Solving Games

Journées MAFTEC 2018

Tristan Cazenave
LAMSADE
Université Paris-Dauphine
PSL

Tristan.Cazenave@dauphine.fr

Outline

- Solving games
- Combinatorial Game Theory
- Proof Number Search
- Product Propagation
- Retrograde Analysis
- Symmetries
- Threat Search
- Monte Carlo Tree Search

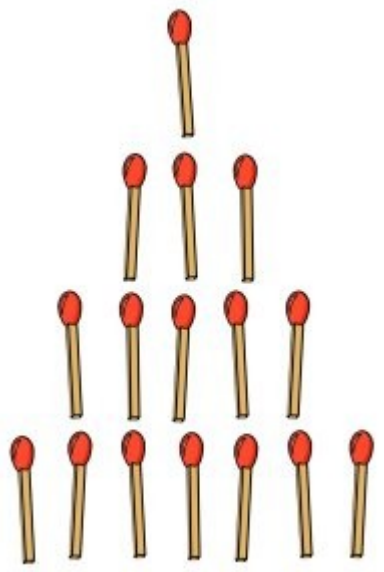
Solving Games

- Alpha-Beta has been used to solve games
- Small board Go
- Atarigo
- Renju
- Lines of Action
- Amazons Endgames
- capturing problems for the game of Go
- Domineering.

Solving Games

- Other algorithms for solving games :
- Proof Number Search : Go-Moku
- PN^2 : Fanorona, 6x6 Lines of Action, 6x5 Breakthrough
- Depth-first Proof Number Search (Df-pn) : 9x9 Hex
- Retrograde Analysis : Awari
- Combination : Checkers
- Monte Carlo Tree Search Solver
- Product Propagation

Le jeu de Nim



L’année dernière à Marienbad

L'Année dernière à Marienbad est un film en noir et blanc de langue française réalisé par Alain Resnais et sorti en 1961. Il reçoit le Lion d'or à la Mostra de Venise la même année.

Le jeu de Nim

- Pour évaluer une position on effectue la somme binaire :

I	0 0 1
III	0 1 1
IIII	1 0 0
IIIIIII	1 1 1
	0 0 1

1 ^ 3 ^ 4 ^ 7 = 1

Le jeu de Nim

- Une position perdante pour celui qui va jouer a une somme binaire de 0.
- Pour choisir un coup on choisit le coup qui amène à une somme binaire de 0.
- Combinatorial Game Theory.

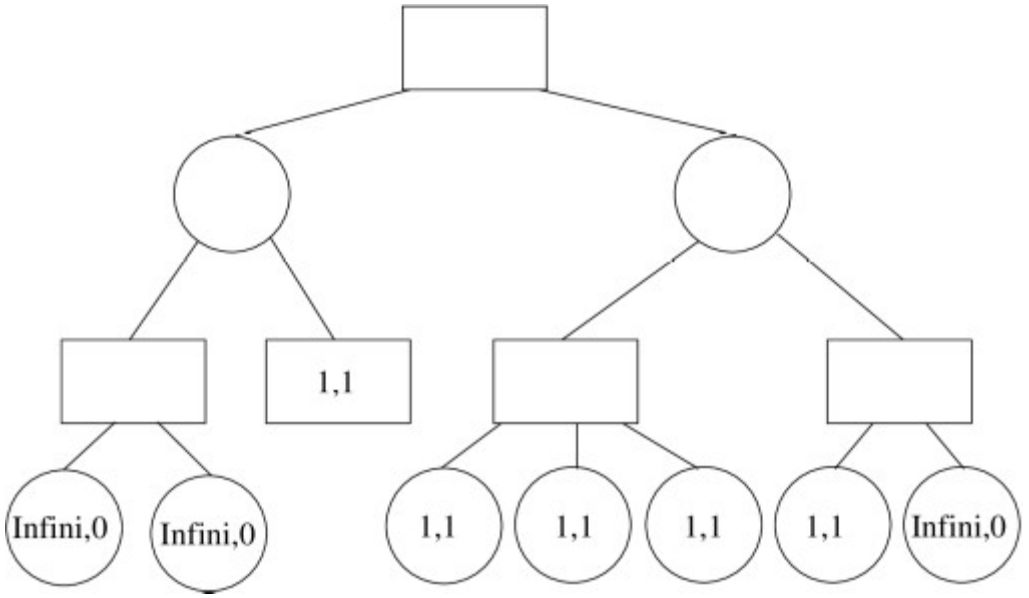
Proof Number Search

- Proof Number Search (PN-search) permet de prouver qu'un jeu ou qu'une position est gagnée pour un joueur.
- Le résultat de l'algorithme est binaire ; il renvoie 1 s'il réussit à prouver le gain et 0 sinon.
- PN-search marche particulièrement bien sur les arbres ET/OU quand le nombre de coups légaux varie beaucoup et qu'on peut élaguer de grandes parties de l'arbre.
- A chaque coup, PN-search cherche à calculer le coût de prouver la valeur de la racine. Pour cela il calcule le coût de prouver la valeur 1 à chaque noeud de l'arbre ainsi que le coût de le prouver la valeur 0.

Proof Number Search

- Chaque noeud comporte un Proof Number (PN) qui estime le coût de prouver 1 et un Disproof Number (DN) qui estime le coût de prouver 0.
- Une feuille est terminale si elle correspond à une position gagnée ou perdue.
- Une feuille non terminale a un PN de 1 et un DN de 1.
- Une feuille gagnée a un PN de 0 et un DN infini
- Une feuille perdue a un PN infini et un DN de 0.
- PN-search choisit de développer la branche qui a le moindre coût.

Proof Number Search



- Quelle est la feuille la plus intéressante à développer pour prouver la valeur 1 à la racine de l'arbre ?

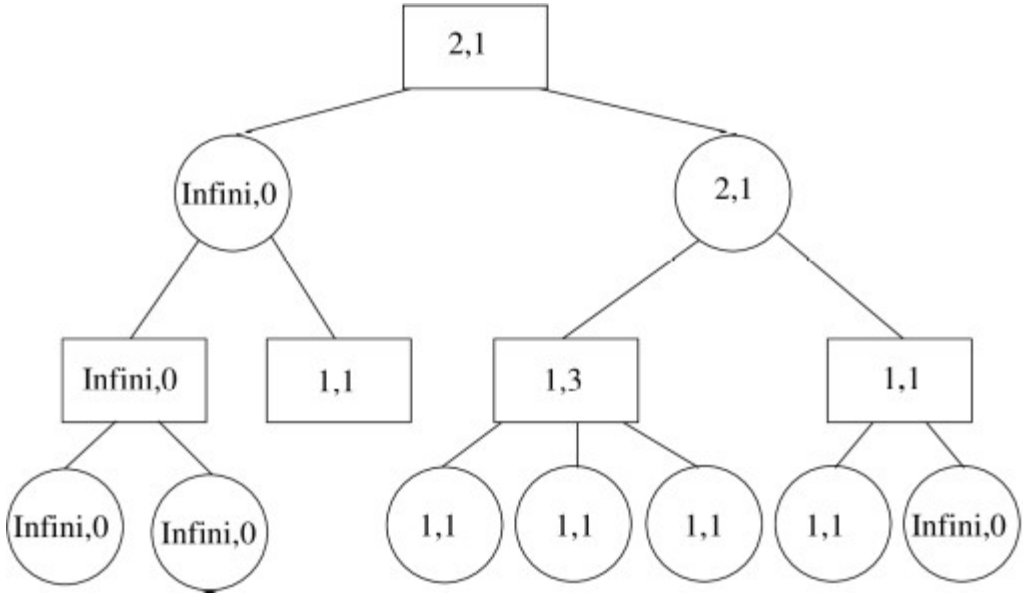
Proof Number Search

- Aux noeuds OU, pour prouver le noeud, il suffit qu'un seul des fils ait la valeur 1.
- Le nombre minimum de coups pour prouver le noeud OU est donc le minimum sur tous les fils du nombre de coups qu'il faut pour prouver chacun des fils.
- En revanche, pour prouver la valeur 0 à un noeud OU, il faut que tous les fils aient la valeur 0.
- Le nombre minimum de coups pour prouver la valeur 0 à un noeud OU (soit le disproof number associé au noeud OU) est donc la somme des disproof numbers des fils.

