UNIVERSITÉ ANGERS

PROJET ÉTUDIANT

Voile d'ombrage et algorithme génétique

Auteur :
Alexandre ROBIN
Théo VOILLEMIN
Vincent HÉNAUX
Nathan DESAGES

Superviseur :
Frédéric LARDEUX

19 Février 2018

université angers

FACULTÉ
DES SCIENCES
Unité de formation
et de recherche
DÉPARTEMENT
INFORMATIQUE

Résumé

Dans ce papier, nous proposons une étude de l'utilisation d'algorithme génétique sur le problème de voile d'ombrage, notamment l'implémentation de différents opérateurs afin de démontrer leur utilité, ainsi que d'une interface graphique visant à rendre la comparaison de résultats plus accessible et intuitive. L'étude consistera en la démonstration de résultats visant à montrer l'efficacité de ces différents opérateurs sur le problème de voile d'ombrage, l'explication de l'implémentation de l'interface graphique ainsi que des différents tests démontrant l'efficacité de notre implémentation.

0.1 Mots-clés

Algorithme génétique, voile d'ombrage.

0.2 Catégories et sujets

Intelligence Artificielle, Résolution de problèmes.

0.3 Termes Généraux

Algorithme, Interface graphique.

1 INTRODUCTION

Le problème de voile d'ombrage consiste en la résolution du problème suivant : tenter de maximiser le taux d'ombrage d'une surface prédéfinie à l'aide de la position du soleil ainsi que de la projection d'ombre produite par une voile étendue à partir de n points d'accroches, ces derniers définis selon les trois coordonnées de l'espace en trois dimensions.

A cela s'ajoute une difficulté, à savoir que l'on souhaite non pas maximiser ce taux d'ombrage sur une seule projection de l'ombre de la voile sur la surface, mais sur toute une période pouvant aller de quelques heures à plusieurs mois provoquant ainsi une projection d'ombre différente de la voile à chaque période, la voile étant fixée.

Les algorithmes génétiques sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle. Ils sont composés d'une population contenant des individus représentant des solutions potentielles, arbitrairement choisies. Ces individus sont évalués pour déterminer leur performance (fitness) relative quant à la solution qu'ils proposent au problème initial. En se basant sur ces mesures de performances, une nouvelle population est créée par le biais d'opérateurs évolutionnaires : la sélection, le croisement et la mutation. Ce cycle étant recommencé jusqu'à établissement d'une solution satisfaisante.

Les algorithmes génétiques, de par leur simplicité de mise en application et leur efficacité pour des problèmes complexes, se sont vus être utilisés dans de nombreux domaines telle l'optimisation de fonctions, la finance, la théorie des jeux, etc...

Concernant le problème de voile d'ombrage, les algorithmes génétiques peuvent offrir des avantages que d'autres méthodes, telle la recherche locale, n'ont pas. Car même si les algorithmes génétiques sont plus coûteux en espace mémoire, notamment pour stocker plusieurs dizaines d'individus stockant chacun une position de voile, l'aléatoire offert par les opérateurs de mutations et de croisements offrent des espaces de recherches plus grand que la recherche locale tout en offrant un risque moins grand d'atteindre un optimum local si la population est bien gérée.

Ce papier présente donc rapidement l'encodage du problème de voile d'ombrage, d'algorithme génétique et d'interface graphique avant de présenter les différents tests effectués et résultats obtenus.

2 Préliminaires

2.1 Voile d'ombrage

Le problème de voile d'ombrage fait intervenir le problème de projection d'un plan étant la voile sur un autre plan représenté par le sol. Cette projection est réalisée en fonction des trois coordonnées des vecteurs directeurs représentant la lumière du soleil $directeur[\ ]$ afin de pouvoir calculer l'ombre projetée au sol par une voile accrochée. Nous fixerons donc tout d'abord pour chaque point de la projection une hauteur égale à 0 du fait que nous considérons notre sol à une altitude nulle par rapport à la hauteur de la voile ou du soleil. Enfin, nous pourrons appliquer la formule de projection sur tous les points d'accroches de la voile pour la largeur et la profondeur des points (x, y et z) de telle sorte que la projection d'un point soit :

$$projectionX = (-directeur[0] * \frac{z}{-directeur[2]}) + x$$

$$projectionY = (-directeur[1] * \frac{z}{-directeur[2]}) + y$$

$$projectionZ = 0$$

Grâce à cette projection, une évaluation pourra être effectuée pour chaque individu représentant des points d'accroches différents. Le calcul du taux d'ombrage se servira ensuite de cette projection afin de maximiser cette ombre par notre algorithme génétique. Pour cela, voyons comment est défini ce dernier.

