Une maladie foudroyante pour l'un, un accident stupide pour l'autre, Martial Vivet et Jean-Marc Fouet nous ont quittés au cours de l'année 1999/2000. Parmi les plus anciens (ils étaient en 1974 à Caen au premier colloque d'Intelligence Artificielle), ils ont non seulement participé à de nombreux autres colloques, mais se sont aussi chargés, à plusieurs reprises d'en organiser et de nous accueillir dans leur université au Mans ou à Lyon.

Ils ont marqué les colloques et les réunions d'équipe par la qualité de leur travaux, leur rigueur, leur enthousiasme, leur dynamisme, et leur souvenir y restera toujours attaché.

Ces actes leur sont dédiés.

Sommaire

Introduction

L'équipe Métaconnaissances du LIP6 ainsi que des chercheurs d'EDF se sont réunis du 13 au 15 septembre 2000 dans l'île de Berder. Nous avons réuni dans ce rapport le texte de quelques unes des interventions qui y ont été faites.

La résolution de problèmes est au cœur de l'IA puisque presque toutes nos activités nous demandent de résoudre un problème. Il est souvent nécessaire de développer une arborescence, particulièrement dans les jeux. La procédure alpha-bêta permet d'en limiter beaucoup la taille, tout en n'éliminant que des branches qui ne conduiront certainement pas à la solution. Tistan Cazenave fait le point sur les différentes améliorations qui ont été apportées à cet algorithme. Fabrice Kocik développe une autre approche pour le cas où les informations dont on dispose sont imprécises ou inconnues, en particulier dans le cas des jeux de cartes où l'on ne connaît pas les cartes des adversaires. Il faut savoir monitorer le développement de l'arborescence en ne considérant pas les branches dont la probabilité est trop faible pour mériter d'être envisagées. Une autre difficulté rencontrée en résolution de problèmes est le cas où l'on a des solutions pour plusieurs sous-problèmes ; il faut composer ces solutions, qui sont en général partiellement incompatibles, pour définir la solution globale. Le système MARECHAL, qui joue à des war-games, a un module pour composer ainsi des solutions partielles. Par ailleurs, le système MALICE, qui est le résolveur général de problèmes de MACISTE, monitore la recherche de la solution pour en améliorer l'efficacité. La mise en œuvre de ce mécanisme de monitoring montre la nécessité d'inclure un métamonitoring pour éviter que le système ne perde trop de temps à monitorer cette recherche. Il est enfin toujours utile de comprendre comment les êtres humains se comportent lors des activités que nous avons de la peine à faire effectuer par un système d'IA. Aussi Dominique Pastre a-t-elle observé le comportement humain pendant la résolution d'une grande variété de problèmes, aussi bien mathématiques que de type casse-tête.
Nous nous intéressons également aux applications de l'IA à l'enseignement. Le groupe Combien ? montre comment un système d'enseignement peut gérer les erreurs de l'élève en utilisant une interface pédagogique adaptée. Par ailleurs, le système ARGOS s'attache à produire une explication claire des exercices qu'il a résolus de façon à pouvoir donner à l'élève une preuve en français qui soit adaptée à son niveau mathématique.

Nous remercions Fabrice Kocik et Tristan Pannérec qui ont parfaitement pris en charge l'organisation matérielle de ce colloque. Merci aussi à Fabrice Kocik qui s'est chargé de l'édition de ces actes.

Jacques Pitrat

Des Optimisations de l'Alpha-Béta

Tristan Cazenave
Laboratoire d'Intelligence Artificielle
Département Informatique, Université Paris 8,
2 rue de la Liberté, 93526 Saint Denis, France.
cazenave@ai.univ-paris8.fr

Résumé. Nous passons en revue les principales optimisations apportées à l'algorithme Alpha-Béta. Nous présenterons tout d'abord le minimax, puis le négamax, ensuite l'Alpha-Béta lui même. Nous évoquerons ensuite les optimisations classiques comme l'approfondissement itératif, les tables de réfutations, la quiescence, l'ordonnancement, les coups nuls, les fenêtres nulles, les tables de transpositions. Nous en viendrons alors a décrire nos propres optimisations qui permettent des gains très conséquents: la recherche abstraite de preuves et l'élargissement itératif.

Mots clés: Alpha-Béta, recherche abstraite de preuve, élargissement itératif.

1. Introduction

Nous nous intéressons à l'optimisation de la recherche arborescente et plus particulièrement des arbres Min-Max. Les techniques développées pour cette sorte d'arbre peuvent se révéler directement utiles, ou être à la source d'autres idées pour des systèmes qui développent d'autres formes d'arbres comme les démonstrateurs de théorèmes, les résolveurs de problèmes, ou les systèmes de satisfaction de contraintes.
L'algorithme Minimax et ses améliorations est utilisé dans la plupart des jeux à deux joueurs, à somme nulle et à information complète. Cet algorithme a été proposé il y a plus de cinquante ans par von Neumann et Morgenstern [Neumann et Morgenstern 1944] pour trouver un coup aux Echecs. Alan Turing [Turing 1953] a proposé des stratégies de recherche basées sur le principe du Minimax, et une amélioration importante fut l'alpha-Beta. Une forme affaiblie d'Alpha-Béta est apparue dans les premiers programmes d'Echecs comme NSS, de Newell, Shaw et Simon [Newell et al. 1958].
On se place maintenant dans le cadre des jeux à deux joueurs. L'algorithme MiniMax, ses variantes et améliorations sont utilisés dans la majorité des programmes de jeux à deux joueurs, à somme nulle et à information complète.

2. Améliorations usuelles du Minimax

On rappelle qu'une fonction d'évaluation prend en entrée une position dans un jeu et donne en sortie une évaluation numérique pour cette position. L'évaluation est d'autant plus élevée que la position est meilleure.

Si une fonction d'évaluation est parfaite, il est inutile d'essayer de prévoir plusieurs coups à l'avance. Toutefois, pour les jeux un peu complexes comme les Echecs, on ne connaît pas de fonction d'évaluation parfaite. On améliore donc un programme si à partir d'une bonne fonction d'évaluation on lui fait prévoir les conséquences de ses coups plusieurs coups à l'avance. L'hypothèse fondamentale du MiniMax est que l'adversaire utilise la même fonction d'évaluation que le programme.

Notre but est de trouver le coup qui **maximise** la fonction d'évaluation, alors que le but de l'adversaire est de choisir le coup qui **minimise** la fonction d'évaluation. Or les deux adversaires jouent chacun leur tour et en général c'est le joueur ami qui joue en premier puisqu'on cherche le meilleur coup à jouer pour le joueur ami. On va donc choisir les coups maximisant aux niveaux pairs et les coups minimisant aux niveaux impairs de l'arborescence, si on fait l'hypothèse que le premier niveau est le niveau 0.

Algorithme MiniMax, sachant qu'on appelle MiniMax(Position,0) pour trouver le meilleur coup :

```
#define PROFMAX 5 // Marche pour tous les niveaux
#define INIFNI MAXINT
#define odd(a) ((a)&1)
int MiniMax(char *Position,int profondeur)
   {
    int valeur,Best,i,N;
    char *PositionSuivante[100];

    if (profondeur==PROFMAX)
      return Evaluation(Position);

    N=TrouveCoupsPossibles(Position,PositionSuivante);

    Best=-INFINI;
    for (i=0; i<N; i++)
       {
              valeur=MiniMax(PositionSuivante[i],profondeur+1)
              if (odd(profondeur)) // niveaux impairs, on minimise
          {
           if (valeur<Best)
                 Best=valeur;
              }
              else if (valeur>Best) // niveaux pairs, on maximise
                Best=valeur;
            }
    return Best;
   }
```

2.1. Le NegaMax

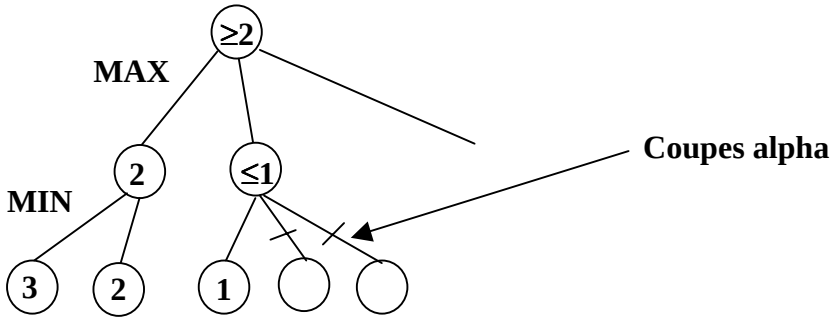
Plutôt que de tester si on est à un niveau pair ou impair pour savoir si on cherche à maximiser ou à minimiser l'évaluation, on peut inverser le signe des évaluations à chaque niveau, et toujours chercher à maximiser. On a alors l'algorithme NegaMax :

```
#define PROFMAX 4 // Ne marche que pour les niveaux pairs
#define INIFNI MAXINT
int NegaMax(char *Position,int profondeur) {
    int valeur,Best,i,N;
    char *PositionSuivante[100];

    if (profondeur==PROFMAX)
      return Evaluation(Position);

    N=TrouveCoupsPossibles(Position,PositionSuivante);

    Best=-INFINI;
    for (i=0; i<N; i++) {
              valeur=-NegaMax(PositionSuivante[i],profondeur+1)
              if (valeur>Best)
                Best=valeur;
           }
    return Best;
}
```

2.2. L'algorithme Alpha Beta

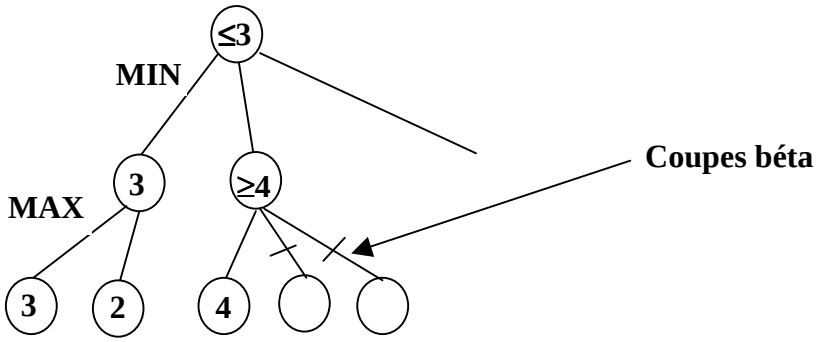
La coupure alpha se fait aux niveaux Min. Elle est basée sur l'observation que si la valeur d'un niveau Min est plus petite que la valeur du niveau Max supérieur, quelques soient les valeurs des nœuds suivants le niveau Min, ils ne changeront pas la valeur du niveau Max supérieur.

Exemple :



La coupure béta est la coupure symétrique de la coupure alpha pour les niveaux Max.

Exemple :



Propriété : lorsque le minimax trouve un coup en n nœuds, l'alpha béta trouve ce même coup en 1+2√(n) nœuds si les coups sont bien ordonnés [Knuth 75].L'ordre des coups est important car du bon ordre dépend le nombre de coupes.

Algorithme AlphaBeta, on appelle la fonction avec AlphaBeta (Position,0,-INFINI,INFINI) :

```
#define PROFMAX 4 // Ne marche que pour les niveaux pairs
#define INFINI MAXINT
#define odd(a) ((a)&1)
int AlphaBeta(char *Position,int profondeur,int alpha,int beta) {
    int valeur,Best,i,N;
    char *PositionSuivante[100];

    if (profondeur==PROFMAX)
      return Evaluation(Position);

    N=TrouveCoupsPossibles(Position,PositionSuivante);

    if (odd(profondeur)) Best=INFINI;
    else Best=-INFINI;
    for (i=0; i<N; i++) {

valeur=AlphaBeta(PositionSuivante[i],profondeur+1,alpha,beta);
        if (odd(profondeur)) { // on minimise
    if (valeur<Best) {
       Best=valeur;
       if (Best<beta) {
          beta=Best;
          if (alpha>beta) return Best; // coupure alpha
       }
    }
    else if (valeur>Best) { // on maximise
      Best=valeur;
      if (Best>alpha) {
```

```
      alpha=Best;
       if (alpha>beta) return Best; // coupure beta
      }
    }
  }
  return Best;
}
```

2.3. Le NegaMax avec coupures Alpha Béta

On peut utiliser des coupures Alpha et Béta dans le negamax. On obtient alors l'algorithme suivant :

```
#define PROFMAX 4 // Ne marche que pour les niveaux pairs
#define INFINI MAXINT
int NegaAlphaBeta(char *Position,int profondeur,int alpha,int beta)
{
    int valeur,Best,i,N;
    char *PositionSuivante[100];

    if (profondeur==PROFMAX)
      return Evaluation(Position);

    N=TrouveCoupsPossibles(Position,PositionSuivante);

    Best=-INFINI;
    for (i=0; i<N; i++) {
        valeur=-NegaAlphaBeta(PositionSuivante[i],profondeur+1,-
beta,-alpha);
        if (valeur>Best) {
          Best=valeur;
          if (Best>alpha) {
            alpha=Best;
            if (alpha>beta) return Best;
          }
        }
    }
    return Best;
}
```

2.4. L'effet d'horizon, approfondissement sélectif et quiescence

Il y a deux inconvénients au fait qu'on doive fixer une profondeur maximum. Le premier étant que le programme ne peut prévoir les effets d'un coup à une profondeur dépassant la profondeur maximum. Le second est que cela introduit des effets pervers dans les choix du programme : le programme fera toutes les menaces qu'il peut et qui sont pourtant inutiles pour repousser au-delà de son horizon un événement qui lui est défavorable. C'est ce qu'on appelle l'effet d'horizon : le programme repousse les événements défavorables au-delà de son horizon.

La solution utilisée dans Deep Blue comme parade à l'effet d'horizon est d'utiliser une méta-

fonction d'évaluation qui n'évalue pas la position mais plutôt qui évalue le type de la position. Cette méta-fonction d'évaluation évalue si une position est stable (essentiellement, s'il reste des pièces en prise). Elle est utilisée par Deep Blue pour savoir si une position est évaluable ou si on doit continuer à la développer. Une position stable a une évaluation en laquelle on peut avoir confiance alors qu'une position instable est mal évaluée. Deep Blue utilise ce mécanisme de quiescence pour continuer de développer les positions basses de l'arbre jusqu'à des positions stables.

Les options de quiescence couramment utilisées dans les programmes d'Echecs sont les coups de capture, les coups de promotion (sauf quand le Roi est en Echec).

On peut aussi utiliser **l'élagage de futilité** lorsqu'on développe les positions instables : si la valeur de la position est inférieure à alpha-V alors seuls les captures et les échecs sont considérés.

2.4.1. Selective deepening

Si un coup semble intéressant, on continue à chercher à une plus grande profondeur que la profondeur habituelle. Si un coup semble mauvais, on arrête de chercher à une profondeur plus petite que la profondeur habituelle.

Par exemple, si Chinook analyse un coup qui perd 3 pions, plutôt que de continuer à analyser la position jusqu'à une profondeur 10, il va réduire son analyse à seulement 5 coups à l'avance en faisant l'hypothèse qu'il y a de bonnes chances que le coup soit très mauvais. Par contre, si le programme joue un coup qui paraît très bon, il augmentera la profondeur de l'analyse de 10 à 12 coups à l'avance.

D'après J. Schaeffer, c'est une décision d'investissement : on investit son capital (le temps d'analyse) là où on espère avoir le meilleur bénéfice (on cherche le plus d'information possible).

Un autre mécanisme de selective deepening est utilisé dans Deep Blue. Il analyse la structure de l'arborescence pour savoir si une branche de l'arbre développé est uniforme ou pas. Ainsi une branche dans laquelle beaucoup de coups sont bons est considérée comme ayant une évaluation sûre. Alors qu'une branche pour laquelle 1 seul coup parmi 30 est bon est considérée comme peu sûre. Deep Blue développe alors plus que les autres la partie de l'arborescence qui ne contient qu'un seul bon coup sur 30.

On peut considérer les évaluations utilisées pour activer le selective deepening comme des méta-fonctions d'évaluation.

2.4.2. Recherche de Quiescence

Un exemple de procédure de recherche de quiescence est donné ci-dessous. La fonction TrouveCoupsPossibles ne selectionne que les coups liés à la quiescence (captures et promotions). A chaque feuille de l'arborescence Alpha-Beta principale, on appelle la fonction quiescence pour évaluer la position.

```
int TableauCoups[PROFMAX][NB_COUPS_MAX];
char Position[TAILLE_POSITION];
int Quiescence(int profondeur,int alpha,int beta) {
    int valeur,Best,i,N;
    int *CoupsPossibles=TableauCoups[profondeur];

    Best=-Evaluation(-beta,-alpha);

    N=TrouveCoupsPossibles(Position,CoupsPossibles);

    for (i=1; i<CoupsPossibles[0]+1; i++) {
        JoueCoup(Position,CoupsPossibles[i]) ;
        valeur=-Quiescence(profondeur+1,-beta,-alpha);
        DejoueCoup() ;
        if (valeur>Best)
   Best=valeur;
}
    return Best;
   }
```

2.5. Le coup nul

Un coup nul (null move) consiste à changer le tour de jeu, ce qui est équivalent pour un joueur à passer son tour. Le coup nul est un coup légal au jeu de Go où un joueur peut passer, c'est même de cette façon que la fin de partie est décidée : lorsque les deux joueurs passent consécutivement. Aux Echecs par contre, ce n'est pas un coup légal, et il existe un petit nombre de positions pour lesquelles cela pourrait être utile. Ce sont les positions de Zugzwang, le premier joueur qui joue dans une position de ce type a perdu.

L'heuristique du coup nul permet de détecter des coups inutiles et de gagner du temps en ne développant pas des arborescences inutiles. On suppose que jouer le coup améliore la position (pas de Zugzwang). Toutefois cette heuristique peut parfois masquer une arborescence qui si elle avait été explorée aurait changer le résultat de l'Alpha-Béta. Contrairement aux coupes de l'Alpha-Béta qui ne changent pas le résultat du MiniMax, les coupes dues au coup nul peuvent changer le résultat de l'Alpha-Béta.

Le joueur joue deux coups de suite et effectue une recherche à un profondeur inférieure. Si le résultat de la recherche est inférieur à alpha, alors le coup n'est pas étudié plus profondément. On utilise un facteur de réduction R pour choisir la profondeur de recherche du coup nul. Si

on est dans un nœud pour lequel on s'apprête à faire une recherche de profondeur P, la profondeur de la recherche associée au coup nul sera P-R. On choisit en général R=2 ou R=3.

2.6. L'approfondissement itératif

L'approfondissement itératif commence par effectuer une recherche de profondeur 1, puis recommence avec une recherche complète à profondeur 2, et continue ainsi à faire des recherches à des profondeurs de plus en plus grandes jusqu'à ce qu'une solution soit trouvée.

Puisqu'une telle recherche n'engendre jamais un nœud tant que les nœuds de profondeurs plus petites n'ont pas été engendrés, elle trouve toujours la solution la plus courte. De plus si l'algorithme se termine à une profondeur d, sa complexité en espace est en O(d), c'est à dire linéaire en fonction de la profondeur de recherche.

A priori, l'approfondissement itératif perd beaucoup de temps dans les itérations précédant la solution. Toutefois, ce travail supplémentaire est généralement beaucoup plus petit que la dernière itération. S'il y a n coups possibles en moyenne pour chaque position, le nombre de positions à la profondeur d est n^d. Le nombre de nœuds à la profondeur d-1 est de n^{d-1} pour la d-1ème itération et chaque nœud est engendré deux fois, une fois pour l'itération finale, une fois pour l'avant dernière itération. Le nombre de nœuds engendré est donc de $n^d + 2n^{d-1} + 3n^{d-2} + 4n^{d-3} + \ldots + dn$. Ce qui est en $O(n^d)$. Si n est assez grand, le premier terme est nettement plus grand que les autres, c'est donc la dernière itération qui prend la plus grande partie du temps. Par rapport à la recherche en largeur d'abord, l'approfondissement itératif engendre n/(n-1) fois plus de nœuds. Si on considère l'asymptote, l'approfondissement itératif est un algorithme de recherche optimal aussi bien en temps qu'en espace.

Les algorithmes de recherche en profondeur d'abord ont un autre avantage important sur les algorithmes en meilleur d'abord ou en largeur d'abord : il permettent d'utiliser à moindre coût des informations incrémentales sur la position. L'ordre de recherche des positions permet de connaître les informations sur la position avant le coup qui a mené à cette position. On peut utiliser ces informations pour recalculer plus rapidement les propriétés d'une position, en ne calculant que la différence avec la position précédente induite par le coup.

2.7. Les tables de réfutation

Une heuristique qui fonctionne bien avec l'approfondissement itératif est de retenir à chaque itération la suite de coups optimaux calculés par l'Alpha-Béta en réponse à chaque coup à la racine. Chaque coup à la racine est donc associé à une séquence de coups qui le suivent et qui

constitue sa réfutation. On a ainsi les séquences de coups qui sont jugée les meilleures pour les deux joueurs à une profondeur donnée. L'heuristique consiste à essayer les coups de la table de réfutation en premier dans l'arborescence. Ce sont ceux qui ont le plus de chances d'être les meilleurs puisqu'ils étaient les meilleurs à la profondeur précédente. Or on a vu qu'essayer les meilleurs coups en premier permet de maximiser le nombre de coupes Alpha Béta, et donc de minimiser le temps de recherche.

2.8. Le coup qui tue et l'heuristique de l'historique

L'ordre dans lequel on considère les coups a une grande influence sur l'efficacité de l'algorithme Alpha Béta. Un coup qui marche beaucoup mieux que les autres coups du même niveau dans un sous-arbre a de bonnes chances de bien marcher dans un autre sous-arbre. On va donc essayer ce 'killing move' en priorité dans les sous-arbres suivants. Cette heuristique est utilisée dans de nombreux programmes de jeux, entre autres Deep Blue et GoTools (résolution de problèmes de vie et de mort au jeu de Go). En général, les programmes d'Echecs utilisent cette heuristique en mémorisant le meilleur coup ou les deux meilleurs coups pour chaque profondeur de recherche.

Une généralisation des coups qui tuent est l'heuristique de l'historique (*history heuristic* [Schaeffer 83,89]) : Une note est mise à jour pour chaque coup légal rencontré dans l'arbre de recherche. A chaque fois qu'un coup est reconnu comme le meilleur dans une recherche, sa note est ajustée d'un montant proportionnel à la profondeur du sous-arbre exploré. On ajoute $2^{\text{ProfondeurMax-p}}$ à la note du meilleur coup, p étant la profondeur à laquelle a été essayé le coup. On ordonne les coups à tester dans l'Alpha-Béta en fonction de leur note. J. Schaeffer a montré que l'heuristique de l'historique associée aux tables de transposition est responsable de 99% des réductions de recherche dans l'Alpha-Béta [Schaeffer 89] .

2.9. La recherche aspirante

Plutôt que d'appeler l'Alpha-Beta avec des valeurs initiales de –INFINI pour alpha et +INFINI pour beta, on peut lui permettre de couper plus de branches, si on augmente (resp. diminue) la valeur initiale de alpha (resp. beta). Si la valeur finale trouvée est comprise entre les alpha et beta initiaux, le résultat sera tout de même juste, bien que l'on ait coupé plus de branches inutiles qu'avec des valeurs infinies.

La recherche aspirante permet de régler les valeurs initiales pour alpha et beta en prenant en compte les résultats de la recherche précédente. Au début de chaque itération, les valeurs maximales (resp. minimales) sont initialisées avec les valeurs remontées de l'itération

précédente, additionnées (resp. diminuées) de la valeur d'un pion. Si la recherche échoue (les valeurs ne sont pas comprises dans la fenêtre), la fenêtre est ajustée à (-INFINI,valeur) si on échoue vers le bas, ou (valeur,+INFINI) si on échoue vers le haut.

2.10. La fenêtre minimale, La variation principale, La fenêtre nulle

En poussant cette idée jusqu'au bout, on obtient l'idée de la fenêtre nulle. Cela consiste à appeler l'alpha-béta avec une fenêtre (valeur,valeur+1), sachant que la fonction d'évaluation est entière avec des différences d'évaluation minimales de un point. Les arbres développés sont alors plus petits, et on peut ajuster vers le haut ou vers le bas la valeur en fonction de ce que retourne l'arbre. Il a été montré que cet algorithme, qui associé aux tables de transposition s'appelle MTD(f), développe les feuilles de l'arbre dans le même ordre qu'un algorithme en meilleur d'abord qui a été prouvé meilleur qu'alpha-béta: SSS*.

2.11. Les Tables de Transposition

Une table de transposition sert à stocker les positions déjà rencontrées dans une table de hachage. Lorsqu'on rencontre une position, on commence par vérifier si elle a déjà été vue. Pour cela, on a un problème de mémoire : coder la position sur 32 ou 64 bits. De plus, la rapidité de codage est importante. La méthode la plus fréquemment utilisée est le hachage de Zobrist : on associe un nombre aléatoire pour chaque position de pièce possible, et pour le joueur qui a la main.
Au Go cela revient a avoir : 2 nombres aléatoires par intersection + couleur qui joue + Ko. Le hachage de la position est le XOR de tous les nombres présent sur la position.
Au Echecs on a 12 nombres aléatoires par case (un par pièce différente) + 4 nombres pour les droits de roques + 8 nombres pour les captures en passant + 1 pour la couleur qui joue. Soit 64*12+4+8+1=781 nombres aléatoires.
Il y a deux avantage à utiliser le XOR pour coder une position :

- Le XOR est une opération très rapide sur les bits.
- La valeur de hachage d'une position peut être calculée incrémentalement.

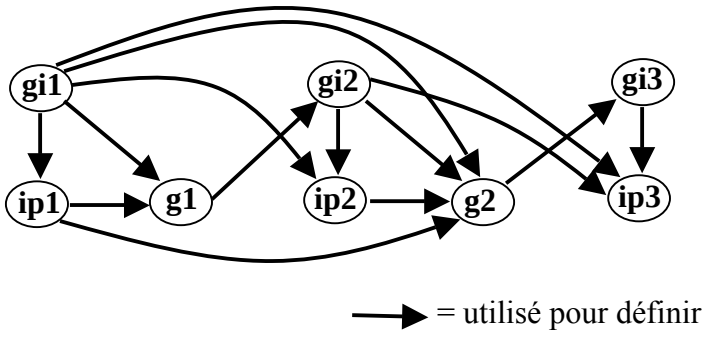
3. La Recherche Abstraite de Preuves

La recherche abstraite de preuve est un algorithme qui démontre efficacement des théorèmes dans les jeux. Il permet des gains de temps appréciables sur l'Alpha-Béta avec toutes ses optimisations, de plus il renvoie toujours des résultats exacts, ce que ne garantit pas l'Alpha-

Béta dans le cas des jeux où les coups sont sélectionnés à chaque nœud. La recherche de preuve abstraite permet de sélectionner les coups avec confiance.

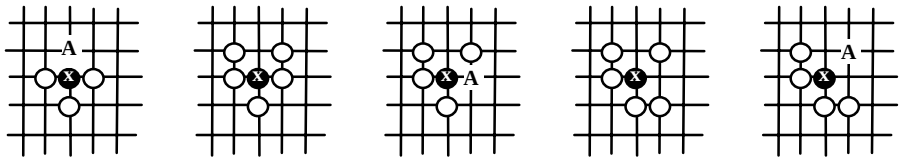
3.1. Les Ensembles de Coups Abstraits

Les coups qui permettent de changer l'issue d'une recherche peuvent être facilement trouvés lorsque le but est presque atteint. Toutefois, lorsque le but n'est plus à un ou deux coups, cela devient plus difficile. Nous traiterons dans cette section de la sélection de l'ensemble complet des coups possibles utiles à examiner lorsque le but ne peut pas être directement atteint. Par exemple, si une chaîne de pierres au jeu de Go peut être capturée en 5 coups, nous voulons trouver tous les coups abstraits qui sont susceptibles d'empêcher cette chaîne d'être capturée. Un coup abstrait est un coup qui est défini en utilisant des propriétés abstraites des chaînes ou du damier comme par exemple la liberté d'une chaîne.
Nous utiliserons des noms pour les différents états de jeux. Le nom des jeux est suivi par un nombre qui indique le nombre minimal de coup que l'attaquant doit jouer pour atteindre le but. Un jeu qui peut être gagné si l'attaquant joue est appelé 'gi'. Un jeu ou le défenseur doit jouer un coup pour ne pas être battu est appelé 'ip'. Un jeu gagné pour l'attaquant est appelé 'g. Un jeu est toujours associé à un joueur et dans le cas des jeux 'gi' et 'ip' à des coups de ce joueur.



Les dépendances entre jeux

Cette figure donne les dépendances entre les différents jeux. Un jeu peut être défini en utilisant les jeux d'indice inférieurs. Par exemple le jeu g1 pour Blanc est défini par : le jeu est ip1 pour Noir, et tous les coups ip1 amènent à des jeux gi1 pour Blanc.



Exemples de jeux

La figure ci dessus donne des exemples de jeux pour le jeu de la prise. Un exemple de la façon dont les coups abstraits pour un jeu sont trouvés est par exemple le passage des jeux gi aux jeux ip : le seul coup qui peut modifier une intersection vide au Go est de jouer dessus. Toutes les intersections vides utilisées pour définir un jeu gi, sont donc des coups possibles du jeu ip associé. Un jeu ip est associé à l'ensemble des coups qui peuvent prévenir un jeu gi.
Ces ensembles abstraits de coups sont trouvés automatiquement par Introspect, mais le code engendré est très volumineux. Il serait intéressant de développer des outils permettant de définir plus simplement (avec des programmes plus concis) ces ensembles.

3.2. Résultats Expérimentaux

Les test comparatifs de l'Alpha-Béta et de la recherche de preuve abstraite ont été fait sur des ensembles de problèmes standards [Kano 1985a,b,1987] sur un ordinateur muni d'un processeur K6-2 450 MHz. Prevent-ip3-1s-10000N correspond à l'Alpha-Béta utilisant toutes ses optimisations, stoppé au bout d'une seconde ou de 10000 nœuds. La fonction de sélection des coups est la même que celle de la recherche abstraite de preuve à l'exception près que les définitions de jeux ne sont pas utilisées pour sélectionner encore plus les coups. Prevent-ip3-1s revient au même sans la barrière des 10000 nœuds. ip3-1s-10000N est l'algorithme de recherche abstraite de preuves.

Algorithm	Total time	Number of nodes	% of problems
Preventip3-1s-10000N	19.79	109117	99.12%
Preventip3-1s	19.79	109117	99.12%
ip3-1s-10000N	11.82	10340	99.12%

Table 1. Résultats pour ggv1

Algorithm	Total time	Number of nodes	% of problems
Preventip3-1s-10000N	113.20	836387	78.47%
Preventip3-1s	118.60	870938	77.78%
ip3-1s-10000N	34.13	42382	88.19%

Table 2. Résultats pour ggv2

Algorithm	Total time	Number of nodes	% of problems
Preventip3-1s-10000N	65.61	449987	65.28%
Preventip3-1s	74.25	483390	65.28%
ip3-1s-10000N	21.13	27283	73.61%

Table 3. Résultats pour ggv3

Des tests ont aussi été effectués pour voir l'évolution de l'algorithme quand plus de temps lui était donné. Les gains sont alors encore plus grands par rapport à l'Alpha-Béta.

Algorithm	Total time	Number of nodes	% of problems
Preventip3-10s-100000N	635.20	4607171	79.17%
ip3-10s-100000N	63.57	81302	90.28%

Table 4. Résultats pour ggv2 avec plus de temps et de nœuds

Algorithm	Total time	Number of nodes	% of problems
Preventip3-10s-100000N	726.40	4319840	70.83%
ip3-10s-100000N	23.97	33936	73.61%

Table 5. Résultats pour ggv3 avec plus de temps et de nœuds

Des tests ont aussi été effectués pour tester l'intérêt de l'heuristique du coup nul. Ces tests se révèlent concluants puisque le temps de recherche est inférieur. Toutefois l'utilisation de l'heuristique du coup nul enlève la propriété de preuve à l'algorithme. Les résultats peuvent alors être faux. En pratique, d'après les résultats expérimentaux, cela n'a toutefois pas l'air de diminuer ses performances.

Algorithm	Book	Total time	nodes	%
Preventip3-1s-10000N-NM	ggv1	13.34	69582	98.25%
Preventip3-1s-10000N-NM	ggv2	66.55	518398	77.08%
Preventip3-1s-10000N-NM	ggv3	30.50	230724	65.28%
ip3-1s-10000N-NM	ggv1	10.58	9401	99.12%
ip3-1s-10000N-NM	ggv2	31.57	39220	88.89%
ip3-1s-10000N-NM	ggv3	16.93	20902	73.61%

Table 6. Results with null move forward pruning

3.3. Conclusion sur la recherche abstraite de preuve

La recherche abstraite de preuve a deux grands avantages sur l'Alpha-Béta classique : ses résultats sont fiables, et il sont calculés plus rapidement. Avec les mêmes contraintes de temps, elle résout plus de problèmes, et l'algorithme réagit mieux que l'Alpha-Béta à l'augmentation des capacités CPU. Cet algorithme marche pour un grand nombre de jeux. Les seuls inconvénients étant de définir une méthode pour trouver les ensembles de coups abstraits associés à chacun des jeux.

4. L'Elargissement Itératif

4.1. L'algorithme

La recherche sélective consiste à ne regarder qu'une partie des coups possibles. Un problème des algorithmes de type Alpha-Béta est l'ordonnancement des coups : on veut essayer en priorité les coups qui simplifient la situation et qui marchent souvent.
L'élargissement itératif consiste à ne considérer que les coups simples pour une première recherche, et s 'ils ne marchent pas envisager des coups moins standards. Pour cela, on définit des ensembles de coups abstraits : $S1 \subset S2 \subset \ldots \subset Sn$. L'élargissement itératif consiste à faire une recherche avec approfondissement itératif pour S1.
Si elle échoue, recommencer avec S2, ainsi de suite jusqu'à Sn, ou jusqu'à ce que le temps imparti soit écoulé.
Au niveaux ET de l'arbre de preuve, il est naturel de définir les ensembles de coups abstraits à partir des définitions de jeux. Les coups S1 aux nœuds ET (appelés AND1) seront donc les

coups ip1 et ip2, alors que les coups de S2 (appelés AND2) seront les coups ip1, ip2 et ip3. Au nœuds OU, S1 (OR1) sera l'ensemble des libertés de la chaîne à capturer et S2 (OR2) tous les coups intéressants y compris les libertés.

L'élargissement des coups peut se faire dans différents ordres, nous avons testé ceux-ci :

- OR2-2AND2-2: C'est l'algorithme original sans élargissement itératif. Les coups OR2 sont utilisés au nœuds OU et les coups AND2 aux nœuds ET.
- OR1-2AND2-2: L'algorithme commence avec une recherche utilisant les ensembles OR1 et AND2 , et si la recherche échoue, il en effectue une autre avec les ensembles OR2 et AND2.
- OR2-2AND1-2: L'algorithme commence avec une recherche utilisant les ensembles OR2 et AND1 , et si la recherche échoue, il en effectue une autre avec les ensembles OR2 et AND2.
- AND1-2OR1-2: OR1, AND1 => OR1, AND2 => OR2, AND2.
- OR1-2AND1-2: OR1, AND1 => OR2, AND1 => OR2, AND2.
- ORAND1-2: OR1, AND1 => OR2, AND2.
- OR1-2ANDOR1-2: OR1, AND1 => OR2, AND1 => OR1, AND2 => OR2, AND2.
- ORAND1-2AND1-2: OR1, AND1 => OR1, AND2 => OR12, AND1 => OR2, AND2.

4.2. Résultat Expérimentaux

Les expériences ont été effectuées sur un Pentium 266 MHz.
On peut voir qu'un algorithme général et indépendant du jeu permet des gains substantiels: c'est le OR2-2AND1-2. Il ne dépend que des définitions des jeux ip pour l'élargissement, ce qui est une heuristique très générale. Le temps de recherche est pratiquement divisé par deux par rapport à l'algorithme sans élargissement.

Algorithm	Time	Nodes	Problem
OR2-2AND2-2	18.15	4809	99.12%
OR1-2AND2-2	17.67	2667	99.12%
OR2-2AND1-2	12.81	4291	99.12%
AND1-2OR1-2	12.26	2576	99.12%
OR1-2AND1-2	12.38	3044	99.12%
ORAND1-2	12.11	2730	99.12%
OR1-2ANDOR1-2	12.31	2913	99.12%
ORAND1-2AND1-2	12.13	2587	99.12%

Table 7. Résultats pour ggv1

Algorithm	Time	Nodes	Problems
OR2-2AND2-2	62.96	30182	86.81%
OR1-2AND2-2	47.99	19096	86.11%
OR2-2AND1-2	32.62	28008	86.81%
AND1-2OR1-2	39.74	19721	86.11%
OR1-2AND1-2	37.15	24244	87.50%
ORAND1-2	39.57	19566	87.50%
OR1-2ANDOR1-2	35.99	23450	87.50%
ORAND1-2AND1-2	45.85	19544	85.42%

Table 8. Résultats pour ggv2

Algorithm	Time	Nodes	% Problems
OR2-2AND2-2	41.43	21226	78.67%
OR1-2AND2-2	30.03	15526	77.33%
OR2-2AND1-2	23.70	22647	81.33%
AND1-2OR1-2	23.78	15073	77.33%
OR1-2AND1-2	20.68	16281	81.33%
ORAND1-2	25.11	13206	78.67%
OR1-2ANDOR1-2	21.85	18106	80.00%
ORAND1-2AND1-2	32.74	13844	74.67%

Table 9. Résultats pour ggv3

5. Conclusion

L'optimisation de l'alpha-béta est encore un sujet de recherche active. Notamment en ce qui concerne la recherche sélective. Nous avons décrit les optimisations usuelles de l'alpha-béta ainsi que deux optimisations très efficaces pour les jeux a but simple et avec un grand facteur de branchement. Les évolutions futures de l'Alpha-Béta peuvent venir de plusieurs horizons : l'ajout (automatique ?) de connaissances, une sélectivité accrue, une plus grande rapidité pour les optimisations déjà existantes, l'utilisation d'informations incertaines et probabilistes, ainsi qu'une possible liaison avec des algorithmes de recherche en meilleur d'abord. Enfin de nombreuses autres optimisations sont sans doute possibles, et plus particulièrement dans les jeux avec un grand nombre de coups possibles comme le Go ou le Phutball. De plus l'utilisation de techniques proches de celles de l'alpha-béta peuvent être à la source d'idées dans des domaines connexes comme la satisfaction de contraintes ainsi que l'a montré la réutilisation d'Iterative Broadening [Ginsberg 1992] par Meseguer et Walsh [Meseguer et Walsh 1998] pour la satisfaction de contraintes.

6. Bibliographie

[Breuker 99] Breuker T.: *Memory versus Search in Games*. PhD thesis, Maastricht. 1999.

[Cazenave 98] Cazenave T.: *Metaprogramming Forced Moves*. Proceedings ECAI98 (ed. H. Prade), pp. 645-649. John Wiley & Sons Ltd., Chichester, England. ISBN 0-471-98431-0. 1998.

[Cazenave 99] Cazenave T.: *Intelligence Artificielle: Une Approche Ludique*. Notes de cours. 1999.

[Cazenave1 00-1] Cazenave T.: *Generating Search Knowledge in a Class of Games*. submitted. http://www.ai.univ-paris8.fr/~cazenave/papers.html. 2000.

[Cazenave 00-2] Cazenave T.: *Iterative Widening*. Workshop of the 2000 computer Olympiad, Londres. 2000.

[Cazenave 01] Cazenave T.: *Abstract Proof Search*. Proceedings of the Computer and Games 2000 conference, publiée en LNCS, 2001.

[Ginsberg 92] Ginsberg M. L., Harvey W. D. : *Iterative Broadening*. Artificial Intelligence 55 (2-3), pp. 367-383. 1992.

[Ginsberg 97] Ginsberg M.: *Partition Search*. AAAI 97.

[Kano 85] Kano Y.: *Graded Go Problems For Beginners*. Volume One. The Nihon Ki-in. ISBN 4-8182-0228-2 C2376. 1985.

[Kano 85] Kano Y.: *Graded Go Problems For Beginners*. Volume Two. The Nihon Ki-in. ISBN 4-906574-47-5. 1985.

[Kano 87] Kano Y.: *Graded Go Problems For Beginners*. Volume Three. The Nihon Ki-in. ISBN 4-8182-0230-4. 1987.

[Marsland 00] Marsland T. A., Björnsson Y.: *From Minimax to Manhattan*. Games in AI Research, pp. 5-17. Edited by H.J. van den Herik and H. Iida, Universiteit Maastricht. ISBN 90-621-6416-1. 2000.

[Meseguer 98] Meseguer P., Walsh T. : *Interleaved and Discrepancy Based Search*. Proceedings ECAI98 (ed. H. Prade). John Wiley & Sons Ltd., Chichester, England. ISBN 0-471-98431-0. 1998.

[Schaeffer 00] Schaeffer J.: *Search Ideas in Chinook*. Games in AI Research. Edited by H.J. van den Herik and H. Iida, Universiteit Maastricht. ISBN 90-621-6416-1. 2000.

[Thomsen 01] Thomsen T.: *Lambda-search in game trees – with application to go*. CG 2000. LNCS, 2001.

Gestion des erreurs dans une interface pédagogique

Groupe COMBIEN
Duma J.[1], Giroire H.[2], Le Calvez F.[3], Tisseau G.[2], Urtasun M.[3]

Résumé : Une interface pédagogique de résolution de problèmes doit permettre à un élève utilisateur de formuler une solution à un problème qui lui est posé, mais surtout d'exploiter au mieux cette activité pour augmenter ses connaissances et ses capacités. Dans une telle interface la gestion des erreurs commises par l'élève ne concerne pas uniquement l'aspect ergonomique et fonctionnel, comme dans toute interface, mais relève en premier lieu d'une décision pédagogique. Il faut définir quelles erreurs seront rendues impossibles à commettre de par la structure même de l'interface, et comment réagir pédagogiquement aux autres erreurs jugées « instructives ». Nous affirmons que la conception d'une interface pédagogique doit reposer sur une analyse préalable des erreurs possibles et nous proposons une méthode de conception fondée sur ce principe. Nous illustrons cette méthode dans le contexte d'interfaces pour résoudre des exercices de dénombrement. Nous avons répertorié et catégorisé les erreurs possibles, ce qui n'est pratiquement réalisable que si l'on se fixe d'abord un cadre précis : une classe de problèmes, une méthode de résolution et un modèle conceptuel rendant la résolution équivalente à l'édition d'un objet structuré. Cela permet d'identifier les concepts autour desquels doit être organisée l'interface, et qui vont correspondre aux boutons de validation proposés à l'élève. Nous montrons ensuite comment implémenter la gestion des erreurs à l'aide d'un *agent pédagogue*.

Mots clés : interfaces pédagogiques, erreurs, méthode de conception, EIAO, dénombrement.

1. Introduction

Le projet "Combien ?" a pour but de définir une méthodologie de conception de différents composants d'un EIAO (Environnement Interactif d'Apprentissage avec Ordinateur). Pour valider nos réflexions, nous réalisons un EIAO pour l'apprentissage des dénombrements. Les exercices correspondant sont de la forme : "Etant donnés des ensembles servant de référentiels, compter dans un certain univers les éléments vérifiant des contraintes de

[1] Lycée Jacquard, Paris

[2] LIP6, Pôle IA, SYSDEF, Université Pierre et Marie Curie, Paris VI

[3] CRIP5, SBC, Université René Descartes, ParisV

sélection". Nous avons défini les fondements mathématiques d'une méthode de résolution (la méthode constructive) adaptée aux conceptions usuelles des élèves et permettant d'accéder à la théorie mathématique du domaine [Tisseau et al. 96], [Le Calvez et al.97]. Nous avons défini une classification des problèmes du domaine et les schémas de résolution associés aux différentes classes. Nous avons introduit pour chaque classe une "machine à construire une solution". Chaque machine se présente pour l'élève sous forme d'une interface pédagogique qui conduit l'élève à construire la solution d'un exercice de la classe considérée.

Dans une interface pédagogique la gestion des erreurs est très importante pour optimiser l'efficacité pédagogique de l'interface. Contrairement à une interface usuelle, il faut pouvoir laisser à l'élève la possibilité d'exprimer sa solution et donc de commettre des erreurs. Généralement on essaie dans une interface de canaliser l'activité de l'intervenant pour qu'il effectue le moins d'erreurs possibles [Choplin and al. 98]. Il est en effet techniquement possible de concevoir l'interface pour que l'utilisateur ne puisse pas commettre certaines erreurs. En effet, les erreurs sont provoquées par des actions de l'utilisateur (cliquer sur un bouton, choisir un item de menu) et il est parfois possible de prévoir avant même que l'action soit exécutée si elle pourra provoquer une erreur ou non. Dans ce cas, on peut faire en sorte que les possibilités d'action offertes à l'utilisateur ne puissent pas provoquer d'erreur, et cela dynamiquement en fonction de la situation (rendre indisponibles les boutons qui provoqueraient des erreurs ou ne faire figurer dans un menu que les items qui ne provoquent pas d'erreur). Les autres erreurs sont détectées sur-le-champ, signalées et éventuellement corrigées. Dans une interface pédagogique il faut quelquefois laisser l'élève s'exprimer même si l'on prévoit une erreur ou si l'on en a détecté une, soit pour qu'il s'en aperçoive tout seul plus tard, soit pour lui montrer qu'il arrive à une impasse et profiter de l'occasion pour lui apprendre quelque chose.

Cependant toutes les erreurs ne sont pas de la même difficulté, ni du même intérêt pédagogique. Une étude sur les différents types d'erreurs doit permettre de décider quelles sont celles qui seront permises à l'élève et celles qu'il ne pourra commettre. D'autre part, les erreurs peuvent être de type différents (par exemple, provoquées par l'utilisation de l'interface ou correspondant à des concepts mathématiques). Dans certains cas on essaiera de proposer à l'élève des possibilités d'interaction où il ne peut pas faire ces erreurs.

Dans cet article, nous inventorions les différentes catégories d'erreurs dans la machine "ConstructionEnsemble". Nous proposons une méthode de conception des interfaces pédagogiques qui s'appuie sur la gestion des erreurs. Nous indiquons ensuite, comment nous réalisons la gestion des erreurs dans cette machine.

2. La machine ConstructionEnsemble

Les machines permettent pour chaque classe de problème de construire un élément de l'ensemble à dénombrer (en suivant la méthode constructive), puis à partir de cette construction de compter le nombre d'éléments de cet ensemble. [Tisseau et al. 00-1] [Tisseau et al. 00-2]

Les exercices de la machine ConstructionEnsemble sont du type : "Soit un ensemble donné (le référentiel), compter le nombre de configurations contenant X éléments de ce référentiel (l'ensemble des configurations est appelé univers), et satisfaisant certaines contraintes. Les contraintes sont de la forme : exactement X1 éléments appartenant au sous ensemble R1 du référentiel, exactement X2 éléments appartenant au sous ensemble R2 du référentiel etc. Les sous ensembles Ri étant tous disjoints deux à deux."

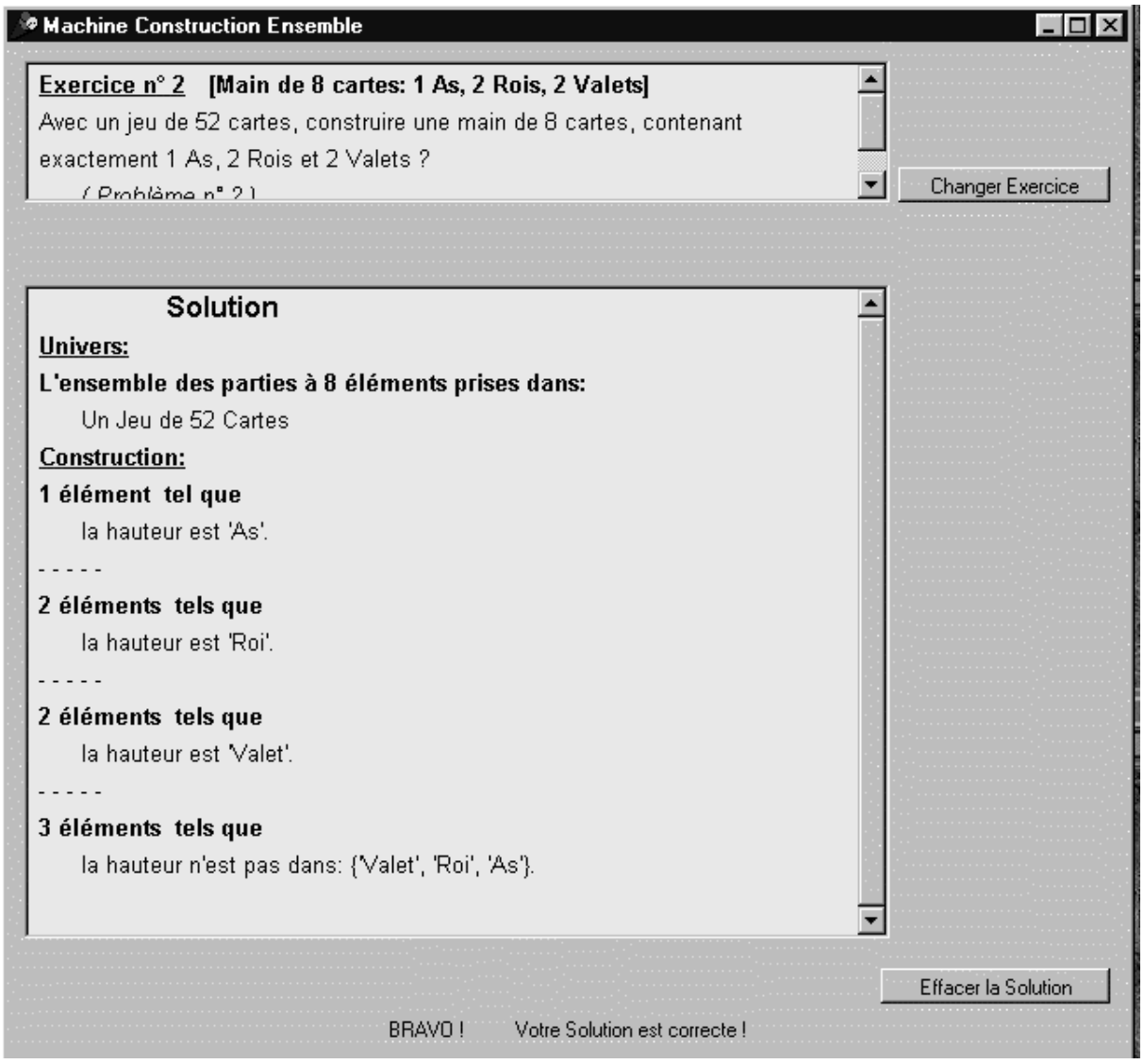


Figure 1

Dans la suite de l'article, l'exercice suivant : "Avec un jeu de 52 cartes, combien y-a-t-il de mains de 8 cartes contenant exactement 1 as, 2 rois et 2 valets ? ", sera utilisé pour illustrer nos propos. Cet énoncé est un énoncé pris dans un livre d'exercices, il signifie : combien y-a-t-il de mains de 8 cartes contenant exactement 1 as, exactement 2 rois et exactement 2 valets et 3 cartes qui ne sont ni as, ni roi, ni valet ?
Le modèle conceptuel que nous avons défini pour représenter le domaine des dénombrements est décrit dans. [Tisseau et al. 00-1] [Tisseau et al. 00-2]. Dans le cas du jeu de 52 cartes, chaque carte est représentée en utilisant deux attributs **couleur** et **hauteur**. La figure 1 nous montre l'aspect de la machine après construction d'un élément de l'univers des mains de 8 cartes.
A partir de cette construction et dans cette machine, le nombre de configurations possibles est le produit des possibilités à chaque choix.
Il faut bien voir que cette image n'est que l'aspect final de la réponse et ne montre pas les différents éléments d'interaction qui ont permis la génération de cette réponse (boutons, menus, etc.). Ceux-ci apparaissent et disparaissent dynamiquement en fonction de l'état d'avancement de la réponse.

3. Les erreurs dans la machine ConstructionEnsemble

Dans toute machine, la première chose que fait l'élève est de choisir l'exercice à résoudre. Celui-ci fait partie de la classe de problèmes associée à la machine. Il y a ensuite deux grandes phases dans le travail de construction : la définition de l'univers et ensuite les différentes étapes qui correspondent aux contraintes de sélection. La figure 2 montre un état de la machine où l'exercice a été choisi et validé, l'univers aussi, ainsi que certaines étapes de la construction. Mais la construction elle-même n'a pas encore été validée. Nous avons dressé un inventaire des erreurs possibles dans ces deux phases pour les catégoriser et décider de leur traitement. Dans la suite de cet article, nous donnons les catégories d'erreurs pour la machine ConstructionEnsemble en les illustrant sur l'exercice de la figure 2.
Dans la définition de l'univers, il faut préciser le référentiel et le nombre d'éléments de chaque élément de l'univers (c'est-à-dire de chaque configuration). Les erreurs possibles portent donc sur un mauvais choix du référentiel ou une erreur dans le nombre d'éléments de la configuration. Cette dernière erreur peut-être de deux sortes : nombre impossible (main de 56 cartes prises dans un jeu de 52 cartes) et nombre incorrect c'est-à-dire ne correspondant pas à l'énoncé du problème (main de 9 cartes alors que l'énoncé de l'exercice parle de mains de 8 cartes).