Proof Number Search

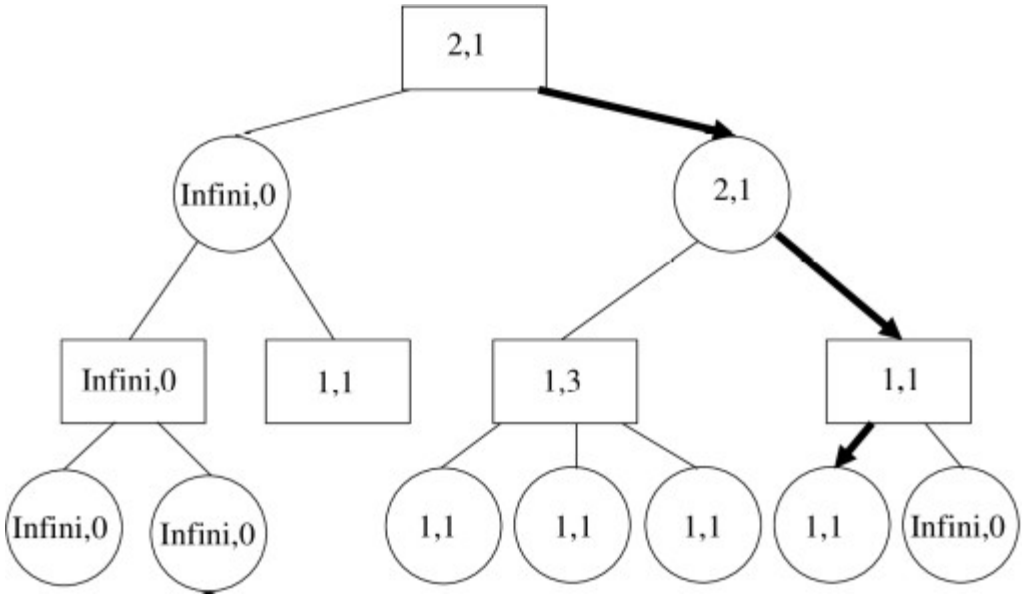
- De manière symétrique, aux noeuds ET, le proof number sera la somme des proof numbers des fils (il faut que tous les fils soient à 1 pour que le noeud ET soit à 1).
- Le disproof number sera le minimum des disproof numbers des fils (il suffit qu'un seul des fils soit à 0 pour que le noeud ET soit à 0).

Proof Number Search



- La remontée des valeurs de l'arbre.

Proof Number Search



- Choix de la feuille à développer.

Proof Number Search

Node type	PN	DN
Win	0	∞
Lose	∞	0
Frontier	1	1
Max	$\min_{c \in \text{chil}(n)} \text{PN}(c)$	$\sum_{c \in \text{chil}(n)} \text{DN}(c)$
Min	$\sum_{c \in \text{chil}(n)} \text{PN}(c)$	$\min_{c \in \text{chil}(n)} \text{DN}(c)$

- Multiple Outcome Proof Number Search [Saffidine & Cazenave 2012].

PN^2

- PN search sature rapidement la mémoire.
- Pour réduire la consommation mémoire au prix d'une résolution plus lente on utilise PN^2.
- Principe : faire une recherche PN secondaire à chaque feuille de la recherche PN principale.
- Le nombre de nœuds de la recherche secondaire est égal au nombre de nœuds de la recherche principale.
- On développe des arbres qui ont des tailles proportionnelles au carré de ce que l'on peut faire avec PN.

Product Propagation

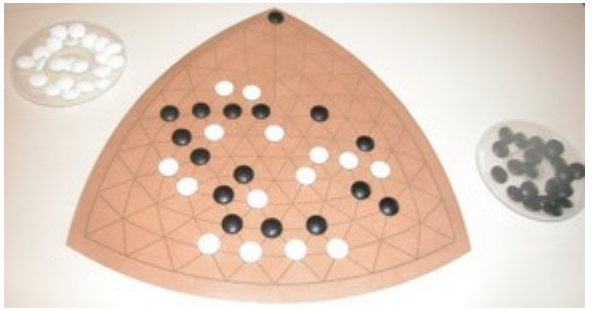
	Node label	PPN
`info-term`	*Max* wins	1
	Max loses	0
`init-leaf`		$\frac{1}{2}$
`update`	*Max*	$1 - \prod_C (1 - \text{PPN})$
	Min	$\prod_C \text{PPN}$

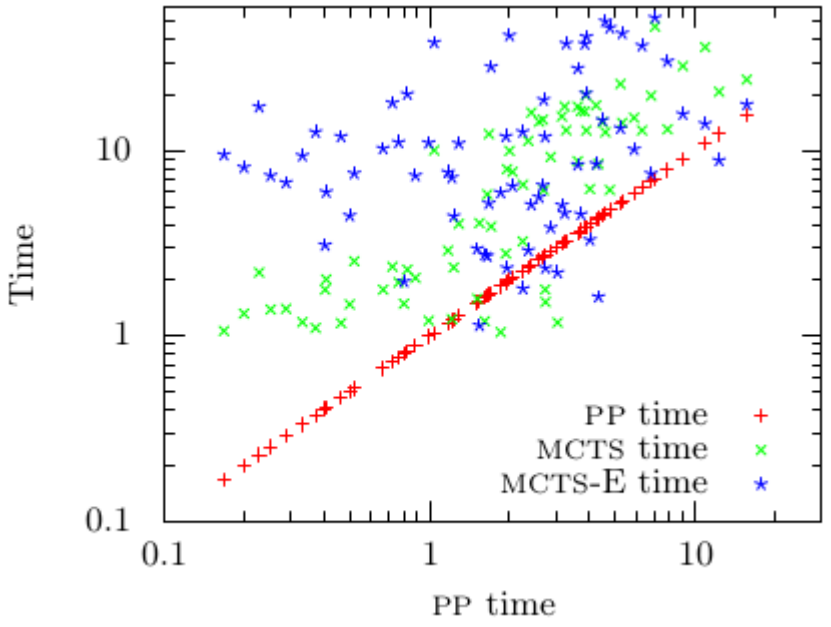
Node label	Chosen child
Max	$\arg\max_C \text{PPN}$
Min	$\arg\min_C \text{PPN}$

- Best first search.
- PP^2 [Saffidine & Cazenave 2013].

