A Complete Solver for Constraint Games

Thi Van-Anh Nguyen, **Arnaud Lallouet**

GREYC, Normandie University, France

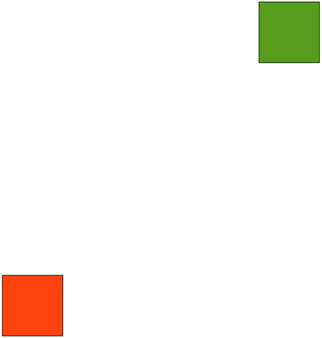
COSI, June 10, 2014

Thi-Van-Anh Nguyen is supported by Microsoft Research grant MRL-2011-046

Contents

Strategic games: the setting

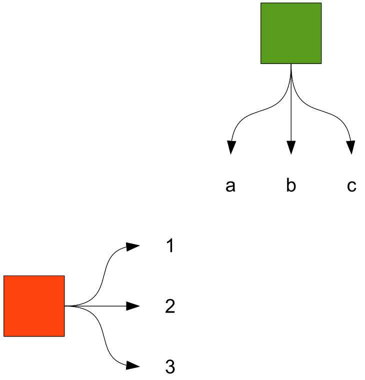
- A set of Players
- Each player performs Actions...
- ... and wants to maximize an Utility depending on other players actions
- Different players have different utilities
- Strategic form, also called multimatrix model



In this talk: finite games only (also simultaneous, perfect information, selfish players)

Strategic games: the setting

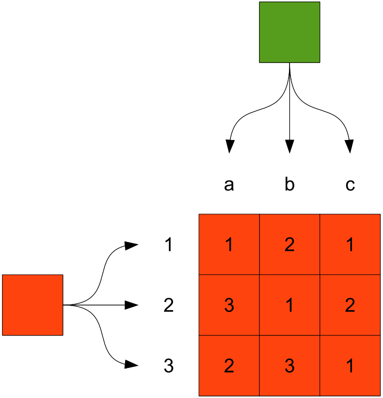
- A set of Players
- Each player performs Actions...
- ... and wants to maximize an Utility depending on other players actions
- Different players have different utilities
- Strategic form, also called multimatrix model



In this talk: finite games only (also simultaneous, perfect information, selfish players)

Strategic games: the setting

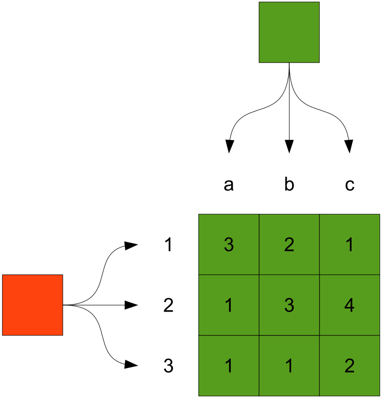
- A set of Players
- Each player performs Actions...
- ... and wants to maximize an Utility depending on other players actions
- Different players have different utilities
- Strategic form, also called multimatrix model



In this talk: finite games only (also simultaneous, perfect information, selfish players)

Strategic games: the setting

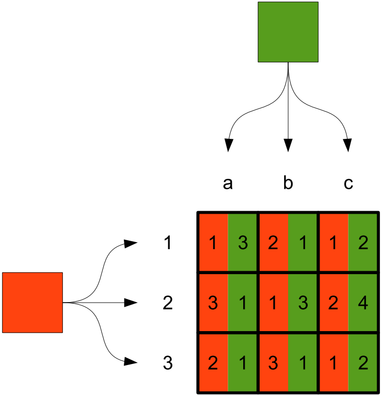
- A set of Players
- Each player performs Actions...
- ... and wants to maximize an Utility depending on other players actions
- Different players have different utilities
- Strategic form, also called multimatrix model



In this talk: finite games only (also simultaneous, perfect information, selfish players)

Strategic games: the setting

- A set of Players
- Each player performs Actions...
- ... and wants to maximize an Utility depending on other players actions
- Different players have different utilities
- Strategic form, also called multimatrix model



In this talk: finite games only (also simultaneous, perfect information, selfish players)

Solution concept

How can we tell when a player is satisfied?

