Projet de résolution de problèmes : Satisfaction de contraintes pour le Wordle Mind

Alessia LOI, Antoine THOMAS

18 avril 2022

1 Modélisation et résolution par CSP

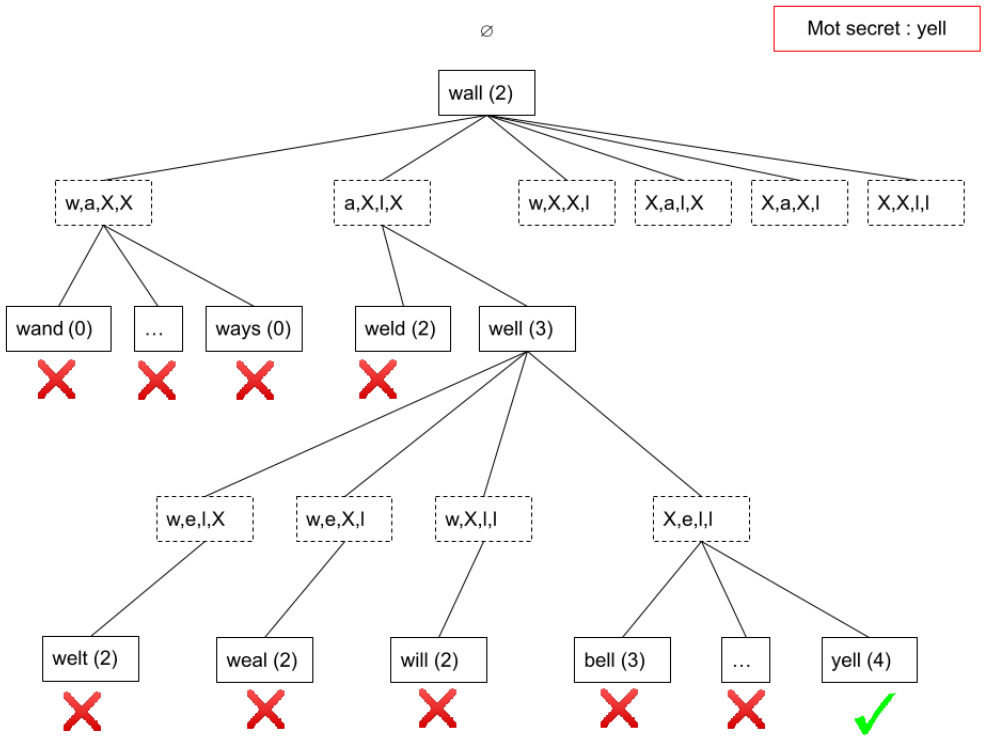
Nous considérons le domaine de départ comme l'ensemble des mots du dictionnaire. En fonction des mots trouvés et des résultats de la fonction d'évaluation, ce domaine se restreint.

1.1 Retour arrière chronologique

L'algorithme de backtracking a été implémenté de la manière suivante : Pour un mot de n lettres, on teste son score (on ne regardera que le résultat du nombre de lettres bien placées). Si on a n lettres bien placées alors le mot et le bon. Si on a 0 lettres bien placées, on choisit un autre mot. Sinon on crée plusieurs domaines en fonction du nombre de lettres bien placées. Par exemple pour un mot de 4 lettres avec 2 lettres bien placées, on a 6 domaines différents qui correspondent aux combinaisons de lettres bien placées. On choisit ensuite un domaine puis on le parcourt en instanciant les mots du domaine et on teste leur score. Si le score d'un mot est plus élevé (dans notre exemple 3 lettres bien placées), on sélectionne ce mot et on crée de nouveaux domaines à partir de ce mot. Si on ne trouve aucun mot avec un score plus élevé alors on backtrack et on change de domaine. Dans tous les cas, cela nécessite une instanciation des mots du domaine visé.

