APPRENTISSAGE ET OPTIMISATION

Comparaison de différentes techniques d'optimisation pour l'apprentissage non-supervisé

Alexandre Bargeton, Benjamin Devèze
Université Pierre et Marie Curie, Master IAD, rapport de projet de recherche (PRREC)

RÉSUMÉ **Ce rapport traite des différentes techniques d'optimisation pour l'apprentissage non-supervisé. Il dresse un rapide état de l'art des principales techniques de clustering. Une part importante est consacrée aux métaheuristiques d'optimisation. Nous explicitons ensuite comment il a été proposé d'exploiter ces techniques d'optimisation dans le cadre de l'apprentissage non-supervisé. Nous présentons également nos expérimentations et exposons quelques résultats empiriques. Nous dressons enfin un bilan général insistant sur les avantages et les inconvénients des techniques présentées.**

Mots-clés : apprentissage ; clustering ; optimisation ; métaheuristique ; recuit ; tabou ; génétique

INTRODUCTION

Nous vivons dans un monde empli de données. C'est ainsi que, quotidiennement, les sociétés humaines génèrent et véhiculent des masses considérables d'information dont la plus grande part reste inexploitée. Avec l'augmentation des capacités de stockage et de traitement de l'information, l'analyse intelligente de données s'avère un domaine de plus en plus fondamental.

La classification est une tâche effectuée quotidiennement par les êtres humains. C'est ainsi que pour apprendre un concept ou comprendre de nouveaux phénomènes, nous tentons d'en extraire les caractéristiques les plus saillantes, avant de les comparer avec les concepts et phénonèmes connus qui nous semblent les plus proches, au sens d'un critère de similarité basé sur leurs caractéristiques. Il s'agit donc d'une problématique qui se trouve au coeur de l'analyse de grandes masses de données, ce qui justifie pleinement l'intérêt qui lui est porté.

Nous dresserons dans une première section un rapide état de l'art des méthodes de *classification automatique dans le cadre de l'apprentissage non-supervisé*. Nous ne prétendons évidemment pas à l'exhaustivité mais tenterons de dégager les éléments essentiels en nous efforçant de renvoyer le lecteur intéressé aux références bibliographiques pertinentes. Une seconde section abordera le domaine, non moins vaste, des *métaheuristiques d'optimisation*. Nous mettrons en avant leurs intérêts, leurs points communs mais également leurs limitations en nous attardant plus particulièrement sur les techniques les plus classiques, à savoir, le *recuit simulé*, la *recherche tabou* ainsi que les *algorithmes génétiques*. Forts de ces informations, nous verrons, dans une troisième section, comment il a été proposé d'utiliser ces techniques de recherche locale pour améliorer les résultats des méthodes de classification automatique. Nous décrirons dans une quatrième section les expérimentations que nous avons conduites sur cette question. Nous dresserons enfin un bilan de notre étude.

LE CLUSTERING

La classification automatique peut être définie comme regroupant l'ensemble des méthodes visant à découper un ensemble d'objets en plusieurs classes en fonction des variables qui les décrivent. Il faut immédiatement ici faire la distinction entre *apprentissage supervisé* et *apprentissage non supervisé*, qui seul nous intéressera dans le présent rapport.

Dans le cadre de l'apprentissage supervisé, l'objectif est d'apprendre un modèle de classification à partir d'une base d'apprentissage étiquetée, i.e. la classe de chaque objet est connue a priori. Le modèle doit ensuite être capable de calculer la sortie la plus vraisemblable sur de nouvelles entrées.

Dans le cadre de l'apprentissage non-supervisé, il s'agit d'induire de nouvelles informations sans ces connaissances, simplement à partir d'une base d'exemples. La classification automatique (*clustering*) s'intéresse au processus d'extraction d'une organisation, i.e. à la manière de classer les données observées au sein de structures dans lesquelles elles se regroupent "naturellement". L'objectif du clustering est de regrouper dans le même *cluster* les observations jugées similaires[1] (homogénéité intra-classe) et de placer les observations jugées dissimilaires dans des *clusters* distincts (hétérogénéité inter-classe).

Le clustering est un sujet actif de recherche qui se trouve au confluent de plusieurs disciplines dont les statistiques, l'apprentissage automatique, le datamining, le textmining. Il intervient dans des domaines très variés : médecine, biologie, chimie, géologie, sociologie, économie, assurances, marketing et bien d'autres. Il peut être utilisé pour lui même ou également comme prétraitement d'autres algorithmes, c'est le cas par exemple en traitement de la parole pour l'apprentissage de modèles de markov cachés.

