Rapport de fin d'études

Optimisation de remplissage de colis.

Développement d'une application capable d'optimiser le remplissage d'un conteneur avec des objets de formes et tailles différentes.

Réalisé au laboratoire Quantup, à Strasbourg

par Laura Mendoza

Responsable de Stage : Maxime Pierson

Année 2011-2012

Master 2 Calcul Scientifique et Sécurité Informatique

Remerciements

Mes remerciements vont tout particulièrement à Maxime Pierson, mon maître de stage pendant ces 6 mois.

Je tiens à remercier Asher Perez et M. Levy pou m'avoir reçue dans leur équipe et m'avoir donnée cette opportunité.

Un grand merci à tous mes professeurs de ces 2 années de Master.

Sommaire

Introduction

Du 6 février au 15 juillet, j'ai effectué mon stage de fin d'études au sein du laboratoire de recherche, Quantup. Pendant ces cinq mois j'ai eu l'opportunité d'être introduite au monde de la recherche et avoir un aperçu du quotidien chercheurs et des ingénieurs spécialisés en différents domaines scientifiques, auprès de qui c'était une honneur (ainsi qu'une opportunité très enrichissante) de travailler.

La recherche de stage de fin d'études, comme toute recherche de stage ou de travail, s'est avérée assez compliquée et de plus du fait que je tenais à trouver un stage qui correspondrai au maximum à mes exigences. En effet tout stage doit, à mon avis, permettre d'élargir ses connaissance et de découvrir le monde du travail, mais les stages de fin d'études déterminent très fortement la vie professionnelle des étudiants. Après l'expérience de première année de Master j'avais déjà compris que même si le domaine de sécurité informatique est très intéressant, je n'aurai jamais pu m'investir totalement dedans du à l'absence totale ou presque des mathématiques. En conséquence il fallait que je trouve un stage qui me permettrai ce que le Master m'a permis : lier les mathématiques avec l'informatique.

C'est ainsi que j'ai trouvé une offre de stage qui parlait d'optimisation, algorithmes et intelligence artificielle. Après l'entretien d'embauche j'étais sûre d'avoir trouvé une opportunité qui me correspondait : un sujet passionnant qui permettait d'appliquer des connaissances mathématiques à l'élaboration d'un programme informatique. En plus j'avais fait la connaissance de Maxime Pierson, qui serai par la suite mon maître de stage, et de M.Perez, professeur de physique et chef de mon équipe au sein de Quantup, deux personnes avec des parcours différents mais très passionnés par leur métier et les sujets qu'ils avaient à me proposer. Le laboratoire, même s'il était en phase de démarrage, semblait être un lieu propice pour un travail dynamique et fructueux.

Afin de présenter au mieux cette période d'apprentissage, il me paraît logique de commencer par une présentation du contexte du sujet. En suite,

j'expliquerai les techniques et outils utilisés dans les problèmes du même type, et plus spécifiquement ceux que j'ai utilisés pour ce problème. Je donnerai un bref résumé des difficultés rencontrées et des solutions abordées pour les résoudre. Et pour finir, une description du prototype résultant ainsi que des éventuels améliorations et pistes pour la continuation du projet.

1 Présentation du sujet

1.1 Contexte du sujet

1.1.1 Logistique

La logistique est l'activité responsable de gérer et planifier les activités de production, transport, stockage, achat et distribution. On va s'intéresser tout particulièrement à la gestion de la chaîne logistique et aux flux de marchandise et d'information. La gestion de chaîne logistique a pour finalité d'optimiser les flux physiques, de la production et la vente, jusqu'à la livraison d'un produit. De nos jours, elle joue un rôle capital dans la compétitivité des prix et elle représente, dans des nombreuses entreprises, des sommes non négligeables.

Regardons plus en détails des exemples des différents problèmes de logistique de l'amont à l'aval de l'achat :

- Fabrication : assemblage, planification, etc.
- Entreposage ou magasinage : des marchandises en grandes quantités doivent être entreposées avant d'arriver à leur destination finale. On doit alors optimiser le temps d'entreposage pour minimiser le coût. D'autres problèmes de rangement spatial peuvent aussi surgir.
- Optimisation des stocks : anticipation des ruptures de stocks, niveaux des stocks d'alerte/de sécurité, temps de réapprovisionnement.
- Transport : usuellement des problèmes du type voyageur de commerce, mais aussi des problèmes de rangement, de suivi des flux, etc.

Actuellement, plusieurs outils et solutions informatiques sont à la disposition. Cependant malgré l'ancienneté de certains de ces problèmes, ils restent des sujets de recherche d'actualité.

1.1.2 Contexte au sein de Quantup

Quantup est un laboratoire de recherche qui se dédie à trouver des solutions informatiques basées sur des connaissances scientifiques aux problèmes de ses clients. Pour ce projet, le client est un éditeur de logiciel de gestion d'entreprise, spécialisé dans les logicielles ERP (logiciel pour la gestion logistique). Une grande majorité de ses clients sont des entreprises du domaine de l'alimentaire, des supermarchés ou hypermarchés. En effet, en ce domaine la logistique joue un rôle très important. Grâce aux nouvelles technologies, une grande partie de la chaîne logistique peut être optimisée en automatisant un maximum des processus.

C'est de cette idée que le besoin d'optimiser une des tâches les plus courantes est surgit. Avec les achats en ligne la plupart de la chaîne logistique est automatisée : gestion des stocks, réapprovisionnements, délais de livraison, etc. Cependant il reste quelques éléments qui n'ont pas du tout été améliorés. Par exemple : l'emballage des produits dans un carton. On peut alors se poser les questions suivantes :

- Est-ce qu'on peut développer un algorithme capable de trouver l'arrangement optimal pour une liste de produits fixée ?
- Est-ce qu'il est possible que l'algorithme prenne en compte des caractéristiques particulières sur les produits ? sur le carton ?
- Peut-on ajouter des contraintes afin que le procédé entier soit effectué par des machines ou pour qu'il y ait un minimum d'intervention humaine ?
- Quel serait le temps de calcul ?
- Quel pourcentage d'espace pourrait être gagné ?

1.2 Enoncé du problème

1.2.1 Caractéristiques du problème

Le problème a été formulé sous plusieurs formes et avec des différentes contraintes. Définissons donc les paramètres élémentaires :

- Objectif : On cherchera à maximiser le nombre d'objets qu'on peut ranger dans une (ou plusieurs) boîte(s).
- Conteneur : Il est de forme rectangulaire, la limite du poids qu'il peut avoir est infinie et il n'y aura aucune restriction d'équilibre de poids. On travaillera avec un nombre de cartons fixés à l'avance et si on travaille avec plus d'un, on assume qu'ils sont tous de mêmes dimensions.
- Objets : Les produits seront de tailles et de formes diverses. On donnera au programme une liste d'objets finie et il n'y aura aucune priorité de rangement. Pour plus de caractéristique sur les objets voir la partie suivante.
- Dimension : Le problème a des versions différentes sur les 3 dimensions (en premier dimension le problème revient à optimiser l'ordre de rangement des objets), mais on ne s'intéresse qu'au 2D et au 3D.
- Contraintes de temps : Le programme devra résoudre les problèmes simples en quelques secondes ou minutes. Si le problème est complexe on pourra accepter un délai un peu plus grande sans dépasser le quart d'heure. En effet la vitesse de calcul est très importante.
- Autres contraintes : on ne prendra pas en compte des contraintes sur les produits (toxicité, fraîcheur, écrasement d'objets, etc.) sur ce papier, par contre il est conseillé de poursuivre les études en incluant un maximum de ces contraintes.

1.2.2 Problèmes C&P

Avec ces conditions, notre problème fait partie des problèmes "Knapsack problem". Plus généralement, c'est un problème de découpage et de pacakaging (*Cutting & Packing Problem* en anglais, ou encore *C&P problems*)

Pour voir plus d'information sur la famille et la typologie des problèmes de la même famille voir la figure suivante (*Figure 1.1*). Une étude plus approfondie sur les différents types de problèmes a été faite en 1997 par E. Hoper and T. Burton[9], de même on peut voir la typologie décrite par Gerhard Wäscher et al. [26] en 2002, qui décrit plus particulièrement les problèmes de packaging. En utilisant la typologie décrite dans ce dernier document, le problème qu'on traite fait partie des problèmes SBSBPP (*Single Bin Size Bin Packing Problem*). On continuera tout de même à étudier les cas similaires pour pouvoir récupérer les bonnes idées de traitement.

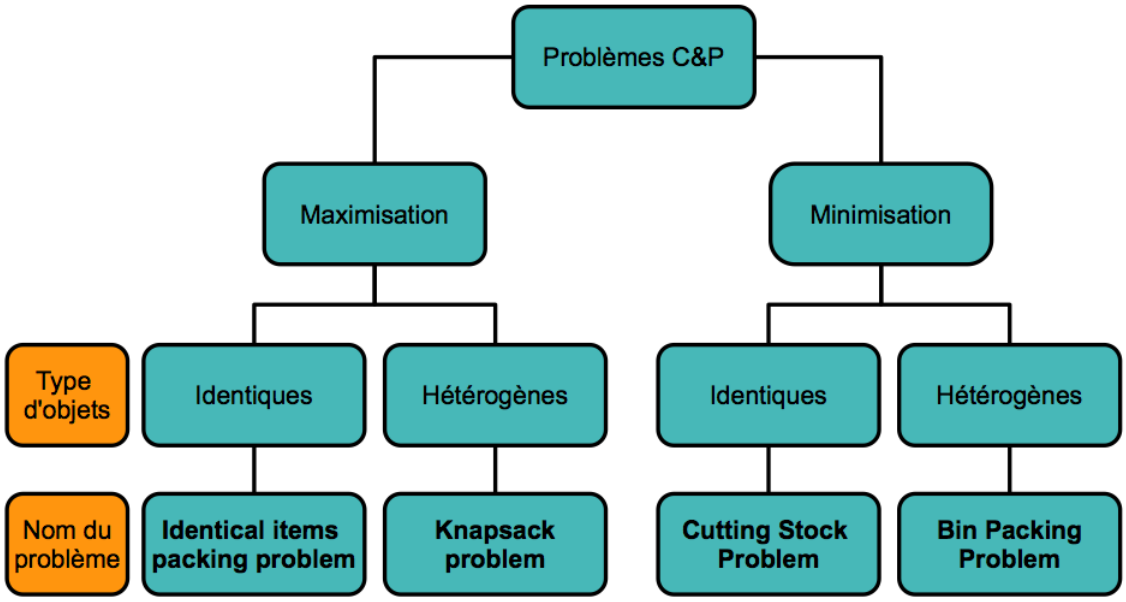


Figure 1.1: Typologie des problème C&P

D'autres applications concrètes des problèmes C&P sont : Découpage de feuilles de métal (problème en 2D), découpage de tissus pour l'industrie de la mode (problème en 2D), arrangement des objets dans une étagère (problème en 3D), emplacement des articles dans un journal (2D), transportation de colis dans un camion (3D), etc. Malheureusement, même si le nombre d'applications est très grand, les solutions comprenant un grand nombre de contraintes sont rares voir inexistantes.

1.2.3 Délimitation et simplification

Pour simplifier l'étude on commencera à réaliser l'étude en deux dimensions. Dans ce cas on peut faire l'analogie avec des problèmes plus connus comme le découpage de feuilles de métal ou découpage de tissus. On traitera également tous les objets comme des polygones (voir la partie suivante 2).

Dès les années 80, grâce à Fowler et al.[7], on sait que ces problèmes sont des problèmes NP-complets. Il est alors impossible de les résoudre en temps polynomiale. En conséquence, des solutions optimales sont sujets de recherche dans le secteur industriel et académique. Une grande majorité de ces recherches sont basées sur des outils d'intelligences artificielle et des heuristique d'apprentissage. Remarquons qu'avant de pouvoir choisir un outils de résolutions, nous devons fixer une modélisation des objets qu'on étudiera. On effet la base pour que toute solution soit efficace c'est que la modélisation soit non seulement fiable mais efficace.

2

Modélisation des objets

Le premier obstacle rencontré par les chercheurs dans les problèmes de remplissage et de découpage c'est la modélisation d'un objet. En effet, l'une des conditions les plus importantes du problème est la non-superposition d'objets. Même si à l'œil humain il est facile de déterminer si deux objets sont séparés, imbriqués ou superposés, définir ces 3 états entre 2 objets peut représenter un vrai défi pour un programme informatique. Avoir une bonne définition de la géométrie des objets est essentiel pour la résolution du problème. On verra par la suite les différentes méthodes utilisées, leurs points fort ainsi que leurs points faibles. Pour plus de détails, on conseille de lire l'étude faite par J.A. Bennell et J.F. Oliveira [3].

2.1 Différents types de modélisation dans la littérature

2.1.1 Pixelisation ou représentation matricielle

Cette méthode est la discrétisation du domaine, c'est à dire la réduction d'informations sur le conteneur par représentation de celui-ci par une matrice. Oliveira et Ferreira (1993) ont proposé la représentation suivante : Les emplacements vides sont codés par 0 et un objet est représenté par 1. Un ajout d'une pièce est donc tout simplement une addition. Si deux pièces se superposent, on peut le détecter très rapidement (une valeur différente de 0 et de 1). Cette modélisation est très intéressante d'un point de vu légèreté et vitesse. Elle permet de modéliser des objets de toute sorte. Cependant la modélisation d'un objet non rectangulaire peut être très grossière et donc pas assez précise et si on veut incrémenter la précision (discrétisation plus fine, c'est à dire dimensions des matrices plus grandes) l'espace mémoire

utilisé ainsi que le temps de calcul seront très vite incrémentés. D'un autre côté, Segenreich et Braga [22], utilisent une méthode très semblable, mais dans ce cas, les bords des pièces sont codés par des 1 et l'intérieur de celles-ci par des 3. Grâce à cette méthode on peut différencier l'emboîtement des objets (deux objets en contact) à la superposition des objets. Malheureusement les autres problèmes restent présents.

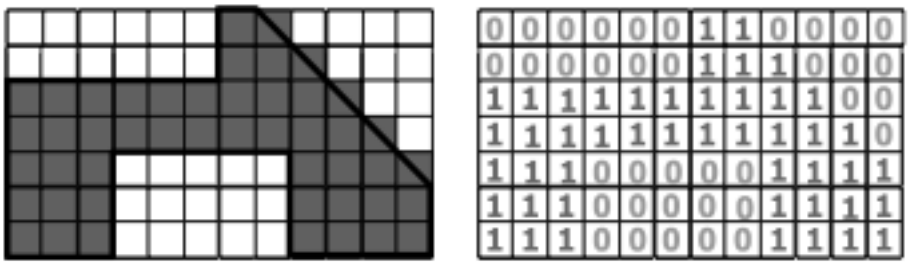


Figure 2.1: Pixélisation (0,1) d'un objet irrégulier.

En 2001, Babu et Babu [2] utilisent une méthode inverse, où seuls les pixels qui sont à l'intérieur du conteneur sont codés avec 0. Les pixels se trouvant aux bords ou à l'extérieur sont codés de droite à gauche, ligne par ligne, commençant par 1 et incrémentant à chaque pixel de 1. Pour les pièces ce sont les pixels aux bords qui sont codés à 0 et on utilise la même méthode pour l'extérieur de la boîte pour l'intérieur de la pièce. Cette méthode se trouve très utile pour des heuristiques [1], et notamment l'heuristique BL (bottom left). Cette méthode est plus adaptée pour les heuristiques mais ne solutionne pas les autres inconvénients des méthodes matricielles.

2.1.2 Trigonométrie et la fonction D.

Cette méthode a pour but de modéliser des objets par des polygones. Cette représentation peut sembler plus propice que la précédente car elle ne dépend pas de la taille de l'objet mais du nombre d'arêtes (ou de sommets). Du côté négatif, la détection de collisions n'est pas immédiate. On peut utiliser les théorèmes et algorithmes déjà connus de la trigonométrie, mais le temps de calcul va être exponentiel par rapport au nombre de côtés

[1] Les heuristiques et nottament l'heuristique BL seront expliquées dans la partie suivante

du polygone (tandis que pour la représentation matricielle le temps était quadratique par rapport à la dimension de la matrice).

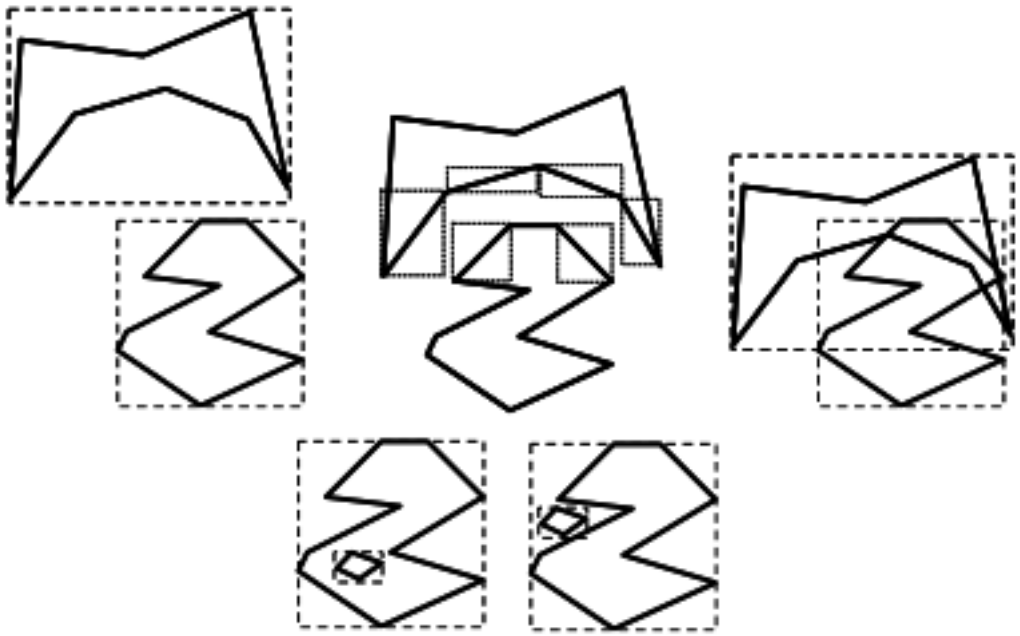


Figure 2.2: Positions relatives différentes entre deux polygones quelconques.

Il existe plusieurs positions possibles entre deux objets (totalement disjointes, disjointes mais les rectangles circonscrits sont superposés, en contact mais sans superposition, croisement de lignes, un polygone dans un autre sans croisement, etc., voir figures 2.2). Si on doit tester une à une chaque option le temps de calcul deviendrai trop important. On peut alors définir tout d'abord des cas plus généraux. Par exemple, on sait que si les pièces se superposent alors les plus petits rectangles qui peuvent contenir les objets se superposent aussi. Il est de même pour les rectangles ayant pour diagonales les segments du polygone. On peut remarquer que les propositions ne sont pas réciproques, mais le temps de calcul en effectuant ces tests pour s'assurer que les pièces ne sont pas disjointes, est diminué dans un nombre de cas non négligeable et souvent par un pourcentage supérieur au 75 %. Ces deux tests restent très simples (niveau implémentation et temps de compilation) mais il faudrait ajouter quelques tests.

Si les tests 1 et 2 ont donné des résultats positifs alors il faut étudier s'il

y a intersection entre les segments qui ont donné le résultat positif au test 2. Pour cela on peut utiliser la fonction D, définie comme il suit :

$$D_{ABP} = ((X_A - X_B)(Y_A - Y_P) - (Y_A - Y_B)(X_A - X_P))$$

La fonction D donne la position relative d'un point par rapport un vecteur orienté $\overrightarrow{AB}$. En effet, elle est strictement positive ($D_{ABP} > 0$) si le point P est à gauche du vecteur, elle vaut 0 si le point P est sur le vecteur, sinon le point est à droite du vecteur. Pour calculer l'intersection de deux segments on doit alors étudier au moins 6 cas différents avec les 4 points délimitant les 2 vecteurs.

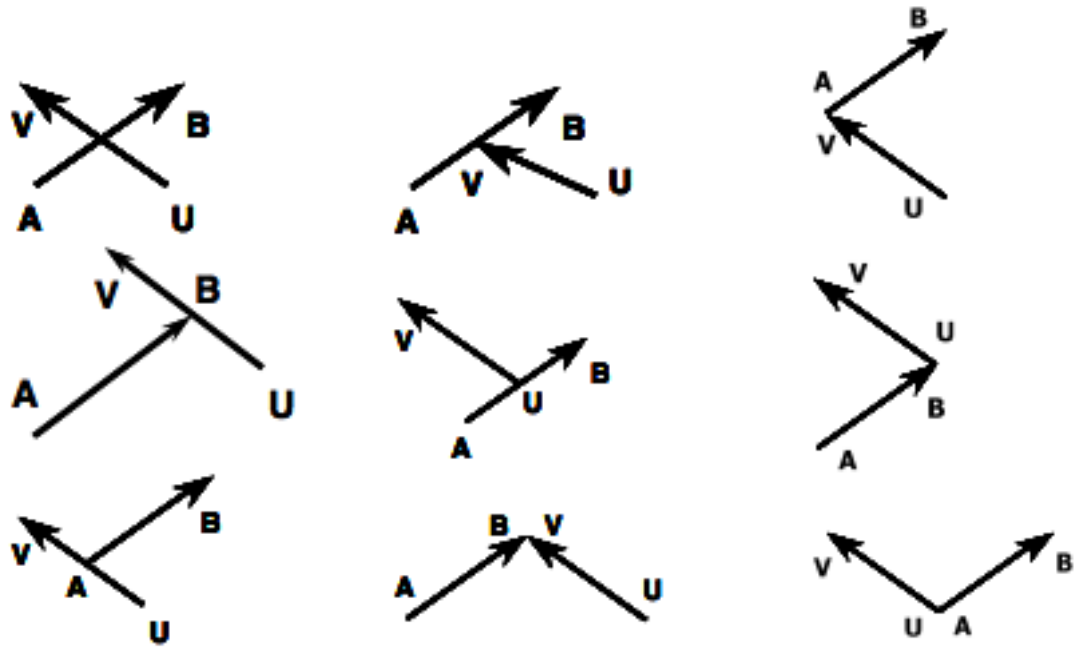


Figure 2.3: Position relative entre 2 vecteurs orientés.

