

Depuis que les progrès de l'algorithmique en ont donné la possibilité, les logiciels de bridge utilisent massivement les méthodes Monte-Carlo, aussi bien durant la phase d'enchères que durant la phase de jeu. Des échantillons de plusieurs dizaines ou centaines de mains aléatoires sont générés, tenant compte des connaissances probables ou certaines acquises sur les mains cachées, et les mains sont résolues à cartes ouvertes. Diverses méthodes permettent ensuite de choisir une enchère ou un coup majoritaire ou dominant. L'intérêt et les faiblesses de ces techniques ont été discutés par Frank (1998), Frank et Basin (2001), Ginsberg (2001). Leurs faiblesses résident principalement dans la fusion des stratégies. Avec AV2 en Sud et RX3 en Nord, l'ordinateur fait toujours 3 levées, puisqu'il prend correctement la Dame en impasse dans chacune de ses analyses à cartes ouvertes.

Une analyse exacte exige un arbre de jeu dans lequel sous chaque noeud MIN, il n'y a qu'un seul noeud MAX (le fait qu'il puisse y en avoir plusieurs non équivalents produit précisément la fusion de stratégies), ce qui définit une stratégie univoque et décidée *a priori* en fonction des probabilités des différents mondes. L'approche combinatoire est alors lourde et complexe puisqu'il faut simplifier des formes disjonctives massives à chaque création de noeud. Nous l'avons réalisée pour le maniement de la couleur qui se limite à 13 cartes, ce qui a permis d'analyser l'ensemble des positions de l'encyclopédie américaine du bridge (Francis & Truscott, 2002) ainsi que le dictionnaire des maniements de couleur de Roudinesco (1995) et de mettre en ligne un fichier exhaustif.

Cette approche n'est pas encore envisageable pour les parties réelles, du moins avec plus d'une quarantaine de cartes, puisqu'elle contraint à prendre en compte tous les mondes cachés possibles simultanément.. Elle le sera peut être dans un avenir proche grâce au calcul parallèle.

Nous présentons ici quelques principaux traits des méthodes de résolution des jeux à cartes ouvertes et à information complète, classiquement appelées solveurs double mort (*double dummy solver* : DDS), puisque, le joueur connaissant les quatre jeux, c'est comme s'il voyait deux morts en plus de sa main. Après avoir rappelé les principes de la recherche par partition, nous présenterons un modèle de structure de données adapté à la construction des tables de transposition.

La recherche par partition

Ginsberg (1997, 2001) a révolutionné l'analyse automatique du jeu de bridge en proposant la recherche par partition (RP) qui a rendue possible l'exploration complète des arbres de jeu.¹ L'auteur estime que la RP a amélioré l'alpha-beta dans les mêmes proportions que l'alpha-beta avait amélioré le minimax.. Le principe de la RP est très intuitif, même si sa mise en œuvre est sensiblement plus complexe.²

Tous les joueurs de bridge font de la recherche par partition car ils savent que beaucoup de positions sont équivalentes entre elles³.

La RP utilise des tables de transposition des positions. Disons, pour ne plus y revenir, qu'elles ne suffisent pas, ajoutée à alpha-beta à fenêtre nulle, à analyser des mains dans des temps raisonnables. Plusieurs algorithmes annexes sont nécessaires, notamment celui qui ordonne les coups, ceux qui détectent les plis rapides et les cibles atteintes, qui retiennent les meilleurs coups à chaque profondeur et nombre d'autres aménagements permettant d'optimiser le code et d'obtenir des coupes supplémentaires. Les données rapportées par Ginsberg (2001) par exemple, incluent nécessairement ces compléments bien qu'ils ne soient pas évoqués. Une revue en est donnée par Bo Haglund (<http://web.telia.com/~u07502278/>) qui met également en ligne une DLL et en publie les sources.

Les tables de transposition

L'expression la plus simple d'une table de transposition consiste à apparier une position P1, stockée lors de la remontée de l'arbre de recherche et dont la valeur est donc connue, avec une position P2 strictement équivalente, générée lors de la descente de l'arbre de recherche, alors que sa valeur n'est pas connue. Cet appariement se fait à la fin d'une levée, sa réussite implique que les cartes tombées soient identiques et que le joueur en main au début de la

¹ Pour donner un ordre de grandeur simple, rappelons que pour résoudre une partie de bridge de difficulté moyenne à cartes ouvertes, un ordinateur actuel utilisant simplement le minimax, aurait besoin d'environ vingt fois l'âge de l'univers en générant deux millions de nœuds par seconde. Ce temps est descendu à moins d'un dixième de seconde maintenant avec les meilleurs DDS.

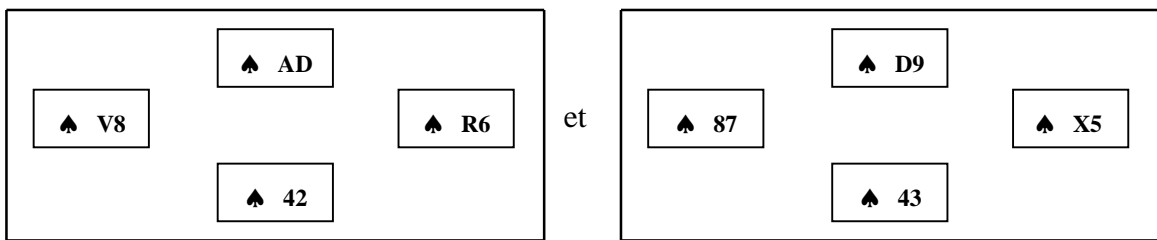
² On ne peut pas dire que Ginsberg (2001) a fait le maximum pour aider le lecteur car, à la façon dont on expliquait la sexualité aux enfants dans le passé en développant le cas des marguerites ou des truites, il expose des exemples tirés du jeu de morpion qui ne facilitent aucunement la compréhension du formalisme ultérieur.

