General Game Playing: a Challenge for AI

Sylvain LAGRUE

lagrue@cril.fr - http://syl.lagrue.net

Université d'Artois - CRIL CNRS - France

AI4Life – 2018-05-09

Game: Definition

> *A game is a system in which **players** engage in an artificial conflict, defined by **rules**, that results in a quantifiable **outcome**.*
>
> — Katie Salen and Eric Zimmerman

Game: Definition

> *A game is a system in which **players** engage in an artificial conflict, defined by **rules**, that results in a quantifiable **outcome**.*
>
> — Katie Salen and Eric Zimmerman

Strategy Game

- Archetype of intelligent behavior for Human Being
- In Strategy Games, physical abilities are not necessary: intelligence, focusing, and knowledge prevail

Game: Definition

> *A game is a system in which **players** engage in an artificial conflict, defined by **rules**, that results in a quantifiable **outcome**.*
>
> — Katie Salen and Eric Zimmerman

Strategy Game

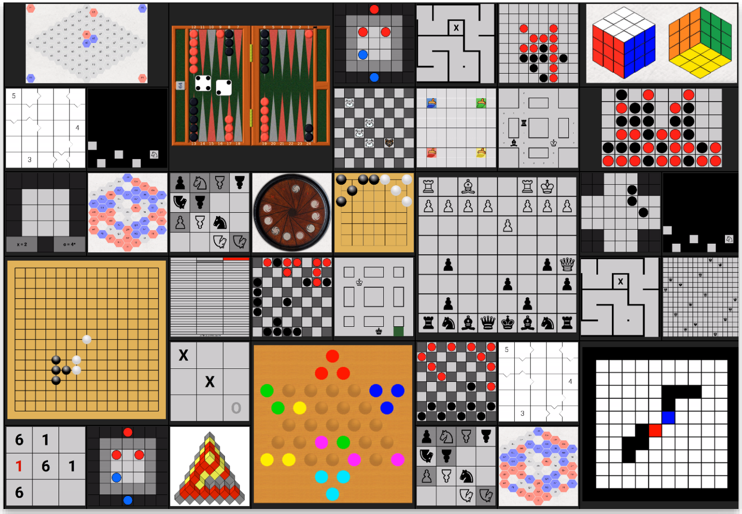
- Archetype of intelligent behavior for Human Being
- In Strategy Games, physical abilities are not necessary: intelligence, focusing, and knowledge prevail

Applications

- Entertainment
- Agent behavior in economics
- Decision support system
- Education (eg. "serious games")

Example of Games

- With complete information: Chess, GO, Checkers, Xiangqi
- Chance Games: Backgammon
- With incomplete information: Poker, Bridge
- Simultaneous games: Rock Paper Scissors
- But also asymmetric games, cooperative games, non-zero-sum games, ...



AI and Games

For scientist and AI researchers

> *Chess is the Drosophila of Artificial Intelligence.*
>
> — Alexander Kronrod (1921-1986)

AI and Games

For scientist and AI researchers

> *Chess is the Drosophila of Artificial Intelligence.*
>
> — Alexander Kronrod (1921-1986)

- Controlled environment (no physical constraints, fixed rules, rational players,...)
- Playground for experimenting many algorithms/architectures
- Technology showcase

AI and Games

For general public: exert fascination...

Fascination...

AI and Games

Timeline et milestones

- 1950 *Article: Programming a Computer for Playing Chess*
- 1979 *BKG 9.8*
- 1997 *Deep Blue*
- 2007 *Checker Solved*
- 2016 *AlphaGo*
- 2017 *Libratus*
- 2018 *AlphaZero*

Programming a Computer for Playing Chess (1950)

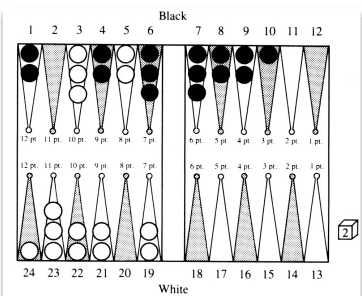
- 1950: Seminal article for Chess programming from **Claude Shannon**
  - 2 algorithms for playing chess
    - "Type A": brute force (adaptation of minimax)
    - "Type B": "fine" selection of interesting branches
  - Shannon also built an automate that plays some endings with up to six pieces
- 1951: **Alan Turing** proposed a program, developed on paper, able to play a full game of chess