Le *clustering conceptuel* est une amélioration des techniques de clustering qui vise à fournir une caractérisation des clusters à l'aide de concepts. Cela permet de raisonner à partir des clusters construits et peut être utilisé afin de classer de nouvelles observations.

Des informations beaucoup plus détaillées ainsi que des compléments bibliographiques pourront être trouvés dans

[1] au regard d'une certaine métrique

[JMF99, HBV01, Ber02, XW05, TSK05].

DÉFINITIONS ET NOTATIONS Afin de fixer le contexte dans lequel on se place et de clarifier les terminologies employées, nous allons tout d'abord introduire quelques définitions et notations.

Nous considérons un ensemble de données d'entrée X constituées de points (également référencés sous les termes objets, instances, exemples) $x_i = (x_{i1}, ..., x_{id}) \in A$, où i varie de 1 à N (nombre d'exemples). A est l'espace des attributs. Chaque composante $x_{il} \in A_l$, où A_l est un attribut numérique ou symbolique. Les attributs symboliques ou catégoriels recouvrent un ensemble limitativement énumérable de valeurs symboliques avec éventuellement une notion d'ordre. Evidemment il convient d'adapter les notions de distance à ce type d'attributs. Les attributs numériques peuvent être à valeurs entières ou réelles.

Le but du clustering est d'assigner les points de X à un nombre fini de k ensembles, les clusters notés $C_1, ..., C_k$. Généralement ces k ensembles forment une partition de X.

Pour ce faire, il est souhaitable de développer des méthodes flexibles, pouvant traiter d'importantes quantités de données, capables de traiter les données bruitées et lacunaires, efficaces et fournissant des résultats interprétables.

DISTANCES ET MESURES DE SIMILARITÉ La notion de ressemblance, de similarité est au coeur du problème de classification automatique. Les mesures utilisées pour calculer la similude entre deux objets sont légion et il n'existe pas de règle pour choisir les mesures les plus appropriées. La littérature abondant sur le sujet, nous ne redéfinissons pas ici formellement ce que sont des mesures de ressemblance, de similarité, de dissimilarité ou des distances.

Nous nous contenterons de mentionner qu'il existe un grand nombre de mesures opérant sur les données catégorielles dont notamment celles de *Russel et Rao*, de *Kendall, Sokal et Michener* ainsi que celle de *Jaccard*.

Les mesures de similarité et de dissimilarité pour les attributs numériques sont encore plus nombreuses et sont également plus ou moins adaptées à chaque application. Citons la distance de *Minkowski*, la classique distance *Euclidienne*, la distance de *Mahalanobis*, la *corrélation de Pearson* ou encore la *similarité cosinus*.

ÉVALUATION DU CLUSTERING Il est essentiel de pouvoir évaluer le résultat des méthodes de clustering. Cela passe par une évaluation de la qualité des classes produites mais également de la pertinence des représentants de ces classes. Cela permet notamment de régler les méthodes et de s'assurer que les mesures utilisées sont bien adaptées. La procédure d'évaluation du résultat d'un algorithme de clustering est connue sous le nom de *cluster validity*.

La première méthode pouvant être utilisée est celle de la *conformation de classe*. L'objectif du clustering est alors de retrouver automatiquement un étiquetage connu des objets. On peut dès lors très simplement évaluer l'erreur de classification par rapport à la catégorisation idéale. Il est également possible d'utiliser des jeux de données créés artificiellement.

Des mesures basées sur l'entroprie de l'information ont également été proposées par Boley. Il s'agit notamment de la mesure de l'*entroprie des clusters*. Cette mesure ne prenant pas en compte le nombre de clusters produits, il est nécessaire de lui associer une mesure d'entropie complémentaire appelée entropie de classe qui mesure la manière dont une même classe est représentée au sein des différents clusters. On utilise ces deux mesures pour évaluer l'entropie globale. Il faut pour cela pondérer chaque valeur par un coefficient de compromis entre entropie de cluster globale et entropie de classe globale. Ces mesures peuvent être utilisées pourvu que les données d'entrée soit proprement étiquetées.