³ Il y a tout d'abord le cas trivial des cartes ex-aequo, qui est traité indépendamment de la RP. Si votre main comporte RD de carreau, vous mettrez indifféremment l'un ou l'autre sans aucune conséquence sur le sous-arbre de jeu, autre que d'en diviser la taille par deux.

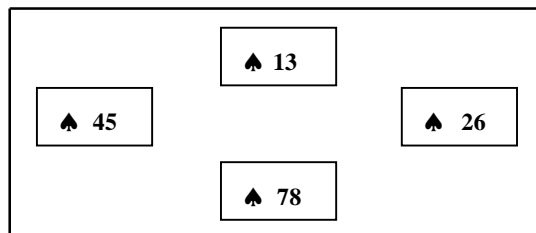
levée suivante soit le même. Cette technique demeure peu productive puisqu'elle n'apparie que des jeux identiques, l'ordre des cartes jouées étant simplement permuté.

A l'étape suivante, on remplace les rangs absolus des cartes par leurs rangs relatifs. On réalise alors une première compression et une véritable partition (tous les exemples de cet article sont donnés pour des contrats à sans atout).

Avec le même joueur en main :

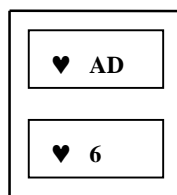


sont identiques et valent, en exprimant les cartes par leur rang:



L'étape suivante, qui est la contribution spectaculaire de Ginsberg, consiste à mettre en forme une idée intuitive : rendre équivalentes les basses cartes qui n'interviendront pas dans le cours de la partie, celles qui ne "jouent pas", que nous appellerons cartes inertes (CI).

Vous êtes en main en Sud avec :



Vous allez naturellement tenter l'impasse au Roi en Ouest. L'issue de cette ligne de jeu visant la réalisation de deux levées ne dépend **que** de la position du Roi. Si Ouest possède par exemple le Roi quatrième et que 10 cartes de la couleur sont encore en jeu, il y a :

$$C(6,3) = 20$$

combinaisons équivalentes des mains. Réussir à le dire au moyen d'un algorithme divise donc le sous-arbre de recherche par 20.

Il n'y a aucune règle permettant de savoir quelles sont les CI substituables les unes aux autres à partir du simple examen **prospectif** des quatre jeux. Ginsberg a défini les conditions **rétrospectives** dans lesquelles cette détermination est possible ainsi que les techniques à utiliser pour la conduire. Nous allons présenter les unes et les autres et nous proposerons ensuite un modèle de structure de données efficace adapté à la RP.

Technique des tables de transposition au jeu de bridge

Les positions équivalentes du point de vue de la valeur du jeu sont stockées dans des tables de transposition (TT). Au bridge les positions sont stockées dans la TT à la fin de chaque levée⁴. Il y a une TT distincte pour chaque joueur en main au début de la levée suivante (i.e. celui qui vient d'emporter la dernière levée) et pour chaque profondeur du jeu, de la quatrième à la quarante-huitième carte, (i.e. 12 profondeurs).

La TT est construite lors de la remontée de l'arbre, c'est-à-dire, après que la valeur du noeud a été obtenue, elle est consultée pour une coupe éventuelle lors de la descente. Elle contient donc les informations nécessaires à transmettre au noeud transposé qui va à son tour les remonter aux niveaux supérieurs.

La TT contient également toutes les conditions qui doivent être satisfaites pour qu'une transposition soit possible. Lors d'une recherche (les DDS utilisent alpha-beta à fenêtre nulle), une transposition aboutit à une redéfinition des bornes (i.e. nombre minimal et maximal de levées que fera MAX dans le reste du jeu), ce qui permet de conserver les mêmes TT avec des cibles différentes. C'est en comparant ces bornes avec les levées déjà réalisées à hauteur de la transposition que l'on peut décider d'une coupe alpha ou beta mais cette coupe n'est pas nécessairement obtenue lors de chaque transposition. Dans le meilleur des cas, une coupe