Find a point where each player chooses the best strategy for him/herself... and for which no player can improve his/her utility by changing to another action:

Pure Nash Equilibrium

Many solution concepts...

- Mixed Nash equilibrium: probability distribution on actions as to maximize esperance of expected utility
- Pareto Nash equilibrium: such that no Pure Nash equilibrium has better utility for all players
- ...

Solution concept

How can we tell when a player is satisfied?

Find a point where each player chooses the best strategy for him/herself... and for which no player can improve his/her utility by changing to another action:

Pure Nash Equilibrium

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

Many solution concepts...

- Mixed Nash equilibrium: probability distribution on actions as to maximize esperance of expected utility
- Pareto Nash equilibrium: such that no Pure Nash equilibrium has better utility for all players
- ...

Solution concept

How can we tell when a player is satisfied?

Find a point where each player chooses the best strategy for him/herself... and for which no player can improve his/her utility by changing to another action:

Pure Nash Equilibrium

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	**2 4**
3	2 1	3 1	1 2

Many solution concepts...

- Mixed Nash equilibrium: probability distribution on actions as to maximize esperance of expected utility
- Pareto Nash equilibrium: such that no Pure Nash equilibrium has better utility for all players
- ...

Solution concept

How can we tell when a player is satisfied?

Find a point where each player chooses the best strategy for him/herself... and for which no player can improve his/her utility by changing to another action:

Pure Nash Equilibrium

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

	a	b	c
1	1	2	1
2	3	1	2
3	2	3	1

Many solution concepts...

- Mixed Nash equilibrium: probability distribution on actions as to maximize esperance of expected utility
- Pareto Nash equilibrium: such that no Pure Nash equilibrium has better utility for all players
- ...

Solution concept

How can we tell when a player is satisfied?

Find a point where each player chooses the best strategy for him/herself... and for which no player can improve his/her utility by changing to another action:

Pure Nash Equilibrium

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

	a	b	c
1	1	2	1
2	3	1	2
3	2	3	1

	a	b	c
1	3	1	2
2	1	3	4
3	1	1	2

Many solution concepts...

- Mixed Nash equilibrium: probability distribution on actions as to maximize esperance of expected utility
- Pareto Nash equilibrium: such that no Pure Nash equilibrium has better utility for all players
- ...

Solution concept

How can we tell when a player is satisfied?

Find a point where each player chooses the best strategy for him/herself... and for which no player can improve his/her utility by changing to another action:

Pure Nash Equilibrium

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

	a	b	c
1	1 3	2 1	1 2
2	3 1	1 3	2 4
3	2 1	3 1	1 2

	a	b	c
1	1	2	1
2	3	1	2
3	2	3	1

	a	b	c
1	3	1	2
2	1	3	4
3	1	1	2

Many solution concepts...

- Mixed Nash equilibrium: probability distribution on actions as to maximize esperance of expected utility
- Pareto Nash equilibrium: such that no Pure Nash equilibrium has better utility for all players
- ...

Pure Nash Equilibrium

Economists point of view:

- Accepted solution concepts do not guarantee uniqueness
- A game with no equilibrium or with multiple equilibria means that the modeler has failed to provide a full and precise prediction for what will happen
- Example: Nash theorem: any finite game in strategic form has a mixed Nash equilibrium

Modeling point of view

Computer scientists point of view:

- Problems sometimes do not have solution
- Game rules are given and the problem is to find a solution
- Example: efficient allocation in electricity grid market consist to connect producers and customers

Solving point of view

Pure Nash Equilibrium

Economists point of view:

- Accepted solution concepts do not guarantee uniqueness
- A game with no equilibrium or with multiple equilibria means that the modeler has failed to provide a full and precise prediction for what will happen
- Example: Nash theorem: any finite game in strategic form has a mixed Nash equilibrium

Modeling point of view

Computer scientists point of view:

- Problems sometimes do not have solution
- Game rules are given and the problem is to find a solution
- Example: efficient allocation in electricity grid market consist to connect producers and customers

Solving point of view

How to solve games?