Cette méthode semble très intéressante surtout lorsqu'on travaille avec des polygones (la modélisation est exacte). Néanmoins, on doit travailler avec des nombre décimaux (flottants) ce qui est plus lent et en plus, à chaque fois qu'on fait une modification (nouvelle pièce ajoutée, rotation, translation, etc.), toutes les étapes et fonctions doivent être calculées à nouveau. Cette solution pourrait être choisie pour des algorithmes constructifs basés sur un ordre prédéfini mais est déconseillée pour des recherches itératives.

2.1.3 La méthode du polygone NFP.

La méthode du polygone NFP (nofit polygon) est de plus en plus utilisée dans les articles. En effet, celle-ci est plus efficace que la modélisation par figures géométriques et garde la représentation par segments des polygones. Elle est basée sur la repré-sentation du polygone obtenu par la juxtaposition de 2 pièces et est utilisée pour savoir si deux polygones sont en contact, superposés ou éloignés. En effet, en combinaison avec la fonction D, la complexité des tests serait de l'ordre de $O(n)$ (où n est le nombre de segments du polygone NFP).

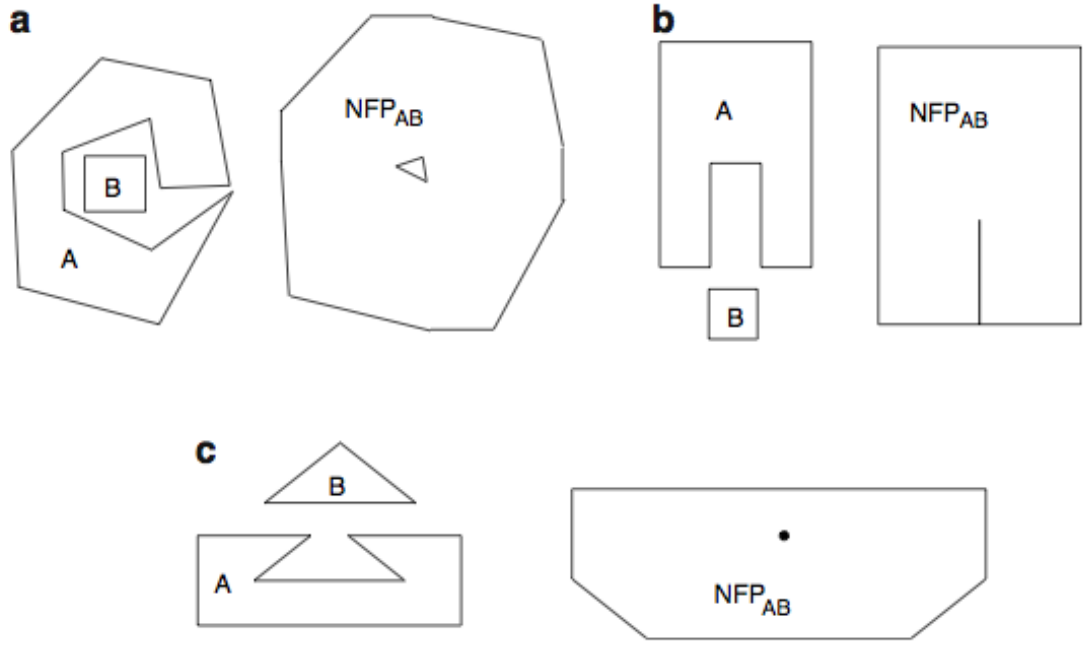


Figure 2.4: NFP obtenu par la combinaison de 2 polygones. (a) Résultat quand la pièce B peut rentrer dans une cavité de A mais ne peut pas glisser dedans. (b) NFP où B peut entrer dans A dans une et une seule direction. (c) Combinaison de 2 polygones où B peut entrer dans une cavité mais ne peut pas se déplacer.

Une des plus grandes difficultés rencontrées lors de cette méthode est lors du traitement des polygones concaves. Dans le cas où une ou les deux pièces sont concaves, le NFP résultant pourrait avoir des segments ou des points à l'intérieur, et les méthodes pour le déterminer ne sont pas très nombreuses (ni simples).

Dans tous les cas, le calcul du NFP n'est pas une tâche triviale et c'est une méthode qui n'a pas de nos jours un software déjà développé pour le traitement des données. Le travail doit être donc fait dès la base et ceci freine les chercheurs. Entre les travaux déjà réalisés il faut savoir qu'ils présentent tous quelques limites, que ça soit le non-traitement de formes concaves ou une complexité élevée dû au nombre de boucles récursives.

D'un autre côté, cette méthode nous permet d'avoir une très bonne représentation des objets et de travailler avec tout type de polygones. Les travaux plus avancés nous permettent même de travailler avec des pièces "à trou" et sont capables de trouver des solutions où ces trous seraient remplis par d'autres pièces. Et une fois que les fonction plus grandes soient définies le temps de calcul peut être très court.

2.1.4 La fonction Phi (Φ-function).

C'est la méthode la plus récente traitant la modélisation d'objets. Son objectif est de présenter toutes les positions relatives entre deux objets. Les premiers articles traitant la modélisation pour les problème C & P grâce à la fonction Φ ont été écrits par Stoyan et al. en 2001 et 2004 [22, 23]. Cependant à la actualité aucun algorithme a été développé pour appliquer la fonction Phi à toutes formes irrégulières.

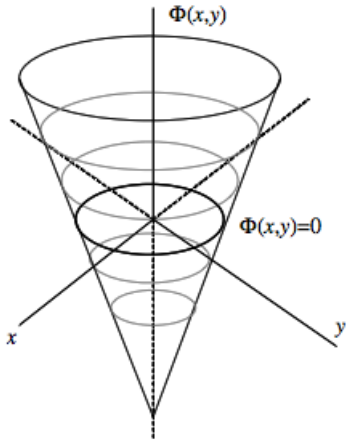


Figure 2.5: Affichage de la fonction Φ pour deux cercles.

2.2 Choix finale pour la modélisation.

2.2.1 Sous forme théorique

Après cette analyse, on a choisit de prendre dans un premier abord la modélisation par des polygones[2]. En effet même si cette méthode n'est pas la plus efficace en terme de temps elle nous permettra d'avoir une structure de base pour le problème. On pourra effectuer quelques tests sur les algorithmes de résolution et les adapter pour obtenir des résultats optimales.

Néanmoins, on gardera toujours l'objectif de passer à une représentation plus complexe, comme la représentation NFP qui est totalement adaptée au problème. La modélisation par NFP reste très récente et continu à être préférée par la majorité des chercheurs car elle reste plus précise pour la modélisation d'objets et plus légère au niveau complexité. De plus on pourrait toujours essayer de faire le pas suivant qui serait la modélisation par la fonction Phi, même si cette approche semble être encore pas assez développée de nos jours pour pouvoir l'appliquer à notre domaine.

2.2.2 Modélisation sous Java

Pour commencer il faut définir la modélisation des objets. Problèmes niveau Java: la classe polygone ne travaille pas avec des points à précision décimale. On doit alors ré-implémenter la classe Point (simplification accès aux points, avant c'était *Point2D.Double* maintenant c'est *Point*). On définit les méthodes d'accès et les modificateurs des champs, ainsi que les opérations basiques sur des points : addition, soustraction, multiplication par scalaire et une méthode permettant de tester si deux points sont égaux ou pas.

On passe à la création d'une classe pour les polygones : *Polygon*, qui hérite de la classe Liste de java (*ArrayList*), et est une liste de points. On n'a pas besoin de redéfinir les opérations de base (accès et modificateurs) vu qu'on a hérité d'une classe. Par contre on doit définir toutes les fonctions mathématiques utiles pour le problème. On commence par chercher des

[2]Voir la sous-partie "Trigonométrie et la fonction D".

formules pour le centre et l'air d'un polygone quelconque (convexe ou pas).

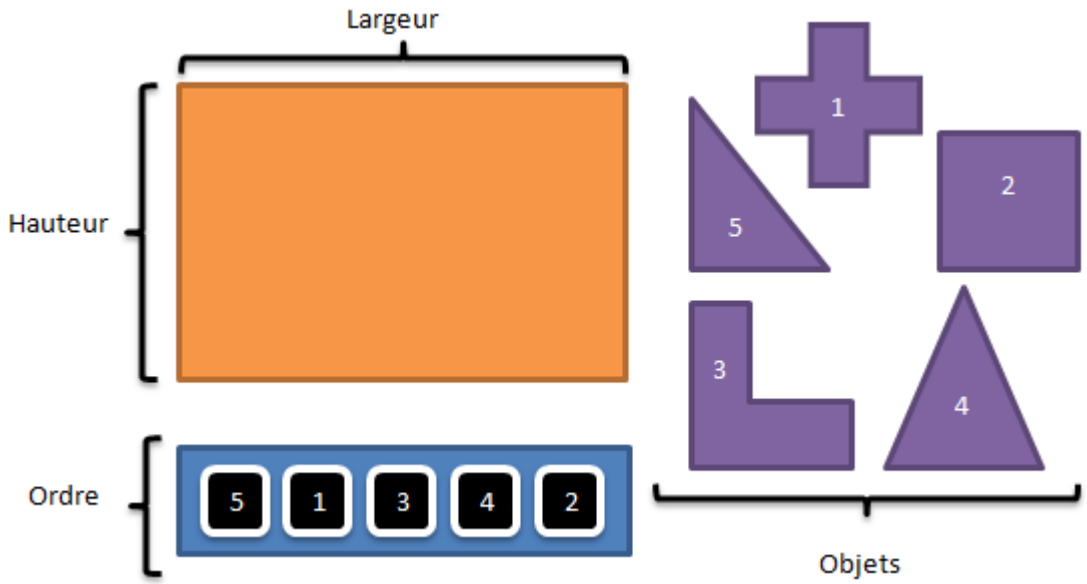


Figure 2.6: Modélisation du problème. Objets et cartons.

On aura souvent besoin du rectangle (*Rectangle*) qui délimite le polygone, on doit alors créer une fonction pour cela (ainsi que redéfinir la classe rectangle). De plus on définit les opérations géométriques élémentaires sur un polygone : homothétie et rotation (par rapport au centre ou par rapport à un point quelconque). Pour les opérations de déplacement on définit des fonctions de translations et de déplacement. Finalement pour les opérations les plus difficiles on a les calculs d'intersection et de soustraction de deux polygones (qui doivent renvoyer le polygone résultant de l'inter/soustraction), on utilise des opérations prédéfinies sur java. On note qu'on ne peut pas utiliser directement ses opérateurs, ils nécessitent notamment d'une perte de précision (double -> int).

Pour finir on veut créer une méthode qui permet de savoir si un point est dans le polygone. Mon premier réflexe est d'utiliser la fonction D, qui permet de savoir la position relative d'un point par rapport à une droite, elle nécessite donc de la création d'une classe *Ligne*, mais cette classe sera tout de même définie car elle sera utile pour d'autres calculs. L'astuce de cette première approche est de supposer qu'un point est dans un polygone

si et seulement si le point se trouve toujours du même côté par rapport à toutes les vecteurs du polygone. Cette méthode marche mais seulement avec les polygones convexes.

On cherche alors une solution plus globale. Je décide alors d'utiliser une méthode vue dans le cours de Graphes de M1 : on trace deux demi-droites horizontales (ou verticales) une infinie vers la gauche et l'autre vers la droite (ou une vers le haut et l'autre vers le bas), toutes les deux ayant pour point de départ le point P et on calcule le nombre d'intersections entre les droite et les segments du polygone. Les nombres d'intersections des deux côtés doivent être impairs si le point est dans le polygone. Cette fonction reste vraie quelle que soit la forme du polygone.

On a vu qu'on aura besoin d'une classe ligne, *Line* qui sera définie de deux points : P_1 et P_2. Si cela est utile on utilisera cette classe comme un vecteur mathématique avec la direction $\overrightarrow{P_1P_2}$. On aura les fonctions attendues comme : coefficient directeur, ordonnée à l'origine, produit scalaire entre deux vecteurs, et la fonction D. Comme une ligne peut être une droite infinie ou un vecteur (de longueur fixée) on va différencier les différents types d'appartenance d'un point à une ligne, ou l'intersection de deux lignes.

Une autre classe qu'on a déjà mentionné mais pas définie est la classe pour les rectangles, *Rect*. Elle va être notamment utile pour définir un conteneur. Elle va être une extension de la classe *Rect2D.Double*, classe java qui présente beaucoup de lacunes pour notre travaille. J'ai hésité à la faire hériter des polygones (un rectangle est un polygone finalement), mais je pense qu'on n'a pas besoin de toutes les différentes méthodes qu'on a défini pour les polygones, une classe plus légère semble plus cohérent. Comme fonction importantes on aura les méthodes qui permettent d'accéder aux différents points du rectangle.

Liste des objets : J'avais initialement voulu rendre la classe conteneur (qui contient une liste d'objets), en définissant appart une classe de liste d'objets avec tous les opérandes sur cette classe qui ne nécessiteraient pas de renseignements sur le conteneur. Mais finalement cette classe ne con-

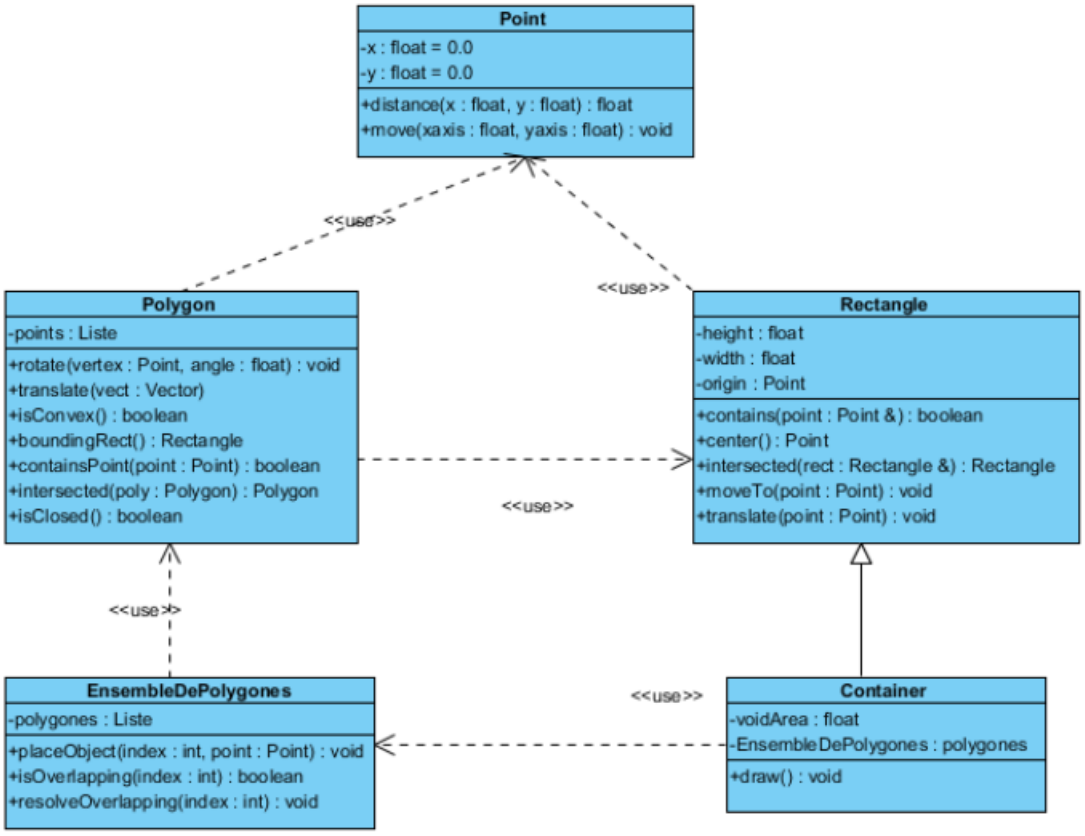


Figure 2.7: Diagramme de classe pour la géométrie du problème.

tenait aucun élément propre à elle et semblait donc inutile et banale par rapport aux autres classes. Maxime m'a donc conseillé de fusionner les classes de conteneur et de liste d'objets.

Un conteneur, ou Container, comme on a vu contient une liste d'objet et est un rectangle. Il possède aussi d'autres attributs tels que l'espace vide qui lui reste à disposition et le nombre de polygones qui sont effectivement dans la boite. La taille d'un conteneur est la taille de la liste d'objets et toutes les méthodes d'accès ou de modification sans indication (comme add, set, remove, etc), s'appliquent directement à la liste d'objets.

3
Algorithmes de résolution

Dès que le problème a été confirmé NP-complet, les chercheurs se sont intéressés à trouver la solution la plus optimale. Par conséquent, nombreux algorithmes et heuristiques ont été développés et/ou adaptés pour pouvoir trouver une solution optimale dans un temps de calcul acceptable.

3.1 Heuristiques.

Du au fait que le problème est NP complet, on sait qu'il est impossible de trouver une solution universelle qui marcherait quelles que soient les données initiales. On peut alors écarter toutes les algorithmes déterministes (*si... alors... sinon...*) et chercher dans le domaines des heuristiques. H. Terashima-Marìn et al. [25] ont fait un recensement des différentes heuristiques utilisées dans les problèmes de découpage et de packaging. Ils définissent également des hyper-heuristiques qui sont des heuristiques combinées ou modifiées de telle sorte que l'heuristique finale soit plus performante. On pourra utiliser les heuristiques pour différents aspects du problème.

3.1.1 Heuristiques de sélection

Par exemple, si on a un nombre d'objets définis à placer dans un nombre de boîtes infini, pour savoir dans quelle boîte on placera un objet donné on utilise des heuristiques de sélection, dont les plus connues sont :

- First Fit : On essaye de placer l'objet dans les conteneurs un à un (dans un ordre fixé) et dès que l'objet peut être placé (s'il reste assez de place dans le conteneur), on le range dans la boîte.

- First Fit Decreasing/Increasing : On ordonne les pièces en ordre décroissant (ou croissant) et on utilise l'algorithme de FF.
- Best Fit : Positionnement de chaque objet de telle sorte que l'espace laissé 'libre' dans le conteneur soit minimal.
- Next Fit : On place l'objet suivant à positionner dans le même conteneur que pour l'objet courant s'il y a assez de place pour le placer, sinon on utilise un nouveau conteneur.

L'utilisation de ces heuristiques reste conseillée pour des problèmes simples car elles sont très simples à implémenter, légères en complexité et les résultats sont très vite améliorés. En ce qui concerne les heuristiques de sélection, on optera surement par une heuristique combinée ou encore mieux par un algorithme ACO (qu'on verra plus en détails plus bas dans le document).

3.1.2 Heuristique de placement

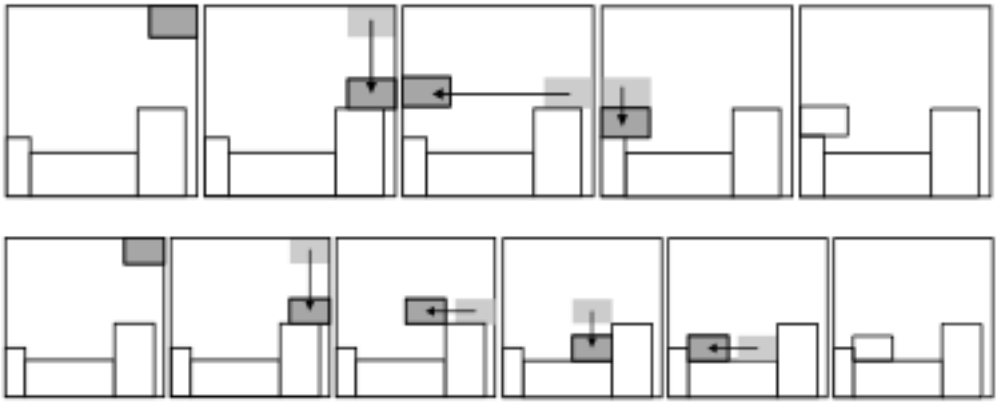


Figure 3.1: Heuristique Bottom-Left comparée à l'heuristique Better Bottom-Left.