2.2 Algorithme génétique

Un algorithme génétique est défini par :
— *Individu* : une solution potentielle du problème ;
— *Population* : un ensemble d'individus de l'espace de recherche ;
— *Environnement* : l'espace de recherche ;
— *Fonction de fitness* : la fonction que nous cherchons à maximiser.

Nous appelons un individu I de longueur l(I) une suite I = $\{i_1, ..., i_n\}$ représentant une solution potentielle du problème de telle sorte qu'elle peut être codée en binaire, dans ce cas :

$$\forall i \in [1, l], a_i \in v = \{0, 1\}$$

ou encore comme une suite de nombres réels :

$$\forall i \in [1, l], a_i \in v \in \mathbb{R}$$

Cet encodage de solution peut aussi être appelé génome.

La fitness est donnée par une fonction à valeurs positives réelles. Le but d'un algorithme génétique étant simplement de trouver le meilleur individu maximisant ou minimisant cette fonction.

Pour arriver à maximiser ou minimiser la fitness, l'algorithme génétique fait appel à différents opérateurs qu'il applique sur les individus de sa population :

- opérateur de sélection : Cet opérateur consiste en la sélection des individus qui seront par la suite croisés. Certains opérateurs accordent plus de crédit à l'aléatoire tandis que d'autres favorisent plus la qualité des individus à choisir.
- opérateur de croisement : Cet opérateur s'occupe de croiser l'encodage de solution des individus précédemment sélectionnés. Cela consiste à mélanger plusieurs individus ensemble afin de créer des individus enfants qui partagera une partie d'information contenue dans chacun de ses parents.
- opérateur de mutation : Cet opérateur s'occupe de muter l'individu enfant précédemment créé. L'opérateur s'applique directement sur l'encodage de l'individu et peut avoir plusieurs buts distincts comme favoriser la diversité des solutions représentées dans la population en faisant du bruit au sein de l'individu, mais il peut aussi plutôt vouloir améliorer au mieux l'individu, notamment à l'aide, parfois, de recherche locale.
- opérateur d'insertion : Cet opérateur consiste en l'insertion des individus enfants mutés dans la population actuelle. La population étant généralement de taille fixe, cette opérateur s'occupe de savoir s'il faut intégrer le nouvel individu dans la population, et si oui, à la place de qui. Certains de ces opérateurs préfèrent prendre en compte la qualité des individus tandis que d'autres préfèrent favoriser l'âge, le temps passé dans la population des individus.

Un algorithme génétique consiste donc en l'instanciation d'une population, souvent de manière aléatoire afin de favoriser la diversité, puis s'applique un cycle des quatre opérateurs et d'une évaluation de la population afin de mettre à jour la performance de tous les individus. L'arrêt de ce cycle se fait selon plusieurs conditions. Soit l'on connaît la valeur maximale associée à la fonction de fitness, valeur maximale correspondant à la solution optimale de notre problème, et dans ce cas on peut s'arrêter lorsqu'un individu atteint cette fitness ; dans d'autres cas

on préférera attribuer un temps d'exécution ou un nombre fixe de cycle avant l'arrêt de l'algorithme et le retour du meilleur individu trouvé jusqu'alors.

Plusieurs paramètres sont à prendre en compte lors de l'exécution d'un algorithme génétique. Il s'agit notamment de la taille de la population mais aussi de taux, comme le taux de croisement et de mutation.

3 Implémentation d'un algorithme génétique appliqué au problème de voile d'ombrage

3.1 Calcul des positions du soleil

Avant de calculer la projection de l'ombre de la voile à partir de la source lumineuse, à savoir le soleil, il est primordial de connaître sa position, ses coordonnées sphériques. Pour cela, nous nous sommes aidé d'une libraire C/C++ nommée SolTrack permettant de calculer l'azimut et l'altitude du soleil en donnant la date et l'heure, ainsi que de la latitude et la longitude de notre position.

Nous nous sommes donc servi de cette librairie pour calculer et récupérer toutes les positions du soleil sur une période donnée à un endroit précis. Le temps d'exécution d'un algorithme génétique variant énormément sur le nombre de données à traiter, et sachant que la période dont on souhaite maximiser l'ombrage peut varier de quelques jours à quelques mois, nous avons implémanté la possibilité de varier le pas entre deux positions sur une période, allant de toutes les minutes à une par jour.

SolTrack retournant, pour chaque intervalle, les positions du soleil sous la forme de son azimut et de son altitude, il nous a fallu les traduire en coordonnées polaires. Nous récupérons donc finalement un vecteur contenant toutes les coordonnées polaires nécessaires au calcul de la projection de l'ombre de la voile.

3.2 Individu et fitness

Les individus de notre population contiennent assez peu d'informations, ils encodent chacun uniquement les trois coordonnées de chacun des trois points représentant l'accroche de la voile, ainsi que le pourcentage d'ombrage provoqué.