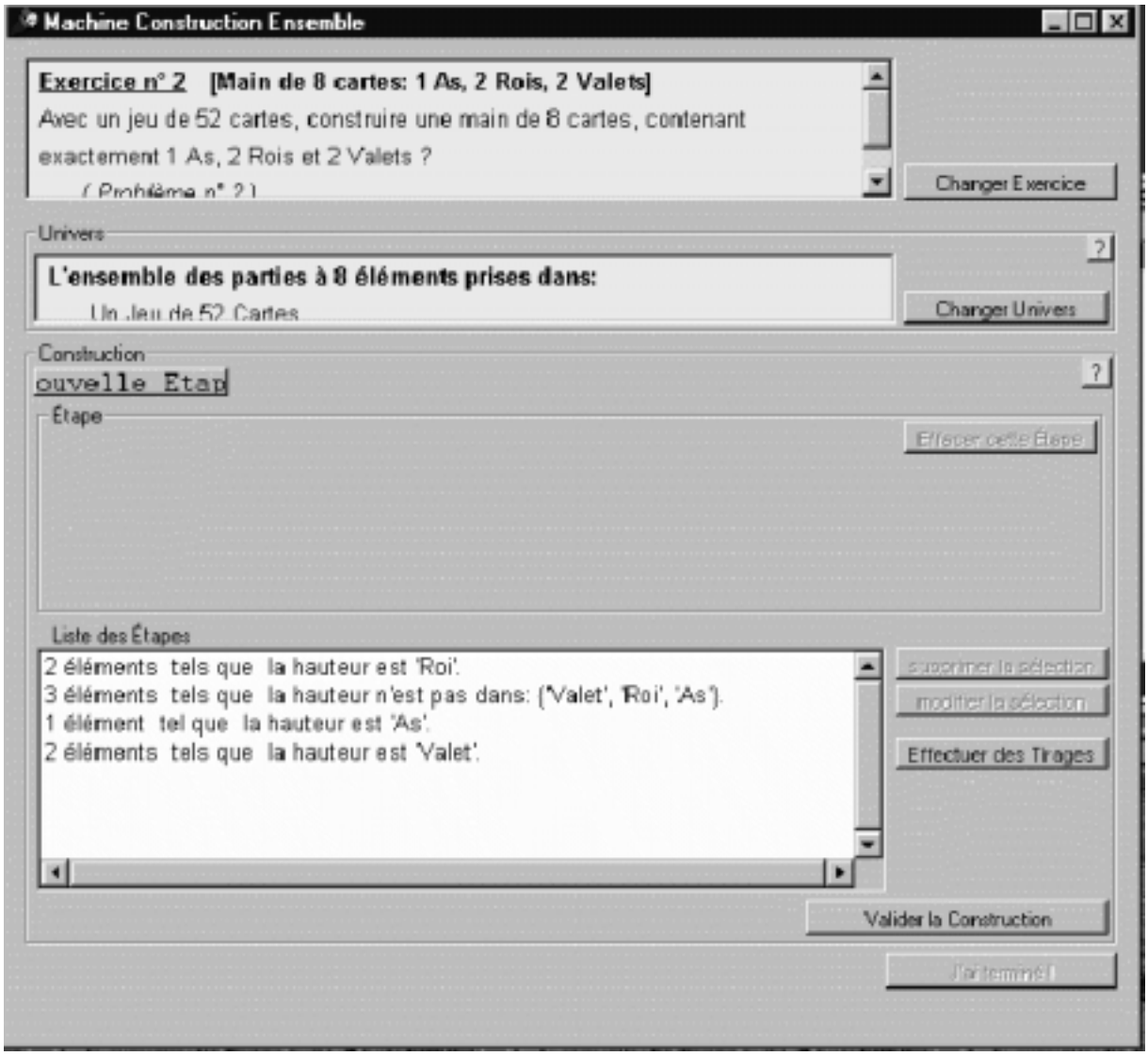


Figure 2

Au niveau de la définition des contraintes dans la solution, il y a à chaque étape définition du nombre d'éléments de la configuration sur lesquels porte la contrainte et la définition du sous ensemble (SE) auquel ils doivent appartenir.

A chaque étape, à propos du nombre des éléments, on retrouve les mêmes sortes d'erreurs que pour le nombre d'éléments dans une configuration : nombre impossible (14 cœurs dans un jeu de 52 cartes) et nombre incorrect (2 as alors que l'énoncé de l'exercice parle de mains contenant exactement 1 as). La définition de SE se fait à l'aide de sélection sur les attributs ("hauteur est roi", ou "hauteur est roi ET couleur n'est pas noire"). A ce niveau les erreurs possibles concernant l'attribut sont : attribut inexistant pour le référentiel de l'exercice (masse pour une carte) ou valeur erronée pour un attribut (hauteur = 2 dans un jeu de 32 cartes). Une première erreurs sur SE est de définir pour SE l'ensemble vide (2 cartes de couleur pique et de couleur rouge). Les autres erreurs sur une seule étape font intervenir des sous-ensembles qui ne font pas partie d'une solution correcte de l'exercice à résoudre. L'élève définit la contrainte

"2 cartes de hauteur dame" alors que l'on attend 2 valets ou 2 rois ou bien il définit la contraintes "2 cartes de couleur rouge" alors que l'on attend 2 cœurs.
Les erreurs que l'on vient de décrire correspondent à une étape isolée. Certaines erreurs apparaissent lorsque l'on considère l'ensemble des étapes.
Il peut y avoir d'une part des erreurs sur la somme des nombres définis à chaque étape. Il y a deux types d'erreurs : des incohérences et des incorrections.

- Les incohérences sont internes à la solution de l'élève, il a défini les mains de 8 cartes pour l'univers et de 10 cartes (ou de 3) à la fin des étapes.
- Les incorrections sont externes par rapport à la solution de l'élève. Sa solution est cohérente mais ne correspond pas à une solution de l'exercice à résoudre. Il a défini les mains de 5 cartes pour l'univers et a bien défini les étapes pour 5 cartes alors que dans l'exercice on demande des mains de 8 cartes.

Il peut y avoir d'autre part des erreurs sur la définition des différents SE. La méthode constructive implique pour cette machine que les SE soient disjoints. L'erreur consiste à avoir des SE non disjoints (le SE défini par "hauteur est as" et celui défini par "couleur est rouge").
Même pour un exercice aussi simple, on constate qu'il existe un grand nombre d'erreurs possibles, mais aussi qu'il apparaît une catégorisation à visée pédagogique possible.

En résumé, nous avons distingué trois catégories d'erreurs :
Des erreurs purement internes, qui sont détectables sans prendre en compte une solution de l'exercice, et ne font pas intervenir la méthode constructive. Nous avons utilisé pour elles les termes "impossible" ou "incohérent". Nous pouvons les considérer comme des erreurs d'inattention.
Des erreurs qui font intervenir la méthode constructive et qui sont internes. C'est l'utilisation de SE non disjoints pour la machine ConstructionEnsemble. Ces erreurs sont délicates à expliquer, car elles correspondent à des conditions uniquement imposées par la méthode, et dont l'élève ne pourra voir la raison d'être que plus tard, au moment de compter. Pour lui, elles peuvent présenter un caractère arbitraire.
Des erreurs dites externes, elles proviennent de la comparaison des solutions correctes et de la solution proposée par l'élève ne comportant pas d'erreurs internes.

Ces catégories (internes, faisant intervenir une méthode de résolution, externes) nous semblent générales pour un EIAO.

4. La définition et la gestion des erreurs

Pour spécifier l'interface une fois que les erreurs ont été répertoriées et catégorisées, il faut

décider quand et comment les détecter, les signaler et les expliquer.

4.1. Cadre conceptuel : structure à éditer

Nous nous plaçons dans le cadre suivant : les interfaces pédagogiques que nous considérons sont des outils offerts à un élève pour exprimer la solution de problèmes d'une classe particulière dans un domaine particulier, cette solution pouvant être formalisée comme instance d'une certaine *structure* arborescente prédéfinie. Cette structure n'est pas entièrement figée : certains nœuds peuvent être polymorphes (leur classe peut être choisie parmi plusieurs classes différentes) ou contenir une valeur multiple de cardinal non spécifié a priori (une liste d'éléments). D'autre part, cette structure est « à dimension humaine » : le nombre de nœuds et de niveaux qu'elle comporte permet l'édition de ses instances par un élève, interactivement, à travers une interface qui permet d'en remplir les champs.
Il faut noter que la théorie d'un domaine d'étude donné ne fournit pas automatiquement une telle structure. Celle-ci est à inventer à chaque fois qu'on a l'intention de créer une interface pédagogique destinée à un certain type d'élèves possédant certaines connaissances et en fonction d'un certain niveau d'approfondissement et d'expertise visé. C'est ce que nous avons dû faire pour les dénombrements.

4.2. Validité d'une solution, erreurs

La validité d'une solution est définie par une conjonction de *conditions de validité* portant sur la structure. Une *erreur* est définie comme étant une violation d'une de ces conditions de validité. On peut ainsi distinguer des *types d'erreur* correspondant aux conditions de validité : une erreur est de tel type si elle viole telle condition.
On peut assigner à chaque type d'erreur un intérêt pédagogique : lorsque l'élève commet une erreur et qu'on la lui signale, quel bénéfice peut-il en retirer du point de vue de la compréhension des concepts du domaine, de l'identification des difficultés et de l'apprentissage de méthodes de résolution ? Suivant les buts pédagogiques visés, une erreur pourra être jugée *instructive* ou non.
Les erreurs non instructives sont considérées comme des diversions hors sujet qui encombrent le chemin vers l'apprentissage visé. Cela ne signifie pas qu'elles sont sans importance théorique, mais simplement qu'elles éloignent trop du but particulier choisi. Il est bon alors que l'interface soit conçue de telle façon que ces erreurs ne puissent pas apparaître. Cela revient à dire que c'est le système lui-même qui doit assurer automatiquement le respect des conditions de validité associées.
Par exemple, si une condition de validité non instructive spécifie qu'un nombre x doit être un nombre entier compris entre 1 et 5, l'interface pourra offrir à l'élève un menu avec les cinq

valeurs 1, 2, 3, 4, 5 pour la saisie de x (plutôt qu'un champ de texte où l'élève édite librement un texte quelconque). Notons que cette configuration de l'interface pour satisfaire les conditions de validité peut être dynamique : si une condition spécifie que x doit être un entier compris entre 1 et y et que l'élève a saisi y, le menu pour x sera mis à jour pour comporter la liste des entiers de 1 à y.

4.3. Aspects dynamiques et interactifs

La saisie de la solution par l'élève est vue comme une tâche d'édition d'une structure arborescente. Avec les structures que nous considérons, cette tâche n'est pas atomique ni immédiate. Elle passe par différentes étapes qui permettent d'engendrer la structure complète petit à petit, incrémentalement. Pour tenir compte de cet aspect, nous avons construit des interfaces qui prennent en compte les erreurs de façon incrémentale. Cela est adapté au type de tâche demandé, mais aussi et surtout, cela permet de profiter le mieux possible des instants où l'élève est dans une situation d'apprentissage optimale, c'est-à-dire au moment où il commet des erreurs.
L'incrémentalité est prise en compte en décomposant le traitement des erreurs en différents sous-traitements, qui eux-mêmes sont exécutés en plusieurs occurrences à différents moments. Ces sous-traitements sont : la *détection* d'une erreur (qui est une procédure interne au système, « silencieuse » car elle n'est pas visible par l'élève), son *signalement* (par exemple l'affichage d'un message pour l'élève : « il y a une erreur ») et son *explication*. Dans notre EIAO, nous voulons pouvoir choisir pour chaque erreur le moment où on la détecte, celui où on la signale et celui où on l'explique. Cela contraste avec d'autres EIAO où le travail de l'élève n'est évalué que lorsqu'il valide la solution finale de l'exercice.
Pour les erreurs "intéressantes", il faut décider **à quel moment elles sont détectées**. Le fait de les détecter n'implique pas d'en faire part à l'élève, c'est un choix pédagogique que de déterminer **le moment du signalement** de cette erreur et donc si l'élève peut ou non continuer, bien qu'ayant commis une erreur. Il faut ensuite décider comment se manifeste le signalement : on peut refuser de valider sa proposition (soit en ayant un bouton qui reste grisé, soit en indiquant que la proposition n'est pas valide), ou bien on peut indiquer qu'il y a une erreur sans autre explication et le laisser continuer s'il le désire pour qu'il découvre quelle est cette erreur. Enfin, il faut décider à quel **moment on lui donne une explication** sur ses erreurs et quel type d'explication donner.

A partir de ces réflexions nous avons défini une méthode de conception de nos interfaces pédagogiques, qui nous semble générale.

5. Une méthode conception "dirigée par les erreurs"

La méthode de conception "dirigée par les erreurs" d'une interface pédagogique que nous proposons comporte un certain nombre d'étapes décrites ci-dessous:

Définir la structure à éditer
Choisir les éléments validables
Choisir les possibilités de modification données à l'élève
Choisir le moment de signalement des erreurs détectées
Définir les explications et les aides

Figure 3

5.1. Définir la structure à éditer

Nous avons vu qu'une interface pédagogique est un éditeur de structure arborescente. Le travail que nous avons fait précédemment [], nous a permis de définir un modèle du domaine du dénombrement y compris des problèmes et des solutions. A partir de ce modèle nous définissons les structures à éditer pour les différentes machines correspondant aux différentes classes de problèmes. Dans le cas de la machine ConstructionEnsemble, la structure à éditer est celle de la figure 4. L'élève choisit son exercice et construit sa solution.

```
Exercice : un Exercice
Solution : une Solution
        Univers : un UniversParties
                Référentiel : unRéférentiel
                Cardinal : unEntier
        Construction : une ConstructionDePartie
                SousParties : * ConstructionSousPartie
                        Cardinal : unEntier
                        Filtre : unFiltre
                                Propriétés : 0 , 1 ou 2 Propriétés
                                        IdentificateurAttribut
                                        Appartenance/Comparateur/estVrai
                                        Domaine/ValeurCible/IdentificateurPredicat
```

Figure 4

Essayons de voir comment, à partir de la déclaration ci-dessus, on peut faire des choix pédagogiques et ergonomiques qui conduisent à l'interface finale. Le premier choix concerne les éléments à valider.

5.2. Choisir les éléments validables

Nous appelons *élément validable* une partie de la structure arborescente dont l'édition se manifeste à l'écran dans une zone identifiable comportant un bouton « valider » s'appliquant à toute la zone. A l'intérieur de la zone, en cours de saisie, toutes les informations sont modifiables et annulables à volonté. Mais lorsque l'élève clique sur le bouton « valider », il signifie par là qu'il assume tous les choix qu'il a faits dans la zone et celle-ci est alors figée avec les informations saisies (il apparaît cependant éventuellement un bouton « annuler » qui permettra de revenir sur la saisie plus tard).
Le choix des éléments validables relève de la pédagogie choisie, en fonction du domaine et de la méthode de résolution que l'on veut faire utiliser. A partir du travail sur nos interfaces, nous avons relevé quelques critères de choix.

- L'élément à valider correspond à un concept qu'on veut **rendre visible à l'élève**. On rencontre souvent ce problème en pédagogie : tous les concepts de la théorie ne sont pas à mettre au même niveau. Présenter explicitement certains concepts à l'élève trop tôt lui compliquerait les choses et l'empêcherait de se concentrer sur l'essentiel. Ici, par exemple, on souhaite que l'élève retienne qu'une solution est formée d'un univers et d'une construction, celle-ci étant une liste d'étapes. Pour chacun de ces concepts, on doit introduire un mot qui le désigne (pour l'élève, le mot étape correspond au concept de SousPartie de la structure), une définition et des exemples.

Les concepts non validables ne sont pas forcément explicités à l'élève. Il faut alors que l'interface les rende " évidents ". Pour cela, on utilisera le langage naturel, une métaphore, des structures graphiques etc.. Ainsi, dans la structure ci-dessus, l'élève ne valide pas chaque élément de la SousPartie. L'interface lui fait remplir une phrase à trous à l'aide de menus.

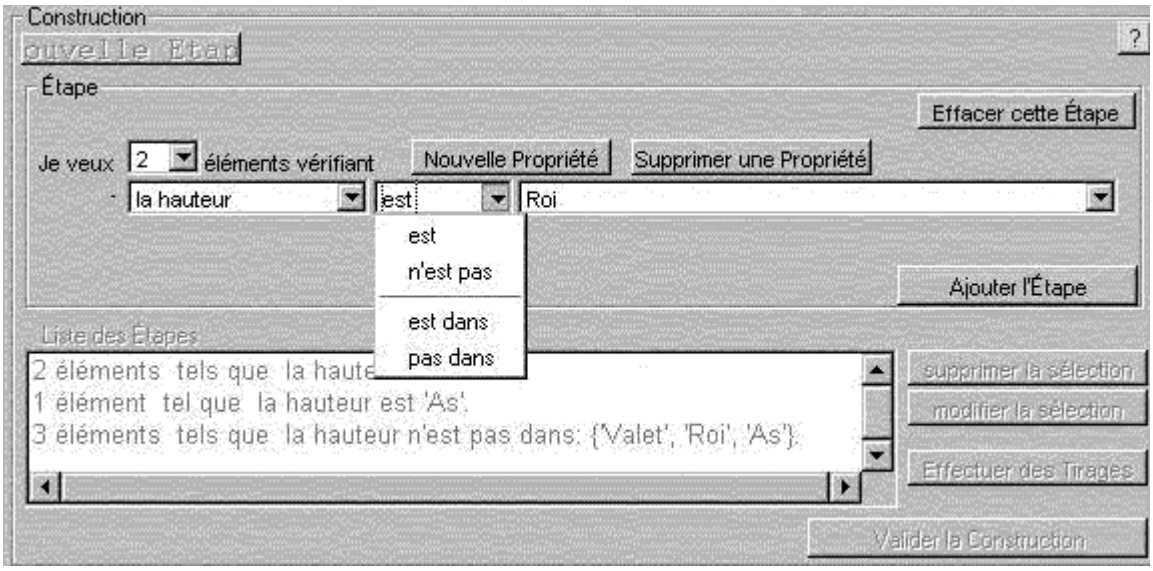


Figure 5

- La validation de l'élément peut donner lieu à des **signalements d'erreurs instructives**. Ce sont les erreurs qui permettent à l'élève d'avancer dans son apprentissage. Pour chaque validation, on établit la liste des erreurs possibles et pour chacune d'elles les informations à apporter à l'élève. On a choisi ici de ne pas signaler d'erreur pendant la saisie dans la zone. On attend que l'élève ait validé sa réponse. Cela comporte un aspect sécurisant pour lui : cela signifie qu'on ne l'observe pas pendant son activité de saisie et qu'on reconnaît qu'il est normal que celle-ci puisse comporter des essais au brouillon, des fautes de frappe et d'inattention. Par contraste, cela souligne l'importance accordée au concept validé.
- Le **nombre d'éléments auxiliaires** à obtenir pour autoriser la **validation ne doit pas être " trop grand ".** Le découpage en parties validables est un garde-fou qui permet à l'élève de ne pas s'enferrer dans une succession d'erreurs qui rendraient le diagnostic et les corrections trop compliqués ou même infaisables. Par exemple, nous avons choisi de faire valider l'univers (ensemble des mains de 8 cartes) qui est composé d'un référentiel et d'un cardinal de façon à ce que l'élève ne puisse pas continuer s'il n'a pas choisi les bons ensembles de départ. Le parcours est balisé par les validations. Dans notre machine, par rapport à la structure décrite figure 4, on a choisi de ne faire valider à l'élève que : le choix de l'exercice, celui de l'univers, de la construction et de chaque sous-partie.

5.3. Choisir les possibilités de modification données à l'élève

Il y a plusieurs façons de permettre à l'élève de revenir sur ce qu'il a fait :

- Avant validation, simplement en saisissant une nouvelle valeur (l'annulation d'un seul item) ou en annulant toutes les informations en cours de saisie dans le contexte à valider.
- Après validation l'élève peut vouloir modifier un élément qui a été validé. Dans ce cas-là, se pose le problème du contexte à restituer à l'élève. Suivant les cas, on peut pré-remplir

l'éditeur avec les informations venant de l'élément annulé ou on peut présenter un éditeur vide. C'est un choix d'ergonomie. Cela peut aussi être un choix pédagogique fondé sur le coût d'une annulation. Si annuler demande un gros travail pour saisir à nouveau un élément, l'élève y réfléchira à deux fois la prochaine fois qu'il devra valider. L'annulation d'un élément annule tout le sous-arbre issu de cet élément.
Actuellement, voici les choix de correction explicite qui ont été faits :

Changer exercice (après validation, efface l'exercice)
Changer Univers (après validation, efface l'univers)
Effacer étape (avant validation)
Modifier une étape de la liste des étapes (après validation, permet de l'éditer)
Modifier construction (permet de l'éditer)

Figure 6

5.4. Choisir le moment de signalement des erreurs détectées

Contrairement à ce qui se passe dans d'autres interfaces, n'attendons pas que la solution de l'élève soit complète pour l'analyser et pour signaler des erreurs. Les erreurs sont détectées lors des demandes de validation au cours de la construction de la solution et selon les concepts à valider retenus. Pour des raisons pédagogiques, le moment de signalement des erreurs peut être repoussé après le moment de détection. Par exemple, on pourrait attendre la validation finale pour signaler les erreurs externes, c'est-à-dire celles qui correspondent à une solution d'un exercice de la classe de la machine mais qui ne correspondent pas à une solution de l'exercice à résoudre.
Pour mettre en œuvre ce retard dans le signalement de l'erreur, il faut mémoriser les erreurs en tenant compte des annulations et modifications. Cela nécessite un mécanisme de mise à jour (du type historique, ou pile).
Certaines erreurs peuvent être signalées mais pas expliquées sur le moment. Leur explication peut être reportée à la validation finale. Lors de la détection, on voit juste un message : " Attention, il y a une erreur. Vous pouvez essayer de la corriger, sinon vous pouvez continuer mais vous devrez attendre la validation finale pour avoir l'explication détaillée".
Ce problème de détection et de signalement des erreurs est très différent des principes d'ergonomie qui disent : il faut tout contrôler le plus possible pour éviter l'apparition d'erreurs et s'il y en a elles doivent être détectées et signalées le plus tôt possible, avec des explications. Ici on peut introduire volontairement la possibilité pour l'élève de commettre des erreurs, on peut décider de ne pas les signaler tout de suite ou de ne pas les expliquer

complètement.
Actuellement, dans la machine ConstructionEnsemble, les erreurs sont signalées dès qu'elles sont détectées.

5.5. Explications et aides

Lorsqu'une erreur est signalée, on peut décider de ne pas expliquer tout de suite à l'élève pourquoi il y a une erreur. Il peut être plus pédagogique de lui laisser chercher la cause de l'erreur. Cependant il ne faut pas qu'un élève se sente bloqué ou qu'il ne comprenne pas. C'est pourquoi nous avons décidé de donner à l'élève la possibilité d'avoir des explications sur son erreur en interrogeant le système (par un bouton).
D'autre part, l'élève ne doit jamais être bloqué dans l'utilisation d'une machine. Nous avons donc prévu une aide contextuelle afin de lui expliquer à tout instant ce que le système attend de lui.
Il nous a semblé alors logique de rassembler aide contextuelle dans le maniement de la machine et explication sur l'erreur qui vient d'être signalée dans une même boîte de dialogue comportant plusieurs onglets. De la même façon, l'élève a besoin d'aide contextuelle sur le dénombrement. Cette boîte de dialogue comporte aussi un onglet dénombrement qui lui présente la partie du cours dont il a besoin (le cours correspondant à la classe de la machine).
Ainsi à tout moment de l'utilisation de notre interface, l'élève peut interroger au moyen du bouton  le système et obtenir soit le cours, soit l'explication d'utilisation de la machine, soit l'explication de l'erreur qui vient de lui être signalée. Dans nos machines, à chaque contexte de validation est associé un bouton  .
Associé à chaque machine, il y aura donc un cours "distribué", "réparti" et présenté par nécessité et opportunité. De cette façon, une même notion pourra être présentée de façon différente selon le contexte.
Les figures 7 et 8 montrent au moment de la validation de l'univers, les aides proposées à l'élève respectivement du point de vue du cours et du point de vue de l'interface.

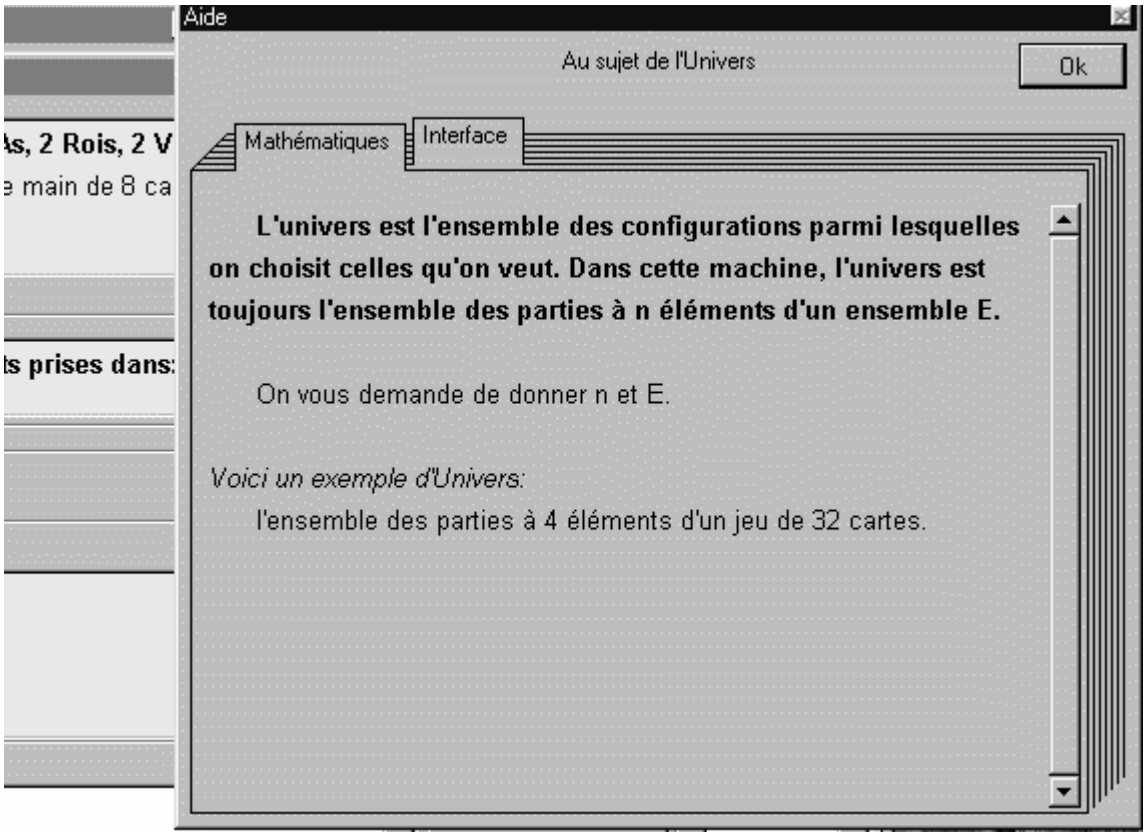


Figure7

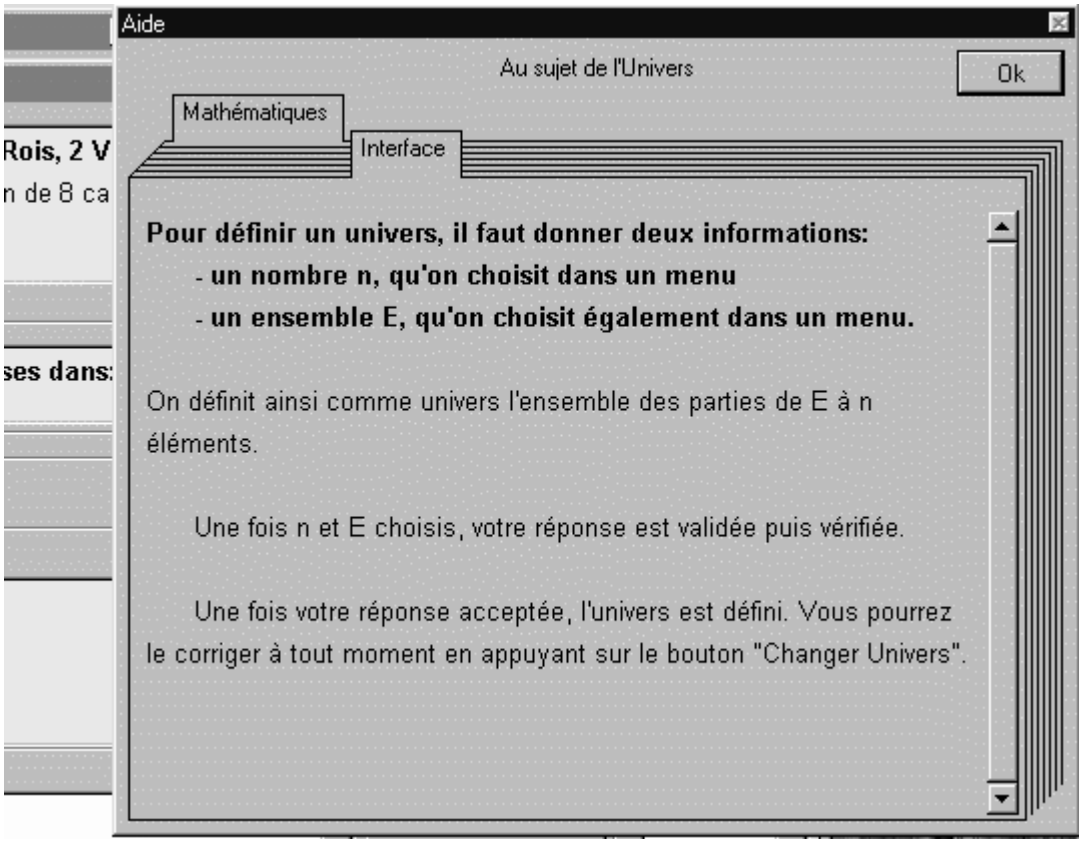


Figure 8

6. Vers une conception-implémentation

Nous abordons la phase de réalisation de la prise en compte des erreurs. Nos avons choisi les concepts à valider. A chacun d'entre eux est associée une condition de validité. La condition de validité s'exprime par une conjonction de conditions. Quand l'élève clique sur le bouton "valider" associé au concept, la condition de validité correspondante est examinée. Si elle n'est pas vérifiée, cela veut dire qu'il y a une ou plusieurs erreurs. Celles-ci sont alors mémorisées sous la forme d'une liste d'objets. Chacune des erreurs contient les informations nécessaires à son explication.

Pour réaliser ceci, nous avons associé à chaque machine un agent pédagogue. Son rôle est de gérer les erreurs. Le pédagogue demande à la partie « experte » du système le test des différents éléments de la condition de validité. Il détecte si il y a une erreur ou pas, et dans le cas où il y en a il va les analyser, puis prendre les décisions de réaction vis à vis de ces erreurs. Ces décisions concernent le moment du signalement, les explications et la réponse à la demande de validation de l'élève. Le pédagogue transmet alors ses décisions à la machine qui va les matérialiser pour l'élève.

Tout pédagogue est représenté par une structure qui contient la machine à laquelle il est associé, la session et les diagnostics. La session comporte toutes les informations liées à l'exercice à résoudre (en particulier la solution du système) et s'enrichit de la solution de l'élève au fur et à mesure de sa construction au moyen de la machine. Il y a autant de diagnostics que de conditions de validité. Chacun d'entre eux est représenté par un booléen (valeur de la condition de validité) et une liste d'erreurs éventuellement vide. La figure 9 montre la structure du pédagogue ConstructionEnsemble associée à la machine ConstructionEnsemble (cf. § 2).

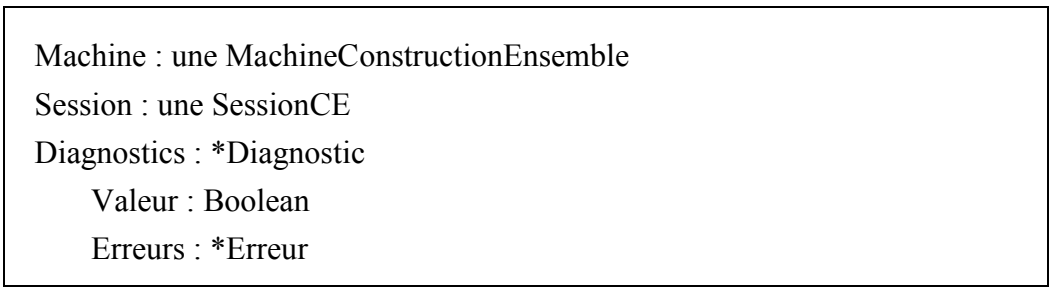


Figure 9

Dans cette architecture, le pédagogue gère les erreurs en répondant aux demandes de validation de l'élève que la machine lui transmet. Pour cela, il interroge le domaine et prend des décisions qu'il communique à la machine. La gestion des erreurs est donc bien programmée à part ce qui nous donne une grande souplesse qui permettra de personnaliser cette gestion en fonction de l'élève.

7. Conclusion

Dans le projet Combien?, nous cherchons à formaliser au maximum nos réflexions pour qu'elles soient ensuite utilisables dans d'autres contextes. Nous avons ainsi introduit une méthode de conception d'interfaces pédagogiques à partir des erreurs. L'utilisation de la conception objet nous permet d'implémenter nos réflexions de façon très déclarative. Nous avons ainsi un cadre général de construction des systèmes "machines-pédagogues" qui nous permet de faire des expérimentations de nos différentes réflexions. Nous définissons actuellement à partir de cette méthode la machine ConstructionListe.

8. Bibliographie

[Choplin and al. 98] Hugues CHOPLIN, Arnaud GALISSON, Sarah LEMARCHAND : *Hypermédias et pédagogie : comment promouvoir l'activité de l'élève ?* - Actes du quatrième colloque Hypermédias et apprentissages, pp 87-98, J.F.Rouet et B. de la Passardière eds, 1998.

[Le Calvez et al. 97] Le Calvez F., Urtasun M. , Tisseau G., Giroire H., Duma J. *Les machines à construire : des interfaces pour apprendre une méthode constructive de dénombrement*. - Actes des 5èmes Journées francophones EIAO, pp 49-60, M.Baron, P. Mendelsohn, J.F. Nicaud eds, Hermès, 1997.

[Tisseau et al. 00-1] Tisseau G., Giroire H., Le Calvez F., Urtasun M., Duma J., *Principes de conception d'un système pour enseigner la résolution des problèmes par la modélisation*. - RFIA'2000, pp.121-130, Paris, 2000.

[Tisseau et al. 00-2] Tisseau G., Giroire H., Le Calvez F., Urtasun M., and Duma J., *Design principles for a system to teach problem solving by modelling.* Lecture Notes in Computer Science N° 1839, ITS'2000, Springer-Verlag, Montréal, 2000.

Contrôle du raisonnement hypothétique en environnement incertain et incomplet: le jeu de l'ascenseur

Fabrice KOCIK
doctorant de l'université Paris 6
directeur de thèse: Jacques Pitrat

Résumé. Les méthodes arborescentes classiques basées sur le min-max sont souvent très coûteuses et aboutissent à une combinatoire importante. L'alpha-beta et ses optimisations résolvent en partie uniquement ce problème. Dans tous les cas, les explications fournies par les systèmes utilisant de tels algorithmes sont difficilement compréhensibles et reposent sur des hypothèses concernant la stratégie des autres joueurs. Pour résoudre tous ces problèmes, il est nécessaire d'avoir un système qui raisonne au niveau meta d'une façon plus élaborée. Nous proposons ici d'améliorer les techniques arborescentes avec la prise en compte dans la recherche arborescente de connaissances plus abstraites, d'événements incertains probabilisés, et aussi avec l'utilisation de connaissances imparfaites pour la fonction d'évaluation basées sur les notions de nécessité et de plausibilité. Une grande importance dans le contrôle sera donné aux connaissances a priori.
Mot clés: Arbre de recherche, probabilité, contrôle, système expert, incertain.

1. Introduction

Deux problèmes essentiellement se posent à la réalisation d'un système général de résolution de jeux. Il est possible de s'attacher à traiter l'apprentissage de connaissances, a priori à partir des règles du jeu ou a posteriori par une étude des parties déjà jouées. Le second domaine de recherche est l'utilisation des connaissances, qu'elles soient apprises par le système ou données par un expert. L'apprentissage est nécessaire pour la réalisation d'un système autonome. Mais à quoi serviraient les connaissances apprises si celui-ci est incapable de les utiliser correctement et efficacement ?
Les deux principales méthodes cognitives pour la résolution du jeu en IA sont d'une part le système expert par l'emploi de connaissances conseils sur les choix à effectuer donnant directement et de façon plutôt réactive (situation -- connaissances -> décision) la décision à prendre. D'autre part des algorithmes de recherche arborescente font une évaluation des coups légaux à partir de l'évaluation d'un très grand ensemble de situations prédites par déroulement des règles du jeu sur l'ensemble des choix et événements possibles du système et de ses adversaires.

La première méthode offre l'avantage d'être "facilement" explicable à l'être humain car basée sur des connaissances provenant d'experts humains. La seconde méthode reposant sur une exploration trop grande pour la mémoire à court terme de l'être humain est en général difficilement vérifiable par ce dernier, méthode qu'il qualifie d'ailleurs souvent de non élégante. Elle comporte néanmoins l'avantage d'être robuste et générale.
Nous prenons en compte ici des connaissances déclaratives provenant d'un système expert pour guider la recherche arborescente dans des jeux à informations incomplètes multi-joueurs à déroulement variable pour lesquels une fonction d'évaluation est difficile à déterminer. Ce type de jeu très complexe rend nécessaire l'utilisation d'une telle technique qui devrait allier robustesse et compréhensibilité pour une faible complexité ou du moins une complexité contrôlée. Nous pensons que c'est par l'étude de ce type de jeux complexes, et non pas des problèmes jouets, que l'intelligence artificielle doit se tourner, jonglant entre les connaissances du monde facilement modélisables et une complexité importante.
Nous donnons dans 2 les principes de la recherche arborescente dans les jeux, en particulier les jeux à information complète à deux joueurs et à somme nulle. Ces méthodes, inadaptées pour les jeux à informations incomplètes seront ensuite adaptées et améliorées dans 3. Nous verrons enfin dans 4 leur application au jeu de l'ascenseur.

2. Arborescences

Nous présentons un type de raisonnement largement utilisé dans les systèmes actuels pour la résolution de problème, surtout les jeux, à savoir les recherches arborescentes.

2.1. Recherche arborescente dans les jeux

Nous rappelons brièvement le principe des algorithmes de recherche arborescente classique utilisés dans les jeux, à savoir le min-max et l'alpha-beta. Nous exposons aussi rapidement les améliorations moins connues proposées par Berliner dans son B*. [Cazenave 00] fait un tour plus complet des algorithmes de ce type.

2.1.1. Le min-max

Le min-max est l'algorithme de base des recherches arborescentes dans les jeux à 2 joueurs à somme nulle à information complète. Il consiste en une exploration exhaustive jusqu'à une certaine profondeur des situations pouvant être engendrées à partir de la situation actuelle. La recherche peut être représentée par un arbre dont les nœuds correspondent à chacune de ces situations, les arcs de niveau impair aux choix possibles (coups légaux) du système et les arcs

de niveau pair aux coups légaux de l'adversaire. Les nœuds feuilles sont évalués par le système à l'aide d'une fonction d'évaluation. Les valeurs sont ensuite remontées en appliquant l'opérateur min pour une propagation sur des arcs correspondant aux choix de l'adversaire et l'opérateur max pour les choix du système.
Le min-max est basé sur l'hypothèse que le joueur adverse utilise un raisonnement du même type que lui, à savoir un min-max de même fonction d'évaluation à une profondeur n-1 (où n est la profondeur du min-max initial).
La complexité du min-max est de l'ordre de Kn où K représente le nombre de coups légaux par tour et n la profondeur de la recherche. Cet algorithme devient donc rapidement inutilisable lorsque le nombre de coups légaux est important, c'est pourquoi des optimisations de cet algorithme ont été réalisées.

2.1.2. L'alpha-beta

L'alpha-beta est une optimisation du min-max: il permet d'effectuer de nombreuses coupes dans l'arbre par l'élimination de recherches certainement a priori infructueuse car aboutissant assurément à une évaluation moins bonne.
- Coupe alpha
Si l'avancement de la recherche permet de conclure de façon certaine qu'un coup légal C1 est préféré par le système à un autre coup légal C2, alors il est inutile de continuer le développement de C2. Cette constatation générale appliquée au min-max (c-a-d. sous la condition que l'autre joueur raisonne de façon similaire) produit les coupes alpha:
Si un coup C1 d'un niveau max (choix du système) est évalué à f(C1) et qu'un coup C2.1 (coup légal de niveau min immédiatement inférieur à un coup C2 de même niveau que C1) est évalué à $f(C2.1) \leq f(C1)$, alors il est inutile d'explorer plus avant le C2, C1 étant préféré à C2. Il est en effet facile de montrer que $f(C2) \leq f(C1)$.
- Coupe beta
De façon similaire, la coupe beta repose sur la constatation générale que si l'avancement de la recherche permet de conclure de façon certaine qu'un coup légal C1 est préféré par l'autre joueur à un autre coup légal C2, il est inutile de continuer le développement de C2:
Si un coup C1 d'un niveau min (choix de l'adversaire) est évalué à f(C1) et qu'un coup C2.1 est évalué à $f(C2.1) \leq f(C1)$, alors il est inutile d'explorer plus avant le coup C2.
L'alpha-beta est strictement équivalent au min-max du point de vue de la solution obtenue. Son amélioration est uniquement un gain de complexité. En particulier, il est basé lui-aussi implicitement sur l'hypothèse que le joueur adverse utilise un raisonnement du même type que lui.

*2.1.3. Le B**

Le principe original sur lequel repose le B* ([Berliner 79]) est l'utilisation de 2 nombres pour représenter l'évaluation d'une situation: un nombre optimiste et un nombre pessimiste.
La remontée de ces évaluations est réalisée selon un principe analogue au min-max:
A travers les arcs min: la valeur optimiste (resp. pessimiste) du nœud père est obtenue par l'application de l'opérateur min sur l'ensemble des valeurs optimistes (rep. Pessimistes) de ses nœuds fils.
A travers les arcs max: la valeur optimiste (resp. pessimiste) du nœud père est obtenue par l'application de l'opérateur max sur l'ensemble des valeurs optimistes (rep. Pessimistes) de ses nœuds fils.
La valeur optimiste d'un nœud correspond à la valeur que le système est assuré de réaliser à une profondeur n dans la situation représentée par le nœud; la valeur pessimiste à la valeur que le système est certain de ne pouvoir atteindre; ceci sous l'hypothèse que l'autre joueur suit un raisonnement analogue au système à une profondeur n-1.
Autrement dit, il existe au moins une succession de n/2 choix du système à partir de la situation S telle que, quels que soient les choix de l'adversaire, toutes les situations à une profondeur n résultantes auront pour évaluation pessimiste une évaluation pessimiste supérieure ou égale à l'évaluation pessimiste de S; et la valeur pessimiste de S est la plus grande satisfaisant cette contrainte.
D'un autre côté, il n'existe aucune succession de n/2 choix du système à partir de la situation S telle que la plus petite des valeurs optimistes des situations de profondeur n résultantes de chacun des n/2 choix de l'adversaire correspondants, soit strictement supérieure à la valeur optimiste de S; et la valeur optimiste de S est la plus grande satisfaisant cette contrainte.
Les deux idées du B* que nous retenons principalement pour la suite du papier sont que:
- les deux évaluations, optimiste et pessimiste, procurent une information non seulement sur l'évaluation de la situation, mais aussi sur l'imprécision de cette évaluation.
- les conditions des coupes alpha-beta ont été adaptées à des fonctions d'évaluation de nature différente; la règle importante d'établissement des coupes de l'alpha-beta est donc la règle générale déjà mentionnée en 2.1.2:

si *un choix C1 du système est assurément meilleur ou équivalent pour lui qu'un choix alternatif C2*
alors *il est inutile d'étudier C2*

2.1.4. Les jeux à n joueurs

Les algorithmes précédents ne peuvent s'appliquer à des jeux à n joueurs.
Leur adaptation à n joueurs peut paraître dans un premier temps aisé, l'alternance des coups se faisant sur n joueurs plutôt que 2. On obtiendrait ainsi un algorithme min-min-max pour un

jeu à 3 joueurs et, dans le cas général, un algorithme min(n-1)-max.
Cependant, la grande majorité (voir la totalité) des jeux à plusieurs joueurs fait intervenir les participants dans des relations plus complexes que celles supposées jusqu'ici. En effet, un algorithme min(n-1)-max pose implicitement l'hypothèse que les autres joueurs sont non seulement des adversaires pour ce dernier mais en plus, sont tous en coopération pour la réalisation d'un unique but: faire perdre le système. Dans les faits, les jeux offrent au contraire une plus grande variété de relations entre les participants, le seul but commun dont on peut être absolument certain dans l'ensemble est que chacun joue pour lui-même; ceci n'implique pas forcément qu'il joue contre le système ni contre tous les autres.

2.1.5. Les jeux à information incomplète

Pour les jeux à information incomplète, le problème se complexifie de nouveau. En effet, les coups légaux à chaque tour des joueurs adverses ne sont pas connus. Aux jeux de cartes, par exemple, il n'est pas possible en général de connaître les cartes de la main de ses adversaires et par là-même l'ensemble de leurs choix possibles.
Il serait possible d'adapter les algorithmes de recherche arborescente en considérant dans un premier temps toutes, ou une grande partie, des distributions possibles. Le jeu engendré par chacune de ces distributions est alors un jeu à information complète qu'il est possible de résoudre par les méthodes précédentes. Le meilleur coup à jouer est celui qui maximise l'espérance de la fonction d'évaluation sur l'ensemble des distributions testées.
Outre le fait que nous risquons alors de devoir faire face à une grande complexité en utilisant cette technique, elle repose là-encore sur des hypothèses qu'il est difficile de considérer comme satisfaisantes: on considère en effet que chaque joueur (y compris le système) connaît toutes les cartes de tous les autres et qu'il raisonne là-encore à l'aide d'un algorithme de type min-max.

2.2. Arbres de décision

Nous avons déjà mentionné que les arbres présentés en 2.1 permettaient de ne représenter qu'un ensemble de problèmes très restreint, très précisément: les jeux à 2 joueurs à somme nulle à information complète sans hasard à tour alterné.
Les arbres de décision dans l'incertain permettent de combler cette lacune par l'introduction de nouveaux types d'arcs représentant des événements incertains. Ces événements (=> arcs) sont probabilisés permettant une prise de décision.
D'une manière similaire aux algorithmes basés sur le min-max, les évaluations des nœuds feuilles de l'arbre sont remontées à travers chacun des arcs:
- pour les arcs "choix du système", l'opérateur max est appliqué,
- pour les arcs "événements", un calcul d'espérance est réalisé.

La valeur d'un nœud quelconque est donc l'espérance de l'évaluation des situations pouvant résulter de l'ensemble des choix du système maximisant cette espérance. Le système est assuré dans une situation S de pouvoir obtenir une espérance du gain[1] au moins égale à l'évaluation de S; et cette évaluation est la plus grande.

3. Le contrôle de la recherche arborescente pour les jeux à information incomplète à n joueurs à structure temporelle variable

Nous proposons ici la réalisation d'un système basé sur une recherche arborescente des possibilités inspiré du B* de Berliner, pour lequel la fonction d'évaluation est accompagnée d'une méta-fonction d'évaluation. Nous verrons en particulier comment cette méta-connaissance peut-être utile pour le contrôle du développement. Contrairement à l'évolution des algorithmes actuels pour lesquels l'ensemble des connaissances sur les bons coups à jouer réside dans la fonction d'évaluation, nous accorderons une grande importance aux conseils donnés a priori; les réactions des autres joueurs seront modélisés comme des événements extérieurs probabilisés.

3.1. Recherche arborescente dans les jeux à n joueurs à information incomplète à structure temporelle variable

Nous proposons d'adapter les algorithmes de recherche classiques à l'ensemble des jeux plus complexes, à information incomplète, multi-joueurs à structure temporelle variable. La recherche arborescente offre en effet une grande robustesse du fait de la projection dans le futur du système.
La première modification sera l'introduction d'hypothèses abstraites comme type de nœud de l'arbre. Le système pourra ainsi considérer des choix abstraits (choix stratégiques) ou des événements abstraits (ex.: *je ne remporte pas le pli* au lieu de *JoueurX remporte le pli*...). La précision du raisonnement s'en trouve légèrement diminué pour un gain important sur le coût de recherche. Il est, de plus, plus proche de celui de l'être humain.
La deuxième modification proposée, issue de la première, est la représentation des choix adverses à travers les événements probabilisés. Ce choix est nécessaire à partir du moment où le système devient capable de considérer des événements abstraits pouvant comporter à la fois de l'information concernant l'état du monde et des choix de l'adversaire. Par exemple, aux

[1] Le terme "gain" dans une situation S est utilisé en théorie de la décision. Associé à une fonction d'utilité (qui peut être l'identité), ce gain correspond à ce que nous appelons ici "évaluation" de S. Nous utilisons dans ce papier indifféremment les 2 termes.

jeux de cartes, lorsqu'on décide, pour une situation donnée, d'envisager le cas où le système est maître (remporte un pli) ou ne l'est pas, on fait une hypothèse à la fois sur la disposition possible des cartes (pour qu'un autre joueur remporte le pli, il faut qu'il ait les cartes adéquates), et à la fois sur le choix des autres joueurs (ce n'est pas parce qu'un joueur possède une carte plus forte à celle du système qu'il la jouera forcément). De plus, représenter le choix des autres joueurs par des événements probabilisés permet la prise en compte d'un modèle de l'adversaire explicite.
Dès lors où le système fait de la planification sur des événements abstraits, le système raisonne sur le long terme avec un manque d'informations de plus en plus grand. Pour cette raison, nous garderons une représentation de la fonction d'évaluation basée sur 2 valeurs, une valeur optimiste et une valeur pessimiste. Le système est ainsi capable d'avoir une information de précision sur sa fonction d'évaluation sur laquelle il pourra s'appuyer pour contrôler sa recherche.
Le contrôle de la recherche arborescente passe par quatre étapes. La première étape est le choix du nœud à développer, la deuxième le choix du type d'hypothèse à poser, la troisième le choix des valeurs à étudier pour la fonction choisie dans l'étape précédente, la dernière étant la remontée des informations et la désactivation de certains nœuds.

3.2. Choix du nœud à développer

Commençons par préciser quelques concepts utilisés par la suite:
Choix principal: choix pour lequel le système doit prendre une décision. Ce choix correspond dans les arbres classiques à des arcs issus du nœud correspondant à la situation courante.
Choix alternatif: une choix C1 est un choix alternatif à un choix C2 si C1 et C2 correspondent à une même décision dans une même situation. Les arcs correspondant à ces choix dans l'arbre sont donc issus du même nœud.
Nœud Choix: nœud issu d'un arc choix
Nœud Evenement: nœud issu d'un arc événement

La mise au point de l'alpha-beta a rendu primordiale la question du méta-choix de l'ordre des choix à envisager pour un même niveau. Cet ordre est le conditionnement du nombre de coupes effectuées par l'alpha-beta. Ce nombre est d'autant plus grand, et l'algorithme plus efficace, que les meilleurs coups sont considérés en premier par le système. En effet, si le système obtient très tôt la meilleur valeur pour le choix C1, il lui sera inutile de continuer à développer les coups Cx alternatifs à C1 à partir du moment où il aura déjà développé une des branches Cx.1, la valeur trouvée par chacune de ces explorations aboutissant forcément à de nouvelles coupes.

Le choix du nœud à développer doit donc toujours être guidé par l'optique d'obtenir le plus rapidement possible les coupes. Lorsque la fonction d'évaluation est donnée par deux bornes, ces coupes pourront être obtenues en réduisant l'imprécision sur cette fonction de telle sorte que la borne inférieure de la fonction d'évaluation de la situation engendrée par un des choix principaux devienne plus grande que toutes les bornes supérieures des choix alternatifs.
C'est pourquoi:

- **Le premier objectif est la réduction de l'imprécision sur la fonction d'évaluation des situations engendrés par chacun des choix principaux.**
- **Le second objectif est la réduction de cette imprécision en priorité sur les choix les plus prometteurs.**

Nous définissons donc:
La probabilité d'un noeud ?n: la probabilité de la situation correspondant à ?n sachant que le système a voulu y parvenir.