Pour le placement, les heuristique *Bottom-Left* (placer un objet le plus bas et le plus à gauche possible) sont souvent utilisés. L'amélioration la plus connue et utilisée de cette heuristique est : *Bottom-Left-Fill* qui consiste à faire le même travail tout en considérant les espaces vides créés entre les pièces déjà placées comme un espace potentiel pour ranger l'objet. Une étude sur les heuristiques de placement a été faite par Bruke, Kendall et Whitwell dans [4]. Également on peut citer Lodi et al. [18], qui ont fait

une étude très complète à ce sujet en 2002. On remarquera malheureusement que la majorité de ces heuristiques restent très limitées. En effet, elles n'autorisent pas les rotations des objets et ont été développées spécialement pour le rangement de formes rectangulaires.

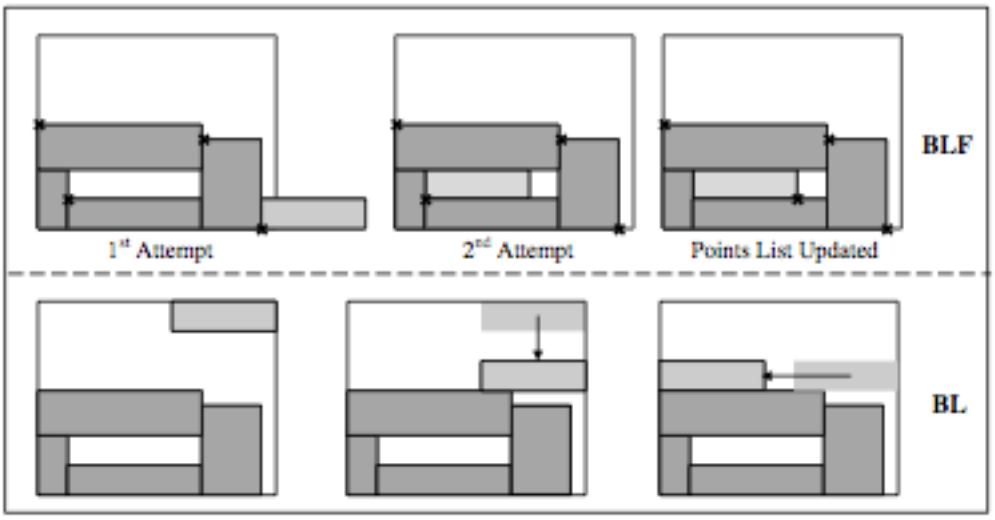


Figure 3.2: Heuristique Bottom-Left comparée à l'heuristique Bottom-Left Fill.

Plus récemment (en 2010), Stephen Leung et Defu Zhang ont utilisé [16] deux nouvelles heuristiques pour résoudre le problème du découpage. Leur approche est de résoudre le problème en "couches", en supposant qu'il y a un nombre fini et fixé de rectangles à découper et qu'on possède une feuille de longueur infinie. Le problème consiste donc à minimiser l'air utilisé. Malheureusement, cette approche ce limite avec des rectangles et n'autorise que des positions verticales ou horizontales pour les rectangles. Mais l'idée peut être intéressante si on travaille avec différentes boîtes, dans ce cas, chaque couche (ou chaque n couches) pourrait correspondre à une boîte.

Julia A. Bennell et al.[8] présentent en 2012 une heuristique en une étape (One-stage approach) qui consiste à ajouter des pièces itérativement à une solution partielle de telle sorte à que le ratio d'air utilisé dans la boîte soit maximale tout en respectant un seuil de tolérance. L'heuristique utilise un arbre de recherche avec différentes solutions en parallèle. Si jamais le seuil d'intolérance est trop important et aucune des pièces restantes peut

être ajoutée, le seuil est diminué (il se peut que le seuil soit nul à la fin). Dans le même document, ils présentent aussi une heuristique en 2 étapes. Celle-ci rassemble une ou deux pièces dans un rectangle et puis ordonne les rectangles de forme optimal. La deuxième heuristique traite le problème de sorte que la solution finale puisse être découpée par une guillotine. Cette condition n'est pas importante pour notre problème, on s'intéressera donc à la première approche qui est en plus, plus rapide.

3.2 Algorithmes génétiques.

En 1975, John Holland introduit un modèle sur les algorithmes génétiques. Des nombreuses applications ont été étudiées depuis cette date et les études sur l'appli-cation aux problèmes de packaging ne sont pas rares. Une bonne introduction à l'utilisation d'algorithmes génétiques pour l'optimisation est donnée dans [27]. Un des premiers documents utilisant des algorithmes génétiques pour la résolution des problèmes C & P (Cutting and Packing Problems) a été publié en 1994 par Shian-Miin Hwang et al. [10]. Celui-ci présentait une technique par arbres de découpage et des notations polonaises inverses. Cependant les recherches se limitaient à des formes rectangulaires et les résultat n'étaient pas totalement optimales. En 1996, le premier algorithme génétique utilisé avec des polygones a été développé par S. Jakobs[14]. Il représente le problème comme la séquence des objets dans l'ordre de rangement. Cette représentation est encore utilisée dans plusieurs documents grâce à la simplicité de représentation et à la facilité pour les croisements entre chromosomes. Le choix des fonctions de croisement et de mutations restaient encore à améliorer.

En 2004, J. Martens [19] présente deux algorithmes génétiques permettant d'appro-cher le problème. Les deux algorithmes diffèrent sur le choix de la modélisation utilisée et le but de l'étude est de trouver quelle méthode est la plus performante, à la fin J. Martens conclue que la représentation non linéaire d'entiers (NLIP) a des meilleurs résultats. En 2003, A. Pasha [20] s'inspire du jeux 'tetris' pour incorporer l'équilibre des pièces dans l'algorithme. Sa solution n'est pas la plus optimale (les espaces libres

laissés par les premières pièces ne sont jamais remplis) mais la solution est stable dans un plan vertical.

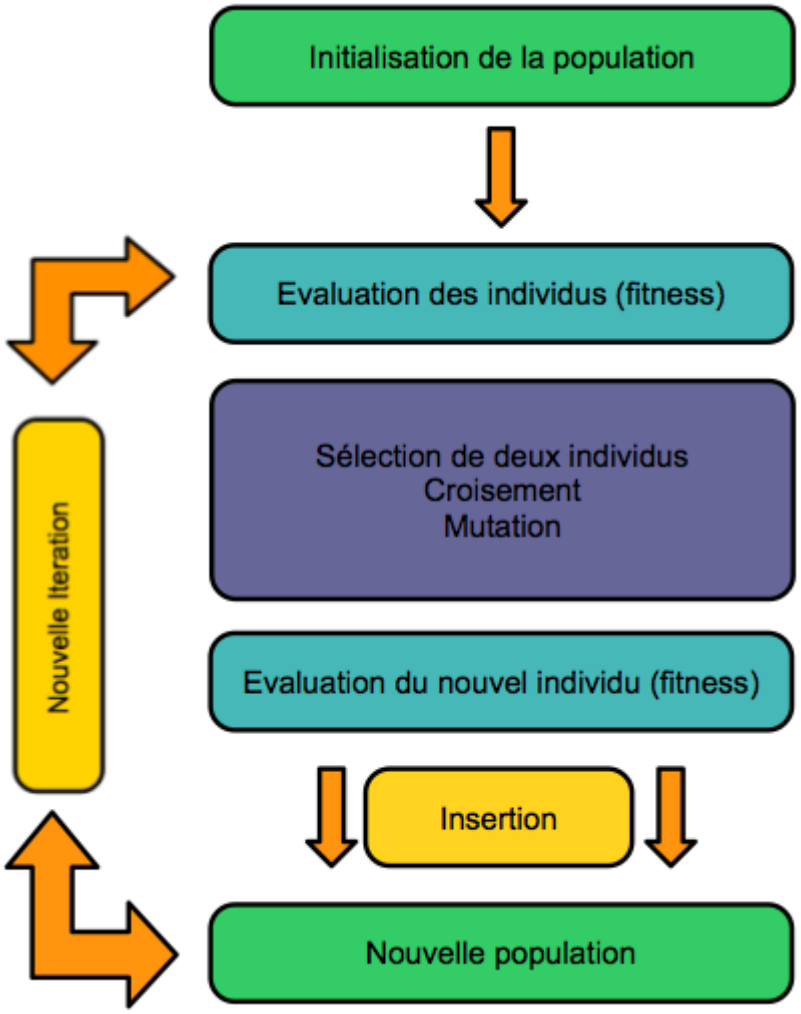


Figure 3.3: Schéma d'un algorithme génétique.

Les algorithmes génétiques sont de plus en plus associés à d'autres heuristiques afin d'augmenter leur efficacité. Par exemple dans [4] on peut voir que l'association de différentes algorithmes peut être recommandable mais pas dans tous les cas. Il est important de savoir que fusionner deux méthodes n'est pas toujours synonyme du fusionnement de 2 bons résultats, souvent les limitations et défauts sont aussi cumulés.

S'il existe une aussi large variété de méthodes différentes utilisant un

algorithme génétique c'est aussi par le choix des différents critères (choix pour la modélisation, choix d'une fonction de croisement, d'une fonction de mutation et d'une fonction permettant calculer la performance (ou fitness) d'une population). Il faut savoir que les opérateurs de modification (croisement et mutation) dépendent entièrement des choix qu'on fait pour les critères.

Intéressons-nous à la fonction de croisement, le premier croisement a été défini par Cohoon et Paris, en 1986[5]. Il choisissait au hasard une portion rectangulaire du génome d'un des parents et le combinait avec celui de l'autre parent. On peut aussi citer les croisements Z3, qui choisit un numéro spécifique de zones de croisement et copie alternativement d'un des deux parents. D'un autre côté les croisements géographiques définis par A. B. Kahng et B. R. Moon[15] sont un type de croisement à eux même, et suivent aussi le principe de découpage aléatoire et interchangeabilité de données. Les résultats de ce type de croisement sont assez bons pour les algorithmes hybrides comme pour les non-hybrides.

3.3 Autres algorithmes utilisés.

Même si plus rares, d'autres études ont pris des approches différentes aux problèmes C & P. On pourra noter tout d'abord l'utilisation des algorithmes de recuit simulé [21] qui apparaissent dans des nombreuses recherches, mais très rarement dissociés d'un algorithme génétique ou évolutif. On peut citer comme exemple les documents de : S. Szykman et J. Cagan [24] (qui l'appliquent au problème en 3D, mais n'utilisent pas d'algorithme génétique), Türkay Dereli et Gülesin Sena Das[6] formulent une solution en 3D qui est assez simple mais n'est pas la solution optimale. Un gros problème de ce dernier document est qu'il ne prend que des parallélépipèdes rectangles.

En 2D, on peut citer Ahonen et al.[1] qui associe à un algorithme de recuit simulé, un algorithme génétique. A chaque génération, une fois l'ordre de pièces défini, on les place dans cet ordre là mais aléatoirement (angle

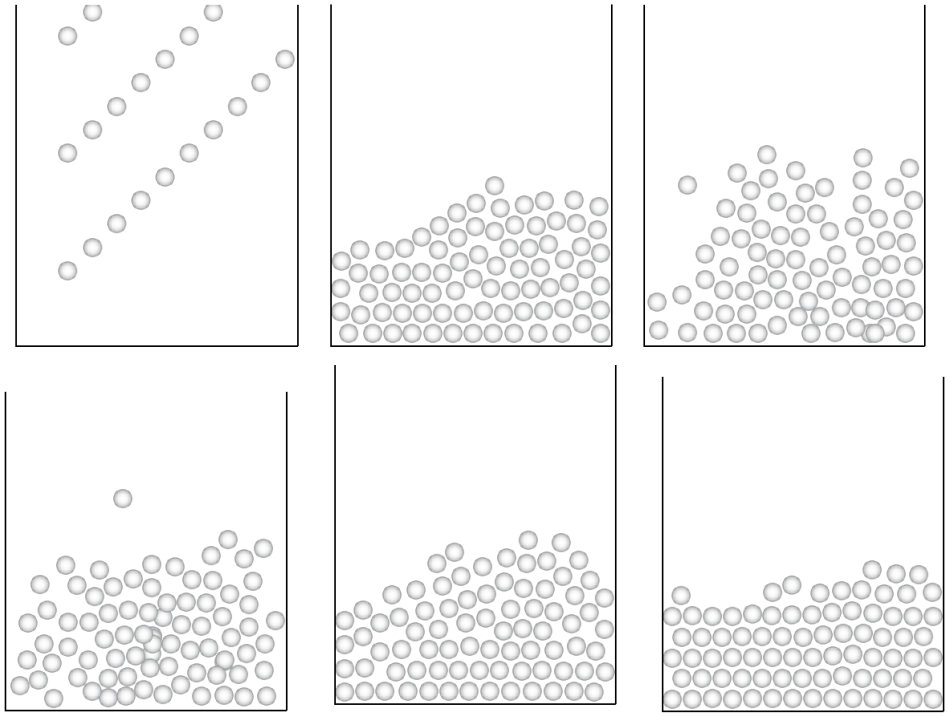


Figure 3.4: Séquence d'une simulation de recuit simulé sur un ensemble de billes dans un récipient. La première rangée est l'initialisation et réchauffement tandis que la deuxième est le moment de refroidissement.

quelconque), puis on effectue l'algorithme de recuit simulé. Pour les premières générations la température sera plutôt élevée et les pièces auront plus la possibilité de se déplacer. Plus on atteindra la solution optimale et moins on fera des changements aux solutions. Cette approche semble être la plus naturelle et simple de celles qu'on a étudié.

Différentes algorithmes de séparation ont été proposés par : Imamichi et al. [11, 12, 13]. Mais le principe reste toujours le même : on commence avec une configuration des objets qui ne répond pas aux critères du problème et l'objectif de l'algorithme est de minimiser le non-respect de ces règles. Les règles que la configuration finale doit respecter vont être modélisées par une fonction de pénalisation. Ces algorithmes sont notamment intéressants pour les problèmes qui traitent différentes contraintes, ils autorisent aussi les rotations de degré arbitraire et tant pour les objets comme pour le conteneur, la forme peut être quelconque.

Pour finir, on citera les algorithmes d'intelligence distribuée (ou *Swarm Intelligence*) tels que les ACO (Ant Colony Optimization). Bien que ces algorithmes ne sont généralement pas associés aux problèmes de C & P, mais plutôt à des problèmes d'optimisation de chemin (le probmème du vendeur, GPS), de routage de réseau ou d'emploi du temps, on a trouvé une application intéressante par Levine et al. en 2004[17], qui s'intéressent à notre problème mais en 1D. On peut alors adapter ces méthodes pour notre problème en 2D ou 3D. En effet, quelque soit l'algorithme d'optimisation choisi, il faut trouver un algorithme de sélection (quels objets vont être rangés dans quelles boîtes).

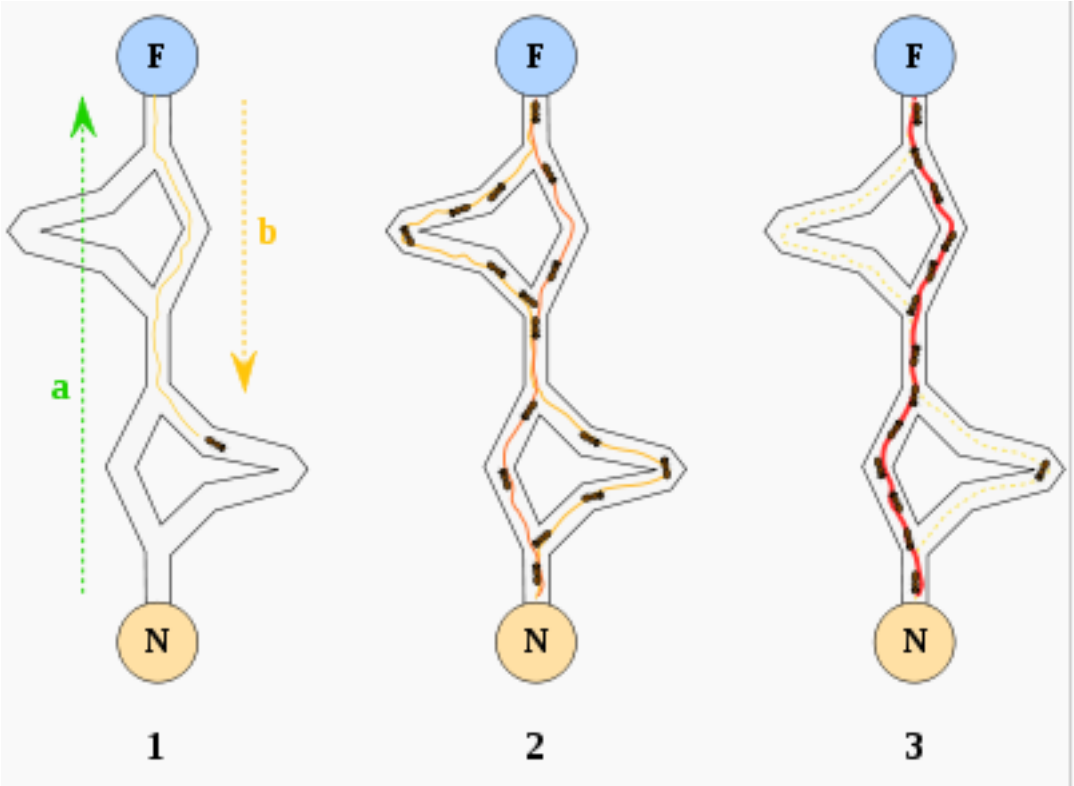


Figure 3.5: 1) la première fourmi trouve la source de nourriture (F), via un chemin quelconque (a), puis revient au nid (N) en laissant derrière elle une piste de phéromone (b). 2) les fourmis empruntent indifféremment les quatre chemins possibles, mais le renforcement de la piste rend plus attractif le chemin le plus court. 3) les fourmis empruntent le chemin le plus court, les portions longues des autres chemins perdent leur piste de phéromones. *(Source : Wikipédia)*

3.4 Choix des algorithmes.

Il semble évident, que comme la plupart des autres études on base notre optimisation dans un algorithme génétique. En effet, ces algorithmes permettent de résoudre tous les problèmes présentés tant que toutes les définitions soient bien posées. Cette une solution évolutive et qui s'adaptera bien à chaque étape du traitement du problème. Le temps de calcul reste acceptable et un AG bien défini donne à chaque génération des résultats différents (ça évite les boucles infinies que rencontrent quelques heuristiques ou des simples algorithmes itératifs). On choisit aussi de modéliser les gènes par la suite (ordonnée) des objets à ranger. Si on retrouvait une autre modélisation plus adaptée à notre problème mais que tous les autres éléments de l'algorithme sont bien fixés, alors le changement sera assez simple.

Jusqu'ici on sait qu'on va commencer avec une population de solutions initiales aléatoires, et qu'à chaque génération on fera des croisements (on pourra utiliser les croisements géographiques) sur les gènes. Cependant comme ces derniers ne représentent que l'ordre de rangement des pièces on peut envisager d'ajouter un algorithme de recuit simulé pour optimiser l'espace occupé par chaque pièce.

Il sera judicieux de commencer le travail en supposant qu'on a un seul conteneur et un nombre fini d'objets. Le but sera alors d'optimiser l'espace occupé par les pièces dans la boîte. Une fois qu'on trouvera une solution acceptable, on pourra ajouter l'algorithme ACO et on cherchera alors à savoir quel est le nombre minimal de boîtes à utiliser pour les objets à ranger.

Remarque : Le choix du langage de programmation (Java), a été fait comme un choix pour le laboratoire Quantup. En effet Java est le langage "universel" qui permet aisément de faire passer une application entre différentes plates-formes. En plus, c'est le langage qui a été utilisé par défaut dans toutes les autres applications Quantup.

4
Evolution du travail

4.1 Analyse de l'application

Dans cette partie on s'intéresse à avoir un meilleur aperçu de la solution à développer pour cela on va utiliser un analyse UML déjà effectué auparavant qui contient les parties suivantes :

- Identification des acteurs
- Recensement des cas d'utilisation
  - Modélisation de nouveaux objets
  - Décomposition analytique d'un fichier
  - Modification des paramètres
  - Visualisation des derniers résultats
  - Optimisation du problème
- Diagramme de classes
  - Package Géométrie
  - Package Algorithmes Génétiques
  - Relation entre différents package
- Conclusion

Voici le document en question :

Optimisation de remplissage de colis avec des objets de formes différentes

Document d'analyse avec U.M.L

Introduction

Dans ce document on cherche à avoir un meilleur aperçu de la solution du problème de packaging. On s'intéresse tout particulièrement aux fonctionnalités principales de la solution. Pour cela, on devra commencer par définir les acteurs ainsi que les différents cas d'utilisation. On étudiera ces derniers plus en détails, en décrivant pour chacun l'objectif principale, les acteurs, le séquencement nominal ainsi que les différents exceptions. On s'aidera des diagrammes UML pour décrire les différents cas. En effet, un diagramme de cas d'utilisation pourra donner un premier aperçu globale et par la suite on pourra utiliser les diagrammes de séquences qui permettront de définir pas par pas l'enchaînement nominal tout en montrant les entités nécessaires (classes, fonctions, etc.) pour la réalisation de chaque cas. Il semble alors évident que pour finir, on s'intéressera à l'architecture de la solution, en décrivant les différents packagings de classes, les classes et leurs méthodes.