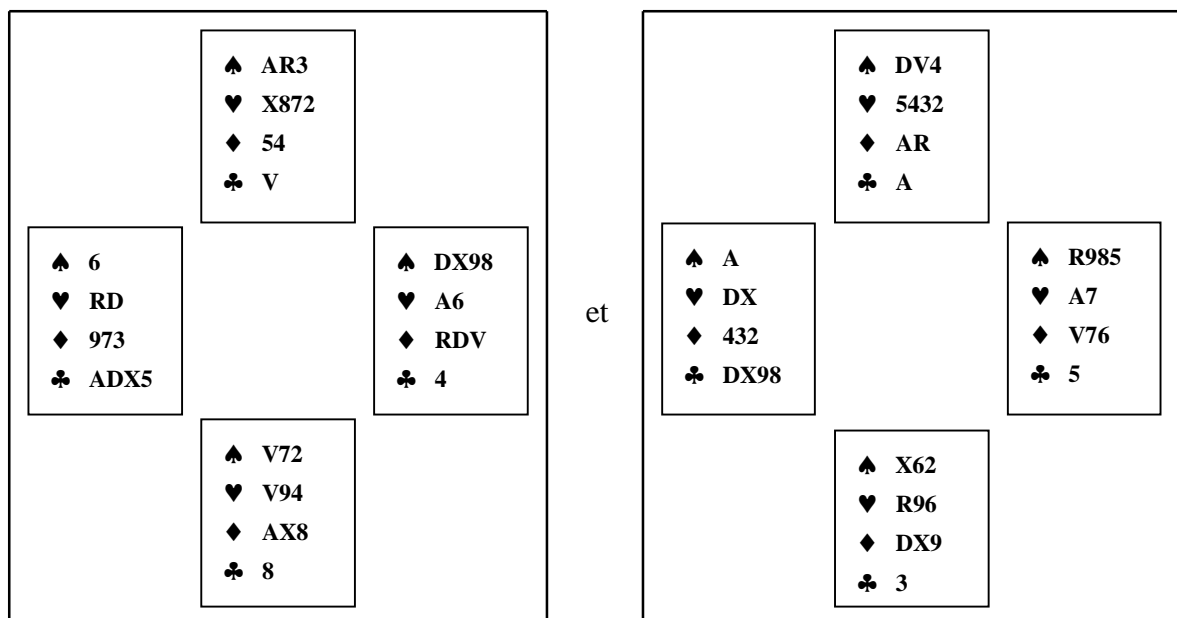
⁴ Il n'y a aucune raison qui empêcherait de construire une entrée pour chaque profondeur, mais deux inconvénients majeurs : une incertitude sur le vainqueur de la levée en cours qui contraint à élargir les marges et un quadruplement du calcul et de la mémoire requise. Nous avons fait quelques essais peu concluants mais la piste reste ouverte.

renvoie à l'oncle du noeud, dans un cas moins favorable (MIN atteint la cible ou bien MAX ne l'atteindra jamais) elle renvoie au frère suivant, dans le pire des cas, l'exploration se poursuit avec la génération des noeuds fils.

Conditions d'une transposition dans la RP

Il y a une seule condition préalable à la recherche d'une transposition, c'est celle d'une identité de la structure des quatre jeux en termes de longueurs des couleurs. Par exemple :

Figure 1.



remplissent cette condition préalable puisque les longueurs des couleurs, rangées par exemple de Sud à Est et de Trèfle à Pique, sont égales :

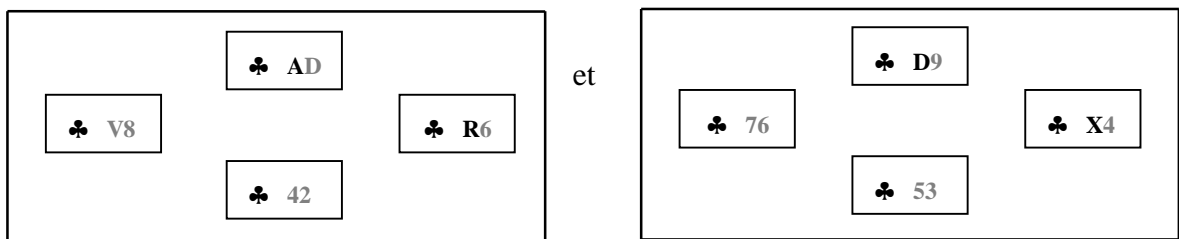
1333 4321 1243 1324

Ce codage de la position constitue l'entrée de la TT, il se fait sur un entier de 64 bits inséré dans un arbre binaire de recherche.

La seconde condition de la transposition est plus complexe, elle requiert que les rangs relatifs des cartes actives (CA, i.e. les cartes non inertes) soient les mêmes pour chaque joueur et chaque couleur.

Cette condition est intuitive :

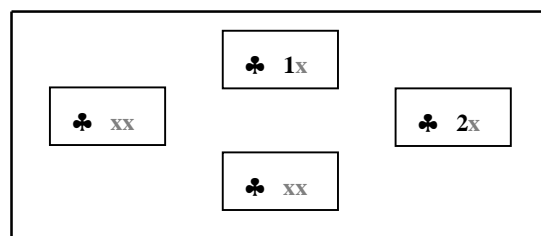
Figure 2.



sont équivalentes sous condition d'un jeu optimal de part et d'autre (i.e. règles du minimax).

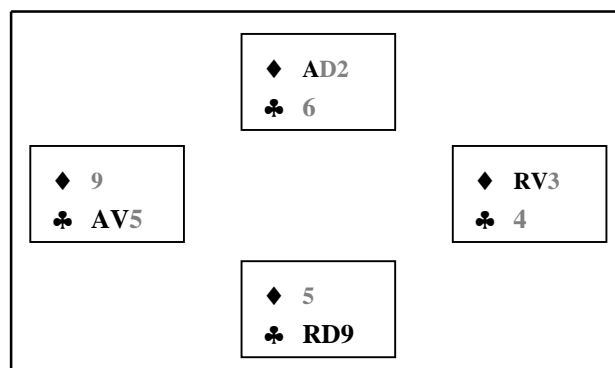
Avec Sud, Ouest ou Nord en main qui fourniront une levée à Nord, il n'y a que deux cartes actives, les autres (en grisé) sont inertes. Les positions pourraient être ramenées à :

Figure 3.