On va réduire les domaines en parcourant en profondeur les branches de l'arbre en choisissant la branche avec le score le plus élevé. Par exemple l'ensemble des mots commençant par ab. Cependant la vérification de la contrainte (n lettres bien placées pour un mot de n lettres) n'arrive qu'à l'instanciation de la dernière variable (c'est à dire lorsqu'on instancie un mot complet). Si le domaine est réduit au maximum et que la contrainte n'est pas vérifiée, alors on fait un backtrack.

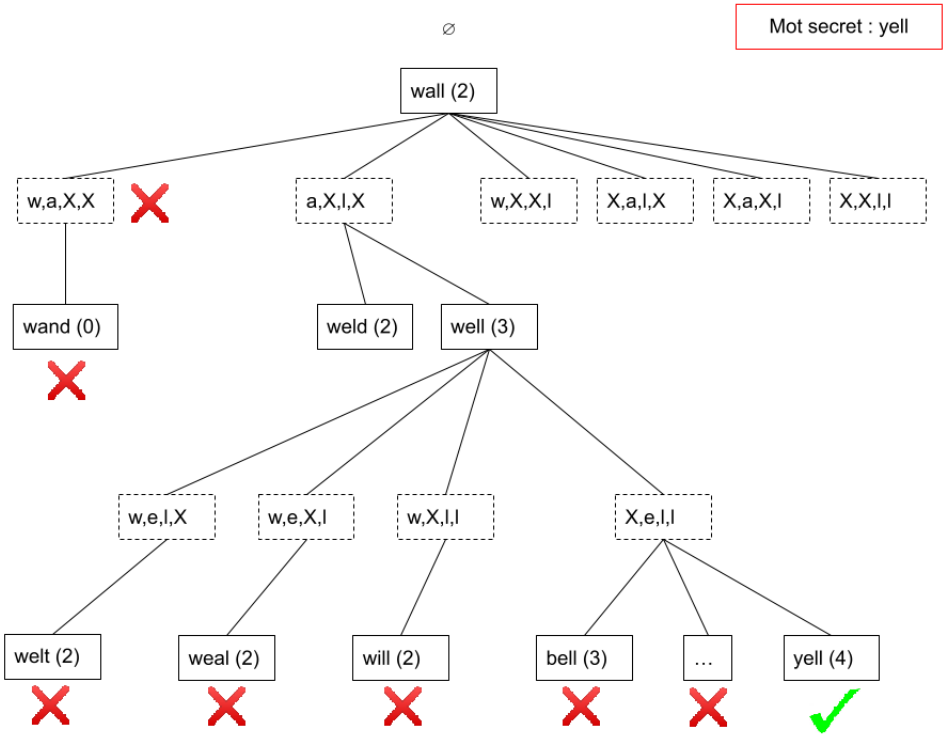
Figure 1: Arbre de l'algorithme backtrack



1.2 Retour arrière chronologique avec arc cohérence

Nous utilisons ici la méthode du forward checking qui permet de détecter les incohérences plus tôt qu'un backtracking en permettant l'élagage anticipé des branches de l'arbre qui aboutiraient à un échec. Cela réduit donc le nombre de calculs. (noeuds à visiter, ce qui se traduit par une complexité inférieure en termes de temps de calcul) Pour implémenter le forward checking, on compare le score du résultat trouvé précédemment avec le nœud en cours. Si le nouveau score est inférieur au score en cours, on élimine le domaine dont provient le score et on passe au domaine suivant.

Figure 2: Arbre de l'algorithme forward



2 Modélisation et résolution par algorithme génétique

Soit un mot secret de longueur n choisi au hasard dans le vocabulaire, la modélisation par algorithme génétique permet d' extraire le prochain mot à jouer parmi un ensemble E de mots compatibles, obtenu en respectant les contraintes déduites à partir des essais précédents, dans le but de deviner le mot secret.