1. Identification des acteurs

On doit commencer l'analyse du problème par identifier les différents acteurs qui interagiront avec le programme. Comme dans la plupart des cas, l'utilisateur est l'acteur principal. Dans ce cas, on a choisi de ne pas faire de différence entre les utilisateurs. En effet, le programme ne traite pas des informations confidentielles ni critiques, il n'est pas alors nécessaire d'ajouter des administrateurs. Ceci dit, toutes les personnes utilisant le programme auront les mêmes droits et pourront effectuer les mêmes changements.

En tant qu'acteur –cette fois-ci, secondaire– on doit ajouter aussi les données car celles-ci ont un impact dans les algorithmes et plus généralement, la définition du problème. On verra aussi que celles-ci peuvent être modifiées par l'utilisateur.

On obtient la grille suivante :

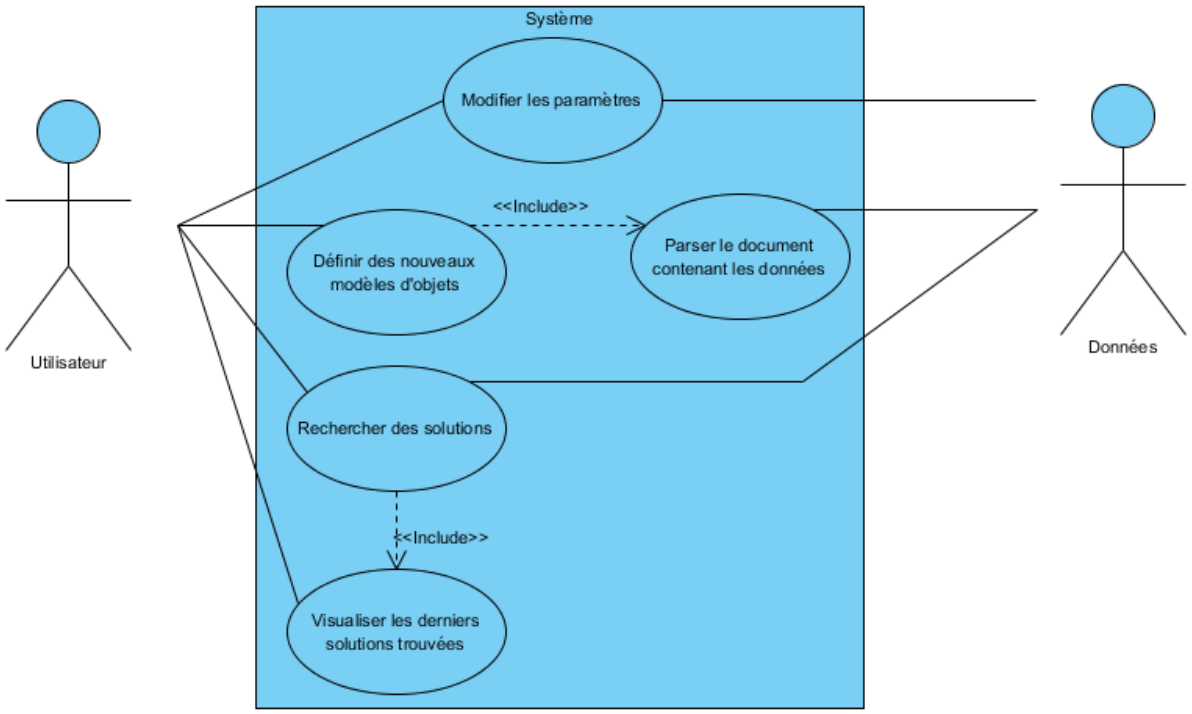
Acteurs	Rôles
Utilisateur	Représente un utilisateur qui cherche à trouver une solution optimale à un problème de rangement de polygones convexes. Il est le responsable de fournir les données nécessaires au programme. Aucune distinction n'est faite entre les utilisateurs.
Données	Contient les informations relatives au problème et aux méthodes de résolution. Gère principalement les paramètres des algorithmes génétiques et de la géométrie du conteneur.

Grille 1.1 – Grilles des acteurs des cas d'utilisation

2. Recensement des cas d'utilisation

A continuation on cherche à détecter les différents cas d'utilisation. C'est-à-dire les fonctionnalités principales que le programme doit comporter. Il est possible que certains cas d'utilisation fassent appels à des cas d'utilisation internes où l'utilisateur ne sera pas à l'origine de l'appel. Cependant tous les cas d'utilisation interagissent avec au moins un des acteurs cités dans la partie précédente. On suppose que les cas identifiés ci-dessous recouvrent la totalité des actions possibles à effectuer par un utilisateur.

Voici le diagramme de cas d'utilisation :



On va étudier plus en détail chacun des cas d'utilisation dans les sous-parties suivantes.

2.1. Modélisation de nouveaux objets

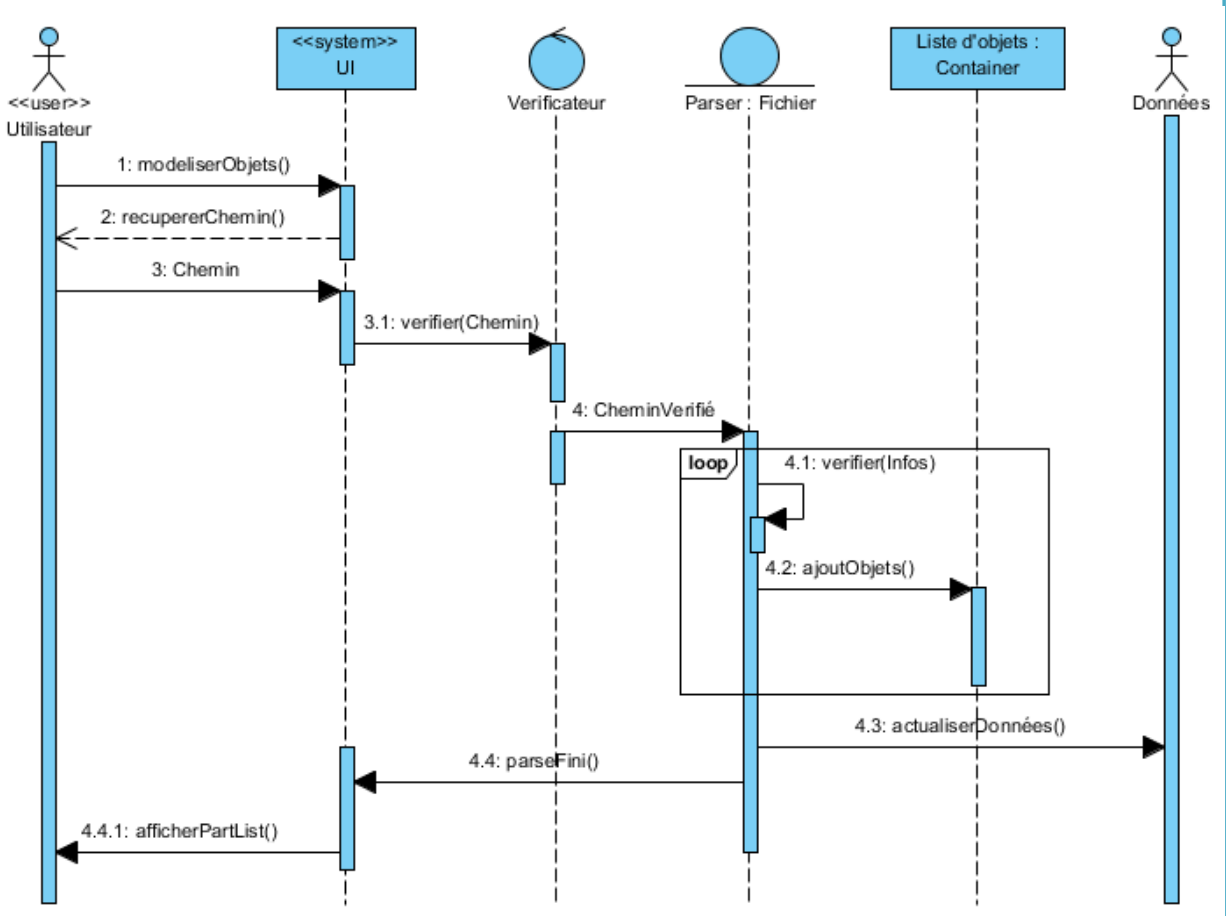
Description du cas : « Modélisation de nouveaux objets »

Identification
Nom du cas : Définir des nouveaux modèles d'objets.
But : Permet à un client de définir des nouveaux objets (représentant les pièces à ranger) à partir d'un fichier respectant les normes de modélisation précédemment fixées.
Acteur principal : Utilisateur.
Acteur secondaire : Données.
Date : le 13/03/2012.

Séquencement
Le cas d'utilisation commence lorsqu'un utilisateur lance la requête de modélisation d'objets.

Préconditions
L'utilisateur possède un fichier contenant les informations nécessaire pour décrire les objets
L'utilisateur connait le chemin d'accès au fichier contenant les données.
Si un objet apparaît en plus d'un exemplaire, la description de l'objet apparaît autant de fois que nécessaire

Diagramme de séquence



Enchaînement nominal

1. **L'utilisateur souhaite ajouter des objets au problème**
2. **Le programme demande le chemin d'accès du fichier contenant les données**
3. **L'utilisateur fournit le chemin d'accès**
4. **Le programme trouve le fichier**
5. **Appel du cas « Parser le document »**
6. **Les informations ont été traduites en formes géométriques**
7. **Le programme affiche les différents polygones décryptés**
8. **L'utilisateur valide les données**

Enchaînements alternatifs
(néant)

Enchaînements d'exception
E1 : Le chemin d'accès au fichier n'est pas valide.
L'enchaînement démarre après le point 3 de la séquence nominale :
4. Le programme avertit l'utilisateur de l'erreur et lui demande de fournir un chemin valide
5. L'utilisateur fournit une bonne adresse d'accès.
La séquence nominale reprend au point 4.

E2 : La requête de modélisation est annulée par l'utilisateur.
L'enchaînement démarre après les points 2, 3, ou 7 de la séquence nominale :
Retour à la fenêtre principale

Post-conditions
Les formes sont ajoutées à la liste d'objets à ranger.

Contraintes non fonctionnelles

Contraintes liées à l'interface homme-machine
Donner la possibilité d'ajouter les données d'un autre fichier.
A tout moment l'utilisateur peut annuler la requête.

2.2. Décomposition analytique d'un fichier

Description du cas : « Parser le document contenant les données »

Identification
Nom du cas : Parser le document
But : Traduire les informations écrites dans un fichier en polygones.
Acteur principal : néant (cas interne inclus dans le cas «Définir des nouveaux modèles d'objets »)
Acteur secondaire : Données
Date : le 13/03/2012.

Séquencement
Le cas démarre au point 5 du cas « Définir des nouveaux modèles d'objets ».
Préconditions

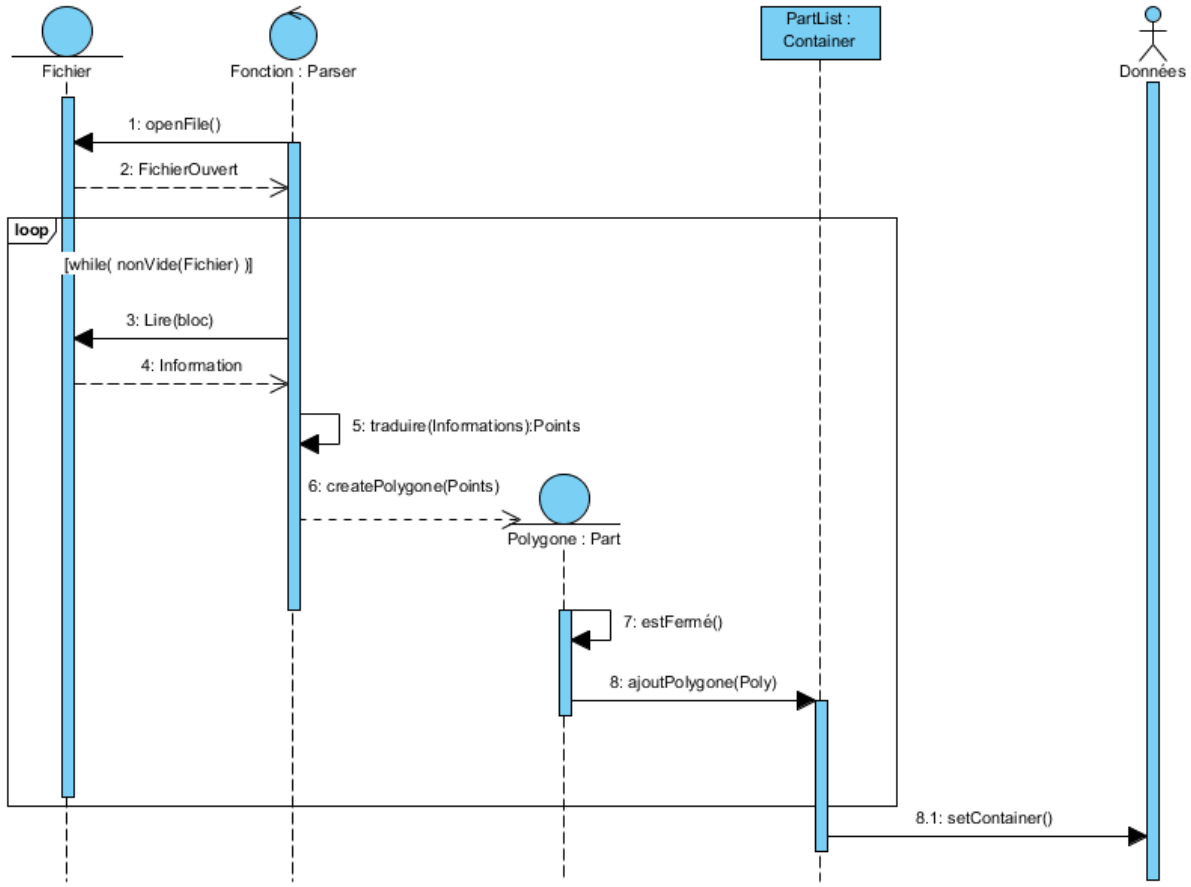
Le fichier respecte les normes sur la description d'objets.
Les objets décrits dans le fichier sont des polygones convexes.

Enchaînement nominal

1. **On ouvre le fichier.**
2. **Tant que le fichier n'est pas vide :**
3. **On lit un premier paquet d'information**
4. **Le programme reconnait un polygone**
5. **Le polygone est ajouté à la liste des formes géométriques**
6. **On recommence au point 2.**
7. **On ferme le fichier**

Enchaînements alternatifs
(néant)

Diagramme de séquence



Enchaînements d'exception

<u>E1 : Le fichier a un mauvais format.</u>
L'enchaînement démarre après le point 4 de la séquence nominale :

5. **Le programme ne reconnaît pas le format des données**
6. **L'utilisateur est informé**

La séquence nominale reprend au point 8.

<u>E2 : Le fichier à lire est vide</u>
L'enchaînement démarre après le point 1 de la séquence nominale :
2. Le fichier est vide
La séquence nominale reprend au point 7.

<u>E3 : Les informations représentent un objet qui n'est pas un polygone.</u>
L'enchaînement démarre après le point 4 de la séquence nominale :
5. Le programme ne reconnaît pas un polygone
6. Le message d'exception sera lancé à l'utilisateur
La séquence nominale reprend au point 8.

<u>E4 : Le polygone ne peut pas être ajouté à la liste d'objets</u>
L'enchaînement démarre après le point 5 de la séquence nominale :
6. La liste d'objets a atteint sa taille limite
7. Le message d'exception est lancé à l'utilisateur
8. Les formes géométriques précédemment ajoutées à la liste sont enlevés
La séquence nominale reprend au point 9.

Post-conditions
Les formes sont ajoutées à la liste d'objets à ranger.

Contraintes non fonctionnelles
(néant)

2.3. Modifier les paramètres

Description du cas : « Modifier les paramètres »

Identification
Nom du cas : Modifier les paramètres
But : L'utilisateur veut modifier les paramètres du problème ou de l'algorithme de résolution du problème. Les paramètres modifiables sont les dimensions du conteneur, le nombre de pièces maximales, des éventuels critères du problème et le temps de calcul maximal.
Acteur principal : Utilisateur
Acteur secondaire : Données
Date : le 13/03/2012.

Séquencement
Le cas démarre lorsqu'un utilisateur souhaite modifier des paramètres

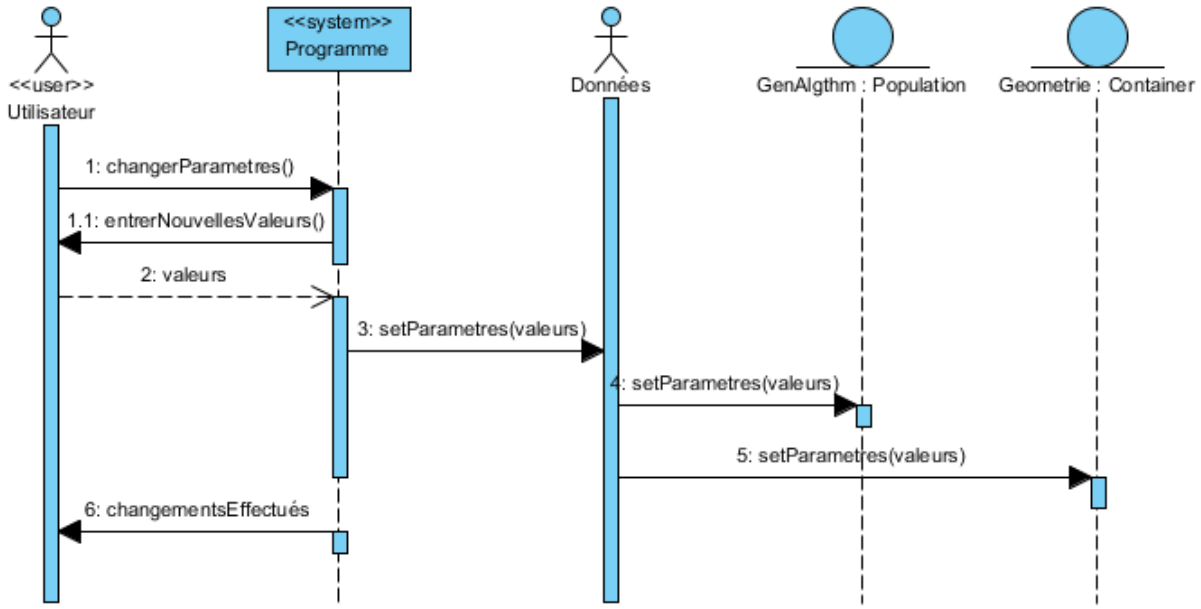
Préconditions
(néant)

Enchaînement nominal
1. **L'utilisateur lance la requête**
2. **On lui présente la liste des paramètres modifiables**

3. **L'utilisateur effectue les changements nécessaires**
4. **L'utilisateur valide les changements**
5. **Les changements sont établis réalisés dans le programme.**

Enchaînements alternatifs
(néant)

Diagramme de séquence



Enchaînements d'exception
<u>E1 : L'utilisateur rentre des valeurs interdites.</u>
L'enchaînement démarre après le point 4 de la séquence nominale :

5. **Le programme reconnaît l'incohérence des données**
6. **Au(x) champ(s) de conflit, les valeurs par défaut sont mises**

La séquence nominale reprend au point 3.

Post-conditions
Les changements sont faits pour tous les parties concernées.

Contraintes non fonctionnelles
(néant)

Contraintes liées à l'interface homme-machine
Pour tous les paramètres modifiables, expliquer ce qu'il représente en termes pratiques.

2.4. Visualisation des derniers résultats

Description du cas : « Visualisation des derniers résultats »

Identification
Nom du cas : Visualiser les derniers résultats
But : Affichage des résultats des dernières optimisations lancées.
Acteur principal : Utilisateur
Acteur secondaire : Données
Date : le 13/03/2012.

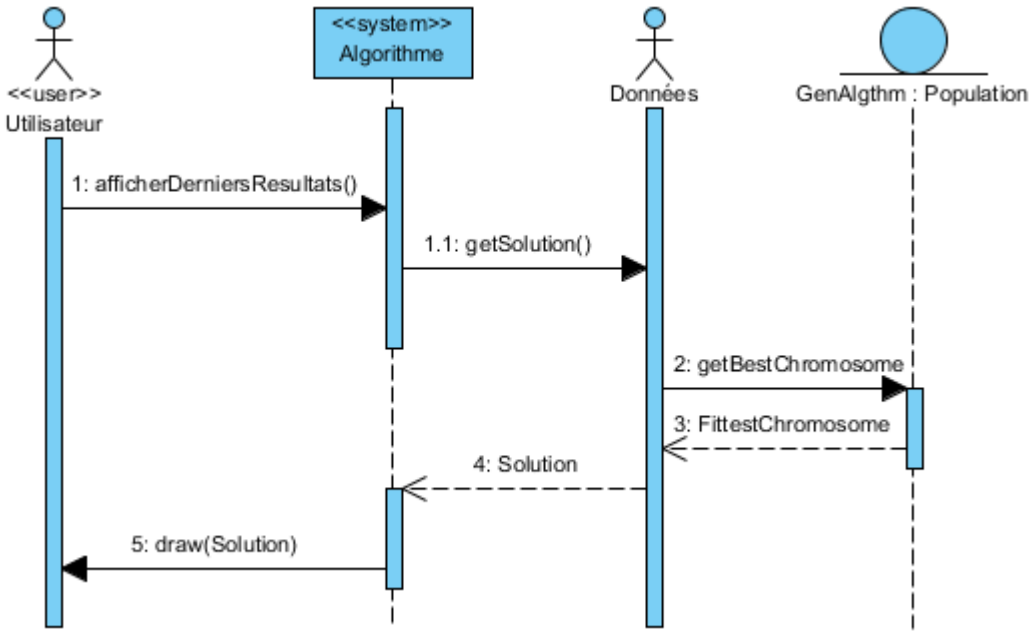
Séquencement
Le cas commence sous requête de l'utilisateur.