Si *=(Pere(?n),?N)*
NoeudChoix(?n)
=(Probabilite(?N),?p)
Alors *=(Probabilite(?n),?p)*

Si *=(Pere(?n),?N)*
NoeudEvenement(?n)
=(Probabilite(?N),?p1)
=(ArcEntrant(?n),?a)
=(Probabilite(?a),?p2)
*Calcul ?p3 := *(?p1,?p2)*
Alors *=(Probabilite(?n),?p3)*

L'imprécision d'un nœud: la différence entre la valeur optimiste et la valeur pessimiste de la fonction d'évaluation associée à ce nœud.

L'intérêt a priori d'un nœud ?n: la somme des intérêt des choix permettant d'aboutir à ?n pondéré par $2^{-\text{profondeur du choix}}$, la profondeur d'un choix C correspondant au nombre de choix ayant été fait sans l'arbre avant C. Cette pondération permet de donner de l'importance au choix les plus hauts tout en favorisant le développement des meilleurs choix a priori d'un même niveau.

L'intérêt du développement d'un noeud est alors fonction de ces trois notions et donné par la formule: Imprecision(?n) * (Probabilite(?n) + k * InteretAPriori(?n)) où k est une constante.

La première terme de la somme (Imprecision(?n) * Probabilite(?n)) correspond à la précision maximum qu'il est possible de gagner sur l'évaluation du choix principal en continuant à développer le nœud. Le système aura donc tendance à ne pas développer les situations trop improbables. Le second terme permet de prendre en compte les connaissances a priori et guider la recherche vers les choix les plus prometteurs.

3.3. Le choix du type d'hypothèse

Alors que pour la plupart des jeux classiques à 2 joueurs, le déroulement du jeu est fixé une fois pour toute par l'alternance des coups de chacun des joueurs, il n'en est pas de même pour la majorité des jeux de cartes. [Kocik 99] montre comment représenter la structure temporelle de cette famille de jeux. Le déroulement d'un tour de jeu est généralement dépendant du déroulement suivant.

Alors que la situation initiale du monde est rendue partiellement inconnue par le système dans les jeux de cartes, l'utilisation d'arcs relatifs à des événements abstraits (rendue nécessaire par cette incomplétude des informations cf 3.1) dans une recherche arborescente ajoute une méconnaissance supplémentaire sur les situations prévisionnelles. Les situations représentées par les nœuds de l'arborescence sont connues de manière incertaine et imprécise; le déroulement même du jeu devient lui aussi vague.

Toutes ces difficultés (déroulement du jeu variable, incomplétude des informations, hypothèses sur des événements abstraits...) rendent nécessaire l'élaboration d'une expertise de choix du type d'hypothèse à poser.

Nous conseillons l'utilisation de connaissances déclaratives à travers l'emploi de métarègles analysant la situation (quelles sont les connaissances disponibles / indisponibles dans la situation en question ?) conseillères sur ce type d'hypothèse.

A l'ascenseur, par exemple, il est primordial pour l'étude d'un pli de connaître l'ouvreur de ce pli (ou du moins savoir si le joueur simulé par le système sera ou non l'ouvreur), les coups légaux des joueurs étant déterminés pour une grande partie par le choix de cet ouvreur. Lorsqu'un nœud de l'arbre de recherche décrit une situation dans laquelle l'ouvreur d'un pli n'est pas connu alors qu'elle comporte une description satisfaisante du pli précédent, il sera bon que le système envisage de nouvelles situations générées par la précédente et fonction du joueur qui sera ouvreur.

Il apparaît clairement que cette expertise dépend fortement des règles du jeu auquel le système est confronté.

Le jeu de l'ascenseur

Pour le jeu de l'ascenseur, nous proposons dans un premier temps les métarègles suivantes qui, malheureusement, sont à la fois trop rigides et trop spécifiques à ce jeu du fait qu'elles résultent de la fusion d'un ensemble de métarègles générales qu'il reste encore à découvrir.

D'un autre côté, elles présentent l'avantage d'être opérationnelles rapidement:
Conseils sur les hypothèses de type Choix du système:

Si *?N est un nœud de l'arborescence*
?N contient un ensemble de conseils de poids ?p (positif ou négatif) sur un choix de type ?C du système
Le choix du type ?C n'a pas encore été déterminé dans ?N
?p1 est le poids le plus élevé de cet ensemble
Alors *il est bon [?p1] de poser une hypothèse sur ce type de choix à partir de la situation ?N*

Cette métarègle est déclenchable aussi bien pour des choix stratégiques que tactiques à partir du moment où des conseils au niveau du raisonnement de base ont permis de conclure sur l'intérêt de ces choix.

Conseils sur les hypothèses de type Evénement:

Si *?N est un nœud*
La carte posée par Moi (moi étant le joueur simulé par le système) est connue pour un pli ?P
Le fait que Moi a remporté le pli ?P ou non n'est pas connu dans ?N
Alors *il est bon de poser l'hypothèse sur le fait que Moi remporte le pli ?P ou non à partir de ?N*

Si *?N est un nœud*
L'ouvreur d'un pli ?P n'est pas Moi dans ?N
La couleur demandée du pli ?P n'est pas connue dans ?N
Alors *il est bon de poser une hypothèse à partir de ?N sur la valeur de la couleur demandée*

Le déclenchement de ces règles aboutit sur un ensemble de conseils sur le type d'hypothèse à poser qu'il est nécessaire d'agréger. Le conseil résultant de poids maximal indique la méta-décision à prendre.
Ces métarègles sont suffisantes pour une première approche de la résolution des problèmes du jeu de l'ascenseur. Nous verrons comment elles s'appliquent dans la partie 4.4.

3.4. Le méta-choix des valeurs

Une fois que le système connaît le type d'hypothèse à poser, il lui reste encore à méta-considérer les instances à considérer.
Une première approche serait de considérer l'ensemble des valeurs possibles: le système envisagerait toutes les cartes autorisées à jouer pour un pli donné, il envisagerait les 2 cas systématiquement où il remporte le pli ou ne le remporte pas, et aussi les cas où la couleur

demandée prend chacune des valeurs possibles des 4 couleurs... L'intelligence de l'être humain semble cependant résider en grande partie dans sa capacité à sélectionner les coups et les situations sur l'ensemble des situations possibles. Nous proposons donc ici d'utiliser des connaissances déclaratives pour sélectionner celles que le système sera amené à envisager.
A partir de l'ensemble des conseils sur un choix du système, nous éliminons tous les coups dont le poids du conseil est inférieur à un seuil égal à la valeur du poids le plus grand moins une constante d'acceptation de choix fixée au départ. Plus cette constante est grande, plus le nombre de coups envisagé est important, plus les chances d'envisager le meilleur coup sont grandes et, en contrepartie, plus le coût d'exploration est grand.
D'une façon analogue, lorsque le système a décidé de poser une hypothèse sur un type d'événement, il élimine tous ceux dont la probabilité est inférieure à une constante de sélection des événements. Plus cette constante est grande, plus le nombre de situations envisagées est grand. Le système n'est donc pas amené à envisager des situations trop improbables.

3.5. Remontée de l'information et désactivation des nœuds

La remontée de l'information est réalisée sur un principe similaire au B* pour les hypothèses de type choix du système. La borne inférieure (resp. borne supérieure) de la fonction d'évaluation d'un noeud dont les arcs sortant sont des choix est obtenue en prenant le maximum de la borne inférieure (resp. borne supérieure) des situations résultantes de ces choix. La borne inférieure (resp. supérieure) d'un noeud dont les arcs sortant sont des événements probabilisés est calculée comme la somme des produits de la probabilité de chaque arc par la borne inférieure (resp. supérieure) des situations résultantes.
Comme pour le B*, lorsque l'espérance pessimiste d'un choix C1 est meilleure que l'espérance optimiste d'un choix alternatif C2, alors le choix C1 est assurément meilleur que le choix C2. D'après la règle générale donnée en 2.1.2, le système conclue qu'il est inutile de développer les nœuds issus du choix C2.

4. Application: un exemple détaillé du jeu de l'ascenseur

Nous montrons sur un exemple tiré du jeu de l'ascenseur une application des idées présentées précédemment.

4.1. Bref rappel des règles du jeu de l'ascenseur

Afin de mieux comprendre cet exemple, un rappel des règles du jeu de l'ascenseur est

nécessaire.

4.1.1. Règles du jeu

Le jeu de l'ascenseur est un jeu de cartes de la famille des jeux de levées à 2 joueurs ou plus. Après distribution d'une partie ou de la totalité du talon à l'ensemble des joueurs, ces derniers doivent annoncer le nombre de levées ou plis qu'ils croient pouvoir réaliser avec le jeu en main, un atout (couleur des cartes les plus fortes) étant fixé par la distribution. Le jeu de la carte se déroule ensuite de façon classique:
- chaque joueur joue une carte et une seule par pli.
- l'ouvreur (celui qui entame) du 1er pli est le joueur suivant le donneur.
- le joueur maître d'un pli (celui qui a posé la carte la plus forte) ouvre le pli suivant.
- l'ouvreur peut jouer n'importe laquelle des cartes qu'il a en main.
- la carte jouée par l'ouvreur détermine la couleur demandée (couleur de cette carte).
- les autres joueurs doivent fournir à la couleur demandée s'ils en possèdent.
- si un joueur ne possède pas de la couleur demandée, il peut jouer n'importe quelle carte de sa main, c'est-à-dire ou bien couper (jouer atout) ou bien se défausser (carte qui n'est ni atout ni de la couleur demandée).

La carte la plus forte du pli est:
- le plus fort atout (le plus haut) posé si au moins 1 atout a été joué.
- la carte la plus haute dans la couleur demandée sinon.

4.1.2. Originalités

Le jeu de l'ascenseur est un jeu qui offre une grande palette de situations différentes et nécessite la mise en oeuvre d'un grand nombre de mécanisme de raisonnement. Le nombre de joueurs, le nombre de cartes du talon initial, le nombre de cartes distribuées (et par conséquent, le nombre de cartes dans le talon après distribution), le jeu avec et sans atout sont autant de paramètres montrant cette diversité.
Mais sa grande originalité par rapport aux autres jeux de levées (et à la large majorité des jeux en général) est la difficulté engendrée par le fait que les joueurs doivent réaliser un nombre de levées bien précis. Il en résulte que certaines parties se déroulent avec une volonté globale de l'ensemble des participants de maximiser le nombre de plis remportés _ le jeu est alors de type qui gagne gagne _ alors que d'autres parties nécessitent de faire attention de ne pas réaliser trop de levées _ le jeu est de type qui perd gagne.
Alors qu'il est possible pour beaucoup de problèmes de jouer correctement avec des connaissances générales, l'utilisation de ces connaissances devient réellement problématique dès lors que les connaissances prodiguant des conseils à notre disposition sont contradictoires et où une fonction d'évaluation de la situation est très difficile à mettre au point.

La nécessité se fait ressentir de développer des arborescences plus ou moins précises et limitées dirigées par les conseils a priori.

4.2. Présentation de l'exemple

Jeu de 52 cartes à 4 couleurs (♠, ♥, ♦, ♣) et 13 hauteurs (As, R, D, V, 10, 9, 8, 7, 6, 5, 4, 3, 2).
Main du Joueur1:
♠: 4
♥: R
♦: As 6
♣: As
Atout: 8♣
Ouvreur: Joueur1
Nombre de joueurs: 4
Le Joueur1 est le joueur simulé par le système.

4.3. Le raisonnement

L'As♣ étant la carte d'atout la plus forte du jeu, Joueur1 réalisera 1 pli certainement avec cette carte. Le 4♠ et le 6♦ étant des cartes très basse, il est quasi-certain qu'elles ne pourront conquérir une levée, même si les joueurs (y compris Joueur1) font tout dans ce sens. L'As♦ est la carte la plus forte à sa couleur, le R♥ l'est très probablement aussi _ il y a de grandes chances pour que l'As♥ soit dans le talon. Vu le faible nombre de cartes dans les mains des joueurs, il est fort probable que pour chacune de ces couleurs, il existe au moins un des autres joueurs qui ait une coupe. La réalisation des plis avec le R♥ et l'As♦ est donc très incertaine.
Joueur1 hésite donc pour une annonce égale à 2 ou à 3 avec une préférence tout de même accrue pour la première proposition.
Joueur1 envisage maintenant de vérifier sa première intuition en étudiant le choix annonce=2 en essayant d'estimer le mieux possible ses chances de réussite.
L'As♦ est assez bien protégé par le 6♦, si bien que s'il décide de ne pas faire le pli avec l'As♦, il est très probable qu'il réussisse. Au contraire, Joueur1 n'a aucun contrôle sur le R♥ dont le comportement dépend uniquement de la distribution et de la volonté des autres joueurs.
Joueur1 considère donc qu'il est raisonnable de jouer le R♥ le plus tôt possible afin de lever l'incertitude sur la réalisation d'une levée avec cette carte. Si le R♥ est maître, la difficulté pour Joueur1 sera d'éviter de réaliser un pli avec l'As♦. Au contraire, si le R♥ est coupé ou remporté par l'As♥, Joueur1 s'efforcera de se défausser de son As♦, ce qui ne devrait pas

poser de problèmes dans le mesure où le seul cas défavorable est celui où ♦ est jouée 2 fois avant qu'il ne soit ouvert 2 fois aussi à n'importe laquelle des autres couleurs: en particulier, si ♥ est ouvert de nouveau...
Mais revenons sur le cas où le R♥ n'est pas maître. Joueur1, désirant alors faire le pli avec l'As♦, a le choix entre 2 stratégies:
jouer le plus tôt possible l'As♦ pour limiter les chances de défausse des autres joueurs à ♦ et donc minimiser la probabilité qu'ils puissent couper cet As. Il faudra donc jouer l'As♦ au 1er pli ouvert à ♦ ou tenter de prendre la main avec l'As♣ le plus tôt possible en coupant dans une autre couleur afin d'ouvrir tout de suite avec l'As♦.
Ou jouer le plus tard possible l'As♦, en espérant que les atouts des autres joueurs soient tombés au moment ultime. Stratégie très risquée et peu maîtrisée du fait de l'impossibilité de prendre la main avec une autre carte que l'As♣ et donc de faire tomber volontairement les atouts.

4.4. Simulation de raisonnement avec SYMOR

Nous donnons à la **Figure 1** le type d'arborescences que le système SYMOR serait susceptible de donner. La fonction d'évaluation est une estimation de la probabilité que le joueur simulé réussisse son contrat: lorsque celui-ci est assuré de le réaliser, la situation est évaluée à 1; lorsqu'il est assuré de le chuter, elle est évaluée à 0. Les valeurs intermédiaires correspondent à une mesure du hasard. Les bornes inférieures et supérieures de ces probabilités (mesures de nécessité et de plausibilité) donnent une information sur le manque de précision sur cette mesure du hasard.

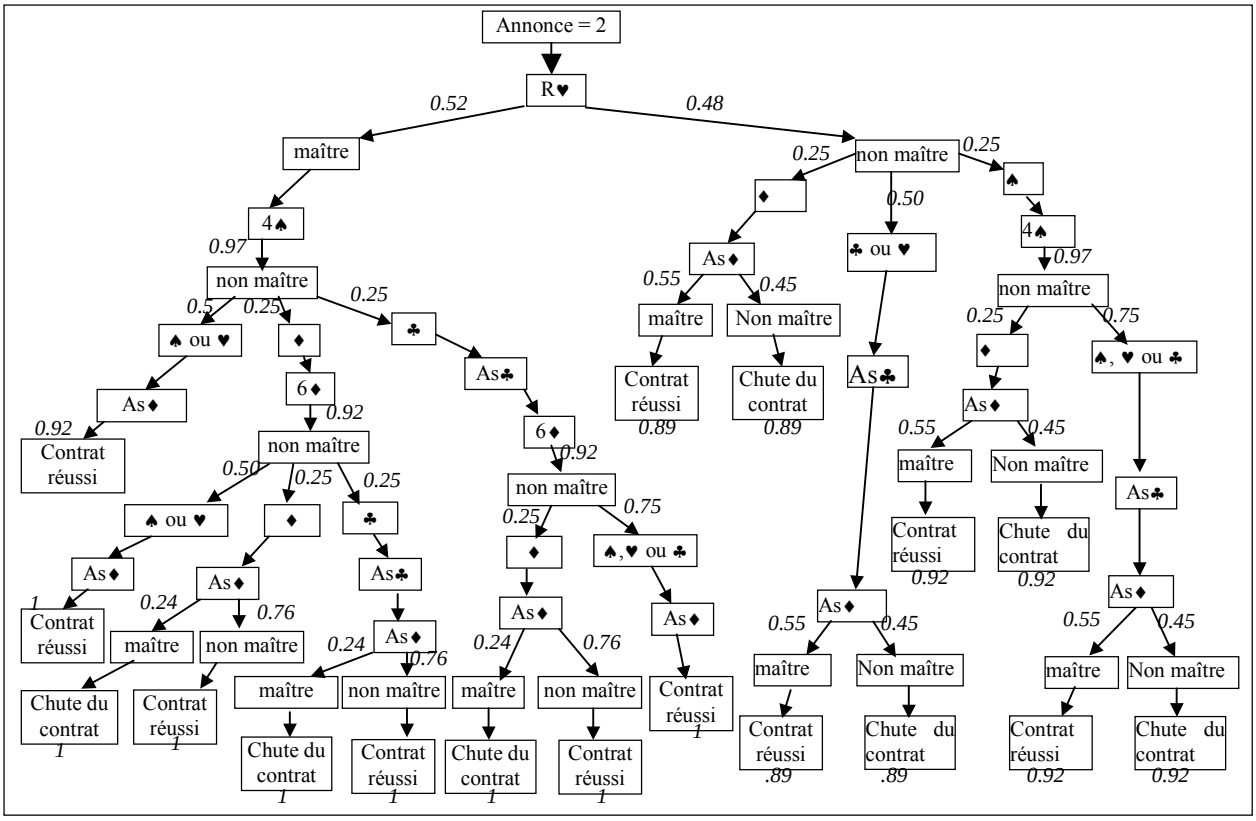


Figure 1: simulation du raisonnement avec SYMOR

5. Conclusion

Les idées données dans cet article sont une première étape pour allier les avantages des méthodes du type système expert par l'utilisation de connaissances déclaratives et ceux offerts par les algorithmes de recherches arborescentes.

Elles se résument dans:

- l'utilisation de connaissances plus ou moins abstraites
- l'étude des situations les plus probables et prometteuses en priorité
- l'étude des situations "floues" en priorité sur les situations "claires" dans le but d'obtenir une description du monde en prévision totalement claire

Elles permettront de:

- limiter la combinatoire en contrôlant la recherche arborescente en largeur et en profondeur.
- d'avoir un meilleur contrôle du temps
- faciliter l'explication des solutions engendrées

Après l'application effective de ces idées sur le jeu de l'ascenseur dans un premier temps, nous pensons qu'il faudra pousser plus loin la mise en place de méta-raisonnements intelligents pour le contrôle de raisonnements de base par:

- la prise en compte du coût de développement des arborescences
- la prise en compte des règles du jeu
- une plus grande flexibilité sur le type d'hypothèses à poser, en particulier par un choix du niveau d'abstraction du raisonnement. Nous voyons en effet dans 4.3 que le raisonnement d'un être humain devient de plus en plus abstrait au fur et à mesure de l'avancée en profondeur dans l'arbre de recherche pour lesquelles les situations ont de moins en moins de chances de se produire réellement.

6. Bibliographie

[Berliner 79] Berliner H., *The B* tree search algorithm: a best-first proof procedure*. Artificial Intelligence 12 (1), p 23-40.

[Cazenave 00] Cazenave T., *Des optimisations de l'alpha-beta*. Actes colloque Berder 2000.

[Kocik 99] Kocik F., *Un système général de jeu de cartes, un domaine intéressant pour l'IA*. Rapport de recherche LIP6 1999/014.

[Laurière 96] Laurière J-L. *propagation de contraintes ou programmation automatique*. Rapport LAFORIA 96/19.

[Nunn 00] Nunn J., *Secrets of Practical Chess*. Eds Gambit 2000, p 1-66.

[Pastre 00] Pastre D. *Chemins détournés, idées fausses et bonnes idées*. Actes colloque Berder 2000.

[Parchemal 88] Parchemal Y., *SEPIAR: un système à base de connaissances qui apprend à utiliser efficacement une expertise*, Thèse de l'Université Paris 6, 1988.

[Pitrat 00] Pitrat J. *La mise en place du monitoring dans MALICE*. Actes colloque Berder 2000.

[Pitrat 99] Pitrat J., *une expérience de monitoring*. Rapport de recherche LIP6 1999/014.

Combinaison d'hypothèses dans les problèmes à solution complexe : méthode a posteriori contre méthode a priori

Tristan PANNEREC,
Tristan.Pannerec@lip6.fr

Résumé: L'utilisation de raisonnement hypothétique dans les systèmes de résolution de problèmes pose la question fondamentale du contrôle des hypothèses développées. Cette question s'accentue encore lorsque la résolution porte sur des solutions complexes à optimiser et que les hypothèses correspondent à des choix sur le contrôle de cette résolution. En particulier, les solutions obtenues selon différentes hypothèses doivent alors être combinées pour obtenir la solution finale. Dans ce papier, nous proposons deux méthodes basées sur des capacités de monitoring pour aborder ce problème et présentons une comparaison de ces deux méthodes d'un point de vue théorique.

Mots-clés: raisonnement hypothétique, combinaison d'hypothèses, optimisation, construction de solutions complexes, résolution de problèmes.

1. Introduction

L'utilisation de raisonnement hypothétique dans les systèmes d'Intelligence Artificielle est apparue très tôt dans l'histoire de cette discipline, tout en prenant des formes diverses. Du backtracking classique en résolution de problème aux algorithmes alpha-beta des jeux, les exemples ne manquent pas. Le principe est de palier un manque d'information en étudiant les différentes valeurs possibles pour déterminer ensuite une solution au problème posé. Le problème récurrent est le contrôle de ce mode de raisonnement, car les hypothèses sont souvent nombreuses et ne peuvent être étudiées systématiquement. Mais, selon le type d'hypothèse gérée, le problème peut différer grandement. Or, dans la plupart des travaux effectués à ce sujet, les hypothèses portent sur des données du problème ou des données de la solution. Lorsque l'on cherche à utiliser des hypothèses sur la manière de résoudre le problème, les voies à explorer, la façon de construire la solution ou d'une manière plus générale sur des « méta-choix »[4] permettant de contrôler et diriger la résolution, le problème de la gestion des hypothèses comporte alors de nombreux aspects spécifiques.

[4] Nous utiliserons plutôt dans la suite de ce papier le terme de choix « abstrait » pour désigner ces choix de contrôle. Le terme de méta-choix sera réservé aux choix concernant l'annulation ou la modification de choix.

Cette question s'est rapidement posé dans le système MARECHAL[5] qui permet de construire des solutions complexes pour des problèmes d'optimisation et qui utilise intensément des choix abstraits. Une première méthode a alors été implémentée pour répondre à ce problème. A posteriori, celle-ci ne s'est pas révélée totalement satisfaisante et une seconde méthode basée sur une philosophie opposée est en cours d'implémentation. Le but de cet article est donc de présenter et comparer ces deux méthodes pour déterminer dans quels cas elles sont intéressantes.
Pour cela, nous commencerons, dans la seconde partie de ce papier, par présenter plus en détail le problème posé, en abordant d'une manière générale la construction de solution complexe et le raisonnement hypothétique. Les deux parties suivantes seront ensuite consacrées à la description des deux méthodes que nous nous proposons de comparer. Chacune de ces deux parties étudiera les avantages et inconvénients respectifs des méthodes.

2. Construction de solutions complexes et raisonnement hypothétique

Le système que nous construisons, baptisé MARECHAL, est un résolveur général de problèmes d'optimisation à solutions complexes. Dans cette partie, nous verrons à quoi correspond ce type de problème et pourquoi ils nécessitent l'utilisation de raisonnement hypothétique et de combinaison d'hypothèses.

2.1. Les problèmes d'optimisation à solutions complexes

Par problème à solutions complexes, nous entendons tout problème de **coordination de choix** pour atteindre un objectif donné. Cela signifie que le système doit **construire** une solution composée d'un ensemble de choix interdépendants. Nous utiliserons indistinctement les termes de choix et de décision pour représenter la sélection d'une possibilité parmi l'ensemble des possibilités initiales d'un choix donné. Plus formellement, soit $E_1..E_n$, n ensembles de la forme $E_i=\{e_{i,1}, .., e_{i,ni}\}$. $E_1..E_n$ représentent les décisions à prendre et $e_{i,1}, .., e_{i,ni}$ les possibilités pour la décision i. Une solution potentielle sera un n-uplet $\{e_{1,j1}, .., e_{n,jn}\}$. La solution finale doit être optimale au sens où elle permet de maximiser un critère ou d'atteindre le plus sûrement ssible l'objectif fixé.

De nombreux exemples de tels problèmes existent. Pour développer le système MARECHAL, nous nous appuyons sur un jeu de stratégie qui consiste à programmer le déplacement de pièces diverses sur un terrain quadrillé. La Figure 2 montre une saisie d'écran de ce jeu qui est

[5] Monitoring-based Autonomous Reasoning Engine for Complex Hypothesis handling

décrit plus en détail dans [Pannérec 00]. Dans ce problème, chaque pièce (dont le nombre varie de 30 à 120) donne lieu à une décision et il y a autant de possibilités pour chacune qu'il y a de mouvements envisageables par pièce (en moyenne, de l'ordre de 10^4). L'objectif peut être de détruire l'adversaire, occuper une position du terrain ou minimiser le nombre de pièce perdue. La difficulté principale réside dans la taille de l'espace de recherche et dans la complexité des règles de transition entre les états successifs du jeu, ce qui rend très difficile une évaluation précise des solutions.

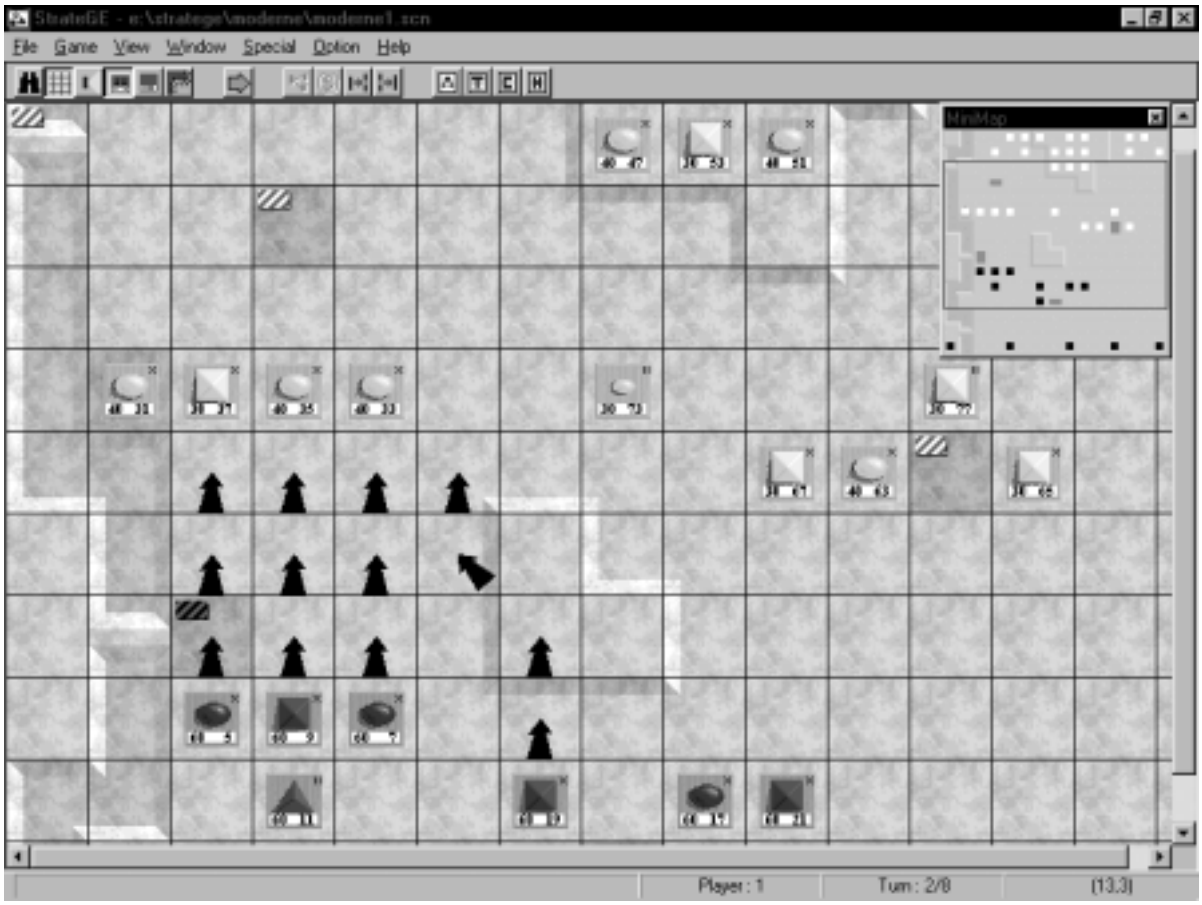


Figure 2 : Saisie d'écran du jeu de réflexion servant d'application au système MARECHAL

Un autre exemple de problème d'optimisation à solutions complexes est le placement de composants électroniques sur un circuit imprimé en fonction du schéma de raccordement. Dans cette application, le système doit faire un choix par composant avec une infinité de possibilités pour chaque choix car les composants peuvent être placés n'importe où sur un espace continu. L'objectif est d'obtenir une configuration viable (pas d'intersection entre des pistes d'équipotentiels différentes) qui optimise de nombreux critères : minimisation du nombre de « straps » entre les couches, minimisation de la place (si la dimension de la plaque n'a pas été donnée), minimisation de la longueur des pistes non protégées des parasites, maximisation de la longueur de la piste d'équipotentiel 0 (masse) etc.

La recherche d'une solution complexe optimale est classiquement connue sous le terme de VCSP (Valued Constraint Satisfaction Problem, voir par exemple [Schiex et al. 97], [Lobjois 99]) en recherche opérationnelle. Ce cadre ne permet malheureusement pas d'utiliser une expertise du domaine pour guider la recherche. Or, la résolution des problèmes soumis au système MARECHAL nécessite une expertise de résolution car l'espace de recherche est trop grand pour être considéré dans sa globalité. Ce cadre n'est en outre pas adapté aux problèmes ayant peu de contrainte dure et des contraintes molles difficilement exprimables et calculables.
De même, les techniques de recherche locale de la recherche opérationnelle se sont révélées inutilisables en pratique à cause de la complexité de la génération des solutions et de leur évaluation. Ces techniques de recherche locale, comme le recuit simulé ([Van Laarhoven et al. 88]), la méthode tabou ([Glover 89], [Glover 90]) ou les algorithmes génétiques ([Goldberg 89]), consistent à produire une ou plusieurs solutions simples puis à améliorer celles-ci par modifications successives. Pour cela, on détermine pour chaque solution courante l'ensemble des solutions proches (voisinage) et on calcule une fonction de préférence pour sélectionner la meilleure transition a priori. La philosophie des recherches locales est d'utiliser une fonction de préférence simple et d'effectuer beaucoup de cycles[6]. Dans le problème qui nous intéresse, il est impossible de trouver une telle fonction et la génération d'un voisinage serait en outre beaucoup trop longue. De plus, la définition même de proximité entre solutions pose un problème : deux solutions structurellement proches peuvent en effet conduire à des valeurs très différentes[7].

2.2. Formalisation par un graphe d'état

Dans les problèmes d'optimisation à solutions complexes, la construction d'une solution suppose d'effectuer une séquence de décisions. Ces problèmes peuvent alors être représentés par une recherche de chemin dans un **graphe d'états**[8]. La notion de **graphe d'états** permet de formaliser la plupart des problèmes d'optimisation et de satisfaction de contraintes. En

---

[6] Pour une présentation des techniques de recherche locale et une mise en parallèle avec la résolution de problèmes en IA, on pourra se reporter à [Grolimund 97].

[7] Par certain coté, les approches que nous décrivons plus loin peuvent s'apparenter au principe de recherche locale, mais avec une philosophie inverse qui consiste à obtenir directement la meilleure transition à effectuer par un raisonnement long et complexe afin de minimiser le nombre de cycles.

[8] Une présentation des graphes d'états apparaît dans [Newell et Simon 72]. Le lecteur souhaitant plus de précisions sur les graphes d'états et les algorithmes de recherche associés pourront se reporter à n'importe quel ouvrage général sur l'Intelligence Artificielle.

fonction des problèmes, les nœuds de ce graphe peuvent représenter des abstractions différentes (état anticipé d'un jeu après une suite de coups, connaissances partielles du fonctionnement d'un mécanisme dans un problème de diagnostic etc.). Les arcs correspondent à des étapes du processus de résolution. Pour les problèmes qui nous intéressent ici, chaque nœud sera associé à une décision et représentera une solution partielle. Les arcs représenteront les différentes possibilités de choix à un instant donné de la construction et pointeront vers les décisions suivantes (dont les possibilités dépendent des décisions précédentes). Dans un tel graphe, chaque feuille constitue une solution potentielle (cf. Figure 3). Le but est alors de trouver le chemin qui conduit à la meilleure solution possible.

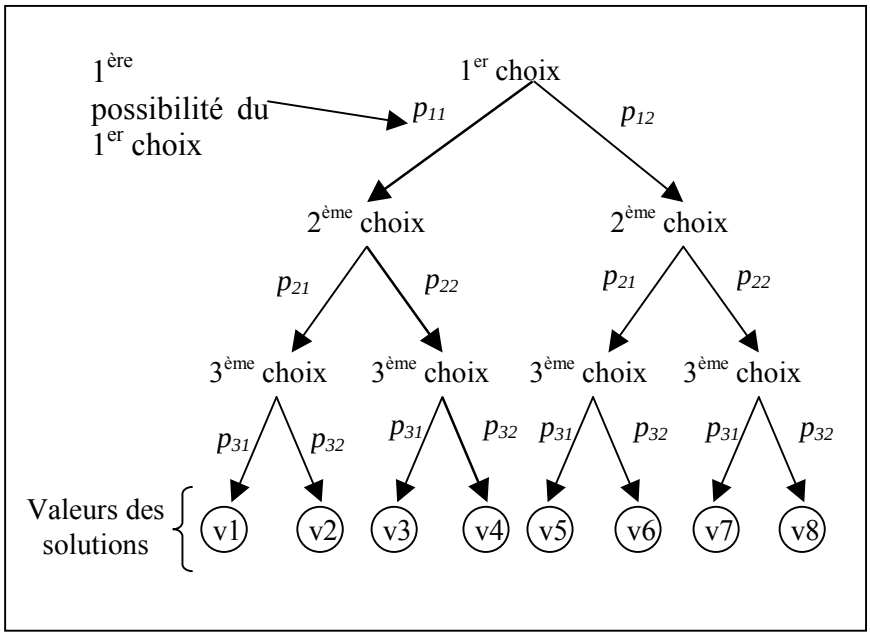


Figure 3 : Représentation d'un problème à solution complexe par une recherche dans un graphe d'état.

2.3. Nécessité de choix abstraits

Naturellement, si la formalisation en graphe d'états est intéressante conceptuellement car universelle, elle n'apporte rien de plus car le parcours exhaustif d'un graphe d'état est en général totalement hors de portée d'une quelconque machine. Si l'être humain considère souvent différentes alternatives pour trouver une solution, seule une infime partie des états possibles du graphe reste étudiée. C'est ce constat qui a motivé l'introduction d'**heuristiques**[9] et de connaissances pour diriger de manière efficace la recherche de la solution. Le principe est d'évaluer a priori les chances de succès d'une branche. Malheureusement, il est en général

[9] Voir [Pearl 84] pour une présentation complète

impossible de trouver des heuristiques parfaites pour un problème, c'est-à-dire conduisant à la solution quelle que soit l'instance du problème. C'est pourquoi les heuristiques ne permettent pas de faire l'économie d'une exploration partielle du graphe d'états. Le problème qui se pose est alors d'explorer intelligemment ce graphe d'états pour trouver les solutions de la manière la plus efficace qui soit. Ce problème peut se résumer par la question désormais classique : « quel état suivant dois-je considérer ? »

En fait, face à l'incroyable combinatoire que présente la plupart des problèmes de décision complexe, l'expert humain raisonne rarement au niveau des choix de base qui constitueront la solution. Au lieu de cela, il réfléchit d'abord sur des choix **abstraits** de haut niveau qui lui permettent ensuite de limiter le nombre de possibilités concrètes à envisager. Par exemple, dans un jeu de stratégie, il se demandera tout d'abord s'il doit plutôt attaquer ou défendre. Dans le premier cas, il pourra ensuite réfléchir à la zone la plus prometteuse pour une offensive. Il n'aura plus alors qu'à considérer les mouvements qui permettent d'attaquer cette zone et ignorera tous les autres. Pour pouvoir transmettre une expertise humaine à un système, il faut donc que celui-ci puisse travailler sur des choix abstraits. Ces choix se rajoutent dans le graphe d'états en le contraignant (cf. Figure 4), mais ne feront pas partie de la solution finale, qui ne comprend que les choix concrets.

Dans ce papier, nous ne nous intéresserons qu'à la gestion des choix abstraits et non à celle des choix concrets. La façon dont le système prend des décisions concrètes est, en effet, présentée en détails dans [Pannérec 00]. Nous supposerons donc qu'en fonction des choix abstraits effectués, le système génère l'ensemble des choix concrets définissant la solution associée.

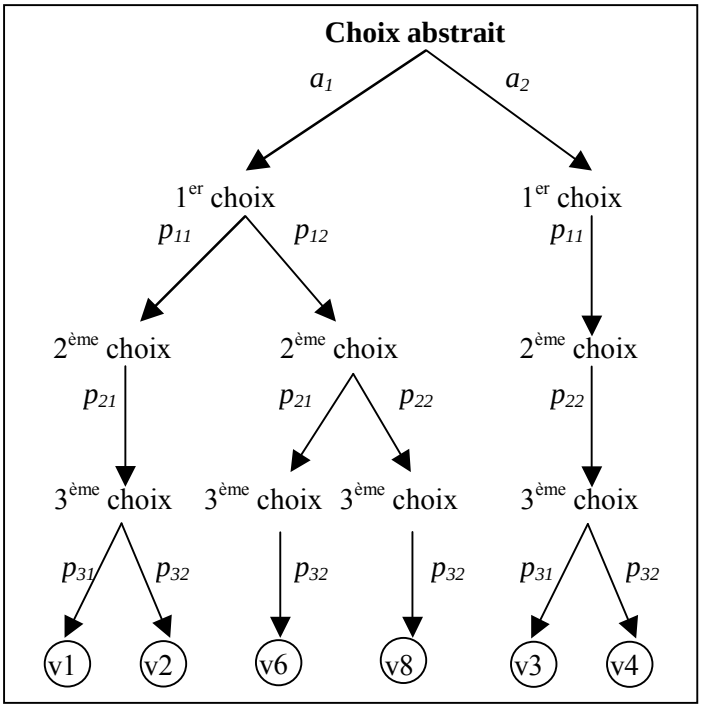


Figure 4 : Graphe d'états avec choix abstraits

2.4. Le raisonnement hypothétique

Dans le système MARECHAL, les possibilités associées aux choix abstraits sont envisagées par le système sous la forme d'**hypothèses**. La possibilité de raisonner sur des hypothèses est l'une des facultés fondamentales de l'intelligence humaine. Le raisonnement hypothétique consiste à supposer vrai une connaissance dont on ne connaît pas en pratique la valeur de vérité, puis à la supposer fausse. Dans les deux cas, on effectue un raisonnement pour résoudre le problème sous l'hypothèse posée (phase de raisonnement), puis on sélectionne la solution globale qui sera proposée (phase de sélection). A partir de ce fonctionnement générique, des différences peuvent ensuite apparaître selon le type d'hypothèse posée.

Pour un système confronté à un environnement dans lequel il doit interagir, les hypothèses peuvent porter sur deux types différents de concept utilisé dans le raisonnement : les concepts contrôlables et les concepts incontrôlables. Les premiers sont constitués des actions personnelles du système dont la maîtrise lui est totalement et exclusivement acquise. Ils sont à la fois la cause et l'objet du raisonnement. Les concepts incontrôlables comprennent, quant à eux, la description de l'état actuel du monde et les évènements futurs générés par ce monde et subis par le système. Les évènements futurs peuvent représenter aussi bien des évènements aléatoires (intervention de la « nature ») pour lesquels la théorie de la décision fournit un cadre formel adapté, que des intentions adverses dans un jeu de stratégie[10] comme le prévoit la théorie des jeux.

Lorsque les hypothèses portent sur des concepts contrôlables, la phase de sélection consiste simplement à choisir la solution de gain maximal. Pour les concepts incontrôlables, le mécanisme est différent car la solution finale doit maximiser l'espérance de gain d'après la répartition de probabilité estimée sur les possibilités testées. L'un des problèmes du système est alors de déterminer cette répartition de probabilité. Il réfléchira par exemple au coup le plus probable de l'adversaire dans un jeu de stratégie, à la probabilité qu'un adversaire ait une carte plus forte dans un jeu de carte etc. Dans la suite de ce rapport, nous nous focaliserons sur les hypothèses portant sur les choix abstraits du système, c'est-à-dire sur des hypothèses contrôlables. Néanmoins, les méthodes décrites sont adaptables aux hypothèses incontrôlables.

2.5. Problématique du raisonnement hypothétique dans le système MARECHAL

Si les choix abstraits permettent en théorie de mieux contrôler la recherche d'une solution, leur gestion dans le cadre de solutions complexes pose en pratique de nouveaux problèmes

[10] Nous parlons ici de jeux simultanés. Dans le cas de jeux alternés comme les échecs, les intentions adversaires peuvent être considérées comme contrôlables puisque l'on peut déterminer en théorie celle qui sera effectivement jouée.

pour le raisonnement hypothétique. En effet, dans les problèmes à solutions simples, c'est-à-dire où il n'y a qu'un seul choix à faire, un choix abstrait à n possibilités engendrera n hypothèses, ce qui est facilement gérable. En revanche, dans les problèmes à solutions complexes, un tel choix engendrera 2^n hypothèses, puisque les possibilités ne sont pas mutuellement exclusives. Plusieurs possibilités peuvent être sélectionnées en même temps, comme par exemple le fait d'attaquer à plusieurs endroits à la fois. Le système doit donc rapidement faire face à une explosion combinatoire et deux approches sont alors possibles.

La première consiste à étudier les n possibilités indépendamment les unes des autres puis à étudier leur combinaison. Cela signifie que les solutions partielles correspondantes à chaque hypothèse devront être combinées pour obtenir une solution globale. Deux solutions partielles peuvent alors être incompatibles a priori car les choix concrets qu'elles contiennent entrent en conflit (typiquement, une pièce ne peut avoir deux chemins de déplacement différents), mais pourtant leur combinaison peut être possible et intéressante si on les modifie plus ou moins (on parlera alors d'affaiblissement ou de dégradation d'une solution partielle). En effet, si v_1 et v_2 sont les meilleures valeurs possibles pouvant être obtenues sous les hypothèses h_1 et h_2, et si v_1' et v_2' sont des valeurs sous-optimales ($v_1'<v_1$ et $v_2'<v_2$) pouvant être obtenues sous ces deux même hypothèses de telle sorte que les solutions partielles sont non conflictuelles, alors on pourra avoir $v_1'+v_2'>v_1$ et $v_1'+v_2'>v_2$. Un tel exemple est montré sur la Figure 5, où le système a trouvé trois zones intéressantes pour une offensive. Comme il n'y a pas de conflit entre la troisième hypothèse et les deux autres, celle-ci pourra être étudiée indépendamment. Par contre, la première hypothèse entre a priori en conflit avec la seconde car certaines pièces sont utilisées dans les deux cas. Il faut donc dans ce cas chercher si une solution combinée existe. Si le nombre de pièces en conflit est faible par rapport au nombre de pièces utilisées pour les deux offensives, il est probable que les deux offensives pourront être construites en se partageant les pièces en conflit. La combinaison de ces deux offensives moins puissantes sera alors certainement plus intéressante qu'une seule des deux.

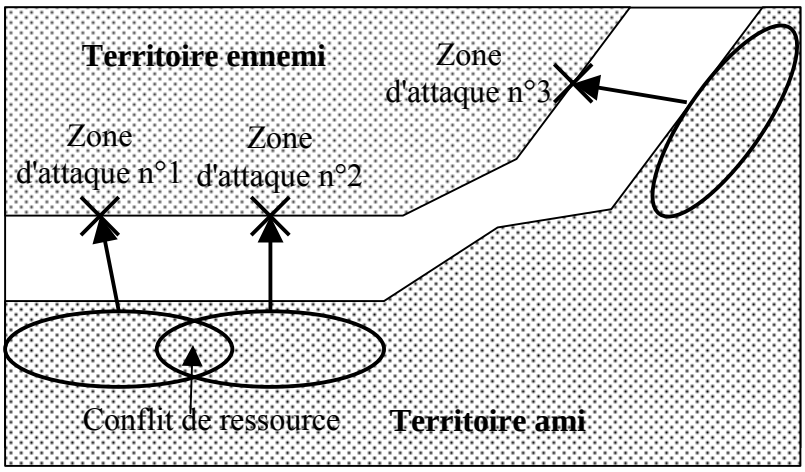


Figure 5 : Exemple de conflit de ressource entre hypothèses

La seconde approche consiste à accepter d'étudier en théorie les 2^n combinaisons de possibilités tout en contrôlant cette étude pour ne parcourir en pratique qu'un nombre raisonnable de solution. Ces deux approches différentes ont été baptisées respectivement « a posteriori » et « a priori », car la première gère la combinaison après avoir étudié les hypothèses alors que la seconde fait les deux en même temps. Le but de ce papier étant de comparer ces deux méthodes, la suite de l'article sera consacrée à décrire et étudier chacune d'elle.

3. Combinaison a posteriori

Face au problème de la combinaison des hypothèses, les méthodes classiques de raisonnement hypothétique doivent être complétées et soutenues par des mécanismes spécifiques. Dans la méthode « a posteriori » décrite dans cette partie, le raisonnement est principalement complété par une phase de combinaison des solutions obtenues par le raisonnement hypothétique.

3.1. Principe de l'approche a posteriori

Dans l'approche que nous avons baptisée « a posteriori », le système commence par étudier indépendamment chaque hypothèse pour un choix abstrait donné. Il va donc poser comme hypothèse la première des différentes possibilités et continuer son raisonnement sous cette hypothèse. Une fois les conséquences déterminées, le système revient au niveau du choix et recommence avec une nouvelle hypothèse. Ce faisant, le système va conserver une trace des hypothèses envisagées et des conséquences évaluées sous la forme d'un arbre (chaque nœud de cet arbre correspond à un choix abstrait et chaque branche à une possibilité pour ce choix).
Lorsque toutes les hypothèses ont été évaluées, le système passe alors en phase de combinaison où il va construire une solution sur la base des informations recueillies lors des tests des hypothèses. Il y a donc ici deux phases totalement distinctes (cf. Figure 6) : la phase d'évaluation qui étudie les hypothèses indépendamment les unes des autres et génère une trace de son exécution et la phase de décision a posteriori qui permet de construire la solution d'après les résultats obtenus. Par rapport à un raisonnement hypothétique classique, il s'agit donc de rajouter une nouvelle étape entre l'évaluation et la sélection finale. Les phases d'évaluation et de sélection étant classiques, nous ne détaillerons dans la suite de cette partie que la phase de combinaison.

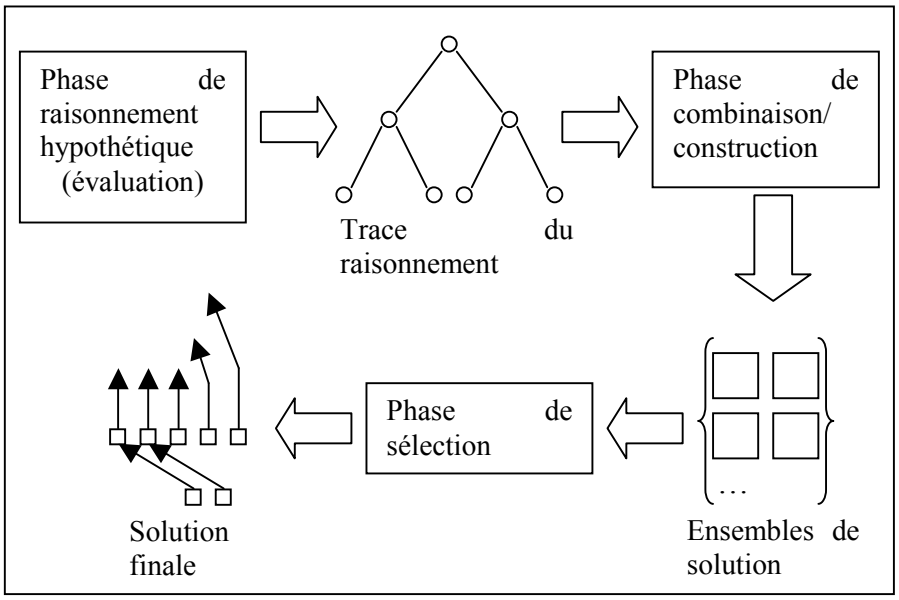


Figure 6 : Principe de l'approche « a posteriori »

3.2. Phase de combinaison

Cette phase consiste à étudier les combinaisons possibles d'hypothèses puis à choisir la solution finale parmi les meilleures combinaisons et les meilleures feuilles de l'arbre des hypothèses. Naturellement, toutes les combinaisons ne peuvent être étudiées en général. Le système sélectionne celles qui lui paraissent les plus intéressantes a priori (il teste les combinaisons des meilleures hypothèses en priorité). Mais, pour une combinaison donnée (i.e., un ensemble de n solutions partielles correspondantes à n feuilles de l'arbre des hypothèses), il existe encore de nombreuses façons d'effectuer la combinaison et celles-ci ne sont pas toutes viables ni équivalentes. En effet, des contraintes d'unicité de ressource (une pièce ne peut avoir deux chemins différents, deux pièces ne peuvent aller au même endroit au même moment etc.) font que la simple addition des solutions partielles donne souvent une solution impossible. Les modifications que l'on peut alors apporter pour rendre la solution viable risquent de changer les conséquences et les intérêts des solutions partielles initiales. Tout le problème est donc de trouver la meilleure façon de combiner l'ensemble de solutions partielles étudié. Pour simplifier les explications, nous supposerons dans la suite que n = 2 (combinaison de deux solutions partielles). Plus formellement, le but de cette recherche est donc de générer deux solutions compatibles s'_1 et s'_2 à partir des solutions incompatibles s_1 et s_2 telles que s'_1 et s'_2 soient les moins dégradées possibles. Il s'agit alors d'un nouveau problème d'optimisation pour le système.

Il existe plusieurs moyens pour supprimer une incompatibilité sans trop dégrader les

solutions. Dans le cas d'un jeu de stratégie, une incompatibilité provient par exemple de l'utilisation de la même pièce dans les deux solutions avec des chemins différents. On peut alors essayer de trouver un chemin remplissant les deux fonctions prévues ou remplacer la pièce dans l'une des solutions par une autre pièce allant au même endroit. Par exemple, sur la Figure 7, les situations de gauche et du milieu sont incompatibles pour noir car elles utilisent toutes deux les pièces 13 et 21[11]. Dans les deux cas, la pièce 13 est utilisée pour contrer la pièce 37. Pour rendre les solutions compatibles, on peut envoyer la pièce 13 à droite de la pièce 7 (mouvements gris de la situation de droite), car à cet endroit, elle attaque la pièce 37 dans les deux situations. Pour la pièce 21, le mieux est de la remplacer par la pièce 23 dans la situation centrale. Cela dégrade un peu la solution car la pièce 23 est moins forte, mais cette dégradation n'est pas grave car l'attaque blanche sera toujours en échec. La situation de droite montre la combinaison des deux solutions obtenues.

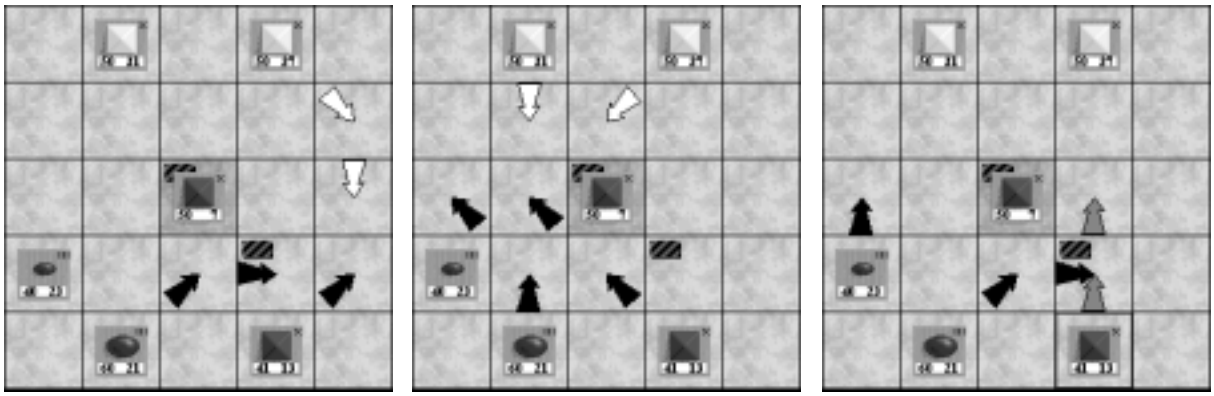


Figure 7 : modification de solutions incompatibles par remplacement

Lorsque ces moyens ne suffisent pas, on peut simplement supprimer l'un des deux chemins en conflit, mais on risque alors de dégrader fortement la solution. Cette méthode ne doit donc être utilisée qu'en dernier recours.