Evidemment, la situation se complique lorsqu'il y a plusieurs couleurs en jeu et que les cartes actives dépassent le nombre de levées. Par exemple, Ouest en main pour deux levées (n'importe quel joueur en main réalise ici deux levées) :

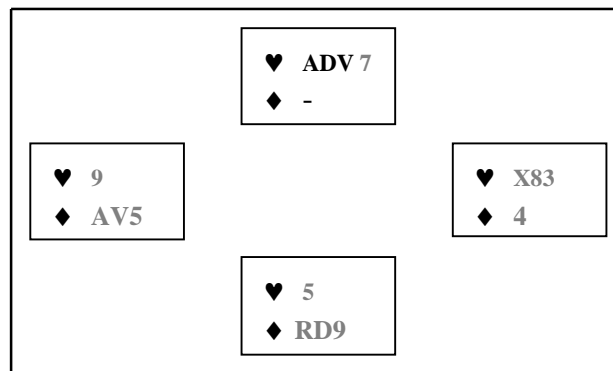
Figure 4.



Il y a huit cartes qui peuvent prétendre faire une levée dans le cadre du minimax et il n'y a que quatre levées au total, donc différentes lignes de jeu et une nécessité de faire l'analyse combinatoire pour détecter les CA. Ne sont donc CA que les cartes ayant réalisé une levée grâce à leur rang dans le sous-arbre et remontées par minimax.

On pourrait donc dire que les CI sont les cartes qui ne font jamais de levées. Ce serait inexact car une basse carte substituable à n'importe quelle autre peut faire une levée si elle demeure seule dans sa couleur. Par exemple, avec Sud ou Nord en main :

Figure 5.

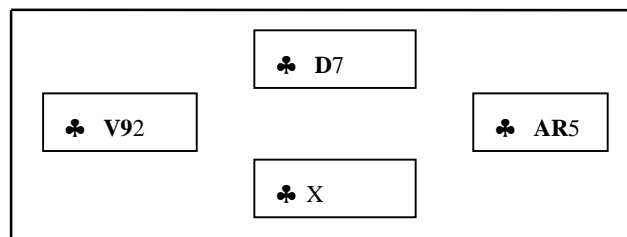


Il n'y a cette fois que trois CA, mais une CI, le 7 de Nord qui fera une levée. Le 7 doit être considéré comme une CI, car on pourrait le substituer au 3 d'Est sans que cela change quoi que ce soit à la valeur du jeu et Nord fera toujours quatre levées, alors que si l'on substitue le V au X, Nord ne fera plus que deux levées et le 7, toujours CI, n'en fera pas. C'est pour cette raison qu'en plus d'une identité de CA, deux positions transposables doivent présenter une identité des longueurs des couleurs.

Les cartes actives sont donc celles qui remportent une levée en raison de leur rang dans le sous-arbre dont le nœud étudié est la racine et les cartes inertes sont les autres, qu'elles remportent ou non des levées. La détermination des CA se fait à la fin de chaque levée ou lors des coupes alpha-beta à l'intérieur d'une levée. Si une carte a gagné une levée en raison de son rang, elle est remontée au nœud supérieur comme CA. Notons que ces règles s'appliquent indifféremment aux contrats à l'atout ou à sans atout.

Une fois déterminées, les cartes actives doivent être stockées en rangs relatifs dans la TT. Les plus faibles des cartes gagnant par rang dans chaque couleur déterminent la frontière inférieure des CA et il suffit de stocker une valeur par couleur, en étant très attentif aux cartes

ex aequo. Une main peut comporter plusieurs cartes de même rang dont la détermination dépend de la position courante, c'est-à-dire de l'ensemble des cartes demeurant dans les mains des joueurs. Si le valet de pique est tombé, le joueur possédant DX peut les considérer ex-aequo (sauf si V a été joué lors de la levée courante que D peut emporter, contrairement à X : 2VD8 et 2VX8 ne laissent pas le même joueur en main). Les cartes ex-aequo sont par définition substitutables, cela vaut aussi pour les cartes actives qui définissent les frontières basses des jeux équivalents dans les TT. Ce point peut devenir délicat et mérite une illustration.



Après **27AX R~~x~~9D 5~~x~~V~~x~~**, le V est remonté comme CA à la fin de la seconde levée, mais c'est le 9 qui doit être remonté à la fin de la première levée car, à ce moment, les deux cartes étaient équivalentes après la chute du X et V aurait pu être joué sur R à la seconde levée. C'est donc, à la fin de la première levée, 5 et non pas 4 CA qui doivent être prises en considération sous peine de réaliser des transpositions illégitimes.

