1. **Algorithmes implantés**
2. Le premier algorithme implanté est un algorithme CSP dans lequel les variables sont les mots entiers. À chaque fois qu’un mot est choisi, une contrainte d’unicité est ajoutée à chacune des lettres qu’il contient et qui est située à une intersection avec un autre mots dans la grille. Cet algorithme est de loin le plus performant étant donné qu’il arrive à trouver une solution en moins d’une seconde (environs 0,5 sec) à des grilles aussi grandes que (25 x 20). Le fichier de cet algorithme a été nommé ”csp\_words.py”.
3. Le deuxième algorithme est un algorithme CSP hybride. Les variables sont les premières lettres de chaque mot. Une fois que la première letter d’un mot est instanciée, la variable deviant le mot constraint à commencer par la lettre qui a été instanciée. Cet algorithme résoud sans problème les petites grilles avec quelques intersections seulement: grilles de (8 x 10) avec 2 ou 3 intersections résolues en moins d’une seconde. Dès que la grille grandit un peu (ex: ajout d’une intersection, une ligne ou une colonne), le temps de resolution monte à une dizaine de seconds, et peut même atteindre plusieurs minutes. Le fichier contenant l’implantation de cet algorithme est “ csp\_hybrid.py”.
4. Le troisième algorithme est un algorithme génétique, dans lequel les chromosomes sont formés en accolant les mots de la grille. Ces derniers sont générés aléatroirement à partir du dictionnaire. La function de fitness donne le nombre de conflits existant pour chaque chromosome généré. La liste des chromosomes générés est triée en function de la valeur de fitness de chaque chromosome et seuls les 100 meilleurs chromosomes sont gardés pour faire des mutations ou croisements afin de générer des enfants dans l’espoir de render nulle la valeur de fitness.

L’algorithme génétique converge rapidement vers un minimum de 1 ou 2 conflits mais ne parvient que très rarement à trouver la solution (pour la petite grille seulement). Pour les grilles plus grandes, le temps de calcul peut atteindre plusieurs heures; on voit alors la valeur de fitness diminuer sans jamais arriver à 0. Ceci peut s’expliquer par la nature meme des algorithmes génétiques dont le but est seulement d’approcher la solution optimale sans toutefois nécessairement l’atteindre.

Le fichier contenant l’implantation de cet algorithme est “ genetic.py”. Comme il est rare d’obtenir une solution, on affiche les valeurs des 5 meilleurs chromosomes trouvés à chaque étape afin d’avoir une idée de la performance de l’algorithme.

1. Une tentative d’implanter un algorithme CSP avec comme variables les lettres de chaque grille et des contraintes sur les intersection a été mené, mais malheureusement, nous n’avons pas réussi à le render fonctionnel.
2. **Performance des algorithmes utilisés et comparaison**
3. Algorithme CSP sur les mots: cet algorithme est de loin le plus performant des trois. Il résoud toutes les grilles, quelles que soient la taille et la complexité, dans un temps extrêmement court. Le tableau suivant montre les résultats obtenus:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grille | Extrasmall | Small l | Medium | Large |
| Taille de la grille | 6 x 6 | 8 x 10 | 17 x 11 | 25 x 20 |
| Temps moyens (pour 10 exécutions) en secondes | 0.055 | 0.075 | 0.193 | 0.638 |

1. CSP hybride: il arrive toujours à résoudre la plus petite grille dans un temps pouvant aller de 5 à 45 secondes. Dès qu’on passe aux grilles de plus grande taille, le temps de résolution devient indéterminé.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grille | Extrasmall | Small l | Medium | Large |
| Taille de la grille | 6 x 6 | 8 x 10 | 17 x 11 | 25 x 20 |
| Temps moyens (pour 10 exécutions) en secondes | 16.04 |  | indéterminé | indéterminé |

1. Algorithme génétique: parvient toujours à trouver la solution pour la plus petite grille (extrasmall) en 1 seconde en moyenne. La grille suivante (small) n’est résolue qu’au bout de plusieurs minutes de calcul (entre 2 et 10 minutes), et ce de façon aléatoire (il lui arrive de ne pas converger du tout vers 0). Les deux autres grilles ne sont jamais résolues avec cet algorithme, pour le temps durant lequel on l’a laissé rouler (plusieurs dizaines de minutes.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Grille | Extrasmall | Small l | Medium | Large |
| Taille de la grille | 6 x 6 | 8 x 10 | 17 x 11 | 25 x 20 |
| Temps moyen (pour 10 exécutions) en secondes | 1.00 | 130.00 | indéterminé | indéterminé |

On peut toutefois voir sa performance en executant le programme qui l’implante et en regardant l’affichage des résultats intermédiaires. On peut y voir la liste des 10 meilleurs chromosomes et leur fitness, le temps de calcul intermédiaire et le nombre de chromosomes générés jusque-là.

1. **Comment exécuter les différents algorithmes**

Pour exécuter un des algorithmes, il suffit de taper dans la console le nom du fichier dans lequel est implanté l’algorithme, suivi du nom du fichier contenant le masque de la grille, suivi enfin du nom du fichier contenant la liste des mot du dictionnaire.

Par exemple, pour exécuter l’algorithme CSP avec les mots comme variables avec la plus grande grille disponible, il faut taper:

*>> csp\_word.py large.mask dico.txt*

À noter qu’il y a 4 grilles à tester: *extrasmall*, *small*, *medium* et *large*.

1. **Conclusion**

Il apparaît clairement que l’algorithme CSP avec contraintes sur les mots présente la meilleure option. L’espace d’états de chaque variables est plus réduit, ce qui le rend plus performant quell que soit le niveau de difficulté de la grille. Les algorithmes hybride et génétique arrivent à résoudre la plus petite grille, mais leurs performances se dégradent rapidement lorsuq’on augmente un tant soit peu la complexité de la grille.