Au total, le système doit faire un ensemble de "méta-choix" portant sur le remplacement ou l'annulation de choix. La difficulté du problème réside dans le fait que ces méta-choix sont dépendants les uns des autres. Le remplacement d'un choix peut ainsi empêcher un autre remplacement. Sur la situation gauche de la Figure 8, on suppose que les pièces 37 et 53 (mouvements blancs) doivent être remplacées pour être utilisées dans une autre solution. Les pièces 35 et 47 peuvent a priori remplacer indifféremment ces deux pièces, mais, en pratique, si on remplace la pièce 53 par la pièce 35 (mouvements gris), la pièce 37 ne peut plus être remplacée par la pièce 47 car celle-ci n'a plus de chemin valide (on suppose que les unités se déplacent ici de deux cases au maximum). Sur la situation droite de la Figure 8, la pièce 35 peut remplacer les pièces 37 ou 47. Si l'on choisit de remplacer 37, la pièce 47 peut ne plus être remplaçable par une autre pièce et l'annulation de son mouvement peut conduire à

[11] Les numéros des pièces sont affichés en bas à droite de chacune d'elles.

dégrader fortement la solution. Il faut alors se demander si la pièce 37 ne pouvait être remplacée par une pièce différente de 35 ou si l'annulation de son mouvement n'est pas moins préjudiciable que celui de 47.
Il n'existe évidement pas de méthode pour obtenir directement la meilleure solution dans tous les cas. Seules des heuristiques plus ou moins imparfaites peuvent guider le raisonnement, et elles nécessitent souvent des retours en arrières. Mais ces retours arrières doivent être limités au maximum car l'exploration exhaustive des possibilités serait beaucoup trop longue. Le nombre de combinaisons de méta-choix possibles est en effet exponentiel en fonction du nombre de pièces concernées et l'évaluation de chaque combinaison est coûteuse. Or, le nombre de combinaisons d'hypothèses que doit étudier le système est élevé et la construction d'une solution combinée doit donc être rapide.

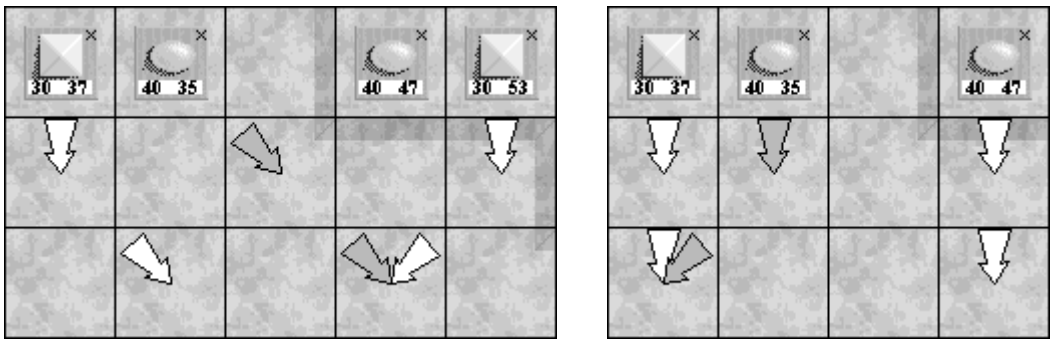


Figure 8 : Exemples de mauvais choix de remplacement

3.3. Implémentation de la phase de combinaison

Ce problème « interne » de recherche de compatibilité possède donc toutes les caractéristiques des problèmes « externes » auxquels s'adresse le système MARECHAL :

- Nécessité de coordonner des choix interdépendants (dans un but d'optimisation)
- Pas de méthode directe
- Recherche exhaustive trop coûteuse
- Possibilité d'heuristiques imparfaites nécessitant d'éventuels retours arrières

Dans un tel contexte, il est alors très intéressant d'utiliser la réflexivité pour appliquer le système à la résolution de ce méta-problème qu'est la recherche de solutions compatibles. Cela permet de faire l'économie d'un mécanisme spécifique et de disposer de toute la puissance du système (retours arrière intelligents, recherche contrôlée etc.). Cela permet en outre de rendre les méta-choix conscients et de fournir la méthode de résolution sous une forme plus déclarative. L'ascension infinie des niveaux méta est empêchée simplement en interdisant au système l'utilisation du raisonnement hypothétique sur les méta-choix. Un tel

mode de raisonnement serait d'ailleurs inadapté ici.
Au lieu d'écrire un mécanisme interne pour rechercher des solutions compatibles, nous avons donc fourni au système un schéma de résolution[12]. Au moment de son appel, les variables $a et $b contiennent les numéros des deux solutions partielles à rendre compatibles. Le schéma est constitué de deux phases, comme le montre la Figure 9 : une phase d'analyse et une phase de choix.

La première phase est une phase d'analyse au cours de laquelle le système répertorie les choix en conflit puis détermine a priori les possibilités de remplacement et d'annulation pour ces choix, en calculant pour chacun un intérêt. Pour cela, le système fait appel à une procédure d'analyse interne (AnalyseIncomp). Cette procédure produit un ensemble de faits de la forme Remplacement(Sit($c) $d $f $e $g) ou Annulation(Sit($c) $d $e). Les premiers permettent d'indiquer que le choix $d de la situation $c peut être remplacé par le choix $e. $f est la ressource en conflit dans le choix $d et $g la ressource équivalente dans $e. Les seconds indiquent qu'il est possible de supprimer le choix $d de la situation $c, $f contenant toujours la ressource en conflit dans le choix $d.

```
// recherche du meilleur compromis de situation compatible à partir
// des situations incompatibles $a et $b
(  Appel[AnalyseIncomp($a $b)]

      // remplacement
   TantQue MeilleurFait[Remplacement(Sit($c) $d $f $e $g)
      Option(RL($f) RL($g))]
   Faire
   (  SuppChoix[$c $d]
         AjoutChoix[$c $e]
         SupprimeFait[Annulation(Sit(?0) ?1 $f)]
   )

      // annulation
   TantQue MeilleurFait[Annulation(Sit($c) $d $e) RL($e)]
   Faire
   (  SuppChoix[$c $d]
   )
).
```

Figure 9 : Schéma de résolution pour la recherche de solutions compatibles

[12] Voir [Pannérec 00] pour la description des formes de connaissance utilisées par le système MARECHAL.

Au cours de la seconde phase, le système se sert des informations produites par la phase d'analyse pour effectuer les méta-choix. Il commence par remplacer au maximum les choix en conflit. Dès qu'il choisit un remplacement, il supprime le choix en conflit, ajoute le choix de substitution et supprime les possibilités d'annulation concernant la ressource en conflit. En effet, si le choix est remplacé, le conflit est résolu et une annulation n'est plus nécessaire. Dans la première boucle « TantQue », RL($f) et RL($g) indique que les ressources en conflits sont également des ressources locales au niveau de la boucle. Ainsi, si un remplacement concernant la ressource $f est effectué en utilisant un choix concernant la ressource $g, les autres possibilités de remplacement pour $f et $g sont supprimées. Un choix ne doit pas en effet être remplacé plusieurs fois et un choix ne peut remplacer plusieurs choix simultanément. A chaque fois que le système cherche à effectuer un remplacement, celui-ci peut échouer au cours de l'ajout du nouveau choix (impossibilité de trouver un chemin par exemple). Comme pendant la résolution d'un problème normal, le système revient en arrière pour effectuer un autre méta-choix[13]. Lorsqu'il arrive à la fin du schéma, le système évalue alors la solution obtenue et peut décider de remettre en question des méta-choix pour tenter d'améliorer cette solution.

La principale difficulté de ce schéma réside dans les instructions « SuppChoix » et « AjoutChoix ». Ces instructions doivent répercuter les modifications de solution sur les situations associées et toutes les variables d'états doivent être mises à jour sans les recalculer complètement. D'autre part, il est difficile pour le système de trouver des retours arrières intéressants à partir d'une première solution car peu de liens existent entre les méta-choix et la valeur de la solution obtenue.

3.4. Forces et faiblesses de l'approche « a posteriori »

Afin de comparer l'approche décrite dans cette partie avec celle qui sera décrite dans la partie suivante, nous étudierons les caractéristiques de chacune en terme de rapidité d'exécution et de qualité de solution obtenue.

3.4.1. Temps de calcul

L'idée sous-jacente de l'approche « a posteriori » est de réduire la complexité du problème par une décomposition. Le fait de tester les hypothèses indépendamment puis d'étudier les combinaisons permet de diminuer la complexité du problème : le problème initial à m choix se décompose en n sous-problèmes de m_i choix chacun (pour tout $i<n$, $m_i<m$). La phase de raisonnement proprement dite (tests des hypothèses) s'en trouve donc largement réduite. Le problème est de savoir si le sur-coût introduit par l'adjonction d'une phase de combinaison est

[13] Voir [Pannérec 00] pour les mécanismes de retour arrière sur les choix concrets.

inférieur au gain sur la phase de raisonnement proprement dite.
Ce sur-coût dépend en fait du degré d'indépendance entre les hypothèses. Lorsque les hypothèses sont indépendantes les unes des autres (cas d'offensives à deux endroits éloignés de la carte ne faisant pas intervenir les mêmes pièces) le sur-coût est quasi nul : Il n'y a alors pas de conflit entre les solutions partielles et leur combinaison est donc triviale. Dans ce cas, l'approche « a posteriori » est donc très intéressante en terme de rapidité de calcul.
Par contre, lorsque les hypothèses sont fortement dépendantes (cas d'offensives proches et faisant donc intervenir les mêmes pièces), nous avons vu que la résolution de la combinaison des solutions partielles était un méta-problème au moins aussi difficile que le problème de base. En pratique, les hypothèses sont rarement indépendantes et le sur-coût induit par la phase de combinaison est supérieur au gain de la phase de raisonnement proprement dite.

3.4.2. Qualité des solutions obtenues

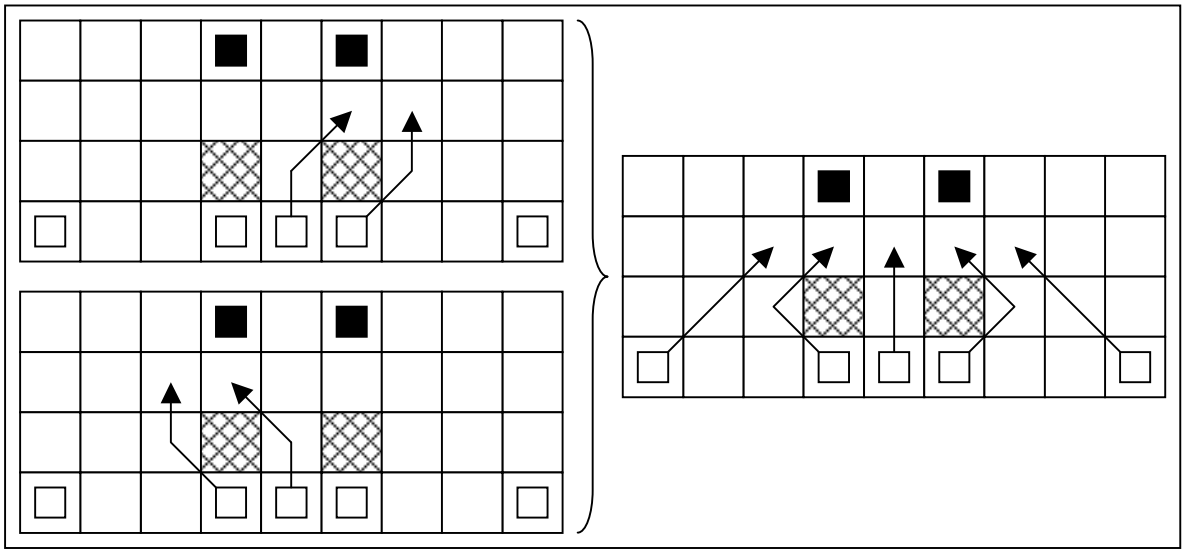


Figure 10 : Exemple de combinaison qui modifie profondément les structures de la solution

Nous avons vu que la combinaison a posteriori de deux solutions partielles était un problème difficile. En pratique, la méthode basée sur la réflexivité que nous avons implémentée ne permet pas, en général, d'obtenir des solutions optimales car la solution combinée ne peut souvent pas se déduire des deux solutions partielles. Par exemple, sur la Figure 10, les structures des deux solutions partielles de gauche ne se retrouvent pas dans la solution finale de droite. Cette dernière est unique car aucune autre solution ne permet de prendre les deux pièces noires simultanément[14].

[14] Dans cet exemple, les cases grisées sont infranchissables et toutes les pièces blanches se déplacent de deux cases. La pièce blanche centrale est supposée plus forte que les autres.

La méthode présentée dans cette partie, qui consiste à construire la solution finale à partir des deux solutions partielles, ne peut donc s'appliquer dans ces cas. Si on cherche à compliquer la méthode de combinaison pour pallier ce problème, on tombe inévitablement sur la nécessité de recommencer le raisonnement en prenant pour hypothèse la conjonction des deux hypothèses à combiner. Il n'y a alors plus indépendance dans l'étude des hypothèses et cela nous conduit à l'approche « a priori » qui est présentée dans la quatrième et dernière partie de ce rapport.

4. Combinaison a priori

Pour résoudre le problème de la combinaison d'hypothèses, nous avons vu dans la partie précédente qu'on pouvait utiliser une approche basée sur l'étude des hypothèses indépendamment les unes des autres et sur une phase de combinaison a posteriori. Dans cette partie, nous allons décrire puis analyser l'approche opposée, qui consiste à étudier les combinaisons des hypothèses en même temps que ces hypothèses sont envisagées. Pour ce faire, celles-ci ne seront plus envisagées indépendamment les unes des autres. Cette deuxième approche est actuellement en cours d'implémentation dans le système MARECHAL pour remplacer celle de la partie précédente.

4.1. Principe

Le principe de l'approche « a priori » consiste donc à combiner les hypothèses lors de leur étude. En théorie, le système passe ainsi en revue toutes les possibilités de combinaisons selon une exploration combinatoire. Cela signifie que dans la trace de raisonnement générée, chaque choix abstrait sera représenté par un sous arbre et non par un nœud simple comme c'était le cas dans l'approche précédente (cf. Figure 12). On voit tout de suite que la trace est beaucoup plus fournie et en pratique, il est nécessaire de limiter l'exploration de même qu'il était nécessaire de limiter le nombre de combinaisons à envisager dans l'approche précédente. Le système utilise donc des mécanismes de contrôle pour explorer intelligemment une partie de l'arbre comme nous allons le voir dans la sous-partie suivante.

4.2. Implémentation

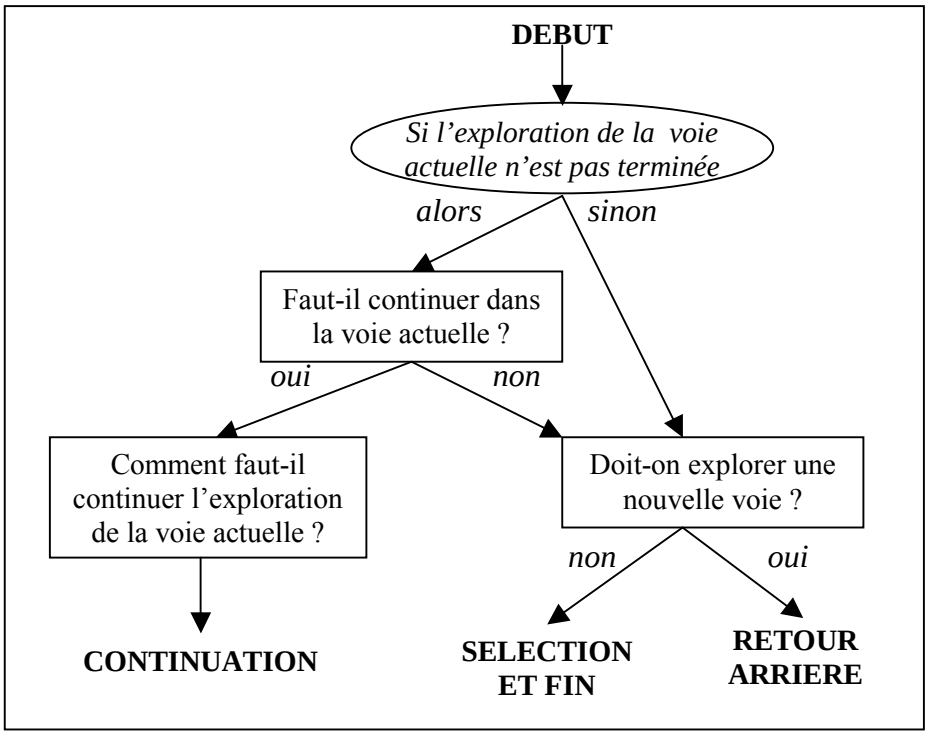


Figure 11 : Algorithme général de contrôle du raisonnement hypothétique

Le contrôle du raisonnement hypothétique fait partie des capacités de monitoring d'un système[15]. C'est en effet une activité de niveau méta qui, à partir de l'étude du raisonnement déjà effectué, guide l'exécution de la suite du raisonnement. Dans le système MARECHAL, ce monitoring se matérialise à travers trois questions que se pose en permanence le système, comme le montre l'algorithme général de contrôle présenté sur la Figure 11. Cet algorithme est exécuté à chaque nouveau nœud dans la trace de la Figure 12. La suite de cette sous-partie discute chaque question en particulier.

[15] Voir [Pitrat 99] pour une étude du monitoring chez les êtres humains et, plus généralement, [Flavell 76] pour une présentation de la métacognition. Voir [Pitrat 91], [Pitrat 00] et [Pollock 89] pour l'utilisation du monitoring dans les systèmes d'Intelligence Artificielle.

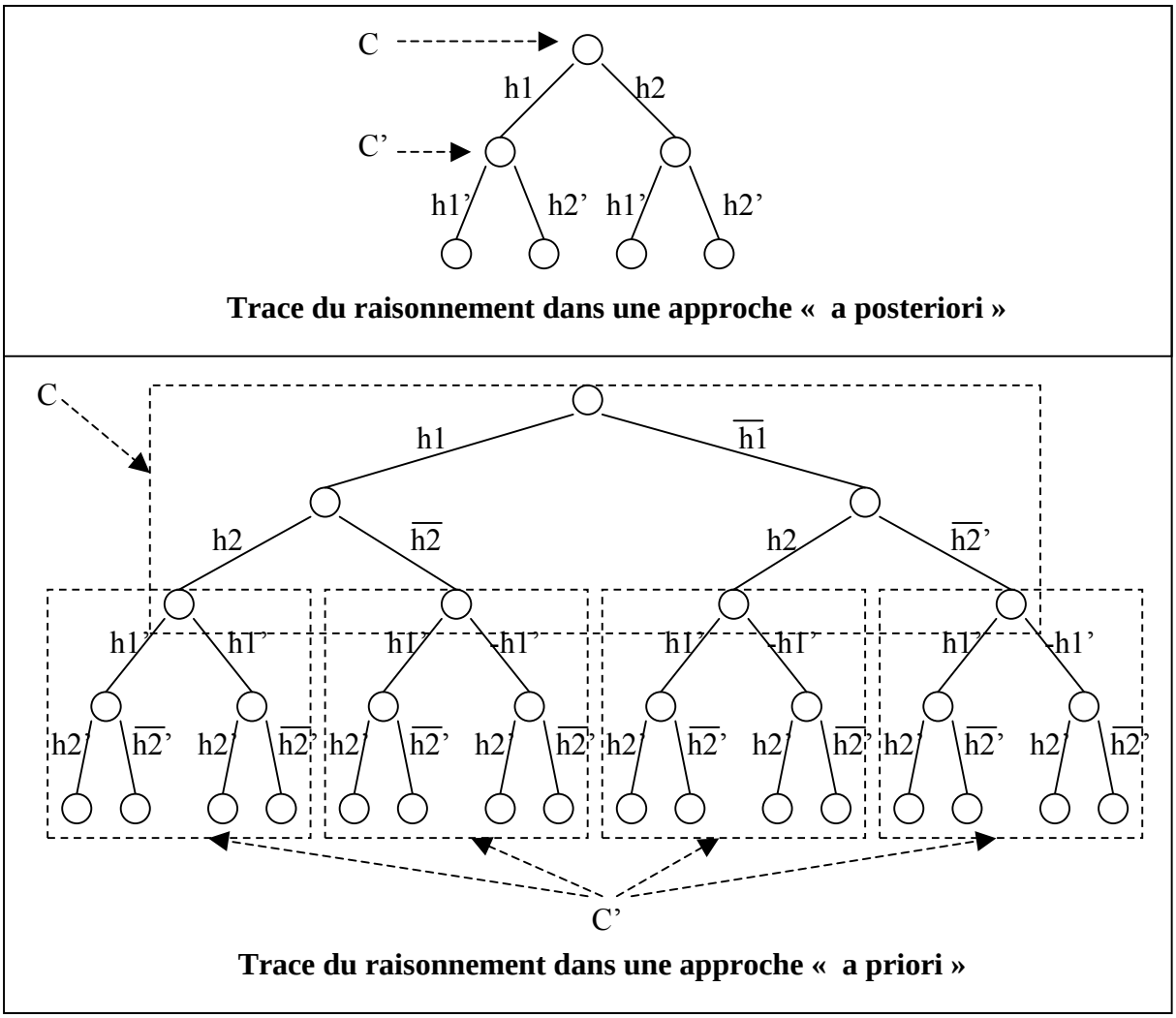


Figure 12 : Comparaison des traces des deux approches décrites dans le cas de deux chois abstraits (C et C') à deux possibilités ((h1,h2) et (h1',h2')). $\overline{hx}$ signifie que l'on se place dans le cas où l'on n'utilise pas la possibilité x (négation de hx).

4.2.1. Faut-il continuer dans la voie actuelle ?

A travers cette question, le système est face à un problème classique du monitoring qui consiste à anticiper les résultats des voies de recherche pour éviter de s'acharner dans une impasse ou de perdre trop de temps sur des choses peu importantes (voir [Pitrat 00]). Pour cela, il doit tout d'abord disposer d'un encadrement du résultat attendu : la borne supérieure permet de savoir si une bonne solution peut être trouvée dans cette voie. Si elle est inférieure à des résultats déjà obtenus par d'autres voies, il n'est évidemment pas nécessaire de continuer. La borne inférieure permet, quant à elle, d'estimer la probabilité de trouver une solution intéressante dans la voie de recherche actuelle. Cette probabilité peut en effet influer sur la décision : par exemple, si la meilleure solution obtenue a une valeur de 100, les

encadrements [0 110] et [100 110] ne seront pas équivalents. Pour anticiper le résultat, le système se base sur l'évaluation de la solution partielle courante et sur les possibilités d'augmentation ou de dégradation à venir. Cette estimation est un point très délicat car il n'est pas facile de prévoir les possibilités à venir.
Le système doit également disposer d'un encadrement du temps nécessaire pour terminer l'exploration de la voie courante. Si le gain anticipé semble faible a priori, la perspective d'une recherche encore longue pourra conduire à stopper l'exploration de cette voie.
C'est donc à partir de la confrontation de ces deux encadrements aux résultats déjà obtenus et au temps de recherche encore disponible que le système va prendre la décision de continuer dans la voie de recherche actuelle ou, au contraire, de chercher une autre voie de recherche. Naturellement, le système ne dispose pas toujours de ces informations d'encadrement ou le temps d'obtention de celles-ci peut être trop élevé pour que leur détermination soit systématique. Dans ces cas, le système continue alors par défaut la voie actuelle. En outre, l'action de changer de voie est coûteuse et doit donc s'effectuer avec parcimonie, uniquement lorsque le système est sûr de sa décision.

4.2.2. Comment continuer dans la voie actuelle ?

Si le système décide de continuer dans la voie de recherche courante, il doit alors déterminer comment le faire. En effet, à chaque choix abstrait rencontré, plusieurs possibilités sont disponibles et le système doit sélectionner la plus intéressante. Cela peut être celle qui a le plus de chance d'être présente dans la solution finale (pour éviter d'avoir à revenir dessus a posteriori) ou celle qui est la plus rapide à étudier pour limiter l'espace de recherche. De même que pour la question précédente, le système doit disposer, pour chaque possibilité, d'un encadrement du résultat anticipé et d'un encadrement du temps de recherche nécessaire. Néanmoins, l'encadrement du résultat attendu est fondamentalement différent car il se fait ici d'après le choix (l'estimation est « a priori ») alors qu'il se faisait d'après la solution déjà obtenue (estimation « a posteriori ») dans la question précédente. Cette estimation est obtenue d'après les connaissances du système sur le domaine (par exemple, « il est bon d'attaquer les pièces les plus faibles de l'adversaire ») et d'après le passé du raisonnement. Ainsi, si la possibilité envisagée a déjà été testée dans une autre voie de recherche et n'a pas fonctionné, l'intérêt de cette possibilité sera diminué. Au contraire, certains choix peuvent être faits automatiquement s'ils ont bien fonctionné dans une autre voie.
A ce mécanisme de sélection, il convient d'ajouter un mécanisme de gestion des hypothèses indépendantes qui utilise de nouveau le passé du raisonnement pour accélérer la recherche. Si, par exemple, le programme a déjà développé l'arbre de la Figure 13 et déterminé que ¬b était préférable à b tout en étant indépendante de a, il pourra directement appliquer ¬b dans l'hypothèse ¬a et se contenter de recopier la solution partielle correspondante à ¬b, sans recommencer cette partie de raisonnement.

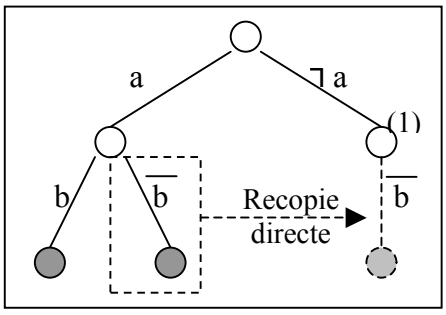


Figure 13 : Gestion d'hypothèses indépendantes

4.2.3. Faut-il explorer une autre voie ?

La question d'une nouvelle voie est la plus importante des trois questions de contrôle. C'est en effet elle qui va diriger la recherche combinatoire par l'intermédiaire d'un mécanisme de retour arrière. A chaque fois que le programme termine une voie de recherche (par arrêt prématuré ou non), il lui faut en effet déterminer à quel niveau du raisonnement il va remonter pour explorer une nouvelle voie, c'est-à-dire, quelle hypothèse il va remettre en question.

Sur le schéma de la Figure 14, les cercles représentent les moments du processus de raisonnement pendant lesquels le système a posé une hypothèse. Au point (1), il décide de changer de voie de recherche et remonte au point (2) où il pose comme hypothèse la négation de la précédente (i.e. qu'il ne faut pas utiliser la possibilité de choix précédemment retenue). Cela le conduit alors sur une nouvelle voie de recherche (3). Naturellement, cela signifie que la voie (4) ne pourra plus être considérée car l'hypothèse qu'il faudrait remettre en question ne fait plus partie de l'ensemble des hypothèses actuelles. Les hypothèses de non utilisation de possibilité sont dites négatives alors que les hypothèses d'utilisation sont dites positives.

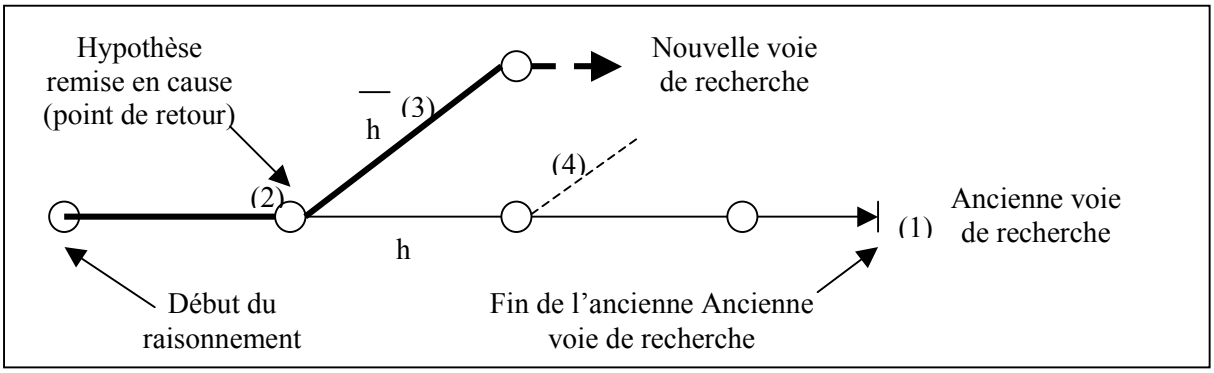


Figure 14 : Exploration combinatoire par retour arrière

Pour trouver le bon point de retour, le système calcule un intérêt d'annulation pour chaque hypothèse positive actuelle. Cet intérêt sera d'autant plus grand que l'hypothèse aura « gêné » la suite du raisonnement et eut un impact négatif ou faible sur la valeur de la solution finale.

Par exemple, si une hypothèse conduit à utiliser des pièces[16] qui auraient été utiles plus loin dans le raisonnement et qu'elle n'a pas un impact bénéfique sur la solution, son annulation sera a priori très intéressante. Selon le temps de réflexion restant, le système va ensuite déterminer un niveau d'intérêt minimum de remise en cause. Le point de retour sera alors l'hypothèse la plus récente dont l'intérêt est supérieur au niveau de remise en cause. Si une telle hypothèse n'existe pas, le raisonnement est conclu par la phase de sélection. La difficulté consiste donc ici à pouvoir déterminer l'impact d'une hypothèse sur la valeur d'une solution et sur les autres hypothèses.

4.3. Forces et faiblesses

De même que dans la partie précédente, les performances de la méthode décrite dans cette partie dépendent du degré de dépendance existant entre les hypothèses.

4.3.1. Temps de calcul

Dans le cas d'hypothèses indépendantes, le temps de calcul nécessaire avec la méthode « a priori » est supérieur à celui de la méthode « a posteriori », même avec le mécanisme de gestion des hypothèses indépendantes que nous avons décrit. En pratique, cela ne pénalise guère la méthode « a priori » car la différence n'est pas énorme et les hypothèses sont de toute façon rarement indépendantes.
En ce qui concerne les hypothèses dépendantes (cas normal de fonctionnement), la méthode décrite dans cette partie est largement plus rapide en théorie car le programme ne perd pas de temps avec un état intermédiaire (les solutions des hypothèses étudiées séparément) qui est souvent inutilisable. Les résultats expérimentaux devront confirmer cette thèse.

4.3.2. Qualité des solutions obtenues

En terme de qualité, les solutions obtenues avec des hypothèses indépendantes sont, bien sûr, rigoureusement identiques. En ce qui concerne les cas normaux où les hypothèses ne sont pas indépendantes, la seconde méthode fournit en théorie des solutions de meilleure qualité surtout si le temps de recherche est élevé. Elle peut en effet parcourir plus systématiquement l'espace des possibilités alors que la méthode « a posteriori » ne peut atteindre qu'une partie restreinte de cet espace. Même avec un temps plus court, la méthode « a priori » fournie des

[16] Une hypothèse étant la sélection d'une possibilité pour un choix abstrait, elle engendre des choix concrets et conduit donc à une modification de la solution courante (i.e., ajout de mouvement dans le cas d'un jeu de stratégie).

solutions plus cohérentes. Là encore, les résultats devront confirmer ces affirmations.

4.3.3. Conclusion sur la comparaison des deux méthodes

En conclusion, on peut affirmer que la méthode « a priori » est meilleure que la méthode « a posteriori », sauf dans le cas particulier d'hypothèses fortement indépendantes où elle est légèrement plus lente. Ces deux méthodes étant donc complémentaires, on pourrait penser à utiliser l'une ou l'autre selon la circonstance. En pratique, cela n'est pas envisageable, car il est impossible de savoir a priori si deux hypothèses sont indépendantes ou non. Ce sont en effet les conséquences des hypothèses qui entrent en conflit et le raisonnement doit avoir eu lieu pour pouvoir étudier ces conséquences.

5. Conclusion

Dans ce papier, nous avons montré que la combinaison d'hypothèses était incontournable dans les problèmes d'optimisation à solution complexe. Nous avons alors décrit et comparé sur le plan théorique deux méthodes permettant de répondre à ce problème de combinaison. La conclusion de cette comparaison est que la méthode consistant à combiner les solutions au cours de leur construction est globalement la meilleure tant au niveau de la rapidité que de la qualité des solutions obtenues. Celle-ci est donc en cours d'implémentation dans le système MARECHAL pour remplacer la précédente. Les objectifs à court terme sont d'obtenir des résultats expérimentaux qui valideront la comparaison théorique réalisée.
Naturellement, le point important de cette méthode réside dans le mécanisme de contrôle et dans les heuristiques qui s'y trouvent. Ce mécanisme n'est pas encore optimal et les perspectives à long terme consistent donc à le compléter et l'améliorer.

6. Bibliographie

[De Kleer 86] De Kleer J. : *An Assumbtion-based TMS*. Artificial Intelligence 28, p. 127-162, 1986.

[Glover 89] Glover F. : *Tabu search - Part 1*. ORSA journal on computing, vol 1-3, p190-206, 1989.

[Glover 90] Glover F. : *Tabu search - Part 2.* ORSA journal on computing, vol 2-1, p4-32, 1990.

[Goldberg 89] Goldberg D. : *Genetic algorithms in search, optimisation and machine*

learning. Addison Wesley, 1989.

[Grolimund 97] Grolimund S. : *Apprentissage de connaissances de contrôle pour l'optimisation combinatoire : intégration du raisonnement à partir de cas dans la méthode tabou*. Thèse de université Paris 6, 1997.

[Lobjois 99] Lobjois L. : *Problèmes d'optimisation combinatoire sous contraintes : vers la conception automatique de méthodes de résolution adaptées à chaque instance*. Thèse de l'ENSAE, 1999.

[Pitrat 91] Pitrat J. : *An intelligent system must and can observe his own behavior*. Cognitiva 90, pages 119-128, Elsevier Science Publishers, 1991.

[Pitrat 99] Pitrat J. : *Une expérience de monitoring*. Rapport de recherche LIP6 1999/014, 1999.

[Pitrat 00] Pitrat J. : *Monitorer la recherche d'une solution*. Actes du colloque Intelligence Artificielle de Berder, pages 3-15, rapport interne LIP6 n°2000/002, 2000.

[Pollock 89] Pollock J.L. : *OSCAR : a general theory of rationality*. Journal of Experimental Artificial Intelligence, vol 1, pages 209-226, 1989.

[Schiex et al. 97] Schiex T., Fargier H. et Verfaillie G. : *Problèmes de satisfaction de contraintes valués*. Revue d'Intelligence Artificielle 11(3), pages 339-373, 1997.

[Van Laarhoven et al. 88] Van Laarhoven P. et Aarts E. : *Simulated annealing : Theory and applications*. Kluwer, 1988.

Chemins détournés, idées fausses et bonnes idées

Dominique Pastre
Crip5 - Université René Descartes

Cet article est dédié à la mémoire de Jean-Marc Fouet

Résumé : Dans le but d'améliorer les systèmes d'intelligence artificielle, il peut être intéressant d'imiter le raisonnement humain. Pour essayer de comprendre comment l'être humain raisonne, il peut être utile de l'observer. Après une auto-observation de quelques mois pendant la résolution d'une vingtaine de problèmes, cet article décrit et analyse les recherches effectuées, essaie de comprendre ce qui a conduit aux bonnes idées et essaie de montrer comment même ce qui était faux ou pourrait sembler inintéressant a pu aider à progresser vers les solutions.

Mots-clés : auto-observation, résolution de problèmes, logique, jeux mathématiques, puzzles

1. Introduction

Les méthodes utilisées en Intelligence artificielle dans la résolution automatique de problèmes peuvent être très différentes de celles utilisées par l'être humain et sont parfois beaucoup plus efficaces. On peut aussi choisir au contraire de définir des méthodes imitant le comportement de l'être humain, par exemple pour faire de la simulation, ou si la machine doit dialoguer avec l'être humain ou lui expliquer sa résolution. On peut aussi s'inspirer de ce que fait l'être humain, en particulier de ses heuristiques et de son expertise si l'on ne connaît pas de méthodes pour résoudre certains problèmes. Néanmoins, ceci est également difficile car on ne connaît pas bien ces heuristiques et cette expertise qui sont en grande partie inconscientes. Un expert n'est souvent pas capable de communiquer sa propre expertise « à froid ». Pour cela, il doit, sans que cela soit toujours suffisant, être mis en situation et être observé en train de résoudre des problèmes. De nombreux chercheurs ont effectué de telles observations [Pastre 78, Chi 81, Schoenfeld 85]. A défaut de pouvoir observer quelqu'un d'autre, il est aussi possible de s'auto-observer. Ces observations sont nécessairement un peu biaisées car il est difficile d'être à la fois l'observé et l'observateur, et d'effectuer, en plus de la tâche de résolution de problèmes la tâche d'observation. Néanmoins, ces observations peuvent être très utiles pour l'Intelligence Artificielle car, même si elles ne donnent pas exactement ce qui s'est passé dans la tête de l'observé, elles montrent des cheminements plausibles dont il est

raisonnable de s'inspirer pour résoudre automatiquement, expliquer, dialoguer. Il est même possible, et utile !, de s'observer en train de s'observer [Pitrat 99].

Les problèmes dont la résolution est analysée dans cet article sont des problèmes de logique ou de mathématiques ne nécessitant pas en général de connaissances mathématiques très approfondies, mais nécessitant parfois une recherche assez longue. Si certains ont été résolus en quelques heures, ou même en quelques minutes, d'autres ont nécessité plusieurs jours, voire plusieurs semaines. Dans ces cas-là, il y a, bien sûr, en plus, maturation entre deux périodes de réflexion. Dans ces conditions, seule une auto-observation était envisageable. Il m'était aussi, même pour une auto-observation, difficile de trouver le temps de passer toutes ces heures et ces journées, tranquillement assise devant mon bureau avec papier-crayon pour tout noter et magnétophone pour enregistrer une réflexion à voix haute. Je devais au contraire profiter de tout moment où mon esprit n'était pas occupé par une autre tâche pour réfléchir, quelquefois avec la possibilité de griffonner quelques équations[17] ou quelques dessins, d'autres fois purement mentalement[18]. Je faisais néanmoins l'effort, dès que possible et chaque fois que cela était nécessaire, de noter rapidement par écrit ce que j'avais fait ou envisagé et pourquoi, le cheminement de ma pensée, même si les idées ou les techniques employées n'avaient pas abouti, même si ce qui avait été fait ou envisagé ne semblait pas intéressant. Je ne dispose donc pas de protocoles complètement détaillés, mais de résumés qui donnent une bonne idée de mes recherches.

On trouvera dans cet article des résumés de ces résumés dans lesquels j'ai essayé de dégager les points importants de mes recherches, qui m'ont conduit ou ne m'ont pas conduit à des solutions, sans omettre les impasses ni les grosses bêtises. On verra d'ailleurs, et c'est un phénomène qui m'a semblé important et qui a donné le titre de l'article, qu'à côté des stupidités impardonnables et sans intérêt, il y a des erreurs ou des inexactitudes qui n'ont certes pas conduit à la solution, mais à des idées qui, elles, ont conduit à la solution. Il ne s'agit donc pas de rédactions de preuves les plus courtes, les plus simples, les plus élégantes possible qui seraient données s'il ne s'était agi « que » de donner les solutions des problèmes et elles ne doivent pas être considérées comme des modèles de preuves, ce n'était pas le but. Outre l'élagage de tout de qui n'est pas nécessaire directement à la preuve, un travail de simplification et de clarification de notations serait alors à faire.

Après avoir donné l'origine et le contexte des problèmes analysés (section 2), les grosses bêtises dont je suis coupable (section 3), j'analyserai des problèmes pour lesquels de

[17] devant une table, ou, plus inconfortablement dans le métro ou pendant une réunion qui traîne en longueur

[18] en marchant ou avant de m'endormir …

premières idées se sont révélées fausses mais ont néanmoins conduit à de bonnes idées (section 4) et des problèmes dans lesquels les bonnes idées sont apparues après des chemins détournés et inutilement compliqués (section 5). On verra également des cas où la mémoire a joué un rôle positif et des cas où elle n'a servi à rien (section 6). Enfin d'autres problèmes seront évoqués rapidement en sections 7 et 8.

2. Sources des problèmes

Les problèmes analysés proviennent des trois origines suivantes.

2.1. Rubrique AFFAIRE DE LOGIQUE du journal Le Monde du lundi (daté mardi)

Ces problèmes sont de difficulté très inégale. Certains se résolvent en quelques minutes. D'autres ont demandé beaucoup d'efforts pour être résolus avant la parution de la solution le lundi suivant. Pour l'un d'eux enfin (section 4.1), seuls le cas particulier facile et la formule dans le cas général étaient trouvés au bout d'une semaine, aussi bien par moi-même que dans la solution du Monde qui, après la formule, ajoutait « Il semblerait qu'on ne puisse pas faire mieux (si des lecteurs peuvent le prouver, qu'ils nous écrivent).» Une deuxième semaine de recherche m'a été nécessaire pour finir de démontrer qu'on ne pouvait pas faire mieux.

2.2. Championnats des jeux mathématiques et logiques (½ finales 1999 et 2000)

La résolution des problèmes de ces jeux est très différente dans un premier temps puisqu'il s'agit de résoudre en trois heures douze problèmes, de difficultés a priori croissantes et qu'il faut donner en guise de réponse, le nombre de solutions pour chaque problème et deux des solutions (éventuellement une seule ou zéro). Il faut donc gérer correctement son temps, non seulement résoudre les problèmes mais aussi évaluer le nombre de solutions, ne pas faire d'étourderies ni dans la résolution, ni dans la recopie des solutions. Dans un deuxième temps, on a tout le loisir de reprendre les problèmes chez soi, de résoudre ceux qu'on avait pas réussi à résoudre, et de découvrir les fautes que l'on a faites dans quelques autres (on est aidé par cela par la différence entre le score obtenu et le nombre de problèmes que l'on croyait avoir résolus correctement …).
Pour les problèmes 2000, je dispose des brouillons complets, que je m'étais imposée, cette année, de faire proprement, sans mélanger les problèmes.

2.3. Page Web de Jean-Marc Fouet (http://www710.univ-lyon1.fr/~fouet/Dir_lic_mai/TP/puzzles.html)

C'est la source d'un des problèmes (section 6.3) qui est assez classique et que j'avais déjà eu l'occasion de rencontrer. On verra pourquoi la mémoire m'a été de peu de secours et pourquoi je pense qu'elle me sera utile dorénavant.

3. Le bêtisier

Dans cette section, on ne trouvera que des problèmes des Jeux mathématiques où le temps limité m'a permis de ne pas oublier quelques grosses erreurs.
Les problèmes 3.1, 3.3 et 3.4 montrent des étourderies stupides, le problème 3.2 un raisonnement faux car trop rapide. Dans le problème 3.4, une grosse étourderie initiale a été difficile à trouver, c'est souvent le cas des erreurs des plus grossières. Dans ce même problème, on verra comment des simplifications et des vérifications sur des cas particuliers simples ont aidé à la résolution, comment une petite erreur a été trouvée après le résultat, ainsi que quelques petites inexactitudes sans conséquence sur le raisonnement.

3.1. La famille Septime (Jeux mathématiques – ½ finales 1999)

Monsieur et Madame Septime ont sept enfants nés, curieusement, tous les sept un 7 juillet. Chaque année, pour leur anniversaire, Madame Septime offre à chacun un gâteau comportant autant de bougies qu'il a d'années. Jean Septime, le plus jeune, se souvient qu'il y a cinq ans, il y avait, au total, deux fois moins de bougies que cette année. Combien de bougies seront allumées cette année ?

Réponse : 35. Non ! Il y en avait bien 35 il y a cinq ans, mais on demande combien il y en a cette année (70) …
C'est une erreur qu'une machine ne ferait pas !
Raison de l'erreur : le calcul se fait naturellement à partir du nombre d'il y a cinq ans, plutôt que l'inverse.

Pour qu'une réponse soit correcte, il ne suffit pas de résoudre le problème, il faut aussi répondre exactement à la question posée.

3.2. Embrouille sur la feuille de matches (Jeux mathématiques – ½ finales 2000)

Pendant les derniers matches avant la finale de la coupe de basket, nous avons bien vu dans les tribunes un espion de notre future équipe adverse. Il notait notre technique habituelle. Il nous faut perturber les repères qu'on a pu prendre. Nous avons donc décidé de redistribuer nos cinq maillots numérotés de façon qu'aucun de nous ne porte son numéro habituel.
De combien de manière pouvons nous effectuer cette redistribution ?

Réponse en 4 minutes, le jour des Jeux, il me reste le brouillon suivant :

5 maillots

1	4 possibilités	4×3×2 = 24
2	3	
3	2	
1		

perm = 5 ! = 120

C'est faux : l'ancien numéro du 2ème peut être égal à celui qu'on a donné au 1er.

Un raisonnement correct a été fait plus tard :

- d'abord directement : avec des maillots initiaux a b c d e

1er joueur : 4 possibilités, par exemple b

2ème joueur : ou bien a, celui du 1er, il reste c d e à distribuer, soit 2 possibilités

ou bien un autre parmi 3, par exemple c, soit b c … alors

3ème joueur : ou bien a, il reste d e à distribuer, une seule possibilité

ou bien d ou e (2 possibilités) et le reste est imposé

Finalement on a 4×(2 + 3×(1+2)) = **44**

Comme je ne suis pas bien sûre, je fais un deuxième raisonnement avec les complémentaires :

Après quelques tâtonnements, soient a_n le nombre de possibilités pour n maillots et c_n le complémentaire, on a

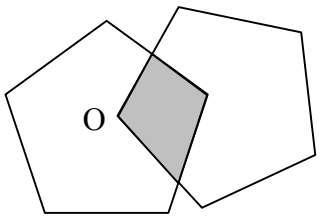
$a_n = n ! - c_n$

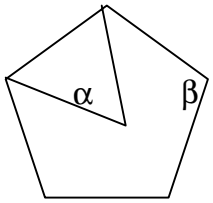
$c_n = n\, a_{n-1} + C_n^2\, a_{n-2} + C_n^3\, a_{n-3} + \ldots + C_n^{n-1}\, a_1 + 1$ (Rem : $a_1 = 0$)

Alors, on retrouve, avec $a_1 = 0$, $c_1 = 1$, $c_2 = 1$, $a_2 = 1$, $c_3 = 4$, $a_3 = 2$, $c_4 = 15$, $a_4 = 9$, $c_5 = 76$, $\mathbf{a_5 = 44}$

3.3. Le radar du Pentagone (Jeux mathématiques – ½ finales 2000)

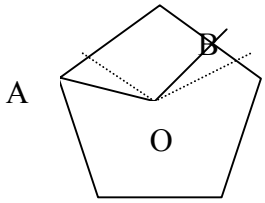
Le siège de l'état-major de l'armée des Etats –Unis est un bâtiment en forme de pentagone régulier, dit le Pentagone. Les services secrets viennent d'y installer un radar révolutionnaire dont la zone de détection, qui couvre également l'extérieur du bâtiment, est un pentagone identique tournant autour d'un sommet situé au centre du Pentagone. Quel est, au maximum le pourcentage de la surface du Pentagone couvert par la zone de détection du radar, en gris sur la figure ? On prendra si besoin est 2,236 pour $\sqrt{5}$, et on arrondira à l'entier le plus proche.





La première des choses à faire est d'évaluer les angles α et β.
On a $\alpha = \frac{2\pi}{5}$ et $\beta = \frac{3\pi}{5}$

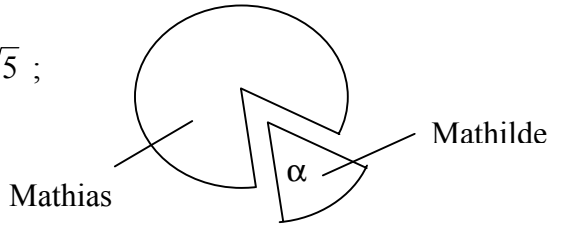
C'est élémentaire mais beaucoup d'erreurs pour ce calcul et un temps déraisonnable pour trouver β le jour des Jeux.
Beaucoup de dessins mais le dessin correct suivant n'a été trouvé que plus tard.



OB est perpendiculaire au côté du Pentagone.
En évaluant la surfaces des petits triangles, on voit
Que la position AOB correspond au maximum et le
pourcentage est donc 1,5/5 =30%

3.4. Les deux cônes (Jeux mathématiques – ½ finales 2000)

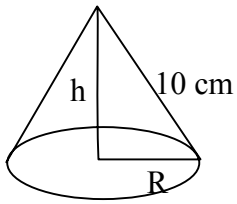
Mathilde et Mathias sont au collège. Leur professeur a donné à chaque groupe de deux élèves un disque de papier de 20cm de diamètre. La consigne est de découper le disque en deux secteurs et d'en faire deux cônes en rapprochant les bords coupés.
Chacun s'affaire en suivant son bon plaisir. Ainsi Mathilde a découpé un secteur et elle donne la partie restante à Mathias (voir figure).
Soigneusement, du papier adhésif est collé pour achever la mise en forme des deux cônes. Ils peuvent alors constater que le cône de Mathilde est deux fois plus haut que celui de Mathias !
Quelle est la valeur de l'angle α du secteur découpé par Mathilde, arrondi au degré le plus proche ?
Pour d'éventuels calculs, on prendra 2,236 pour $\sqrt{5}$;
2,646 pour $\sqrt{7}$; 3,317 pour $\sqrt{11}$;
3,306 pour $\sqrt{13}$ et 3,1416 pour π



Recherche

Il s'agit d'évaluer le rapport des hauteurs h et h' des deux cônes, sachant que l'on veut avoir $\frac{h}{h'} = 2$.

Il est immédiat que l'on a $h^2 + R^2 = 100$ et $h'^2 + R'^2 = 100$ où R et R' sont les rayons des cercles de base.



D'autre part, la circonférence du cercle de base est égale à la longueur de l'arc de cercle du secteur d'angle α. Si on exprime α en radians, on a $2\pi R = 10\alpha$ et $2\pi R' = 10(2\pi - \alpha)$

On pourrait peut-être exprimer α autrement qu'en radians, on pourrait peut-être ne pas considérer ce 10 qui n'est pas significatif, mais je crois gagner du temps en fonçant brutalement.

En éliminant α puis R', on arrive facilement à une équation du second degré en R, pas très sympathique

$$R^2 \left[4 \left(\frac{2\pi}{\alpha} - 1 \right)^2 - 1 \right] = 300$$

Connaissant R, on trouverait α.

Je tire un trait et je recommence, je vais plutôt chercher directement une équation en α.

Je fais alors une grosse erreur d'étourderie : de $10\alpha = 2\pi R$, je tire $R = \frac{\pi}{5\alpha}$ *au lieu de* $\frac{5\alpha}{\pi}$.

Cela conduit à une équation du 4ème degré en α.

J'abandonne.

J'ai repris ce problème plus tard, à tête reposée et … de tête

Je fais d'abord quelques simplifications (*c'est un des avantages du travail de tête, on est forcé de simplifier):*

- 1 au lieu de 10 (1 dm si l'on veut mais aucune importance)
- α : pourcentage de la circonférence au lieu de la valeur en radians, pour ne plus avoir de π

  on a tout simplement $R = \alpha$ et $R' = 1- \alpha$
- rapport k au lieu de 2 pour pouvoir vérifier des valeurs triviales (0, ∞, 1)
- rapport inverse pour avoir un dénominateur plus sympathique que le dénominateur

soit

$$k^2 = \left(\frac{h'}{h}\right)^2 = \frac{1-R'^2}{1-R^2} = \frac{1-(1-\alpha)^2}{1-\alpha^2}$$

J'arrive à une équation simple du second degré

$$(k^2-1)\,\alpha^2 + 2\,\alpha - k^2 = 0$$

Une vérification donne bien, pour k = 0, $\alpha = 0$

pour k = ∞, $\alpha = 1$

pour k = 1, $\alpha = ½$

On voit l'intérêt des simplifications :
- erreurs évitées
- vérifications faciles

(*Autre avantage du travail de tête, comme on n'est jamais absolument sûr de son calcul, on fait des vérifications.)*

Il y a une racine positive

$$\alpha = \frac{-1+\sqrt{k^4-k^2+1}}{k^2-1}$$ (de nouveau vérification pour k = 0, ∞, 1)

avec k = 2

$$\alpha = \frac{-1+\sqrt{13}}{3}$$

de tête, encadrement de $\sqrt{13}$ entre 3 et 4, puis entre 3,6 et 3,7, d'où entre 0,87 et 0,9, soit un peu plus d'un dixième de tour pour $1- \alpha$

Ayant retrouvé papier-crayon, et avec les données du texte ($\sqrt{13} = 3{,}606$)

je trouve $\alpha = 0{,}8687$ soit en degrés 313°

Non, le α du texte est aigu, j'ai interverti les deux cônes, la réponse est $1- \alpha =$ **47°**

Pour faire plus propre, il faudrait prendre k = ½ et trouver directement α

Je reviens alors sur le brouillon et la découverte de l'erreur a été laborieuse, car je vérifiais tout sauf … le passage de $10\alpha = 2\pi R$ *à* $R = \frac{\pi}{5\alpha}$

Remarque

L'histoire du numérateur plus simple que le dénominateur peut sembler stupide, puisqu'il ne s'agit que de proportions. Ecrire $\frac{1}{k^2} = \frac{1-(1-\alpha)^2}{1-\alpha^2}$ devrait être analogue

Pas tout à fait : il me semble alors que montrer que l'équation a une racine négative, donc une seule racine acceptable, est un peu plus compliqué et je laisse ce fait à vérifier, en me disant que j'ai bien fait de faire le choix inverse …

En fait (quelques mois plus tard) : je suis allée un peu vite, il y a bien une racine à éliminer, mais c'est soit parce qu'elle est négative, soit (cas k<1 dans la version rédigée plus haut) parce qu'elle est supérieure à 1.