Préconditions
Au moins une optimisation a été lancée auparavant.

Enchaînement nominal

1. **L'utilisateur lance la requête**
2. **La fenêtre d'affichage s'ouvre**
3. **L'utilisateur choisit un problème entre la liste des derniers problèmes**
4. **Le graphique finale est affiché avec des données sur celui-ci.**

Diagramme de séquence



Enchaînements alternatifs
(néant)

Enchaînements d'exception
(néant)

Post-conditions
Les informations complémentaires sont suffisantes pour pouvoir recréer le problème.

Contraintes non fonctionnelles

Contraintes liées à l'interface homme-machine
L'utilisateur ne peut rien modifier sur cette fenêtre autre que son choix parmi la liste des résultats.

2.5. Optimisation du problème

Description du cas : « Optimisation du problème »

Identification
Nom du cas : Recherche des solutions
But : Recherche d'un rangement optimal pour les objets définis et dans le conteneur fixé.
Acteur principal : Utilisateur. Acteur secondaire : Données

Séquencement
Le cas commence sous requête de l'utilisateur.

Préconditions
La liste d'objets ne doit pas être vide.

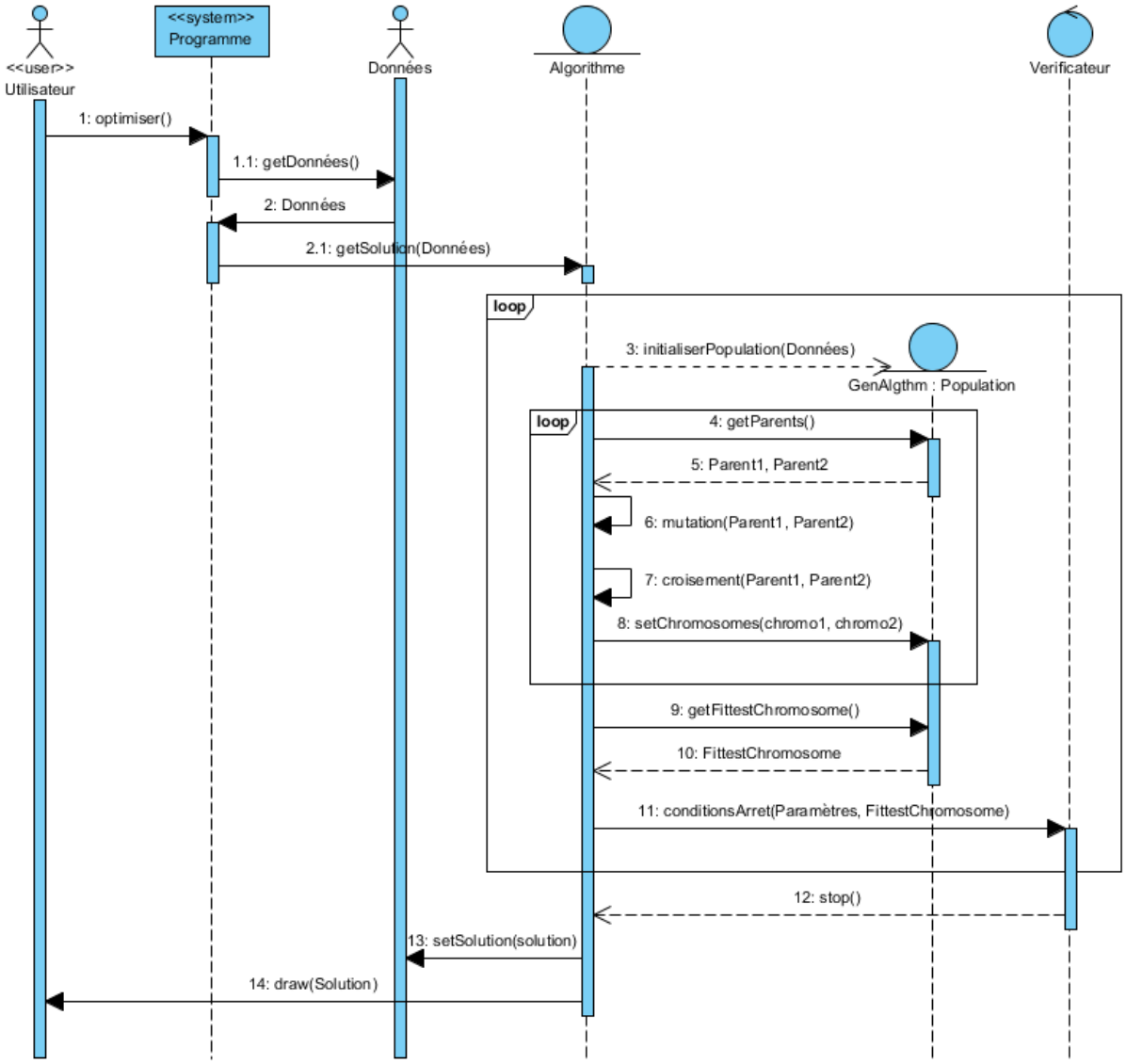
Enchaînement nominal

1. **L'utilisateur lance la requête**
2. **Un récapitulatif des données est affiché**
3. **L'utilisateur valide les données**
4. **Une fenêtre de chargement s'ouvre**
5. **Les algorithmes de résolution sont lancés**
6. **Les données nécessaires sont récupérées**
7. **Lorsqu'un des critères d'arrêt est rencontré, le résultat en cours est gardé**
8. **Appel du cas « Visualisation des derniers résultats »**

Enchaînements alternatifs
A1 : L'utilisateur ne valide pas les données
L'enchaînement démarre après le point 2 de la séquence nominale :
Retour à la fenêtre principale.

Enchaînements d'exception
(néant)

Diagramme de séquence



Post-conditions
Le résultat affiché doit être l'un des plus optimaux trouvés. En cas d'impossibilité de trouver une solution la meilleure est donnée et on indiquera les pièces qui n'ont pas pu être rangées.

Contraintes non fonctionnelles
(néant)

Contraintes liées à l'interface homme-machine
(néant)

3. Diagrammes de classes

En analysant les diagrammes précédents on peut dégager deux grandes parties vitales à la définition du problème. La première étant la partie de la définition de la géométrie et la deuxième la définition des algorithmes qu'on utilisera. Pour plus d'informations sur les choix dans ces domaines voir le document joint (*Etat de l'art*).

Regardons maintenant plus en détail en quoi consiste chacune de ces parties.

3.1. Package Géométrie

On cherche à définir une géométrie qui nous permettra de définir tous les objets et le conteneur. On sait qu'on travaillera avec des polygones et qu'un polygone peut être défini comme une suite de points. On trouve alors facilement qu'on devra définir les classes suivantes : Point, Polygone et Suite (ou, en termes informatiques, une Liste). Pour le conteneur, on sait qu'on travaillera qu'avec des conteneurs rectangulaires, avec une largeur et une hauteur fixées.

Pour ce qu'il s'agit des listes, on sait que dans la plupart des langages informatiques, il existe une bibliothèque contenant une classe « Liste » indépendante du type d'objets qu'elle contient et contenant des fonctions de base (taille de la liste, ajout, recherche, suppression, ordonnancement, etc.). Pour ces raisons, cette classe sera utilisée pour la définition d'un polygone (liste de points) mais aussi pour la définition d'un conteneur qui aura une liste de polygones. Cependant elle n'apparaitra pas dans notre diagramme de classe car ce n'est pas une classe du problème.

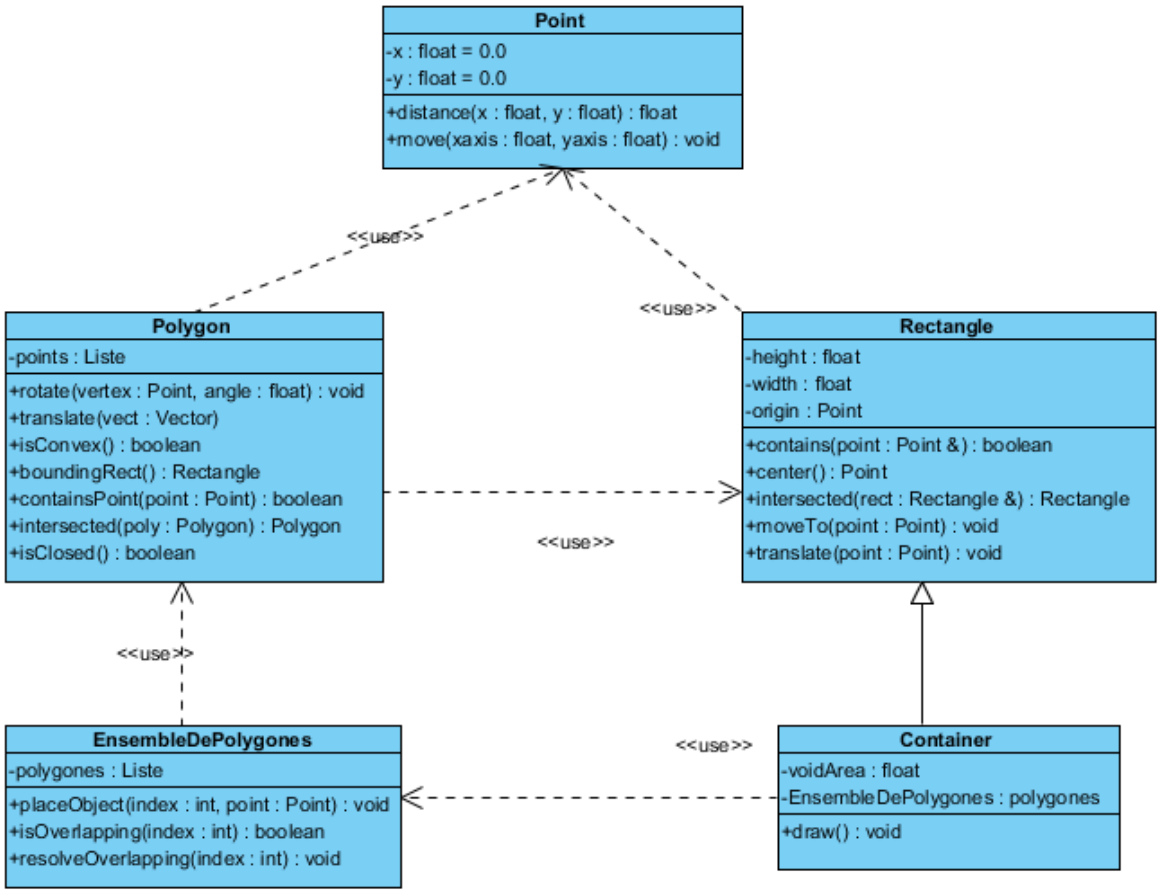
On sait aussi que dans ce travail on aura besoin d'effectuer différents calcules sur les polygones et comme on a vu dans la bibliographie, il est souvent plus efficace de travailler avec les rectangles contenant les polygones, qu'avec les polygones mêmes. On définira alors une classe « Rectangle » qui sera utilisée dans la classe des Polygones. De plus elle sera une généralisation de la classe « Conteneur » qui n'est rien d'autre qu'un rectangle. On évitera de définir un Rectangle comme une spécialisation d'un polygone, car on ne s'intéresse pas à celui-ci comme une liste de point, mais comme un objet avec un coin, une hauteur et une largeur définis.

Finalement le diagramme des classes est :

Un **point** est défini tout simplement par ses coordonnées (deux flottants) et on n'aura pas besoin de d'opérations autre que le calcul de distance par rapport à l'origine et le déplacement d'un point.

Les **polygones** sont une liste de point. On notera l'importance de fonctions géométriques permettant le déplacement, la rotation ou même la détection de collision entre deux polygones.

La classe **Rectangle** ne diffère pas beaucoup de la dernière en ce qui concerne les méthodes mais la définition reste différente, comme on l'avait spécifié précédemment.



On définit une classe pour les listes d'objets, avec des fonctions vitales pour le projet : **EnsembleDePolygones**. Les méthodes les plus importantes sont : une première qui détecte si un polygone est en conflit avec les autres polygones de la liste (superposition des objets) et une deuxième qui déplacera l'objet dont on passe l'index jusqu'à ce que l'objet ne soit plus en superposition avec les autres pièces dans la Liste.

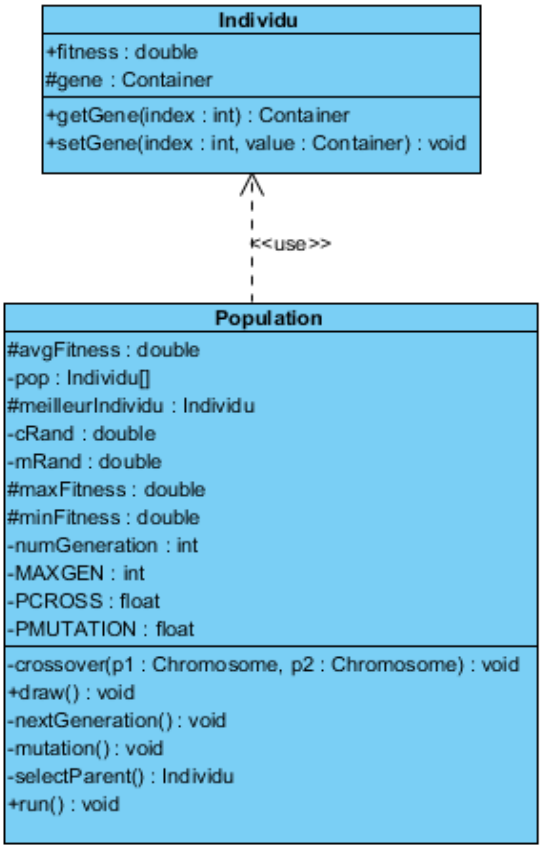
Comme on a déjà vu le **conteneur** est un rectangle. On a décidé d'ajouter à la définition du conteneur la liste des objets qu'il contient à leur position (décrite explicitement par leurs coordonnées), le conteneur est donc non seulement la boîte contenant les objets mais l'ensemble du tout. On a ajouté aussi une variable qui contient l'espace vide dans le conteneur (ce qui servira pour le calcul de la fitness dans les algorithmes) et une méthode d'affichage.

3.2. Package pour les algorithmes génétiques

Il est tout d'abord important de définir un **individu** : on choisit de le définir comme un conteneur (en algorithme génétique on dira que c'est son gène) et sa performance (c'est à dire la fitness associée au conteneur). On supposera que la fitness est calculée directement lors de la création de chaque Individu. Il ne nous reste qu'à définir des méthodes d'accès et des modificateurs.

La classe la plus importante dans ce package est la classe **Population**. Son attribut principal est un tableau d'individus. En effet ce sont eux qui définiront la population. De plus on devra trouver d'autres attributs pour obtenir plus facilement des caractéristiques de la population comme la fitness moyenne, maximale et minimale, le nombre de générations calculées ou l'individu le plus performant rencontré. Les autres attributs seront plutôt des paramètres spécifiques aux algorithmes génétiques tels que le nombre maximal de générations, les probabilités de croisement et de mutation, etc. Finalement en ce qui concerne les méthodes, on trouvera les 2 méthodes basiques pour tous les algorithmes génétiques : les méthodes de croisement et de mutation. Celles-ci seront appelées par une fonction qui permettra passer de génération en génération : *nextGeneration*. Dans cette fonction on aura aussi besoin d'une méthode qui choisira des parents. Et puis on pourra trouver une fonction principale qui s'effectuera en boucle, jusqu'à ce que les conditions d'arrêt soient atteintes (c'est la fonction « *run()* »). Finalement on aura besoin d'une méthode d'affichage.

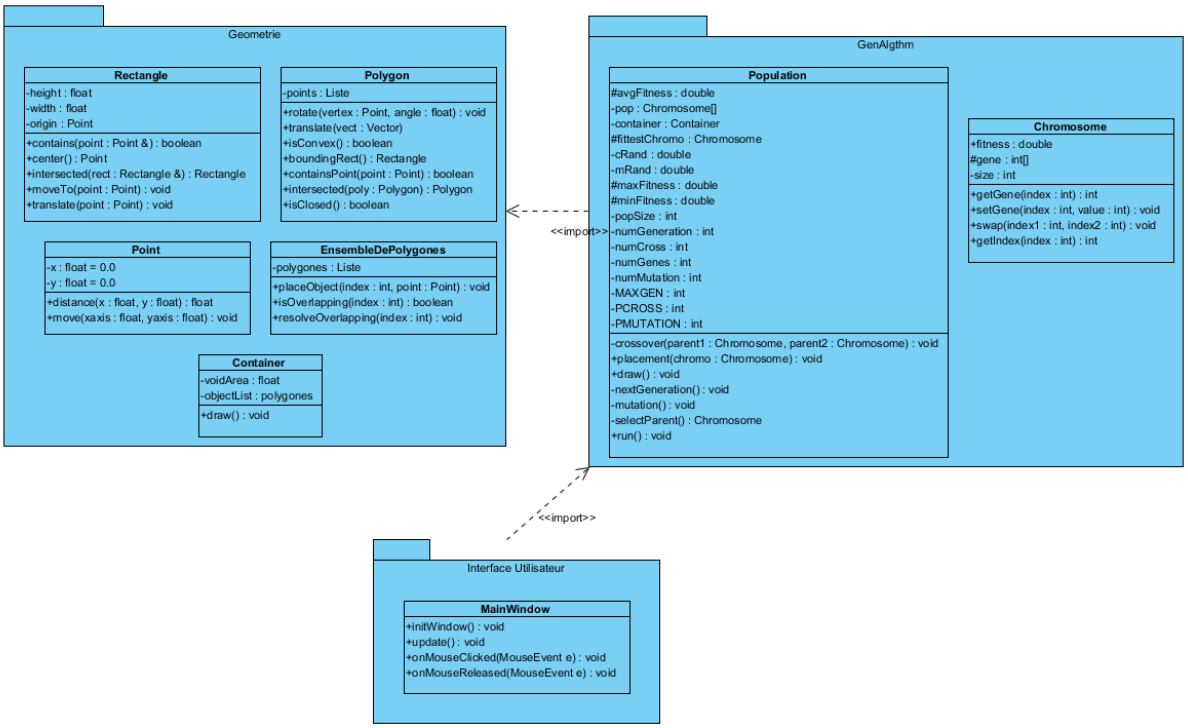
On obtient donc le diagramme de classe suivant :



On notera que les heuristiques de placement n'apparaissent pas explicitement entre les méthodes, mais celles-ci seront implémentées directement à chaque fois qu'un nouveau conteneur est créé, donc à chaque génération.

3.3. Relation entre les différents packages

Maintenant qu'on a défini les classes principales et les paquets auxquels elles appartiennent, on peut étudier les relations entre eux.



On a ajouté un paquetage pour l'interface utilisateur qui sera chargé de l'affichage de la fenêtre principale et des interactions homme-machine.