De nombreux indices de validité ont été proposés lorsqu'aucun étiquetage préalable n'est disponible. Ils se basent sur la recherche d'un compromis entre similarité intra-classe et dissimilarité inter-classe. Citons notamment la mesure de *compacité-séparation*. Certains indices, comme l'*indice de Dunn* ne tiennent compte que de l'organisation des exemples au sein des clusters et négligent la représentation de ces clusters ce qui n'est pas satisfaisant particulièrement dans le cadre du clustering conceptuel.

ÉTAT DE L'ART Nous présentons ici un état de l'art très succint des algorithmes les plus classiques du domaine. Nous avons regroupé ici les différentes techniques suivant une classification relativement classique dans la littérature du domaine. Cette taxinomie n'est évidemment pas unique et repose sur certains grands concepts dont voici les principaux :

- méthodes *agglomératives* (ascendantes) contre *discriminatives* (descendantes). Une méthode agglomérative commence à travailler sur des clusters d'un point et regroupe récursivement les clusters les plus proches. Au contraire, une méthode discriminative commence avec un cluster contenant tous les points qu'elle sépare récursivement. Le processus s'arrête lorsqu'un critère d'arrêt est atteint, fréquemment lorsque le nombre de clusters requis a été obtenu.
- *monothétique* contre *polythétique*. La plupart des algorithmes sont polythétiques c'est à dire que tous les attributs entrent dans le calcul de distance entre deux exemples et les décisions sont basées sur ces distances. D'un autre côté, il est possible de considérer les attributs séquenciellement afin de progressivement diviser les exemples sur chaque composante.
- *hard clustering* contre *fuzzy clustering*. Les méthodes de *hard clustering* allouent chaque exemple à un seul cluster. Les méthodes de *fuzzy clustering* assignent des degrés d'appartenance de chaque exemple aux clusters.
- méthodes *déterministes* contre *stochastiques*.
- méthodes *incrémentales* contre *non incrémentales*.

Ces concepts peuvent évidemment se recouvrir.

Clustering hiérarchique

Les algorithmes de clustering hiérarchique construisent une hiérarchie de clusters (dendogrammes...). Pour ces algorithmes il est nécessaire d'étendre la distance entre deux points à une distance entre ensembles de points. Ces distances ont une grande influence sur les résultats finaux. La distance entre deux ensembles de points implique un calcul entre chaque pair de points de chacun des deux ensembles. On utilise couramment la *distance minimum*, *maximum* ou une *distance moyenne*.

De très nombreuses variantes ont été proposées, citons : *SLINK*, *CLINK*, *AGNES*, *CURE*, *CHAMELEON*.

Dans ce paysage, l'algorithme *COBWEB* proposé par D. Fisher en 1987 se singularise. Il doit en partie sa popularité à deux propriétés intéressantes. Tout d'abord, il est incrémental et ainsi peut construire un dendogramme incrémentalement en traitant les données les unes après les autres. Ensuite, il s'agit d'un algorithme de clustering conceptuel. Chaque cluster peut ainsi être considéré comme un modèle pouvant être décrit intrinsèquement plutôt que par un ensemble de points lui appartenant. Il effectue une recherche à travers un espace contenant des clustering hiérarchiques en utilisant un ensemble d'opérateurs. L'objectif est d'obtenir une hiérarchie de classes étiquetées par des concepts probabilistes. Un concept probabiliste représente chaque classe d'objets à l'aide d'un ensemble de probabilités conditionnelles sur les valeurs d'attributs. Cette recherche est guidée par une mesure heuristique appelée *Category Utility*. Cette mesure peut être vue comme un compromis entre la similarité intra-classe et la dissimilarité inter-classe. Nous n'entrons pas ici dans le détail des équations. L'algorithme considère chaque objet de manière incrémentale et l'insère dans un arbre de classification. Il est dès lors possible d'obtenir une partition des objets à partir des classes situées aux noeuds de l'arbre. Les principaux problèmes rencontrés par COBWEB sont sa grande sensibilitée à l'ordre d'entrée des données et d'autre part la stratégie de recherche locale utilisée qui ne donne aucune garantie sur la qualité du résultat. Il a donné lieu à plusieurs modifications dont notamment *CLASSIT* et *ARACHNE*.

Les avantages du clustering hiérarchique sont sa flexibilité, la possibilité d'utiliser toute forme de similarité ou distance et donc de pouvoir s'appliquer à tous les types d'attribut.