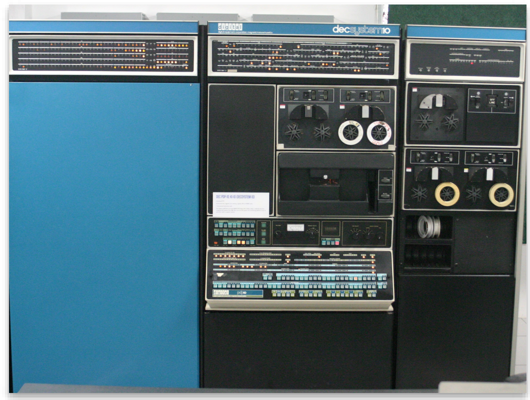
Claude Shannon and the Chessmaster Edward Laske

BKG 9.8, Hans J. Berliner (1979)

- In 1979 BKG 9.8 defeated the world champion of Backgammon, Luigi Villa, by the score of 7–1
- Main idea: Using fuzzy logic for the transitions between the 3 phases of game (opening/middle game/end game)

Hans J. Berliner



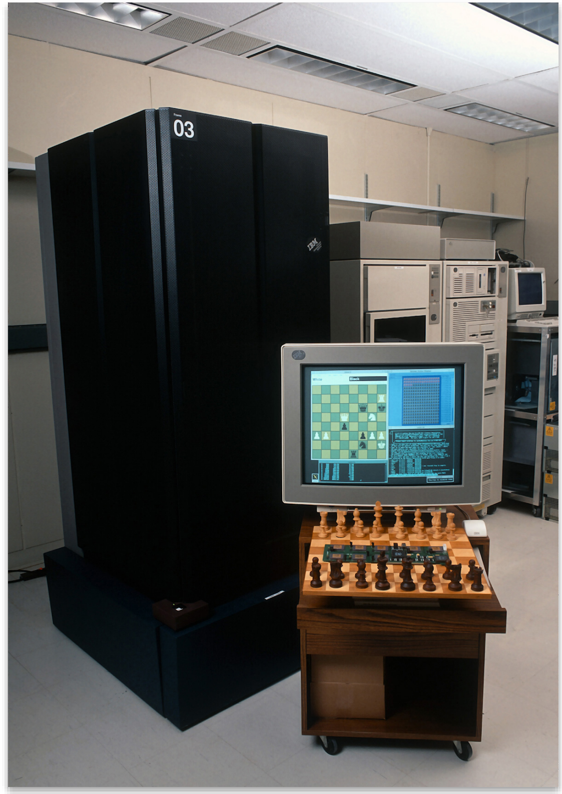


DEC PDP-10

IBM's Deep Blue beats Garry Kasparov (1997)

Deep Blue vs Garry Kasparov (3.5/2.5 - 2w/1w, 3 draws)

- Massively parallel supercomputer (256 dedicated CPUs)
- 11.4 GFlop/s
- Able to evaluate 200 million positions per second



Chinook solved Checkers (2007)

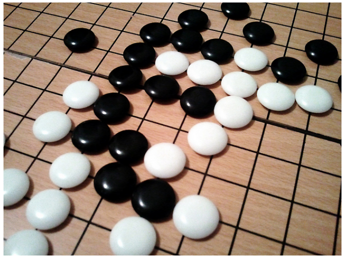
- Jonathan Schaeffer et al.
- "Solved Checkers": after an *exhaustive search*, a strategy that leads to a draw against perfect player was found

Against Marion Tinsley in 1992 (4-2 and 33 draws for Tinsley)

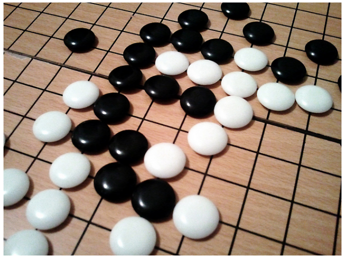
Go

Branching factor:

- Checkers (8x8) = 2.8
- Chess = 35
- Go (19x19) = 250



Go



Branching factor:

- Checkers (8x8) = 2.8
- Chess = 35
- Go (19x19) = 250

Monte Carlo Go (1992)

- First use Monte Carlo Tree Search (MCTS) for Go (Bernd Brügmann)

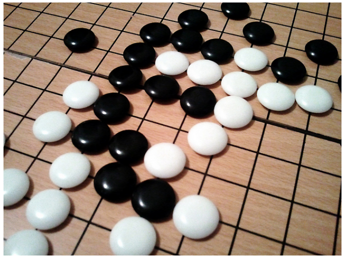
MoGo (2008)

- Introduction of UCT (Upper bound Confidence for Tree = MCTS + UCB Upper Confidence Bounds)