Conclusion

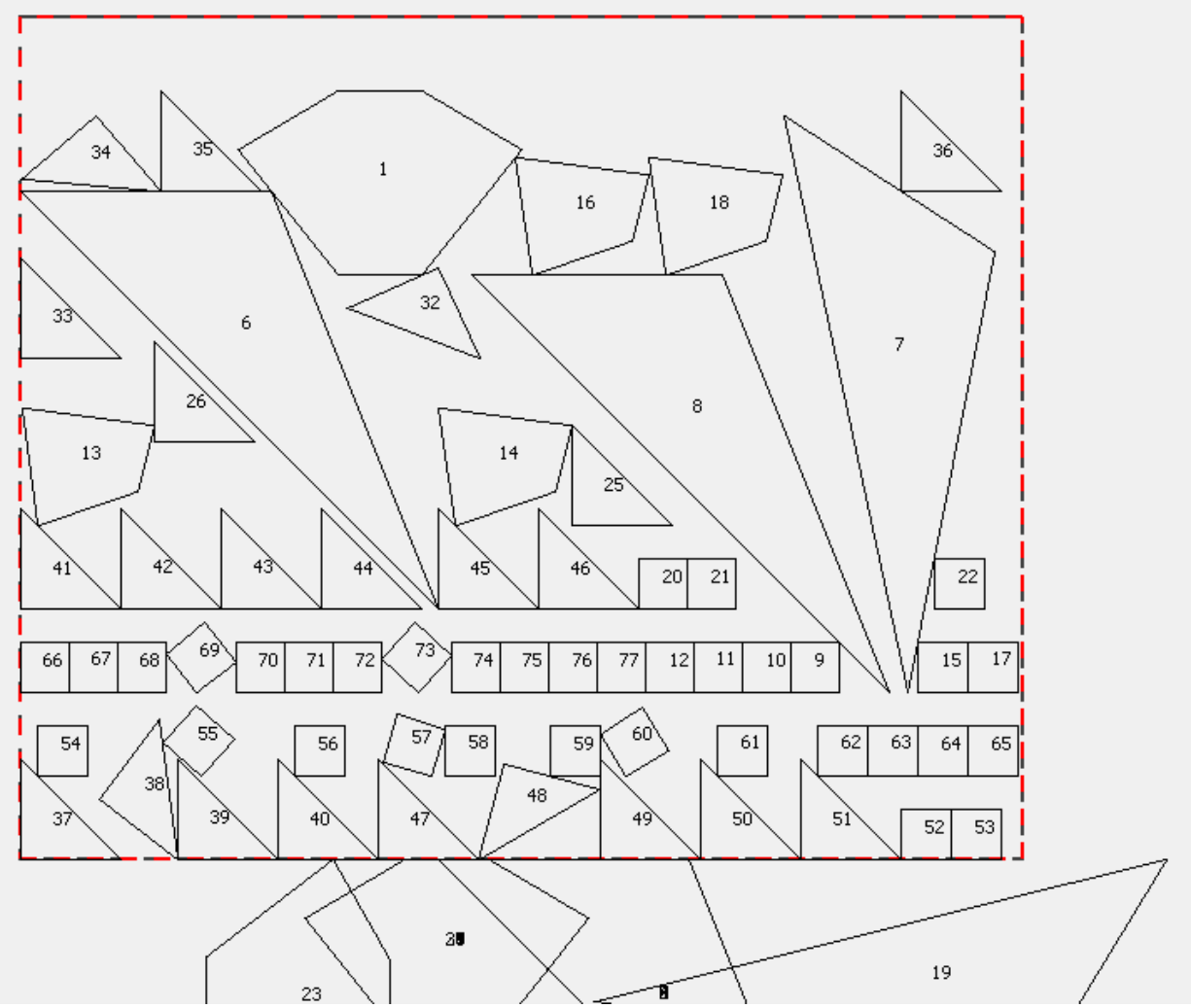
Ce document recense tous les différents cas d'utilisation et les acteurs susceptibles d'agir sur le programme. On a traité tous les nominaux ainsi que les non-nominaux. De cette façon si toutes les préconditions et contraintes sont respectées, aucun autre scénario est envisageable.

De plus, ce document nous permet d'avoir un aperçu de l'architecture que devra suivre le programme. Les diagrammes de classes définissent les éléments indispensables au projet mais en aucun cas ils correspondront exactement aux classes finales.

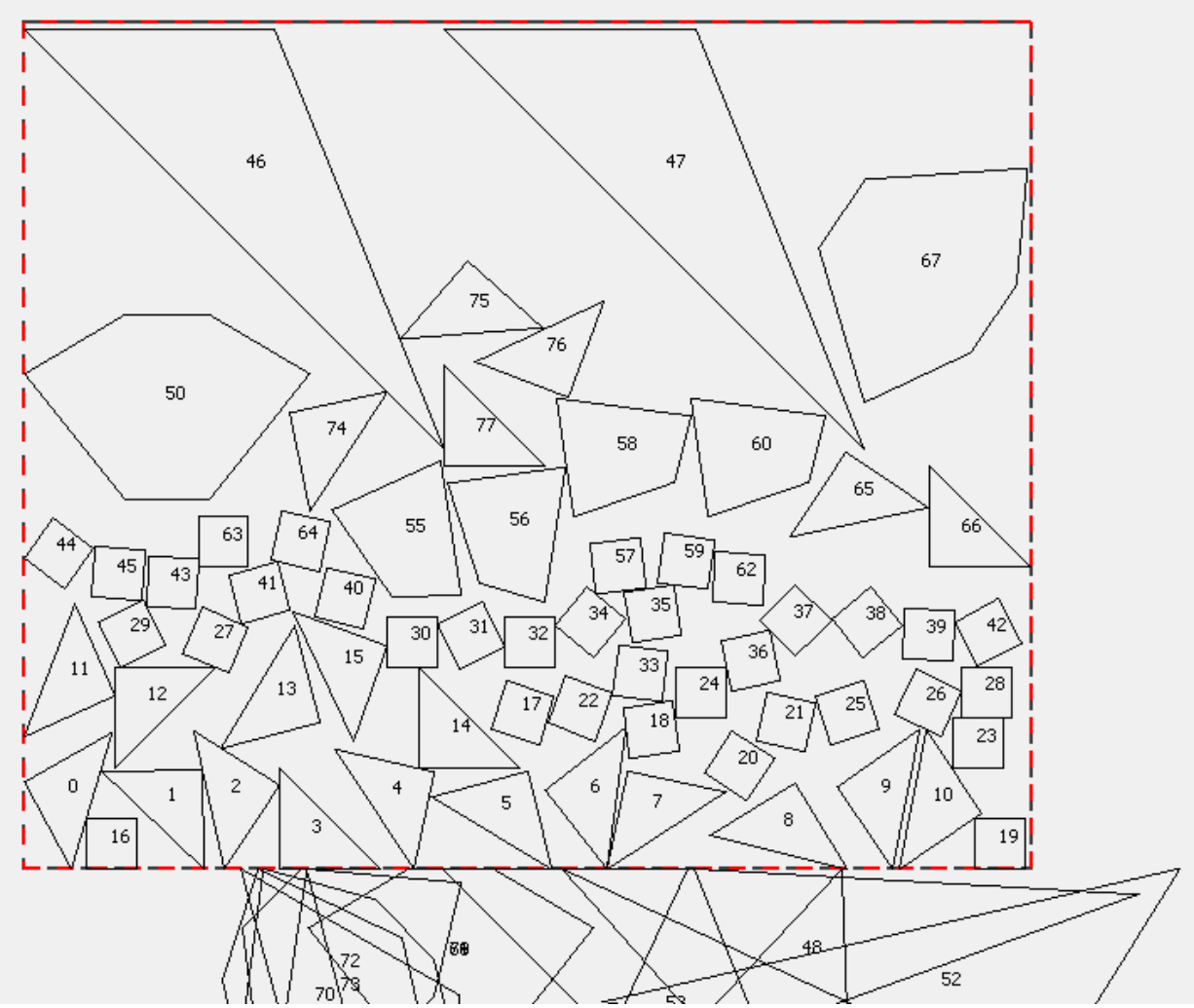
Comme la solution n'est pas axée sur l'interface utilisateur, les classes relatives à celle-ci n'apparaissent pas. Néanmoins on pourra envisager de les ajouter par la suite.

4.2 Résultats

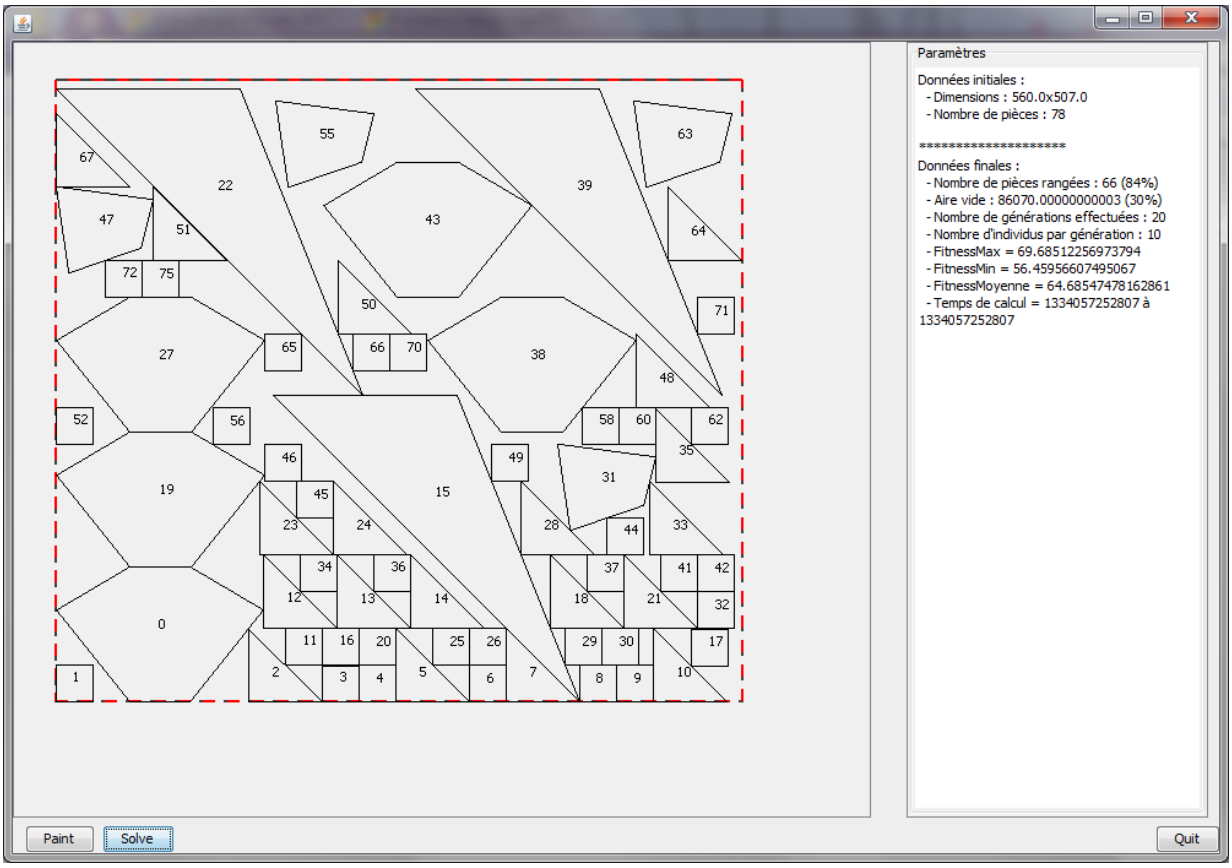
4.2.1 Premiers résultats



Pour pouvoir tester des premiers de solution on commence par créer un jeu de différentes pièces (72 objets) qui est un ensemble très hétérogène. Cependant, la complexité n'est pas très importante car l'algorithme implémenté appliqué est très basique : algorithme en bas à gauche, avec une hauteur minimale de 1.5 fois la hauteur du plus petit polygone (dans ce cas le carré). Les pièces qui se retrouvent hors le cadre sont les pièces qui n'ont pas pu être placées dans le conteneur et qui ont donc été sorties. Une rotation aléatoire est donnée aux pièces avant leur placement. Aucune attention a encore été portée à l'affichage de l'application.



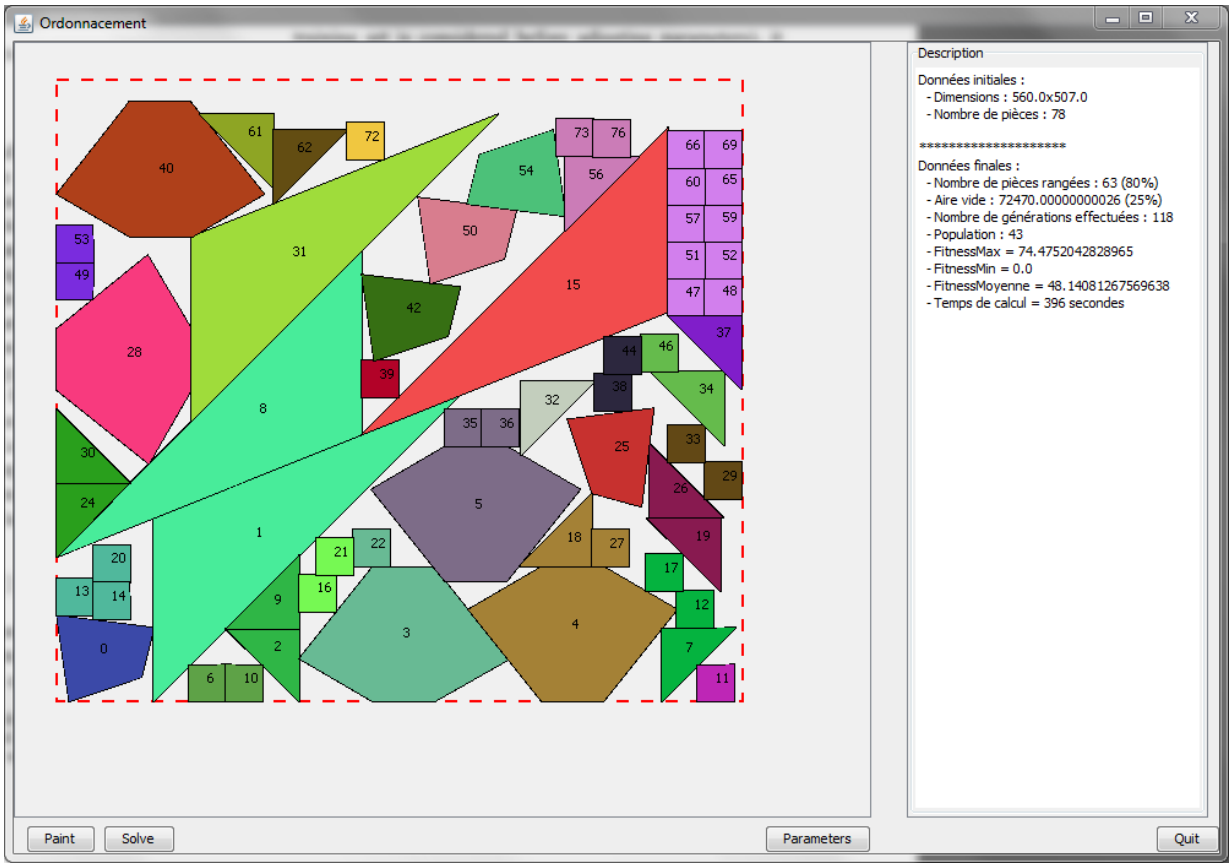
On peut alors appliquer un **algorithme génétique** simple. Les mutations seront des rotations sur un angle entre 0° et 360° avec une probabilité $p_{mut} = 0.15$. Les croisements, comme défini dans la partie précédente s'agissent de croisement entre l'ordre des pièces. Le résultat ici, donne le résultat pour 20 générations. On peut noter une légère amélioration. Les pièces les plus grandes semblent être moins préférées tandis que les petites sont cumulées au fond de la boite. Les mutations d'un angle quelconque semblent pas très intéressantes et en plus la probabilité de mutation semble un peu élevée.



Ajoutons alors quelques données utilisateurs, plus un affichage de fenêtre qui permet de lancer le calcul de l'algorithme et d'afficher la configuration finale. Ici on peut apercevoir des données vitales pour effectuer des bons tests comme le pourcentage d'air vide finale ou encore le pourcentage de pièces utilisées.

La configuration ci-dessous n'accepte pas des rotations. On peut alors voir que des configurations plus "logiques" sont données même si on sait que cela reste spécifique à ce jeu de données spécifique (par exemple : si l'ensemble de données est un ensemble de triangles tous égaux, alors la meilleure configuration aura des rotations sur la moitié des triangles pour pouvoir les combiner à des triangles n'ayant pas subi de rotation pour former un carré).

4.2.2 Modification de la fonction de la calcul de la fitness

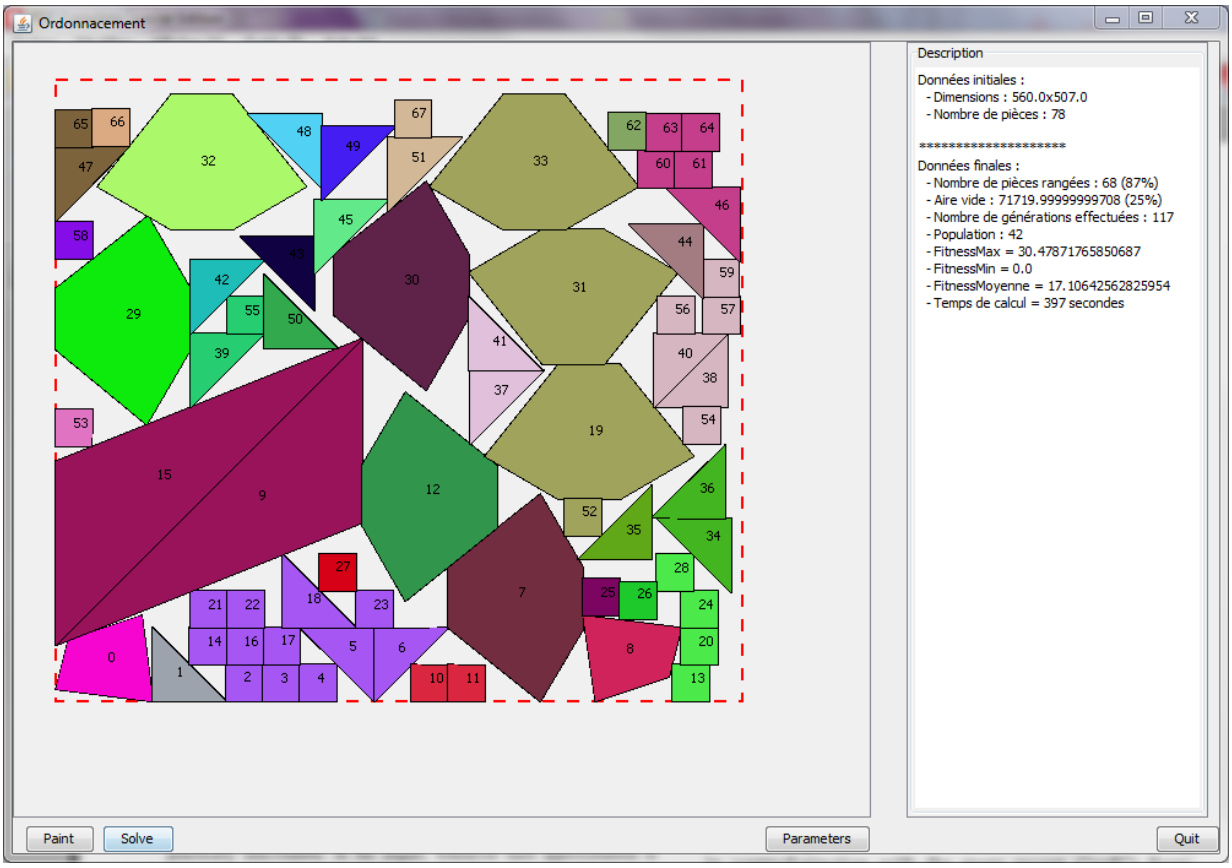


En ajoutant des couleurs différentes aux pièces on obtient le résultat ci-dessus. On rappelle que l'algorithme génétique utilisé calcule la fitness linéairement pour chaque individu, en utilisant la formule qui suit :

$$fitness = (A_{Totale} - A_{vide}) * 100/A_{Totale}$$

Or cette formule prend avec presque autant des probabilité une solution avec un aire vide optimal et un aire vide "presque optimal" on cherche alors une autre formule.

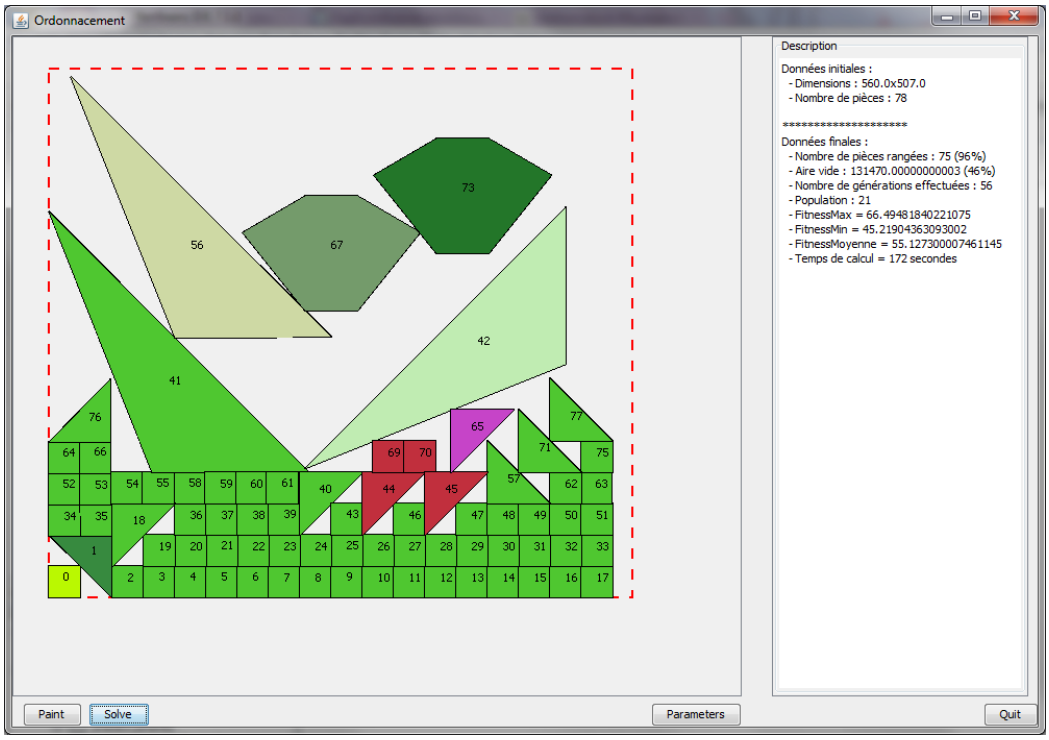
Remarque : Les couleurs ajoutées sont affichées aléatoirement. Ceci est vrai, sauf pour les objets qui sont en contact avec un autre : ils ont alors tous la même couleur. La fonction qui identifie les groupes d'objets reste à améliorer.