Clustering de partitionnement

Il s'agit ici de construire une partition de l'ensemble des objets de départ. Tester tous les sous-ensembles possibles serait beaucoup trop gourmand en temps de calcul pour pouvoir être envisagé. De ce fait, un certain nombre d'heuristiques gourmandes sont utilisées sous la forme de méthodes d'optimisation itérative. En pratique, cela se traduit par des réallocations itératives des points entre les clusters. Contrairement aux méthodes hiérarchiques classiques, dans lesquels les clusters ne sont pas révisés, ces réallocations améliorent graduellement les clusters construits.

La première approche possible est une *approche probabiliste*. Dans ce contexte, on considère que les données ont été générées par un mélange de modèles statistiques dont on cherche à déterminer les paramètres. On suppose que les points ont été produits en choisissant aléatoirement un modèle puis en utilisant la distribution de ce modèle. Dès lors, il est possible d'associer les paramètres (moyenne, variance, ...) de chaque modèle à un cluster. Il est possible d'estimer la probabilité qu'un point ait été généré par un cluster. La vraisemblance des données d'entrée correspond alors à la probabilité qu'elles aient été généré par le mélange de modèles. Pour des raisons pratiques, on utilise la log-vraisemblance qui devient la fonction objectif à maximiser. C'est le domaine de l'algorithme *Expectation Maximization (EM)*. Il s'agit d'un algorithme d'optimisation itératif qui procède en deux phases L'étape E (Expectation) calcule la valeur estimée de la vraisemblance à partir des paramètres courants, l'étape M (Maximisation) calcule de nouveaux paramètres collant plus fidèlement aux données que les précédents. Il convient d'utiliser des modèles adaptées aux types des données fournies en entrée. L'algorithme présente une complexité en temps linéaire par rapport au nombre d'exemples à classer. Il ne nous assure cependant pas que la solution retournée soit l'optimum global. C'est une méthode bien adaptée au traitement de données cachées. Nous renvoyons à [Bil98, Min98] pour plus de détails sur EM. *SNOB*, *AUTOCLASS*, *MCLUST* sont d'autres algorithmes basées sur une approche probabiliste.

Une deuxième approche repose sur la définition d'une fonction objectif à optimiser. Les algorithmes de partitionnement les plus classiques du domaine sont les algorithmes de type *k-medoids*, *k-means* et *k-modes* qui ont servi de base à de très nombreuses variantes.

Dans les méthodes de type k-medoids chaque cluster est représenté par l'un de ses points. Cela permet de traiter n'importe quel type d'attributs et d'avoir une certaine tolérance au bruit. Lorsque les medoïdes sont séléctionnés, les clusters sont définis comme les sous-ensembles de points "proches" de chaque medoid et la fonction objectif est définie par une mesure de dissimilarité entre un point et son medoïde. L'optimisation de l'objectif se fait par réallocation successive des medoïdes et des points qui s'y rattachent. Les algorithmes *PAM*, *CLARA* et *CLARANS* appartiennent à cette famille.

La famile d'algorithmes de type *k-means* est sans doute l'une des plus populaires et des plus utilisées. Son nom vient du fait que chacun des k clusters générés est représenté par un centroïde qui n'est autre que la moyenne pondérée des points qui le composent. L'idée initiale est d'utiliser un processus itératif afin de minimiser la somme des carrés des distances entre les exemples et les centroïdes des clusters. Pour ce faire la plupart des techniques reposent sur deux étapes : une étape d'allocation et une étape de recentrage. En ce sens la méthode est très proche de l'algorithme EM. Durant la première phase, les centroïdes caractérisant chacune des classes sont calculés. Ensuite, chaque exemple peut être réparti au sein des différents clusters en fonction de sa distance aux centroïdes. On répète ces deux étapes jusqu'à convergence. Il s'agit d'un cas particulier des *nuées dynamiques* [Did71]. De nombreux algorithmes basés sur cette technique ont vu le jour. On peut notamment citer *k-modes* qui permet d'étendre le principe au traitement de données catégorielles, et non simplement aux données numériques. Pour ce faire, il suffit d'adapter la mesure de dissimilarité et de considérer la valeur la plus fréquente (mode) de chaque attribut catégorielle plutôt que sa moyenne. La complexité de ces méthodes permet de les exploiter pour des données de taille conséquente. La complexité en temps de k-modes est en effet en $O(Nkl)$ où l est le nombre d'itérations nécessaire à la convergence. Toutefois, il est nécessaire de préciser explicitement le nombre de clusters à générer et la stratégie de recherche, bien que rapide, peut converger vers un optimum local parfois notablement sous-optimal.