Go

Branching factor:

- Checkers (8x8) = 2.8
- Chess = 35
- Go (19x19) = 250



Monte Carlo Go (1992)

- First use Monte Carlo Tree Search (MCTS) for Go (Bernd Brügmann)

MoGo (2008)

- Introduction of UCT (Upper bound Confidence for Tree = MCTS + UCB Upper Confidence Bounds)

Alpha Go (2016)

- Combines MCTS + deep neural networks + reeinforcement learning (from human games and from itself)
- Beat world champion Lee Sedol 4-1 (March 2016)

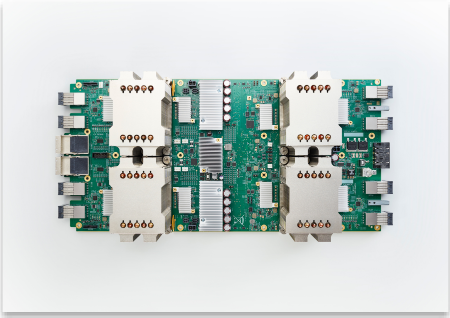
Libratus and poker (2017)

- From Tuomas Sandholm
- Winner against 4 professional players in **heads up no-limit Texas hold'em**
- Deep neural networks + Reinforcement learning **from scratch** (Using CFR+ - counterfactual regret minimization +)
- 15 million core hours (1,712 years) of computation

Alpha Zero (late 2017)

Principles

- Combines MCTS + deep neural networks + reeinforcement learning (**from scratch**)
- Beat best computer programs in Chess (*Stockfish*), Shogi (*elmo*) and Go (*Alpha Go*)
- Works on a computer with only 4 TPUs (Tensor Processing Units)
- Evaluates 80,000 positions per s vs 70,000,000 for Stockfish 8
- Only 9 hours of learning to beat Stockfish (3 days to beat AlphaGo Lee)



A Google Cloud TPU board

Alpha Zero (late 2017)

But...

- Using 5,064 TPUs (5000 1st gen. + 64 2nd gen.) for learning
- 9 x 5,064 ≈ 5 years of TPU time
- 9 x (5,000 x 15 + 64 x 30) ≈ 79 years of CPU (Intel Haswell) time...

datacenter

Alpha Zero (late 2017)

But...

- Using 5,064 TPUs (5000 1st gen. + 64 2nd gen.) for learning
- 9 x 5,064 ≈ 5 years of TPU time
- 9 x (5,000 x 15 + 64 x 30) ≈ 79 years of CPU (Intel Haswell) time...

datacenter

- The game rules are **hard-coded**

Questions

?

> *Question 1: How to create a program that can play "efficiently" to any game without hard-coded rules?*

Questions

? *Question 1: How to create a program that can play "efficiently" to any game without hard-coded rules?*

? *Question 2: How to create a program that can play "efficiently" to any game without hard-coded rules in a decent time?*

Questions

?

Question 1: How to create a program that can play "efficiently" to any game without hard-coded rules?

?

Question 2: How to create a program that can play "efficiently" to any game without hard-coded rules in a decent time?

⇛ General Game Playing

Questions

?

Question 1: How to create a program that can play "efficiently" to any game without hard-coded rules?

?

Question 2: How to create a program that can play "efficiently" to any game without hard-coded rules in a decent time?

⇛ General Game Playing

?

*Question 3: How to create a program that can play "efficiently" to any game without hard-coded rules in a decent time on **my** computer?*

General Game Playing (GGP)

Various approaches have been proposed since the 2000s

- Automatic constructions of evaluation functions
- Logic programming/ASP (Answer Set Programming)
- Monte Carlo methods (MCTS)
- **Constraint-based methods**

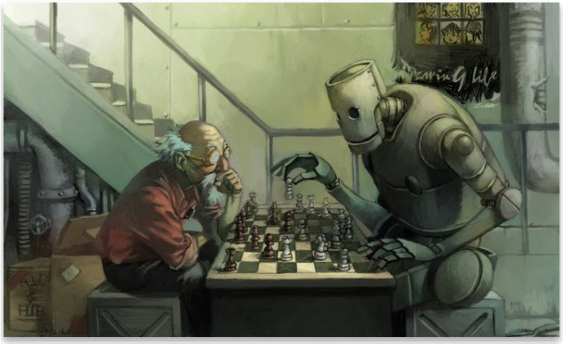
General Game Playing (GGP)

