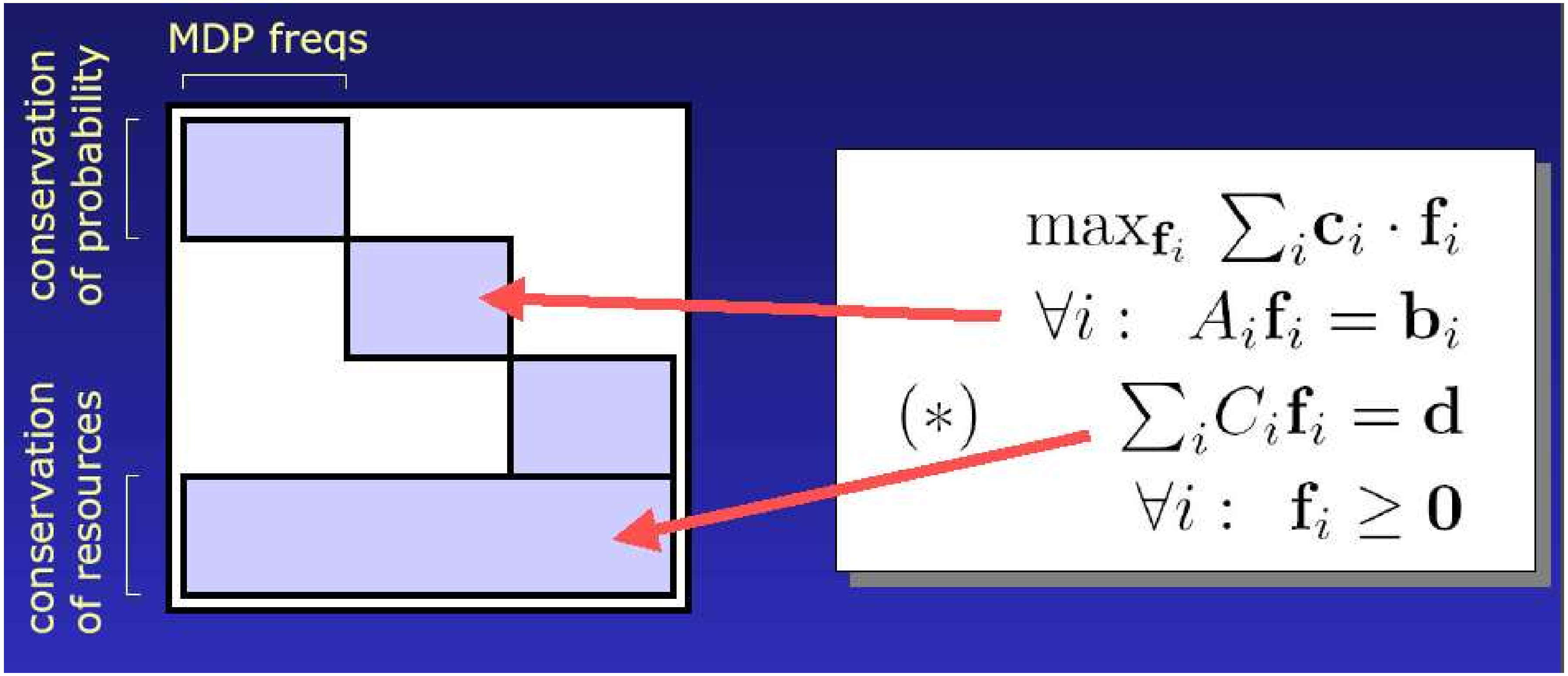
Congestion, two agents





$$
\begin{array}{rcl}
-f_{xy} \quad -f_{xz} \qquad\qquad\qquad\qquad +1 & = & 0 \\
-g_{xy} \quad -g_{xz} \qquad\qquad\qquad\qquad +1 & = & 0 \\
\cdots & & \\
{\color{red} f_{zg} + g_{zg}} & {\color{red}\leq} & {\color{red} 1} \\
f_{xy}, f_{xz}, \ldots, g_{xy}, \ldots & \geq & 0
\end{array}
$$