Clustering basé sur la densité

Les algorithmes basés sur la densité sont capables de mettre à jour des clusters de toute forme et d'ignorer le bruit. Ils reposent sur une notion de densité et de connexité

entre les points. Un cluster pouvant alors être assimilé à une composante connexe définie selon une mesure de densité. Intuitivement un cluster est constitué par un point et ses voisins. Ces méthodes sont capables de traiter correctement certaines données qui posent problème à des algorithmes tels que k-means et sont efficaces sur des entrées de taille conséquente. Les groupes générés apportent toutefois peu d'information et sont difficilement interprétables tel quel.

A partir des concepts de densité et de connexité définis en terme de distribution locale de plus proches voisins, ont été déclinées de nombreux algorithmes tels que *DBSCAN*, *GDBSCAN*, *OPTICS*, *DBCLASD*... Il s'agit tout simplement de définir une distance ϵ définissant le voisinage et un entier $MinPoints$. Ensuite deux points sont dans la même composante si ils sont atteignables en considérant une suite de voisinage ayant tous au moins $MinPoints$ points. Il s'agit d'une relation symétrique et tous les points de la même composante forment un cluster. Les points non attribués, car trop isolés, sont considérés comme du bruit. Nous ne jugeons pas utile de développer outre mesure cette partie qui est assez éloignée de nos préoccupations.

Autres méthodes

Bien d'autres méthodes de clustering ont été développées. Nous les mentionnons brièvement ici.

Les problèmes concrets font régulièrement intervenir un certain nombre de *contraintes*. Le clustering n'y échappe pas, c'est ainsi que les méthodes doivent être capables de gérer des contraintes propres à chaque problème. Il peut par exemple être intéressant de fixer des contraintes sur le nombre minimum de points dans chaque cluster, sur le nombre de clusters, ... Dans cette optique, des recherches récentes ont conduit à des variantes plus flexibles de l'algorithme k-means dont notamment le *frequency sensitive k-means*.

Des méthodes utilisant des *réseaux de neurones* et des *descentes de gradient* ont également été étudiées.

Les méthodes faisant intervenir des *métaheuristiques d'optimisation* seront développées dans une partie ultérieure du présent rapport, mais avant cela nous allons présenté ces techniques d'optimisation, susceptibles de jouer un rôle dans ces méthodes.

LES MÉTAHEURISTIQUES D'OPTIMISATION

De nombreux problèmes pratiques s'expriment comme des problèmes d'optimisation. On se donne alors une fonction objectif et l'on cherche les paramètres qui la maximisent ou la minimisent éventuellement sous des contraintes données. Nous nous intéressons ici à un groupe de méthodes, dénommées *métaheuristiques* ou *méta-heuristiques*, apparues à partir des années 1980 avec une ambition commune : résoudre au mieux les problèmes dits d'*optimisation difficile*. Nous verrons que les métaheuristiques reposent sur un ensemble commun de principes qui permettent de concevoir des algorithmes de résolution pour les problèmes d'optimisation. Une étude complète du domaine ainsi que des références bibliographiques nombreuses pourront être trouvées dans [DPST03] et [Sev04], dont la présente section constitue une synthèse. Il nous est, en effet, apparu judicieux de dresser un état de l'art du domaine afin de bien rappeler de quoi il retourne et de pouvoir s'appuyer sur cet exposé lors de la présentation de nos expérimentations. Il s'agit tout d'abord de comprendre ce qui se cache derrière le terme d'optimisation difficile.

OPTIMISATION DIFFICILE Il faut tout d'abord faire la distinction entre les problèmes discrets et les problèmes à variables continues. Nous nous intéresserons aux problèmes d'optimisation sur domaines discrets qui sont ceux qui interviennent dans le domaine de l'apprentissage non-supervisé qui est au coeur de notre étude.

Dans le *domaine discret*, l'optimisation difficile fait référence à certains problèmes combinatoires pour lesquels on ne connaît pas d'algorithme exact polynomial. C'est, en particulier, le cas des problèmes dits NP-difficiles, pour lesquels il est conjecturé qu'ils ne peuvent être résolus par un algorithme dont le temps d'exécution est borné par un polynôme en la taille de l'entrée du problème. Pour traiter ces problèmes, un grand nombre d'*heuristiques*, produisant des solutions proches de l'optimum, ont ainsi été développées ; toutefois la plupart d'entre elles ont été conçues spécifiquement pour un problème donné, ce qui limite fortement leur champs d'application.

