

Programmation de jeux de stratégie

Ivo Blöchliger & Ulrich Ultes-Nitsche

Département d'informatique
Université de Fribourg

13 septembre 2013

Matin

- Théorie
- Pause café
- Théorie

Après-midi

Exercices pratiques
Approfondissement

SSIE

Société Suisse de l'Informatique dans l'Enseignement

<http://www.svia-ssie-ssii.ch/>

Département d'informatique de l'université de Fribourg

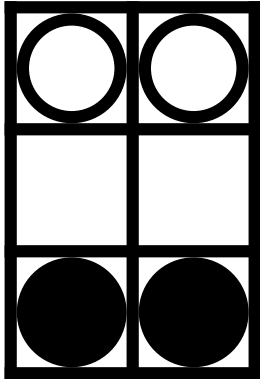
<http://diuf.unifr.ch/>

- 1 Jeux de stratégie
- 2 Jeux résoluble sans arbre
- 3 Jeux plus complexes
- 4 Puissance 4
- 5 Jeux de vérification
- 6 Publicité

Échecs miniature

Règles

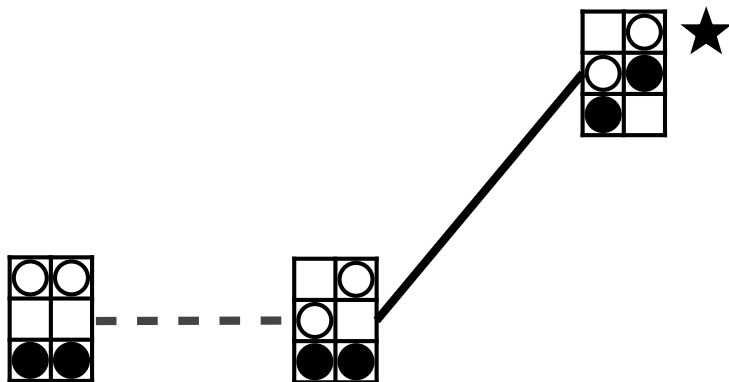
- Plateau de taille réduit
- Uniquement des pions
- Avancer d'une case ou prise diagonale
- Victoire si arrivé au côté opposé
- Defaite si pas de coup possible