In general



$$\max_{\mathbf{f}_i} \sum_i \mathbf{c}_i \cdot \mathbf{f}_i$$
$$\forall i: \quad A_i \mathbf{f}_i = \mathbf{b}_i$$
$$(*) \qquad \sum_i C_i \mathbf{f}_i = \mathbf{d}$$
$$\forall i: \quad \mathbf{f}_i \geq \mathbf{0}$$

Overall algorithm

Send prices to robots

Robots plan individually, using prices to avoid congested areas

Report paths back to master

Master finds best combination of known paths, recalculates prices

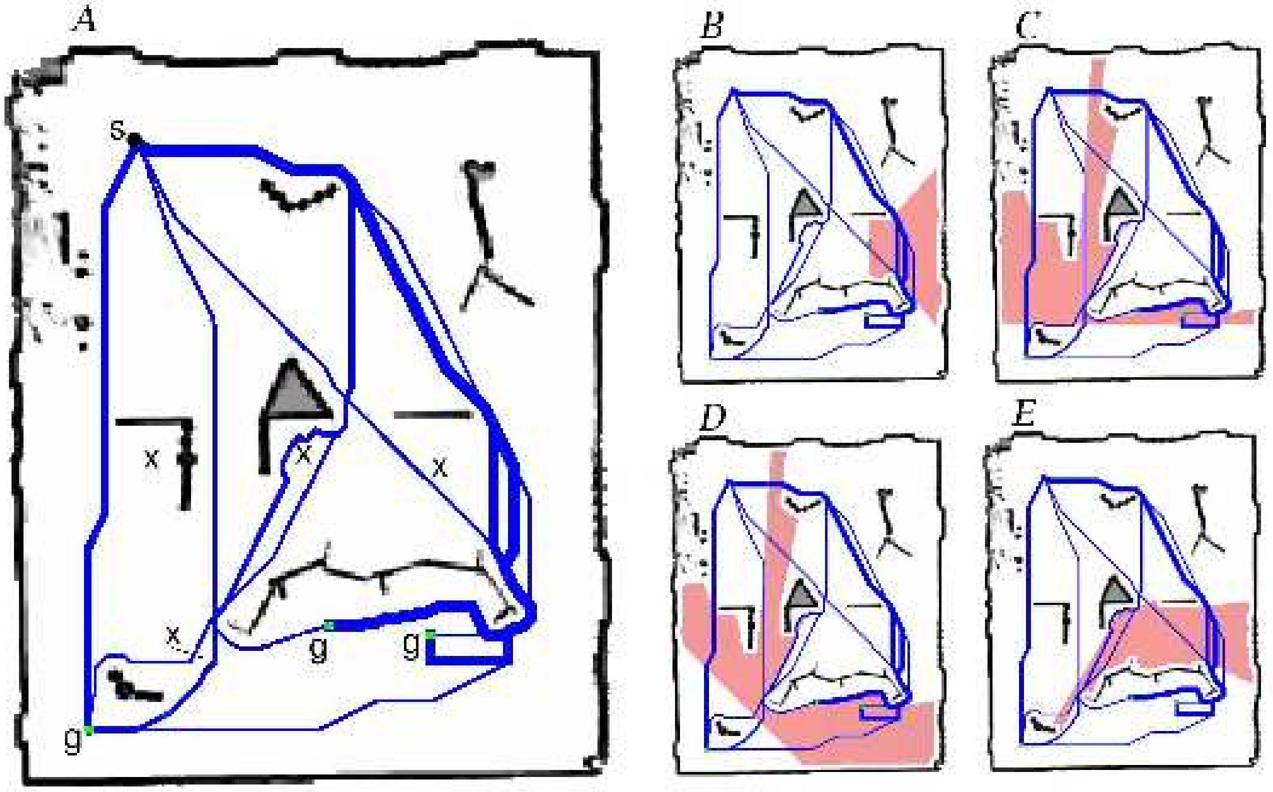
Repeat until done

Return: cost-to-goal for each robot

Example: fuel constraint

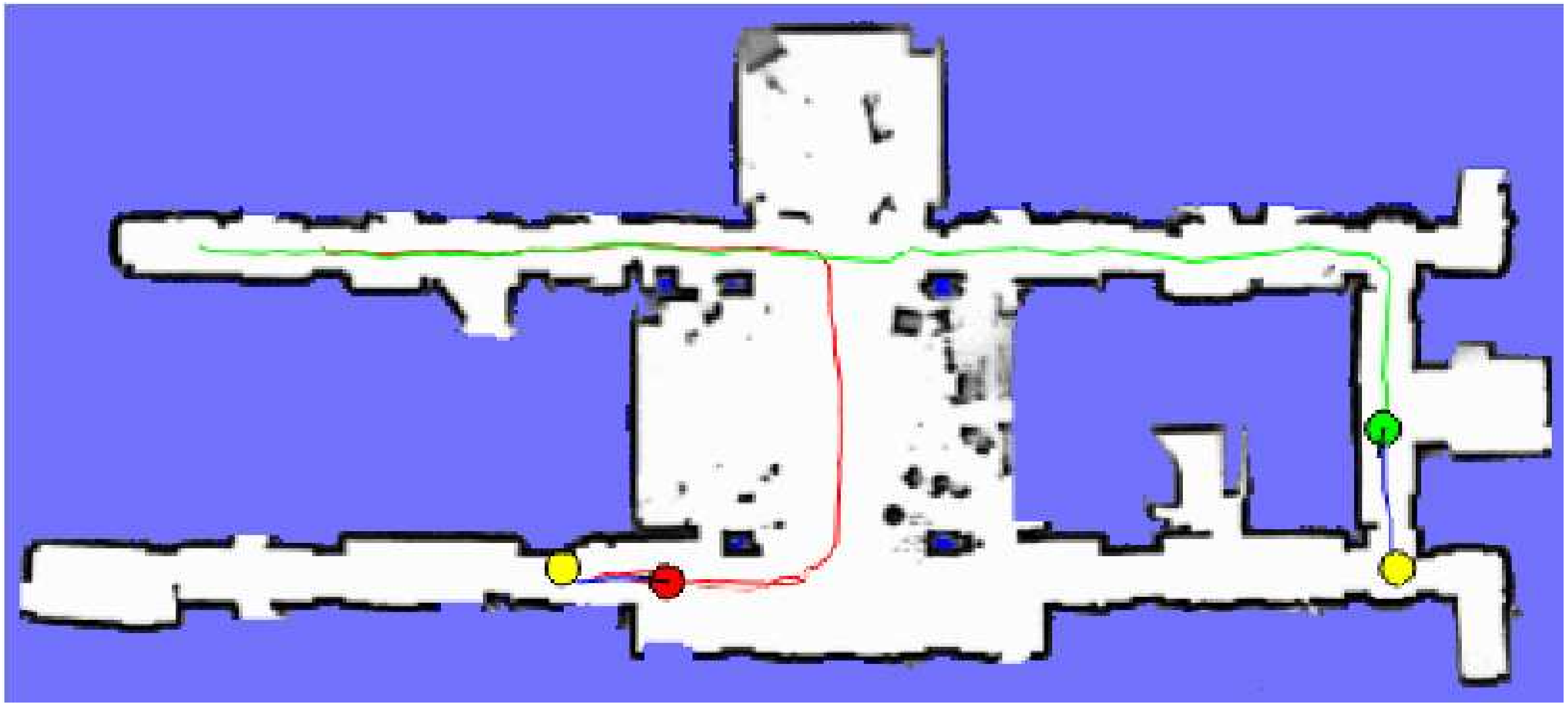
[Animation]

Auctions and no-regret in paintball



[joint work with Curt Bererton, Sebastian Thrun]

Lookahead search



[joint work with Rosemary Emery-Montemerlo, Sebastian Thrun]