Dans le *domaine continu*, il existe un grand nombre de méthodes dites d'optimisation globale, mais ces techniques reposent souvent sur des hypothèses concernant la fonction objectif (convexité, ...) ce qui limite leur utilisation.

PRINCIPALES CARACTÉRISTIQUES Au contraire, les métaheuristiques visent à s'appliquer à toutes sortes de problèmes discrets ou continus. Il est possible d'en dégager un certain nombre de traits communs :

- elles sont *stochastiques* ce qui leur permet de s'attaquer à l'explosion combinatoire des solutions possibles ;
- elles sont *directes*, elles ne font ainsi pas, par exemple, appel à des calculs de gradient de la fonction objectif ;
- elles sont souvent inspirées par des *analogies avec la nature*, la biologie (algorithmes génétiques), l'éthologie (colonies de fourmis) ou la physique (recuit simulé) ;
- elles s'appuient sur un équilibre entre *intensification* et *diversification* de la recherche. L'intensification permet de rechercher des solutions de meilleur qualité à partir des solutions déjà trouvées, la diversification permet d'explorer un plus grand espace de solutions et d'échapper à des minima locaux ;
- elles partagent les *mêmes défauts*.

Nous reviendrons plus en profondeur sur ces caractéristiques.

MÉTHODES DE RECHERCHE LOCALE Les méthodes de recherche locale ou métaheuristiques à base de voisinages s'appuient toutes sur le même principe. A partir d'une solution initiale c_0, la recherche consiste à passer d'une solution à une solution voisine par déplacements successifs. L'ensemble des solutions que l'on peut atteindre à partir d'une solution courante c est appelée *voisinage* de cette solution notée $\mathcal{N}(c)$. La définition du voisinage est évidemment dépendante du problème et conditionne de manière fondamentale l'efficacité des méthodes d'optimisation. Nous y reviendrons.

De manière générale, les algorithmes classiques d'amélioration itératives s'arrêtent lorsqu'une solution lo-

calement optimale est trouvée, c'est à dire lorsqu'il n'existe pas de meilleure solution dans le voisinage. Ceci conduit à un optimum local c_n de la fonction objectif comme représenté sur la figure 1 et n'est évidemment pas satisfaisant. C'est le cas des méthodes de descente présentées ci-dessous.

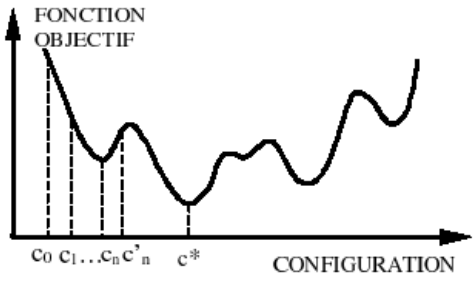

FIG. 1 – Allure d'une fonction objectif en fonction des configurations

Pour surmonter l'obstacle des minima locaux, il s'agit d'autoriser, de temps en temps, des mouvements de remontée, autrement dit d'accepter une dégradation temporaire de la situation, lors du changement de configuration. C'est le cas si l'on passe de c_n à c'_n. Un mécanisme de contrôle des dégradations, spécifique à chaque technique, permet d'éviter la divergence du procédé. Il devient dès lors possible de s'extraire des optima locaux pour éventuellement atteindre c^*.

Les méthodes de descente

A partir d'une solution initiale, par exemple aléatoire ou trouvée par heuristique, on peut très facilement implémenter des méthodes de descente. Ces méthodes s'articulent toutes autour d'un principe simple. Partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante. Leur intérêt est relativement limité mais nous avons décidé de les présenter car nous les avons implémentées pour les comparer avec les métaheuristiques plus évoluées.

L'algorithme 1 présente le squelette d'une méthode de descente simple. Il n'appelle pas de commentaires particuliers.

Algorithme 1 Descente Simple

1: trouver une solution initiale c
2: **repeat**
3: trouver une solution $c' \in \mathcal{N}(c)$
4: **if** $f(c') < f(c)$ **then** $c \leftarrow c'$
5: **end if**
6: **until** $f(c') \geq f(c), \forall c' \in \mathcal{N}(c)$

Une version plus brutale de la méthode de descente est la méthode de plus grande descente présentée ci-après.