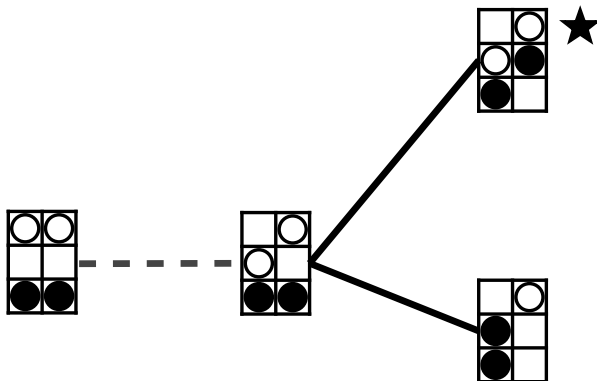
Échecs miniature



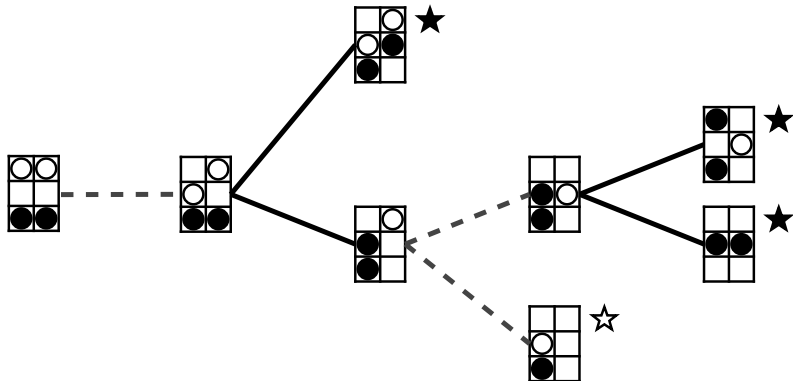
Échecs miniature



Échecs miniature



Échecs miniature



Setting

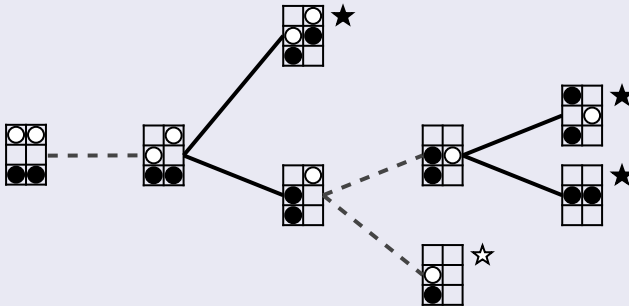
- Deux adversaires
- Pas de hasard
- Décident alternativement d'un coup
- A chaque tour : Nombre fini de coups possibles
- Toute information est disponible (état du jeux)
- Le jeux se termine par une victoire ou un remis

Quelques jeux selon cette définition

- Échecs
- Puissance quatre
- Dames
- Jeux du moulin (charret)
- Go
- Quoridor

Arbre de décision

- Racine : État initial du jeu
- Successeurs : États suivants possibles
- Feuilles : États de jeux terminé

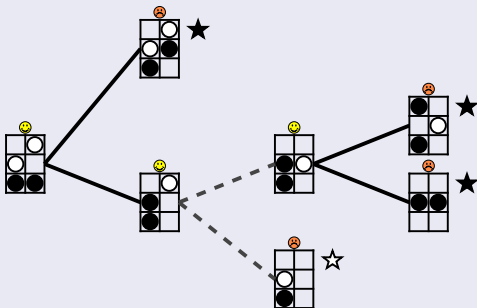


Arbre infini

- Jeux cyclique
- Échecs : 3 fois “le même état de jeux”
États passés font parti de l'état

Analyse d'un état de jeu

- Un état est gagnant pour le joueur courant, s'il existe un état perdant pour l'adversaire parmi les états enfants.
- Un coup gagnant amène à un état perdant pour l'adversaire.



Conclusion

- Arbre fini, jeux connu (revenant depuis les feuilles)
- Connu, si le premier joueur gagne ou perd, si les deux jouent parfaitement.
- Problème : Taille de l'arbre, temps de calcul (la mémoire n'est pas un problème)

Cette fonction retourne $+1$ si l'état est gagnant pour le joueur p

Algorithmme

```
function evaluateState(state  $s$ , player  $p \in \{1, 2\}$ )  
  if Game over then  
    return  $-1$  (or  $1$ ) if player  $p$  has lost (or won)  
  end if  
  for all moves  $m$  for player  $p$  at state  $s$  do  
     $s' \leftarrow m$  applied to  $s$   
    if evaluateState( $s'$ ,  $3 - p$ ) =  $-1$  then  
      return  $1$  ▷ Optionally return winning move  
    end if  
  end for  
  return  $-1$   
end function
```

Problème résolu

- Pour un arbre fini, l'algorithme produit une stratégie optimale.
- Ne parcourt pas forcément tout l'arbre (on s'arrête dès qu'un coup gagnant est trouvé)
- L'arbre peut être immense
- $O(m^d)$ feuilles, m : nombre de coups par état, d : nombre de demi-tours par jeux

Échecs miniature, 3×3

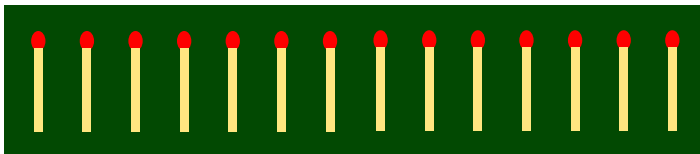
- $m \approx 3$, $c \approx 6$
- $3^6 = 729$ (effectivement 252 états)
- Avec cutoff : 49 états

Arbres pour 3×2 , 4×2 et 3×3

- Arbres complets
- Arbres coupés dès que l'on trouve un coup menant à une situation perdante
- Voir [http ://bit.ly/1a6jyHh](http://bit.ly/1a6jyHh)

Move ordering

- Si par chance on teste toujours le coup gagnant en premier, c'est comme si la profondeur de l'arbre est divisé par deux.
- S'il n'y a pas de coup gagnant, il faut tester tous les coups.
- A la place de n états, on évalue \sqrt{n} états.
- Heuristique pour deviner les coups prometteurs.