La fitness quant à elle a posé plus de problèmes. En effet, pour pouvoir calculer le pourcentage d'ombrage, il a été décidé que nous allions découper la surface cible en un certain nombre de points et que nous allions regarder lesquels de ces points seraient contenus dans la projection de l'ombre de la voile. La difficulté venait du fait qu'il fallait à la fois essayer de garder une précision suffisament fine pour avoir une appréciation réaliste de la qualité de l'ombrage proposée par un individu, tout en minimisant ce nombre de points afin de réduire au maximum le nombre de calculs, chaque itération calculant de nouveau les projections.

Ce nombre de points est fixé selon un pas $step\_points\_floor$ représentant la distance entre deux points d'un même axe. Pour ce calcul, la zone à couvrir

d'ombre devra donc être représentée par un carré ou un rectangle. Des poids ont aussi été attribués à chacun de ses points afin de favoriser l'ombrage de certains points par rapport à d'autre. Ainsi, dans notre implémentation, les points centraux de la terrasse ont été favorisés par rapport aux points de bordure.

3.3 Fonction d'évaluation

Voyons donc comment l'algorithme de calcul de la fitness décrit précédemment est implémenté.

Nous devrons d'abord disposer de trois paramètres indispensables au bon calcul de l'évaluation :

— *La voile*, représentée par ses points d'accroches ;

— *Le vecteur directeur du soleil* ;

— *La terrasse*, représentée par des points définissants les quatres coins.

Le calcul de la projection de la voile sera nécessaire dès le début afin de disposer des trois points représentant l'ombre de la voile sur le sol où $z = 0$ ainsi que la distance maximale $distance\_max$ entre le centre de la terrasse et ses coins. Cela nous permettra donc par la suite de pouvoir attribuer différents poids en fonction de la distance d'un point calculé et le centre de la terrasse. Nous allons maintenant pouvoir boucler sur les points de celle-ci de telle sorte à la parcourir de la même manière qu'une matrice. Pour chaque point entre chaque coin représenté par un tuple (x, y) incrémenté par un pas $step\_points\_floor$, nous calculerons la distance de ce point au centre de la terrasse $distance\_depuis\_centre$ afin d'incrémenter un compteur $total\_distance$ de

$$total\_distance = total\_distance + 1 - \frac{distance\_depuis\_centre}{distance\_max}.$$

Si le point représenté se trouve être dans la projection de la voile, alors un autre compteur *montant* sera incrémenté de la même manière. À la fin du parcours des différents points, le taux d'ombrage est ensuite retourné grâce à la division

$$taux\_ombrage = \frac{montant}{total\_distance}.$$

Comme nous venons de le voir dans ce calcul du taux d'ombrage, nous devons à un moment donné pouvoir déterminer si un point est dans la projection de la voile sur le sol. Pour cela, nous calculerons l'aire de la projection $aire\_projection$ ainsi que l'aire de trois "sous-triangles" formés par le point concerné par les

boucles précédentes et deux des trois points d'accroches de la voile. Une opération booléenne consistera à tester l'égalité entre $aire\_projection$ et la somme des aires des trois "sous-triangles". Si ces aires se trouvent être égales, alors nous savons que notre point est inscrit dans le triangle représenté par la projection de la voile.

Maintenant que nous pouvons calculer le taux d'ombrage pour un individu donné, et une position du soleil, nous devons le faire pour toutes les positions de la période choisie et une moyenne de l'ombrage sera alors réalisée. L'aire de la voile est calculée et normalisée en la divisant par la plus grande aire de voile trouvée dans les individus actuels ou passés. L'évaluation pourra être maintenant déterminée en multipliant le taux d'ombrage par le coefficient choisi pour l'ombre et réciproquement pour l'aire de la voile de manière à obtenir une fitness prenant en compte l'ombre projetée par la voile sur la terrasse que nous voulons maximiser ainsi que la taille de la voile que nous cherchons à minimiser :

$$fitness = (coefficient\_ombre * taux\_ombrage) + (coefficient\_aire * (1 - (\frac{area}{maximum\_area}))) \quad (1)$$

avec $coefficient\_ombre = 0.7$ et $coefficient\_aire = 0.3$.

3.4 Opérateurs de sélection

- select_alea : Choisit aléatoirement les parents au sein de la population.
- select_best : Trie la population selon la fitness et choisit les meilleurs en tant que parents.
- select_tourn : Choisit les meilleurs individus en tant que parent au sein d'une sous-partie de la population.

3.5 Opérateurs d'insertion

- insert_replace_worst : Trie la population selon la fitness des individus et remplace le moins bon par l'enfant.
- insert_alea : Remplace un individu choisi aléatoirement par l'enfant.
- insert_age : Trie la population selon l'âge des individus et remplace le plus ancien par l'enfant.
- insert_up : Insère l'enfant à la manière de *insert_replace_worst* si sa fitness est au-dessus d'un certain seuil.