Various approaches have been proposed since the 2000s

- Automatic constructions of evaluation functions
- Logic programming/ASP (Answer Set Programming)
- Monte Carlo methods (MCTS)
- **Constraint-based methods**

Some applications

- Educational purpose
- A game companion
- It can model sequential decision problems in mono or multiagent environments

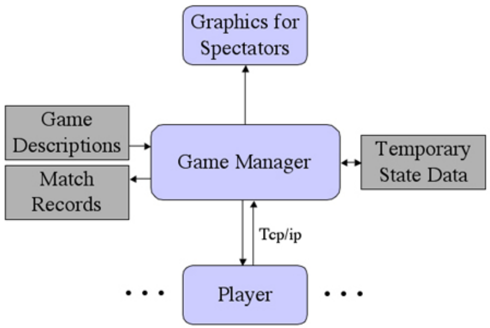


The International General Game Playing Competition

IGGPC

- Organized by AAAI/Stanford University
- From 2005, last in 2016, next in February of 2019 (at the AAAI conference)
- http://ggp.stanford.edu/iggpc

Rules



Game manager description from http://ggp.stanford.edu

- Time to understand rules: from 1' to 20'
- Time per move: from 30" to 3'

What do we need for GGP?

- Representation of game rules
- Understanding these rules (playing legal moves)
- Decision making (playing "best" legal moves)

Game Description Language (GDL)

Generic language for representing any strategy game

- Derived from logic programming with negation and equality
- Players and game-objects are described by constants while fluents and actions by terms
- Atoms are constructed from a finite set of relation symbols and variable symbols

GDL can describe

- All strategy games with complete information
- Simultaneous and sequential games
- Cooperative and competitive games

GDL-II can describe

- All chance games
- All games with incomplete information

Expressiveness

- GDL is Turing-complete, i.e. it can be used to simulate any Turing machine

Game Description Language (GDL)

GDL Keywords

Keyword	Description
`role(P)`	`P` is a player
`init(F)`	the fluent `F` is part of the initial state
`true(F)`	`F` is part of the current state
`legal(P, M)`	`P` can do the move `M`
`does(P, M)`	the move of `P` is `M`
`next(F)`	`F` is part of the next state
`terminal`	the current state is terminal
`goal(P, N)`	`P` receives `N` as a reward in the current state

Keyword Description

Game Description Language (GDL)

GDL Keywords

Keyword	Description
`role(P)`	`P` is a player
`init(F)`	the fluent `F` is part of the initial state
`true(F)`	`F` is part of the current state
`legal(P, M)`	`P` can do the move `M`
`does(P, M)`	the move of `P` is `M`
`next(F)`	`F` is part of the next state
`terminal`	the current state is terminal
`goal(P, N)`	`P` receives `N` as a reward in the current state

GDL-II Keywords

Keyword	Description	Note
`random`	is the player environment	games of chance
`sees(P, R)`	`P` perceives `R`	games with incomplete information

A simple example: Tic-Tac-Toe

roles

```
role(xplayer)
role(oplayer)
```

A simple example: Tic-Tac-Toe

roles

```
role(xplayer)
role(oplayer)
```

initial state

```
init(cell(1, 1, blank))
init(cell(1, 2, blank))
...
init(cell(3, 3, blank))
```

A simple example: Tic-Tac-Toe

roles

```
role(xplayer)
role(oplayer)
```

initial state

```
init(cell(1, 1, blank))
init(cell(1, 2, blank))
...
init(cell(3, 3, blank))
```

```
init(control(xplayer))
```

A simple example: Tic-Tac-Toe

roles

```
role(xplayer)
role(oplayer)
```

initial state

```
init(cell(1, 1, blank))
init(cell(1, 2, blank))
...
init(cell(3, 3, blank))
```

```
init(control(xplayer))
```

legal moves

```
legal(PLAYER, mark(X, Y)) ← true(cell(X, Y, blank)), true(control(PLAYER))
```

```
legal(xplayer, noop) ← true(control(oplayer))
legal(oplayer, noop) ← true(control(xplayer))
```

A simple example: Tic-Tac-Toe (2)

game state and control updates

```
;; new marked cell

next(cell(X, Y, x)) ← does(xplayer, mark(X, Y)))
next(cell(X, Y, o)) ← does(oplayer, mark(X, Y)))
```

A simple example: Tic-Tac-Toe (2)

game state and control updates

```
;; new marked cell

next(cell(X, Y, x)) ← does(xplayer, mark(X, Y)))
next(cell(X, Y, o)) ← does(oplayer, mark(X, Y)))
```

```
;; all cells not marked in this turn

next(cell(X, Y, M)) ←
    true(cell(X Y M)), does PLAYER (mark M N),
    distinct X M, distinct Y N
```