Jeux d'alumettes

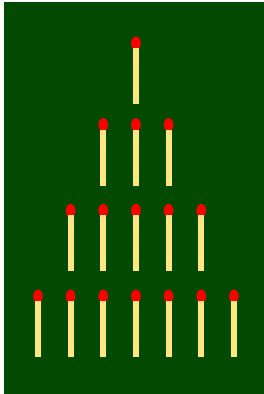
- Donné n alumettes.
- Alternativement les joueurs peuvent ôter 1, 2 ou 3 alumettes.
- Celui qui peut prendre la dernière gagne.

Jeux d'alumettes

- S'il reste 0 alumettes, on a perdu.
- S'il reste 1, 2 ou 3 alumettes, on gagne.
- Donc s'il reste 4 alumettes, on perd.
- On gagne donc s'il reste 5, 6 ou 7 alumettes.
- On perd s'il en reste 8
- ...
- On gagne, si on laisse $4m$ alumettes pour un $m \in \mathbb{N}$

Invariante

- Taille de l'arbre $\approx 3^{\frac{n}{2}}$.
- Calcul directe si la situation est gagnante pour n arbitraire.



Jeux de Nim

- Prendre d'une seule rangée un nombre arbitraire d'allumettes.
- Celui qui prend la dernière allumette gagne.
- <http://bit.ly/1dGqxec>

Histoire du jeux de Nim

- Connu depuis le 16^{ième} siècle.
- Analysé et résolu en 1901 par Charles L. Bouton
- Implementé en 1939, Nimatron à New York.

Analyse

- Noeds dans l'arbre $\approx m^d \approx 8^8 \approx 2.4 \cdot 10^7$
- Effectivement 10'292'376 noeuds
- Mais seulement $2 \cdot 4 \cdot 6 \cdot 8 = 384$ situations possibles

Approche par programmation dynamique

- $f(s)$: vrai/faux, si l'état s est gagnant/perdant
- $N(s)$: Ensemble de situations atteignables depuis s
- $f(\emptyset) = \text{faux}$
- $$f(s) = \bigvee_{s' \in N(s)} \neg f(s')$$

Implementation

- Enregistrer tout résultat intermédiaire
- Considérer une configuration comme un nombre en base mixte
- Programme en Ruby

Output

Number of nodes: 10292376

Number of configs: 384

Loosing positions:

```
[0, 0, 0, 0] [0, 0, 1, 1] [0, 0, 2, 2] [0, 0, 3, 3]
[0, 0, 4, 4] [0, 0, 5, 5] [0, 1, 2, 3] [0, 1, 4, 5]
[0, 2, 4, 6] [0, 2, 5, 7] [0, 3, 4, 7] [0, 3, 5, 6]
[1, 1, 1, 1] [1, 1, 2, 2] [1, 1, 3, 3] [1, 1, 4, 4]
[1, 1, 5, 5] [1, 3, 5, 7]
```

Stratégie

- Représenter le nombre d'allumettes par ligne en binaire n_r
- Situation perdante si
$$g(s) = n_1 \text{ XOR } n_2 \text{ XOR } \dots \text{ XOR } n_r = 0$$

Esquisse de preuve

- Pas d'allumettes : Situation perdant, $g(\emptyset) = 0$, ok.
- $g(s) = 0 \Rightarrow g(s') \neq 0 \quad \forall s' \in N(s)$
- $g(s) \neq 0 \Rightarrow \exists s' \in N(s)$ avec $g(s') = 0$

Propriétés de l'opération XOR

- Commutative
- $a \text{ XOR } b = 0 \iff a = b$
- $0 \text{ XOR } a = a \quad \forall a$

$$g(s) = 0 \Rightarrow g(s') \neq 0 \quad \forall s' \in N(s)$$

- Soit $s' \in N(s)$ et r la ligne changée.
- Donc $g(s') = g(s) \text{ XOR } n_r \text{ XOR } n'_r = n_r \text{ XOR } n'_r \neq 0$ comme $n_r \neq n'_r$.

Preuve pour la stratégie pour Nim

$$g(s) \neq 0 \Rightarrow \exists s' \in N(s) \text{ avec } g(s') = 0$$

- Soit $g_r = g(s) \text{ XOR } n_r$ (Somme sans la ligne r).
- Il existe toujours $g_r < n_r$, alors réduire la ligne r à g_r .