Ultérieurement, lors de la visite de la table de transposition pour un nœud donné, si la première étape des longueurs égales est franchie (identité de l'entier de 64 bits évoqué ci-dessus), la seconde étape consistera à transformer les cartes de la position courante en rangs relatifs et, si toutes les CA ainsi converties appartiennent aux mêmes joueurs dans la position stockée en TT et dans la position courante, alors les deux positions sont équivalentes et il y a un appariement. Nous donnons ci-dessous l'algorithme de Haglund (ce dernier enrichi de toutes les procédures d'élagage annexes), pour situer les temps de remontée des CA et de constitution des TT. (L'algorithme plus formel de la recherche par partition de Ginsberg, 2001, est donné en annexe).

Figure 6. Pseudo-code du DDS de Bo Haglund (2008).

```
int Search(posPoint, target, depth) {
    if (player_side_to_move) {
        value=FALSE;
        moveExists=TRUE;
        while (moveExists) {
            Make;
            value=Search(posPoint, target, depth-1);
            Undo;
            if (value==TRUE) {
                MergeMoveData;
                goto searchExit;    }
            MergeAllMovesData;
            moveExists=NextMove;
        }
    } /* Opponents to move */
    else {
        value=TRUE;
        moveExists=TRUE;
        while (moveExists) {
            Make;
            value=Search(posPoint, target, depth-1);
            Undo;
            if (value==FALSE) {
                MergeMoveData;
                goto searchExit;    }
            MergeAllMovesData;
            moveExists=NextMove;
        }
    }
}
searchExit:
AddNewTEntry;
return value;
}
if (no_move_yet_in_trick) {
    TargetTooLowOrHigh;
    if (target_already_obtained)
        return TRUE;
    else if (target_can_no_longer_be_obtained)
        return FALSE;
    QuickTricks;
    LaterTricks;
    if (cutoff_for_player_side)
        return TRUE;
    else if (cutoff_for_opponent_side)
        return FALSE;
    RetrieveTResult;
    if (transposition_table_entry_match) {
        if (target_reached)
            return TRUE;
        else
            return FALSE;
    }
}
```

```

if (depth==0) {
    evalRes=Evaluate;
    if (evalRes.tricks >= target)
        value=TRUE;
    else
        value=FALSE;
    return value;
}
else {
    GenerateMoves;
    MoveOrdering;
    CheckMovesForCutoff;
}

```

Note : La remontée des cartes actives gagnant une levée par leur rang se fait avec les procédures *MergeAllMovesData* pour un noeud normal et *MergeMoveData* pour une coupe. La fabrication de la TT se fait dans *AddNewTTEntree*, sa visite dans *RetrieveTTResult* ainsi que dans *CheckMovesForCutoff*. Toutes les procédures sont détaillées sur (<http://web.telia.com/~u07502278/>).

Structure de données

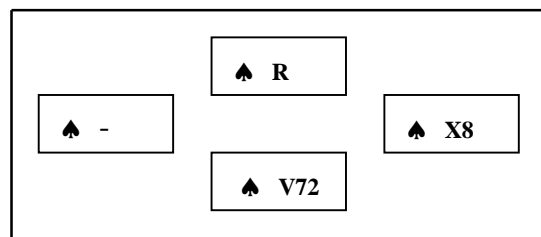
Le codage de la position sur 64 bits ne pose pas de problème. Celui des CA en revanche est plus compliqué. Une première solution, adoptée par Bo Haglund, était de lier à chaque noeud de la TT (dont la structure d'entrée est un arbre binaire) un arbre de CA. Lorsqu'une branche entière est parcourue avec succès, c'est-à-dire que toutes les cartes de la branche appartiennent aux joueurs de la position courante, il y a appariement. La dernière feuille pointe sur un noeud qui contient les caractéristiques de l'ensemble de positions associées à cette position: les bornes de la valeur du jeu pour Max principalement.

La solution que nous proposons apporte deux innovations. Tout d'abord stocker dans un même noeud toutes les CA d'une couleur et non les CA une à une et stocker les couleurs dans un ordre donné (e.g. TKCP). De cette façon l'arbre à explorer a une profondeur fixée de quatre noeuds et il est plus restreint. Elle permet ensuite de contourner une partie importante de la procédure de vérification. En raison du fait, trivial en soi, que les propriétaires des cartes ne changent jamais au cours de la partie (si l'As de Pique est en Sud, il y restera...), il est possible d'éviter les étapes de codage des cartes en rangs relatifs et de faire en sorte que toute l'information nécessaire soit simplement donnée par leur nombre. Pour ce faire, avant de commencer l'analyse, on dresse la liste de tous les sous-ensembles des cartes de chaque couleur (il y en a 8191 avec un codage des cartes allant de 0 à 12) et on y indexe un vecteur contenant les propriétaires des cartes qu'ils contiennent (ce qui requiert moins d'un centième de seconde pour les quatre couleurs). En codant chaque joueur sur 2 bits (par exemple Sud =

00; Ouest = 01; Nord = 10 et Est = 11, un registre de 32 bits stocke le vecteur des propriétaires de n'importe quel sous-ensemble de 13 cartes (taille maximale de 26 bits significatifs).

Reprenons la position de la figure 1. Si, par exemple, il reste à un moment donné de la partie :

Figure 7.