A simple example: Tic-Tac-Toe (2)

game state and control updates

```
;; new marked cell

next(cell(X, Y, x)) ← does(xplayer, mark(X, Y)))
next(cell(X, Y, o)) ← does(oplayer, mark(X, Y)))
```

```
;; all cells not marked in this turn

next(cell(X, Y, M)) ←
    true(cell(X Y M)), does PLAYER (mark M N),
    distinct X M, distinct Y N
```

```
;; control

next(control(xplayer)) ← true(control(oplayer))
next(control(oplayer)) ← true(control(xplayer))
```

A simple example: Tic-Tac-Toe (3)

terminal states

```
terminal ⟵ line(x)
terminal ⟵ line(o)
terminal ⟵ not open
```

A simple example: Tic-Tac-Toe (3)

terminal states

```
terminal ← line(x)
terminal ← line(o)
terminal ← not open
```

rewards

```
goal(xplayer, 100) ← line(x)
goal(oplayer, 0) ← line(x)

goal(oplayer, 100) ← line(o)
goal(xplayer, 0) ← line(o)

goal(PLAYER, 50) ← not line(x), not line(o), not open
```

A simple example: Tic-Tac-Toe (4)

additional functions

```
row(M) ←
    true(cell(X, 1, M)),
    true(cell(X, 2, M)),
    true(cell(X, 3, M))

column(M) ←
    true(cell(1, Y, M))
    true(cell(2, Y, M))
    true(cell(3, Y, M))

diagonal(M)) ←
    true(cell(1, 1, M))
    true(cell(2, 2, M))
    true(cell(3, 3, M))

diagonal(M)) ←
    true(cell(1, 3, M))
    true(cell(2, 2, M))
    true(cell(3, 1, M))

line(M) ← row(M)
line(M) ← column(M)
line(M) ← diagonal(M)

open ← true(cell(X, Y, blank))
```

A Constraint based method for GGP

- A **Constraint Satisfaction Problem** (CSP) consists of a set of variables, a set of possible values for each variable, and constraints on the valuation of the variables
- A **Stochastic** Constraint Satisfaction Problem is a CSP with some stochastic variables

A Constraint based method for GGP

- A **Constraint Satisfaction Problem** (CSP) consists of a set of variables, a set of possible values for each variable, and constraints on the valuation of the variables
- A **Stochastic** Constraint Satisfaction Problem is a CSP with some stochastic variables

A SCSP is a 6-tuple 〈**X** , **Y** , **D**, **P**, **C**, **θ**〉:

- **X** is an ordered set of **n** variables
- **Y** is the subset of **X** specifying stochastic variables
- **D** is a mapping from **X** to finite domains
- **P** is a mapping from **Y** to probability distributions over the domains of stochastic variables
- **C** is the set of constraints
- **θ** is a threshold value in the interval [0, 1]

Example

- $\mathbf{X} = \{ x_1, x_2, y \}$
- $\mathbf{Y} = \{ y \}$
- $\mathbf{D_{x_1}} = \mathbf{D_{x_2}} = \{ 1, 2 \}$
- $\mathbf{D_y} = \{ 0, 1, 2 \}$
- $\mathbf{C} = \{ x_1 = x_2, y < x_1 \}$
- **P** = uniform distribution on D_y
- $\boldsymbol{\theta} = 2/3$

Example

- $\mathbf{X} = \{ x_1, x_2, y \}$
- $\mathbf{Y} = \{ y \}$
- $\mathbf{D_{x_1}} = \mathbf{D_{x_2}} = \{ 1, 2 \}$
- $\mathbf{D_y} = \{ 0, 1, 2 \}$
- $\mathbf{C} = \{ x_1 = x_2, y < x_1 \}$
- **P** = uniform distribution on D_y
- $\boldsymbol{\theta} = 2/3$

Definition µSCSP

> *A Stochastic Constraint Satisfaction Problem at one stage (µSCSP) is a SCSP where all decision variables have higher priority than all stochastic variables.*

Solution of a SCSP

Definitions

- Policy: ordered tree on **X**
- Utility of a policy: sum of the leaf utilities weighted by their probabilities
- *Solution of a SCSP*: policy **π** whose expected utility is greater than or equal to the threshold **θ** and satisfying all constraints