Preuve par l'absurde, supposons $g(s) \text{ XOR } n_r > n_r \forall r$

- Soit x la position du bit le plus significatif de $g(s) \neq 0$.
- $g(s) \text{ XOR } n_r$ change le bit x de n_r (et d'autres moins significatifs).
- $g(s) \text{ XOR } n_r > n_r$ implique que le bit x de n_r est 0, pour tout r .
- Si pour tout r le bit x de n_r est 0 alors le bit x de $g(s)$ est 0.
- Contradiction.

Algorithme

Algorithme pour calculer un coup pour le jeux de Nim

Compute $g(s) = n_1 \text{ XOR } n_2 \text{ XOR } \dots \text{ XOR } n_r$

if $g(s) = 0$ **then**

 Make random move

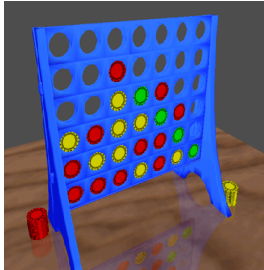
else

 Pick r s.t. $g_r = g(s) \text{ XOR } n_r < n_r$ and reduce line r to g_r .

end if

Conclusion

- Peu d'états, alors sauvegarder les résultats intermédiaires.
- Dans des rares cas, un calcul direct est possible.
- Utiliser la page nim avec `?cheat=true`



Puissance 4 comme exemple

- Plateau de jeux de 7×6 cases
- Les pièces tombent de haut en bas
- Celui qui aligne 4 de ses pièces gagne

Histoire

Résolu en 1988 : Blanc gagne commençant au milieu, noir peut forcer le remis si blanc commence sur une position adjacent à la position milieu, sinon noir gagne.

Présent

Aujourd'hui, le jeu peut être résolu en quelques minutes.

Analyse

- 7 possibilités pour un joueur
- 42 coups maximum par jeu
- L'arbre à $\approx 7^{42} \approx 3 \cdot 10^{36}$ noeuds
- Un maximum de $3^{42} \approx 10^{21}$ positions possibles
- Mieux : $(1 + 2 + 4 + 8 + \dots + 64)^7 \approx (2^7)^7 = 2^{49} \approx 5 \cdot 10^{15}$ positions possibles
- Effectivement : $4531985219092 \approx 5 \cdot 10^{13}$ positions possibles

Difficultés

- Arbre trop grand
- Stockage / comparaison de positions
- Détection de positions gagnante efficace

Astuces

- Heuristique pour évaluer une position
- Limiter la profondeur de l'arbre
- α, β -pruning
- Bit-fields pour stocker les positions

Heuristique d'évaluation $h(s, p)$

- $h(s, p)$ retourne -1 si la situation est perdante pour p , ($+1$ gagnante)
- $h(s, p)$ retourne une valeur dans $[-1, +1]$ sinon
- Une valeur proche de 1 signifie une situation favorable
- $h(s_1, p) > h(s_2, p)$ signifie que l'on estime s_1 plus favorable que s_2
- $h(s, p) = -h(s, 3 - p)$ (signe inversé pour la perspective de l'adversaire)

Limiter la profondeur de l'arbre

- Si à une profondeur d_{\max} le jeu n'est pas terminé, on utilise h à la place d'évaluer les situations enfants.

Évaluation à l'aide d'une heuristique

```
function evaluateState(state  $s$ , player  $p$ , depth  $d$ ,  $d_{\max}$ )  
  if  $d = d_{\max}$  or Game over then  
    return  $h(s, p)$   
  else  
     $b \leftarrow -1$  ▷ Best evaluation the current player can get so far  
    for all moves  $m$  for player  $p$  at state  $s$  do  
       $s' \leftarrow m$  applied to  $s$   
       $e \leftarrow \text{evaluateState}(s', 3 - p, d + 1, d_{\max})$   
      if  $-e = 1$  then  
        return 1 ▷ Optionally return winning move  
      else  
        if  $-e > b$  then  
           $b \leftarrow -e$  ▷ Optionally remember best move so far  
        end if  
      end if  
    end for  
    return  $b$  ▷ Optionally return best move remembered  
  end if  
end function
```