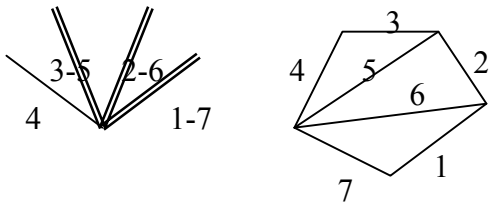
4. Idées fausses et bonnes idées

On verra comment des idées fausses ont pu conduire malgré tout à de bonnes idées et comment, dans la recherche de propriétés pertinentes, des raisonnements incomplets ou inexacts ont été repris jusqu'à la mise au point de la bonne propriété. On verra également l'importance des essais, des cas simples et des dessins.

4.1. Potins et commères (Le Monde 14 mars – Affaire de Logique – Problème n° 163)

Chacune de ces cinq commères connaît un potin croustillant qu'elle voudrait bien faire partager à ses quatre complices.
Mais voilà, les cinq commères sont dispersées ce jour-là aux cinq coins de la ville. Heureusement, elles possèdent des téléphones portables.
Combien d'appels téléphoniques, au minimum, seront nécessaires pour que chacune des cinq commères possède chacun des cinq potins ? Et combien d'appels seront nécessaires avec un nombre quelconque de commères ?
N.B. Evidemment, aucune n'a un abonnement permettant la conversation à trois (ou plus) ni même le signal d'appel.

Premiers essais



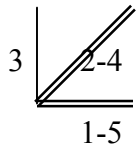
deux méthodes pour trouver une solution en 7 appels

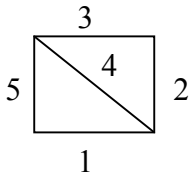
généralisables pour n quelconques (n-1)+(n-2) = 2n-3

Pour les premières valeurs de n cela donne
n=2, 2 appels

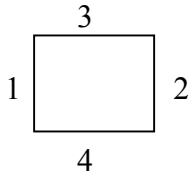
n=3, 3 appels

n=4, 5 appels

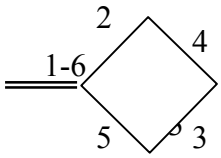




mais on peut faire mieux pour n=4 : 4 appels



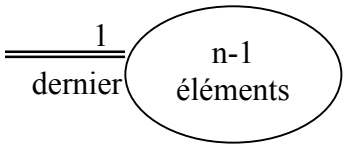
alors pour n=5 :



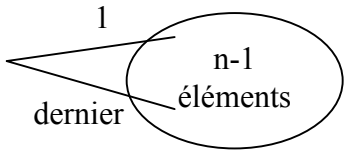
on trouve alors qu'on peut faire 2n-4 pour tout n>=4

1- par récurrence sur n :

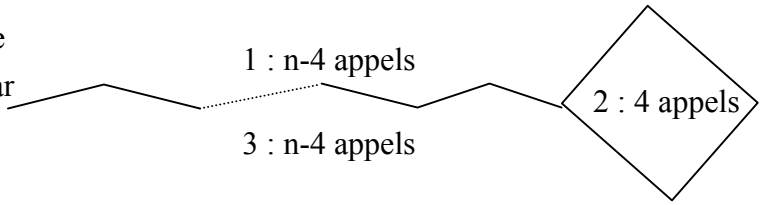
$a_n = a_{n-1} + 2$



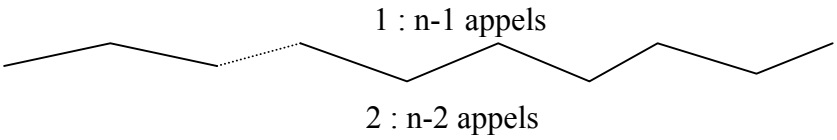
ou



2- en mettant en évidence un groupe de 4 à la fin (au début ça ne suffirait pas car il faudrait le compléter à la fin) :



L'aller-retour simple donnant 2 n –3 appels



Remarque : on passe de n à n+1 en rajoutant 2 appels sauf pour 1 à 2 et 3 à 4 (et 0 à 1)

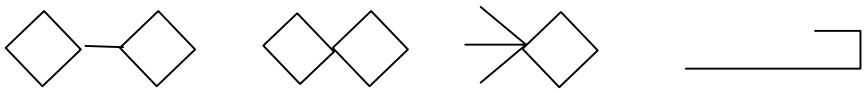
Peut-on faire mieux ?

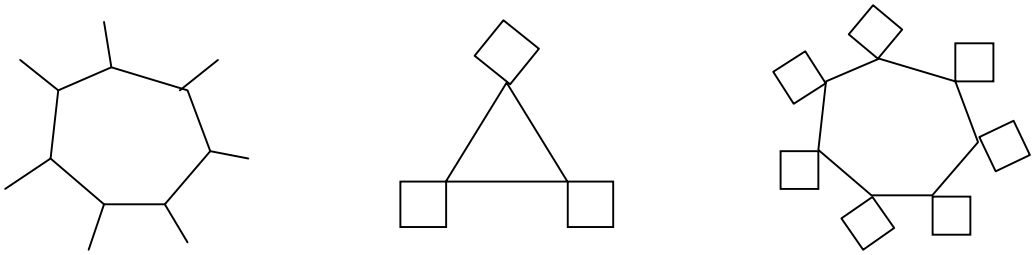
Idée : regrouper 2 par 2 ou par paquets de 4

Travail sur les « aller-retours », sur l'appel qui rend la première commère « savante ».

Importance du sous-ensemble de 2 et importance du dernier morceau de 4

Nombreux essais

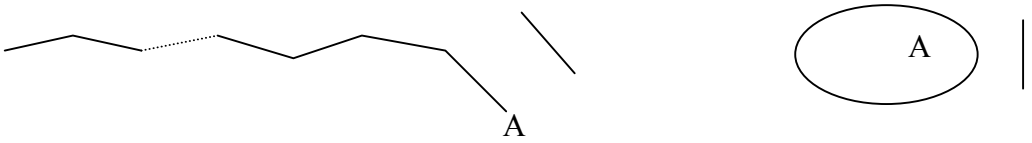




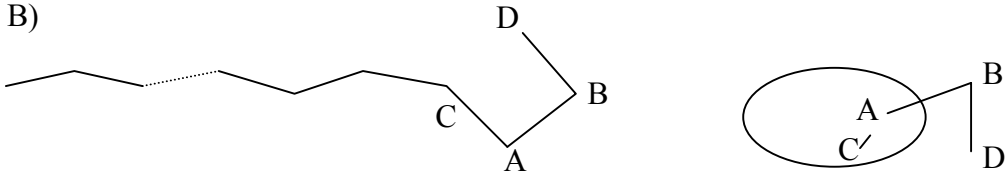
Inutile d'essayer de faire apparaître plusieurs morceaux de 4 complets car il faudra recompléter (sauf le premier). *(C'était une idée fausse, voir plus loin.)*

Importance de 2 et 4, en fait 4 est important car = 2+2

Idée forte le samedi : c'est 2 qui est important, travail avec n-2 d'une part, 2 d'autre part



puis on connecte la commère A des n-2 qui connaît tout des n-2 (au moins n-3 appels) et une des 2 autres, (n-1 appels pour avoir la première commère qui connaît tout, et elles sont deux, A et B)



et on complète le reste, soit n-2 à compléter, soit a priori n-2 mais il y a un couple «complémentaire» qui se complète en un seul appel (C et D), d'où seulement n-3 appels, soit au total (n-3)+1+1+(n-3)= 2n-4

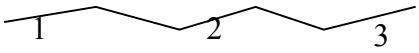
Confusion fréquente dans le raisonnement entre essayer de faire au mieux et montrer qu'on ne peut pas faire mieux.

Pour montrer qu'on ne peut pas faire mieux, il faut montrer qu'on ne peut pas compléter en moins de n-3 appels. J'y arrive si l'appel qui rend la première commère complète est bien le $(n-1)^{ème}$, car il y a alors un seul couple complémentaire.

Et si l'appel qui rend la première commère complète était plus tard, soit n-1+p, il pourrait y avoir plusieurs couples complémentaires.
L'idée de couple complémentaire semble importante.
Il faut aussi envisager que l'appel qui rend la première commère complète n'est pas entre un sous-ensemble de n-2 et un de 2.
Raisonnements un peu confus, certitude que ça doit marcher en alternance avec l'idée que cela repose sur un théorème difficile de théorie des graphes.
Puis impression que mon raisonnement doit marcher, j'attends lundi pour voir si, ou bien la solution est celle que j'entrevois et comment elle est expliquée, ou bien la solution repose sur une idée simple que je n'ai pas vue.

Lundi : solution du Monde : « … on parvient à communiquer en 2 N-4 appels. Il semblerait qu'on ne puisse pas faire mieux (si des lecteurs peuvent le prouver, qu'ils nous écrivent).»

Reprise des raisonnements sur le nombre de couples complémentaires, et sur leur disparition en fonction des appels. Je n'arrive pas à préciser mon raisonnement.
Introduction de la notion de chaîne de couples complémentaires, et de cycle.
Exemples :
5 couples, mais à compter comme 3



3 couples, à compter comme 2

Introduction de la notion de couples isolés, difficiles à compter

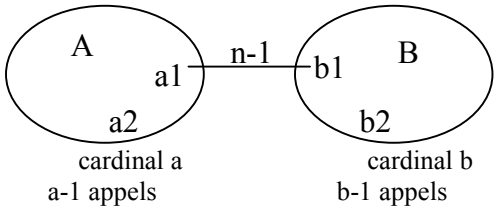
Je m'aperçois que dans la configuration «4» fondamentale,
il n'y a pas 2 couples complémentaires, mais 4 !
Idée d'introduire le nombre de sommets impliqués dans des couples complémentaires et de diviser par 2. On retrouve bien 2 dans le dessin précédent.

C'est toujours aussi confus, abandon proche.

Puis ***idée de la connexité,*** **obtenue également en au moins n-1 appels, qui peut arriver avant la complétude**. J'abandonne alors l'idée de découper avec un morceau de 2 ou de 4. On considère plutôt deux sous-ensembles A et B de cardinaux a et b éléments connexes reliés par un appel qui rend l'ensemble connexe, a et b étant au moins égaux à 2.

Plusieurs cas sont envisagés :

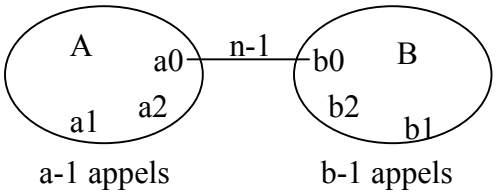
1er cas : la complétude et la connexité sont obtenues au (n-1)ème appel entre a1 et b1 qui connaissent tout. Il y a exactement deux a qui connaissent tout de A. Idem pour B. Le couple a2-b2 est complémentaire. Il faut au moins (a-2) + (b-2) +1 = (a-1)+(a-1) -1 = n-3 pour compléter.



Quelques remarques importantes se précisent (démonstrations par récurrence) :

- pour qu'une commère connaisse tous les potins, il faut au moins n-1 appels. Il y a alors au moins deux commères connaissant tous les potins. Si cette propriété arrive exactement au (n-1)ème appel, il y en a exactement deux. Si cette propriété arrive au (n-1+p)ème appel, il y en a au plus 2p+2.
- si une commère connaît tous les potins, le graphe est connexe. La réciproque n'est pas vraie, la connexité peut arriver avant qu'une seule commère connaisse tous les potins.
- Pour que le graphe soit connexe, il faut au moins n-1 appels.
- (n-1)ème appel peut conduire à la fois à la connexité et à la complétude. Intuitivement, c'est l'optimal. (Mais ce n'est pas nécessaire, voir plus loin.)

2ème cas : le (n-1)ème appel, a0-b0, apporte la connexité mais pas encore la complétude. Il y a au plus deux a (a1 et a2) qui connaissent tous les potins de A. Idem pour B. Il faut au moins (n-1)+(a-2)+(b-2)+2 = 2n-3 pour compléter.

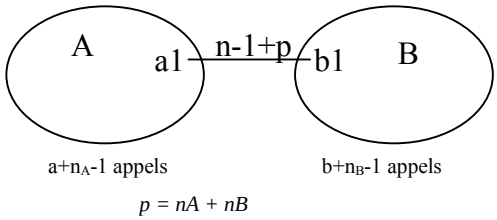


3ème cas : la connexité n'apparaît qu'au (n-1+p)ème appel, après n_A appels chez A et n_B chez B. Il y a au plus $2n_A+2$ commères de A connaissant tous les potins de A et au moins $a-2n_A-2$ ne les connaissant pas. Il y a donc, après le $(n-1+ n_A + n_B)$ème appel entre a1 et b1 :

X_A : au moins $a-2n_A-2$ commères à qui il manque au moins un potin de A et tous ceux de B (sauf éventuellement a_1 qui connaît des potins de B)

Y_A : au plus $2n_A+2$ commères qui connaissent tous les potins de A et aucun potin de B, sauf éventuellement une (a_1), soit $2n_A+1$ connaissant tous les potins de A et aucun de B et une connaissant tout.

Idem pour B.



Il faut donc compléter a-1 (ou a) commères de B et b-1 (ou b) commères de B. Un même

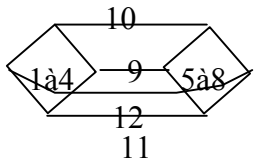
appel peut compléter à la fois deux commères de Y_A et Y_B (couples complémentaires) mais chaque commère de X_A et X_B nécessite un appel. Il faut donc au moins $(a-2n_A-2)+(a-2n_B-2)+\sup(2n_A+1,2n_B+1) = n-3-2\ \inf(n_A,n_B)$ pour compléter. Au total, il faut donc $(n-1+n_A+n_B)+(n-3-2\ \inf(n_A,n_B)) = 2n-4+\sup(n_A,n_B)-\inf(n_A,n_B) \geq 2\ n - 4$

La rédaction a été laborieuse, mais il m'a semblé tenir la bonne idée.

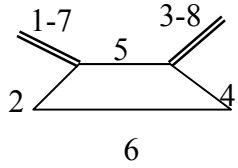
La rédaction définitive, plus concise et plus livresque, ne comporte plus que le 3ème cas, le 1er et le 2ème en étant des cas particuliers.

On voit alors que l'optimal correspond au fait que le $(n-1+ n_A + n_B)^{ème}$ appel conduit à la fois à la complétude et à la connexité, comme attendu (sinon aurait pris $2n_A+2$ au lieu de $2n_A+1$) dans le calcul), mais qu'il n'est pas nécessaire que $n_A=n_B=0$, il peut suffire que $n_A=n_B$.

Exemple avec $n_A=n_B=1$, n=8, a=b=4
(Cet exemple m'avait échappé au début de la recherche, quand je cherchais à recoller des sous-ensembles de 4, je pensais qu'il était en 2n-3)



Autre remarque : le découpage en (n-2)+2 ou (n-4)+4 n'est pas toujours nécessaire pour avoir l'optimal (mais sans doute seulement pour n≤4) . Exemple n=6, a=b=3 .
J'ai souvent oublié, dans les raisonnements, que si une commère connaît tous les potins d'un sous-ensemble, elles sont deux à tout connaître.



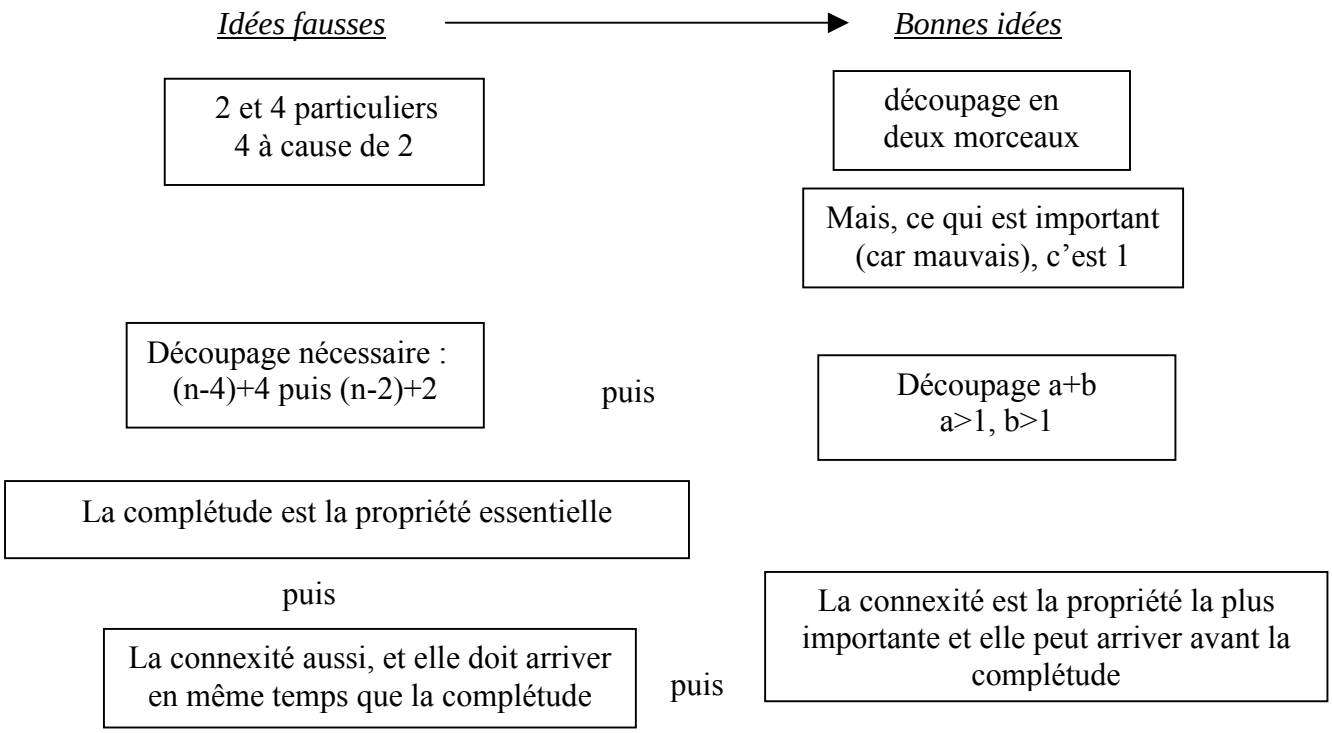
On voit aussi que si a ou b = 1, on n'a pas l'optimal. En effet, si a=1, $n_A=0$, il faut, pour compléter après le $(n-1+ n_B)^{ème}$ appel, au moins $(b - 2\ n_B -2) + (2\ n_B +1) = b-1 = n-2$, soit au total au moins $n - 1 + n_B + n - 2 \geq 2\ n - 3$.
Ceci, par contre, avait été perçu dès le début (la commère isolée – voir récurrence – doit appeler la première et la dernière).

Dernière remarque : si on avait $a-2n_A-2 <0$, on remplacerait $a-2n_A-2$ par 0 et $2n_A+2$ par a et on ne pourrait qu'augmenter la borne.

Finalement, le nombre exceptionnel n'est ni 2 ni 4 mais 1, et n=1 à 3 sont particuliers (l'optimal n'est pas 2n-4) car ils ne se décomposent pas en somme de deux nombres >1, cad l'ensemble des commères ne peut pas être partagés en deux sous-ensembles dans lesquels on

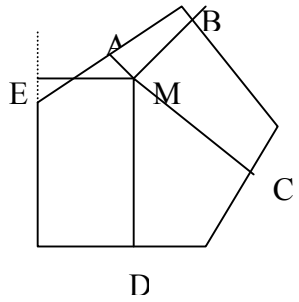
trouvera deux commères connaissant tout de leur sous-ensemble.

Des idées fausses ont ainsi conduit aux bonnes idées, quelquefois simplement en précisant le raisonnement entrevu.

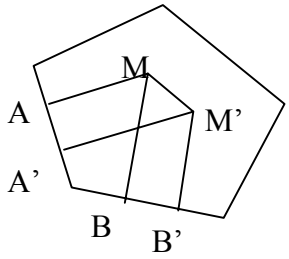


4.2. Invariant pentagonal (Le Monde 15 février - Affaire de Logique- Problème n° 159)

Le pentagone convexe ci-dessous a ses cinq côtés égaux (leur longueur est un même nombre a). A un point M intérieur au pentagone, on associe MA + MB + MC + MD + ME, somme des distances de M aux cinq côtés du pentagone (ou à leurs prolongements).
Sauriez-vous expliquer pourquoi cette valeur ne dépend pas du choix du point M ?
La propriété est-elle conservée si le pentagone a cinq angles égaux sans que les côtés le soient ?

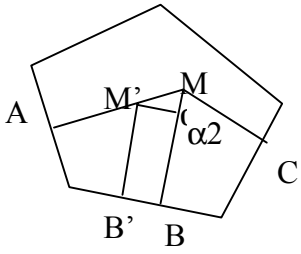


Ce problème a une solution immédiate (voir à la fin) mais je n'en ai pas eu l'idée, ni même aucune autre idée dans un premier temps.



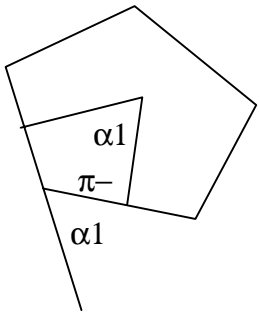
Puis, j'ai l'idée d'observer les conséquences du déplacement du point M.
Pour simplifier, d'abord, le long du segment MA (on pourra ensuite

décomposer un déplacement en deux sous-déplacements le long de MA et MB).
Si MM'=d, MA augmente ou diminue de d, MB augmente ou diminue de d*cos $\alpha 1$, où $\alpha 1$=angle(AMB). MB augmente ou diminue de d*cos(AMC), soit d*cos ($\alpha 1+\alpha 2$), où $\alpha 2$=angle BMC.



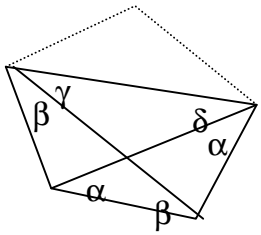
On arrive ainsi à une variation de d multiplié par
$1 + \cos(AMB) + \cos(AMC) + \cos(AMD) + \cos(AME)$ ou
$1 + \cos\alpha 1 + \cos(\alpha 1+\alpha 2) + \cos(\alpha 1+\alpha 2+\alpha 3) + \cos(\alpha 1+\alpha 2+\alpha 3+\alpha 4)$
(Avec petit raisonnement sur les signes selon que les angles sont aigus ou obtus)

Pour un déplacement quelconque de M, on aura
$\cos\alpha 0 + \cos(\alpha 0+\alpha 1) + \cos(\alpha 0+\alpha 1+\alpha 2) + \cos(\alpha 0+\alpha 1+\alpha 2+\alpha 3)+\cos(\alpha 0+\alpha 1+\alpha 2+\alpha 3+\alpha 4)$
avec $\alpha 0$ = angle M'MA



Il reste donc à montrer que cette somme est nulle.
Je remarque que ces angles sont à π près les angles du pentagone.

Evaluer cette somme est immédiat mais je n'ai pas eu tout de suite la bonne idée simple.



Ces angles ne sont pas quelconques puisque les côtés sont égaux. En fait, deux angles sont quelconques, et les autres sont fonctions de ces deux angles. On va donc essayer de les calculer.
Une première relation est simple : $\alpha + \beta = \gamma + \delta$
Il faut une autre équation pour tout connaître en fonction de α et β .

Des calculs compliqués suivent, en utilisant la formule donnant la longueur d'un côté en fonction des angles d'un triangle (retrouvée avec quelque effort). En voici quelques exemples, où les αi désignent maintenant les angles du pentagone

$$A_0A_2 = 2\sin\frac{\alpha 1}{2}$$

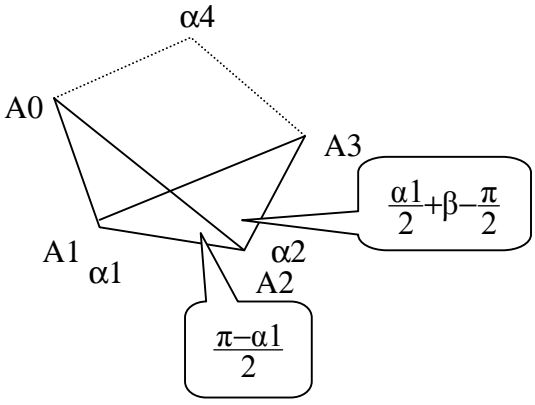
$$(A_0A_2)^2 = 4\sin^2\frac{\alpha 1}{2} = 2 - 2\cos\alpha 1$$

$$(A_0A_3)^2 = 4\sin^2\frac{\alpha 1}{2} + 1 - 4\sin\frac{\alpha 1}{2}\sin(\frac{\alpha 1}{2}+\alpha 2)$$

On a aussi

$$(A_0A_3)^2 = 4\sin^2\frac{\alpha 4}{2} = 2 - 2\cos\alpha 4$$

En égalant les deux expressions de $(A_0A_3)^2$,



on a la relation qui manque …

D'autres calculs, faux (de tête), m'ont conduit à des expresssions de la forme $\frac{\alpha 1}{4}+\frac{\alpha 2}{2}+\alpha 3$, etc.

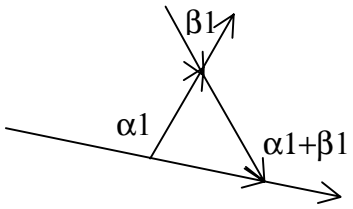
J'en conclus que je suis sur une mauvaise voie, même si, théoriquement, le calcul est possible, je n'aurai pas une expression simple que je pourrai calculer. *Il faut absolument faire apparaître les sommes* $\alpha 1+\alpha 2$, $\alpha 1+\alpha 2+\alpha 3$, *etc.* Il y a confusion avec les αi de tout à l'heure, mais c'est sans importance puisque ce sont les mêmes à π près et leur donner le même nom aide plutôt à y voir clair.

(En fait, contraitrement à ce que je pensais, en reprenant, par curiosité les calculs plus tard, on peut faire apparaître $\alpha 1+\alpha 2$ car $(A_0A_3)^2$ s'écrit aussi $3 - 2\cos\alpha 1 - 2\cos\alpha 2 + 2\cos(\alpha 1+\alpha 2)$, d'où

$\cos\alpha 4 = \cos\alpha 1 + \cos\alpha 2 - \cos(\alpha 1+\alpha 2) - \frac{1}{2}$

Avec la même relation, entre $\alpha 0$, $\alpha 1$ et $\alpha 3$, on a $\cos\alpha 3 - \cos\alpha 1 + \cos(\alpha 1+\alpha 2+\alpha 3+\alpha 4) - \cos(\alpha 2+\alpha 3+\alpha 4) = \frac{1}{2}$

On n'est pas encore au bout de nos peines …)

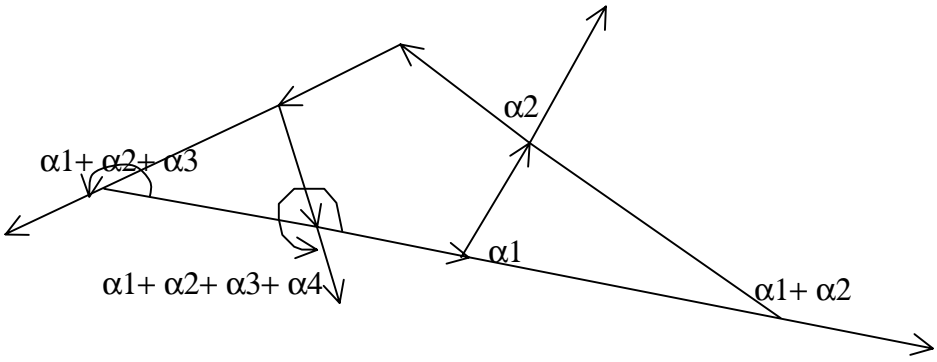


Concentrée sur les sommes des αi en particulier sur $\alpha 1+\alpha 2$ *, j'ai enfin la bonne idée* qui consiste à considérer les angles extérieurs, qui sont égaux aux αi de tout à l'heure. $\alpha 1+\alpha 2$ est égal à l'angle, orienté, du 1er côté avec le 3ème.

Les sommes des angles considérées sont les angles d'un des côtés avec successivement tous les autres. Les cosinus, multipliés par la longueur commune a, sont les projections des vecteurs-côtés sur l'un d'eux, celui-ci contribuant pour a et la somme

$$1 + \cos\alpha 1 + \cos(\alpha 1+ \alpha 2) + \cos(\alpha 1+ \alpha 2+ \alpha 3) + \cos(\alpha 1+ \alpha 2+ \alpha 3+ \alpha 4)$$

est bien nulle puisque la somme des 5 vecteurs-côtés est nulle !



On peut aussi faire directement le cas du déplacement MM' quelconque en projetant sur une droite perpendiculaire à MM'.

En ce qui concerne la deuxième question, avec des angles égaux, je pensais d'abord que la réponse était non. Mais de ce qui précède, je déduis immédiatement que la réponse est oui, en considérant un pentagone régulier inscrit dans le cercle trigonométrique et en projetant la somme des vecteurs-côtés sur l'axe des x.

La réponse du Monde est d'une simplicité telle que je regrette de ne pas l'avoir trouvée : « L'aire du pentagone est la somme des aires des cinq triangles de sommet M. » La suite est alors triviale. Par contre, la réponse à la deuxième question est moins simple que la mienne : on part du pentagone régulier et on le déforme.

Conclusion : Il est difficile de savoir pourquoi l'idée très simple des aires n'est pas apparue.
En ce qui concerne la solution trouvée, on remarque que c'est l'*idée fausse* « je ne peux pas avoir $\alpha 1 + \alpha 2$ » qui a conduit à conduit à la *bonne idée* : il faut se concentrer sur cette somme $\alpha 1 + \alpha 2$, la faire apparaître directement.

5. Chemins détournés et bonnes idées

On verra, dans le premier problème un raisonnement géométrique absolument pas nécessaire, mais qui a conduit aux bonnes relations algébriques. Dans le deuxième problème, la bonne idée de la parité n'est apparue qu'à l'occasion d'un sous-problème simple, mais cette idée est bonne pour le problème initial et la décomposition en sous-problèmes est inutile. Enfin dans le troisième, on voit apparaître des configurations plus complexes que nécessaire.

5.1. La grande famille (Le Monde 11 avril – Affaire de Logique – Problème n° 167)

> Le patriarche de cette famille a eu des enfants, des petits-enfants, des arrière-petits-enfants et même des arrière-arrière-petits-enfants. Tous sont vivants, en bonne santé, et sont venus souhaiter (sans leurs conjoints) à l'ancêtre son anniversaire. Des chaises ont été installées en carré (il y a autant de chaises dans chaque rangée que de rangées de chaises), et elles sont toutes occupées lorsque l'ancêtre se rassoit à la fin de son discours.
> Chose extraordinaire, toutes les personnes réunies (sauf bien sûr les arrière-arrière-petits-enfants, encore en bas âge) ont eu le même nombre d'enfants.
> Combien ?

Soit b le nombre d'enfants et c la longueur du carré, on doit avoir

$$1 + b + b^2 + b^3 + b^4 = c^2$$

1. Je trouve *rapidement une solution à la main* : b=3 et c=11

Je *vérifie à la calculette, qu'il n'y a pas d'autre solution* jusqu'à b=10, ce qui est raisonnable comme maximum pour b.

2. J'essaie des calculs très compliqués avec des *factorisations, des parités, des « descentes ».* Sans succès. Je trouve juste que b doit être impair.

3. Avec des *caculs modulo* 2, 4, 3, 10, je trouve juste que b doit être impair ou multiple de 4 et que c doit être congru à 1 ou 5 modulo 10.

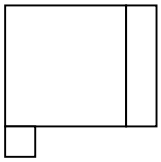
4. *J'essaie des encadrements.*
Ayant en tête que $(b+\frac{1}{2})^2 = b^2 + b+\frac{1}{4}$, c'est-à-dire est compris entre $b^2 + b$ et $b^2 + b+1$ utilisé dans un problème résolu peu de temps auparavant (problème n° 10), et en partant de $b^4 < c^2 < (b+1)^4$ ou mieux de $(b+\frac{1}{4})^4 < c^2 < (b+\frac{1}{2})^4$, et en majorant ou minorant par les entiers les plus proches, on trouve

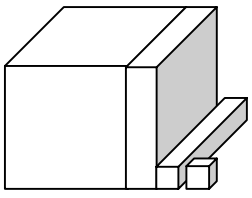
$$b^2 + \frac{b}{2} + \frac{1}{16} = (b+\frac{1}{4})^2 < c < (b+\frac{1}{2})^2 = b^2 + b+\frac{1}{4}$$

$$b^2 + \frac{b}{2} + \frac{1}{2} \le c \le b^2 + b$$

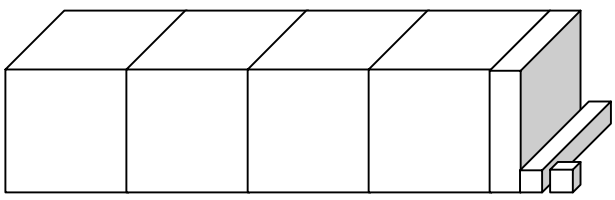
Je cherche à mieux encadrer, avec quelque chose de la forme $(b+\frac{1}{4}+ \ldots)^2$, par exemple $(b+\frac{3}{8})^2$. Ne réussissant pas, j'abandonne cette voie.

5. *J'ai alors l'idée d'un raisonnement géométrique* (en pensant à la démonstration géométrique de $\sum_{i=1}^{n}(2i+1) = \sum_{i=1}^{n} (2i+1) = (n+1)^2$.

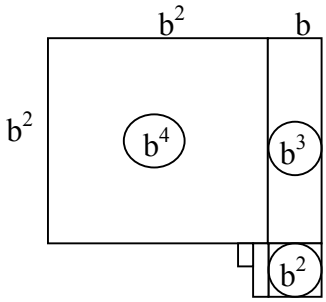
$1 + b + b^2 =$ 

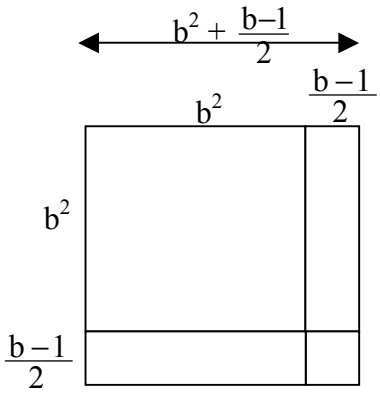
$1 + b + b^2 + b^3 =$ 

mais pour $1 + b + b^2 + b^3 + b^4$ il faudrait une $4^{ème}$ dimension, ou alors dupliquer :



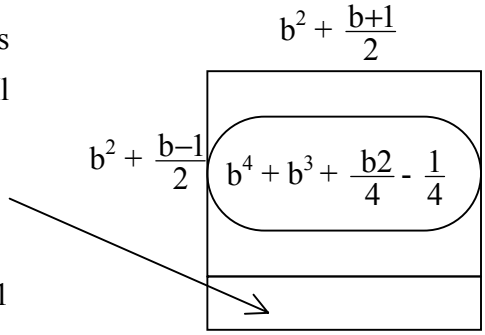
Ou alors considérer un carré de côté b^2 et le compléter.
Le rectangle $b \times b^2$ est trop grand, je le coupe en deux.





Je considère le cas ou b est impair (ce que je sais), et je prends un rectangle de côté $\frac{b-1}{2}$. Je pourrai ainsi, en complétant le carré de $b^2 + \frac{b-1}{2}$ obtenir $b^4 + b^3 - \frac{3b^2}{4} - \frac{b}{2} + \frac{1}{4}$ qui est trop petit.

Je m'intéresse au rectangle juste un peu plus grand de côtés $b^2 + \frac{b}{2} + \frac{1}{2}$ et $b^2 + \frac{b}{2} - \frac{1}{2}$. Sa surface est $b^4 + b^3 + \frac{b2}{4} - \frac{1}{4}$. Il reste $\frac{3b^2}{4} + b + \frac{5}{4}$ à placer pour occuper au moins $b^2 + \frac{b}{2} + \frac{1}{2}$.



On en déduit que $\frac{3b^2}{4} + b + \frac{5}{4} \geq b^2 + \frac{b}{2} + \frac{1}{2}$

soit $b^2 - 2b - 3 \leq 0$. On trouve alors que b ne peut être que 1 ou 3. Seul 3 convient.

Si on se savait pas que b est impair, on peut faire un raisonnement analogue pour b pair avec $\frac{b}{2}$ au lieu de $\frac{b-1}{2}$ et on trouve que ne peut être que 0.

6. *Raisonnement formel*

Finalement, ce que j'ai fait, c'est simplement comparer $(b^2 + \frac{b}{2} + \frac{1}{2})^2$ et $c^2 = b^4 + b^3 + b^2 + b + 1$

On savait dèjà que $b^2 + \frac{b}{2} + \frac{1}{2} \leq c$. En élevant au carré, on obtient $\frac{b2}{4} - \frac{b}{2} - \frac{3}{4} \leq 0$ qui est la même inéquation que précédemment !!!

On peut même faire encore plus court :

On a $(b^2+\frac{b}{2}+\frac{1}{2})^2 \le c^2 = b^4 + b^3 + b^2 + b + 1 \le (b^2+\frac{b}{2}+1)^2$

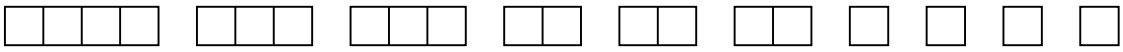
la dernière inégalité est stricte sauf si b=0

On en déduit que, si b≠0, c ne peut être égal qu'à $b^2+\frac{b}{2}+\frac{1}{2}$, ce qui donne l'équation $b^2 - 2b - 3 = 0$ c'est-à-dire b=3. Sinon, b=0 et c=1, solution trivale non retenue.

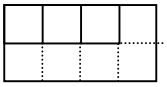
5.2. Touché-coulé (Le Monde 3 juin – Affaire de Logique – Problème n° 175)

Le jeu de bataille navale se joue sur un damier. Une flotte comporte un porte-avions (4 cases), deux cuirassés (3 cases, trois croiseurs (2 cases) et quatre sous-marins (1 case) . deux navires distincts ne peuvent se toucher, même par un coin.

Pouvez-vous, en respectant ces règles, placer deux flottes complètes dans un quadrillage de 10 cases sur 10 ?



Des *premiers essais* montrent qu'un flotte occupe la moitié du quadrillage, mais qu'on ne peut pas mettre la deuxième flotte sans toucher la première.



Rapidement, je remplace les navires de n cases par des blocs de 2×(n+1) cases :
La contrainte de ne pas se toucher disparaît et il faut considérer un quadrillage de 11x11.

Un rapide calcul montre que chaque flotte occupe 10+8×2+6×3+4×4 = 60 cases et qu'il faut donc placer un total de 120 cases dans un quadrillage de121 cases. Il n'y a donc qu'un case en trop !

Première idée : regarder les différentes façons d'obtenir 11 avec les longueurs et les largeurs des blocs :

11 = 5+4+2 = 5+3+3 = 5+2+2+2 = 4+3+2+2 = 3+3+3+2

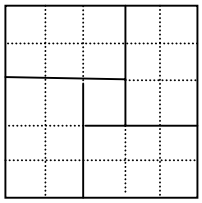
et aussi 10, qui concernera les deux lignes (horizontale et verticale) où se trouve la case vide :

10 = 5+3+2 = 4+4+2 = 3+3+2+2

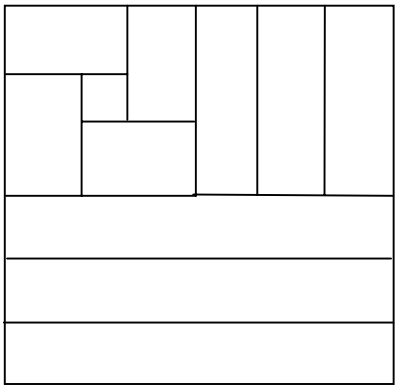
Un décompte des différentes possibilités n'est pas jugé assez contraint.

Je m'intéresse alors à la case vide (qui peut être considéré comme un bloc supplémentaire de une case à placer, il s'agit alors de recouvrir entièrement le quadrillage).

On s'aperçoit facilement qu'elle ne peut se situer ni dans un coin ni sur un bord car on ne pourrait pas l'entourer avec les autres blocs. On la met donc à distance des bords et on regarde comment on peut l'entourer. Il semble nécessaire de procéder de la façon suivante :



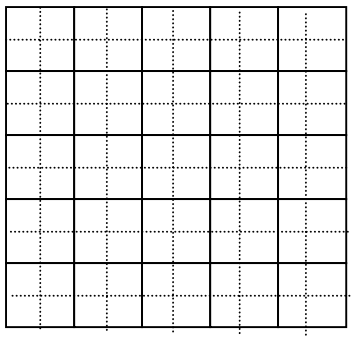
J'essaie alors de compléter ce sous-ensemble 5×5 par un quadrillage 6×6 et par un 6×11, et je m'intéresse aux blocs de 6, c'est-à-dire 4+2 ou 3+3 ou 2+2+2. J'en mets 3 dans le 6×6 et essaie de remplir des 11×2 dans le 11×6.
Je n'y arrive pas et m'aperçois que c'est impossible car je n'ai plus assez de 3. Il manque toujours 1 pour obtenir 11. En effet, il faut un impair, il y a un seul 5, et je n'ai plus de 3.
Ceci est vrai quel que soit l'endroit où l'on place le 5x5, étant donné le nombre de lignes horizontales et verticales à compléter.



J'exploite alors l'idée de la parité *pour résoudre le problème de départ de la façon suivante :*
En considérant la case vide comme un bloc 1x1 à placer, il faut obtenir 22 fois la somme 11. Les seuls impairs sont un 5 (largeur 2, 2 flottes), trois 3 (largeur 2, 2 flottes), un 1 (largeur 1, 2 sens). On ne pourra alors obtenir que 1×2×2 + 3×2×2 + 1×2 = 18 sommes impaires.
Le problème est donc impossible.

L'idée, simple et puissante, de la parité n'est pas venue tout de suite, mais en travaillant sur un sous-ensemble de blocs à placer, celui-ci étant un sous-problème du problème initial, issu d'un premier raisonnement finalement inutile.

La solution du Monde est beaucoup plus élégante :
Elle est également plus simple en un certain sens puisqu'on reste sur une grille 10×10 et qu'on raisonne sur les bateaux de n cases et non sur des blocs de 2×(n+1) cases.



« Si on divise la grille en 25 carrés de deux cases sur deux, on remarque qu'aucun de ces carrés ne peut abriter d'éléments provenant de deux navires différents. Les porte-avions et les cuirassés des deux flottes occupent chacun, dans le meilleur des cas, deux de ces carrés, ce qui en mobilise au moins 12. Les 14

autres bateaux en mobilisent au moins un chacun. D'où la nécessité de 26 carrés pour 25 disponibles. »

5.3. Zigzag numérique (Le Monde 4 avril – Affaire de Logique – Problème n° 166)

Avec les chiffres de 1 à 6, on peut fabriquer des nombres « zigzag ». Ces nombres, que nous appellerons des « 6-zigzags », ont 6 chiffres qui vont d'abord croissant, puis décroissant, puis croissant …
Par exemple, 231546 est un « 6-zigzag ».
Combien existe-t-il de « zigzags » ?
Pour les spécialistes. On peut étendre le problème aux suites « zigzag » des n premiers nombres entiers : les nombres vont d'abord croissant, puis décroissant, puis croissant …
Sauriez-vous construire un algorithme permettant de les dénombrer ?

Enumération des 6-zigzags.
Je remarque que l'on retrouve des sous-ensembles de 4-zigzags (à un renommage près).
Je recommence en utilisant explicitement les 4-zigzags et je corrige une erreur.
Je cherche les 5-zigzags, je les utilise pour les 6 zigzags, et je corrige une autre erreur.

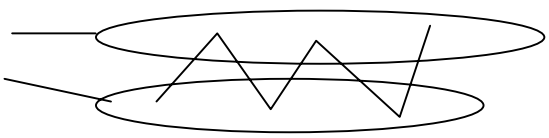
Soit a_n le nombre de n-zigzags, on a
$a_2 = 1$, $a_3 = 2$, $a_3 = 5$, $a_4 = 16$, $a_6 = 61$

Cas général, que signifie un algorithme ? Une relation de récurrence ? Je ne vois rien d'autre.

En *regardant la construction* pour n=5 et 6, je trouve, en regardant les zigzags commençant par 1, 2, …, n-1
$a_5 = a_4 + a_4 + (a_3 + a_3) + a_3 = a_4 + a_4 + (a_4 - a_2) + a_3 = 16$
$a_6 = a_5 + a_5 + (a_4 + a_4 + a_3 + a_3) + 2\,a_4 + a_4 = a_5 + a_5 + (a_5 - a_3) + 2\,a_4 + a_4 = 61$
$a_7 = a_6 + a_6 + (a_6 - a_4) +$?????? $+ 2\,a_5 + a_5$
Je n'arrive pas à généraliser.

Nouvelle idée : je m'intéresse aux couples mal placés. Non généralisable.

Nouvelle idée : regarder ce qui est « en haut » et « en bas »



après un raisonnement facile pour n= 4 et 5,
et avec les remarques suivantes pour n=6 :

- 123 en bas et 456 en haut : $(3\ !)^2 = 36$ possibilités
- on ne peut pas avoir deux parmi 123 en haut, ni deux parmi 456 en bas
- si un parmi 123 en haut alors un parmi 456 en bas
- 6 ne peut pas être en bas
- si 5 en bas, il ne peut être qu'à gauche, car seul 6 est plus grand que lui, il reste 1234 à placer, soit $a_4 = 5$ possibilités
- si 4 en bas ou bien à gauche et le suivant est 5 ou 6, et dans chaque cas, $a_4 = 5$ possibilités
  ou bien au milieu, entouré de 5 et 6 (2 possibilités) et il reste 123 à placer
    soit 3 à gauche, $a_3 = 2$ possibilités
    soit 1 à gauche et 2 à droite, 3*1 = 3 possibilités

Au total 36 + 5 + 2*5 + 2*(2+3) = 61
Difficilement généralisable, pour n=7, ça devrait encore aller, mais pour n=8, par exemple, si 5 est en bas, il faut envisager deux parmi 678 à côté. Pour n plus grand, ce sera encore pire.

Enfin une bonne idée : plutôt que de s'intéresser à ce qui pourrait être en haut ou en bas, regarder ce qui est nécessairement en haut, soit 6, et remplir à gauche et à droite par 12345 :
- si 6 est le $2^{ème}$, un quelconque parmi 12345 à gauche (5 possibilités) et une 4-zigzag à droite avec ce qui reste (a_4=5 possibilités), soit 5*5 = 25 possibilités
- si 6 est le $4^{ème}$, deux parmi 12345 à droite ($C_5^2 = 10$ possibilités) et une 3-zigzag à droite avec ce qui reste (a_3=2 possibilités), soit 10*2=20 possibilités
- si 6 est le $6^{ème}$, a_5=16 possibilités pour le reste

.1 *Soit au total 25 + 20 + 16 = 61 possibilités,* $a_6 = C_5^1 a_1 a_4 + C_5^3 a_3 a_2 + C_5^5 a_5 a_0$

Et c'est facilement généralisable :

$$a_n = \sum_{i=1 \text{ par pas de } 2} C_{n-i}^{i} a_i a_{n-i-1}$$

soit, pour n=7, $a_7 = C_6^1 a_1 a_5 + C_6^3 a_3 a_3 + C_6^5 a_5 a_1 = 6*1*16+20*2*2+6*16*1 = 272$

L'algorithme du Monde (c'est bien un algorithme, et pas seulement une formule, même de récurrence) :
« On construit le triangle suivant : sur la ligne n, on inscrit successivement le nombre de « n-zigzag » se terminant par 1, par 2, par 3, … par n.

Pour n=1				1				
Pour n=2			0		1			
Pour n=3		1		1		0		
Pour n=4	0		1		2		2	

On obtient chaque nombre de la ligne 5 en ajoutant tous les nombres de la ligne 4 qui se trouvent à sa droite :

Pour n=5	5	5	4	2	0

En effet, il y a autant de « 5-zigzag » se terminant par 1 que de « 4-zigzag » (en décalant les chiffres d'une unité). Il y a autant de « 5-zigzag » se terminant par 2 que de « 4-zigzag » se terminant pat 2, 3 ou 4 (en décalant les chiffres à partir de 2 d'une unité), etc.
De même, on obtient chaque nombre de la ligne 6 en ajoutant tous les nombres de la ligne 5 qui se trouvent cette fois à sa gauche :
Pour n=6 0 5 10 14 16 16
Le total de la sixième ligne vaut bien 61.
Plus généralement, si n est impair, la ligne se termine par 0 ; et, si n est pair, elle commence par 0.
On obtient les nombres de la ligne n+1 en ajoutant successivement et alternativement, de droite à gauche, puis de gauche à droite, les nombres de l ligne n. »

Algorithmique astucieux ! Comment en avoir eu l'idée ?

6. Peut-on compter sur la mémoire ?

On verra deux exemples où la mémoire a immédiatement permis de résoudre le problème, soit par réflexe, soit parce que la solution repose sur une idée se trouvant dans un travail particulièrement intéressant dont je me souvenais très bien. Dans un autre exemple, la mémoire n'a servi à rien parce que je n'avais pas moi-même trouvé la bonne idée dans le passé et que les points clés n'avaient pas été dégagés. Maintenant que j'ai résolu moi-même, avec difficulté, ce problème, je pense que je me souviendrai longtemps des points clés.

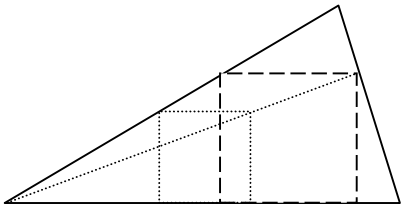
6.1. Le carré inscrit (Le Monde 28 mars – Affaire de Logique – Problème n° 165)

Il n'y a qu'une façon d'inscrire un carré dans un triangle ABC de telle sorte que deux des sommets soient sur AB, un troisième sur AC et le quatrième sur BC.
Sauriez-vous tracer un tel carré à l'aide d'une construction géométrique simple ?

Je me souviens immédiatement que c'est un problème de construction résolu par le programme de la thèse de [Buthion, 1975][19] au moyen d'une méthode très intéressante utilisant les transformations géométriques et en particulier l'homothétie.

[19] 3000 instructions Fortran, 68 règles, plusieurs modules spécialisés dont un module « homothétie », 141 problèmes proposés, 119 succès

On dessine un carré respectant les contraintes sauf une : un des sommets du carré n'est pas sur le triangle. Par une homothétie, dont il faut trouver le centre et le rapport, on transforme ce carré dans le carré cherché.

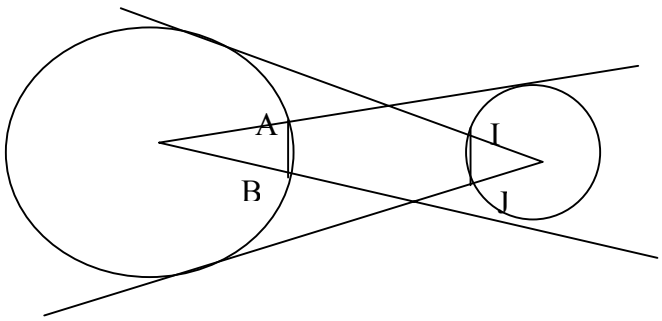


La raison pour laquelle je m'en souviens si bien tient sans doute au fait que j'ai été impressionnée par le travail de Michel Buthion et en particulier par ses idées sur le transformations géométriques et l'implémentation de cette méthode.