Product Propagation

- Y :

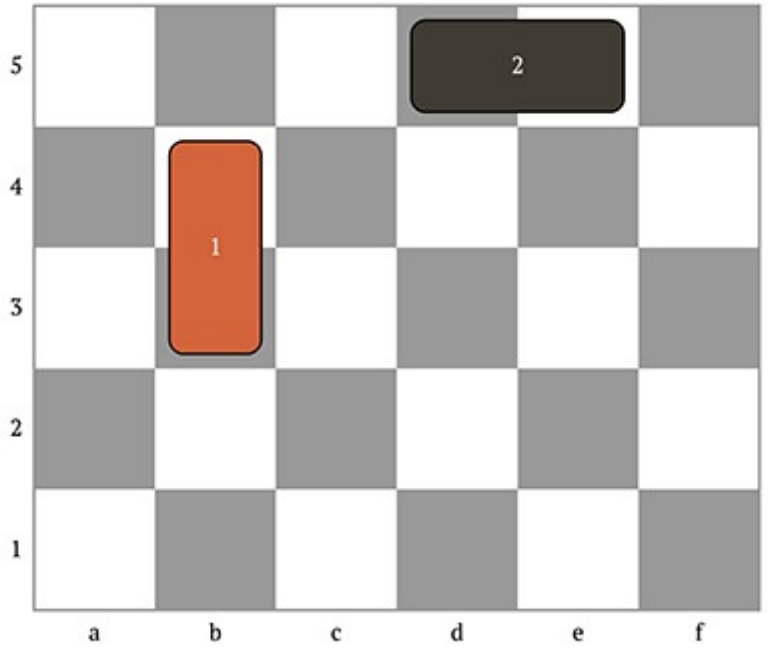




Product Propagation

- Domineering :

	5×6	6×6	7×6
$\alpha\beta$	701,559	38,907,049	6,387,283,988
PNT	1,002,277	$>10^7$	$>10^7$
PN^2	17,236	>154,107	>511,568
PP	101,244	5,525,608	$>10^7$
PPT	27,766	528,032	4,294,785
PP^2	3,634	24,190	145,757



Analyse rétrograde

- L'analyse rétrograde consiste à créer une table des résultats pour toutes les états possibles.
- On part des états terminaux et on remonte la valeur des états en déjouant les coups sur les états dont on connaît la valeur.
- On s'arrête lorsque tous les états ont été évalués.

Analyse rétrograde

- Aux Echecs, l'analyse rétrograde a permis de trouver la valeur exacte des finales à 6 pièces ou moins.
- Certaines finales que les grand maîtres croyaient nulles sont en fait gagnées (mais difficilement).
- Pour Awele, le jeu a été complétement résolu par analyse rétrograde sur un ordinateur parallèle.
- Pour les Checkers Chinook a résolu le jeu en partie grâce aux bases de finale.

Analyse rétrograde

- Deux joueurs A et B jouent chacun leur tour.
- A commence.
- Un joueur doit choisir un nombre entre 1 et 10.
- Le jeu se termine lorsque le total des nombres choisis par les deux joueurs atteint 100.
- Analyser le jeu si celui qui atteint 100 gagne.
- Analyser la version misere où celui qui atteint un total ≥ 100 perd.

Analyse rétrograde

- 90 à 99 : celui qui joue gagne
- 89 : celui qui joue perd
- 79 à 88 : celui qui joue gagne
- 78 : celui qui joue perd
- 68 à 77 : celui qui joue gagne
- 67 : celui qui joue perd
- 57 à 66 : celui qui joue gagne
- 56 : celui qui joue perd
- 46 à 55 : celui qui joue gagne
- 45 : celui qui joue perd

Analyse rétrograde

- 35 à 44 : celui qui joue gagne
- 34 : celui qui joue perd
- 24 à 33 : celui qui joue gagne
- 23 : celui qui joue perd
- 13 à 22 : celui qui joue gagne
- 12 : celui qui joue perd
- 2 à 11 : celui qui joue gagne
- 1 : celui qui joue perd
- 0 : celui qui joue gagne

Analyse rétrograde

- Version misère :
- 99 : celui qui joue perd
- 89 à 98 : celui qui joue gagne
- 88 : celui qui joue perd
- 78 à 87 : celui qui joue gagne
- 77 : celui qui joue perd
- 67 à 76 : celui qui joue gagne
- 66 : celui qui joue perd
- 56 à 65 : celui qui joue gagne
- 55 : celui qui joue perd

Analyse rétrograde

- 45 à 54 : celui qui joue gagne
- 44 : celui qui joue perd
- 34 à 43 : celui qui joue gagne
- 33 : celui qui joue perd
- 23 à 32 : celui qui joue gagne
- 22 : celui qui joue perd
- 12 à 21 : celui qui joue gagne
- 11 : celui qui joue perd
- 1 à 10 : celui qui joue gagne
- 0 : celui qui joue perd

Analyse rétrograde

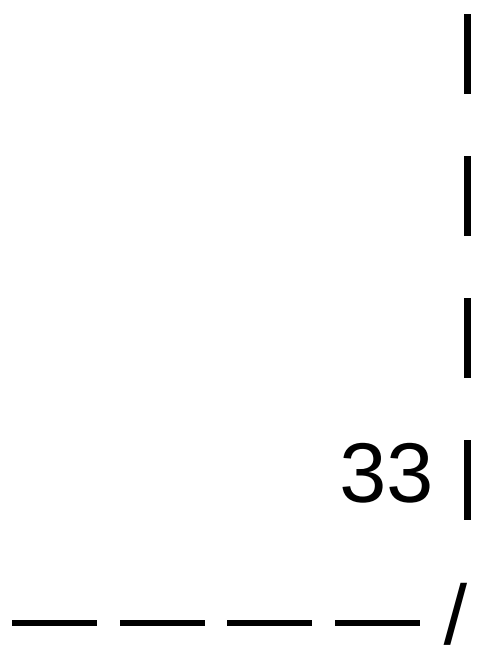
- Pour stocker un échiquier dans une finale sans pions on utilise les symétries pour réduire la taille de stockage.
- Combien y a-t-il de façons de placer les deux rois si on élimine toutes les symétries ?

Analyse rétrograde

- Pour un état donné, il y a huit états symétriques qui sont équivalents.
- On peut toujours amener le premier roi dans le huitième d'échiquier en bas à droite.
- Il n'y a donc que 10 positions possibles pour le premier roi.

Analyse rétrograde

- On compte le nombre de cases possibles du deuxième roi pour chaque case du premier roi :



Analyse rétrograde

- On compte le nombre de cases possibles du deuxième roi pour chaque case du premier roi :

  30 |

  55 30 |

  55 55 30 |

  58 58 58 33 |

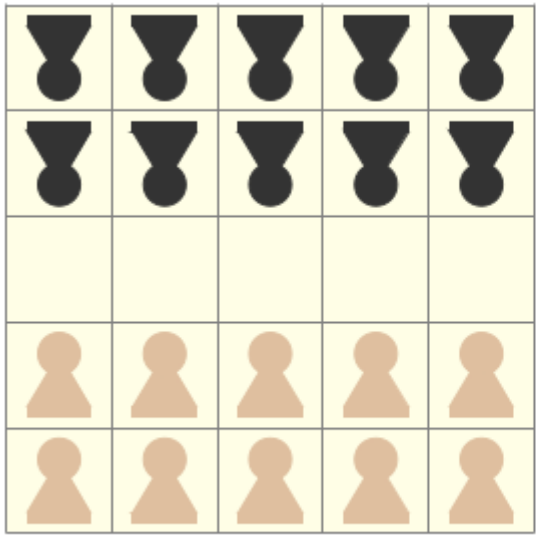
  — — — — /

- => 462 façons de placer les deux rois.

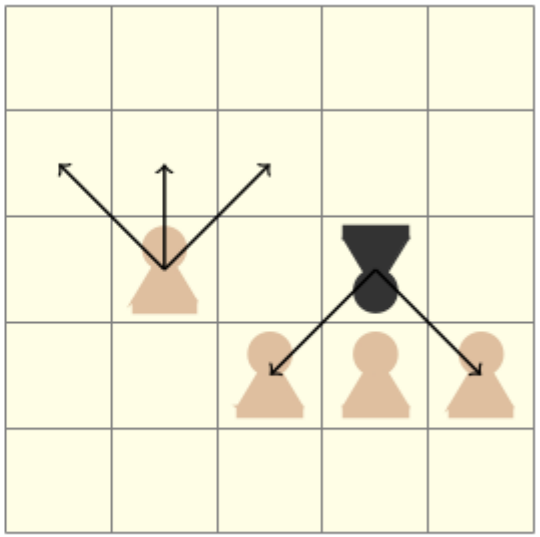
Awari

Solved in 2003 by Romein & Bal with retrograde analysis.