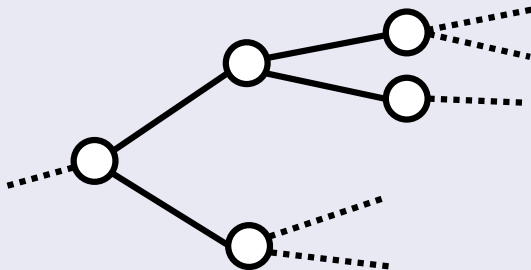
Bottlenecks

- Vitesse d'évaluation de $h(s)$
 - Bit-Fields
 - Function incrémentale
- Vitesse de génération de s'
 - Nouvelle structure de données, ou bien
 - modification de la structure courante puis revenir en arrière

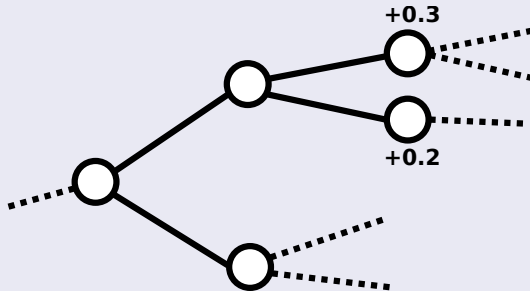
Exercice sur papier, Évaluations dans $[-10, +10]$, “Tree 6”

[http ://bit.ly/1dGzF2o](http://bit.ly/1dGzF2o)

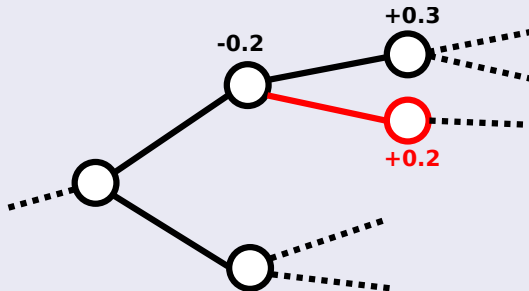
Idée de base du α, β -pruning



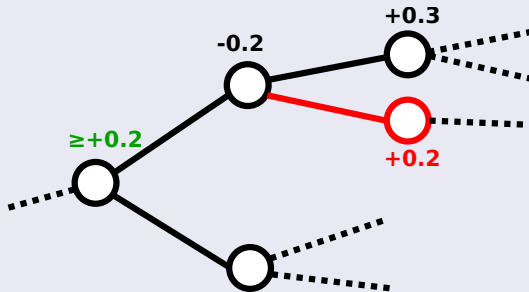
Idée de base du α, β -pruning



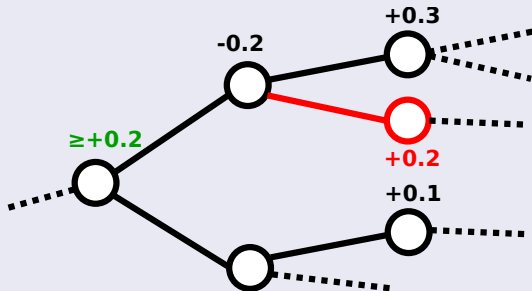
Idée de base du α, β -pruning



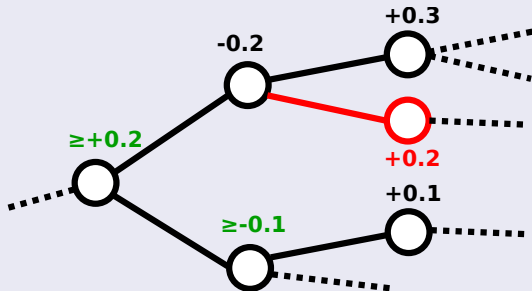
Idée de base du α, β -pruning



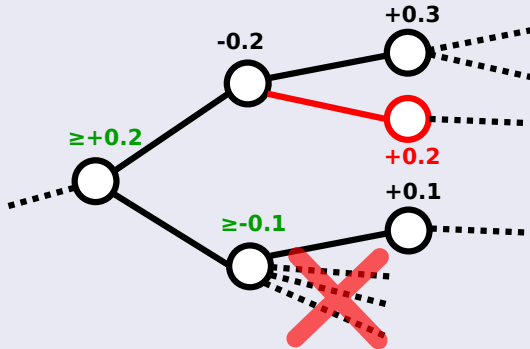
Idée de base du α, β -pruning



Idée de base du α, β -pruning



Idée de base du α, β -pruning



α, β -pruning

- A chaque appel, $[\alpha, \beta]$ est l'intervalle des évaluations intéressantes.
- Pour l'appel initiale, $[\alpha, \beta] = [-1, 1]$.
- $-\beta$: meilleure évaluation pour l'adversaire pour l'instant.
- Si je trouve mieux que β , l'adversaire ne choisira pas cette branche. Arrêter l'exploration.
- Si pour la situation courante l'intervalle est $[\alpha, \beta]$, pour la suivante c'est $[-\beta, -\alpha]$.
- α est mis à jour dès qu'une meilleure évaluation est trouvée.