A bit of formalism...

A game is a 3-uple $G = (P, A, U)$

- P is a set of players
- $A = (A_i)_{i \in P}$ is a set of actions for each player
- $U = (u_i)_{i \in P}$ is a set of utility functions for each player, $u_i : \Pi A \rightarrow \mathbb{R}$

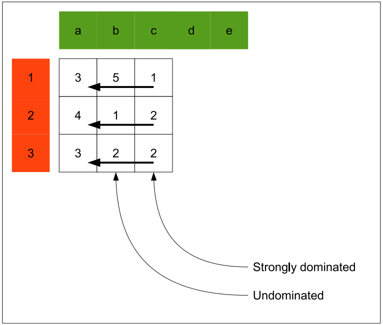
Strategies and Equilibrium

- A strategy for player i is the choice of an action $s_i \in A_i$
- A strategy profile is the given of a strategy for each player (a tuple $s \in \Pi A$)
- We denote by s_{-i} the strategy profile of players other than i, $s = (s_i.s_{-i})$
- s is winning for i if $\forall s'_i \neq s_i, u_i(s'_i.s_{-i}) \leq u_i(s_i.s_{-i})$
- s is a Pure Nash Equilibrium (PNE) if $\forall i, s_i$ is winning for i

Dominance

Types of dominance

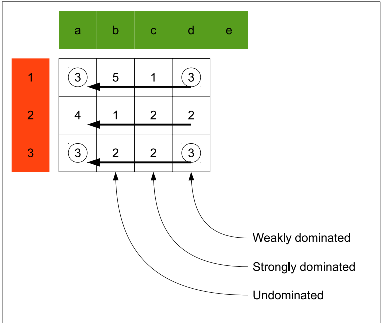
- s_i is strongly dominated by s'_i if $\forall s_{-i}, u_i(s_i.s_{-i}) < u_i(s'_i.s_{-i})$
- s_i is weakly dominated by s'_i if $\forall s_{-i}, u_i(s_i.s_{-i}) \leq u_i(s'_i.s_{-i})$
- s_i is never best response if $\forall s_{-i}, \exists s'_i \in A_i$ s.t. $u_i(s_i.s_{-i}) < u_i(s'_i.s_{-i})$



Dominance

Types of dominance

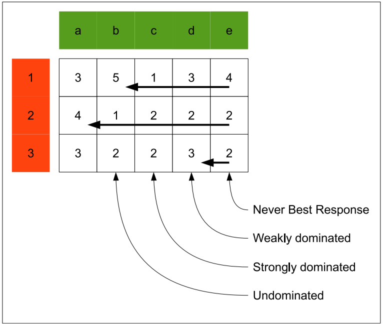
- s_i is strongly dominated by s'_i if $\forall s_{-i}, u_i(s_i.s_{-i}) < u_i(s'_i.s_{-i})$
- s_i is weakly dominated by s'_i if $\forall s_{-i}, u_i(s_i.s_{-i}) \leq u_i(s'_i.s_{-i})$
- s_i is never best response if $\forall s_{-i}, \exists s'_i \in A_i$ s.t. $u_i(s_i.s_{-i}) < u_i(s'_i.s_{-i})$



Dominance

Types of dominance

- s_i is strongly dominated by s'_i if $\forall s_{-i}, u_i(s_i.s_{-i}) < u_i(s'_i.s_{-i})$
- s_i is weakly dominated by s'_i if $\forall s_{-i}, u_i(s_i.s_{-i}) \leq u_i(s'_i.s_{-i})$
- s_i is never best response if $\forall s_{-i}, \exists s'_i \in A_i$ s.t. $u_i(s_i.s_{-i}) < u_i(s'_i.s_{-i})$