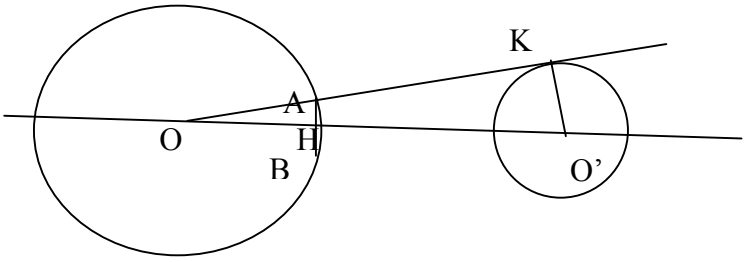
La solution du Monde est la même, mais exprimée en termes beaucoup plus élémentaires, et certainement moins facile à trouver directement, et moins facile à mémoriser.

6.2. Des sous ! des sous ! (Le Monde 28 mars – Affaire de Logique – Problème n° 164)

Placez sur un papier une pièce de 5 francs et, un peu plus loin, une pièce de 10 centimes.
Menez du centre de la grande pièce deux tangentes à la petite, et du centre de la petite deux tangentes à la grande, comme sur le dessin.
Il semblerait que les longueurs des segments verticaux AB et IJ soient égales, est-ce vrai ?



Je *vois* immédiatement les deux triangles semblables : OHA et OKO', d'où l'on déduit facilement que $AB = \frac{R \times R'}{OO'}$. Idem pour IJ.



Ici, il s'est agi d'un ***réflexe****.*

6.3. La fausse pièce (Le Monde 25 janvier – Affaire de Logique – Problème n° 156)

Vous disposez d'une balance à deux plateaux et de douze pièces d'aspect identique. Onze sont d'authentiques pièces de collection, la dernière n'est qu'une imitation, si bien faite qu'elle a le même aspect que les autres. Heureusement, sa masse diffère légèrement de celle des vraies pièces, mais vous ignorez si elle est plus lourde ou plus légère. Comment, en un minimum de pesées, démasquer la fausse pièce et dire si elle est plus lourde ou plus légère ?

6.4. et **http://www710.univ-lyon1.fr/~fouet/Dir_lic_mai/TP/puzzles.html**

On a douze pièces, onze bonnes et une fausse, et une balance (une balance qui indique équilibre ou déséquilibre, pas qui donne une mesure). La fausse pièce est plus lourde ou plus légère que les vraies. En trois pesées, déterminer la fausse pièce et dire si elle est plus lourde ou plus légère.

L'énoncé de Jean-Marc Fouet comporte de plus en note de bas de page : « Ce problème est très dur, il n'y a, à ma connaissance, pas d'astuce pour le résoudre. Ma solution prend deux pages, c'est pourquoi je ne vous propose pas de zone pour donner la vôtre. Mais, même si vous ne trouvez pas, je pense que le fait d'y réfléchir est bon pour les neurones »

J'avais cherché à résoudre le problème du Monde quelques mois avant de trouver le texte de Jean-Marc sur sa page Web. Je n'avais pas trouvé mieux que 4 pesées pour démasquer la fausse pièce, sans pouvoir dire, dans tous les cas, si la pièce est plus lourde ou plus légère (une pesée supplémentaire serait alors parfois nécessaire).
Mon raisonnement procédait simplement par dichotomie :
1- on laisse la moitié (6) et on met sur chaque plateau de la balance la moitié (3) de l'autre moitié, soit notation 3↑3 + 6. Selon qu'il y a équilibre ou non, la fausse pièce se trouve dans une des moitiés (6 billes), l'autre moitié est bonne.
2- on recommence avec 2↑2 +2 ou 1↑1 + 4
3- puis avec 1↑1 + 2
4- enfin, on pèse une des billes du paquet de 2 dans lequel on sait que se trouve la fausse pièce avec une bonne pièce. Si c'est elle, on sait de plus si elle est plus lourde ou plus légère. Si non, on sait que c'est l'autre sans savoir si elle est plus lourde ou plus légère, il faut une $5^{ème}$ pesée pour conclure.

Ce raisonnement marcherait aussi pour16 billes. On a l'impression qu'avec seulement 12, on pourrait faire mieux, et surtout, on n'exploite pas l'information recueillie en cas de déséquilibre : d'un côté, il ne peut y avoir qu'une plus lourde, de l'autre qu'une plus légère. D'où une idée qui semble meilleure : un tiers sur chaque plateau et un tiers restant : 4↑4 +4.

S'il y a équilibre, on ramené au problème pour 4 et on démasque la fausse pièce en 2 pesées, sinon on doit considérer le problème pour 8 pièces, mais en sachant qu'on a une lourde parmi 4 ou 1 légère parmi 4 autres.
Néanmoins je n'avais pas pu faire mieux.

Le Monde donnait une solution en 3 pesées et je me souvenais que je l'avais trouvée compliquée. Je ne me souvenais pas si je l'avais analysée (ou si j'avais remis l'analyse à plus tard et oubliée). Je ne me souvenais pas si l'on pouvait conclure à coup sûr si la fausse pièce était lourde ou légère. Et je ne me souvenais absolument pas de la dite solution.

J'ai donc recommencé une recherche, avec cependant les informations suivantes :
- il y a une solution en 3 pesées
- on peut conclure sur le sens de l'erreur
- la solution de JMF comporte certainement une part de combinatoire.

Je démarre avec 4↑4 +4 et j'utilise la notation suivante :
+ : bille qui ne peut être que bonne ou trop lourde,
- : bille qui ne peut être que bonne ou trop légère,
? : bille dont on ne sait rien,
b : bonne bille.
Cette première pesée conduit donc à bbbbbbbb ???? ou bien à ++++----bbbb. Pensant que le premier cas est résolu, je m'intéresse au deuxième et j'essaie de mélanger, par exemple, peser ++-- ↑ ++--.
Divers essais infructueux.

Combinatoire me conduit à réfléchir à un algorithme que l'on pourrait, avec quelques heuristiques, développer à la main.
Qu'est-ce que je ferais si je devais écrire un programme ? Je l'écrirais récursif avec comme paramètres le nombre de +, de -, de ? et de b. Il faudrait alors écrire
- une descente,
- un arrêt.

La descente ne me semble pas simple, du moins si on imagine qu'on l'exécutera à la main.
Je cherche plutôt l'arrêt (on a une seule bille + ou une seule -), ou juste avant l'arrêt. Je cherche les situations où l'on peut conclure en une seule pesée ? Je trouve ++, +-, --, ?-, à condition qu'il y ait au moins un b. Je trouve aussi ++- et +--, à condition qu'il y ait au moins 2b (on pèse +- ↑ bb ; je trouverai mieux plus tard, sans utiliser de b : +↑+).
J'ai alors la solution pour le cas ++++---- (1ère pesée en déséquilibre) :
en effet, on pèse alors (2ème pesée) ++- ↑ ++-
- si équilibre, il reste +- qui est résolu en une 3ème pesée

- sinon, on a ++- ou qui est aussi résolu en une 3ème pesée

Je crois avoir terminé, mais je m'aperçois que l'autre cas après la 1ère pesée (4 ?), j'avais la mauvaise pièce, mais pas toujours le sens de l'erreur, en faisant

2ème pesée : ? ↑ ?

- si déséquilibre, on a +- qui est résolu en une 3ème pesée
- mais sinon, on a ?? qui, en une pesée ne donne que la pièce (pas toujours le sens de l'erreur).

Je cherche les situations où l'on peut en 2 pesées ? J'essaie +++-- en pesant ++- ↑ bbb. Ca marche.

J'essaie ++++--- en pesant +++-- ↑ bbbbb, mais de toute façon, on n'a pas 5b après la première pesée ci-dessus.

Je remarque qu'en cas d'équilibre aux deux premières pesées, il faut qu'il ne reste qu'un ?, sinon, on ne peut pas conclure.

Je fais alors des essais infructueux en commençant par la première pesée 5 ↑ 5 + 2, car alors c'est l'autre cas de la 1ère pesée qui ne marche plus.

Je décide alors de nouveau de remonter depuis l'arrêt en cherchant systématiquement le nombre de pesées nécessaires pour de petits ensembles de billes. En faisant cela, l'idée suivante apparaît : il faut absolument que de ????, on obtienne, au pire ++-.

?? ↑ bb ne marche pas car il reste ??

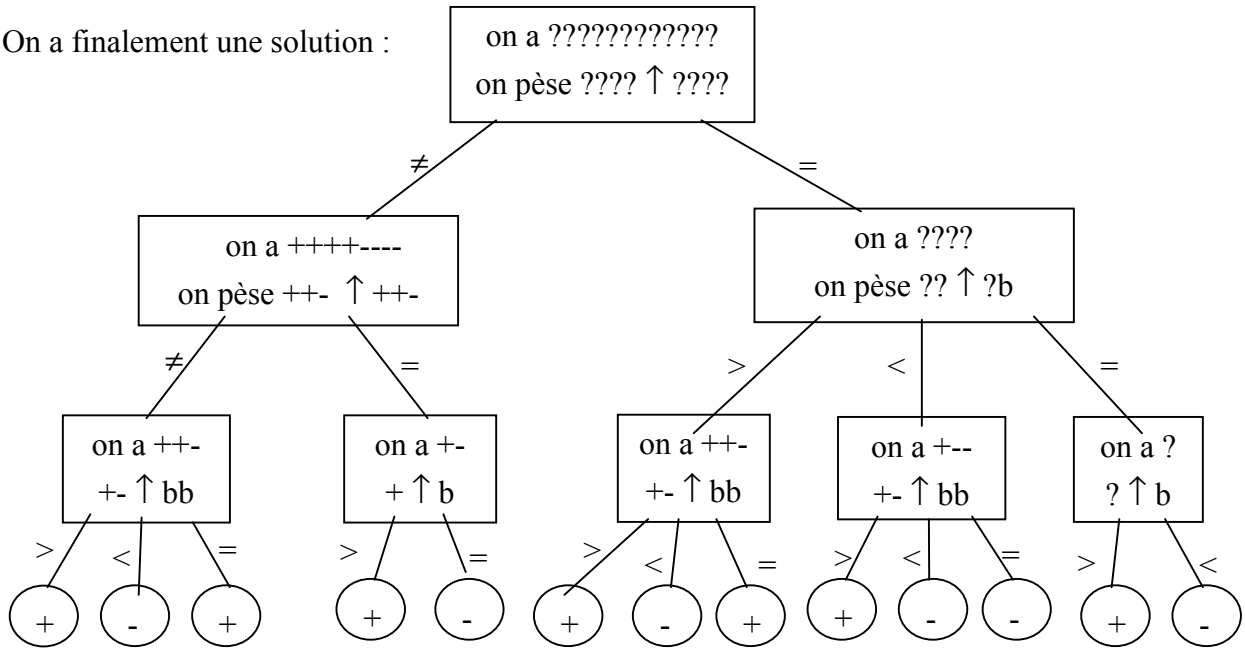
??? ↑ bbb ne marche pas, car on ne peut finir avec +++ (*Cette idée est fausse, +++ marche aussi bien que ++- en une pesée, mais peu importe pour la suite*)

Idée : prendre une sorte d'intermédiaire.

On en prend alors 3, complétés par un b, soit ?? ↑ ?b

- si équilibre, on pèse ? restant en une 3ème pesée
- sinon, on a ++- ou +-- qui est résolu en une 3ème pesée

On a finalement une solution :



Une petite précision semble nécessaire : quand on pèse ++- ↑ ++- , en cas de déséquilibre, la pièce fausse se trouve soit parmi les deux + du plateau le plus lourd, soit le – du plateau le plus léger, les trois autres pièces étant bonnes. On se retrouve donc avec ++- mais ce ne sont pas ceux d'aucun des plateaux précédents.

Les idées fortes ont été les deux suivantes :

++- se résout en une pesée

++- peut être obtenu en pesant ?? ↑ ?b

Ce sont certainement des idées dont je me souviendrai et je pourrai maintenant retrouver cette solution, même dans longtemps, contrairement à la solution du Monde. *Il est en effet plus utile et plus facile de mémoriser les points clefs d'une solution plutôt que la totalité des étapes.*

De nombreuses variantes existent, avec les mêmes idées.

Je signale également une idée qui m'a été donnée par Jean-Yves Lucas : après la première pesée concluant sur ++++---- que l'on numérote $1^+2^+3^+4^+5^-6^-7^-8^-$ on pèse bbb5^-↑ $4^+6^-7^-8^-$. On obtient ainsi des informations à la fois sur les pièces 4 et 5 qui ont été échangées et sur les pièces 123 qui ont été remplacées par bbb. Si la balance penche à gauche, on a $6^-7^-8^-$, si elle penche à droite 4^+5^{--}, s'il y a équilibre $1^+2^+3^+$.

On m'a aussi indiqué que ce problème était analysé dans [Farreny 87] qui est un livre que je pensais pourtant bien connaître, mais j'avais oublié que ce problème y figurait. Il illustre en effet le choix heuristique d'un sous-problème dans un arbre ET/OU. La théorie de l'information conduit à choisir le sous-problème qui apporte le plus d'information. Appliqué à ce problème, pour 8 pièces, les auteurs montrent que le meilleur choix au départ est de mettre

3 pièces sur chaque plateau. Cette méthode permet d'écrire un programme qui résoudra le problème pour un nombre quelconque de pièces. Et c'est certainement cette méthode qu'a appliquée Jean-Marc Fouet et qui l'a conduit à une solution longue (deux pages) mais dont on est sûr qu'elle aboutira.

7. **Déformation professionnelle** ou **Métarésolution de problème mal posé**

Dans l'exemple suivant, étonnée de trouver deux solutions, j'ai tout simplement abandonné un peu de rigueur dans l'interprétation de l'énoncé …

7.1. Enquête (Le Monde 30 mai – Affaire de Logique – Problème n° 173)

Trois malfaiteurs sont soupçonnés de meurtre. Un – et un seul – des trois est coupable. Les enquêteurs ont recueilli trois déclarations de chacun d'eux :

André : (A1) – Je suis innocent.
(A2) – D'ailleurs, à l'heure du crime, j'étais à 10 kilomètres de là avec Béatrice.
(A3) – Claude est coupable.

Béatrice : (B1) – Je suis innocente.
(B2) – André aussi.
(B3) – Mais il n'était pas avec moi à l'heure du crime.

Claude : (C1) – Je suis innocent.
(C2) – Béatrice aussi.
(C3) – André a menti trois fois.

Sachant que chacun des suspects a menti au moins une fois, qui est coupable ?

J'ai d'abord considéré que « j'étais à 10 kilomètres de là avec Béatrice » n'était pas exactement la négation de « il n'était pas avec moi à l'heure du crime » car André aurait pu être avec Béatrice à l'heure du crime, mais pas à 10 km de là.
Trouvant alors deux solutions (Béatrice ou Claude coupable), j'en ai conclu que mon interprétation n'était pas la bonne. non(A2) entraîne (B3) et Béatrice est coupable.
C'est la solution du Monde et cela était également clair pour d'autres lecteurs de mon entourage …

8. Autres problèmes

Le premier de ces problèmes montre que, quand on est très savant, on résout facilement les problèmes théoriquement, mais qu'avec un peu de ténacité on le résout concrètement. Les deux suivants sont des problèmes combinatoires pour lesquels des techniques type ALICE donnent de bons résultats. Néanmoins dans le problème de crypt-arithmétique, il est utile de ne pas se contenter d'écrire les équations entre les chiffres mais de considérer aussi les entiers. Enfin, le dernier problème conduit à une réflexion sur la répartition des nombres qui m'a étonnée.

8.1. Préfixes pour puissances (Le Monde 22 février – Affaire de Logique – Problème n° 160)

En multipliant le chiffre 2 par lui-même un certain nombre de fois, on obtient une puissance de 2, qui s'écrit avec un « 2 », suivi, en exposant, de ce nombre de fois.
Ainsi, les premières puissances de 2 sont : 2^1=2, 2^2=4, 2^3=8, 2^4=16, 2^5=32, 2^6=64, 2^7=128, …
Une puissance de 2 peut-elle commencer par le chiffre 7 ?
Et par les quatre chiffres 2 0 0 0 ?

Je pense d'abord que non …

Je *remplace 7 par* $2^3 - 1$ et j'écris $(2^3 - 1) \times 10^n + a = 2^p$

De *nombreuses manipulations* (regroupement des puissances de 2, factorisations, log) ne donnent rien.

Je regarde les premiers : 1248 / 136 / 125 / 1248 / 136 / 125 / … . Ca semble recommencer toujours.

Avec une calculette, je trouve $\mathbf{2^{46} = 7.0368 \times 10^{13}}$ (C'est faisable à la main.)

Pour 2000, je divise successivement par 2 :

```
2000[1-9]…
1000[0-4]…
 500[0-2]…
 250[0-1]…
 1250[0-9]…
  625[0-4]…
  312[5-7]…
```

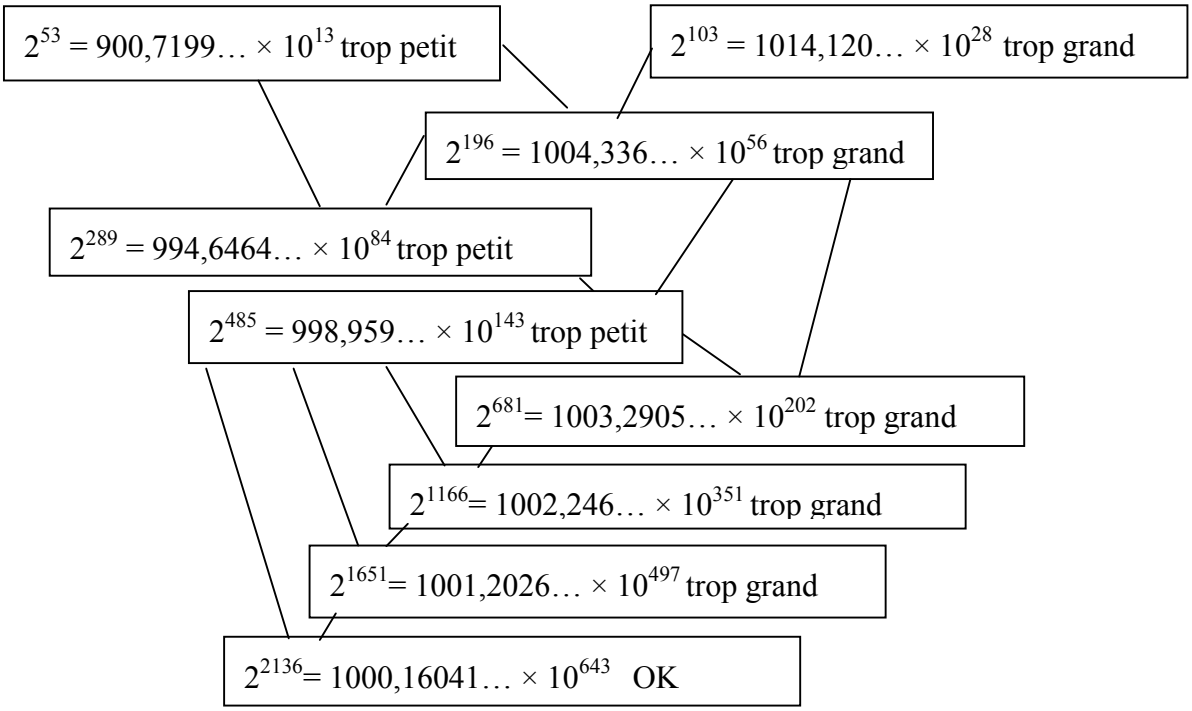
156[2-3]…

781…

D'où, s'il y a une solution, elle sera obtenue en multipliant une solution au problème précédent par 2^8. Je vais plutôt m'intéresser à 1000[0-4]… . Je multiplie 2^{46} par 2^7.

$2^{53} = 900…$ est trop « petit » . Je vais multiplier les nombres obtenus par des nombres un peu « grands » ou un peu plus « petits » que 1000…. Pour commencer, on connaît $2^{10}=1024$ et $2^{53} = 900…$

On obtient ainsi



D'où $2^{2136} = 2000{,}32…. \times 10^{643}$

Remarques :

- les calculs ont été faits avec une calculette (sans les puissances de 10, bien sûr), mais ils sont faisables à la main
- je n'étais pas assurée d'arriver à une solution en un temps (manuel) raisonnable
- le problème pour 7 aurait pu être résolu par la même méthode en partant de 8 et en multipliant par 1024, etc.

Solution du Monde :

Pour 7, les auteurs partent de 64 et multiplie 4 fois par 1024. On a la solution.

Pour 2000, ils se contentent de dire que la réponse est "oui, mais la justification n'est pas simple, elle fait appel à un "passage" aux logarithmes décimaux : on montre qu'il existe deux entiers K et n tels que $K \leq n \log 2 < K + \log(1{,}005)$. C'est parce que log 2 n'est pas un nombre

fractionnaire que c'est toujours possible". Alors "$2 \times 10^K \leq 2^{n+1} < 2001 \times 10^{K-3}$ c'est-à-dire 2n+1 commence par 2000.

Le théorème, plus général, auquel il est fait allusion est le suivant :
Pour tout réel r, pour tout ε il existe des entiers m et n tels que $n \leq m\ r < n + \varepsilon$, c'est-à-dire $\frac{n}{m} \leq r < \frac{n}{m} + \frac{\varepsilon}{n}$
Je connais et je sais démontrer que $\frac{n}{m} \leq r < \frac{n}{m} + \varepsilon$ mais ici, le ε dépend du dénominateur (il diminue au fur et à mesure que le dénominateur augmente !) et je n'arrive pas à démontrer ce résultat.
J'interroge des mathématiciens de mon UFR. Plusieurs "… ne voient pas comme ça, sans réfléchir …". Enfin deux mathématiciens entament un dialogue que l'on peut résumer ainsi :
- oui, et ça marche même avec $\frac{1}{n^2}$
- oui, mais pas avec ε quelconque
- ça se fait avec des fractions continues
- c'est le théorème de Hurvitz, on le trouve dans "Solutions to the theory of numbers" par Hardy Right
- pour $\frac{1}{n^2}$, c'est $\frac{\sqrt{5}}{n^2}$ (ça vient des fractions continues)

L'un d'eux se décide à s'intéresser à *mon* problème, plus simple.
On considère l'ensemble $\{E(n\ r) | \ n \in N\}$ qui est inclus dans [0,1[où E(x) est la partie entière de x. Cet ensemble est discret ou infini. On divise [0,1[en intervalles de longueur < ε. Alors il existe deux entiers différents m_1 et m_2 tels que $E(m_1\ r)$ et $E(m_2\ r)$ appartiennent au même intervalle, soit $0 \leq E(m_1\ r) - E(m2\ r) < \varepsilon$
Il existe alors n_1 et n_2 tels que $0 \leq (m_1\ r - n_1) - (m_2\ r - n_2) < \varepsilon$
Et on a le résultat en prenant $m=m_1-m_2$ et $n=n_1-n_2$
Remarque : j'avais eu l'idée de prendre les parties fractionnaires de n r, mais je voulais à tout prix les faire se rapprocher directement de 0. Ici on prend la différence de deux proches.

8.2. Une multiplication (Jeux mathématiques – ½ finales 2000)

Dans cette multiplication, le chiffre 7 apparaît une fois et une seule. Ainsi, chaque étoile (*) représente un chiffre de 0 à 9 différent de 7 . De plus l'écriture d'aucun nombre ne commence par 0.
Quel en est le résultat ?

					*	*	*	*
			×		*	*	7	*
				*	*	*	*	*
				*	*	*	*	
		*	*	*	*			
	=*	*	*	*	*	*	*	*

Des techniques, type ALICE, conduisent rapidement à la solution.
Mais … il y a beaucoup de chiffres inutiles, je fais plutôt intervenir, dans les inégalités, le premier nombre (entier) et les résultats intermédiaires (entiers) des multiplications, en utilisant leurs nombres de chiffres

8.3. Le carnet de timbres (Jeux mathématiques – ½ finales 2000)

Il est possible d'obtenir toutes les sommes entières de 1 à 36 en découpant un ou plusieurs timbres dans un carnet de timbres rectangulaire de deux timbres sur trois portant les valeurs 1, 2, 3, 5, 8 et 17, les timbres restant formant toujours un ensemble d'un seul tenant.
Pouvez-vous reconstituer ce carnet en plaçant la plus petite des quatre valeurs de coin en haut et à gauche ?
L'exemple de la figure ci-dessous ne convient pas car il est impossible d'obtenir les sommes 7, 10, 12, 15, 24, 27 et 32.

1	2	3
5	8	17

Résolu de manière combinatoire et après pas mal de cafouillages … . (J'ai même à, plusieurs reprises, cru que le problème était impossible).

On énumère les nombres, autres que 1, 2, 3, 5, 8, 17
- qui ne peuvent s'obtenir que d'une seule façon, 4=1+3, 7=2+5, 12=1+3+8, 15=2+5+8 et les « complémentaires » : 32, 29, 24
- qui peuvent s'obtenir de deux façons : 6=1+2+3=1+5, 9=1+3+5=1+8, 10=2+6=2+3+5, 13=2+3+8=5+8, 14, 16, 17, 18, et les « complémentaires »

- qui peuvent s'obtenir de trois façons : 11

Remarques qui n'ont été faites qu'après un long moment de recherche :
- pour les ensembles de décomposition unique, il faut que l'ensemble *et* son complémentaire soient connexes
- pour les autres, il suffit qu'un des ensembles soit connexe et que le complémentaire d'un ensemble (le même ou un autre) le soit aussi, sauf pour 18 (égal à son complémentaire), pour lequel il suffit qu'un des ensembles ou son complémentaire soit connexe.

8.4. Les sauts (A partir d'un sujet d'examen)

Les étudiants de DEUG de Paris 5 ont eu à répondre à cette question à leur examen d' « Initiation à la programmation » :
« On considère la fonction entière f(n) = $[n+\sqrt{n}+½]$ où [ ] désigne la partie entière.
Pour tout entier a, si l'entier n parcourt l'intervalle [1, a^2], f(n) atteint a^2+a en sautant exactement a entiers.
Ecrire un programme Pascal qui demande à l'utilisateur de fournir une valeur de a et affiche les valeurs entières que f(n) ne prend pas. »

Pourriez-vous déterminer les valeurs « sautées » par la fonction f(n) directement par un raisonnement mathématique ?

On trouve facilement que les valeurs sautées sont tous les carrés.
On trouve, moins facilement, que le saut a lieu pour le passage de p^2+p à p^2+p+1.
Je n'ai pas gardé de protocole de la recherche, mais un idée fausse sur la répartition des racines carrées des nombres compris en p^2 et $(p+1)^2$ m'a beaucoup gênée. Je pensais naïvement, à cause de l'image mentale de la courbe $\sqrt{x}$ qu'il y en avait beaucoup plus dont la partie décimale était supérieure à ½ que inférieure.
On a en effet les variations suivantes, en considérant les nombres de la forme n= p^2 + a compris entre p^2 et $(p+1)^2$ et en remarquant que $[\sqrt{n}+½]$ est l'arrondi à l'entier le plus proche de $\sqrt{n}+½$:

a	0	p	p+1/4	p+1	2 p +1
n	p^2	p^2+p	$(p+1/2)^2$	p^2+p+1	$(p+1)^2$
$\sqrt{n}$	p		p+1/2		p+1
$[\sqrt{n}+½]$	p	p	p+1	p+1	p+1

On constate qu'il y a p nombres entiers dont la partie décimale est inférieure à ½ et autant

dont la partie décimale est supérieure à ½. Si on considère les nombres réels, il y en a plus dont la partie décimale est supérieure à ½, mais par beaucoup, le rapport est en effet égal à $\frac{p+3/4}{p-1/4} = 1 + \frac{2}{4p+1}$. S'il y en a 3 fois plus entre 0 et 1 et 40% de plus entre 1 et 4, il y en a déjà moins de 10% en plus entre 25 et 36. Cela veut dire que la courbe ressemble très vite à une droite.

Ces remarques conduisent alors rapidement à une solution rapide du problème (mais elle n'a été obtenue qu'après plusieurs autres plus laborieuses).

a	0	↗	p	↗	p+1 ↗	2 p +1
n	p^2	↗	p^2+p	↗	p^2+p+1 ↗	$(p+1)^2$
$\sqrt{n}$	p	→			p+1	
$[\sqrt{n} + ½]$	p	→	p	↗	p+1 →	p+1
$[n+\sqrt{n} + ½]$	p^2+p	↗	$p^2+ 2\ p$	↗	p^2+2p+2	$(p+1)^2+p+1$

On constate que l'on saute $p^2+ 2\ p + 1 = (p+1)^2$

9. Conclusion

Ces analyses mettent en évidence que les problèmes ont rarement été résolus directement mais la plupart du temps après de nombreux essais, tâtonnements, dessins, cas particuliers. La rigueur dans le détail n'est pas indispensable dans un premier temps. Ce qui est important par contre, c'est de manipuler les objets mathématiques, et une idée, même inexacte, permet de manipuler, donc de progresser.

On voit aussi qu'entre le plus court chemin vers la solution et des impasses sans espoir, il y a des raisonnements inutilement compliqués qui permettent aussi d'aboutir. Une fois la solution trouvée, il serait alors aisé de trouver la justification la plus simple et la plus élégante, telle que celle que l'on trouve dans les livres. Actuellement, les systèmes d'intelligence artificielle cherchent à aller droit au but, ou au moins le plus directement possible. Peut-être serait-il nécessaire, pour les faire progresser, de leur apprendre à naviguer parmi les idées fausses et le chemins détournés.

L'étourderie est le propre de l'homme. C'est un défaut que les machines n'ont pas. Nous avons quelquefois des difficultés parce que nous sommes trop savants et nous ne voyons pas les choses simples. Pour le moment, les machines peuvent nous surpasser dans ces situations.

Mais, le jour où elles seront aussi savantes que nous, elles auront sans doute les mêmes difficultés.

Enfin, en dehors des domaines pour lesquels nous sommes devenus experts, nous ne pouvons nous souvenir longtemps que de ce qui nous a marqué, en particulier de ce qui nous a semblé très important ou particulièrement intéressant. Le souvenir de quelques points clés est plus utile que la mémorisation d'un grand nombre d'étapes. Même si les capacités mémoire d'une machine sont sans commune mesure avec les nôtres, il faudrait apprendre aux systèmes d'intelligence artificielle à dégager les points importants des raisonnements.

10. Références

[Farreny 87] Farreny H., Ghallab M., *Eléments d'intelligence artificielle*, Hermès 1987

[Pastre 78] Pastre D., *Observation du mathématicien: aide à l'enseignement et à la démonstration automatique de théorèmes*, Educational Studies in Mathematics 9 (1978), 461-502

[Chi 81] Chi M., Feltovitch P., Glaser R., *caracterization and representation of physics problems by experts and novices*, Cognitive Science 5 (1981), 121-152

[Pitrat 99] Pitrat J., *Monitorer la recherche d'une solution*, Colloque Intelligence Artificielle Berder 1999 et rapport de recherche LIP6 2000/002, 3-15

[Schoenfeld 85] Schoenfeld A., *Mathematical Problem Solving*, Academic Press, 1985

La mise en place du monitoring dans MALICE

Jacques Pitrat
CNRS-LIP6
Université Pierre et Marie Curie

Résumé : Pour résoudre un problème, on dispose d'un ensemble de règles qui transforment son énoncé jusqu'à ce que l'on trouve directement la solution ou qu'une approche combinatoire soit envisageable. Un expert dispose de métaconnaissances compilées pour utiliser judicieusement de telles règles ; un des rôles du monitoring est justement de construire ces ensembles de métaconnaissances pour chaque famille de problèmes en examinant comment le système résout les problèmes. Nous décrirons le langage procédural mis en œuvre dans MALICE pour donner ces métaconnaissances et nous en montrerons des exemples d'application à la définition de l'utilisation de règles. Maintenant que le système fonctionne avec ces séquences d'ordres, il faut lui donner la capacité de les engendrer automatiquement. Il faudra aussi être capable de faire du métamonitoring de façon à ne pas perdre trop de temps à faire du monitoring.

1. Principes du système

Un système intelligent doit avoir des métaconnaissances compilées pour des raisons d'efficacité et des métaconnaissances déclaratives pour des raisons de commodité. Il suffit donc qu'il ait la possibilité de compiler les métaconnaissances déclaratives qui lui sont données ou qu'il a trouvées lui-même. Les métaconnaissances compilées correspondent au comportement d'expert et les métaconnaissances déclaratives sont utiles dans le cas où l'on n'est pas encore expert. Notons que d'avoir des métaconnaissances compilées ne signifie pas que l'on résout combinatoirement le problème, car une métaconnaissance indique seulement quelles connaissances il est bon d'utiliser, faire de la combinatoire n'étant qu'une connaissance particulière. D'autres connaissances peuvent être des moyens plus directs d'arriver à la solution. Les métaconnaissances compilées correspondent plutôt à une résolution métacombinatoire du problème où l'on a prévu la succession des essais et où l'on fait tous ceux qui sont autorisés. Mais cela n'entraîne pas l'examen systématique de ce qui se passe pour toutes les combinaisons de valeurs possibles des variables.

Comme pour les êtres humains [Schoenfeld 85], [Pitrat 99], le monitoring est essentiel pour diriger de façon souple le comportement d'un système d'Intelligence Artificielle et pour éviter de faire systématiquement une résolution métacombinatoire. Il doit tout superviser en

examinant régulièrement le déroulement de ce qui se passe, même quand on utilise des connaissances compilées. Grâce au monitoring :

1. Le système ne tourne pas en rond. Il se rend compte quand il fait toujours la même chose ; dans ce cas il change une des actions pour sortir de la boucle.
2. Il remarque si un essai est ou non proche d'un résultat intéressant. Si l'on sait que X est inférieur à 10, trouver que X est inférieur à 10000 montre que l'on était loin du but. Par contre, trouver que X est inférieur à 11 n'apporte pas davantage, mais cela montre que l'essai valait la peine d'être tenté.
3. Il voit que l'on redécouvre des résultats déjà obtenus, pas exemple on recrée une contrainte que l'on avait déjà trouvée.
4. Il se rend compte d'une mauvaise surprise. Il s'attendait à un succès alors que l'on n'aboutit à rien. Cela se produit plus souvent en utilisant des métaconnaissances compilées où l'on croit que la méthode suivie doit mener à la solution, mais il lui arrive parfois d'échouer.
5. Il se rend compte d'une bonne surprise. D'après ses prévisions, on avait peu de chances de réussir et pourtant cet essai a conduit à la solution. Ou bien il devait enlever au plus une valeur possible pour une variable et, en réalité, il en a supprimé une dizaine. Cette fois, cela intervient plus souvent avec des métaconnaissances interprétées, où l'on est beaucoup moins sûr de ce que va donner un essai.
6. Il constate que tout se déroule normalement. Cela permet de conforter l'intérêt d'une méthode.
7. Il décide de fonctionner en utilisant les métaconnaissances compilées ou au contraire d'interpréter les métaconnaissances déclaratives.
8. Il se rend compte qu'il ne sert à rien d'être intelligent pour résoudre un certain sous-problème : il engendre un programme combinatoire qui examine toutes les combinaisons de valeurs possibles pour les variables jusqu'à ce que ce sous-problème soit résolu. Naturellement, il doit créer intelligemment ce programme combinatoire [Laurière 96].
9. Il essaye de modifier l'énoncé du problème de façon à le mettre sous une forme telle que ses heuristiques soient plus efficaces ou bien que la combinatoire soit bien plus faible.
10. Il décide de faire une expérience pour vérifier une hypothèse, en particulier l'efficacité des métaconnaissances compilées qu'il a engendrées.
11. Il sauvegarde des informations sur ce qui s'est passé dans une certaine situation afin d'avoir des éléments pour apprendre ultérieurement à partir de ce qui s'est produit. Il ne faut pas tout retenir, savoir sélectionner ce que l'on réexaminera par la suite est une tâche importante.

L'apprentissage n'est qu'une des phases de la résolution de problèmes dont il est indissociable. C'est au cours de la résolution que l'on trouve la plupart des éléments qui permettront ensuite d'améliorer les performances ; il est même possible d'apprendre pendant la résolution d'un problème de façon à pouvoir en utiliser les résultats immédiatement.

La réflexivité doit intervenir au cours du monitoring. En effet, il ne faut pas que le système passe son temps à examiner ce qu'il faudrait mieux faire, sans rien faire effectivement. Le monitoring doit lui-même être monitoré afin de ne pas passer un temps excessif dans les activités de monitoring. Le même mécanisme de monitoring doit être appliqué dans tous les cas, y compris aux activités d'apprentissage et de monitoring. Le système doit donc faire du méta-monitoring, ou monitoring du monitoring. Il doit en particulier :

Ne pas perdre du temps à faire trop de monitoring dans une situation où cela ne sert à rien de vouloir mieux choisir les essais à faire, par exemple quand il vaut mieux tout examiner le plus vite possible.

Evaluer le coût d'un essai peut être très coûteux. La connaissance de la valeur de ce méta-coût peut aider à limiter le temps passé à évaluer les caractéristiques d'un essai. Il ne faut jamais que l'évaluation de l'intérêt de faire un essai prenne plus de temps que de faire cet essai lui-même !

De façon générale, il ne faut pas faire de monitoring dans des situations où les essais devront de toutes les façons être faits ; le temps passé au monitoring doit être bien plus faible que le temps passé à faire les essais qu'il permettrait d'économiser.

Nous allons maintenant montrer comment ces idées ont commencé d'être implémentées dans la nouvelle version de MALICE, le module du système MACISTE [Pitrat 96] qui est un résolveur général de problèmes inspiré du système ALICE [Laurière 76, 78].

2. Formulation d'un problème dans MALICE

Le formalisme dans lequel les problèmes sont énoncés est basé sur celui d'ALICE. Quatre classes principales d'informations définissent une famille de problèmes, auxquelles peuvent s'ajouter les données particulières à chaque problème de la famille :

SOIT définit des constantes, des vecteurs, des matrices, des ensembles. Il commence par le type (CONSTANTE, VECTEUR, MATRICE, ENSEMBLE) suivi du nom de la variable qu'il définit puis éventuellement d'un moyen de calculer cette valeur. Pour cela, on peut utiliser les expressions MACISTE définissant un nombre ou un ensemble. S'il n'y a pas de définition, le système demandera la valeur pour chaque problème posé.

TROUVER définit une correspondance entre un ensemble de départ E1 et un ensemble d'arrivée E2. Nous pouvons définir le type de la correspondance : fonction,

injection, bijection…Nous pouvons aussi indiquer que la valeur d'un degré sur l'ensemble de départ ou d'arrivée est connu ; DMI et DMA sont les degrés minimaux et maximaux pour l'ensemble de départ et AMI et AMA sont ceux pour l'ensemble d'arrivée. La plupart du temps ces degrés découlent du type de la correspondance, par exemple pour une fonction, DMI=DMA=1 alors que AMI et AMA ne sont pas définis. Les types de correspondances ne sont qu'une façon commode de définir plusieurs de ces degrés en une fois ; aussi ces types disparaissent-ils une fois que l'on a indiqué au système la valeur des degrés qui les caractérisent.

CONTRAINTE définit une contrainte du problème. Nous pouvons avoir des contraintes qui sont des expressions mathématiques, mais aussi des générateurs de contraintes qui engendrent un ensemble de contraintes qui changera selon les autres données du problème.

SACHANT est lié à une des correspondances définies précédemment. Cet ordre peut définir un graphe ou des cliques de disjonction.

Nous pouvons définir un graphe en indiquant explicitement les liens possibles entre les éléments de l'ensemble de départ et ceux de l'ensemble d'arrivée. Par exemple, pour le Cavalier d'Euler, on peut choisir d'avoir une bijection de l'ensemble des cases (départ) sur l'ensemble des cases (arrivée) ; le graphe indique alors pour chaque case les cases accessibles par un déplacement de Cavalier. Si le graphe n'est pas défini explicitement, tous les liens entre les éléments des deux ensembles sont possibles. Un des buts de la résolution est de trouver des liens certains parmi les liens possibles entre les éléments des ensembles de départ et d'arrivée. Cela se fait souvent en montrant que des liens possibles doivent être éliminés parce qu'ils ne peuvent certainement pas être certains.

Nous pouvons aussi indiquer l'existence de cliques de disjonction entre les éléments de l'ensemble de départ. Cela signifie que si un élément d'une clique a un lien certain avec l'élément A de l'ensemble d'arrivée, les autres éléments de cette clique ne pourront être associés à cet élément A. Bien entendu, il est inutile d'utiliser une clique quand AMA vaut 1, car tous les éléments de l'ensemble de départ sont automatiquement dans la même clique.

Ce langage est très proche de celui de ALICE, dont certaines possibilités, comme la recherche d'une solution optimale, n'ont toutefois pas encore été implémentées.

A titre d'exemple, donnons l'énoncé qui décrit une formulation générale possible des problèmes de cryptaddition. Cette forme utilise des retenues, mais il est parfaitement possible de définir dans le même formalisme bien d'autres formulations du même problème.

```
SOIT  ENSEMBLE LIGNES = [1 : NL]
SOIT  ENSEMBLE COLONNES = [1 : NC]
SOIT  ENSEMBLE RETENUES = [0 : NC]
SOIT  ENSEMBLE CHIFFRES = [0 : 9]
```

```
SOIT  ENSEMBLE VALRET = [0 : SUB(NL,2)]
SOIT  ENSEMBLE ARGUMENTS = [1 : PRED(NL)]
SOIT  ENSEMBLE LETTRES = SET[A{I,J}; I∈ LIGNES, J∈ COLONNES, A{I,J}≠' ']
SOIT  CONSTANTE  NL
SOIT  CONSTANTE  NC
SOIT  MATRICE  A  LIGNES X COLONNES→LETTRES
TROUVER  FONCTION  R  RETENUES→VALRET
TROUVER  INJECTION  F  LETTRES→CHIFFRES
AVEC  R{0} = 0
AVEC  R{NC} = 0
AVEC  SOMME[R{J}, SIGMA[F{A{I, J}} ; I∈ ARGUMENTS]] =
SOMME[F{A{NL,J}}, PRD[10, R{PRED{J}}]]  POUR J∈ COLONNES
AVEC  F{A{I,J}}≠0  POUR I∈ LIGNES, J∈ COLONNES, A{I,J}≠ ' ',
NONEX[K∈ COLONNES, K<J, A{I,K}≠' ']
```

L'ordre de définition des éléments n'a aucune importance. On définit les ensembles de lignes et de colonnes, dont les tailles sont NL et NC qui seront lus pour chaque nouveau problème. Il y a une retenue de plus que de colonne pour que la formule définissant la contrainte d'addition sur une colonne soit toujours applicable. Les valeurs des retenues sont définies comme étant au plus le nombre de lignes moins 2 (ceci n'est pas indispensable, si on lui donnait une limite moins stricte comme NL, MALICE éliminerait sans aucune difficulté les valeurs supérieurs à la limite précédente). Les arguments représentent les éléments que l'on additionne. La matrice rectangulaire A devra être lue pour chaque nouveau problème. L'ensemble des lettres est formé des éléments de cette matrice qui ne sont pas des blancs.

On doit trouver deux correspondances, d'une part une fonction R qui définit la valeur des retenues et d'autre part une injection F (donc DMI=DMA=AMA=1) entre les lettres de la cryptaddition et les nombres compris entre 0 et 9. Notons que si la taille de l'ensemble des lettres est égal à 10, on aura une bijection, mais c'est à MALICE de découvrir et d'utiliser cette contrainte supplémentaire qui indique que le AMI de la correspondance F vaut aussi 1.

Nous avons enfin les contraintes. D'abord deux valeurs initiales des retenues qui valent 0 pour les valeurs extrêmes. Nous avons ensuite un premier générateur de contraintes qui crée une contrainte par colonne. Chacune traduit la règle de l'addition avec des retenues : la somme de la retenue de la colonne précédente et des valeurs des lettres des opérandes pour cette colonne est égale à la lettre du résultat plus dix fois la valeur de la retenue de la colonne. Un deuxième générateur de contraintes traduit le fait que la lettre située au début de chaque ligne ne doit pas avoir la valeur 0 ; il crée donc une contrainte par ligne.

Des exemples classiques de cryptadditions sont :

SEND+MORE=MONEY

DONALD+GERALD=ROBERT

AIN+AISNE+DROME+MARNE=SOMME

La deuxième de ces cryptadditions conduira à une bijection, car elle a dix lettres différentes ; MALICE trouve la solution et en montre l'unicité en faisant seulement deux choix binaires. Pour la troisième, NC=NL=5 ; elle a la propriété d'être aussi vraie si on met les numéros de départements (1+2+26+51=80), mais elle a malheureusement trois solutions.

Au départ, on connaît un certain nombre de liens possibles entre chaque élément de l'ensemble de départ d'une correspondance et certains éléments de l'ensemble d'arrivée de cette même correspondance. On progresse vers la solution en établissant des liens certains entre les éléments de l'ensemble de départ et les éléments de l'ensemble d'arrivée. L'élimination de liens possibles est très utile ; par exemple si l'on sait que DMI, le degré minimum pour l'ensemble de départ, vaut 1 et qu'il ne reste plus qu'un seul lien possible, alors ce lien est certain. Plus on a de degrés connus et plus ces méthodes sont capables d'éliminer des possibilités et de trouver des certitudes ; démontrer qu'une injection est en réalité une bijection augmente ainsi considérablement l'efficacité de ces méthodes.

Le problème est résolu quand on a trouvé des liens certains entre les éléments des ensembles de départ et d'arrivée de toutes les correspondances qui respectent les degrés, les graphes et les cliques éventuellement définis et qui donnent la valeur VRAI à toutes les contraintes. On peut vouloir chercher toutes les solutions ou s'arrêter à la première solution trouvée, il suffit de préciser son choix au système. On peut aussi prouver que le problème n'a pas de solution si l'on montre que tous les choix permis conduisent à une contradiction.

3. Les règles qui permettent de résoudre un problème

Un problème est résolu en utilisant les connaissances qui permettent d'atteindre la solution en créant de nouvelles contraintes, en découvrant une contradiction, en éliminant des liens possibles d'un élément de départ ou d'arrivée d'une correspondance, en trouvant des liens certains pour un de ces éléments, etc. Il existe pour cela un certain nombre de règles ; elles sont générales, car valables pour n'importe quel problème, mais elles peuvent être sans intérêt pour certains de ces problèmes. Quand on les donne au système, elles ne contiennent que l'indication de ce qui est nécessaire pour qu'on ait le droit de les appliquer, mais pas de métaconnaissances sur leur mode d'emploi, comme des indications sur les situations où il est bon de les appliquer, ni sur celles où il ne faut pas les utiliser, non parce que ce serait illégal, mais parce que cela ne servirait à rien.

Nous allons prendre pour exemple une règle , que nous appellerons R1, qui exprime que si l'on a une contrainte de la forme X=Y et si une borne supérieure de X est inférieure à

une borne inférieure de Y, on a alors une contradiction.

Nous définissons d'abord des éléments qui peuvent apparaître dans la règle :

CONTRAINTE : R

TERME : X

TERME : Y

Nous indiquons ensuite des ensembles d'instanciations possibles qui décrivent les liens entre ces éléments. Cela permet de les déterminer les uns à partir des autres :

X∈ UNDESFILS(R)

Y=AUTREFILS(R, X)

UNDESFILS(R) a pour valeur l'ensemble des termes qui dépendent de la connective principale de l'expression R, deux si la connective est binaire. Si la connective est binaire, AUTREFILS(R, X) a pour valeur le terme de R qui n'est pas X.

Nous avons aussi des conditions pour que l'on ait le droit d'appliquer la règle, ici :

COPRI(R)=@EQ

COPRI(R) ayant pour valeur la connective principale de l'expression R qui doit donc ici être l'égalité. Rappelons que dans MACISTE les objets constants ont leur nom précédé du caractère '@' dans le cas où ils pourraient être pris pour une variable.

Une deuxième condition est :

SUP(X) < INF(Y)

SUP(X) a pour valeur une borne supérieure de l'expression X et INF(Y) une borne inférieure de l'expression Y. Il n'est pas nécessaire que ce soient les meilleures bornes possibles, tout dépend de la qualité des expertises qui vont les déterminer. Plus elles sont précises et plus la règle aura des chances de s'appliquer. S'il y a beaucoup de variables, une détermination très précise peut s'avérer trop coûteuse. Déterminer la méthode que l'on utilisera pour évaluer ces bornes est d'ailleurs une décision de monitoring.

Par exemple, considérons la contrainte :

F{'A'} + F{'B'} + F{'C'} + F{'D'} = F{'E'}

si les valeurs des éléments A, B, C, D et E sont positifs et inférieurs à 10, si de plus ils sont en disjonction (ou bien AMA pour F vaut 1), la règle précédente s'appliquera si la fonction INF est capable d'utiliser les disjonctions. En effet le premier membre vaut au moins 1+2+3+4 soit 10 alors que le deuxième membre vaut au plus 9.

Cette règle a enfin une action : CONTRADICTION, qui indique que le sous-problème en cours n'a pas de solution.

Décrivons maintenant plus rapidement une règle, R2, un peu plus complexe. Nous avons d'abord la définition des types des variables :

CONTRAINTE : R

TERME : X

TERME : Y
TERME : X_i
et les ensembles d'instanciations possibles :
$X \in$ UNDESFILS(R)
Y=AUTREFILS(R, X)
$X_i \in$ ARGUMENTS(X)
Nous avons ensuite la condition :

$$\text{MATCHE}(X, \text{SOMME}[X_1, X_2, \ldots, X_n])$$

qui indique que le terme X est la somme de n termes, n étant quelconque supérieur à 1. Nous avons alors une action qui est la création d'une nouvelle contrainte :

$$\text{CONTRAINTE } X_i \leq \text{SUP}(Y) - \text{INF}(\text{SOMME}[X_1, X_2, \ldots X_{i-1}, X_{i+1}, \ldots X_n])$$

Cette nouvelle contrainte est beaucoup plus simple que la précédente puisqu'elle ne comporte que le terme Xi de la contrainte initiale, le deuxième membre de la contrainte est en effet une constante. Par exemple si la contrainte R est :

$$F\{\text{'A'}\} + 3*F\{\text{'B'}\} + F\{\text{'C'}\} + F\{\text{'D'}\} = F\{\text{'E'}\}+7$$

si les valeurs de F{'A'}, F{'C'} et F{'D'} sont supérieures ou égales à 1, si elles sont en disjonction et si la valeur de F{'E'} est inférieure ou égale à 9, on aura la nouvelle contrainte :

$$3*F\{\text{'B'}\} \leq 9+7-(1+2+3)$$

soit $3*F\{\text{'B'}\} \leq 10$ ou enfin $F\{\text{'B'}\} \leq 3$.

Remarquons qu'il est utile de créer une deuxième famille de conditions qui sont de nature heuristique. Mais alors ce ne sont plus des connaissances, mais des métaconnaissances puisqu'elles restreignent les cas où il est bon d'appliquer la règle. On a le droit d'appliquer la règle si elles sont fausses, mais cela risque d'être inutile. Pour la règle R2, nous pouvons avoir comme condition de ce type qu'il n'y ait qu'une seule variable dans l'expression X_i. En effet, la contrainte engendrée indique qu'une expression est inférieure ou égale à une constante. La nouvelle contrainte donnera ainsi une bonne supérieure dans le cas où il y a une seule variable, ce qui permet souvent d'éliminer plusieurs liens possibles alors qu'une inégalité avec plus d'une variable serait moins facile à utiliser. On ne gardera pour exécution ultérieure que les essais qui correspondent à une expression X_i ayant au plus une variable. Remarquons que le cas où il n'y a pas de variable peut aussi être intéressant, car il peut conduire à une contrainte dont la valeur sera FAUX, ce qui donnera alors une contradiction. Actuellement, les conditions heuristiques d'une règle sont données au système, mais elles devront ultérieurement être découvertes et éventuellement modifiées par le système qui apprendra ainsi à mieux sélectionner ses essais. C'est encore au monitoring de gérer la création et la modification des conditions heuristiques.

Le système dispose actuellement d'une soixantaine de règles qui peuvent être données

indépendamment les unes des autres. Chacune d'entre elles a plusieurs utilisations possibles selon ce qui lui est donné au moment de l'instanciation. Par exemple, la règle R1 peut être utilisée chaque fois que l'on a une nouvelle contrainte d'égalité, et aussi chaque fois que l'on augmente la valeur de la borne inférieure d'une variable (on prendra comme terme Y le membre d'une égalité qui la contient) et enfin chaque fois que l'on diminue la valeur de la borne supérieure d'une variable (on prendra cette fois comme terme X le membre d'une égalité qui la contient). Il est possible de l'appliquer dans d'autres situations, mais cela ne donnera en principe rien d'utile.

Si l'on utilisait systématiquement toutes les règles dans toutes les situations où elles ont une chance, même très faible, de donner un résultat, on aurait une métacombinatoire énorme. En contrepartie, on aurait dans certains cas des solutions très élégantes demandant une combinatoire très réduite. Il est nécessaire d'avoir des métaconnaissances capables de se servir des règles à bon escient et de faire un arbitrage judicieux entre le temps perdu à faire de la combinatoire et celui à faire de la métacombinatoire.

4. Le langage procédural de résolution de problèmes

Un expert résout généralement un problème en allant directement à la solution une fois qu'il a identifié le type du problème. J'ai voulu simuler ce comportement en donnant un langage dans lequel on définit des plans de résolution d'une famille de problèmes. Dans une première étape, je construis ces plans ; mais le langage dans lequel ils sont écrits devra permettre ultérieurement au système soit de les modifier, soit d'en créer de nouveaux. Ce langage permet d'indiquer au système à la fois quelles actions il doit considérer et dans quel ordre il devra les exécuter.