3.6 Opérateurs de mutation

Plusieurs opérateurs de mutation ont été implémentés, nous ne présenterons ici uniquement ceux utiles et efficaces à la résolution.

— mutation_Xcoord : Mutation des coordonnées de X piliers de l'individu choisis aléatoirement puis augmentées ou diminuées sur un intervalle de d centimètres fixés arbitraitement.

— replace_mutation : Mutation remplaçant une coordonnée d'un pilier de l'individu en une valeur choisie totalement aléatoirement.

— local_optimum : Mutation inspirée des méthodes de recherches locales visant à modifier les valeurs d'un individu en le rapprochant de son minimum local.

3.7 Opérateurs de croisement

— cross_mono : Sépare les parents en deux pour chaque pilier, le premier enfant prend la première partie du premier parent et la deuxième partie du second, le deuxième enfant l'inverse.

— cross_uni : Le premier enfant a, pour chaque coordonnée de chaque pilier, une chance sur deux de prendre la valeur de chaque parent. Le deuxième enfant prend le reste.

— cross_aver : L'enfant généré reçoit comme coordonnées la moyenne de chaque coordonnée de chaque pilier des deux parents.

— cross_best : Pour l'axe d'un pilier, l'opérateur regarde en analysant les deux parents, dans quelle direction la fitness est la meilleure. Cette coordonnée pour l'enfant est donc la valeur optimum de la coordonnée de ce pilier dans cette configuration des autres piliers.

4 Expérimentations et résultats

Nous allons étudier ici les différents aspects de l'algorithme génétique, et ce en deux axes. Tout d'abord nous comparerons chaque classe d'opérateur pour en déduire les plus efficaces. Grâce à cette étude nous utiliserons, dans une deuxième partie, les meilleurs opérateurs de chaque classe pour expérimenter l'algorithme sur différentes périodes de l'année (i.e. sur différentes données - ensembles d'azimuts).

4.1 Étude d'efficacité des opérateurs

4.1.1 Protocole expérimental

Pour étudier les efficacités des différents opérateurs, nous devons pouvoir assurer que les conditions d'expérimentations sont les mêmes pour chaque opérateur ; et ce dans chaque classe d'opérateur - i.e. nous voulons des efficacités relatives pour chaque mutation, insertion... et donc, si les conditions d'expérimentations changent d'un opérateur d'insertion à un opérateur de sélection, ce n'est pas un problème. Pour avoir de telles conditions nous pouvons fixer les opérateurs dont la classe n'est pas évaluée. C'est à dire que, si l'on évalue les insertions, alors on les évalue toutes avec toujours les mêmes autres opérateurs. Nous pouvons aussi agréger toutes les combinaisons d'opérateurs en un seul test, et donc avoir l'efficacité relative de l'opérateur en dehors d'un cadre expérimental fermé. C'est cette dernière solution que nous utiliserons. Ainsi nous lancerons $nombre\_de\_selection \times nombre\_de\_crossover \times nombre\_d'insertion \times nombre\_de\_mutation$ instances de l'algorithme, avec chacune en sortie un fichier avec la fitness en fonction du nombre d'itérations. Enfin, pour chaque opérateur, nous pouvons faire la moyenne de la fitness de chaque instance où cet opérateur est utilisé. Ce qui nous donnera l'efficacité de cet opérateur en fonction du nombre d'itérations.

4.1.2 Opérateurs de mutation

Comme nous pouvons le voir sur le graphique Fig. 1, sur cette classe d'opérateur, aucun doute n'est possible. L'opérateur de mutation à base de recherche locale est bien plus efficace que les autres. Cependant cette efficacité vient avec un coût élevé en temps de calcul, en effet cette mutation fait appel à la fonction d'évaluation, qui peut être lente en fonction du nombre d'azimuts. Néanmoins ce coût est peu limitant étant donné le problème que l'on tente de résoudre. En effet ce problème ne doit être résolu qu'une seule fois.