Pseudo-code pour l'algorithme findNextTry

$eSet \leftarrow \emptyset$
$pop \leftarrow$ liste d'individus générés à partir du mot essayé précédemment passé en paramètre
$popFitnesses \leftarrow$ liste d'évaluations, une pour tout individu de la population
$eSet \leftarrow$ sélection des meilleurs individus dans la population compatible avec les essais précédents

Répéter

Sélectionner les meilleurs individus dans la population en fonction de leur fitness
$pop \leftarrow$ nouvelle liste d'individus générés à partir des individus sélectionnés de la génération précédente
$popFitnesses \leftarrow$ liste d'évaluations, une pour tout individu de la population
$eSet \leftarrow$ sélection des meilleurs individus dans la population compatible avec les essais précédents

Jusqu'à
Condition 1 : taille($eSet$) $\leq$ taille maximale $eSet$ définie
Condition 2 : nombre de générations $<$ nombre de générations maximal défini
ou, si taille($eSet$) $= 0$, jusqu'à ce que le temps supplémentaire accordé pour la recherche d'un mot compatible ne soit pas épuisé

Retourner un mot au hasard parmi les éléments de l'ensemble $eSet$ si $eSet$ non vide, *None* sinon.

La compatibilité d'un individu avec les essais précédents correspond à la propriété de tel individu de respecter toutes les contraintes définies en fonction des tentatives précédentes.

Une contrainte est un triplet (mot, nombre de lettres correctes bien placées, nombre de lettres correctes mal placées), défini lorsqu'un nouveau mot est joué et l'algorithme nous retourne les deux heuristiques sur le nombre de lettres correctes par rapport à la ressemblance avec le mot secret. Une contrainte supplémentaire est fournie par le tableau forbiddenLetters, qui stocke les lettres qui ne peuvent pas apparaître dans le mot secret. Cette information est déduite lorsqu'on a zero lettres correctes bien placées et zéro lettres correctes mal placées pour une tentative.

L'évaluation d'un individu (fitness) correspond dans cette partie au nombre de contraintes respectées.

L'algorithme génétique utilise différentes méthodes de croisement pour la génération de la nouvelle population :

- onePointCrossover : l'individu résultant sera constitué d'une partie de lettres d'un parent1 et d'une partie des lettres du parent2, à droite et à gauche d'un point de coupure aléatoire.
- twoPointsCrossover : en fonction de deux points de coupure p1 et p2 choisis aléatoirement, l'individu résultant sera constitué des lettres du parent1 aux extrémités et des lettres du parent2 au centre

Chacune de ces méthodes est appliquée à la population avec une probabilité définie dans la variable crossRate.

L'algorithme génétique utilise différentes méthodes de mutation pour la génération de la nouvelle population :

- aleaCharMutation : l'individu résultant est obtenu en modifiant une lettre de son parent. Le caractère est choisi parmi ceux qui n'appartiennent pas au tableau forbiddenLetters.
- swapMutation : l'individu résultant est obtenu en échangeant la position de deux lettres de son parent.

Chacune de ces méthodes est appliquée à la population avec une probabilité définie dans la variable mutationRate.

L'algorithme génétique utilise différentes méthodes de sélection pour définir la construction de la nouvelle population :

- kTournament : les k meilleurs individus (avec une fitness plus élevée) sont utilisés pour générer les enfants. La population suivante sera constituée des meilleurs parmi les parents et les enfants.
- uPlusLambdaSelection : u parents sont conservés pour générer lambda enfants, la nouvelle population sera constituée des parents plus les enfants (on privilégie ici l'exploitation de l'espace de recherche).
- lambdaSelection : u parents sont conservés pour générer lambda enfants, la nouvelle population sera constituée que des lambda enfants (on privilégie ici l'exploration de l'espace de recherche).

Vous pouvez trouver le jeu de paramètres le plus efficace qui a été testé dans le fichier bestParams. Il représente un bon compromis d'exploration de l'espace des solutions possibles car il selectionne les meilleurs individus à chaque génération, et chaque nouvelle population est constituée d'une bonne quantité de lettres différentes.