Pour chaque type d'événement, on définit une séquence d'ordres. Un événement peut être la création d'un lien certain entre un nœud de l'ensemble de départ et un nœud de l'ensemble d'arrivée, la destruction d'un lien possible entre deux tels nœuds, la création d'une nouvelle contrainte, la détermination de la valeur d'un degré pour une des correspondances, etc. De plus, pour initialiser le processus, une séquence particulière doit s'exécuter au début de la résolution d'un problème. Pour chaque séquence, sauf celle d'initialisation, on appelle foyer de l'événement l'élément sur lequel a porté l'événement, par exemple le nœud de l'ensemble d'arrivée auquel on a ajouté un lien certain ou la nouvelle contrainte. La valeur de ce foyer est souvent utilisée dans l'interprétation d'un ordre. Il existe quatre types d'ordres :

1. L'ordre indiquant d'envisager d'appliquer une règle : ENVISAGE R A P E. On envisage d'exécuter la règle R, la variable A, qui figure dans la règle R, a pour valeur le foyer. L'expression P définit la priorité avec laquelle on va exécuter ultérieurement cet essai. Elle

peut contenir des variables, en particulier la variable A, mais aussi contenir n'importe quelle autre variable définie dans la règle R. Il est de plus possible de définir d'autres variables par des expressions qui figurent alors dans l'ensemble E. Un exemple simple de tel ordre est :

ENVISAGE R6 S @MOYEN

qui dit d'appliquer la règle R6 avec une valeur de la priorité @MOYEN ; la variable S est une variable de la règle R6 qui aura comme valeur le foyer. Par exemple, si l'événement était la création d'une contrainte, S aura pour valeur cette contrainte, s'il était la découverte d'une nouvelle borne supérieure pour un nœud de départ D d'une correspondance, la valeur de S sera alors ce nœud D. Cet essai est mis en attente et sera exécuté quand on aura fini d'effectuer tous les essais dont la priorité est supérieure à @MOYEN. Ici, l'ensemble E est vide.

Donnons un exemple un peu plus complexe, basé sur la règle R2 que nous avons vue plus haut. Elle indique que si l'on a une contrainte R de la forme :

$$SOMME[X_1, X_2,\ldots, X_n]=Y$$

alors on a n nouvelles contraintes possibles :

$$X_i \leq SUP(Y) - INF(SOMME[X_1, X_2,\ldots X_{i-1}, X_{i+1},\ldots X_n])$$

La priorité P d'envisager un de ces essais, par exemple celui correspondant à l'essai qui crée une borne supérieure pour l'expression X_i figurant dans la somme, sera :

PROG(COND[Q=2 :@ELEVE, Q=3 :@ASSEZELEVE, Q=4 :@MOYEN,
Q=5 :@ASSEZBAS, Q=6 :@TRESBAS], COND[M>1,M≤8 :1, M>8 :2, :0])

où Q et M sont définis par deux éléments dans E : Q=CARVAR(R), donc Q est le nombre de variables de R, et M=FACTEUR(X_i). La fonction FACTEUR a pour valeur le facteur constant qui multiplie éventuellement la variable dans son argument, et la valeur 0 si le terme X_i n'est pas le produit d'une expression par une constante. Si X_i est l'expression 5*V, la valeur de M est 5 ;par contre, si X_i est l'expression W+T, sa valeur est 1.

La fonction PROG(A,B), où A est une priorité et B un entier, a pour effet d'augmenter de B unités la valeur de la priorité A (si B est négatif, il la diminue de -B unités). Par exemple, PROG(@ASSEZBAS,2) a pour valeur @ASSEZELEVE. Si B est nul, qui est ici la valeur par défaut de M, la valeur de A est inchangée. L'utilisation de cette fonction a l'intérêt de favoriser les essais où le terme premier membre de l'inégalité contient un coefficient, et ce d'autant plus que ce coefficient est grand. En effet, $10*V \leq 13$ donne une restriction très sévère sur les valeurs de V qui est inférieur ou égal à 1 alors que $V \leq 13$ est bien moins contraignant.

2. L'ordre indiquant que l'on peut envisager d'envisager une règle : ENVENV R A P E PP EE. Le travail se fait maintenant en deux étapes et chacune a sa propre priorité : on regarde d'abord s'il y a lieu d'envisager de faire l'essai, ce qui détermine une première priorité P, celle d'envisager l'essai. Quand on fait plus tard cet essai effectivement, on déterminera une

nouvelle priorité PP qui sera celle d'exécuter réellement la règle. Le but est de ne pas passer beaucoup de temps immédiatement pour considérer l'intérêt de faire un essai plus tard s'il y a de grandes chances que cet essai n'ait qu'un intérêt moyen alors que l'examen précis de chacun des essais qu'il entraîne est coûteux. C'est en particulier ce qui se passe avec la règle R2 précédente qu'il vaut mieux définir avec ENVENV. En effet, envisager d'appliquer une règle est souvent une opération assez coûteuse en elle-même, car on peut être amené à chercher la valeur de certains arguments. Par exemple si l'on veut savoir si l'on va évaluer une contrainte modulo un certain nombre, il faut déterminer quels nombres on va considérer, il peut y en avoir plusieurs qu'il faudra trouver et, en plus, pour chacun on devra calculer la priorité de l'essai. De même dans l'exemple de la règle R2, il faut considérer chacune des deux valeurs possibles de X, terme qui contient une SOMME, puis chacun des éléments de cette somme. Cela peut nécessiter beaucoup de temps et il vaut mieux se demander rapidement d'abord quand cela vaut la peine de le faire. L'ordre ENVENV permet de différer ces calculs coûteux quand ils risquent de ne pas avoir un intérêt élevé ; si cela se trouve, on ne les examinera jamais parce que l'on aura trouvé la solution sans eux. Mais il faut bien alors deux priorités : P est celle de l'examen approfondi des modalités d'exécution de la règle et PP est celle l'exécution réelle de la règle.

Les arguments R et A sont définis comme précédemment mais P indique la priorité avec laquelle on va envisager (et non plus exécuter) l'essai et E définit éventuellement des variables apparaissant dans P. PP indique avec quelle priorité on va exécuter l'essai et EE contient des expressions utiles pour évaluer PP. Voici une nouvelle définition de l'ordre qui indique comment utiliser la règle R2 quand le foyer est une nouvelle contrainte :

ENVENV R2 R
COND[Q=2 :@ELEVE, Q=3 :@ASSEZELEVE, Q=4 :@MOYEN, Q=5 :@BAS,
Q=6 :@TRESBAS]
Q=CARVAR(R)
PROG(COND[Q=2 :@ELEVE, Q=3 :@ASSEZELEVE, Q=4 :@MOYEN,
Q=5 :@ASSEZBAS, Q=6 :@TRESBAS], COND[M>1,M≤8 :1, M>8 :2, :0])
Q=CARVAR(R), M=FACTEUR(X_i)

Nous voyons que l'examen sera bien plus rapide dans le cas ENVENV puisque X_i n'y apparaît pas et que les essais n'ont pas à être définis tout de suite. Les X_i ne sont considérés que quand on envisage la règle, pas quand on envisage de l'envisager. Il suffit donc de compter les variables dans la contrainte foyer et de placer l'examen approfondi à une étape ultérieure qui sera d'autant plus proche que le foyer contient peu de variables. Des priorités définies en utilisant le nombre de variables d'une contrainte sont fréquentes : on préfère toujours les contraintes qui ont peu de variables.

E contient un seul élément permettant de calculer la variable Q qui est le nombre de

variables apparaissant dans R. Cela signifie que l'on va envisager la règle R2 avec pour valeur de la variable R de cette règle la nouvelle contrainte engendrée, que la priorité pour l'envisager sera élevée si elle ne contient que deux variables et très basse si elle en contient 6, la fonction CARVAR calculant le nombre de variables différentes apparaissant dans l'expression qui est son argument. Si la contrainte a une seule variable ou plus de six variables, la règle R2 ne l'examinera pas du tout puisque la priorité ne sera pas définie. Quand on exécute un ordre, on met en attente d'exécution les essais ainsi déterminés, chacun avec leur priorité P qui est évaluée, en tenant compte éventuellement des expressions qui sont dans l'ensemble E.

L'utilisation d'un ordre ENVENV se fait donc en deux temps : d'abord une première estimation indique la priorité d'envisager de faire l'essai (ou de l'éliminer totalement). Quand on en vient plus tard à cette priorité, on crée un ou plusieurs essais bien définis qui ont chacun leur propre priorité. On exécutera chacun d'entre en fonction de cette dernière priorité. Avoir un premier filtre évite de perdre du temps à déterminer immédiatement les arguments et l'intérêt de multiples essais qui de toutes les façons ne sont certainement pas très intéressants.

3. L'ordre de test. Cet ordre est un ensemble de conditions qui indiquent d'exécuter une certaine séquence d'actions si toutes les conditions sont vraies. Ces conditions utilisent la variable dont la valeur est le foyer. Quand on a exécuté toutes les actions de cette sous-séquence, on revient à la séquence principale. Par exemple quand on a une nouvelle contrainte, on regarde si elle ne comporte qu'une variable. Si cela est le cas, on exécute une règle qui regarde pour toutes les valeurs possibles de cette variable si elles rendent la contrainte vraie ou fausse en les substituant successivement. Si pour une des valeurs possibles, la contrainte est fausse, on élimine cette valeur comme possible. C'est ainsi que si la contrainte est F{'A'}<4, on éliminera les liens possibles entre le nœud correspondant à 'A' et les nœuds d'arrivée correspondants à 0, 1, 2, et 3. Ceci n'est bien évidemment utile que s'il n'y a qu'une seule variable dans la contrainte et on gagne du temps en n'essayant pas cette règle dans tous les cas où elle ne s'applique pas. Mais si elle peut s'appliquer, il faut l'appliquer rapidement.

4. L'ordre d'arrêt. Il indique qu'il est inutile de poursuivre l'exécution d'une séquence. Cela est seulement utile quand on a une sous-séquence issue d'un test et que l'on ne désire pas revenir à la séquence principale. Par exemple, après avoir fait l'examen précédent d'une contrainte contenant une seule variable, on met un ordre d'arrêt, parce que l'on a tiré tout ce que l'on pouvait de cette contrainte et il est inutile, voire néfaste, de faire davantage d'essais avec cette contrainte.

Actuellement, ces ensembles de séquences d'actions sont définis pour des groupes de problèmes manuellement. Mais le but à long terme est qu'ils soient déterminés a priori en examinant les caractéristiques du problème à partir de sa définition formelle et améliorés à partir de ce que le monitoring a observé au cours des premières résolutions de problèmes de la famille. Le système se comportera de plus en plus comme un expert au fur et à mesure qu'il construira des séquences d'actions de plus en plus efficaces.

5. Implémentation du système

MALICE reçoit l'énoncé d'un problème et les ensembles d'ordres à exécuter pour chaque événement qui peut se produire : enlèvement d'un lien possible, ajout d'un lien certain, création d'une nouvelle contrainte, détermination de la valeur de AMI pour une correspondance, etc. Il traduit en programmes les règles avec leurs modes d'emploi ainsi définis par les ordres. Cela conduit à plusieurs programmes pour la même règle. Par exemple, pour la règle R2, nous pouvons décider d'envisager de l'envisager quand on a une nouvelle contrainte R. On crée alors trois programmes, l'un pour quand on envisage de l'envisager, un pour quand on l'envisage (et qui détermine toutes les situations où on pourra l'exécuter) et un pour quand on l'exécute effectivement. Mais on peut aussi décider de s'y intéresser quand la borne supérieure d'une des variables figurant dans l'expression Y va être changée, car cela va diminuer peut-être la valeur de SUP(Y). On va peut-être aussi vouloir envisager de l'envisager puis de l'envisager, d'où deux nouveaux programmes, celui où on l'exécute étant le même que précédemment. Mais on peut aussi vouloir la considérer quand la borne inférieure d'une des variables apparaissant dans X est augmentée, car cela va augmenter la valeur de INF(SOMME[X_1, X_2,…X_{i-1}, X_{i+1},…X_n]). Si nous envisageons de l'envisager, puis l'envisageons, cela fait encore deux nouveaux programmes, soit au total sept programmes liés à cette unique règle. Naturellement, ces programmes sont engendrés automatiquement. On inclut dans ces programmes, certaines des informations figurant dans la définition de la règle, mais pas forcément toutes : par exemple, quand on envisage d'envisager une règle, on ne met pas les actions qui sont effectuées quand toutes les conditions sont satisfaites. De même, on ne met que les conditions que l'on peut évaluer sans utiliser les variables qui peuvent être créées avec des valeurs multiples par les expressions d'instanciation parce qu'elles sont définies comme éléments d'un ensemble, ce qui était le cas de X_i pour la règle R2. Par contre, quand on veut envisager un essai, on inclut les expressions qui permettent de définir toutes les variables nécessaires.. Comme les programmes ainsi créés peuvent décider de stocker le nouvel essai, le système y inclut aussi le calcul de la valeur des priorités associés à ENVENV ou à ENVISAGE selon les cas.

Par exemple, si l'on connaît un nœud D, on aura les contraintes qui contiennent ce

nœud en ajoutant R∈CONTRAINTES(D). Dans le cas de ENVISAGE, on a besoin de définir la valeur de toutes les variables qui seront nécessaires pour l'exécution, par exemple la valeur de l'expression valeur de X_i pour la règle R2, c'est à dire qu'il faudra aussi ajouter une expression permettant de définir X_i en balayant les éléments de la somme ; cela est inutile pour le cas ENVENV qui ignore la variable X_i. C'est pour cela que l'on y gagne à sérier les examens, le cas ENVENV étant bien moins coûteux que le cas ENVISAGE.

La combinatoire est plus restreinte que dans la version précédente de MALICE puisque les nombreuses règles permettent de réduire la taille de l'espace de recherche à l'aide de nouvelles contraintes. Mais surtout, la métacombinatoire est diminuée de façon appréciable, car on sélectionne les essais de façon plus stricte puisque l'on a plus de possibilités pour indiquer comment les choisir. Enfin, on a une explication et une méta-explication de ce que l'on a fait, ce qui sera précieux pour avoir un monitoring et un apprentissage efficaces. Le système est rapide puisque toutes les règles sont compilées sous autant de formes que le besoin s'en fait sentir. La seule partie interprétée est l'enchaînement de l'exécution des ordres liés à l'apparition d'un événement. Cela ne pénalise guère le système car, alors que l'exécution d'un ordre peut demander pas mal de temps, le temps d'interprétation est négligeable. Il est intéressant d'interpréter, car la méta-expertise de monitoring pourra ultérieurement changer dynamiquement l'ordre de l'interprétation en décidant de supprimer certains essais, d'en examine d'autres prioritairement, de transformer un ENVISAGE en ENVENV ou le contraire, etc.

Le système trouve ainsi les solutions des problèmes plus rapidement qu'avec l'ancienne version de MALICE, qui limitait déjà sérieusement la métacombinatoire. C'est normal, puisqu'il est métaguidé de façon bien plus stricte par les successions d'ordres liés à chaque événement. La difficulté est de les donner. Il est admissible de les donner manuellement dans la phase de mise au point du système. Mais la méthode n'aura tout son intérêt que quand ces séquences seront déterminées automatiquement par le système lui-même.

6. Développements futurs

Maintenant que le système fonctionne avec les métaconnaissances données manuellement et un monitoring statique et prévu à l'avance, il faut lever ces limitations. Le système devra d'abord être capable de déterminer automatiquement les séquences d'actions à entreprendre après chaque événement en tenant compte des règles connues et de l'énoncé formel du problème à résoudre. Il devra aussi pouvoir s'affranchir de cet ordre s'il lui paraît maladroit ; il aura également à noter des surprises parmi ce qui s'est passé de façon à les utiliser ultérieurement pour apprendre de meilleures séquences d'actions.

J'ai déjà fait quelques expériences pour découvrir comment construire des séquences d'actions. Elles sont basées sur l'examen des règles. En effet, la décision d'utiliser une règle dépend de l'importance de ses conséquences, des conditions qui permettent son déclenchement, du coût de son exécution et de ses chances de succès. Il est facile de déterminer de façon grossière les conséquences d'une règle en l'examinant : création d'une contrainte, élimination de liens possibles, création de liens certains, découverte d'une contradiction…Il est plus difficile, mais possible, de les déterminer avec plus de précision, par exemple de voir qu'une règle crée une contrainte d'égalité linéaire contenant quatre variables. Il est également possible d'évaluer le coût de l'exécution d'une règle à partir de celui d'éléments de base comme de normaliser une contrainte ou de déterminer la borne supérieure d'une expression ; il faut naturellement tenir compte de la taille des ensembles que l'on parcourt. Le plus difficile est d'évaluer a priori les chances de succès d'appliquer une règle, car cela dépend beaucoup des caractéristiques et des données de chaque problème. C'est là que le monitoring dynamique peut être très utile.

Une fois que l'on connaît les conséquences, les conditions, le coût et les chances de succès d'une règle, il doit être possible de définir une méta-expertise générale pour déterminer les événements où elle doit être déclenchée et les priorités qui y sont associées. L'apprentissage doit ensuite adapter cette décision initiale en fonction de ce qui a été observé au cours du monitoring.

Une difficulté du monitoring dynamique est le problème de l'attention : que doit-on regarder. Il faudra choisir quelles anomalies on va surveiller, on ne peut pas tout observer. Les méthodes utilisées par SEPIAR [Parchemal 88] seront une source d'inspiration. Quand tout se déroule normalement, on examine de moins près ce qui se passe ; par contre, quand la solution a l'air d'être plus lointaine que prévu, il devient utile de commencer à faire des statistiques sur l'utilisation des règles. Le système doit se rendre compte que l'utilisation d'une certaine règle, en général considérée comme utile, conduit toujours à un échec dans le cas présent. On peut immédiatement en limiter l'utilisation au cours de la résolution, mais il est plus difficile d'en tirer des conclusions générales pour les problèmes futurs, car pour cela il faut comprendre pourquoi une règle en général utile a un rôle néfaste dans certaines circonstances. Pour cela, le système devra avoir une théorie de la résolution de problèmes et pouvoir établir des propriétés du type : la règle R5 est intéressante pour les problèmes où il y a peu de contraintes et où chaque variable a moins d'une dizaine de valeurs possibles. Le système devra utiliser des méthodes de démonstration de théorèmes dans des environnements où les lois sont très incertaines. C'est comme cela que j'ai supposé qu'il valait mieux appliquer la règle R2 à des termes où la variable est multipliée par un nombre le plus grand possible. Mais que veut dire un nombre grand ? Dans certains cas, 9 est considéré comme grand alors que dans d'autres il est petit !

Un autre aspect à approfondir est celui des expériences. Le système devra être capable

de lancer des essais en comparant ce que donnent deux méthodes. Cela aidera à choisir celle qui conduit aux meilleures performances dans des cas où les connaissances du système sur la théorie de la résolution de problèmes ne lui permettent pas de trancher. Un autre type d'expérience est de faire des essais en «ratissant» plus large que d'habitude, par exemple en acceptant de faire des essais que l'on s'interdirait de faire en temps normal. En effet, si l'on ne se permet pas de relâcher les contraintes, on fait toujours la même chose et l'on ne se rend pas compte qu'il existe des situations où des méthodes considérées comme inappropriées peuvent être en réalité les meilleures. Il faut se donner la chance d'avoir parfois de bonnes surprises, d'avoir du succès là où l'on ne s'y attend pas. Si l'on a un comportement d'expert qui va droit à la solution sans examiner d'autres chemins, on risque de se fixer sur des maxima secondaires. Une fois un problème résolu, un système peut se payer le luxe de voir s'il n'y avait pas d'autres choses à faire dans les périodes où on ne lui a pas donné de tâche précise.

7. Le métamonitoring

Le but du monitoring est de faire gagner du temps au système. Il est donc hors de question qu'il passe un temps considérable à faire un choix qui ne fera gagner qu'un temps minime. Il faut donc que le monitoring soit contrôlé comme le résolveur de problèmes, il doit être lui-même monitoré. Nous devons toutefois écarter le risque d'une ascension infinie des niveaux méta. Nous utilisons d'abord les mêmes méthodes pour monitorer la résolution d'un problème et pour monitorer le monitoring ; on a donc ainsi un seul fichier de métaconnaissances. Mais nous les restreignons dans le cas du métamonitoring de façon à éviter sûrement d'y passer un temps. Par exemple, nous pouvons limiter les risques de dérives en interdisant, dans le cas du métamonitoring, toute possibilité de changer les séquences d'actions définies au départ. Il est utile de faire des propositions de changements, mais elles ne seront pas utilisées automatiquement pour changer les méta-expertises, elles serviront seulement à aider l'utilisateur qui souhaite changer certaines séquences de métamonitoring quand le monitoring fonctionne mal.

Dans le cas du métamonitoring, les événements ne sont plus la découverte d'une nouvelle contrainte ou l'enlèvement d'un lien possible. Ils sont l'apparition d'une surprise, par exemple l'échec d'une méthode reconnue comme sûre ou la constatation que l'utilisation des conséquences d'un événement demande un temps bien plus grand que ce qui est prévu ou habituel. Pour chaque événement de monitoring, comme un écart entre le temps prévu et le temps effectué pour une action, une séquence de méta-actions sera prévue, par exemple de rendre cette action moins souvent applicable ou de réduire sa priorité ou même de ne plus du tout l'exécuter par la suite. Le rôle du métamonitoring est de vérifier que ces méta-actions ne

prennent pas un temps inconsidéré, auquel cas il prendra sur ces séquences de méta-actions des décisions analogues à celles prises sur les actions. Nous éviterons toutefois d'y inclure des méta-actions qui peuvent être dangereuses parce que difficilement contrôlables. Un autre rôle important du métamonitoring sera de contrôler les expériences que le système fait pour vérifier une hypothèse ou déterminer la meilleure valeur de certains coefficients. Sans précautions, cela peut amener à des pertes de temps considérables si l'on fait des essais innombrables et sans grand intérêt.

L'expertise de métamonitoring doit être simple pour qu'elle ne demande pas trop de temps et que l'on puisse facilement s'assurer de sa sécurité. Avec ce garde fou, il devra être possible de faire un système complètement autonome pour lequel on n'aura pas les risques de dérive auxquelles a dû faire face un système comme EURISKO [Lenat 83]. Le niveau supérieur doit être à la fois simple et inchangeable. Dans un premier temps, les opérations «dangereuses» seront seulement proposées ; ce sera à l'utilisateur de décider s'il y a lieu de les implémenter. Pour arriver à supprimer l'intervention humaine, il faudra passer au stade supérieur, celui du métamétamonitoring qui vérifierait que les changements proposés ne sont pas trop dangereux, qui surveillerait de près ce qu'ils donnent et qui aurait des garde-fous simples et sûrs pour contrôler l'ensemble du fonctionnement du système.

8. Références

[Laurière 76] Laurière J.-L., *Un langage et un programme pour énoncer et résoudre des problèmes combinatoires*, Thèse de l'Université Paris6, 1976.

[Laurière 78] Laurière J.-L., *A Language and a Program for Stating and Solving Combinatorial Problems*, Artificial Intelligence 10, 1978, 29-127.

[Laurière 96] Laurière J.-L., *Propagation de contraintes ou programmation automatique*, Rapport LAFORIA 96/19, 1996.

[Lenat 83] Lenat D., *EURISKO: A Program that Learns New Heuristics and Domain Concepts*, Artificial Intelligence 21, 1983, 61-98.

[Parchemal 88] Parchemal Y., *SEPIAR : un système à base de connaissances qui apprend à utiliser efficacement une expertise*, Thèse de l'Université Paris 6, 1988.

[Pitrat 96] Pitrat J., *Implementation of a Reflexive System*, Future Generation Computer Systems 12, 1996, 235-242.

[Pitrat 99] Pitrat J., *Une expérience de monitoring*, Rapport de recherche LIP6 1999/014, 1999.

[Schoenfeld 85] Schoenfeld A., *Mathematical Problem Solving*, Academic Press, 1985.

ARGOS: Un Démonstrateur qui Résout des Problèmes de Géométrie, Explique ce qu'il Fait et Affine ses Stratégies de Résolution en Apprenant à partir des Exercices déjà Trouvés

Jean Pierre Spagnol
(*) Crip5, Université Paris5, rue des saints pères, http://www.math-info.univ-paris5.fr ,
spaj@math-info.univ-paris5.fr

Résumé: L'objectif de cette recherche est la conception d'un système à base de connaissances, capable de résoudre de façon automatique des exercices de géométrie niveau collège ou lycée. Le système est ensuite capable de filtrer la base des faits déduits pour en produire, à partir du graphe de la démonstration, une solution rédigée en français adaptée au niveau de la classe considérée. Après chaque exercice résolu, ARGOS met à jour un certain nombre de stratégies en analysant la démonstration qu'il a produite. Dans cet article nous décrivons un certain nombre des techniques mises en œuvre, construction automatique de règles opérationnalisant les connaissances données de façon déclarative, apprentissage de stratégies de résolution, possibilité d'appels de conjectures en cours de démonstration, rédaction de preuve en français. Conçu à partir du système Muscadet, Argos essaye d'en garder l'esprit qui est basé sur le principe de la déduction naturelle. Ce système est destiné à une utilisation en EIAO.

Mots clés: DAT, apprentissage, automatisation du raisonnement mathématique, EIAO, explication.

1. Introduction

Ce système est conçu à partir du système Muscadet [Pastre 84] réécrit en langage Prolog [Pastre 99], de nom Puscadet (ou Muscadet version 2). Il est ainsi possible grâce à la bivalence du langage Prolog d'exprimer les connaissances de façon déclarative et de faire du procédural quand cela est nécessaire, en particulier il devient possible d'appeler des procédures de calcul ce qui est parfois nécessaire en géométrie.

Il est destiné à une utilisation en EIAO pour permettre à un élève de lycée d'améliorer son intuition et d'étayer sa recherche en lui permettant de proposer des conjectures vérifiables (dans les limites des compétences d'Argos) par le système à partir d'un énoncé donné. Il est

aussi destiné aux professeurs en leur permettant d'exprimer les connaissances mathématiques ainsi que des savoir-faire heuristiques, sur un thème donné, de façon déclarative et de concevoir des exercices s'y rapportant. De plus un gros effort d'explication est requis pour ce système. Il apparaît en effet difficilement concevable qu'un démonstrateur, résolve des exercices et démontre des théorèmes sans que les preuves puissent être lisibles et vérifiables par un humain. Dans cet article, après de brèves remarques générales, nous montrerons le fonctionnement général et nous l'illustrerons par un exemple détaillé. Nous montrerons ensuite comment le système reconstruit l'arbre de preuve à partir des faits déduits et à l'aide de celui-ci produit une preuve rédigée en français de l'exercice proposé. Nous évoquerons ensuite la possibilité pour le système de construire automatiquement des règles à partir du savoir mathématique exprimé de façon déclarative. Nous examinerons la possibilité qu'a le système de lancer des appels de conjectures, en cours de recherche, si certaines conditions sont réunies dans la base de faits et aussi comment ARGOS, après avoir résolu un exercice, construit une base de données relative à l'exercice et apprend de nouvelles connaissances de savoir-faire en mettant à jour des stratégies qu'il a lui-même mis au point de façon automatique en résolvant des exercices. Une utilisation du système est aussi possible dans le cadre d'un EIAO, nous montrerons de quelle manière. Nous conclurons en évoquant la principale difficulté des démonstrateurs qui consiste en la mise au point de réelles et efficaces heuristiques de création d'objets.

2. Fonctionnement général du système

L'idée de départ était de construire un démonstrateur robuste et fiable et d'utiliser ses potentialités comme support permettant un apprentissage de la géométrie par un élève de lycée ou collège. ARGOS est un système à base de connaissances, conçu à partir du système MUSCADET de Dominique Pastre et travaillant dans le domaine de la géométrie. Il se situe dans la continuation des travaux de JM Bazin [Bazin 93] et Miguel Pintado [Pintado 94] et cherche entre autre à produire des preuves compréhensibles par un humain. Il s'inscrit parmi les travaux actuels en DAT, Explication et EIAO comme par exemple Projet Mentoniezh de D Py [Py 96], Cabri-Euclide de V Luengo [Luengo 99], PROVERB par H Horacek [Horacek 99] ou bien encore les travaux d' Erika Melis [Melis-Uron 99].
La spécificité du travail résulte dans le fait que le système est capable de résoudre un exercice de géométrie, d'affiner ses stratégies de recherche à l'issue de la résolution d'un nouvel exercice et de produire, automatiquement, une preuve rédigée en français adaptée à un certain nombre de désiderata de l'utilisateur. Le système est capable en présence de certaines situations classiques de faire des appels internes de conjectures, à partir de règles, pour aller rechercher et déduire des informations non présentes dans la base de faits.

Il est aussi possible de faire fonctionner ARGOS, à partir d'un exercice donné, comme outil de vérification de conjectures proposées par un élève à partir de l'observation de la figure, l'élève pouvant introduire de nouveaux objets ou non. Si la conjecture est vérifiée le système peut renvoyer à l'élève une explication restreinte ou globale de celle-ci. Une utilisation du système comme compagnon d'apprentissage est en cours de mise au point. Nous allons maintenant détailler les différentes étapes de résolution.

2.1. Lecture et analyse de l'énoncé

Le système va lire un énoncé donné au système sous forme d'implication, puis en extrait les concepts présents en hypothèse et en conclusion. De plus dans le même temps il procède à l'élimination de certains symboles fonctionnels pour leur donner des noms internes. Argos récupère aussi les souhaits de l'utilisateur quant au déroulement de la recherche et au type de preuve rédigée voulue (niveau de précision et niveau scolaire). L'utilisateur a ainsi la possibilité de paramétrer le système. Il peut définir des domaines prioritaires induisant la preuve dans tel ou tel sens auquel cas les règles d'autres domaines seront décalées, il peut interdire les règles d'un ou plusieurs domaines, comme par exemple les règles de trigonométrie très spécifiques et alourdissant la recherche.

2.2. Sélection et chargement des règles adéquates, dans un certain ordre.

Le système va alors constituer une liste de règles en fonction des concepts présents, des connaissances mathématiques dont il dispose et des stratégies connues relatives aux différentes conclusions demandées. Il charge en premier les règles universelles nécessaires indépendamment de tout énoncé (manipulation logique de l'énoncé, sortie du système, cohérence au niveau des objets et règles nécessaires quel que soit l'énoncé), puis les règles d'introduction des objets initiaux, les règles de condition suffisante de la conclusion permettant de conclure rapidement si un critère suffisant de la conclusion est utilisable dans la base de faits. Il charge ensuite la ou les listes de règles, appelées stratégies, ayant été constituées à partir de toutes les résolutions d'exercices de mêmes buts que l'exercice en cours.

ARGOS doit maintenant charger les règles qu'il juge utile pour la démonstration et qui n'ont pas encore été choisies. Ce sont les règles obligatoires avec la terminologie Muscadet. En fonction des concepts présents en hypothèse il charge des règles de démarche en avant. Ces règles extraient de la figure toutes les configurations prégnantes sans être guidées par le but. Pour les concepts figurant en conclusion il cherche celles de démarche en fonction du but.

Le démonstrateur peut ensuite charger les règles appelant une conjecture. Ces règles se déclenchent si le système détecte des configurations incomplètes et intéressantes en fonction

de ce qu'il y a à démontrer. Ceci est une option que peut choisir un utilisateur et il doit alors l'indiquer au système.
Le système va charger enfin des règles un peu plus dangereuses de création d'objets dont la présence n'est pas vue comme nécessaire par simple analyse de l'énoncé mais qui sont susceptibles de servir à l'application de règles utiles pour l'exercice en cours. Un utilisateur a la possibilité s'il le désire de mettre le système en mode apprentissage ce qui va consister à charger à la suite toutes les autres règles à priori plus éloignées de l'exercice en cours.
Le " moteur d'inférences " entre alors en action et va chercher à "instancier" successivement chacune des règles ayant été placées dans la liste des règles actives, dans l'ordre où elles se présentent en recommençant au début chaque fois que l'une d'entre elles se déclenche. L'utilisateur peut avoir, s'il le désire, donné un temps de recherche maximum pour l'exercice. Il peut avoir donné pour chaque règle chargée un temps de recherche maximum au delà duquel, si la règle dépasse ce temps, le système décale la règle dans la liste des règles actives.

2.3. Déductions du système.

Un énoncé est écrit sous la forme : H_1 et H_2 et ….. et H_n => C_1 et C_2 et …. et C_k . Le système, à l'aide de règles, procède à un découpage de l'énoncé et commence à structurer la base de faits selon le principe de Muscadet qui correspond aussi à la démarche du mathématicien. La base de faits se structure en une partie Hypothèses où vont se rajouter tous les faits déduits à partir des hypothèses initiales par application des règles chargées, et d'une partie Conclusions où figurent toutes les conclusions à démontrer. Chacune d'elles sera mise à vrai si le fait correspondant est déduit par l'application d'une règle.
D'autres règles vont alors introduire les objets liés à l'énoncé et pouvant être utiles à la preuve. Une règle s'applique si sa partie condition est instanciée, et alors la partie action se déclenche et rajoute la plupart du temps un fait nouveau qui devient alors une nouvelle hypothèse utilisable. Il y a aussi mémorisation des faits et de la règle ayant permis son émergence ainsi que d'un modèle explicatif instancié de cette déduction. Ceci va permettre le filtrage de la base pour récupérer le graphe de la démonstration et en donner une explication.

2.4. Sortie du système

Lorsque toutes les conclusions partielles ont été prouvées, le système sort sur un succès et indique que l'exercice a été résolu. Si par contre le temps de recherche maximum, donné par l'utilisateur, a été dépassé ou si le moteur est arrivé au bout de la liste des règles actives sans qu'aucune ne puisse s'appliquer le système sort sur un échec et va chercher un autre énoncé pour recommencer.

2.5. Récupération du graphe de démonstration

Si le théorème a été démontré une procédure spéciale filtre la base de faits pour ne conserver que les déductions utiles à la démonstration du théorème et récupère les noms internes de tous les objets dont on aura besoin pour produire une rédaction de la démonstration compréhensible par un élève de lycée ou de collège. Le système sauvegarde aussi toutes les déductions faites et les explications correspondantes en vue d'une analyse de la résolution et une utilisation éventuelle en mode conjecture élève.

2.6. Production automatique de la rédaction en français de la démonstration

A partir du graphe de la démonstration, ARGOS va construire une preuve rédigée en français adaptée à l'utilisateur. La procédure de construction de l'explication va parcourir l'arbre de preuve, en profondeur et va descendre jusqu'aux hypothèses initiales. Chaque nœud ayant une explication rentrant dans le cadre des désirs de l'utilisateur sera expliqué, une fois et simplement rappelée si elle sert dans une autre branche de preuve.
Chaque nœud de l'arbre est développé ou non, grâce au modèle explicatif fourni par l'expert et instancié avec les bonnes variables, en fonction du type de preuve, détaillée ou non, et du niveau scolaire précisés par l'utilisateur. Le système peut y adjoindre, si cela a été prévu par l'expert ayant donné les connaissances, un commentaire explicitant, dans un langage proche de l'élève et de son niveau d'étude, pourquoi cette conjonction d'hypothèses a permis cette déduction.

3. Illustration par un exemple détaillé

L'exemple suivant est issu de [Pintado 94].

3.1. Enoncé

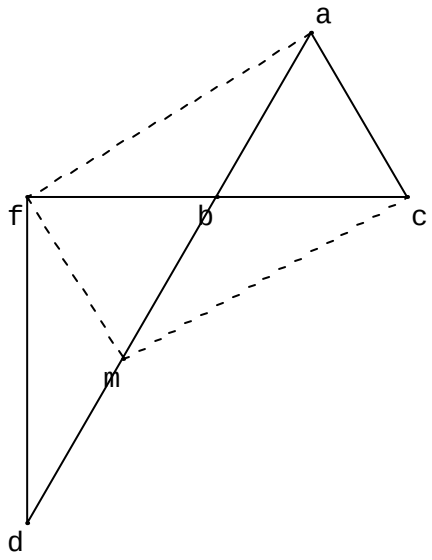
Enoncé livresque:

Soit abc un triangle équilatéral, m et d deux points tels que b soit le milieu du segment [am] et m le milieu du segment [bd]. Soit f le projeté orthogonal de d sur la droite (bc). Montrer que acmf est un rectangle.

Enoncé donné au système et utilisant la notation fonctionnelle:

equilateral(a, b, c) et milieu(a, m):b et milieu(b, d):m et projeteOrthog(d, droite(b, c)):f => rect(a, c, m, f).

figure :



3.2. Analyse de l'énoncé et élimination des symboles fonctionnels

Le système extrait les concepts : triangle équilatéral, milieu, projeté orthogonal, droite en hypothèse et rectangle en conclusion. Le système donne un nom, dr, à la droite (bc) et range le fait que dr est le nom de (bc) en mémoire pour un traitement ultérieur.

3.3. Sélection et chargement automatique des règles adéquates, dans un certain ordre.

Une typologie des règles, ayant été construite automatiquement, existe exprimant des liens entre concepts et règles construites. Argos a lui aussi tissé, au fil des exercices résolus, un réseau complémentaire de liens de ce type. Il connaît, par apprentissage, un certain nombre de règles ayant déjà servi à découvrir des rectangles et il les a rangées en fonction de leur fréquence d'utilisation pour constituer une stratégie. Il va alors construire la liste des règles actives en fonction de cette typologie à l'aide des concepts ayant été extraits de l'énoncé puis lancer la recherche. Le triangle équilatéral abc est introduit, les milieux et le projeté orthogonal f. Il faut aussi introduire le quadrilatère acmf et certains éléments susceptibles d'être utiles à la démonstration. Ceci est réalisé grâce à des règles traduisant l'expertise du domaine. Lorsque le mathématicien construit la figure il est déjà en train de faire des déductions et introduit donc les objets de façon dynamique. Le système n'a pas cette possibilité et doit donc introduire une liste minimale d'objets. L'introduction d'autres objets ne se fera que si Argos n'a plus rien à déduire, ceci par les règles de type création ou à l'occasion d'appels de conjectures. S'il existe une stratégie pour montrer que acmf est un rectangle et si le système est dans une situation déjà rencontrée, les règles correspondantes vont se déclencher et la démonstration va être accélérée. Si les règles appelant des conjectures ont été chargées et si le système repère des configurations incomplètes ou situation propice, susceptibles de faire avancer vers le but alors elles se déclencheront en créant éventuellement

de nouveaux objets. Dans l'exercice proposé, le système sait qu'il doit montrer que acmf est un parallélogramme et il sait aussi que b = mil(a, m) il pourra donc lancer, si l'utilisateur a sélectionné l'option correspondante, la conjecture pour montrer que b = mil(c, f). Ces appels de conjecture sont donc en général guidés par le but. Ceci est un moyen d'introduire dynamiquement des objets, un peu comme le fait le mathématicien quand il construit la figure.

3.4. Déductions par instanciations de règles

Que fait alors le système ? Puisque le triangle fbd est rectangle en f alors mf = mb. Le système déduit alors que a, b, m et d sont alignés dans cet ordre car on a une succession de milieux, que $\widehat{abc}$ est aigu puis que f, b et c sont alignés dans cet ordre. On a alors $\widehat{mbf} = 60°$ car $\widehat{abc}$ et $\widehat{mbf}$ sont des angles opposés par le sommet. Le système aura dû créer l'angle $\widehat{mbf}$. Donc mbf est un triangle équilatéral car il est isocèle avec un angle de 60 degrés. On en déduit donc que bf = bm et donc b = milieu(f, c). Ceci était le point important à prouver. acmf est alors un parallélogramme ayant ses diagonales de même longueur donc un rectangle.

4. Graphe et preuve rédigée d'un théorème

4.1. Obtention du graphe de démonstration

Tous les faits obtenus successivement par le démonstrateur sont classés par ordre chronologique et ont un numéro d'apparition. Pour chaque fait nouveau déduit on mémorise les hypothèses déjà présentes dans la base de faits qui ont permis son émergence. On peut donc remonter, à partir de chaque conclusion partielle obtenue, jusqu'aux hypothèses initiales ayant permis son émergence. L'arbre de démonstration est alors la réunion de tous les sous-arbres obtenus pour chaque élément de la conclusion globale. On récupère aussi les noms d'objets ayant servi dans la preuve, pour pouvoir, dans la rédaction, remplacer des noms barbares par la notation fonctionnelle du mathématicien. Si la droite (ab) a comme nom dr1 on pourra remplacer dans la trace rédigée dr1 par la notation plus intéressante (ab). On fait de même pour les longueurs, les angles, etc.

4.2. Obtention de la preuve rédigée

4.2.1. Preuve complète

A partir du graphe le système **construit** automatiquement une preuve rédigée en français respectant la chronologie des déductions. Voici la preuve obtenue en passant sous silence des faits semblant "évidents" sur la figure (c'est un choix de l'utilisateur de ne pas demander leur explication car Argos a dû lui les vérifier) et un extrait du graphe avec la définition des objets utilisés expliquant comment a été obtenu le chaînon d'explication en gras.

preuve:

(f d) orthog (b c) car f est le projeté orthogonal de d sur la droite (b c)

m f = m b car m est le milieu de [b d] et le triangle f b d est rectangle en f

On a a b = c b car le triangle a b c est équilatéral par hypothèse

On a angleGeom(a, b, c) = 60 car le triangle a b c est équilatéral par hypothèse

Les angles géométriques a b c et m b f sont opposés par le sommet donc ils ont même mesure

Puisque m b = m f et angleGeom(m, b, f) = 60 alors le triangle m b f est équilatéral

Puisque le triangle b m f est isocèle en b alors b m = b f

b est le milieu de [c f] car c , b et f sont alignés et b c = b f

a c m f est un rectangle car ses diagonales se coupent en leur milieu b donc c'est un parallélogramme et a m = c f car a b = b c (demi-diagonales de même longueur)

4.2.2. Exemple de développement d'un nœud

Développement d'un nœud:

Ceci est un extrait du graphe (le graphe complet est en annexe 2):

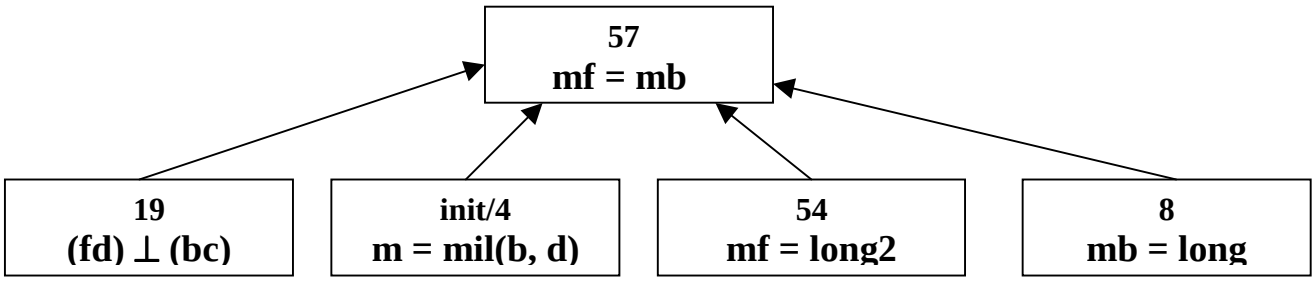
```
etiquette(57, long2=long).
fils(57, init/4).  fils(57, 19).  fils(57, 54).  fils(57, 8).
provientDe(57, triangleRectangleDeduction).
etiquette(init/4, milieu(b, d):m).
etiquette(19, dr2 orthog dr).                %(fd) ⊥ (bc)
etiquette(54, longueur(seg6):long2).         %mf = long2
etiquette(8, longueur(seg):long).            %mb = long
hyp(0, segment(m, b):seg, 7).  hyp(0, segment(m, f):seg6, 53).
hyp(0,  droite(b,  c):dr,  init/1).   hyp(0,  droite(f,  d):dr2,
16).
```

L'explication fournie sera alors:

mf = mb car les

droites (fd) et (bc) sont

perpendiculaires et m est le milieu du segment [bd] (en effet dans un triangle rectangle le milieu de l'hypoténuse est le centre du cercle circonscrit)



Un commentaire, lié au niveau de l'utilisateur, peut être associé à la règle et donne une explication complémentaire. Une même règle peut avoir plusieurs commentaires différents suivant le niveau de classe. On n'explique pas de la même façon une connaissance mathématique au collège et au lycée par exemple.
Il y a possibilité de demander au système une rédaction dans un certain esprit en précisant des domaines prioritaires. On sait bien qu'un même problème de géométrie peut être résolu par des outils très différents : géométrie des configurations, vectorielle, analytique, barycentrique ou autre. Certains types de problèmes, par exemple des problèmes d'alignement en seconde, peuvent être déjà re-traduits en utilisant la géométrie vectorielle ou analytique. L'utilisateur suggère au système, sans garantie de succès d'ailleurs, une preuve d'un certain type en lui demandant de charger en premier les règles d'un certain domaine. ARGOS va alors filtrer les stratégies pour ne garder que les règles des domaines choisis.

4.3. Problèmes rencontrés au niveau de l'explication

Il n'existe pas de rédaction universelle d'une preuve. Toute explication s'adresse à un public donné qui est plus ou moins compétent ou réceptif sur le sujet. Le problème central est donc de définir ce qui va être dit, comment on va le dire et ce qui ne sera que suggéré ou passé sous silence. Ceci relève d'une autre expertise que l'expertise mathématique. Il faut donc dissocier le module démonstration de théorème du module explication et rédaction de la preuve qui nécessite des connaissances d'un autre ordre. Il

faut bien se persuader que la tâche d'explication est une tâche de résolution de problème au même titre que la DAT. On peut imaginer qu'ARGOS produise une explication de la preuve d'un théorème en suivant la même démarche que celle lui permettant de trouver la démonstration. Ceci permettrait une plus grande souplesse au niveau de la mise à jour de la base de connaissances d'explication et une maintenance de celle-ci par un utilisateur n'ayant aucune connaissance du fonctionnement interne du système.

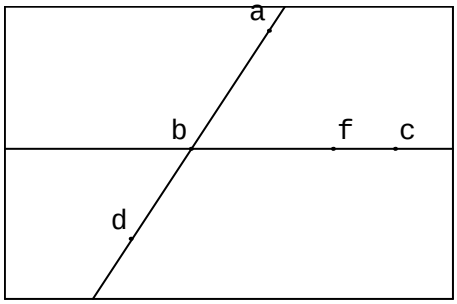
Pour l'instant le système s'appuie sur l'expert en ce qui concerne la construction de l'explication. Il possède cependant un dictionnaire lui permettant de traduire en français tous les prédicats internes déduits et il est capable d'aller rechercher la définition d'un nom interne pour une règle plus générale où ne serait pas spécifié la nature des objets utilisés.

4.3.1. Gestion des prédicats symétriques ou de dénomination fonctionnelle multiple

On accepte que dans la preuve le parallélogramme acmf apparaisse à certains endroits sous la dénomination acmf et à d'autres sous la forme facm par exemple. Ceci ne doit pas entraîner de surcharge cognitive trop forte pour le lecteur car celui-ci a la figure sous les yeux et voit instantanément qu'il s'agit du même objet. Une droite reçoit un nom interne à partir de deux points et ce nom sera systématiquement retranscris sous la forme droite(a, b) même si ce n'est pas vraiment avec ces deux points qu'on veut l'utiliser.

4.3.2. Gestion des implicites

Dans une rédaction type tout n'est pas justifié noir sur blanc. Il y a des implicites qui dépendent du contrat didactique passé entre le professeur et les élèves ou entre le correcteur et le candidat à un examen. Ceci dépend aussi du niveau d'étude. Une rédaction de solution pour un exercice de géométrie donné ne sera pas la même en troisième, en seconde ou en terminale. Il y a des choses que l'on passera sous silence pour éviter des lourdeurs de rédaction ou parce qu'on pense qu'à un certain niveau le lecteur est capable de faire la déduction manquante par lui-même. Il y a un bel exemple de ce genre de problème dans l'exercice décrit plus haut. La règle rajoutant que $\widehat{fbm} = \widehat{abc}$ car ce sont des angles opposés par le sommet est obligée de vérifier que f, b et c sont alignés dans cet ordre car sinon elle ferait déduire des arguments faux au démonstrateur dans d'autres circonstances. Avec une figure comme ci-dessous, en oubliant la condition précédente, la règle rajouterait que : $\widehat{abc} = \widehat{fbd}$.



Pourtant quand on donne cet exercice à un expert, celui-ci n'évoque jamais ce problème. Il le lit sur la figure et considère cette propriété comme allant de soi. Mais si on lui demande de le prouver ça n'est pas immédiat. Pour le démonstrateur le contrat didactique est simple, rien ne doit être passé sous silence. Par contre au moment de l'argumentation ceci sera possible et même souvent souhaitable d'un point de vue pédagogique.

4.3.3. Construction de la preuve rédigée

Il s'agit ici de résoudre un problème, celui d'expliquer à un utilisateur comment l'exercice a été résolu. A partir des hypothèses de départ, ARGOS a pu déduire, étape par étape, ce qu'il fallait démontrer. Comment justifier chaque chaînon intermédiaire ? La procédure de base est celle de l'explication d'un fait déduit intervenant dans la démonstration.
L'expertise d'explication doit donner, en fonction du profil de l'utilisateur, un certain nombre de conseils.

- Doit-on expliquer le fait ou non ?
- Quel commentaire doit accompagner la donnée des faits ayant permis l' émergence de celui-ci?
- Comment doit-on enchaîner les différentes étapes de la preuve ?
- Comment articuler les différents chaînons du raisonnement, reconnaître des pas de déductions du même type (c'est le « de même » du mathématicien), et de façon plus générale éviter les redondances qui lassent le lecteur ?
- Comment structurer la rédaction en faisant ressortir les éléments essentiels ?

4.3.4. Conclusion

Le but ultime est de produire des traces rédigées qui retranscrivent fidèlement l'arbre de démonstration mais avec la plus grande économie de moyens possible et en fonction du public auxquelles elles s'adressent. Ceci nécessite des connaissances spécifiques que le système doit utiliser pour construire automatiquement son explication.

5. Expressions des connaissances

Le principe est d'exprimer les connaissances de façon déclarative dans une syntaxe proche du langage mathématique. Le choix actuel pour donner les connaissances au système est le suivant. L'utilisateur va structurer les connaissances comme il l'entend et indiquer la façon dont lui-même expliquerait les déductions correspondantes obtenues par application de chaque propriété exprimée.

5.1. Connaissances du domaine et leur explicitation

Les connaissances fournies au système sont de deux ordres:
les propriétés mathématiques classiques exprimées à différents niveaux de généralité.
des heuristiques de créations d'objets conditionnées ou non par le but.

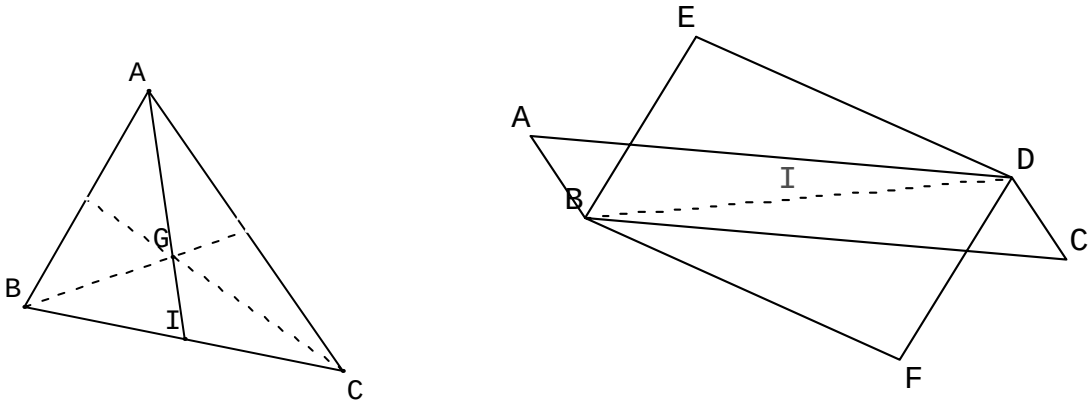
Voici un exemple de propriété:

"Si G est le centre de gravité du triangle ABC alors il est situé sur chaque médiane**.**"
exprimée dans la syntaxe d'ARGOS par:

```
(centreGravite(A,B,C):G et milieu(B,C):I =>
G app droite(A,I) )
```

Voici un exemple d'heuristique de création:
" Si ABCD et EBFD sont deux parallélogrammes distincts alors créer le milieu de leur diagonale commune. " traduite par le prédicat:

```
heuristique(plgAyantDiagCommune, plg(A,B,C,D) et plg(E,B,F,D)
et distincts(A,E) => creer(milieu(B,D):_)).
```



Les connaissances sont données avec leur modèle explicatif sous forme de prédicats dont voici quelques exemples:

Exemple1:

propriete([general],[obligatoire],centreGravite,

si G est le centre de gravité du triangle ABC et I = mil(B,C) alors G ∈ (AI),

[G est le centre de gravité du triangle ABC donc il appartient à la médiane issue de A, c'est à dire la droite (AI)]

).