Breakthrough



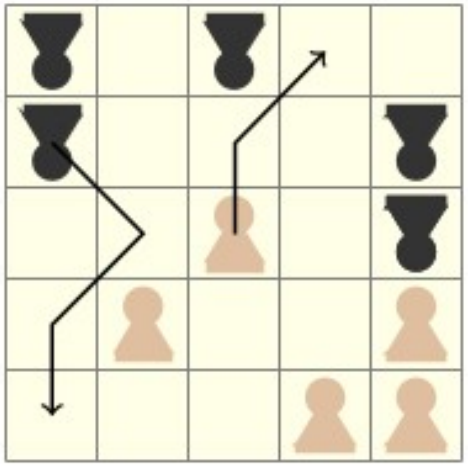
(a) Starting position on size 5×5.



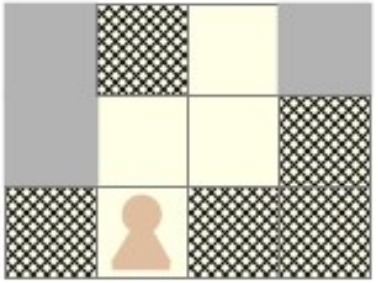
(b) Possible movements.

Breakthrough

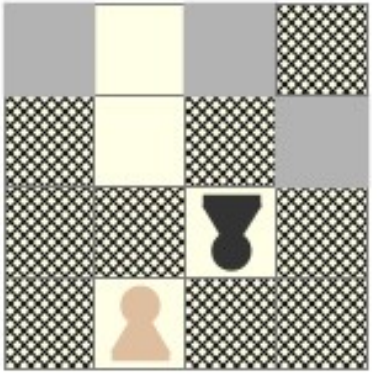
- Race patterns :



(a) Sample game with Black to play. White player can force a win.



(b) Two-move second player win pattern.



(c) Three-move first player win pattern.

[Saffidine & al. 2011]

Breakthrough

- PPN^2 with 64 clients :

Board size	Search size	Patterns	Time	Expanded
5×4	1k	No	2s	4132
5×4	1k	Yes	1s	72
4×5	1k	No	161s	241k
4×5	1k	Yes	27s	4k
5×5	1k	Yes	927s	78k
5×5	100k	No	29,170s	208k
5×5	100k	Yes	2959s	3k
6×5	10k	Yes	25,638s	14k
6×5	100k	Yes	47,134s	21k

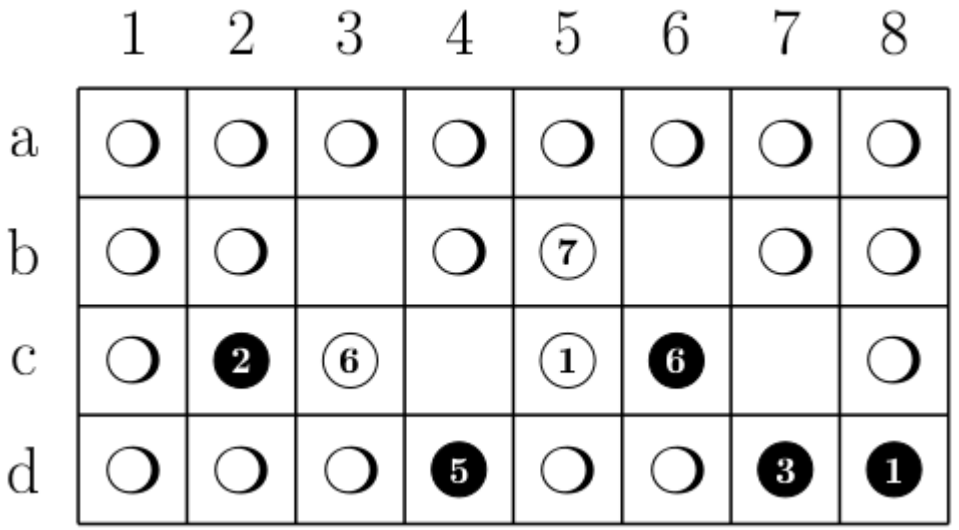
Material Symmetry

- Chinese Dark Chess :

Names	Representation	Capture rule	Note
King (K)	帥 將, 7 7	all opponent piece except pawns	
Guard (G)	仕 士, 6 6	all inferior opponent pieces	
Bishop (B)	相 象, 5 5	all inferior opponent pieces	
Knight (N)	馬 馬, 4 4	all inferior opponent pieces	
Rook (R)	車 車, 3 3	all inferior opponent pieces	
Cannon (C)	炮 炮, 2 2	all opponent pieces	jump capture
Pawn (P)	兵 卒, 1 1	opponent king and pawns	

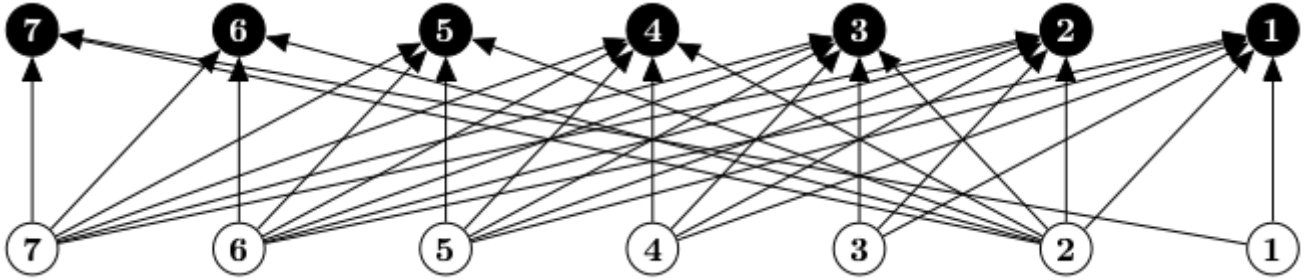
Material Symmetry

- Chinese Dark Chess :



Material Symmetry

- Chinese Dark Chess :

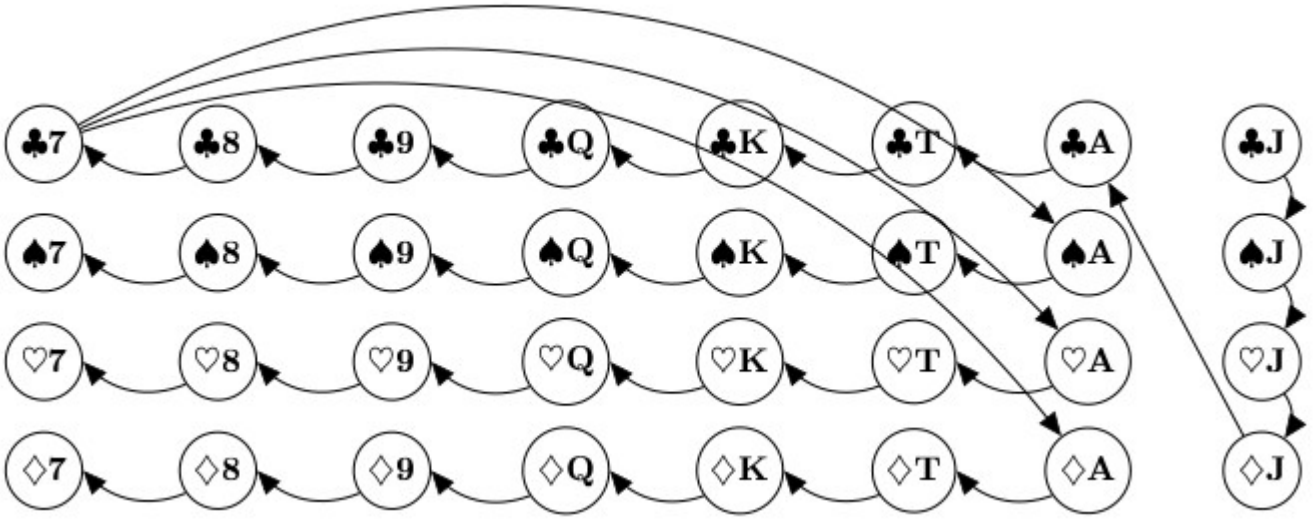


Capture relationships for White

[Saffidine & al. 2013]