3 Détermination de la meilleure tentative

L'optimisation de l'algorithme génétique de la partie 2 du projet dans l'objectif de sélectionner une meilleure tentative dans l'espace des solutions possibles s'appuie sur les améliorations suivantes :

- Le premier mot à jouer ne sera plus choisi au hasard dans le vocabulaire, nous préférons choisir un mot qui contient le plus grand nombre de lettres différentes. Cette stratégie devrait permettre une plus grande exploration de l'espace des solutions admissibles déjà à la première génération.
- Nous avons ajouté un mécanisme, géré par la fonction "checkStagnation", qui permet de détecter des situations de bouclage dans la génération de nouveaux individus : si la génération aléatoire boucle sur les mêmes lettres pendant un certain nombre de générations, et si les fitnesses ne progressent pas, alors il se déclenche une stratégie ponctuelle qui introduit des nouveaux mots distants au sens de la distance de Hamming dans la nouvelle population. Ce mécanisme devrait permettre de sortir de situations de optimum local qui représentent bien le respect de contraintes mais malheureusement ne ressemblent pas au mot secret.
- La fitness de chaque individu de la population est calculée de manière à emphatiser les distances entre individus par rapport aux heuristiques fournies et donc raffiner la valeur informative correspondante. Détail du calcul implémenté :

  – Soit un individu ind appartenant à la population à sélectionner. Soit gp le nombre de lettres égales et à la même position entre deux mots m1 et m2. Soit bp le nombre de lettres égales mais à des positions différentes entre deux mots m1 et m2. Pour tout mot prec appartenant à l'ensemble des essaies précédents (tableau contraintes) :

  * Si $gp(ind) = gp(prec)$ et $bp(ind) >= bp(prec)$, alors $reward = (rp \times 2)^2 + (bp \times 2)^2$
  * Si $gp(ind) = gp(prec)$ mais $bp(ind) < bp(prec)$, alors $reward = (rp \times 2)^2$
  * Si $gp(ind) > gp(prec)$ et $bp(ind) >= bp(prec)$, alors $reward = rp + bp$
  * Si $gp(ind) > gp(prec)$ mais $bp(ind) < bp(prec)$, alors $reward = rp$

  Le reward obtenu pour chaque contrainte est sommé et a l'objectif d'attribuer une évaluation plus élevée aux individus qui ont une valeur de gp et bp plus similaire aux valeurs des essais précédents, en priorisant l'effet de la valeur de gp par rapport à celui de bp.
- Le choix de la prochaine tentative ne se fait plus au hasard en piochant un mot de l'ensemble E. Dans cette version nous préférons choisir le mot qui a collecté une fitness plus élevée suite à la comparaison avec l'ensemble des essais précédents (tableau contraintes). Pour cela, l'ensemble E contient désormais des doublets où l'on associe chaque mot à la fitness obtenue pour sa sélection. Nous espérons ainsi non seulement de deviner plus rapidement le mot secret, mais aussi de fournir un mot avec une évaluation élevée utile pour la constitution de la génération d'individus successive.