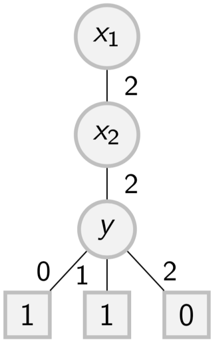
Solution of a SCSP

Definitions

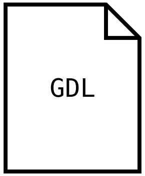
- Policy: ordered tree on **X**
- Utility of a policy: sum of the leaf utilities weighted by their probabilities
- *Solution of a SCSP*: policy **π** whose expected utility is greater than or equal to the threshold **θ** and satisfying all constraints

Example

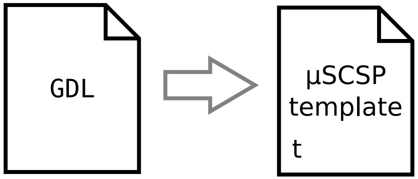
- The two decision variables $\mathbf{x_1}$ and $\mathbf{x_2}$ take the value **2**
- According to the uniform distribution, the stochastic variable **y** can take the values **0**, **1** and **2**
- Expected utility = **2/3**



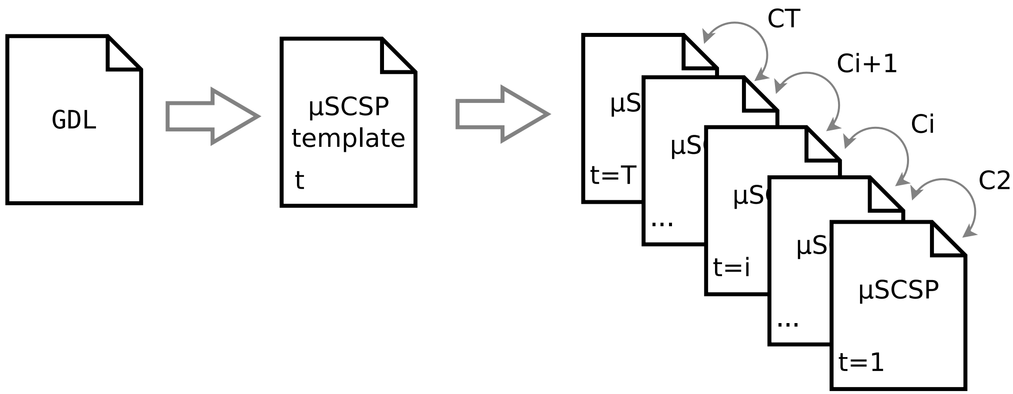
From GDL to SCSP



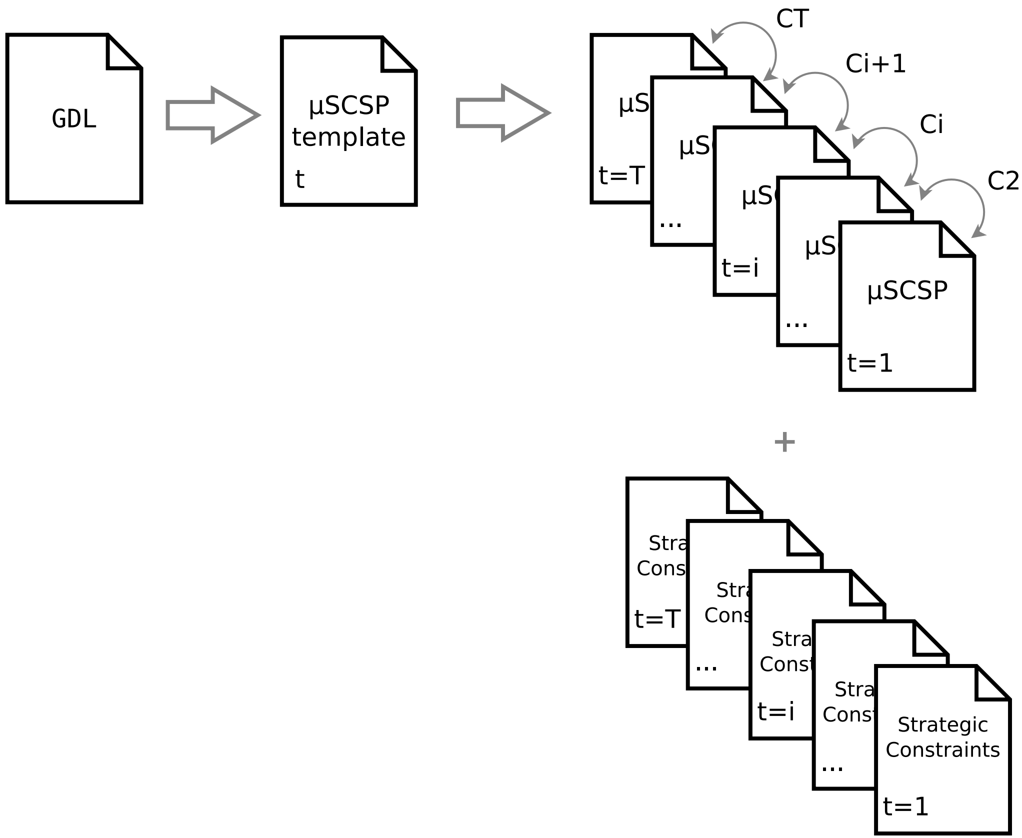
From GDL to SCSP



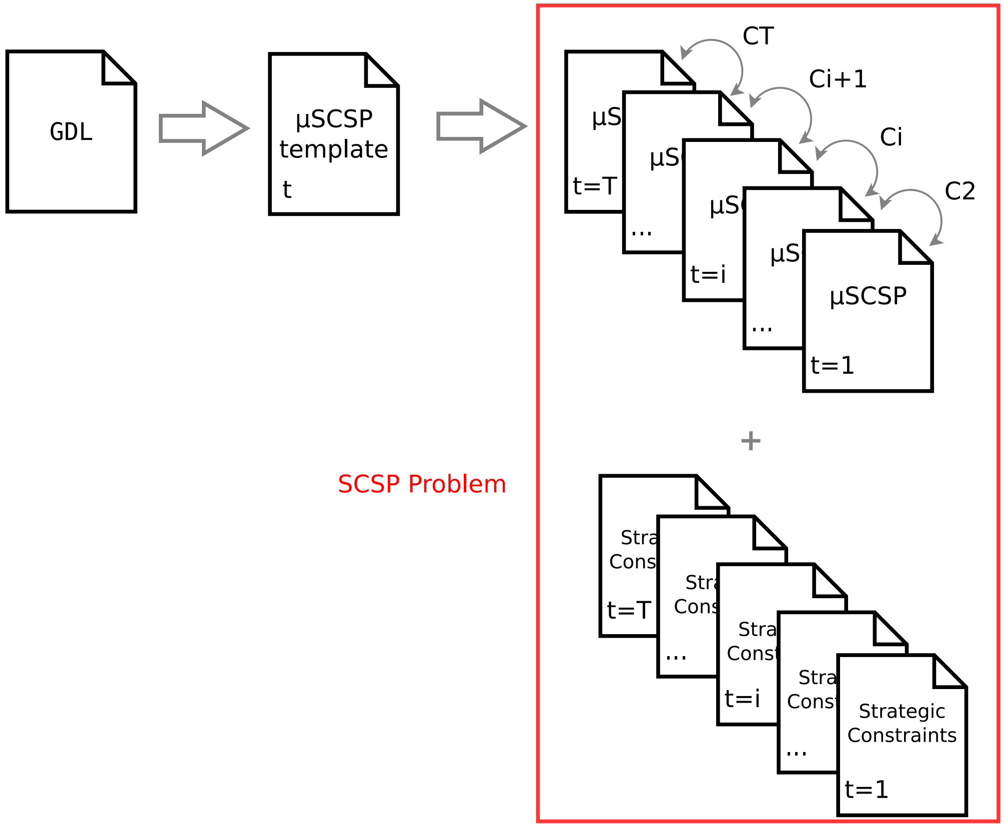
From GDL to SCSP: Legal moves



From GDL to SCSP: Choosing a move



From GDL to SCSP: Resolution



Resolution (2)

- MAC-UCB
  - Some preprocessing (constraint fusion, Single Arc Consistency, etc.)
  - Evaluation of the rewards of non final states: Monte-Carlo (UCB)
  - No-good tables
- Taking symmetries into account
  - Structure symmetries
  - Strategy symmetries

Resolution (2)

- MAC-UCB
  - Some preprocessing (constraint fusion, Single Arc Consistency, etc.)
  - Evaluation of the rewards of non final states: Monte-Carlo (UCB)
  - No-good tables
- Taking symmetries into account
  - Structure symmetries
  - Strategy symmetries

Formal results

Under small restrictions, and given a horizon T, all our SCSP encodings and resolution processes are proved to be valid with respect to the semantics of GDL and GDL + `random`.