On prend alors un calcul exponentiel qui aura une importance plus grande chez les "bons" résultats et sans aucune forte préférence si les résultats sont moins bons.

Data: A_{total} : air total du conteneur
Result: fit : fitness d'un individu
double aireTotale = container.getHeight() * container.getWidth();
double x = (aireTotale - container.getVoidArea()) * 4.61512052 / aireTotale;
//valeur exponentiel d'un nombre entre 0 et 4.61512052, donne des //valeurs entre 1 et 101. On calcule alors l'exp(x)-1
fitness = Math.expm1(x);

Les résultats ne présentent pas des nettes améliorations



Voici une dernière fonction de fitness : une fonction qui préférerai des combinaisons d'objets où les objets seraient plus "agglomeraient". C'est le calcul le plus complexe jusqu'à présent et il ne semble pas très efficace dans des cas généraux même si dans des listes d'objets précis, il converge nettement plus rapidement.

Result: *fit* : fitness d'un individu
begin
  compacite ⟵ 0;
  for *(int i = 0; i < container.size(); i++)* **do**
    if *(this.container.isInContainer(i))* **then**
      this.container.countRelations(i);
      compacite += this.container.compacite(i);
    end
  end
  fitness = compacite / container.size() / 4.0 + (aireTotale - container.getVoidArea()) / aireTotale * 100.0 * 3.0 / 4.0;
end

4.2.3 Modification de la fonction de croisement

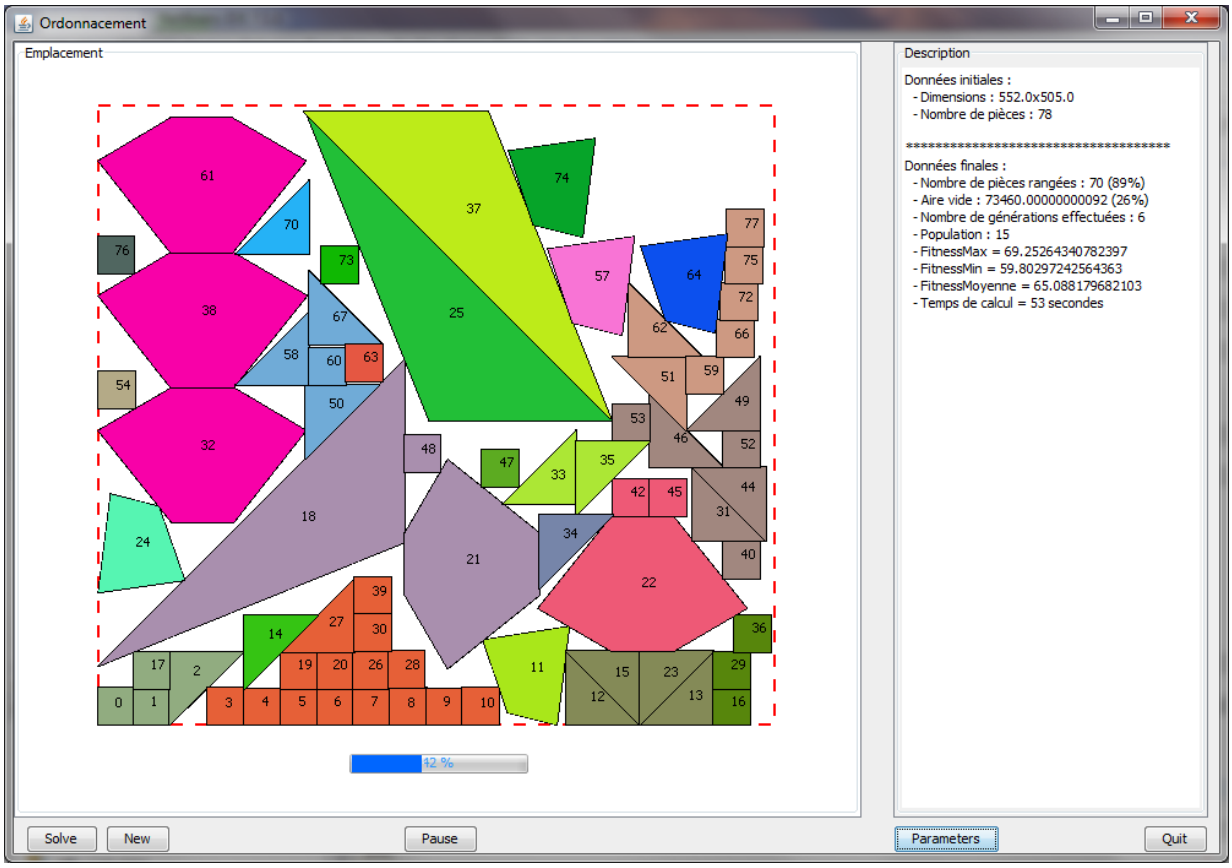


Figure 4.1: Résultat pour la fonction de croisement par relation

La fonction croisement par relation effectue le croisement entre deux individus parents. L'opérateur de croisement prend un ensemble (de relations) des pièces du père et les associe à celles de la mère tout en évitant les incohérences (doublons, etc.)

Les groupes d'éléments sont choisis avec une plus grande probabilité s'il est composé d'un grand nombre de pièces. Les conflits de pièces sont résolus en gardant l'emplacement où la pièce ait un plus grand nombre de relations entre les différents objets.

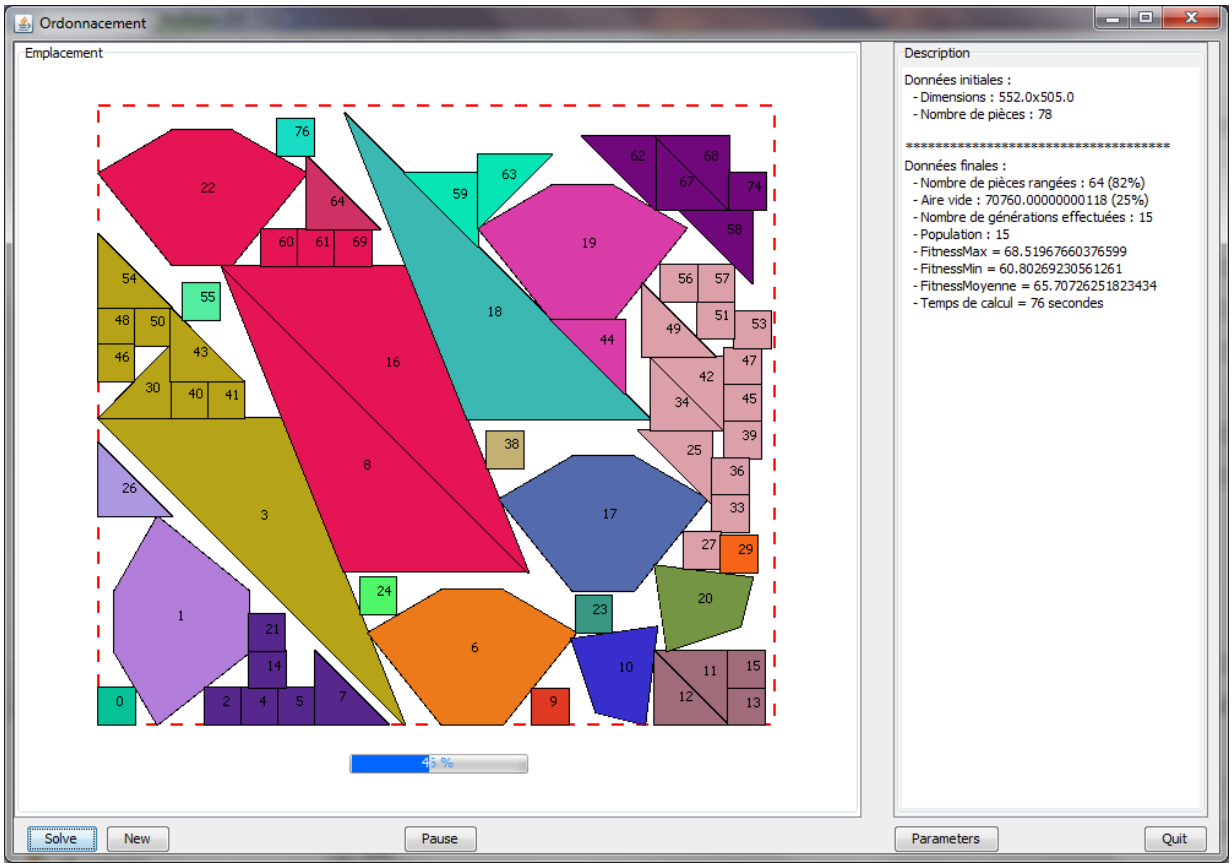


Figure 4.2: Résultat pour la fonction de croisement par compacité

Opérateur de croisement prenant en compte la compacité des objets dans le conteneur fils. On prend l'objet le plus en bas et le plus à gauche du conteneur d'un des parents on le met dans le conteneur fils et on calcul la compacité, on fait de même avec l'autre parent et on garde la pièce qui a une compacité plus élevée. On repète le procédé pour toutes les pièces.

4.2.4 Modification de la fonction de mutation

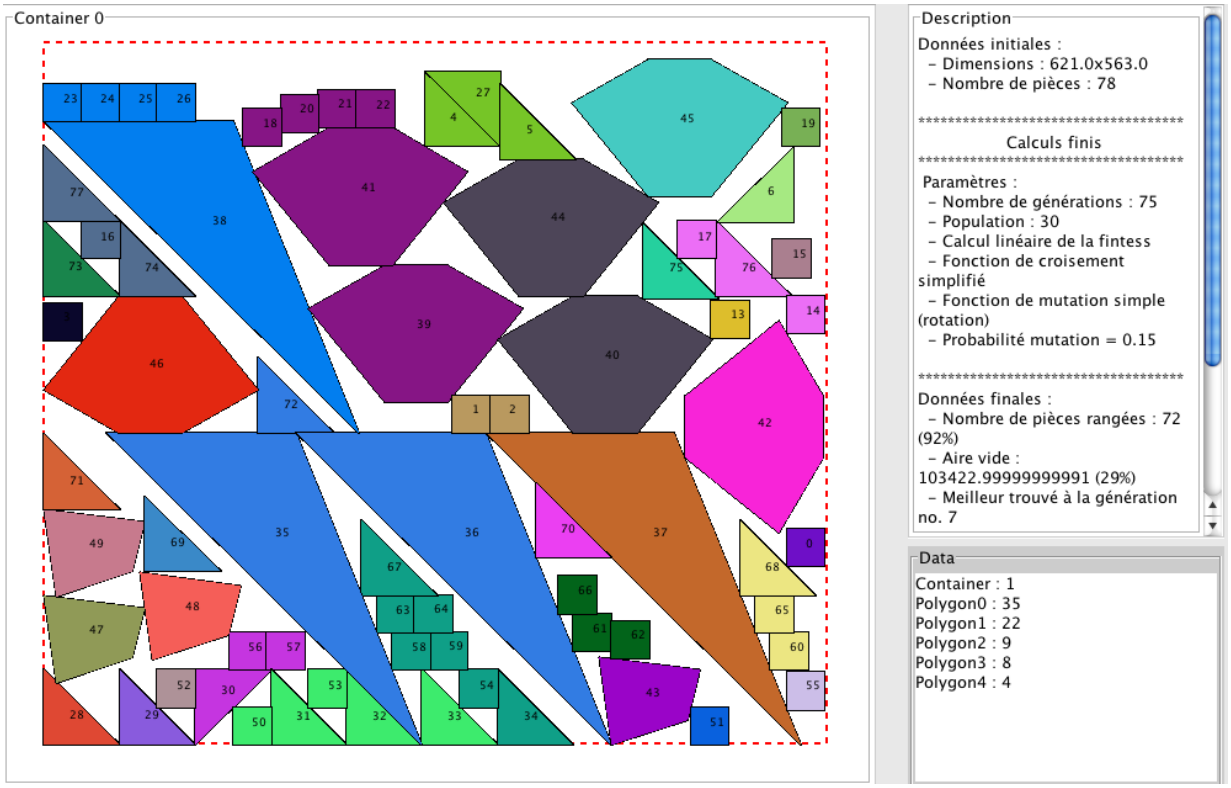


Figure 4.3: Résultat pour la fonction de mutation simple

Fonction de mutation simple : rotation d'un polygone, d'un degré quelconque. La mutation se fait sur chaque polygone avec une probabilité pMutation et toutes les polygones sont examinés. La mutation se fait avant le placement pour éviter les soucis de sur-emplacement.

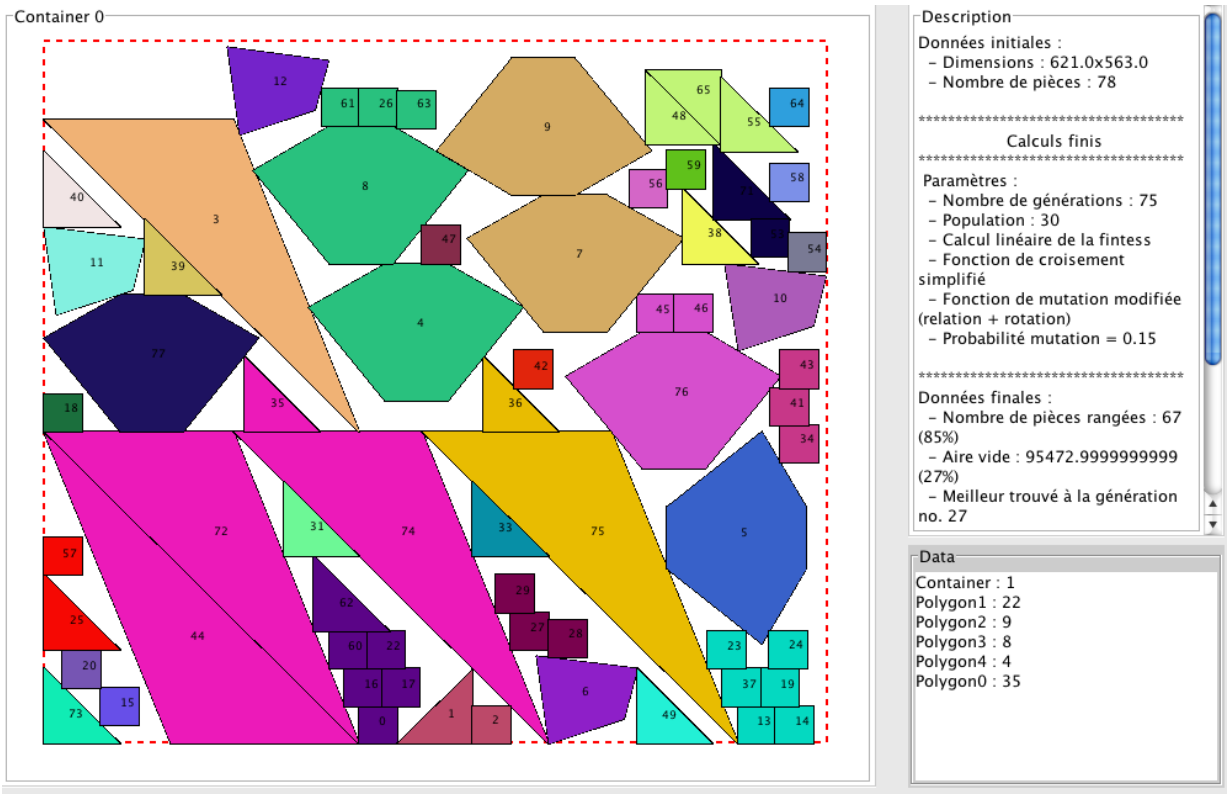


Figure 4.4: Résultat pour la fonction de mutation modifiée

Cette fonction utilise les "relations" entre objets. Comme la précédente elle va traiter polygone par polygone. Pour chacun elle analyse le nombre de relations qu'elle a (entre 2 objets les relations suivant sont possibles : ils sont disjoints - valeur 0 -, ils sont en contact sur un axe - valeur 2 -, ils sont en contact avec un sommet et un axe -valeur 4- et puis ils ont exactement un segment en contact - valeur 8-), si un polygone est trouvé sans relations, alors la fonction créé une nouvelle relation avec cet objet et un autre objet sans relations avec une probabilité p_{mut} sinon, on effectue une mutation simple (rotation).

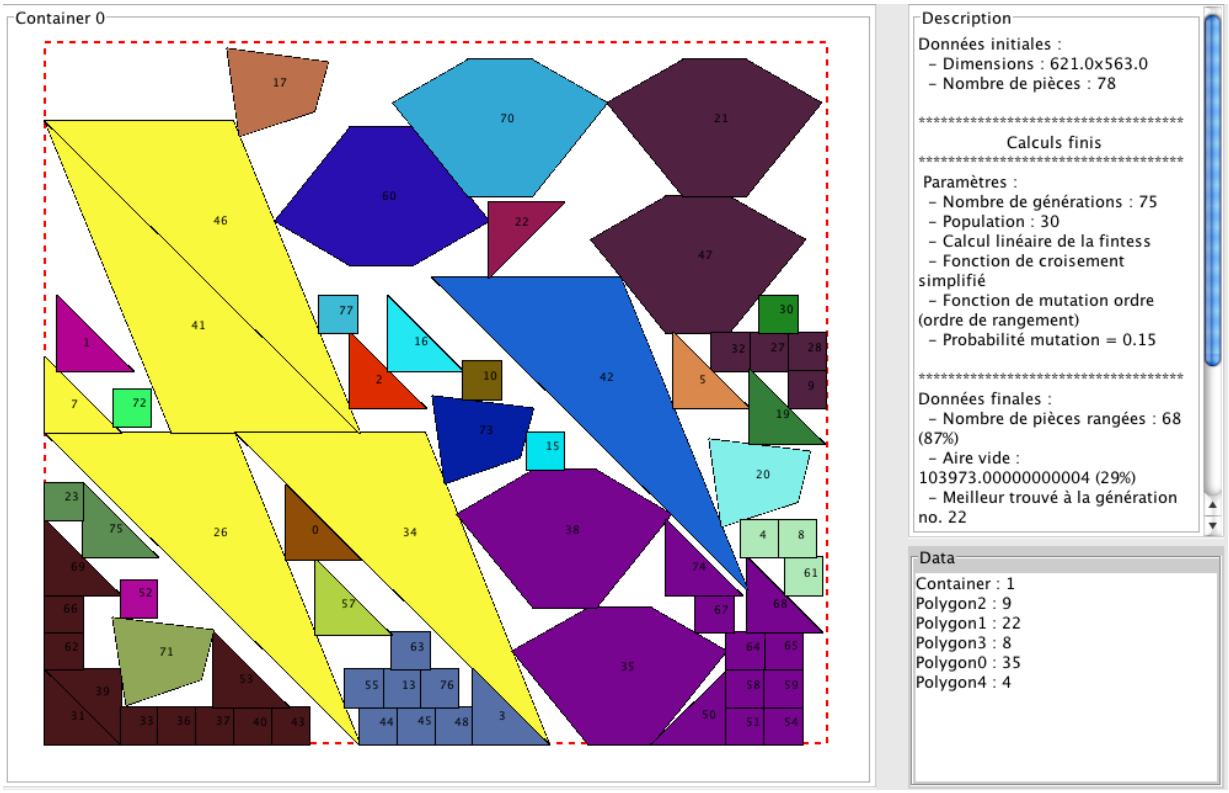


Figure 4.5: Résultat pour la fonction de mutation par ordre

On réfléchis maintenant à ce qu'est essentiellement un individu : une liste de polygones. On pense alors qu'il est peut être intéressant de faire tout simplement une permutation entre les différents objets. La fonction choisi donc aléatoirement deux indices d'objets et les inter-change.