4 Analyse des résultats expérimentaux

Figure 3: Temps moyen d'exécution du backtracking sur 20 instances

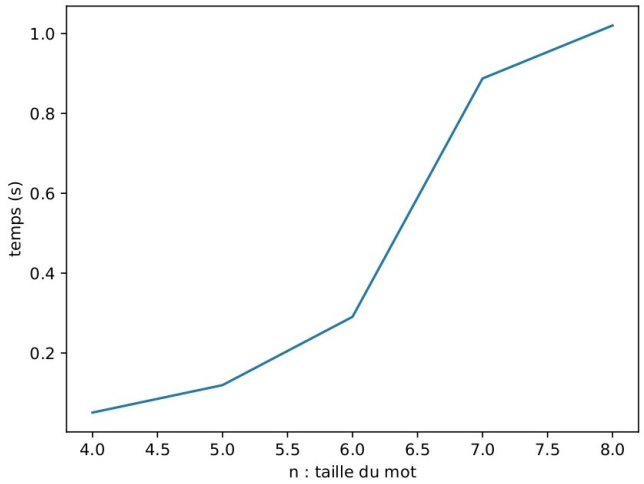


Figure 4: Nombre moyen d'essais du backtracking sur 20 instaces

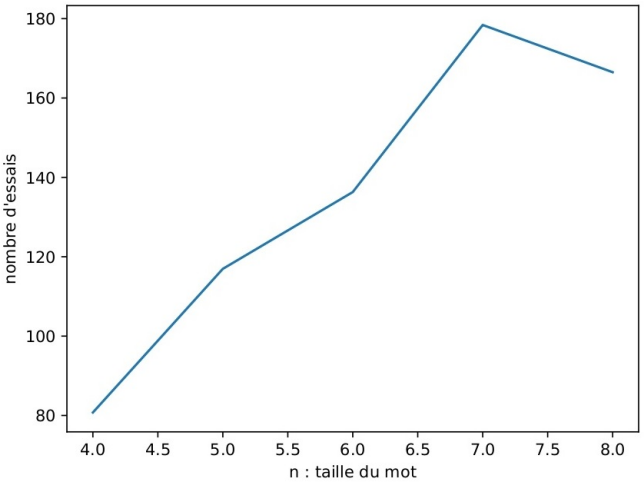


Figure 3, figure 4. L'algorithme du backtracking se révèle très rapide en termes de temps de calcul, par contre le nombre d'essaies nécessaires avant de trouver la solution est assez élevé et s'atteste autour de 80 essais pour un mot de 4 lettres, jusqu'à environ 160-180 essais pour un mot au de la de 7 lettres.

Figure 5: Temps moyen d'exécution du forward checking sur 20 instances

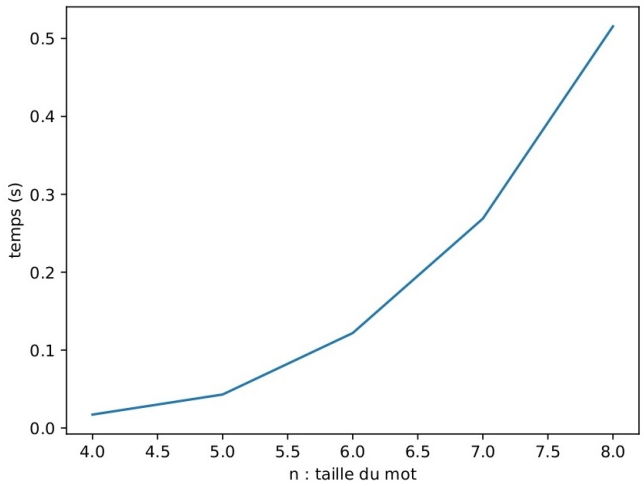


Figure 6: Nombre moyen d'essais du forward checking sur 20 instaces

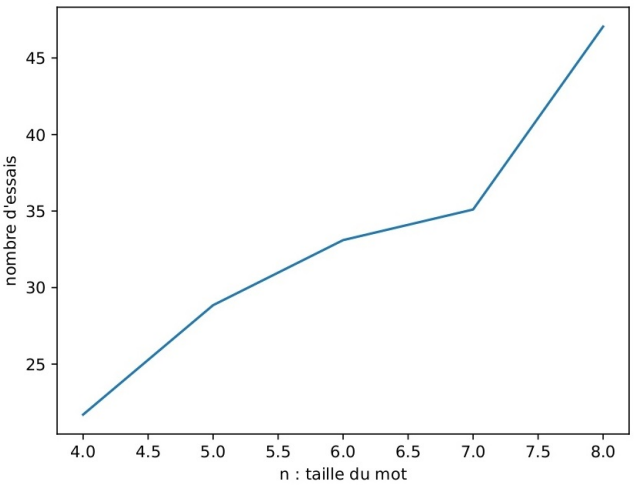


Figure 5, figure 6. L'algorithme du forward checking se révèle plus performant par rapport aux nombre d'essais, car il arrive à élaguer des sous-ensembles de tentatives en fonction des informations obtenues dans les essais précédents. Le mécanisme d'arc consistance limite comme prévu l'exploration de l'espace des mots compatibles. Le temps moyen d'exécution est deux fois plus rapide que le backtracking.