Exercice sur papier, $[\alpha, \beta] = [-10, +10]$, "Tree 42"

<http://bit.ly/1dGzF2o>

α, β -pruning

```
function evaluateState(state  $s$ , player  $p$ , depth  $d$ ,  $d_{\max}$ ,  $\alpha, \beta$ )
  if  $d = d_{\max}$  or Game over then
    return  $h(s)$ 
  else
     $b \leftarrow -1$  ▷ Best evaluation the current player can get so far
    for all moves  $m$  for player  $p$  at state  $s$  do
       $s' \leftarrow m$  applied to  $s$ 
       $e \leftarrow \text{evaluateState}(s', 3 - p, d + 1, d_{\max}, -\beta, -\alpha)$ 
      if  $-e = 1$  then
        return 1 ▷ Optionally return winning move
      else
        if  $-e > b$  then
           $b \leftarrow -e$  ▷ Optionally remember best move so far
          if  $-e > \alpha$  then
             $\alpha \leftarrow -e$ 
            if  $\alpha \geq \beta$  then
              return  $b$  ▷ Optionally return this good enough move
            end if
          end if
        end if
      end if
    end for
    return  $b$  ▷ Optionally return best move remembered
  end if
end function
```

Détails d'implémentation

- Représentation en Bit-Field
- Évaluation d'une situation
- Mémoriser des résultats intermédiaires
- Iterative Deepening

Bit-Field

6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43
0	7	14	21	28	35	42

Encodage pour un joueur

- 49 bits au total pour 42 cases.
- Deux nombres 64-bit pour les deux couleurs c_1 et c_2 .
- Ligne en haut normalement 0.
- $c_1 \vee c_2$ donne un Bitmap des positions occupées.

Bit-Field

6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43
0	7	14	21	28	35	42

Encodage d'une situation

- Soit c_b le nombre avec les bits d'en bas mis à 1.
- $c_1 \vee c_2 + c_b$ donne l'enveloppe des positions occupées.
- $(c_1 \vee c_2 + c_b) \vee c_1$ encode la position de manière unique en 49 bits.
- Tous les opérations \vee sont disjoint, remplacer par $+$.
- $2c_1 + c_2 + c_b$ est la même chose.

Bit-Field

6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43
0	7	14	21	28	35	42

Détection de gain verticale

- $d \leftarrow c_1 \wedge (c_1 \gg 1)$
- d marque les positions avec deux pièces consécutives.
- $e \leftarrow d \wedge (d \gg 2)$ marque les positions avec quatre pièces.
- Quand $e \neq 0$ joueur 1 gagne.
- Ligne vide en haut est encore utile.

Bit-Field

6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43
0	7	14	21	28	35	42

Détection de gain horizontale

- $d \leftarrow c_1 \wedge (c_1 \gg 7)$
- d marque les positions avec deux pièces consécutives.
- $e \leftarrow d \wedge (d \gg 14)$ marque les positions avec quatre pièces.
- Quand $e \neq 0$ joueur 1 gagne.
- Les Bits 49 à 54 doivent être zéro.

Bit-Field

6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43
0	7	14	21	28	35	42

Détection de gain diagonale ↗

- $d \leftarrow c_1 \wedge (c_1 \gg 8)$
- d marque les positions avec deux pièces consécutives.
- $e \leftarrow d \wedge (d \gg 16)$ marque les positions avec quatre pièces.
- Quand $e \neq 0$ joueur 1 gagne.
- Les Bits 49 à 55 doivent être zéro.

Bit-Field

6	13	20	27	34	41	48
5	12	19	26	33	40	47
4	11	18	25	32	39	46
3	10	17	24	31	38	45
2	9	16	23	30	37	44
1	8	15	22	29	36	43
0	7	14	21	28	35	42

Détection de gain diagonale ↘

- $d \leftarrow c_1 \wedge (c_1 \gg 6)$
- d marque les positions avec deux pièces consécutives.
- $e \leftarrow d \wedge (d \gg 12)$ marque les positions avec quatres pièces.
- Quand $e \neq 0$ joueur 1 gagne.