A generic algorithm to solve games

Solve

function solve(s): tuple
for $s \in \Pi A$ **do**
  if nash(s) **then**
    return s
  end if
end for
return not found

Deviation

function deviation(s, i): boolean
for $v \in A_i$, $v \neq s_i$ **do**
  if $u_i(v.s_{-i}) > u_i(s)$ **then**
    return true
  end if
end for
return false

Nash

function nash(s): boolean
for $i \in P$ **do**
  if deviation(s, i) **then**
    return false
  end if
end for
return true

Analysis

- Inefficient but still the baseline algorithm
- Implemented in the Gambit solver[a] along with IESDS [McKelvey and al, 2010]

[a] http://www.gambit-project.org/

Contents

Constraint Satisfaction Problems

Constraint Programming is a way of stating and solving problems using variables and constraints.

Definition (CSP)

A Constraint Satisfaction Problem (or CSP) is built out of 3 parts:

- V: a set of variables
- D: a set of domains
- C: a set of constraints

Here, we focus on Finite Domain CSP

Constraint Satisfaction Problems

Definition (CSP)

A CSP is a set of constraints.

Logically, it means the conjunction of the constraints.

Definition (Solution)

A solution is an assignment of all variables that satisfies all the constraints simultaneously.

Example ($X < Y < Z$)

With $X, Y, Z \in [1..3]$, the unique solution is $X = 1, Y = 2, Z = 3$.

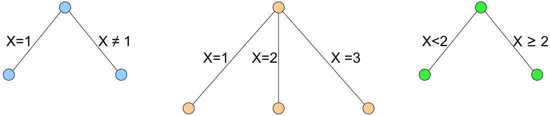
Tree search

Search state:

Let $\mathcal{C} = (V, D, C)$ a CSP.
A search state is composed of a current domain for each variable(a subset of D_X for each X).

Basic algorithm:

- If the current state represents a solution tuple, return this solution
- If the current state represent a non-solution tuple, fail
- Else, create a tree by adding to each branch a constraint:

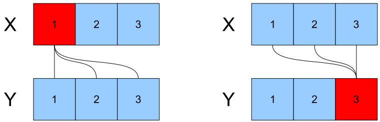


NP-complete problem: Importance of a good heuristics !

Local consistency

Reasoning on the CSP:

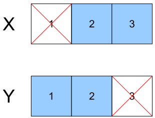
Consider $X > Y$:



It is not possible for X to take value 1 and being greater than Y
It is not possible for Y to take value 3 and being lesser than X

Domain Reduction for our Example

We can prune safely these two values:



Iteration of the technique up to a fixed point yields *consistency*

Modeling Language

CP as a modeling language includes other facilities:

- global constraints: common modeling element with a specific efficient algorithm
  - `all-different` ensures that all variables take different values
  - `cumulative` ensures that a schedule under resource constraints is feasible
  - `element` relates a value to its position in a table
  - ... the Global Constraint Catalogue records more than 300 global constraints [Beldiceanu and al]
- choice of heuristics
- modeling language: OPL, MiniZinc, ...
- multiple domains
- multiple extension: optimization, soft constraints, quantification, ...

Contents

Game representation problem

Normal form

- Normal form needs a matrix for representing utilities
- Matrix grow exponentially with the number of players
- 100 players $\times$ 2 actions $= 100 \times 2^{100}$ entries !