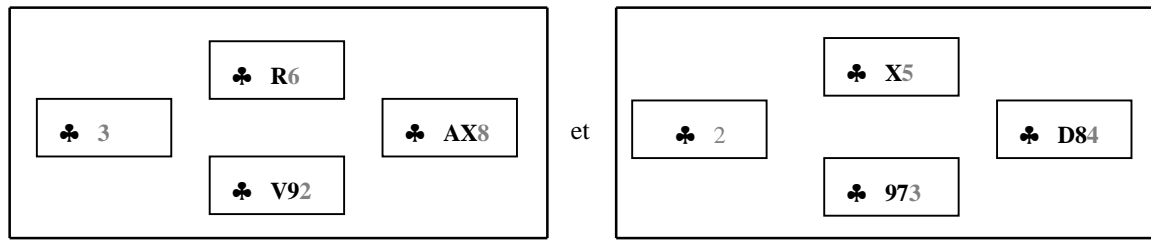
Le sous-ensemble résiduel **RVX872** (entrée 2913 du tableau des propriétaires : $2^{11}+2^9+2^8+2^6+2^5+2^0$) a pour propriétaires **NSEESS**.

La clé de ce codage est qu'il transforme automatiquement les rangs absolus en rangs relatifs et évite des conversions lourdes: nous savons, dans l'exemple ci-dessus, que N possède la carte de rang 1, S celle rang 2, etc. et le codage va autoriser des comparaisons directes de cartes en valeurs absolues entre positions. Quelles qu'elles soient, si elles appartiennent au même joueur et sont dans la même position sur le vecteur, alors leur rang relatif est identique. Si 3 cartes sont actives dans l'exemple ci-dessus, n'importe quel sous-ensemble de cartes dont les 3 premiers propriétaires seront NSE sera apparié.

Lors de la recherche d'appariement, on compare le vecteur des propriétaires des cartes de la position stockée en TT avec celui de la position courante. Si, sur la longueur des n CA stockées en TT, ces vecteurs sont équivalents, alors l'appariement est réalisé. On peut le faire commodément avec un masque comportant des 1 pour les CA et des 0 pour les CI. Ce masque est stocké dans la TT.

Comparons :

Figure 8.



Le vecteur des propriétaires des cartes pour la position de gauche est :

$$G = 111000110011100100 \quad (\text{ENSESEENOS})$$

pour la position de droite, il est :

$$D = 111000110010110001 \quad (\text{ENSESNESO})$$

Il y a 5 CA, donc le masque suivant (2 bits par CA, taille des propriétaires des cartes):

$$M = 111111111100000000$$

Le test est le suivant :

$$\text{Si } (G \text{ and } M) = (D \text{ and } M) \text{ alors appariement.}$$

Ou bien, c'est identique:

$$\text{si } (G \text{ xor } D) \quad \text{and } M = 0 \text{ alors appariement.}$$

Structure de la table de transposition

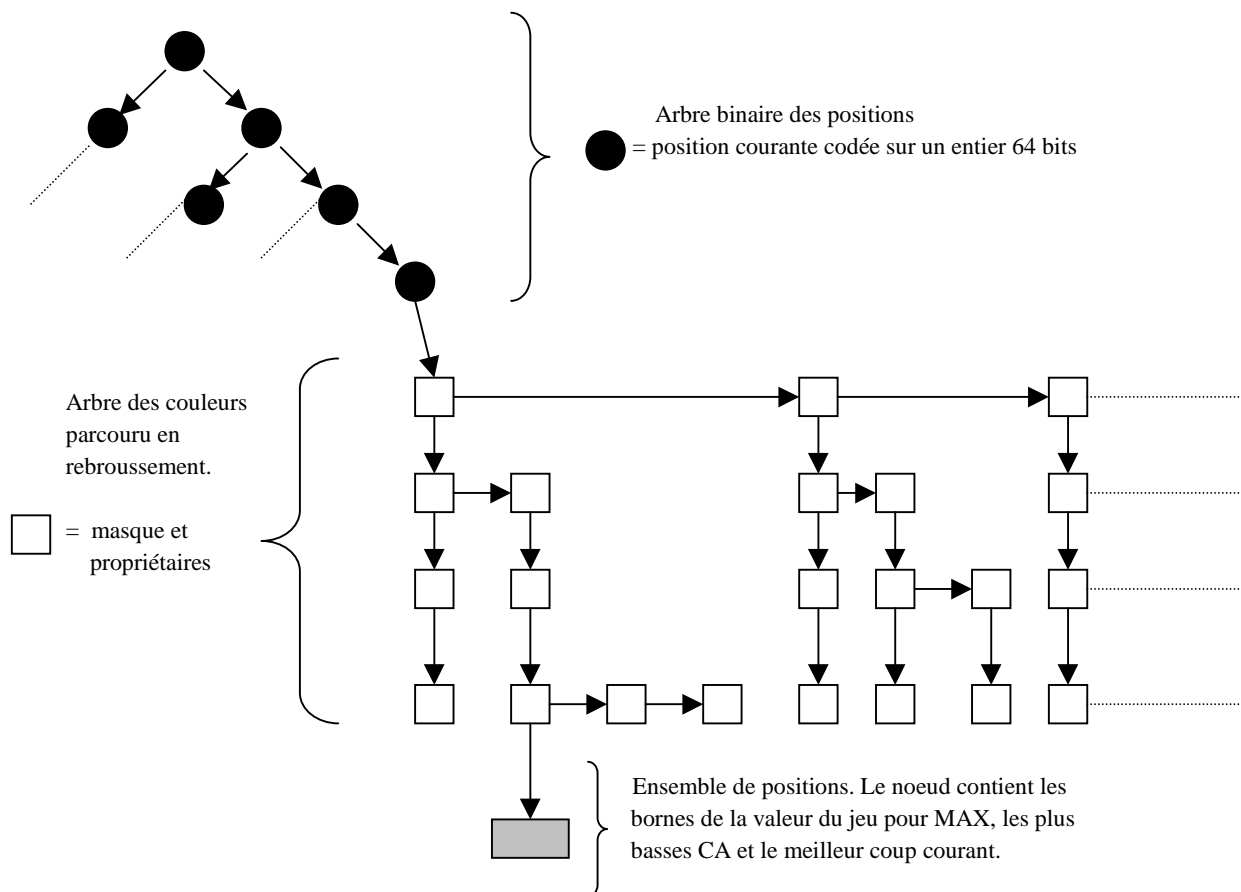
Lors de la constitution de la TT, on connaît pour chaque position le nombre de CA dans chaque couleur. Lors de la recherche d'appariement d'une position en revanche, on ne connaît pas ce nombre puisque le sous-arbre du dernier noeud n'a pas encore été exploré. Cela conduit à une indétermination qui fait que la position courante peut être appariée à plusieurs entrées