Heuristique d'évaluation

- Très rapide (incrémentale ?)
- Précis (??)

Trade off

- Rapide : Grande profondeur possible
 - Utile vers la fin
 - Exploration complète possible
- Précis : Petite profondeur
 - Utile au début
 - Exploration complète impossible

Implémentation

- Compter le nombre de possibilités de faire 4 avec les pièces déjà posées
- Rapide
- Moyenne qualité
- Implémenté à l'aide d'opérations logique par bits

Hashtable

- Tester, si déjà évalué
- Pour chaque état, enregistrer l'évaluation
- Selon le jeu, gain considérable
- Encodage en 49 bits, excellent pour un hashtable
- Attention : L'évaluation peut être incomplète si $\alpha \geq \beta$

Move ordering

- Ordonner les coups selon une évaluation précédente
- Meilleur pruning.

Iterative deepening

- Augmenter la profondeur de recherche jusqu'à ce qu'un temps limite s'est écoulé
- Peu de overhead (facteur $\approx \frac{1}{m-1}$, m : nombre de coups en une situation)
- Utiliser les évaluations pour le move-ordering

Jeux sans stratégie d'ordinateur potable

- Go
 - 19×19
 - 150 à 250 coups par situation
- Quoridor
 - 11×11
 - 4 directions et 80-180 positions pour positionner un mur.

Résumée

- Représentation et évaluation efficace d'une situation ★
- Heuristique d'évaluation ★
- α, β -pruning
- Move ordering
- Hashtable
- Iterative deepening

Vérification de systèmes.

Vérification de systèmes.

Un *vérificateur* (verifier) joue contre un (modèle d'un) système et essaye, de mettre le système dans une situation d'erreur (le vérificateur gagne).

Vérification de systèmes.

Un *vérificateur* (verifier) joue contre un (modèle d'un) système et essaye, de mettre le système dans une situation d'erreur (le vérificateur gagne).

Si le vérificateur ne réussit pas (le système gagne), on considère le système étant correct.

Concrètement ?

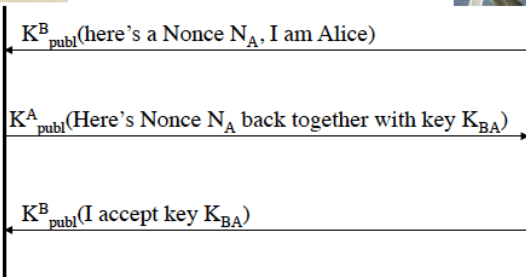
On considère un petit exemple pour comprendre l'idée de cet approche.

Le protocole Needham-Schroeder

Bob



Alice



Le protocole Needham-Schroeder comme jeux : les coups

L'objective du jeux consiste à echanger de manière sécurisé une clé secrète entre deux joueurs. Les coups possibles consistent en :

- Joueur 1 envoie un message à joueur 2, lisible uniquement par joueur 2, identifiant de manière unique joueur 1 et contenant un grand nombre aléatoire N fraîchemnent généré.
- Joueur 2 envoie un message à joueur 1, lisible uniquement par joueur 1, où il repète N et y ajoute un deuxièm grand nombre aléatoire K fraîchement généré.
- Joueur 1 envoie un message à joueur 2, lisible uniquement par joueur 2, répétant K .

Le protocole Needham-Schroeder comme jeux : Les règles

Les règles sont :

- Le vérificateur participe au jeux.
- Le vérificateur peut tout faire, ce qui est cryptographiquement possible :
 - Lire des messages, s'il dispose de la clé correspondant.
 - Intercepter des messages et les envoyer à une autre destination.
 - Changer la structure du message, si la structure est visible.
 - Rencrypter des messages, s'il dispose de la clé correspondant.
- Le vérificateur gagne, s'il connaît une clé, dont un autre joueur croit, qu'il n'est pas possible de le connaître.
- Le protocole gagne, si le vérificateur ne gagne pas.

Les joueurs ne peuvent pas se voir. Ils se basent uniquement sur les messages qu'ils reçoivent.

Les joueurs ne peuvent pas se voir. Ils se basent uniquement sur les messages qu'ils reçoivent.