Compact representation is needed!

and many games have a natural understanding

Language for utilities

- if the utilities are not just random, they can (often) be expressed in a language
- better to understand utilities in terms of simple relationships than lookup in enormous tables

Constraint Games

The idea

Use CSP to express utilities

Constraint Satisfaction Game

A Constraint Satisfaction Game (CSG) is a 4-uple $CG = (P, V, D, G)$ where

- P is a set of players
- V is a set of variables, Player i controls $V_i \subseteq V$
- $D = (D_x)_{x \in V}$ defines a (finite) domain for each variable
- $G = (G_i)_{i \in P}$ is a family of CSP

Preferences

- CSPs provide a compact and natural formalism to express satisfaction for a player: G_i is called Goal of Player i
- Goals express preferences and an equilibrium may hold if a player is not satisfied (and cannot be)

Example of CSG

A simple example:

- Players: $P = \{X, Y, Z\}$
- Each player owns one variable: $V_X = \{x\}, V_Y = \{y\}$ and $V_Z = \{z\}$ with $D(x) = D(y) = D(z) = \{0, 1, 2\}$
- Goals are $G_X = \{x \neq y, x > z\}$, $G_Y = \{x \leq y, y > z\}$ and $G_Z = \{x + y = z\}$

Payoff multimatrix

$z = 0$		y = 0	y = 1	y = 2
x	0	(0,0,1)	(0,1,0)	(0,1,0)
	1	(1,0,0)	(0,1,0)	**(1,1,0)**
	2	(1,0,0)	(1,0,0)	(0,1,0)

$z = 1$		y = 0	y = 1	y = 2
x	0	(0,0,0)	(0,0,1)	(0,1,0)
	1	(0,0,1)	(0,0,0)	**(0,1,0)**
	2	(1,0,0)	(1,0,0)	**(0,1,0)**

$z = 2$		y = 0	y = 1	y = 2
x	0	(0,0,0)	(0,0,0)	**(0,0,1)**
	1	(0,0,0)	**(0,0,1)**	*(0,0,0)*
	2	**(0,0,1)**	*(0,0,0)*	*(0,0,0)*

in bold are **Nash equilibria** and italics *Nash equilibria with no player satisfied*

COG and hard constraints

Constraint Optimization Games

Constraint Programming provides an easy way to express optimization: add $\min(X)$ or $\max(X)$ to the goal of each player

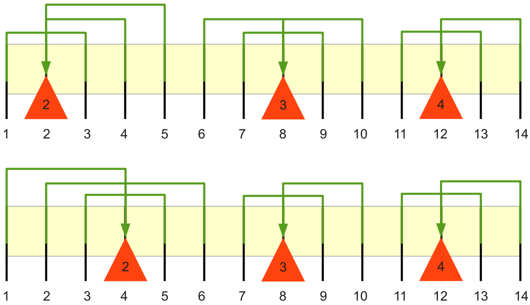
- Allows to represent in a natural way many useful games (see examples after)

Hard constraints

CSG/COG can be enhanced with a set of hard constraints (HC) to forbid invalid equilibria

- a strategy profile which does not satisfy HC cannot be an equilibrium and should not be checked for deviations
- impossible to represent in the matrix model (even by giving a dummy value)

Location Game (Hotelling, 1929)



Variables

- $P = \{1, .., n\}$
- $\forall i \in P, V_i = \{l_i\}$
- $\forall i \in P, D(l_i) = \{1, \ldots m\}$
- cost_{ic}: define the cost customer c has to pay if he/she chooses the stand of seller i.
- min_c: defines the minimal cost customer c has to pay for an ice cream.
- choice_{ic}: boolean variable takes 1 if customer c chooses seller i.
- benefit_i: defines the number of customers actually buying from seller i.