4.2.5 Edition des paramètres

Avec une base déjà établie on commence a avoir un besoin de pouvoir éditer plus de critères en tant qu'utilisateur de l'application. On ajoute alors une fenêtre de modifications qui est divisée en différentes sous catégories :

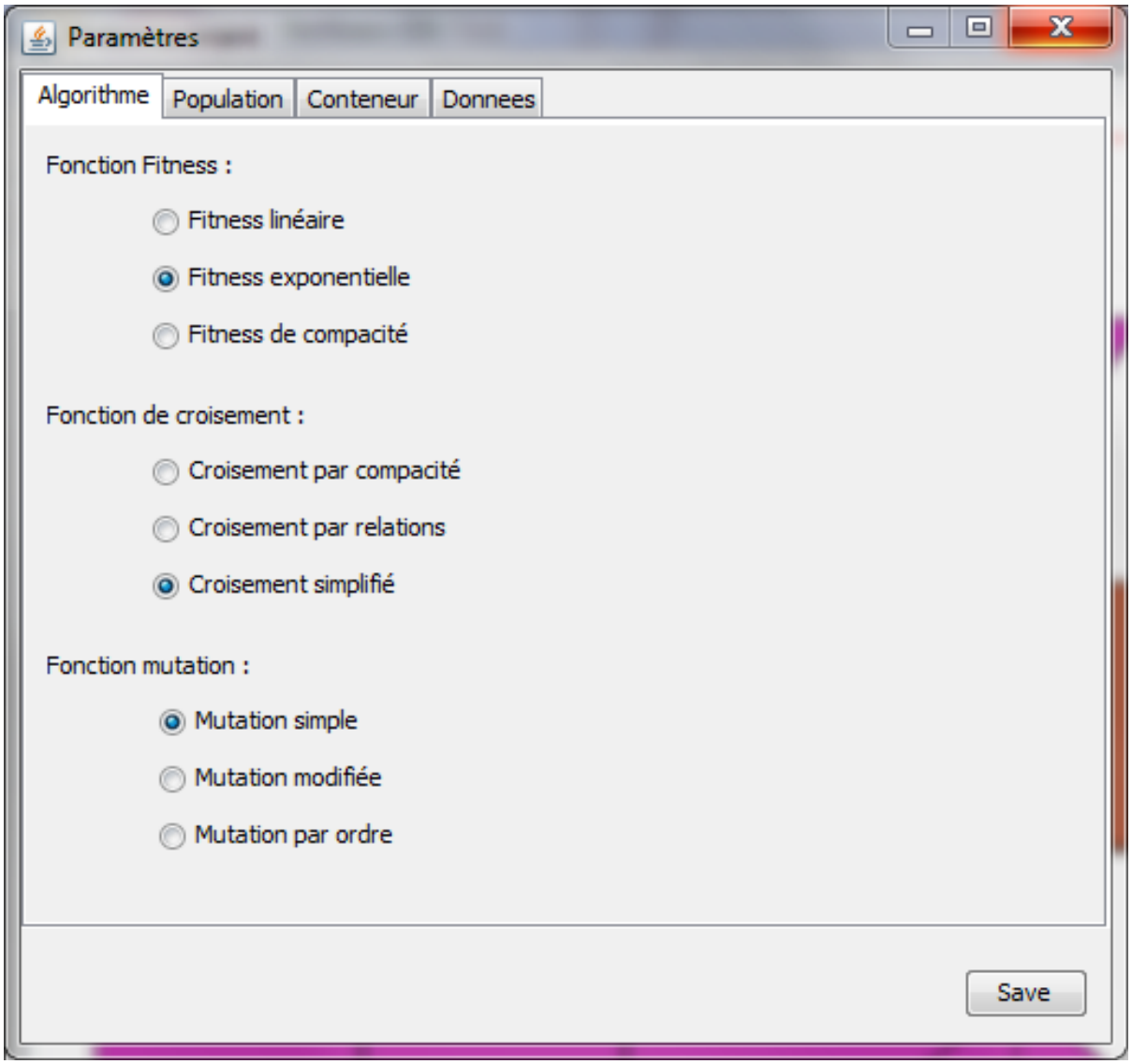


Figure 4.6: Affichage de la fenêtre paramètres permettant de modifier l'algorithme génétique

On peut changer dans cette fenêtre la fonction de calcul de la fitness, la fonction de croisement utilisé, et la fonction de mutation.

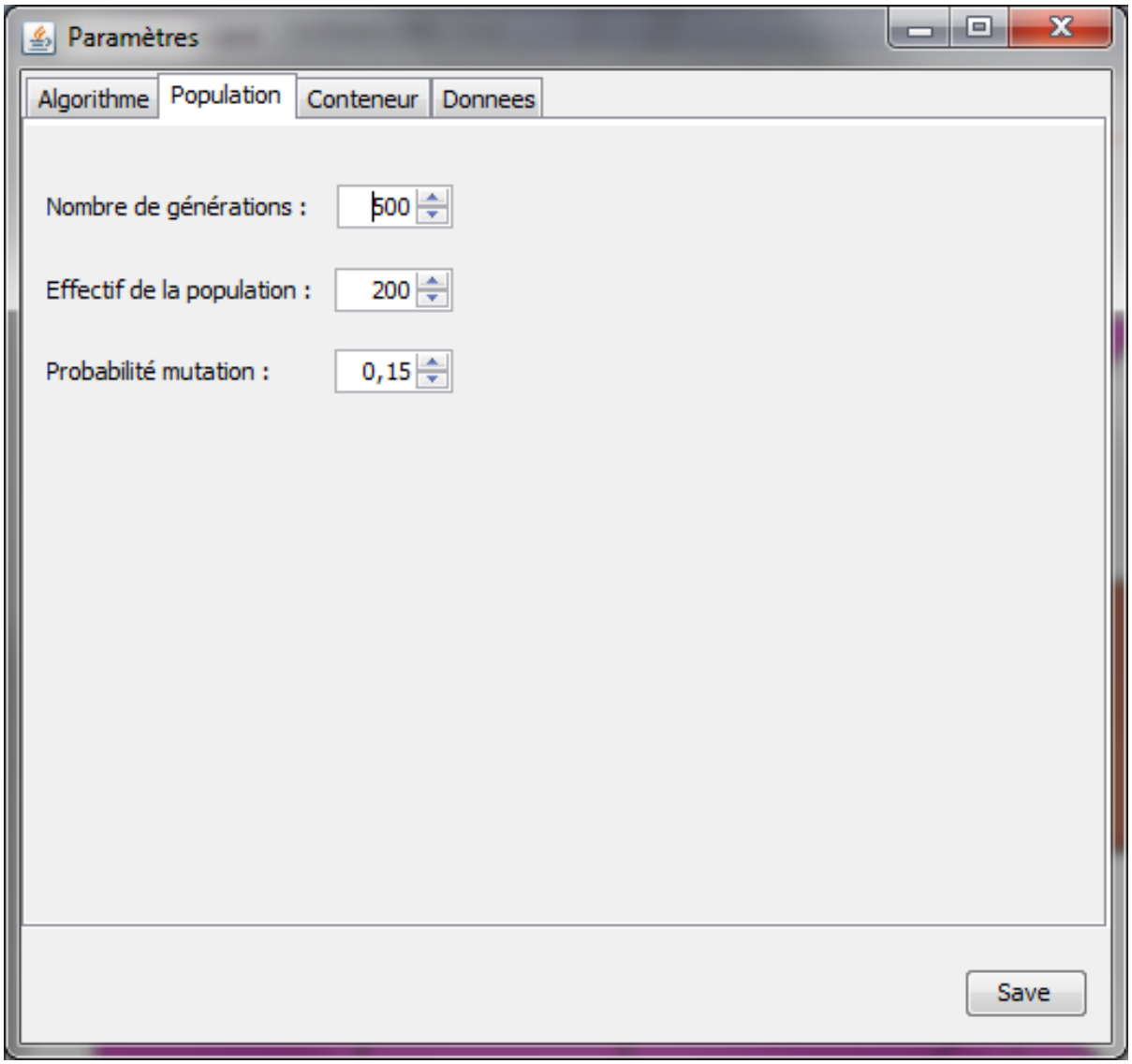


Figure 4.7: Affichage de la fenêtre paramètres permettant de modifier la population de l'AG

On peut changer dans cette fenêtre les paramètre de la population utilisée lors de l'algorithme génétique. On peut modifier le nombre de générations à calculer, le nombre d'individus dans chaque population et la probabilité de mutation.

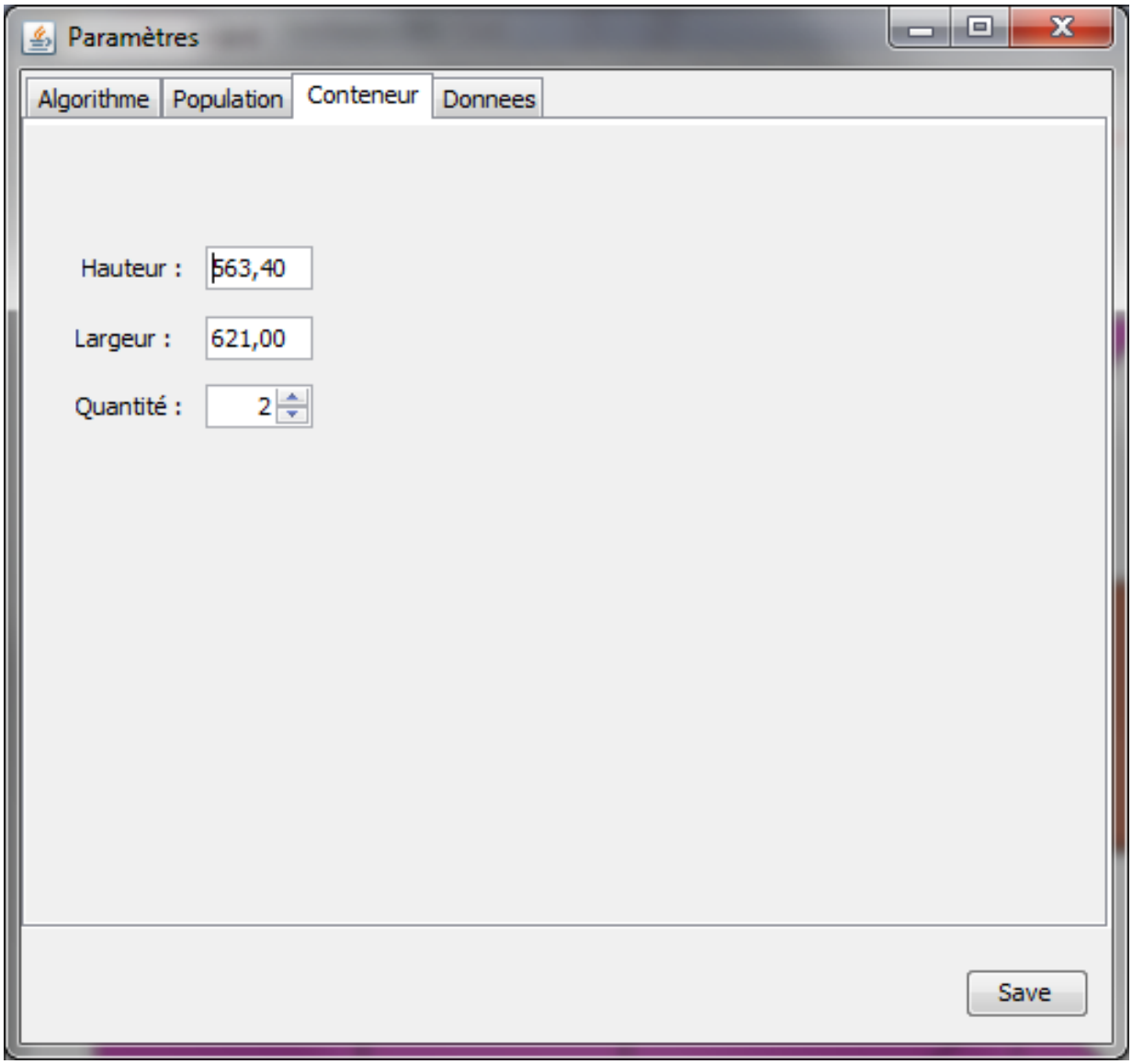


Figure 4.8: Affichage de la fenêtre paramètres permettant de modifier les données du conteneur

On peut changer dans cette fenêtre les paramètre du conteneur utilisé pour ranger les objets. On peut modifier la hauteur, la largeur et la quantité de conteneurs.

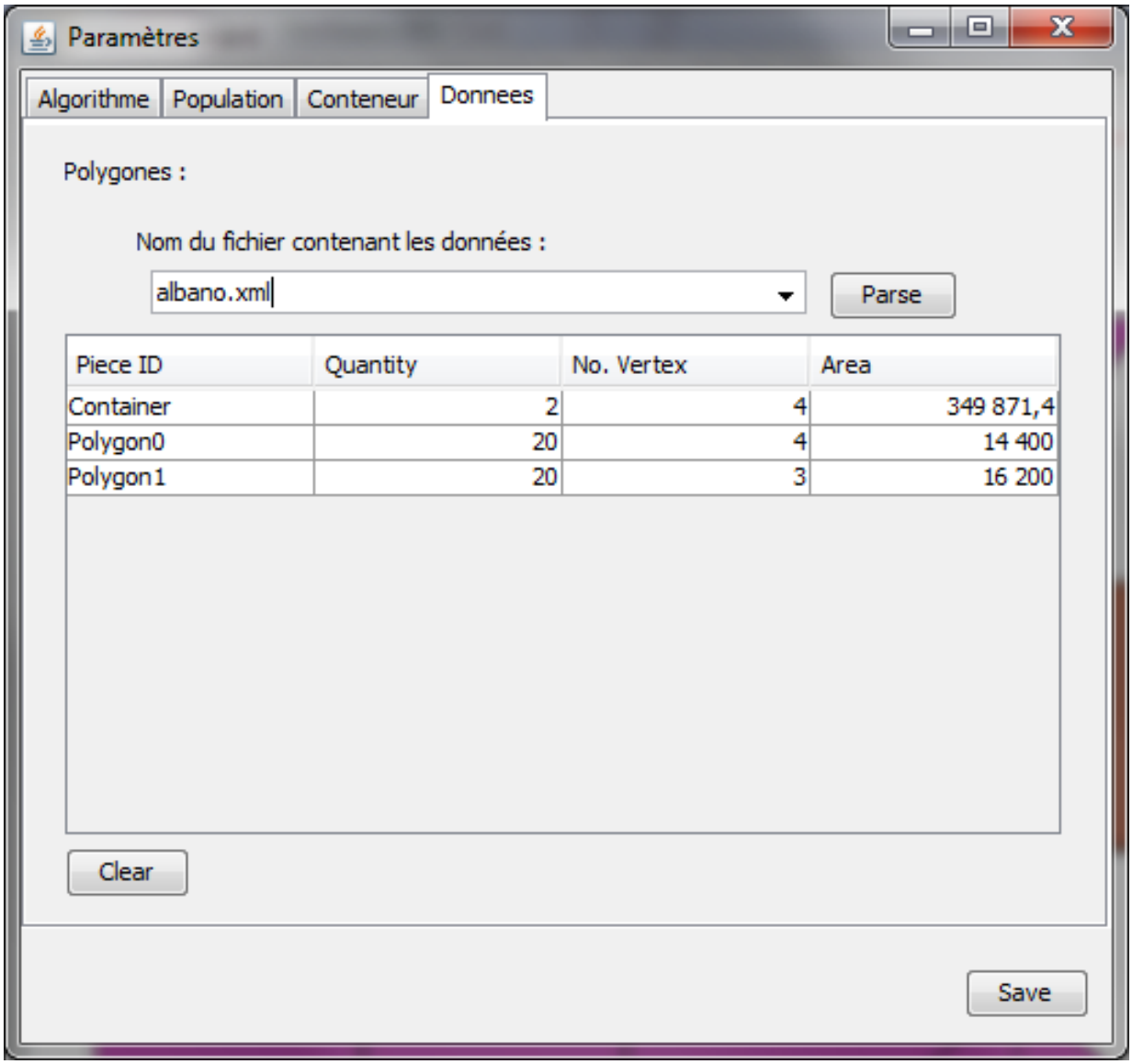


Figure 4.9: Affichage de la fenêtre paramètres permettant de définir les données de l'algorithme

On peut changer dans cette fenêtre les données du problème. On peut donner l'adresse d'un fichier .html contenant les définitions des différents objets et du conteneur. Dans le tableau affiché, on pourra voir les données lises dans le fichier. On peut voir qu'on peut voir le nombre de côtés que chaque polygone a, la quantité de polygones et l'air occupé par chacun.

5
Conclusion

Pendant les premiers jours de stage j'ai dû me mettre au niveau de ce que l'on attendait de moi, j'ai eu besoin de beaucoup de recherche, lecture et auto-formation. Même si cela s'explique, à mon avis, au fait que ce stage est basée plus dans le domaine de l'informatique, se fut une épreuve très fructueuse qui m'a permis d'acquérir les bases nécessaires pour ma mission.

Dans les semaines qui ont suivi, j'ai pu me familiariser d'avantage avec tout les méthodes de travail et je ne pense pas avoir passé un seul jour sans avoir appris quelque chose, et en plus sur différents domaines d'études. Je connais maintenant les algorithmes génétiques, les principes des algorithmes d'intelligence artificielle tels que : les réseaux de neurones, les colonies de fourmis, etc.

La partie pratique, m'a déjà permise d'apprendre le langage Java que je ne connaissait point et qui est souvent un requis pour certains emplois. Mais en plus c'est grâce à celle-ci que tous les concepts, termes et méthodes qui étaient abstraits dans mon esprit après l'étude d'état de l'art, se sont éclaircis. Je suis consciente que sans la partie étude théorique il aurait été impossible que je réalise ces tests et je suis impatiente d'apprendre de nouvelles méthodes et concepts pour pouvoir les appliquer dans l'avenir.

Des études reposant sur l'intelligence artificielle ont déjà été faits dès les années 70, voir les années 60. Néanmoins ça reste un domaine d'actualité et le laboratoire Quantup est loin d'être le seul laboratoire de recherche intéressé sur ce domaine.

6
Bibliographie

[1] Hannu Ahonen, Arlindo Alvarenga, Attilio Provedel, and Victor Parada. A client-broker-server architecture of a virtual enterprise for cutting stock applications. *International Journal of Computer Integrated Manufacturing*, 14(2):194–205, 2001.

[2] A. Ramesh Babu and N. Ramesh Babu. A generic approach for nesting of 2-d parts in 2-d sheets using genetic and heuristic algorithms. *Computer-Aided Design*, 33(12):879 – 891, 2001.

[3] Julia A. Bennell and Jose F. Oliveira. The geometry of nesting problems: A tutorial. *European Journal of Operational Research*, 184(2):397 – 415, 2008.

[4] E. K. Burke, G. Kendall, and G. Whitwell. A New Placement Heuristic for the Orthogonal Stock-Cutting Problem. *Operations Research*, 52(4):655–671, July 2004.

[5] J.P. Cohoon and W.D. Paris. Genetic placement. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(6):956 – 964, november 1987.

[6] Türkay Dereli and Gülesin Sena Das. A hybrid simulated annealing algorithm for solving multi-objective container-loading problems. *Applied Artificial Intelligence*, 24(5):463–486, 2010.

[7] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inform. Process. Lett.*, 12(3):133–137, 1981.

[8] Wei Han, Julia A. Bennell, Xiaozhou Zhao, and Xiang Song. Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints, 2012.

[9] E. Hopper and B.Turton. Application of genetic algorithms to packing problems - a. review. *Proceedings of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing, Springer Verlag*, pages 279–288, 1997.

[10] Shian-Miin Hwang, Cheng-Yan Kao, and Jorng-Tzong Horng. On solving rectangle bin packing problems using genetic algorithms. In *Systems, Man, and Cybernetics, 1994. 'Humans, Information and Technology'., 1994 IEEE International Conference on*, volume 2, pages 1583 –1590 vol.2, oct 1994.

[11] Takashi Imamichi, Yohei Arahori, Jaeseong Gim, Seok-Hee Hong, and Hiroshi Nagamochi. Removing node overlaps using multi-sphere scheme. In Ioannis Tollis and Maurizio Patrignani, editors, *Graph Drawing*, volume 5417 of *Lecture Notes in Computer Science*, pages 296–301. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-00219-9_28.

[12] Takashi Imamichi and Hiroshi Nagamochi. Designing algorithms with multi-sphere scheme. In *Informatics Education and Research for Knowledge-Circulating Society, 2008. ICKS 2008. International Conference on*, pages 125 –130, jan. 2008.

[13] Takashi Imamichi, Mutsunori Yagiura, and Hiroshi Nagamochi. An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem. *Discrete Optimization*, 6(4):345 – 361, 2009.

[14] S. Jakobs. On genetic algorithms for the packing of polygons. *European Journal of Operational Research*, 88(1):165–181, January 1996.

[15] Andrew B. Kahng and Byung Ro Moon. Toward more powerful recombinations. In Larry J. Eshelman, editor, *ICGA*, pages 96–103. Morgan Kaufmann, 1995.

[16] Stephen C. H. Leung and Defu Zhang. A new heuristic approach for the stock-cutting problems. *World Academy of Science, Engineering and Technology*, (53), 2009.

[17] J. Levine and F. Ducatelle. Ant Colony Optimization and Local Search for Bin Packing and Cutting Stock Problems. *The Journal of the Operational Research Society*, 55(7):705–716, 2004.

[18] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, September 2002.

[19] J. Martens. Two genetic algorithms to solve a layout problem in the fashion industry. *European Journal of Operational Research*, 154(1):304–322, April 2004.

[20] Arfath Pasha. *Geometric Bin Packing Algortihm for Arbitrary Shapes*. PhD thesis, University of Florida, 2003.

[21] K. Rose. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proceedings of the IEEE*, 86(11):2210–2239, November 1998.

[22] Solly Andy Segenreich and Leda Maria P. Faria Braga. Optimal nesting of general plane figures: A monte carlo heuristical approach. *Computers amp; Graphics*, 10(3):229 – 237, 1986.

[23] Yurij Stoyan, Guntram Scheithauer, Nikolay Gil, and Tatiana Romanova. functions for complex 2d-objects. *4OR A Quarterly Journal of Operations Research*, 2(1):69–84, 2004.

[24] S. Szykman and J. Cagan. A simulated annealing-based approach to 3d component packing, 1995.

[25] H. Terashima-Marín, P. Ross, C. Farías-Zárate, E. López-Camacho, and M. Valenzuela-Rendón. Generalized hyper-heuristics for solving 2d regular and irregular packing problems. *Annals of Operations Research*, 179:369–392, 2010. 10.1007/s10479-008-0475-2.

[26] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130, December 2007.

[27] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, June 1994.