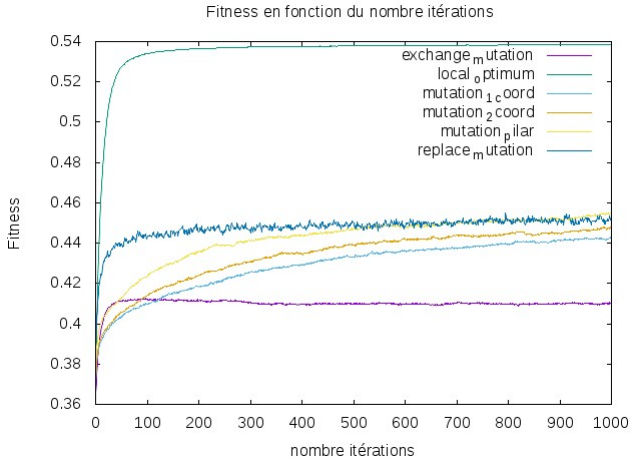

FIGURE 1 – Fitness des opérateurs de mutation en fonction du nombre d'itérations

4.1.3 Opérateurs de sélection

En ce qui concerne les opérateurs de sélection, le meilleur est bien plus difficile à définir. Comme nous pouvons le voir sur le graphique Fig. 2 les courbes des efficacités des opérateurs se chevauchent sans que l'on puisse en isoler une. Ce résultat nous informe que peu importe quel individu la sélection choisie, les opérateurs de crossover et de mutation éloignent trop les individus résultant du crossover pour en déterminer la parenté avec les individus sélectionnés. Au vu des performances de l'opérateur de mutation de recherche locale, nous avons effectué un autre tuning similaire au premier sans l'opérateur de recherche locale, et ce dans l'espoir de pouvoir départager les opérateurs de sélection. Le résultat de cette deuxième expérience est présenté par le graphique Fig. 3. Si l'efficactité relative présentée par ce graphique est faible, elle nous aide tout de même à classer les différents opérateurs. La sélection des meilleurs individus est plus efficace que les autres sélections. De plus la sélection aléatoire est la moins efficace, c'est ce qu'on attendait de cet opérateur.

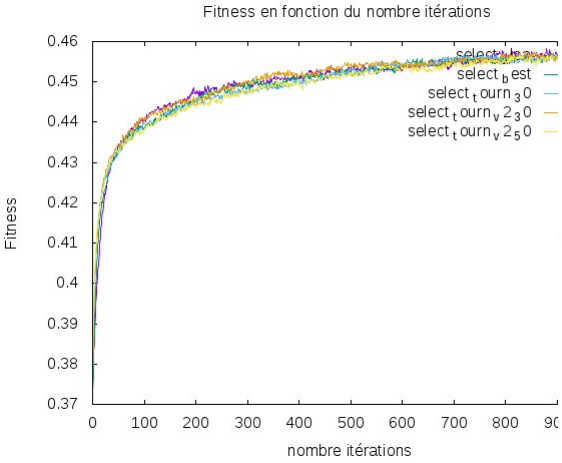

FIGURE 2 – Fitness des opérateurs de sélection en fonction du nombre d'itérations

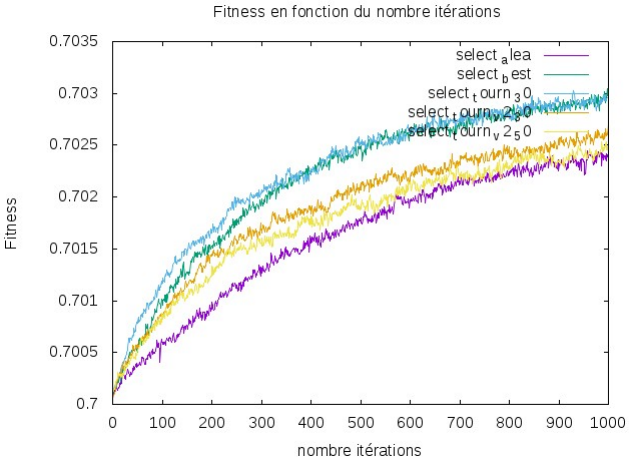

FIGURE 3 – Fitness des opérateurs de sélection en fonction du nombre d'itérations

4.1.4 Opérateurs d'insertion

Le graphique Fig. 4 montre l'efficacité des opérateurs d'insertion. Nous pouvons montrer que les opérateurs *insert_replace_worst* et *insert_up* sont plus efficaces que les deux autres. Et le premier un peu plus efficace que le second.

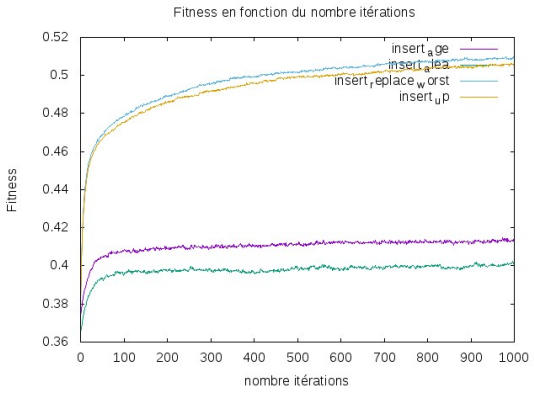

FIGURE 4 – Fitness des opérateurs de mutation en fonction du nombre d'itérations

4.1.5 Opérateurs de crossover

Le graphique Fig. 5 montre l'efficacité des opérateurs de crossover. Il nous montre que l'opérateur de crossover *cross_best* est plus efficace que les autres de la même classe.