Location Game

Hard constraints

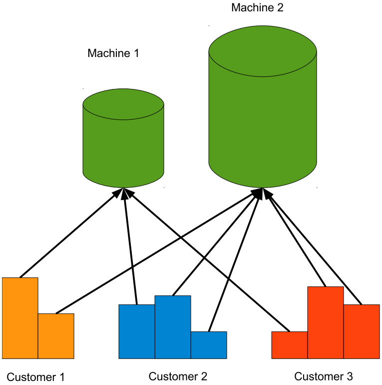
- no two vendors are located at the same place: *all_different*$(l_1, l_2, \ldots, l_n)$
- $\forall i \in P, \forall c \in [1..m]$, $\text{cost}_{ic} = |c - l_i| + p_i$
- $\forall c \in [1..m]$, $\min_c = \min(\text{cost}_{1c}, \ldots, \text{cost}_{nc})$
- $\forall c \in [1..m], (\min_c = \text{cost}_{ic}) \leftarrow (\text{choice}_{ic} = 1)$
- $\forall c \in [1..m], \sum_{i \in P} \text{choice}_{ic} = 1$

Goal

- G_i: $\text{benefit}_i = p_i . \sum_{c \in [1..m]} \text{choice}_{ic}$
- Optimization condition $Opt_i = \max(\text{benefit}_i)$

Cloud Resource Allocation Game [Jalaparti and al, 2010]

- Cloud provider: m machines
- n Customers. Customer i wants to allocate m_i tasks
- Machine m_j has capacity c_j and cost $l_j(x) = x \times u_j$
- Clients choose their machine and minimize cost
- Machines capacities should be respected



CRAG constraint model

- $P = \{1, .., n\}$
- $\forall i \in P, V_i = \{r_{i1}, ..., r_{im_i}\}$
- $\forall i \in P, \forall k \in [1, ..., m_i], D(r_{ik}) = \{1, \ldots m\}$
- C is composed of the following constraints:
  - channelling constraints: $(r_{ik} = j) \leftrightarrow (choice_{ijk} = 1)$
  - capacity constraints: $\forall j \in [1, .., m]$, $\sum_{i \in [1..n]} \sum_{k \in [1..m_i]} choice_{ijk} \times d_{ik} \leq c_j$
- $\forall i \in P$, G_i is composed of the following constraint:

$$cost_i = \sum_{j=1..m} \sum_{k=1..m_i} choice_{ijk} \times l_j(d_{ik})$$

- $\forall i \in P, Opt_i =$ Minimize $(cost_i)$

ConGa: A Complete Algorithm

A result by [Gottlob and al, 2005]

- Nash Constraint N_i for Player i: encodes tuples $t = (s_i, s_{-i})$ such that s_i is a best response to s_{-i} (not unique)
- Theorem: $\bowtie_{i \in P} N_i = PNE$

In Conga, we compute incrementally the N_i

Tree-search algorithm

- The idea is to traverse all tuples of the search space using a complete ordering of players and values
- Record each player's undominated strategies in a table
- Pruning when a tuple has already been proved subject to deviation (complete detection)
- Pruning when a tuple is NBR (partial detection)
- Constraint solver is used to compute hard constraints and deviations

Recording Nash Constraints

Nash checking for a tuple s:

- Each player is examined in turn, in decreasing order from n to 1
- First lookup in tables for already computed deviations
- If not found, compute deviation with the solver and record best response in table
- If stable, then check previous player
- If Player 1 is stable, then record Nash equilibrium

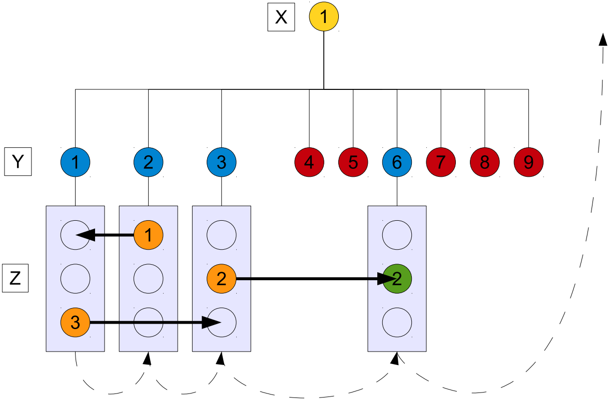
Deleting unuseful table entries

Tables may grow very large

- In theory, tables for Nash constraints can be exponential in size
- In practice, the size is kept reasonable
- Complete ordering of variables and values gives a lexicographic traversal of the search space
- Players at high level only record Nash candidates which have been checked by lower levels
- Once a player has backtracked, all subsequent players can delete tables