pour une même couleur (la position qui est fittée avec le noeud requerrant 4 CA à pique et la liste L de propriétaires des cartes, sera aussi fittée avec les noeuds en requerrant 3, 2, 1, 0 avec les listes L' compatibles avec L sur les 3, 2, 1, 0 premières CA).

Il en résulte que le constat d'échec d'un appariement exige le parcours total de l'arbre par rebroussement et non pas la simple recherche d'une liste comme si l'on était dans le cas d'un classement exhaustif et exclusif.

La figure 9 détaille la structure de la TT.

Figure 9. Structure d'une des 48 entrées d'une table de transposition.



Données expérimentales

Cette structure de données a été implémentée dans un DDS utilisé par le logiciel *DeepQueen* qui sera destiné à la compétition. Elle n'a pas été dans ce cadre directement comparée à d'autres structures de données, mais nous possédons, grâce à Bo Haglund, une

évaluation assez précise de son intérêt. Nous lui avons en effet suggéré de remplacer ses anciennes tables de transposition par le modèle présenté ici. Il a réalisé cette modification de façon très précise et a obtenu, sur un échantillon de 70 000 donnes une diminution régulière du temps global de calcul de 10 à 15% selon la difficulté des donnes⁵. Cette diminution étant obtenue sur les seules procédures de fabrication de TT et de recherche d'appariement (procédures *AddNewTTentry*, *RetrieveTTResult* et *CheckMovesForCutoff* de la figure 7) elle est assez considérable et témoigne de la double économie qu'elle permet de réaliser en traitant globalement les couleurs et non les cartes une à une, et en évitant des conversions qui sont latentes dans le tableau des propriétaires des cartes.

Il n'est pas possible de comparer ce modèle avec ceux utilisés par les quelques DDS dont les performances globales sont meilleures (notamment *Jack*, *Wbridge* et *GIB*) puisque le détail de ces derniers n'est pas connu. (Il existe un championnat du monde annuel des logiciels de bridge et les enjeux de la compétition font que ce que nous pouvons lire sous la plume de leurs auteurs demeure très allusif.) Toutefois, nous ne pensons pas, en accord avec Bo Haglund, que la supériorité de ces DDS réside maintenant dans les structures de données ou les techniques de calcul, mais plutôt dans des procédures d'élagage plus abouties, notamment au niveau des noeuds de fort branchement comme les attaques de levée, les défausses pures, les coupes ou défausses et la détection des déblocages d'une couleur. Les méthodes statistiques d'ordonnement des coups (régression multiple pouvant être conduite avec des réseaux de neurones) sont inférieures à des systèmes de règles capturant l'essentiel de l'expertise du jeu à cartes ouvertes, probablement plus simple que celle du jeu normal. Les systèmes de règles demeurent combinatoires dans leur structure et leur approche des cas complexes expose à des erreurs de moindre amplitude que celles de la régression. D'après diverses comparaisons directes ou indirectes, on peut estimer qu'un bon système de règles d'ordonnement des coups conduit à explorer en moyenne trois fois moins de noeuds qu'un système statistique sophistiqué.

Conclusion

Derrière l'intérêt que nous portons au jeu de bridge, il y a des préoccupations d'intelligence artificielle mais aussi d'intelligence de synthèse car les performances humaines sont encore loin d'être égalées dans ce domaine et, malgré les échecs continus pendant des décennies des

⁵ Cette implémentation s'est révélée assez simple et n'a généré aucune bogue. Les 70 000 donnes, probablement exactes puisque analysées de façon convergente par plusieurs DDS (dont celui de *Jack*, de Hans Kuijf, plusieurs fois champion du

logiciels de planification du jeu, il serait sans doute regrettable de se couper d'un dialogue avec la psychologie cognitive. Le jeu de bridge est un exceptionnel modèle de l'expertise humaine qui combine l'approche probabiliste liée à l'information non complète, à de puissants langages d'enchères et à une conceptualisation poussée et efficace du jeu de la carte. Il est à notre sens beaucoup plus stimulant que le jeu d'échecs, dont l'intérêt psycho-social est nul, mais auquel beaucoup de chercheurs et de psychologues, conservent un attachement historique.