Material Symmetry

- Skat :

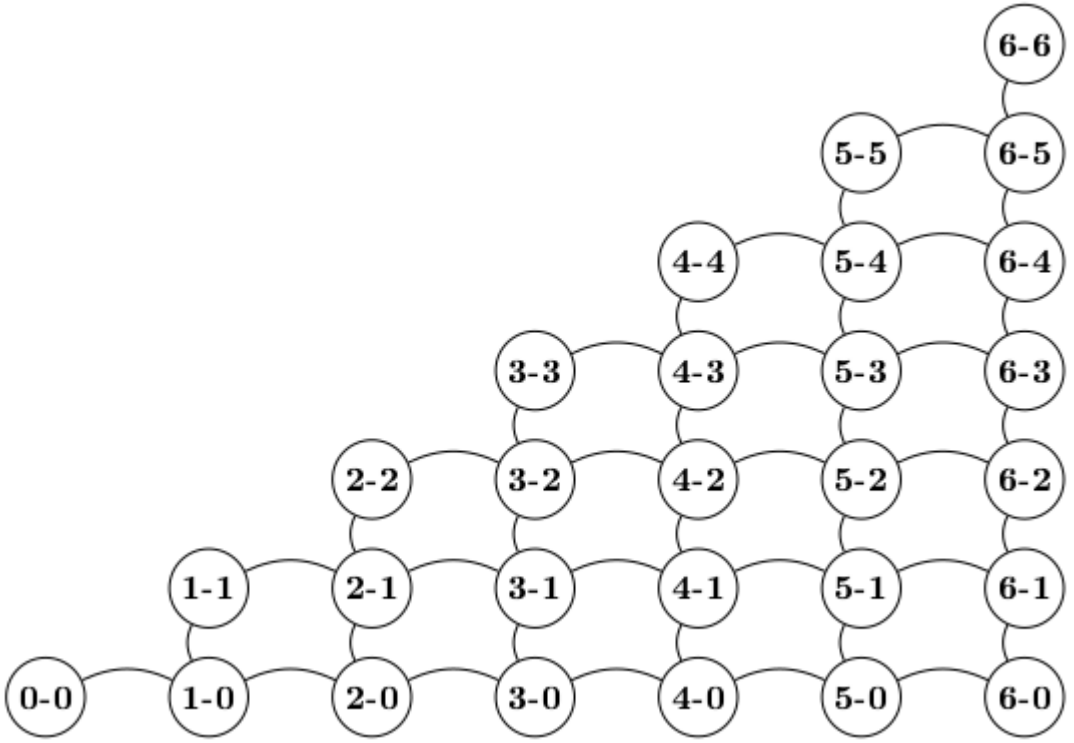


Capture relationships

[Saffidine & al. 2013]

Material Symmetry

- Dominoes:



Playability relationships

[Saffidine & al. 2013]

Material Symmetry

- Reduction Factor :

Game	Elements	Total		Representatives		Reduction
		Positions	Memory	Positions	Memory	factor
CHINESE DARK CHESS	2	4.961×10^{4}	2.97 KB	7.936×10^{3}	496 B	6.13
	3	9.999×10^{6}	610.3 KB	1.131×10^{6}	69.02 KB	8.84
	4	1.140×10^{9}	67.92 MB	1.142×10^{8}	6.81 MB	9.92
	5	9.036×10^{10}	5.26 GB	1.013×10^{10}	603.8 MB	8.92
	6	5.440×10^{12}	316.7 GB	8.002×10^{11}	46.58 GB	6.80
	7	2.601×10^{14}	14.78 TB	5.247×10^{13}	2.98 TB	4.96
	8	1.014×10^{16}	576.1 TB	2.756×10^{15}	156.7 TB	3.68
DOMINOES	3	21,168	2.58 KB	92	11.50 B	230.09
	4	550,368	67.18 KB	996	124.50 B	552.58
	5	8,026,200	979.76 KB	7,854	981.75 B	1021.93
	6	82,556,550	9.84 MB	53,790	6.57 KB	1541.99

[Saffidine & al. 2013]

Recherche avec menaces

- Les algorithmes de recherche avec menaces permettent de sélectionner un petit nombre de coups intéressants.
- Ils permettent de faire des recherches étroites et profondes qui s'approchent des recherches faites par les joueurs expérimentés.
- Ils fonctionnent lorsque jouer quelques coups de suite du même joueur amène souvent à la victoire.

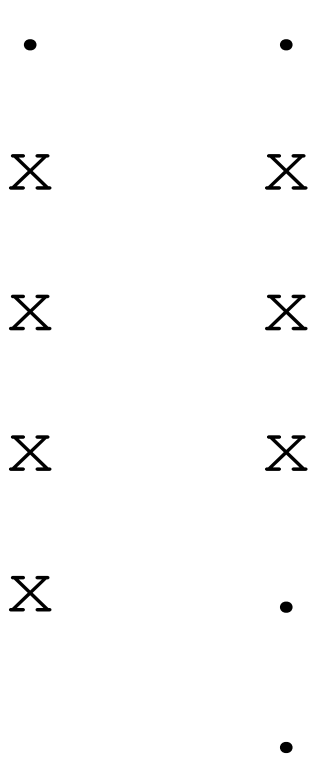
Recherche avec menaces

- On définit gagnant par induction :
- $gagnant_0(P,J)$: J peut gagner si c'est à lui de jouer.
- $gagne_k(P,J)$: on peut vérifier $gagnant_{k-1}(P,J)$ et tous les coups de l'adversaire amènent à une position $gagnant_{k-1}(P,J)$
- $gagnant_k(P,J)$ est vrai s'il existe un coup gagnant ou si un coup amène à une position $gagne_k$.

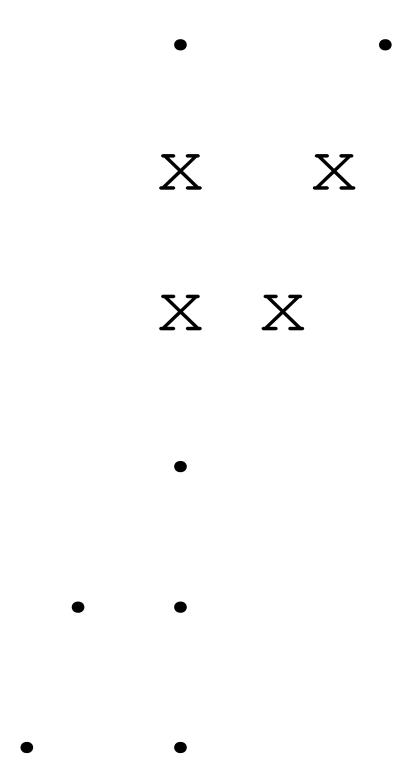
Recherche avec menaces

- Trouver des positions gagnant$_0$ à gagnant$_3$ au Go-Moku (but = aligner 5 pierres).