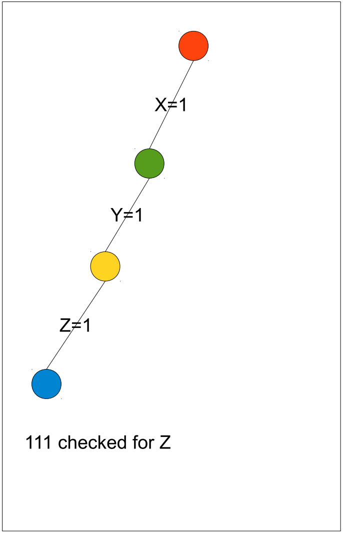
Never Best Responses pruning

Online detection of NBR

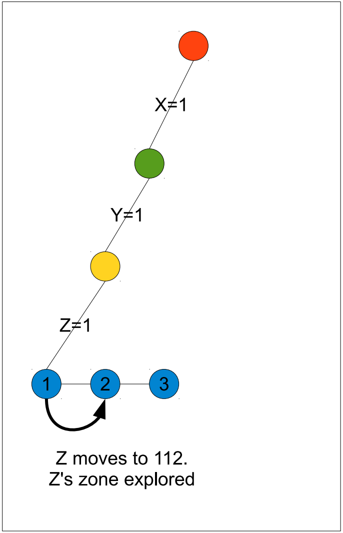


- We use a counter to record how many elements of the subsequent subspace have been checked
- Once the counter reaches 0, only recorded subsequent elements are checked
- Needs to check the end of the table
- Then backjump to upper level