Excepté pour le jeu de la carte, où ils fournissent des prestations du niveau de bons joueurs régionaux, les logiciels sont toujours assez faibles dans les enchères et laissent encore ouvertes d'innombrables pistes de réflexion. Il est probable que les développements du calcul parallèle, et à plus long terme des machines quantiques, rendront dérisoires les gains d'optimisation apportés par une analyse plus profonde du jeu et l'élucidation plus ou moins achevée des structures conceptuelles humaines. C'est regrettable car le risque sera grand de passer à côté de régularités remarquables du jeu et de perdre contact avec le travail de l'esprit qui, dans l'expertise humaine, réalise un considérable travail de compression de l'information et élabore des structures de données encore très au-dessus de nos meilleurs modèles. Les DDS ont coupé les ordinateurs de la simulation du jeu humain (ils n'ont par exemple aucun besoin des théories du *squeeze* pour le jeu de la carte et ils les réalisent parfaitement) et sa compréhension souffre à l'évidence des maux de cette optimisation trop rapide.

Références.

Frank, I . *Search and planning under incomplete information : a study using bridge card play*. Springer-Verlag, Distinguished Dissertation Series, 1998.

Frank, I & Basin, D. A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science*, 252, 2001, p. 217-256.

Francis, H.G., Truscott, A.F. *The official encyclopedia of Bridge. 6th edition*. New-York : D.A. Francis, 2002.

Ginsberg, M. GIB : *Imperfect information in a computationnaly challenging game*. *Journal of Artificial Intelligence Research*, 14, 2001, p.303-358.

Roudinesco, J.M. *Le dictionnaire des maniements de couleur*. Paris : Editions du Rocher, 1995.

ANNEXE. *Algorithme de recherche par partition, d'après Ginsberg (2001).*

Given a game (G, p_i, s, ev) and (P, R, C) a partition system for it, a position $p \in G$, cutoffs $[x, y] \subseteq [0, 1]$ and a transposition table T consisting of triples $(S, [a, b], v)$ with $S \subseteq G$ and $a < b, v \in [0, 1]$,

to compute $\alpha\beta(p, [x, y])$:

```

if there is an entry  $(S, [x, y], z)$  with  $p \in S$  return  $(z, S)$ 
if  $\text{ev}(p) \in [0, 1]$  then  $(v_{\text{ans}}, S_{\text{ans}}) = (\text{ev}(p), P(p))$ 
if  $\text{ev}(p) = \text{max}$  then
   $v_{\text{ans}} := 0$ 
   $S_{\text{all}} := \emptyset$ 
  for each  $p' \in s(p)$  do
     $(v_{\text{new}}, S_{\text{new}}) = \alpha\beta(p', [\max(v_{\text{ans}}, x), y])$ 
    if  $v_{\text{new}} \geq y$  then
       $T := T \cup (S_{\text{new}}, [x, y], v_{\text{new}})$ 
      return  $(v_{\text{new}}, S_{\text{new}})$ 
    if  $v_{\text{new}} > v_{\text{ans}}$  then  $(v_{\text{ans}}, S_{\text{ans}}) =$ 
       $(v_{\text{new}}, S_{\text{new}})$ 
       $S_{\text{all}} := S_{\text{all}} \cup S_{\text{new}}$ 
    if  $v_{\text{ans}} = 0$  then  $S_{\text{ans}} = C(p, S_{\text{all}})$ 
    else  $S_{\text{ans}} = R(p, S_{\text{ans}}) \cap (p, S_{\text{all}})$ 
if  $\text{ev}(p) = \text{min}$  then
   $v_{\text{ans}} := 1$ 
   $S_{\text{all}} := \emptyset$ 
  for each  $p' \in s(p)$  do
     $(v_{\text{new}}, S_{\text{new}}) = \alpha\beta(p', [\min(v_{\text{ans}}, x), y])$ 
    if  $v_{\text{new}} \leq x$  then
       $T := T \cup (S_{\text{new}}, [x, y], v_{\text{new}})$ 
      return  $(v_{\text{new}}, S_{\text{new}})$ 
    if  $v_{\text{new}} < v_{\text{ans}}$  then  $(v_{\text{ans}}, S_{\text{ans}}) = (v_{\text{new}}, S_{\text{new}})$ 
       $S_{\text{all}} := S_{\text{all}} \cup S_{\text{new}}$ 
    if  $v_{\text{ans}} = 1$  then  $S_{\text{ans}} = C(p, S_{\text{all}})$ 
    else  $S_{\text{ans}} = R(p, S_{\text{ans}}) \cap C(p, S_{\text{all}})$ 
   $T := T \cup (S_{\text{ans}}, [x, y], v_{\text{ans}})$ 
return  $(v_{\text{ans}}, S_{\text{ans}})$ 

```

Note: S est un ensemble de positions, R_0 et C_0 les ensembles de positions qui peuvent atteindre S ou qui sont contraintes de l'atteindre. Les fonctions R et C servent à construire ces ensembles par le biais de la remontée des cartes gagnant par rang (CF. Ginsberg, 2001, § 2.3. pour le détail du formalisme.