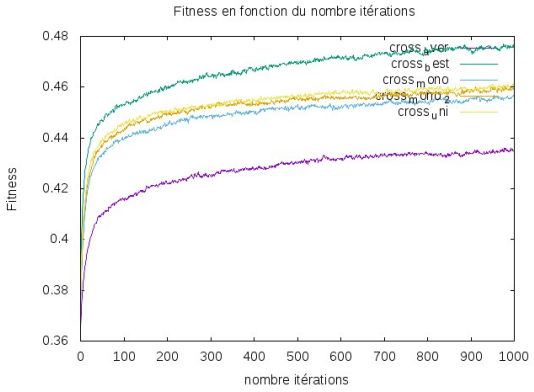

FIGURE 5 – Fitness des opérateurs de mutation en fonction du nombre d'itérations

4.1.6 Conclusion

En conclusion nous utiliserons pour la prochaine étude les opérateurs suivants : la mutation *local_optimum*, la sélection *select_best*, l'insertion *insert_replace_worst* et pour le crossover le *cross_best*.

4.2 Étude de l'algorithme sur différentes périodes de temps

4.2.1 Protocole expérimental

Pour cette étude le protocole est très simple. Tout d'abord les opérateurs seront fixés comme précisé dans la partie précédente. Enfin nous étudierons des instances choisies au préalable pour montrer les différents aspects de l'algorithme.