En plus on suppose, que la cryptographie employée fonctionne parfaitement.

Illustrons au tableau ce qui se passe, si le vérificateur prend les rôles de Alice ou de Bob.

La “vérification” du protocole

Illustrons au tableau ce qui se passe, si le vérificateur prend les rôles de Alice ou de Bob.

Le protocol fonctionne donc ?

Plusieur joueurs (comme une partie simultanée d'échecs)

Illustrons au tableau ce qui se passe, si trois joueurs participent au jeux.

Plusieur joueurs (comme une partie simultanée d'échecs)

Illustrons au tableau ce qui se passe, si trois joueurs participent au jeux.

Aïe !

L'erreur dans le protocole Needham-Schroeder

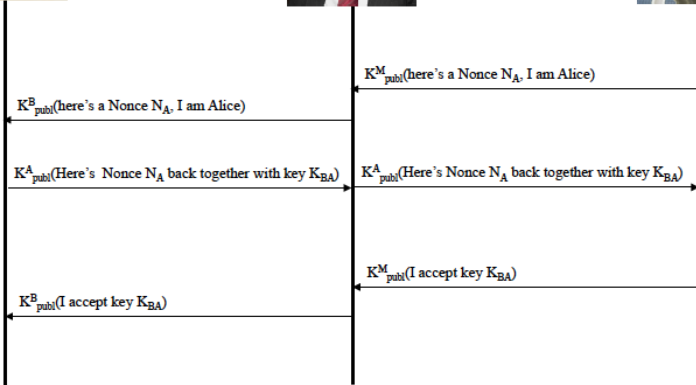
Bob



„man in the middle“



Alice



C'est possible qu'un jeux de vérification est gagné ou perdu seulement après un nombre de coups arbitraire.

C'est possible qu'un jeu de vérification est gagné ou perdu seulement après un nombre de coups arbitraire.

On risque de ne pas survivre aussi longtemps. . .

C'est possible qu'un jeu de vérification est gagné ou perdu seulement après un nombre de coups arbitraire.

On risque de ne pas survivre aussi longtemps. . .

Quel est le point alors ???

Quel est alors l'objectif ?

Quel est alors l'objectif ?

J'aimerais trouver une stratégie qui choisi les coups de sorte que je peux gagner (peut être seulement après un nombre arbitraire de coups).

Quel est alors l'objectif ?

J'aimerais trouver une stratégie qui choisi les coups de sorte que je peux gagner (peut être seulement après un nombre arbitraire de coups).

Je cherche seulement une *stratégie gagnante*.

Quand cette méthode de vérification fonctionne-t-elle ?

Quand l'espace de jeux est fini.

Quand cette méthode de vérification fonctionne-t-elle ?

Quand l'espace de jeux est fini.

Le jeu peut être représenté par un automate fini (avec des états acceptant et non-acceptant).

Quand cette méthode de vérification fonctionne-t-elle ?

Quand l'espace de jeux est fini.

Le jeu peut être représenté par un automate fini (avec des états acceptant et non-acceptant).

À l'aide d'un automate on peut tester algorithmiquement, s'il existe une stratégie gagnante du vérificateur.

C'est déjà tout pour l'instant à propos des jeux infinis

On espère que l'idée de base et l'application de vérification d'un système est devenu plus claire.

Questions ?

- 6 Publicité
 - Castor informatique
 - Cybercamp 2014
 - SSIE / SVIA

Castor informatique

- <http://castor-informatique.ch/>
- Concours online 11 au 15 novembre 2013
- Concours précédents disponibles :
http://concours.castor-informatique.ch/index.php?action=user_competitions

Cybercamp 2014

- [http ://cybercamp.unifr.ch/](http://cybercamp.unifr.ch/)
- 7 au 11 juilllet 2014
- Découvrir pendant une semaine le monde de l'informatique.

SSIE / SVIA

- [http ://svia-ssie-ssii.ch/](http://svia-ssie-ssii.ch/)
- Société Suisse de l'Informatique dans l'Enseignement
- Échange entre enseignants
- Informations sur divers offers
- Promotion de l'informatique aux écoles

Et maintenant la pratique

Au clavier!

Questions ?

Au clavier!

Questions ?

Alors ...

- Installer les logiciels
- Ouvrir les codes sources dans Netbeans
- Générer la JavaDoc
- Lire la JavaDoc