Savoir-faire associé.

savoirFaire(centreGravite, [Pour montrer qu'un point G est sur une médiane du triangle ABC montrer que c'est le centre de gravité]

).

Exemple2:

propriete([general], [obligatoire], reciproquePythagore,

si $BC^2 = AC^2 + AB^2$ alors le triangle ABC est rectangle en A,

[On a $BC^2 = AC^2 + AB^2$ donc le triangle ABC est rectangle en A] **).**

savoir-faire associé:

savoirFaire(reciproquePythagore ,[Pour montrer qu'un triangle ABC est rectangle en A montrer que: $BC^2 = AC^2 + AB^2$]**).**

Exemple3:

propriete([calculVectoriel],[obligatoire, seconde], vectDroiteMilieux,

si I = mil(A,B) et J = mil(A,C) alors vect.IJ = ½ vect.BC , [Puisque I = mil(A,B) et J = mil(A,C) alors vect.IJ = ½ vect.BC] **).**

Exemple4:

propriete([general],[fin],anglesCorrespondants,

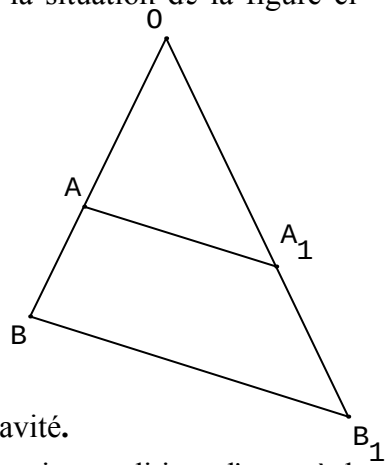
si (AB) // (CD) et D ∈ [O,B) et C ∈ [O,A) alors les angles $\widehat{OAB}$ et $\widehat{OCD}$ sont correspondants, les angles $\widehat{OAB}$ et $\widehat{OCD}$ sont correspondants car (AB) // (CD) et D ∈ [O,B) et C ∈ [O,A). En effet, ainsi, (B,A) et (D,C) sont de même sens. **).**

5.2. Mécanismes et heuristiques de vérification

Chaque type d'objet défini comme concept génère, s'il n'en existe pas une procédure de recherche de la présence d'un tel objet dans la base de faits. C'est à ce niveau que sont gérées les propriétés de symétrie. Des heuristiques de recherche peuvent être rajoutées, par l'expert, qui vérifieront certaines conditions particulières de la présence du concept.

(1) Par exemple le but **plg(N,A,B,C,D,Q),** les majuscules représentant des variables, recherchera la présence d'un parallélogramme ABCD avec les variables partiellement, complètement ou pas du tout instanciées. Le but est atteint si on a en hypothèse l'une des huit façons de traduire que ABCD est un parallélogramme.

(2) **memeSens(N,A,A1,B,B1,Q)** vérifie si les vecteurs AA1 et BB1 sont de même sens dans certaines conditions concrètes comme par exemple la situation de la figure ci-dessous.



(3) **centreGravite(N,A,B,C,G,Q)** recherchera un centre de gravité**.**

Voici un exemple de recherche d'instanciation de la partie condition d'une règle rajoutant qu'un centre de gravité est sur chaque médiane:

Règle utilisée:

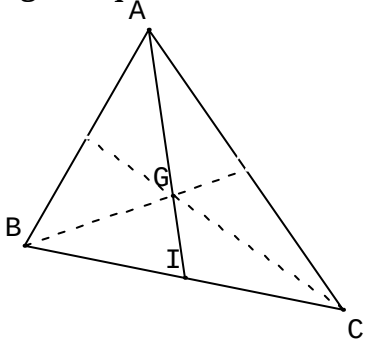
En français: Si I = mil(B,C), G est le centre de gravité du triangle ABC, J est la droite (AI) et si on ne sait pas que I est sur la droite (AI) alors rajouter l'hypothèse qu'il y appartient.

Dans la syntaxe du système (règle générée automatiquement):

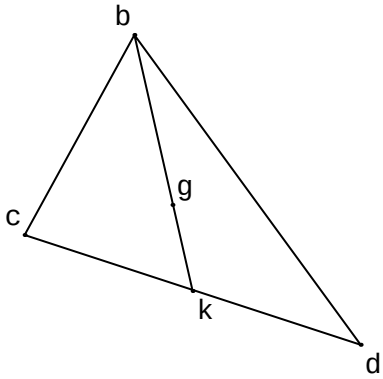
```
regle(A, centreGraviteDeduction, B) :-
        milieu(N,B,C,I, F),              %I = mil(B,C)
        centreGravite(N,A,B,C,G, Q),%G centre de gravité de ABC
        droite(N, J, A,I, K),            %J = (AI)
        not hyp(N, G app J, L), %on ne sait pas que G ∈ (AI)
        ajhyp(N, G app J, M),            %rajouter que G ∈ (AI)
        memoriser([centreGraviteDeduction, F, Q, K], M),
        memoriserExpl([obligatoire], [H, 'est le centre de
gravité du triangle', G, C, D, 'donc il appartient à la
médiane issue de', G, 'cad la droite (', G, E, ')'], M).
```

A partir de la situation générique du corps de la règle ci-dessus, le système va chercher à instancier la partie condition de la règle dans la base de faits.

situation générique:



situation concrète:



En supposant que le système sache déjà que k est le milieu de [cd], le premier prédicat de la règle, I = mil(B,C), sera instancié par k = mil(c, d). Il y a alors appel du sous-but suivant: centreGravite(0,A,c,d,G,Q) partiellement instancié. Si en hypothèse le fait que g est centre de gravité du triangle bcd figure sous la forme: hyp(0,centreGrav(c, b, d):g,23) alors la procédure de recherche va chercher à vérifier l'une des 3! = 6 façons d'exprimer ce fait. hyp(0, centreGrav(A, c, d):G,Q) échouera ainsi que hyp(0, centreGrav(A, d, c):G,Q), etc jusqu'à hyp(0, centreGrav(c, A, d):G,Q) qui donnera une réponse positive avec le jeu d'instanciations: A = b, G = g et Q=23.

5.3. Procédures de calcul

Tout objet mesurable (longueur, aire, angle géométrique, etc) a, parmi ses attributs, une valeur exacte donnée par hypothèse, déduite ou inconnue. Les relations numériques liant ces différentes grandeurs s'expriment sous forme d'hypothèses ayant été déduites ou comme données initiales. Une règle peut appeler des procédures de calculs si l'une des grandeurs non encore affectée est calculable à partir d'une de ces relations.

Exemple:
La règle construite, à partir de la propriété ci-dessous, traduisant la relation de Pythagore rajoutera si elle s'applique une relation numérique:
propriete([general], [obligatoire], pythagore,
si le triangle ABC est rectangle en A alors $BC^2 = AC^2 + AB^2$,
[On a $BC^2 = AC^2 + AB^2$ car le triangle ABC est rectangle en A]
).

La règle issue de ci-dessous, exprimée ici à un niveau de généralité supérieur car on ne précise pas la nature des objets, qui sont tout de même des grandeurs mesurables, calcule alors L1 car L et L2 ont une valeur exacte connue:
propriete([general], [obligatoire], valeurExacte,
si $L^2 = L2^2 + L1^2$ et L mesure Le et L2 mesure Le2 et le résultat du calcul rac(Le^2 - $Le2^2$) est Res alors L1 mesure Res,
[L1 = Res car $L^2 = L2^2 + L1^2$ et L = Le et L2 = Le2 et donc L1 = rac(Le^2 - $Le2^2$)]
).
Puisque la relation $L^2 = L2^2 + L1^2$ figure en hypothèse et que deux grandeurs sont instanciées alors la règle appèle une procédure de calcul qui va renvoyer dans la variable Res la valeur exacte du calcul: rac(Le^2 - $Le2^2$). L'explication d'un tel pas de raisonnement va nécessiter de la part d'ARGOS une recherche des noms internes des objets longueurs pour permettre la construction d'une phrase pouvant être par exemple, celle-ci dépendant des choix explicatifs

de l'expert:
ab = 2*rac(3) car bc^2 = ab^2 + ac^2 et bc = 3*rac(2) et ac = rac(6) et donc ab = rac((3*rac(2))^2 - (rac(6))^2)

6. Construction automatique des règles

6.1. Principe de construction

Il s'agit de faire construire, par le système, de façon automatique, des règles de différents types à partir de connaissances données de façon déclarative par un utilisateur. ARGOS va alors pouvoir construire automatiquement, à partir d'une même connaissance mathématique, des règles de type déduction démarche en avant, déduction en fonction du but, création d'objet manquant, et des règles de condition suffisante de la conclusion.

6.2. Démarche

6.2.1. Donner un nom à la règle

Il faut donner un nom à la règle en fonction du type voulu, des concepts présents et de ce qu'elle va faire (ceci pour améliorer le confort de l'utilisateur expert qui veut relire sa base de règles). A partir du concept centre de gravité, par exemple, seront fabriquées automatiquement des règles qui déduiront des propriétés à partir d'un centre de gravité existant et d'autres qui, si les conditions sont réunies dans la base de faits, rajouteront que tel point est un centre de gravité. Des règles de création d'objets manquants seront aussi construites ainsi que des règles de condition suffisante de la conclusion sortes de raccourcis guidés par le but.

6.2.2. Constituer le corps de la règle.

- Il faut que la règle vérifie la présence des objets nécessaires.
- Il faut constituer la partie condition nécessaire à l'application de la règle.
- Il faut introduire un test de non répétition de la règle, puis constituer la partie action et mémorisation pour le graphe de la preuve ainsi que son explication.

6.2.3. Optimisation de l'ordre des sous-buts et mise au point finale.

Pour qu'une règle puisse s'appliquer de façon optimale, l'ordre des sous-buts est très important. Il faut par exemple placer les configurations rares ou spécifiques en premier pour instancier le plus tôt possible la règle avec les bonnes variables. D'autre part il faut qu'une variable soit instanciée avant que l'on puisse l'utiliser. Il faut aussi maîtriser le retour arrière. Il a donc fallu définir une hiérarchie dans la recherche des conditions d'application: vérification prioritaire, configuration, vérification de base, appel de procédures de calcul. Chaque règle construite est aussi associée à un ou plusieurs domaines, elle peut être éventuellement associée à un savoir-faire et elle va être typée en fonction des concepts présents dans son corps en partie hypothèse ou conclusion ceci en vue du chargement de règles adaptées à l'exercice donné.

6.3. Différents types de règles:

Le système construit des règles :

-déduction tout venant: très proche de la propriété mathématique, tous les objets nécessaires doivent être présents dans la base de faits. Voici la démarche de construction :

*récupération des objets nommables au sein de la propriété

*découpage de l'énoncé en cas de conclusion conjonctive

*construction des prédicats de vérification des hypothèses, et des objets nommés

*vérification de non présence du fait à déduire

*ajout de la partie action et mémorisation (nom de la règle, hypothèses dont provient le nouveau fait déduit et modèle explicatif)

-Règles de type création où certains des objets nécessaires sont absents:

*définir les objets que l'on pourra créer en cours de preuve (concepts objets "créables").

*constituer la même partie condition que dans les règles de déduction sauf pour la vérification des objets nécessaires.

*la règle vérifiera la présence de tous les objets sauf un pour lequel la non présence donnera comme partie action sa création.

*mémoriser la règle qui a amené cette création et les conditions de la création.

-Règles de condition suffisante:

***pour chaque concept déductible par une démonstration on balaie l'ensemble des propriétés et pour chaque propriété où figure une conclusion déductible on fabrique une règle de condition suffisante.

6.4. Exemples de règles construites :

L'exemple précédent donnera les règles suivantes écrites en français pour leur lisibilité (voir annexe4):

Déduction : si G est centre de gravité de ABC avec I = mil(B,C) et que le système ne sait pas que G est sur la droite (AI) alors rajouter ce fait et mémoriser la raison de son apparition.
Condition suffisante : si on veut montrer que G ∈ (AI) avec I = mil(B,C) et que G est centre de gravité du triangle ABC alors mettre le sous-but correspondant à vrai et mémoriser pourquoi.
Création : si G est centre de gravité de ABC avec I = mil(B,C) et que la droite (AI) n'existe pas pour le système alors la créer et mémoriser les conditions de création.

7. Appels de conjectures en cours de preuve

Le système a la possibilité d'appeler des conjectures, guidées par le but en cas de configurations incomplètes permettant d 'avancer vers une des conclusions partielles à prouver. Un exemple de règle appelant une conjecture se trouve dans la section 7.3.

7.1. Que se passe t'il en cas d'appel ?

De nouveaux concepts sont introduits et des objets créés en fonction de ce que demande la conjecture. Il y a chargement de nouvelles règles et ré-ordonnancement car le système reconstruit une nouvelle liste de règles actives**.** Pour l'instant tout fait déduit au sein de la conjecture ainsi que les objets créés sont conservés par la suite. Si la conclusion finale est prouvée en milieu de conjecture alors le système sort et la démonstration est terminée. Une conjecture prouvée apparaît comme un nouveau fait déduit et on a mémorisé le sous-arbre ayant permis son émergence. Si il y a échec lors d'un appel, on reprend le fonctionnement global avec éventuellement appels d'autres conjectures. Si Argos a eu besoin de faire appel à une conjecture pour démontrer, il l'indique dans la rédaction de la preuve comme une sorte de question intermédiaire, un lemme, qu'il a dû montrer avant de s'attaquer à l'exercice proprement dit. (voir Annexe5)

7.2. Problèmes

Un appel de conjecture est une démarche typiquement heuristique. Cet appel doit être fait au bon moment. Trop tôt et c'est l'échec, trop tard ça ne sert plus à rien. Il ne faut pas que le démonstrateur s'égare. Il apparaît utile de supprimer les règles de création d'objet lors d'un appel de conjectures et de n'accepter que des règles de création ciblées en fonction de la conclusion. Un mathématicien ne fait pas un appel de conjecture par hasard. Dès le codage de la figure il en émet. Il fonctionne par exemple par analogie. Il a déjà rencontré une situation similaire et il essaye de voir si, sur le problème du moment, cela va fonctionner de la même façon. C'est en fonction de son expérience qu'il fera tel ou tel choix. Comment acquérir la **« certitude »** de celui qui **« voit »** sur la figure et à qui il ne manque que le cheminement déductif?

7.3. Exemple de règle appelant une conjecture :

Exemple en français: Si on veut montrer que ABCD est un parallélogramme, rectangle, carré ou losange, si le milieu d'une des diagonales existe déjà et qu'un appel de conjecture pour montrer que c'est le milieu de l'autre n'a pas déjà été lancé alors essayer de montrer que c'est aussi le milieu de l'autre diagonale.

Traduction interne :

```
regle(N, conjecture,_) :- appelConjecture(non),
     (eltConcl(N, plg(A,B,C,D))
     ;eltConcl(N, rect(A,B,C,D))
     ;eltConcl(N, losange(A,B,C,D))
     ;eltConcl(N, carre(A,B,C,D))
     ),
            (milieu(N,A,C,I,_),
             not milieu(N,B,D,I,_),
                    not conj_traite(N, milieu(B,D):I),
                    assert(conj_traite(N, milieu(B,D):I)),
             X = B, Y = D
             ; milieu(N,B,D,I,_),
             not milieu(N,A,C,I,_),
                    not conj_traite(N, milieu(A,C):I),
                     assert(conj_traite(N, milieu(A,C):I)),
             X = A, Y = C
            ),
%écrire ce message sur le brouillon de recherche
ecrire1([comme,I,'est milieu d''une des diagonales montrons
```

```
que’  ,I,=,mil,'(',X,Y,')  pour  montrer  que’,A,B,C,D,’est  un
parallélogramme’]),
conj1(N,milieu(X,Y):I,_).
```

On peut envisager la possibilité de construction automatique d’une telle règle à partir d’énoncés déclaratifs du type ci-dessus. Un utilisateur mathématicien pourra ainsi exprimer des heuristiques qui pourront être transformées en règles, automatiquement, par le système.

8. Apprentissage

8.1. En quoi le système devient-il différent après avoir résolu un exercice?

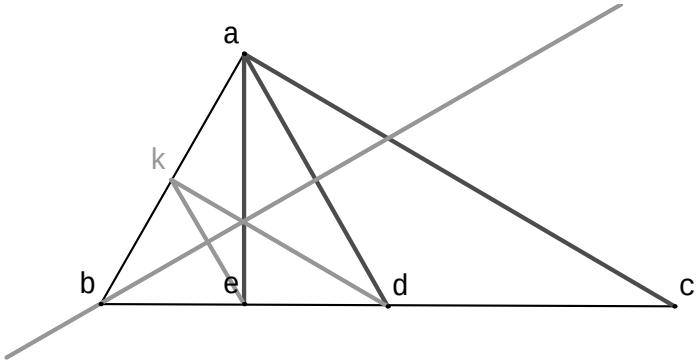
Après avoir résolu un exercice, ARGOS accomplit un certain nombre de tâches automatiquement.
Il constitue tout d’abord une base de données contenant les règles qui se sont déclenchées, celles qui ont été vraiment utiles à la preuve, les concepts ayant été introduits et utilisés, les concepts figurant dans la partie conclusion, les objets ayant été nécessaires et ayant été créés en cours de preuve, la raison pour laquelle on les a créés et ce à quoi ils ont servi, puis la durée de la démonstration.
Il va alors mettre à jour ses connaissances stratégiques. Il rajoute des liens entre les règles utiles et les concepts de la conclusion sauf s’ils existaient déjà. Il mémorise donc que telle règle est une règle utile pour montrer tel élément de conclusion. Chaque règle possède un poids, mesurant son importance par rapport à un concept conclusif donné. Ce poids est lui aussi mis à jour pour chaque règle ayant été utile.
Ceci va alors permettre une mise à jour des stratégies existantes pour démontrer tel ou tel type de conclusion. Par exemple si telle règle a été utile pour montrer qu’un triangle était équilatéral son poids est augmenté de 1. Ceci pourra entraîner un changement de l’ordre des règles dans la liste correspondant à la stratégie pour montrer qu’un triangle est équilatéral.

8.2. Discussion et problèmes.

Le but de cet apprentissage est bien sûr de modéliser les mécanismes d’apprentissage de l’humain. Un travail énorme reste à faire.
Comment généraliser une technique à partir d’un ou plusieurs exercices résolus de la même façon ? Comment le système pourrait-il s’apercevoir que plusieurs solutions sont du même type ?

Comment découvrir des « schémas » d'utilisation d'une propriété qui pourraient déboucher sur la construction automatique de nouvelles règles à partir d'une connaissance donnée ?
Comment l'être humain classifie t-il et réorganise t-il ses connaissances après la recherche d'un nouvel exercice ?
Il faut donc fournir au système des méta-connaissances de contrôle au sens de J. Pitrat [Pitrat 90] qui lui permettent de " monitorer " ses actions d'apprentissage [Pitrat 99]. Il faut que le système « soit capable de juger de la qualité de ce qu'il apprend » [Pintado 94].
Expérimentalement on constate une amélioration du démonstrateur après apprentissage quand il n'a pas eu besoin de créer des objets nouveaux en cours de preuve. Dans le cas contraire comme les règles de création sont en bout de liste de règles actives on ne note aucune amélioration après apprentissage. Il manque au système des stratégies de création d'objets bien ciblées en fonction du but.
Voici un exemple d'exercice posant problème au niveau de l'apprentissage car nécessitant beaucoup de créations d'objets non immédiates.

Exemple:
Soit abd un triangle, e le milieu de [bd] et d le milieu de [bc] et ab = dc. Montrer que les angles géométriques $\widehat{ead}$ et $\widehat{dac}$ sont égaux.
Figure:



Pour montrer que $\widehat{ead}$ = $\widehat{dac}$ ARGOS va devoir créer des objets qui n'apparaissent pas de prime abord même au yeux de l'expert devant la figure à l'état brut. Il doit créer k le milieu de [a, b], l'angle $\widehat{adk}$, les droites(kd) et (ek) et en plus il devrait penser lui-même à introduire la réflexion d'axe la médiatrice de [ad]. Quelles heuristiques de création d'objet va t-il pouvoir apprendre qui pourront lui être utile dans d'autres situations semblables, sans pour cela générer d'explosion combinatoire dans tous les autres cas ? C'est un problème ouvert qui n'a pas de solution satisfaisante pour le moment. On peut seulement lui donner par exemple l'heuristique suivante :
Si un triangle est isocèle, si le milieu d'un côté autre que la base existe, alors créer son symétrique par rapport à l'axe de symétrie. Ceci pourra servir souvent. Mais on ne peut pas

créer de façon systématique tout axe de symétrie d'une figure donnée. Pourtant l'expert a toujours ces considérations présentes et prêtes à servir.

9. Utilisation d'ARGOS en EIAO

9.1. Problématique.

En quoi le système peut-il aider un élève à progresser dans l'art de chercher et résoudre un exercice de géométrie? Il va de soi que fournir une preuve rédigée, même obtenue automatiquement, d'un exercice donné n'a que peu d'intérêt d'un point de vue pédagogique. Il faut donner la possibilité à l'élève de construire lui-même sa solution et d'en construire une rédaction. On sait d'ailleurs que la principale difficulté des élèves est de bien organiser leur pensée même si en géométrie se surajoute aussi le fait d'avoir la "bonne idée" qui fera avancer vers la solution.

9.2. Que peut faire l'élève? (fonctionnalités d'Argos)

L'élève commence par donner l'énoncé au système. Il a ainsi un travail de lecture et d'analyse de l'énoncé livresque, pour en extraire les hypothèses et ce qu'il faut démontrer, puis lance la recherche de solution(s) par ARGOS. Pendant le temps de la recherche d'une solution, l'utilisateur pourra construire la figure avec un logiciel de construction géométrique (Geoplan ou Cabri par exemple) ou à la main. Ceci lui permettra de vérifier expérimentalement ses intuitions.
Après examen de la figure l'élève peut, en toute liberté, émettre des conjectures, en introduisant ou pas des objets nouveaux, qu'il peut soumettre à Argos. Le système va alors vérifier si la conjecture émise est déjà présente dans la base comme un fait déduit. Sinon il va lancer une recherche avec comme élément de conclusion la question de l'élève, plus d'ailleurs la conclusion de l'exercice si celle-ci n'a pas été obtenue par la seule compétence du système. Ceci peut permettre une sorte "d' échange" entre la machine et l'utilisateur. L'élève peut ensuite, si la conjecture est prouvée, en obtenir une explication complète à partir des hypothèses initiales ou bien seulement au premier niveau pour lui donner des idées mais l'obliger quand même à trouver et rédiger de lui-même. Si par exemple, dans un exercice, l'élève a demandé pourquoi le point i est milieu de [ab], Argos pourra par exemple lui répondre que c'est parce que ajbc est un parallélogramme et que i est à l'intersection des diagonales ou toute autre indication du même type. Le système est paramétrable et pourrait l'être par exemple par un professeur qui voudrait faire travailler ses élèves dans cet esprit sans

que l'élève puisse avoir une réponse complète mais seulement partielle. Cette dynamique, où l'on cherche à rendre actif l'élève, devrait permettre d'aiguiser sa démarche heuristique, alors que celui-ci abandonne souvent ses idées en cours de route par manque de confiance en soi.
Pour un type de conclusion donnée, l'élève peut demander au système une liste de savoir-faire permettant de la déduire. Voici un exemple de genre de savoir-faire qu'il pourrait obtenir en demandant comment faire pour montrer qu'un triangle ABC est rectangle en A:
Pour montrer qu'un triangle ABC est rectangle en A
-montrer que: BC^2 =AC^2+BA^2
-ou montrer que (AB) $\perp$ (AC)
-ou montrer que $\widehat{ABC} + \widehat{ACB} = 90$ etc
L'utilisateur peut demander une preuve dans un certain esprit (analytique, configurations, vectoriel, transformations, etc) en faisant charger d'abord les règles du domaine correspondant comme expliqué dans **1.1**.
Il peut demander une preuve reposant sur un certain savoir-faire en ne permettant à Argos de ne conclure qu'avec la règle correspondante. Le système recherchera toutes les règles concluant sur le type de but voulu et ne chargera que la règle associée au savoir-faire voulu. Ceci va évidemment diminuer les chances de succès pour le système.

9.3. Expérimentation

Cette façon de travailler a été testée, sur le mode leçon particulière, avec la restriction qu'il n'existe pas encore d'interface permettant à l'élève de rentrer ses données et questions, sans avoir à respecter une certaine syntaxe, par exemple par le biais de boîtes de dialogue où il ne reste plus qu'à instancier les variables. Ceci alourdit la démarche de l'apprenant et suppose une motivation forte. La présence du professeur est indispensable pour guider l'élève et l'aider à formuler les hypothèses de l'exercice et ce qu'il y a à démontrer. On ne peut imaginer, pour l'instant, une utilisation en autonomie qu'avec une grande pratique de la part de l'utilisateur.

9.4. Transformer l'élève en Expert!

Il s'agit bien sûr d'un souhait légitime mais de l'ordre de la prospective. Il est connu que l'on ne maîtrise vraiment bien un sujet que quand on a à l'enseigner ou à l'expliquer aux autres. On est alors obligé de synthétiser, d'aller à l'essentiel, de trier la connaissance et surtout d'être actif et **à l'initiative de** ce qui va se faire. On peut imaginer qu'à partir de la classe de quatrième l'élève lui-même se construise une liste de connaissances qui vont s'enrichir au fil des ans, se modifier et s'enrichir. Il pourra écrire des règles mathématiques, même fausses, sans être jugé ou sanctionné, qu'il pourra voir à l'œuvre sur des exercices concrets. Il s'apercevra alors des insuffisances ou contradictions et pourra avoir envie d'y remédier. Ainsi

il pourra aussi améliorer sa façon d'expliquer en examinant les preuves renvoyées par Argos à partir de ses modèles explicatifs. Argos devient l'apprenant et ceci fait jouer un rôle actif à l'élève qui est ainsi valorisé.

9.5. Insuffisances.

L'élève devrait pouvoir proposer lui-même une solution rédigée de l'exercice, être aidé dans la construction de la preuve et la faire analyser par Argos (renvoi d'un contre-exemple pour une règle mal utilisée, indication des éléments clés, indiquer l'impossibilité d'une déduction, etc). Argos devrait pouvoir dégager l'idée directrice de la démonstration, les éléments clé pour pouvoir guider l'élève.
Il faudrait pouvoir, en fonction des hypothèses présentes, demander à Argos de donner toutes les instanciations correspondantes à un certain type de configurations. (ceci revient en fait à extraire de la figure des sous-figures correspondantes à des configurations classiques)

10. Problèmes rencontrés en Démonstration Automatique de Théorèmes

10.1. Quels objets introduire au départ ?

Il faut effectuer un savant dosage à partir de chaque configuration en hypothèse pour ne pas introduire trop d'objets mais quand même ceux qu'il faut. Par exemple, si un triangle équilatéral est en hypothèse, que doit-on créer ? Après discussion auprès d'experts du domaine on indique que les trois côtés et les trois angles au sommet ont même mesure. L'appel de conjectures en fonction du but est un moyen d'introduire de bons objets en cours de démonstration. Mais ceci doit reposer sur des heuristiques de création données par l'expert ou découvertes par le système.

10.2. Comment charger les bonnes règles ?

Il faut implémenter la démarche de l'expert c'est-à-dire l'extraction des configurations prégnantes qui sautent aux yeux de l'expert. Il faut constituer un étiquetage de la figure au sens de JM Bazin[Bazin 93]. Il faut mieux cerner l'implication d'une règle en fonction de la conjonction de concepts présents. En quoi par exemple la présence des deux concepts équilatéral et orthogonalité va t-elle influer sur le chargement des règles ? La typologie de règles devrait prendre en compte la conjonction de plusieurs concepts. L'ensemble des règles pourrait être structuré selon un modèle hiérarchique. Mais peut-on réellement fabriquer un tel

modèle qui implémenterait vraiment la démarche de l'expert ?

10.3. Ordonner les règles dynamiquement ?

Comment l'expert sent-il qu'il est sur une mauvaise piste ? Ce n'est pas qu'une question de temps passé trop long dans une certaine direction qui oriente l'expert dans une autre direction. Comment lui vient-il une nouvelle idée ? La possibilité d'ordonner les règles dynamiquement va influer sur l'efficacité du démonstrateur. Certaines règles importantes pour la démonstration, placées loin, sont obligées d'attendre que toutes les règles précédentes ne puissent plus s'appliquer pour entrer en action. Un moyen de régler ce problème est la possibilité d'appeler des conjectures avec création d'objet à partir de règles vérifiant la présence de conditions particulières favorables en fonction du but (Dans l'exemple détaillé cela serait d'appeler la conjecture b = mil(f, c) (voir section 3.1.)).

10.4. Création d'objets en cours de démonstration

Il s'agit là du principal problème en DAT. La création d'objets pour le mathématicien n'a pas le même coût que pour le système. Pour ARGOS les règles de création sont placées à la fin et sont des jokers si le démonstrateur ne déduit plus rien (problèmes d'explosion combinatoire). Comment, après avoir démontré un théorème ayant nécessité des créations d'objets, le système peut-il apprendre des heuristiques de création qui lui feraient gagner du temps lors d'autres exercices du même type ?

10.5. Repérer les éléments clés d'une démonstration

Dans l'exemple, le verrou à faire tomber est de prouver que b = mil(f, c). Comment un système informatique pourrait-il repérer ces nœuds de difficultés pour se focaliser sur eux et faire de bons appels de conjecture ? Ceci pourrait aussi aider pour la preuve rédigée, et ne pas placer tous les nœuds de développement sur le même plan.

10.6. Interactions homme / machine

Pour une utilisation en EIAO il faut aussi envisager une interface conviviale avec construction de la figure, qui permette au professeur de donner les connaissances au système sans efforts particuliers, et à l'élève de poser ses problèmes, confirmer ses idées, explorer les différentes déductions possibles à partir de l'énoncé donné et avoir une explication sur tout fait déductible. Il faudrait, en particulier, que, à partir d'un énoncé livresque, le système aide l'élève à extraire les hypothèses données, et lui permette d'écrire l'énoncé sans avoir un

certain langage d'expression des connaissances à apprendre. Un système de menus déroulants, avec boîtes de dialogue, et messages d'erreurs ou de confirmation devra lui permettre d'introduire les objets en n'ayant à sa charge que l'instanciation concrète de ceux-ci.

11. Bilan partiel et perspectives

11.1. Bilan

Après une première version où la base de règles avait été écrite à la main, testée et mise au point sur des exercices niveau lycée, la deuxième version avec construction automatique de règles re-démontre progressivement les mêmes théorèmes en fonction des connaissances fournies au système. Le démonstrateur démontre environ 250 théorèmes, en géométrie des configurations, niveau lycée (parallélogrammes, triangles, milieux, cocyclicité, angles géométriques ou orientés, Thalès, Pythagore etc) ,en calcul vectoriel, géométrie analytique et transformations planes. Les exemples proviennent pour la plupart de livres de secondes [Terracher Hachette Secondes] par exemple, ou ont été proposés, pour certains, par des collègues de mathématique. Les temps de démonstrations sont inégaux suivant les types d'exercices et bien entendu suivant la façon dont sont ordonnées les règles actives. Pour fixer les idées, la plupart sont obtenues en moins d'une minute sur un PC 200 Mz. Le nombre de règles construites est actuellement d'environ 1200 et augmente en fonction des nouvelles connaissances fournies par l'utilisateur. Le démonstrateur démontre tous les théorèmes figurant dans les thèses de Pintado [Pintado 94] et Bazin [Bazin 93], mais en prenant plus de temps en général pour ceux de [Pintado 94] (la base de connaissances est peut-être plus grande). Les temps de résolutions sont améliorés sensiblement par le chargement de stratégies apprises, guidées par le but, mais il manque l'apprentissage automatique d'heuristiques de créations ce qui bloque souvent le démonstrateur dans ses recherches.

11.2. Objectifs

Il s'agit de rendre le démonstrateur suffisamment robuste pour qu'il puisse servir en EIAO. Il apparaît nécessaire de créer une interface utilisateur sur poste fixe ou par internet pour pouvoir vraiment tester cette robustesse et vérifier l'intérêt pédagogique que peut apporter un tel système. Il est possible de guider, pas à pas, la recherche d'un élève en partant du but à partir d'une preuve découverte par ARGOS, car à partir de la conclusion on peut développer la preuve niveau par niveau. On peut imaginer la possibilité de faire construire la figure

automatiquement par un couplage avec des logiciels comme GEOPLANW ou CABRI-GEOMETRE par exemple. Il faut implémenter des méthodes d'apprentissage correspondant au savoir-faire de l'expert qui reconnaît dans l'expérience présente des situations antérieures déjà rencontrées. Dans une situation analogue on mettra en œuvre des techniques semblables. Ceci devrait permettre une meilleure sélection des règles et des liens entre concepts et règles applicables. Un travail reste à faire au niveau de l'explication. Il apparaît essentiel de considérer la tâche d'explication comme une tâche de résolution de problème à part entière. Il faudrait que, pour chaque connaissance rajoutée dans la base, le système soit capable de mettre à jour automatiquement les règles d'utilisation de cette connaissance et les règles permettant de donner des explications dans des pas de déduction où intervient cette connaissance. En utilisant les traces d'exercices avec création d'objets ou non, il faudrait que le système découvre automatiquement de nouvelles propriétés et heuristiques de création d'objets qui resserviraient dans des circonstances analogues et rajouteraient automatiquement de nouveaux théorèmes. Mais comment Argos va-t-il juger de la qualité de ce qu'il va retenir? La communauté des mathématiciens met souvent bien longtemps pour reconnaître l'importance de tel ou tel théorème....

12. BIBLIOGRAPHIE

[Bazin 93] J.M Bazin: *GEOMUS un résolveur de problèmes de géométrie qui mobilise ses connaissances en fonction du problème posé*, Thèse université Paris 6.

[Pastre 84] D Pastre. *MUSCADET Un système de démonstration automatique de théorèmes utilisant connaissances et métaconnaissances en mathématique*, Thèse d'état Paris VI, 1984.

[Pastre 89] D Pastre: *MUSCADET an automatic theorem proving system using knowledge and metaknowledge in mathematics*, Journal of Artificial Intelligence 38, p 257-318, 1989.

[Pastre 93] D Pastre:, Volume 8, No. 3-4,p 425-447 ,1993.

[Pastre 99] D Pastre: *Le nouveau MUSCADET et la TPTP Problem Library*. Actes de Berder, 1999.

[Pintado 94] M Pintado: *Apprentissage et démonstration automatique de théorèmes*, Thèse Paris 6, 1994.

[Pitrat 90] J Pitrat: *Métaconnaissances futur de l'intelligence artificielle*, Editions Hermès, 1990.

[Pitrat 99] J Pitrat. *Monitorer la recherche d'une solution*, Actes de Berder, 1999.

[Bernat 94] P Bernat: *CHYPRE: un exemple pour l'enseignement de la géométrie*, Thèse

Université Henri Poincaré- Nancy 1994.

[Melis 99] E Melis et Uri Leron: *A Proof Presentation Suitable for Teaching Proofs*, Artificial Intelligence in Education, SP Lajoie and M. Vivet(Eds), IOS Press, 1999.

[Luengo 99] Vanda Luengo: *A Semi-Empirical Agent for Learning Mathematical Proof*. Artificial Intelligence in Education, SP Lajoie and M. Vivet(Eds), IOS Press, 1999.

[Koedinger 99] V Aleven, Kenneth R. Koedinger, Karen Cross: *Tutoring Answer Explanation Fosters Learning with Understanding*, P 199- 206, Artificial Intelligence in Education, SP Lajoie and M. Vivet(Eds), IOS Press, 1999.

[Nicaud 99] JF Nicaud, D Bouhineau, C Varlet, Anh Nguyen-Xuan. *Towards a product for teaching formal algebra*, P207-214, Artificial Intelligence in Education, SP Lajoie and M. Vivet(Eds), IOS Press, 1999.

[Nguyen-Xuan 99] Anh Nguyen-Xuan, Anne Bastide, JF Nicaud: *Learning to solve polynomial factorization problems: By solving problems and by studying examples of problem solving, with an intelligent learning environnement*, Artificial Intelligence in Education, SP Lajoie and M. Vivet(Eds), IOS Press, 1999.

[Horacek 99] Helmut Horacek: *Presenting Proofs in a Human-Oriented Way*, CADE-16 Atomated Deduction 1999.

[Py 96] Dominique Py: *Aide à la démonstration en géométrie: le projet Mentoniezh*, Sciences et techniques éducatives, Vol. 3-n^{o} 2/1996.

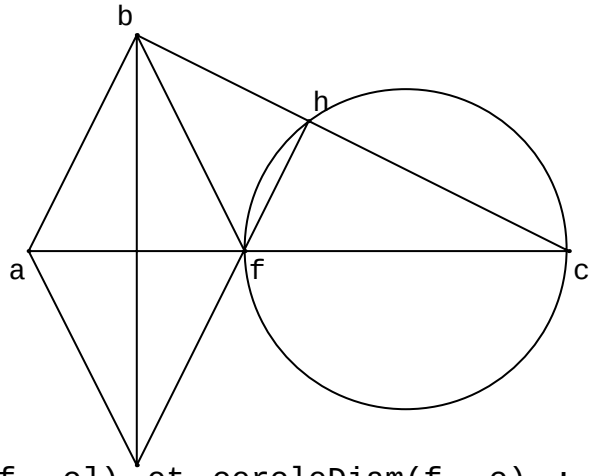
Annexe1:**Exemple d'exercice résolu avec questions enchaînées:**

Enoncé en français :

Soient a,o,f,c 4 points alignés dans cet ordre avec : ao=of=3, ac = 15. Soit b un point tel que ob=6 et (bo) ⊥ (ac). Le cercle de diamètre [fc] coupe la droite (bc) en h. g est le point d'intersection de la droite (hf) avec la parallèle à la droite (bf) passant par a.

1)Calculer les longueurs ab et bc.
2)Montrer que (ab) ⊥ (bc).
3)Montrer que le triangle hfc est rectangle en h.
4) Montrer que (ab) // (fh).
5) Montrer que le triangle baf est isocèle.
6) Montrer que abfg est un losange.

figure:



Enoncé donné au système:

alignesOrd([a, o, f, c])g et cercleDiam(f, c) : cercle et h estDans cercle inter droite(b, c) et droite(b, o) orthog droite(a, c) et longueur(a, c) egal 15 et longueur(a, o) egal 3 et longueur(f, o) egal 3 et longueur(b, o) egal 6 et droite(a, g) paralleles droite(b, f) et g app droite(f, h) => calculer(longueur(a, b)) et calculer(longueur(b, c)) et droite(a, b) orthog droite(b, c) et trRect(h, f, c) et droite(a, b) paralleles droite(f, h) et isocele(b, a, f) et losange(a, b, f, g).

Preuve rédigée du théorème

HYPOTHESES DE DEPART:

les points a , o , f et c sont alignés dans cet ordre

cercle est le cercle de diamètre [f c]
h est à l'intersection de cercle et droite(b, c)
droite(b, o) orthog droite(a, c)
longueur(a, c) = 15
longueur(a, o) = 3
longueur(f, o) = 3
longueur(b, o) = 6
droite(a, g) et droite(b, f) sont des droites parallèles
g est sur la droite droite(f, h)

CONCLUSION(S):

1) calculer(longueur(a, b))
2) calculer(longueur(b, c))
3) droite(a, b) orthog droite(b, c)
4) h f c est un triangle rectangle en h
5) droite(a, b) et droite(f, h) sont des droites parallèles
6) b a f est un triangle isocèle en b
7) a b f g est un losange

DEMONSTRATION:

réponse à la question no: 1

On a b a ^ 2 = o a ^ 2 + b o ^ 2 car le triangle o b a est rectangle en o
Comme a b * a b = b o * b o + a o * a o alors a b = rac(3 * 3 + 6 * 6) = 3 * rac(5)

---

réponse à la question no: 4

Le point h est sur le cercle de diamètre [f c] donc le triangle f c h est rectangle en h

---

réponse à la question no: 2

o appartient au segment [a c] donc a c = a o + o c
Comme a c = a o + o c alors o c = 15 - 3 = 12
On a b c ^ 2 = o c ^ 2 + b o ^ 2 car le triangle o b c est rectangle en o
Comme b c * b c = b o * b o + o c * o c alors b c = rac(12 * 12 + 6 * 6) = 6 * rac(5)

---

réponse à la question no: 3

Le fait: a b = 3 * rac(5) a été déduit d'après la question 1
Le fait: b c = 6 * rac(5) a été déduit d'après la question 2
a c ^ 2 = 225 = b c ^ 2 + b a ^ 2 car a c = 15 b c = 6 * rac(5) et b a = 3 * rac(5)
On a c a ^ 2 = b a ^ 2 + c b ^ 2 donc le triangle b c a est rectangle en b

Les droites (b c) et (b a) sont perpendiculaires car le triangle b c a est rectangle en b
les droites (a b) et (b c) sont perpendiculaires par symétrie !

---

réponse à la question no: 5

Le fait: h f c est un triangle rectangle en h a été déduit d'après la question 4
Les droites (h f) et (h c) sont perpendiculaires car le triangle h f c est rectangle en h
Le fait: les droites (b c) et (a b) sont perpendiculaires a été déduit d'après la question 3
Les droites (f h) et (a b) sont parallèles car (b c) est perpendiculaire (a b) et (f h) est perpendiculaire à (b c)
les droites (a b) et (f h) sont parallèles par symétrie!

---

réponse à la question no: 6

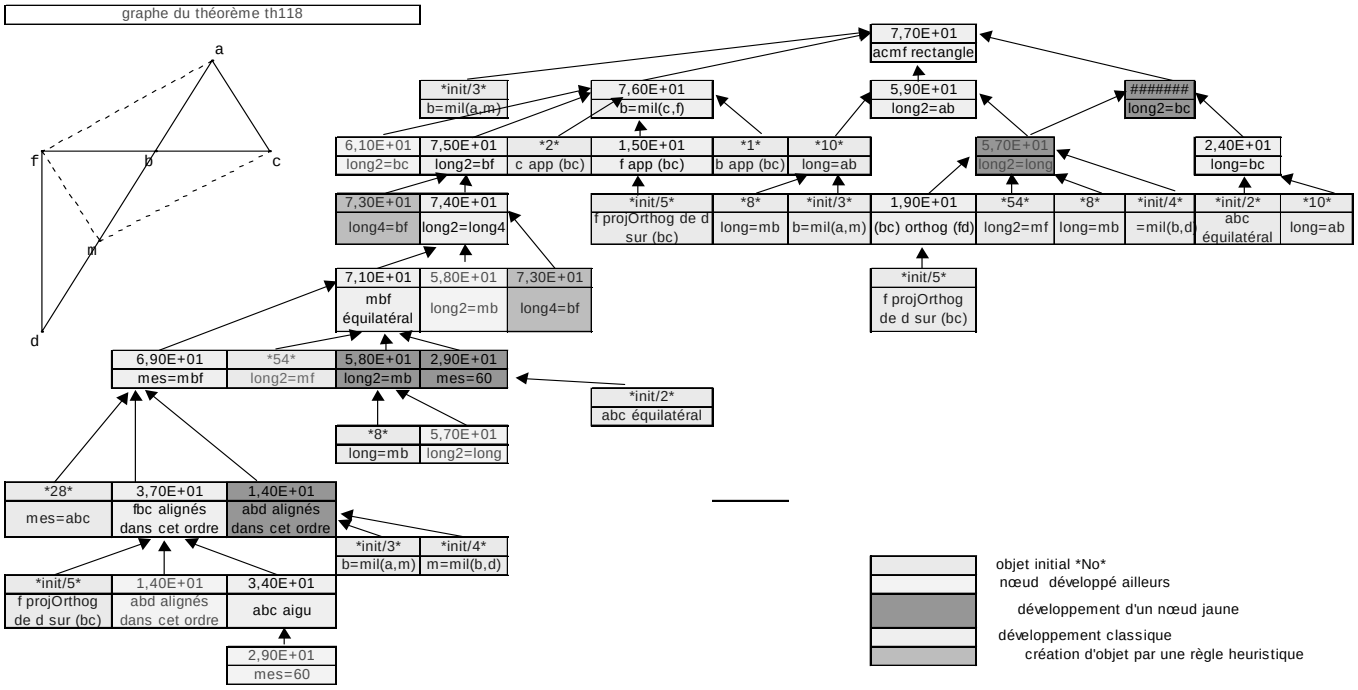
On a a o = f o comme ayant la même valeur exacte: 3 !
On a b f ^ 2 = o f ^ 2 + b o ^ 2 car le triangle o b f est rectangle en o
Comme b f * b f = b o * b o + a o * a o alors b f = rac(3 * 3 + 6 * 6) = 3 * rac(5)
Le fait: a b = 3 * rac(5) a été déduit d'après la question 1
On a a b = b f comme ayant la même valeur exacte: 3 * rac(5) !
b f = b a donc le triangle b f a est isocèle en b
b a f est un triangle isocèle en b par symétrie!

---

réponse à la question no: 7

On a a o = f o comme ayant la même valeur exacte: 3 !
On a b f ^ 2 = o f ^ 2 + b o ^ 2 car le triangle o b f est rectangle en o
Comme b f * b f = b o * b o + a o * a o alors b f = rac(3 * 3 + 6 * 6) = 3 * rac(5)
Le fait: a b = 3 * rac(5) a été déduit d'après la question 1
On a a b = b f comme ayant la même valeur exacte: 3 * rac(5) !
Le fait: les droites (f h) et (a b) sont parallèles a été déduit d'après la question 5
Puisque (g a) // (b f) et (g f) // (b a) alors on en déduit que g a b f est un parallélogramme (côtés opposés parallèles deux à deux)
g a = f b car g a b f est un parallélogramme (côtés opposés d'un parallélogramme)
f g = b a car f g a b est un parallélogramme (côtés opposés d'un parallélogramme)
a b f g est un losange car a b = b f = f g = g a et donc les quatre côtés ont même longueur

Annexe2: graphe complet du théorème ayant servi d'exemple:



Annexe3: Exemple d'objet créé, ici le point h, par le système pour les besoins de la preuve et la façon dont le système l'introduit dans l'explication.

Enoncé en français: (activité 2:ex no3 p 217 Terracher seconde Hachette édition 94)
ABC est un triangle. Le point I est le pied de la bissectrice issue de A. Sur une figure sont indiqués U et V les projetés orthogonaux se I sur (AB) et (AC). En calculant de deux façons chacune des aires: aire(AIB) et aire(AIC), montrer que IB / IC = AB / AC.

th222

ENONCE donné au système:
triangle(a, b, c)et bissInt(b, a, c, droite(a, i))et i app droite(b, c)et projeteOrthog(i, droite(a, b)):u et projeteOrthog(i, droite(a, c)):v=>longueur(i, b)/longueur(i, c)egal longueur(a, b)/longueur(a, c)

HYPOTHESES DE DEPART:
les trois points a b c forment un triangle
bissInt(b, a, c, droite(a, i))
i est sur la droite droite(b, c)
u est le projeté orthogonal de i sur la droite (a b)

v est le projeté orthogonal de i sur la droite (a c)

CONCLUSION à démontrer:
1) longueur(i, b) / longueur(i, c) = longueur(a, b) / longueur(a, c)

DEMONSTRATION:
Soit h le projeté orthogonal du point a sur la droite (b c) %objet introduit par Argos
2*aire(i, c, a) = c a * i v = i c * a h puisque ce sont deux façons de calculer l'aire du triangle à partir des deux hauteurs i v et a h
i c * a h = a c * i v donc i c / a c = i v / a h
aire(a, i, b) = 1/2 * i b * a h car h est le projeté orthogonal de a sur la droite (i b) (formule de l'aire d'un triangle)
i est un point de la bissectrice intérieure de l'angle géométrique b a c donc il est équidistant des deux droites (b a) et (c a) donc i u = i v
aire(i, a, b) = 1/2 * a b * i u car u est le projeté orthogonal de i sur la droite (a b) (formule de l'aire d'un triangle)
Puisque a b *i v = 2 *aire(i a b) = i b *a h alors a b *i v = i b *a h
a b * i v = i b * a h donc a b / i b = a h / i v
Puisque a b /i b = a h /i v = a c /i c alors a b /i b = a c /i c
On a: a b / i b = a c / i c donc i b / i c = a b / a c

Annexe4: Exemple de règles construites à partir d'une même propriété:
propriete([general],[obligatoire],centreGravite,
centreGravite(A,B,C):G et milieu(B,C):I =>
G app droite(A,I)
,[G,'est le centre de gravité du triangle',A,B,C,'donc il appartient à la médiane issue de',A,'cad la droite (',A,I,')']
).

1)Condition suffisante:

```
regle(A, app_CS7, B) :-
     eltConclSym(A, C app D), %on veut montrer une appartenance
     milieu(A, E, F, G, H),          %G est le milieu de [EF]
     centreGravite(A, I, E, F, C, J),  %C est le centre de gravité de IEF
     droite(A, D, I, G, K),          %D est la droite(IG)
     not hyp(A, C app D, L),  % Argos ne sait pas que C appartient à (IG)
     retract1(eltConclSym(A, C app D)),
     ajhyp(A, C app D, M),     %rajouter que C est sur la médiane (IG)
```

```
     memoriser([app_CS7, H, J, K], M),
```
%mémoriser que ce fait provient des faits no: H,J et K et a été obtenu par la règle app_CS7

```
     memoriserExpl([obligatoire], [C, 'est le centre de gravité du triangle', I, E, F, 'donc il appartient à la médiane issue de', I, 'cad la droite (', I, G, ')'], M).
```
%memoriser le modèle explicatif instancié avec les bonnes valeurs de variables

2)Déduction tout venant:

```
regle(A, centreGraviteDeduction, B) :-
     milieu(A, C, D, E, F),
```
%E milieu de [CD]

```
     centreGravite(A, G, C, D, H, I),
```
%H centre de gravité de GCD

```
     droite(A, J, G, E, K),
```
%J = (GE)

```
     \+ hyp(A, H app J, L),
```
%Argos ne sait pas que H app (GE)

```
     ajhyp(A, H app J, M),
```
%rajouter que H app (GE)

```
     memoriser([centreGraviteDeduction, F, I, K], M),
     memoriserExpl([obligatoire], [H, 'est le centre de gravité du triangle', G, C, D, 'donc il appartient à la médiane issue de', G, 'cad la droite (', G, E, ')'], M).
```

3)Règle de création:

```
regle(A, centreGraviteDeductionCreation, B) :-
     milieu(A, C, D, E, F),
     centreGravite(A, G, C, D, H, I),
     distincts(G, E),
     not droite(A, J, G, E, K),
```
%La droite (GE) n'existe pas pour Argos

```
     creerDroite(A, G, E, L, M),
```
%créer la droite (GE)

```
     creePar([centreGraviteDeductionCreation, F, I], M).
```
%mémoriser les conditions de la création

Annexe5:

Enoncé en français:

Soit abcd un trapèze avec (ab) // (cd). Les droites (ad) et (bc) se coupent en o. Soient i et j les milieux respectifs des segments [ab] et [cd]. 1) Montrer que les points o,i et j sont alignés. (indication: on pourra considérer l'homothétie de centre o envoyant a en d). 2) Montrer que les triangles cio et dio ont la même aire.

th221bis (niveau première S)

ENONCE donné au système:

trapeze(a, d, c, b)et o estDans droite(a, d)inter droite(b, c)et milieu(a, b):i et milieu(d, c):j et hom(o, a, d):hom=>aire(c, i, o)egal aire(d, i, o)et j app droite(o, i)

HYPOTHESES DE DEPART:
a d c b est un trapèze
o est à l'intersection des droite(a, d) et droite(b, c)
i est le milieu du segment [a b]
j est le milieu du segment [d c]
hom est l'homothétie de centre o envoyant a en d

LISTE DES CONCLUSIONS à démontrer:
1) aire(c, i, o)egal aire(d, i, o)
2) j est sur la droite droite(o, i)

DEMONSTRATION:
réponse à une conjecture
a i = b i car i = mil(a, b)
aire(c, i, b) = aire(d, i, a) car les deux triangles ont chacun un sommet c et d sur (d c) et i b i a alignés sur (a b) paralléle à (d c) donc ils ont la même hauteur avec i b = i a comme base

---

réponse à une conjecture
aire(o, i, b) = aire(o, i, a) car les deux triangles ont un sommet commun o et i b i a alignés avec i b = i a

---

réponse à la question no: 1
aire(o, i, c) = aire(o, i, d) car aire(o, i, b) = aire(o, i, a) et aire(o, c, b) = aire(o, d, a)

---

réponse à la question no: 2
Puisque droite(a, b) // droite(d, c) et que les points o b et c sont alignés alors l'homothétie de centre o envoyant a en d envoie b en c
Les points a et b se transforment par l'homothétie hom en les points d et c or i est le milieu de [a b] et j est le milieu de [d c] donc j est l'image de i par hom car une homothétie conserve le milieu
Dans une homothétie un point et son image sont alignés avec le centre , donc comme ici i se transforme en j par l' homothétie de centre o et de rapport rap alors les points o i j sont alignés
les points o i et j sont alignés donc j est un point de la droite (o i)