4.2.2 Fonctionnement sur différentes saisons

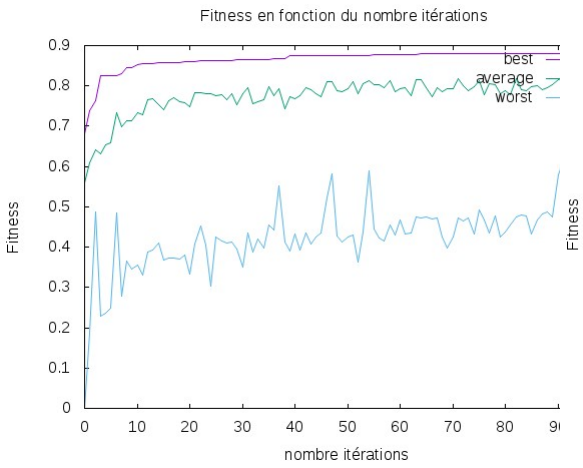

FIGURE 6 – Hiver

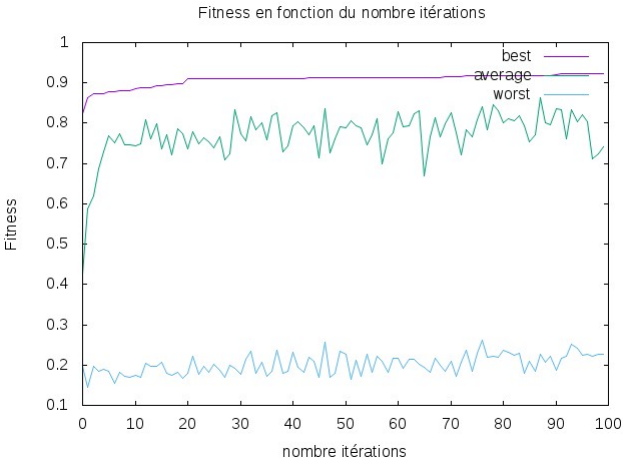

FIGURE 7 – Printemps

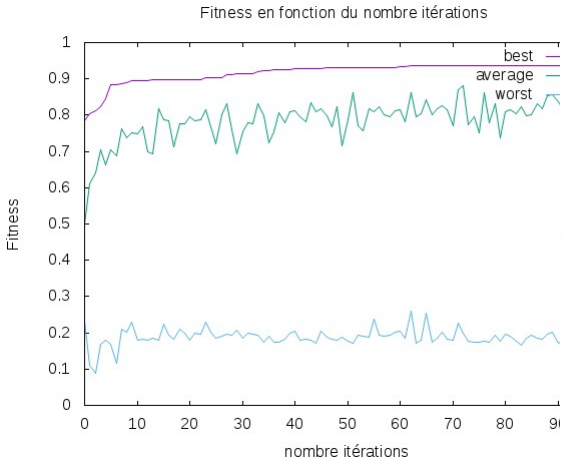

FIGURE 8 – Été

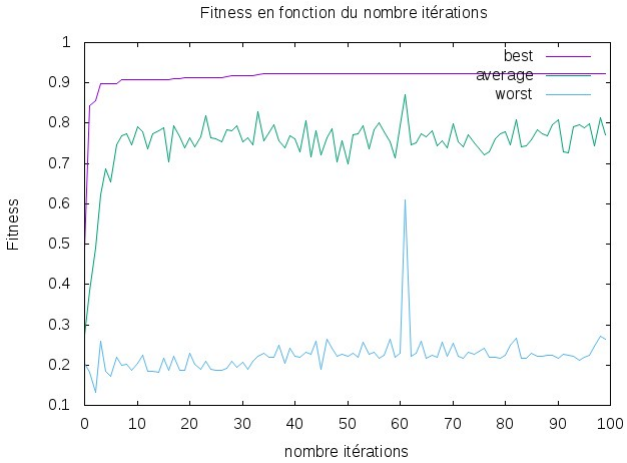

FIGURE 9 – Automne

Les graphiques Fig. 6,7,8 et 9 montrent l'évolution de la fitness en fonction du nombre d'itérations. Si chaque exécution ne performe pas autant que les autres, les résultats sont encourageants. À savoir que la plus petite fitness à l'itération 100 est d'environ 0.85 ce qui reste un score élevé. De plus ce score est atteint pour l'hiver, qui est la saison la plus difficile à couvrir par une voile étant donné les azimuts rasants.

4.2.3 Influence du nombre d'itération maximal

Comme nous pouvons le voir grâce aux graphiques Fig. 10,11 et 12, le nombre d'itérations maximal n'est pas un paramètre qui peut nous faire atteindre de meilleurs résultats. En effet, une fois que l'algorithme ne se trouve que dans des minimums locaux avec une population peu diverse il n'arrive plus à s'améliorer. On peut donc en déduire que non seulement l'initialisation joue un grand rôle pour la suite de l'algorithme et que pour résoudre ce problème il est plus intéressant de lancer l'algorithme plusieurs fois avec peu d'itérations plutôt qu'une seule fois avec un grand nombre d'itérations.

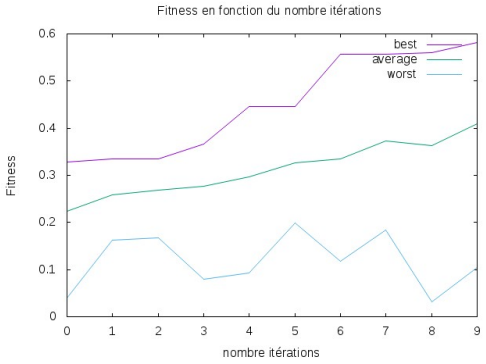

FIGURE 10 – Année complète

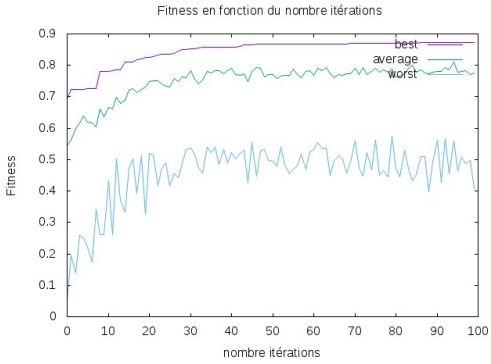

FIGURE 11 – Année complète

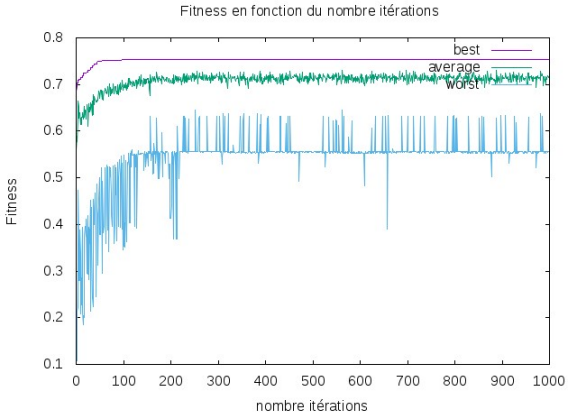

FIGURE 12 – Année complète

5 Interface graphique

De manière à pouvoir visualiser, vérifier et présenter de façon plus claire nos résultats, nous avons décidé d'implémenter une interface graphique à notre application montrant en action le résultat de notre meilleur individu sur la période donnée.

Plusieurs candidats de moteurs graphiques C++ se sont donc offerts à nous, comme Ogre3D ou Irrlicht, Irrlicht étant plus simple d'utilisation mais Ogre étant plus puissant. Nous nous sommes donc penchés sur l'utilisation d'Irrlicht, vu la simplicité de l'interface graphique voulue.

Irrlicht donc, est un moteur graphique 3D temps réel multi-plateforme sous licence libre, permettant son utilisation commerciale. Il utilise entre autres OpenGL comme rendu logiciel et est compatible avec des plateformes comme par exemple Linux, Windows et Mac OS. Étant à la base conçu pour développer des jeux vidéo, il contient énormément de fonctions et d'outils facilitant la programmation et le rendu d'objets en 3D.