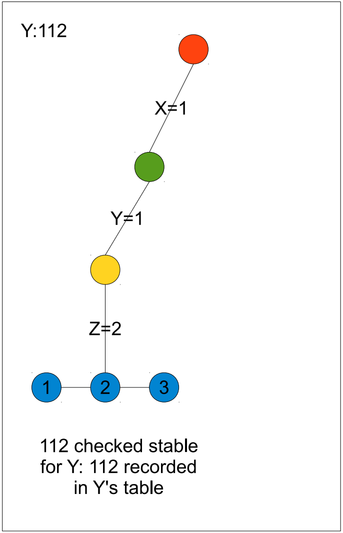
A short example



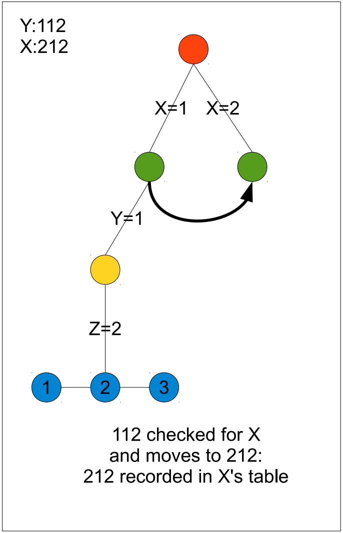
A short example



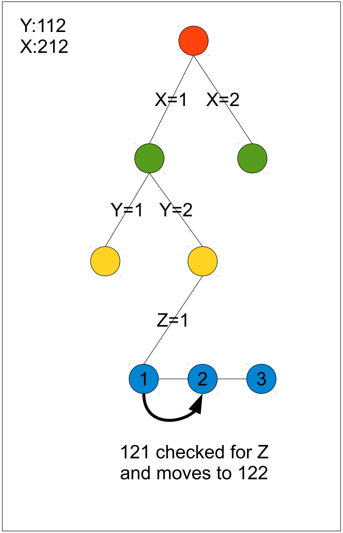
A short example



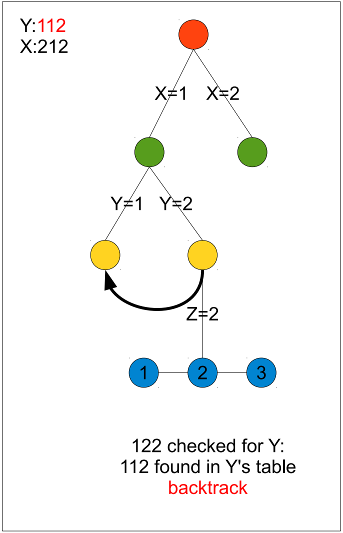
A short example



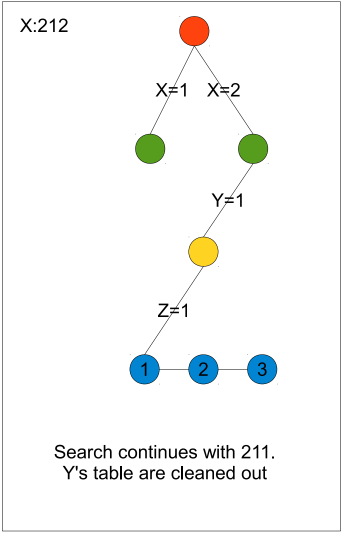
A short example



A short example



A short example



Experimental results

Conga compared to Gambit

Name	NF gen	Gambit	Enum1	ConGa	#PNE
GTTA.3.100	1	17	4	**0**	1
GTTA.4.100	113	1844	312	**2**	1
GTTA.5.100	*TO*	-	4032	**168**	1
GTTA.6.100	*TO*	-	*TO*	**19990**	1
LG(GV).2.1000	1	134	339	**6**	0
LG(GV).2.2000	6	655	1441	**31**	0
LG(GV).2.3500	17	5337	6789	**93**	0
LG(GV).2.5000	34	7786	10000	**200**	0
LG(GV).2.20000	552	*MO*	*TO*	**3389**	0
MEG.3.100	1	13	0	**0**	100
MEG.4.100	91	1555	28	**6**	100
MEG.5.100	*TO*	-	2082	**403**	100
MEG.6.100	*TO*	-	*TO*	**18102**	100
MEG.30.2	8784	*MO*	**423**	503	2
MEG.35.2	TO	-	**10619**	16917	2
TD.3.99	3	14	0	**0**	1
TD.4.99	76	1572	26	**7**	1
TD.5.99	8930	*MO*	2028	**446**	1
TD.6.99	*TO*	-	*TO*	**14731**	1
CG.7.15	253	*MO*	70	**27**	630
CG.8.15	4613	*MO*	1019	**371**	1680
CG.9.15	*TO*	-	17361	**5880**	5040
LG(HC).4.30	N/A	N/A	26	**6**	24
LG(HC).5.30	N/A	N/A	778	**257**	240
LG(HC).6.30	N/A	N/A	*TO*	**13180**	2160
CRAG.7.9	N/A	N/A	323	**57**	1
CRAG.8.9	N/A	N/A	3300	**540**	1
CRAG.9.9	N/A	N/A	17723	**4975**	1

- Times are in seconds
- NF gen = Normal form generation
- enum1 = Constraint Game solver without Nash constraint computation and NBR pruning
- Time out is 9000s for generation and 20000s for solving
- Tables grow up to 240 GB for MEG.5.100
- Improvement of 1 to 2 orders of magnitude over Gambit

Conclusion

Summary

- PNE are useful for implementing agreements between agents
- Constraint Games allow for representing games in a compact and natural way
- Complete solver: Conga outperforms state-of-the art solver Gambit by 1 to 2 orders of magnitude

Perspectives

- Dynamic heuristics
- Propagation of constraints
- Difficulties to include symmetries in the model

Thank you for your attention

Questions?