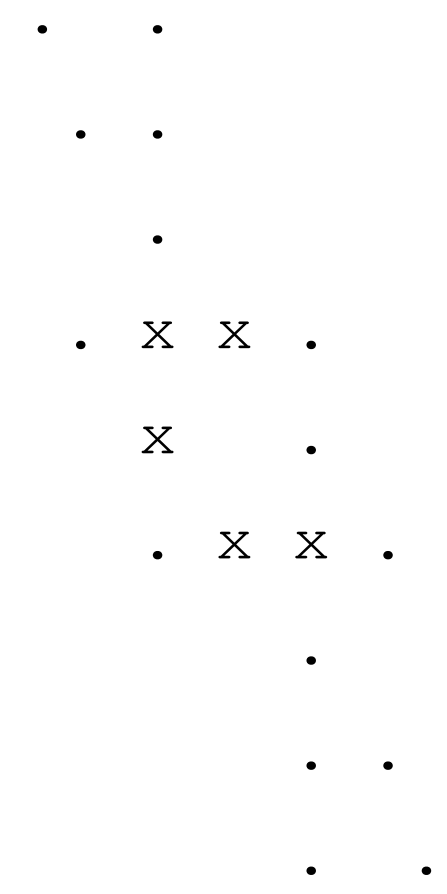
Recherche avec menaces



Recherche avec menaces



Recherche avec menaces



Recherche avec menaces

- La recherche λ consiste à tester des menaces sans limite de profondeur sur les arbres de menaces développés.

Definition 1. A λ^n-tree is a search tree (with the aim to achieve a single well-defined goal) which consists solely of λ^n-moves (defined in definition 2); a λ_a^n-tree is a λ^n-tree where the attacker moves first.

Definition 2. A λ^n-move is a move with the following characteristics. If the *attacker* is to move, it is a move that implies – if the defender passes – that there exists at least one subsequent λ_a^i-tree with value 1, $0 \leq i \leq n-1$. If the *defender* is to move, it is a move that implies that there does not exist any subsequent λ_a^i-tree with value 1, $0 \leq i \leq n-1$.

Recherche avec menaces

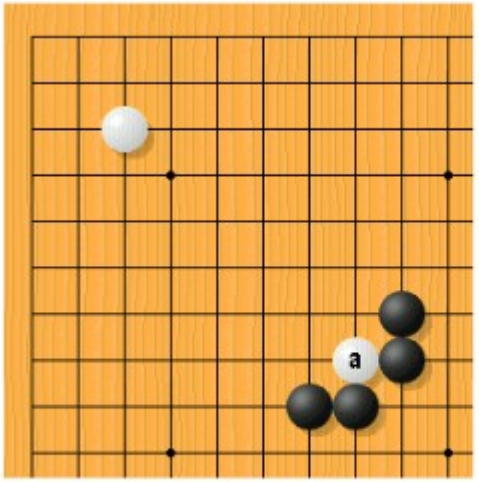


Figure 1: How to capture a?

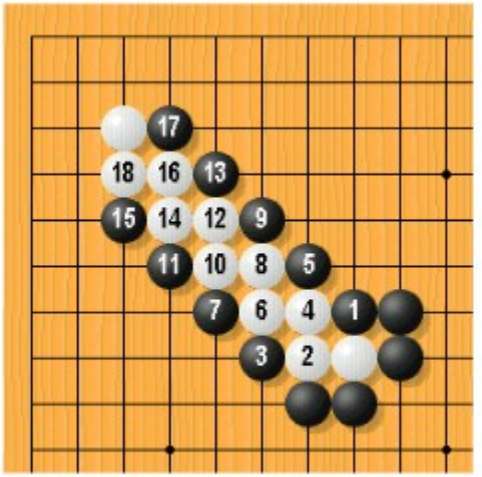


Figure 2: A failing ladder (λ^1=0).

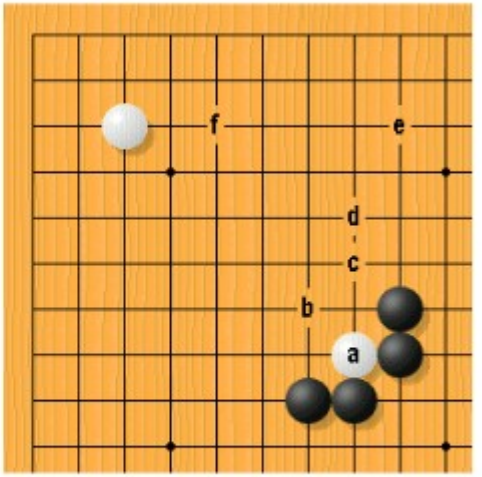


Figure 3: Black λ^2-moves?

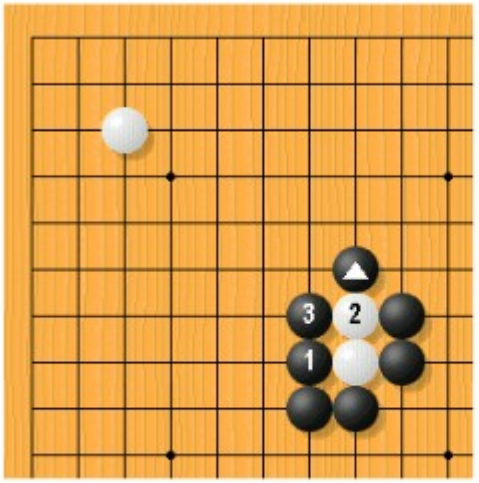


Figure 4: Working ladder (λ^1=1) after black *c*.

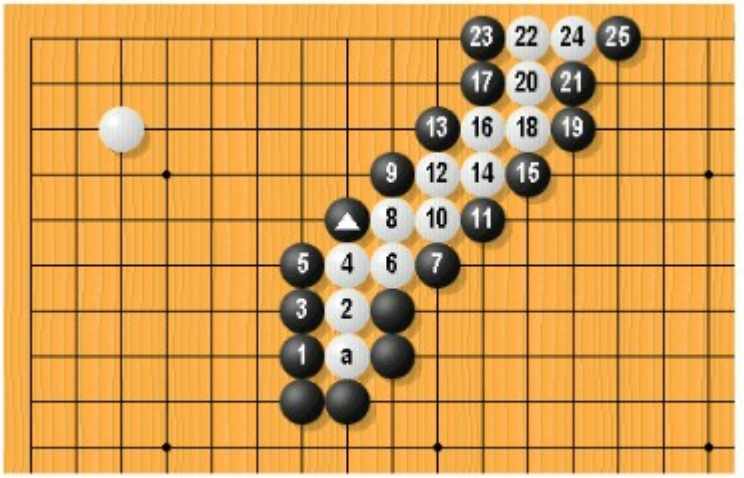


Figure 5: Working ladder (λ^1=1) after black *d*.

Recherche avec menaces

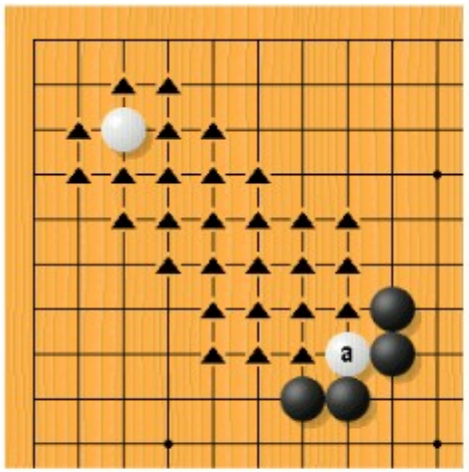


Figure 6: Black λ^2-moves to kill *a*.