Experimental results

Conducted on Intel Xeon E5-2643 CPU 3.3 GHz with 64 GB of RAM and four threads
300 matches for each deterministic game, 1000 matches for each stochastic game

Deterministic GDL games				
Game	MAC-UCB	**uct-sym**	**grave-sym**	**sancho**
Amazons torus 10×10	84.2 (±1.2%)	98.1 (±1.7%)	86.7 (±2.7%)	86.2 (±3.1%)
Breakthrough suicide	93.0 (±2.3%)	81.9 (±3.7%)	73.2 (±2.9%)	77.8 (±4.0%)
Chess	76.4 (±2.5%)	95.3 (±2.1%)	95.4 (±2.5%)	87.9 (±2.1%)
Connect Four 20×20	87.5 (±3.5%)	100.0 (±0.0%)	88.5 (±2.2%)	96.0 (±0.9%)
Copolymer with pie	73.9 (±1.5%)	93.3 (±0.5%)	91.6 (±1.8%)	77.9 (±3.6%)
English Draughts	85.1 (±2.8%)	97.4 (±1.3%)	71.2 (±3.1%)	59.3 (±1.5%)
Free For All 2P	53.4 (±0.7%)	84.8 (±1.9%)	72.3 (±1.6%)	71.2 (±2.3%)
Hex	84.0 (±1.4%)	100.0 (±0.0%)	89.8 (±2.9%)	78.1 (±1.5%)
Pentago	53.1 (±1.5%)	66.2 (±2.8%)	58.4 (±2.8%)	54.3 (±0.9%)
Sheep and Wolf	74.8 (±3.2%)	94.6 (±0.9%)	63.2 (±3.6%)	62.1 (±1.5%)
Shmup	58.0 (±1.7%)	63.7 (±2.2%)	52.1 (±0.2%)	53.0 (±0.6%)
TicTac Chess 2P	94.9 (±3.4%)	96.5 (±0.4%)	93.2 (±2.3%)	86.1 (±3.3%)
TTCC4 2P	84.4 (±2.3%)	97.2 (±2.1%)	85.7 (±3.1%)	65.8 (±4.1%)
Reversi Suicide	72.2 (±3.2%)	100.0 (±0.0%)	78.7 (±2.2%)	58.2 (±2.2%)
Stochastic GDL games				
Backgammon	92.1 (±2.7%)	96.1 (±1.4%)	86.8 (±3.9%)	100.0 (±0.0%)
Can't Stop	88.2 (±1.7%)	96.8 (±1.7%)	93.7 (±3.2%)	100.0 (±0.0%)
Kaseklau	73.5 (±3.6%)	72.1 (±0.9%)	60.2 (±3.2%)	88.1 (±2.6%)
Pickomino	75.4 (±1.8%)	82.4 (±2.8%)	95.6 (±1.0%)	92.1 (±2.9%)
Yahtzee	87.4 (±1.6%)	83.1 (±3.3%)	60.9 (±2.5%)	91.8 (±3.3%)

Conclusion

Programs of General Game Playing

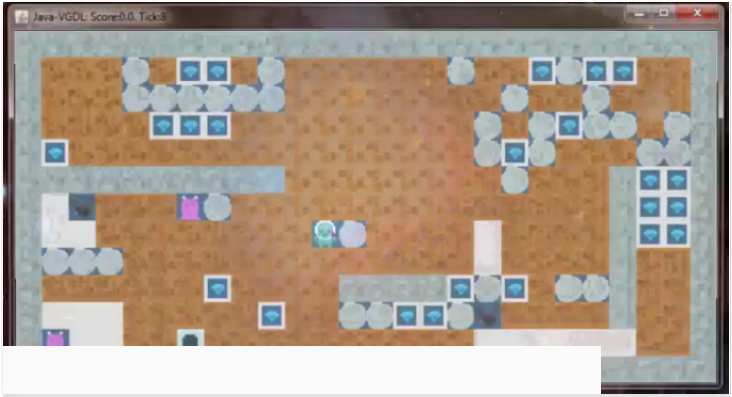
- Far from the level of the dedicated programs...
- ... But interesting level in a short time!
- Not hard-coded rules

Our contributions

- GGP player based on SCSP
- 2016 IGGPC winner (reigning world champion)
- Some alternatives to deep learning exist when ressources are limited!

Some hot topics

- Learning game rules from matches (Inductive GDL)
- Using GDL for explanation: open the black box!
- General Video Games AI (GVG-AI) and the Video Game Definition Language (VGDL)



Xin cảm ơn!

sapin

fullscreen 16/9/fullscreen 1280x1024

Xin cảm ơn!