Le but étant de ne pas interférer avec l'algorithme génétique de manière à ne pas ralentir les calculs, notre interface graphique intervient à la toute fin de notre programme, servant donc de présentation des résultats de notre meilleur individu. Elle a été programmée "à part" de manière à ne pas interférer avec le reste du code et se lance juste via la fonction *graphic*.

Cette fonction reprend les paramètres de notre algorithme (zone à ombrager, directions du soleil, génome du meilleur individu) et crée dans l'espace un sol, la zone à ombrager, et crée un objet *CSailShadeSceneNode* représentant notre voile avec ses 3 piliers. Cet objet est dérivé de l'objet *ISceneNode* représentant un élément dans l'espace 3D d'Irrlicht.

Par dessus cela, on ajoute une lumière et une projection de l'ombre selon la direction du soleil donnée. Pour montrer le résultat obtenu tout au long de la période, la lumière est déplacée sur une petite animation, pour toutes les directions du soleil données par SolTrack. Enfin, pour bien pouvoir visualiser nos résultats, nous pouvons déplacer et orienter la caméra grâce aux touches ZQSD et la souris.

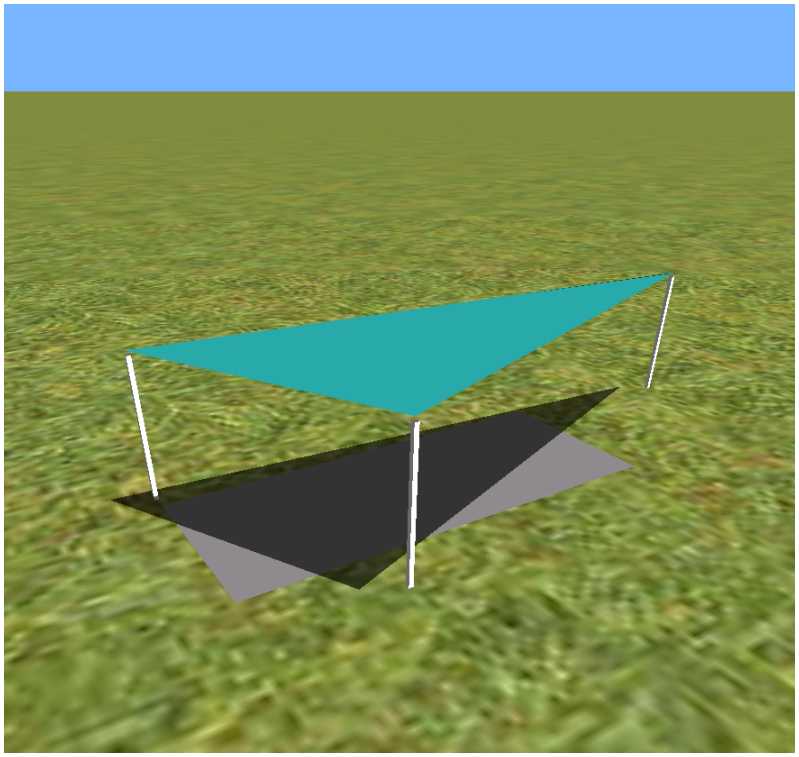

FIGURE 13 – Exemple de résultat observé via l'interface graphique

6 Conclusion

Il a donc été développé un algorithme génétique visant à la résolution du problème de voile d'ombrage. Cette méthode de résolution semble être adaptée pour ce problème, même si des résultats similaires pourraient certainement être obtenus à l'aide uniquement d'une recherche locale, réduisant ainsi le temps d'exécution d'un tel algorithme, ainsi que sa complexité en taille. En revanche, cette méthode ne s'adapterait sûrement qu'à des cas simples de ce problème.

En effet, pour le moment, seuls trois points d'accroches sont modulables et le problème s'en retrouve être certainement trop simplifié pour profiter pleinement des qualités d'un algorithme génétique. Un problème plus complexe pourrait ainsi peut-être répartir une meilleure charge de travail et d'efficacité des différents opérateurs de mutation et de croisement.

Une interface graphique très modulable a aussi été implantée afin de favoriser une certaine facilité à la démonstration de l'efficacité de l'algorithme génétique à résoudre le problème de voile d'ombrage.

L'algorithme peut aussi être amélioré afin de gérer des cas plus complexes du problème de voile d'ombrage. On pourrait ainsi gérer l'ombrage d'un objet fixe, tel un mur par une simple projection supplémentaire comme réalisée avec la voile, ou encore gérer une voile dont les points d'accroches seraient variables et plus nombreux.