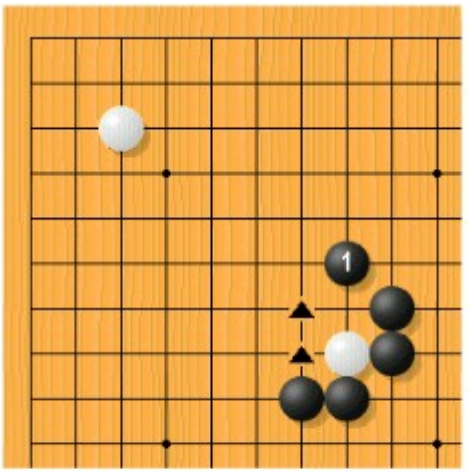


Figure 7: White λ^2-moves after black 1.

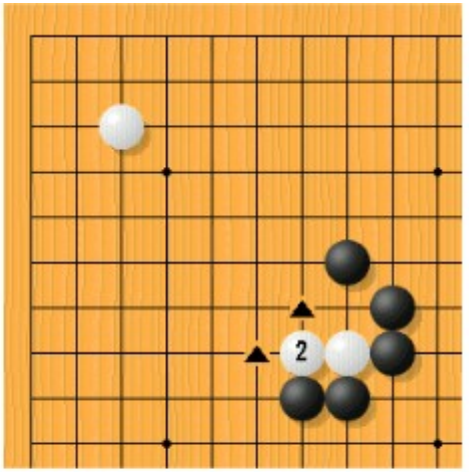


Figure 8: Black λ^2-moves after white 2.

Recherche avec menaces

- La recherche λ est améliorée si on réduit les coups possibles pour max aux seuls coups menaçant un gain en n-1 coups.
- De même les seuls coups à envisager pour invalider une menace sont ceux de la zone pertinente de la menace.
- La zone pertinente est l'ensemble des cases modifiées et testées par une recherche λ.

Recherche avec menaces

- Les heuristiques de l'Alpha-Beta comme les coups qui tuent fonctionnent aussi pour la recherche λ.
- Une autre heuristique efficace est l'élargissement itératif.
- A chaque nœud on fait une recherche d'ordre 1, puis 2, puis 3, etc.

Recherche avec menaces

- Jeux concernés :
- Atarigo / captures au jeu de Go
- Phutball
- Connect6
- Gomoku

Monte Carlo Tree Search

- Computer Go
  - AlphaGo combines MCTS and Deep Learning
- General Game Playing
  - Winners of the annual competitions since 2007
- Chess and Shogi
  - AlphaZero
- Computer Hex
  - Winners since 2008

Monte Carlo Tree Search

- Upper Confidence bounds applied to Trees
- Play random games
- Use the mean of previous playouts of the current state to choose the move to play
- Use the cumulative regret to choose the move
- $\text{argmax}_m(\text{mean}_m + c * \text{sqrt}(\log(p) / p_m))$

Monte Carlo Tree Search Solver

- Completely explored subtrees [Winands & al. 2008]
- Score Bounded MCTS when more than two outcomes for a game [Cazenave & Saffidine 2010].

Monte Carlo Tree Search Solver

If n is an internal max-node then

$\text{pess}(n) := \max_{s \in \text{children}(n)} \text{pess}(s)$

$\text{opti}(n) := \max_{s \in \text{children}(n)} \text{opti}(s)$

If n is an internal min-node then

$\text{pess}(n) := \min_{s \in \text{children}(n)} \text{pess}(s)$

$\text{opti}(n) := \min_{s \in \text{children}(n)} \text{opti}(s)$

Monte Carlo Tree Search Solver

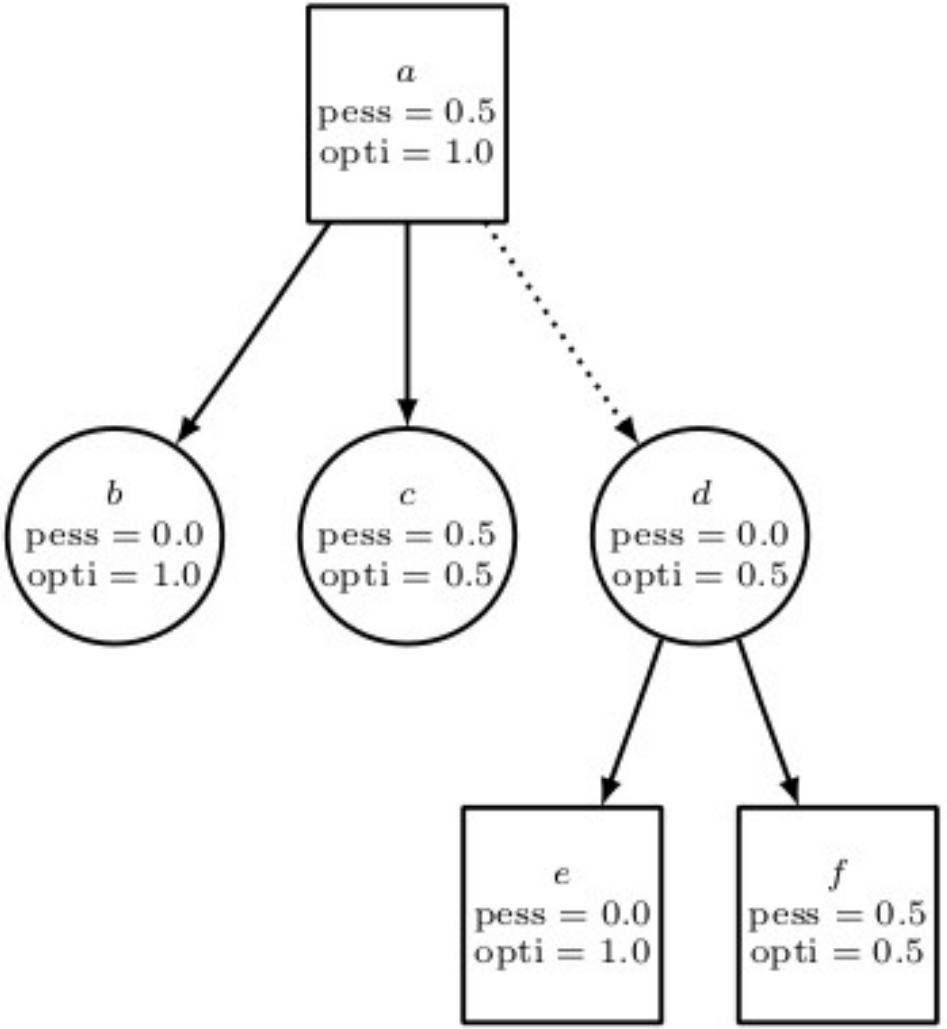


Fig. 1. Example of a cut. The d node is cut because its optimistic value is smaller or equal to the pessimistic value of its father.

Monte Carlo Tree Search Solver

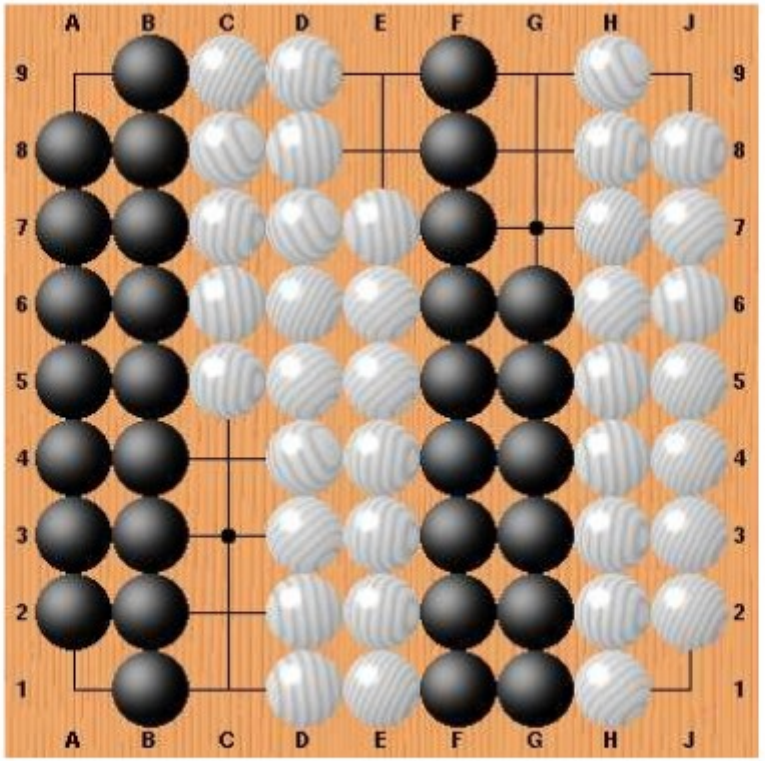


Fig. 4. A test seki with two shared liberties, three liberties for the *Max* player (Black) and four liberties for the *Min* player (White).

Monte Carlo Tree Search Solver

Min liberties	*Max* liberties					
	1	2	3	4	5	6
1	359	479	1535	2059	10 566	25 670
2	1389	11 047	12 627	68 718	98 155	28 9324
3	7219	60 755	541 065	283 782	516 514	79 1945
4	41 385	422 975	>1 000 000	>1 000 000	>989 407	>999 395
5	275 670	>1 000 000	>1 000 000	>1 000 000	>1 000 000	>1 000 000
6	>1 000 000	>1 000 000	>1 000 000	>1 000 000	>1 000 000	>1 000 000

Table 3. Number of playouts for solving Sekis with two shared liberties

Min liberties	*Max* liberties					
	1	2	3	4	5	6
1	137	259	391	1135	2808	7164
2	501	1098	1525	3284	13 034	29 182
3	1026	5118	9208	19 523	31 584	141 440
4	2269	10 094	58 397	102 314	224 109	412 043
5	6907	27 947	127 588	737 774	>999 587	>1 000 000
6	16 461	85 542	372 366	>1 000 000	>1 000 000	>1 000 000

Table 5. Number of playouts for solving Sekis with two shared liberties, bounds on score, node pruning, biasing with $\gamma = 0$ and $\delta = 10000$

Conclusion

- Proof Number Search
- Product Propagation
- Retrograde Analysis
- Symmetries
- Threat Search
- Monte Carlo Tree Search

Move Ordering

- For iterative deepening Alpha-Beta there are general heuristics to order moves :
  - Killer moves
  - History heuristic

- Depth first Alpha-Beta relies more on domain dependent move ordering heuristics.
- Question : Is it better to use Depth-first Alpha-Beta or Iterative deepening Alpha-Beta to solve games ?

Monte Carlo Move Ordering

- Play random games with MCTS
- Solve the game with depth-first Alpha-Beta
- Use the mean of the moves from MCTS to order the moves in the Alpha-Beta search

Nogo

- Played on a Go board
- The first player to capture has lost
- Played at the 2011 combinatorial game theory workshop in Banff
- The best programs use MCTS

Move Ordering

- Experiments on Nogo
- Compare :
  - depth first Alpha-Beta and Monte Carlo move ordering
  - Iterative deepening Alpha-Beta and the history heuristic
- In both algorithms two killer moves are tried first

Nogo

- For small Nogo boards that have less than 15 intersections we did not use move ordering
- For boards with more than 15 intersections we used 100,000 playouts
- For boards with more than 18 intersections we used 1,000,000 playouts before the Depth-first Alpha-Beta search.

Nogo

	1	2	3	4	5	6	7	8	9	10
1	2	1	1	2	1	1	1	1	1	1
2	1	1	2	2	1	1	1	1	2	2
3	1	2	1	2	1	1	1	1		
4	2	2	2	2	1	1				
5	1	1	1	1						
6	1	1	1	1						
7	1	1	1							
8	1	1	1							
9	1	2								
10	1	2								

Winner for various size of Nogo

Nogo

	1	2	3	4
1	1			
2	2	8		
3	4	59	312	
4	11	355	19,170	7,255,990
5	10	1,354	131,424	196,783,634
6	22	11,375	2,091,260	
7	82	32,990	444,202,190	
8	142	160,082	118,264,438,878	
9	185	53,057,906		
10	554	895,014,128		

Number of nodes for solving Nogo boards of various sizes with Depth-first Alpha-Beta

Nogo

	1	2	3	4
1	1.00			
2	1.00	3.00		
3	0.75	3.05	5.85	
4	1.36	2.63	3.30	1.38
5	2.80	3.05	**9.32**	1.58
6	1.59	2.29	4.54	
7	4.33	5.65	4.04	
8	2.05	5.46	**3.02**	
9	4.58	**0.72**		
10	5.51	1.34		

Speedups of Depth-first Alpha-Beta versus Iterative Deepening Alpha-Beta for solving Nogo boards of various sizes

Atarigo

- Played on a Go board
- The first player to capture has won
- 6x6 Atarigo has been solved in 2006 with Alpha-Beta and threats.

Atarigo

- We used a move ordering heuristic that focuses on the opponent string that has the less liberties.
- The liberties of this opponent string are tried first, then the other possible moves are tried.
- We also use a sort on the moves on the liberties of the opponent string: the moves on the liberties that increase the most the number of liberties for the opponent are tried first

Atarigo

	1	2	3	4	5	6	7	8	9	10
1	2	2	1	1	1	1	1	1	1	1
2	2	2	2	2	1	2	1	1	1	1
3	1	2	1	1	1	1	1	1		
4	1	2	1	2	1	1				
5	1	1	1	1	1					
6	1	2	1	1						
7	1	1	1							
8	1	1	1							
9	1	1								
10	1	1								

Winner for Atarigo boards of various sizes

Atarigo

	1	2	3	4	5
1	1				
2	3	25			
3	3	61	129		
4	5	277	1,413	66,212	
5	12	855	25,744	1,202,069	880,310,201
6	20	7,339	329,201	295,000,829	
7	49	18,415	4,114,618		
8	80	75,390	371,538,460		
9	178	397,805			
10	343	3,949,515			

Number of nodes for solving Atarigo boards of various sizes with Depth-first Alpha-Beta

Atarigo

	1	2	3	4	5
1	1.00				
2	1.00	1.56			
3	0.67	2.62	7.09		
4	1.80	2.94	10.28	4.80	
5	1.67	1.96	2.38	8.16	**1.28**
6	4.00	2.90	3.35	**6.42**	
7	2.75	4.05	0.98		
8	5.42	4.67	**1.83**		
9	4.53	5.77			
10	6.45	4.65			

Speedups of Depth-first Alpha-Beta versus Iterative Deepening Alpha-Beta for solving Atarigo

Conclusion

- Depth first Alpha-Beta with Monte Carlo move ordering outperforms Iterative Deepening Alpha-Beta at solving Nogo
- Depth first Alpha-Beta with a simple move ordering heuristic outperforms Iterative Deepening Alpha-Beta at solving Atarigo.

Thank you