Algorithme 2 Plus Grande Descente

1: trouver une solution initiale c
2: **repeat**
3: trouver une solution $c' \in \mathcal{N}(c) \ / f(c') \leq f(c''), \forall c'' \in \mathcal{N}(c)$
4: **if** $f(c') < f(c)$ **then** $c \leftarrow c'$
5: **end if**
6: **until** $f(c') \geq f(c), \forall c' \in \mathcal{N}(c)$

Evidemment ces méthodes très simples sont criticables et resteront bloquées dans des minima locaux. Elles ne présentent aucune forme de diversification. Un moyen très simple de diversifier la recherche peut consister à exécuter plusieurs fois l'un des algorithmes avec une solution initiale différente. Comme l'exécution de ces méthodes est souvent très rapide, on peut alors inclure cette répétition au sein d'une boucle générale. On obtient alors l'algorithme 3 dit de descente à départs multiples.

Algorithme 3 Descente à Départs Multiples

1: trouver une solution initiale c
2: $k \leftarrow 1$
3: $f(B) \leftarrow +\infty$
4: **repeat**
5: choisir aléatoirement une solution initiale c_0
6: $c \leftarrow$ résultat de descente simple ou de plus grande descente
7: **if** $f(c) < f(B)$ **then**
8: $B \leftarrow c$
9: **end if**
10: $k \leftarrow k + 1$
11: **until** critère d'arrêt satisfait

Cette technique très frustre peut malgré tout s'avérer payante pour certains problèmes.

Le recuit simulé

La méthode du recuit simulé a été introduite en 1983 par Kirkpatrick et al. dans [KGV83]. Cette méthode originale est basée sur les travaux antérieurs de Métropolis et al. [MRR+53]. Cette méthode que l'on pourrait qualifier de première métaheuristique "grand public" a reçu l'attention de nombreux travaux et a trouvé de nombreuses applications.

Le principe de fonctionnement s'inspire d'un processus d'amélioration de la qualité d'un métal solide par recherche d'un état d'énergie minimum correspondant à une structure stable de ce métal. L'état optimal correspondrait à une structure moléculaire régulière parfaite. En partant d'une température élevée où le métal serait liquide, on le refroidit progressivement en tentant de trouver le meilleur équilibre thermodynamique. Chaque niveau de température est maintenu jusqu'à obtention d'un équilibre. Dans ces phases de température constante, on peut passer par des états intermédiaires du métal non satisfaisants, mais conduisant à la longue à des états meilleurs.

La méthode du recuit simulé transpose le procédé du recuit à la résolution d'un problème d'optimisation : la fonction objectif du problème, analogue à l'énergie d'un matériau, est alors minimisée, moyennant l'introduction

d'une température fictive, qui est un simple paramètre de contrôle de l'algorithme.

En pratique, la technique exploite le critère de Metropolis, qui permet de décrire le comportement d'un système en équilibre thermodynamique à une certaine température T : partant d'une configuration donnée, on fait subir au système une modification élémentaire ; si cette transformation a pour effet de diminuer la fonction objectif du système, elle est acceptée ; si au contraire elle provoque une augmentation ΔE de la fonction objectif, elle peut être acceptée avec la probabilité $e^{-\frac{\Delta E}{T}}$. On itère ensuite ce procédé, en gardant la température constante, jusqu'à ce que l'équilibre thermodynamique soit atteint, soit concrètement au bout d'un nombre "suffisant" de modifications. On abaisse alors la température, avant d'effectuer une nouvelle série de modifications. L'algorithme 4 en présente la structure globale.

Algorithme 4 Recuit Simulé

trouver une solution initiale c
fixer un schéma de refroidissement τ
fixer la température initiale T
repeat
    trouver une solution $c' \in \mathcal{N}(c)$
    $\Delta E = f(c') - f(c)$
    tirage aléatoire (uniforme) de r dans $[0, 1]$
    if $\Delta E < 0$ ou $e^{-\frac{\Delta E}{T}} > r$ **then**
        $c \leftarrow c'$
    end if
    mettre à jour T selon τ
until critère d'arrêt satisfait

On comprend donc comment dans cet algorithme l'équilibre entre intensification et diversification est respecté. On intensifie en acceptant tous les mouvements strictement améliorants. La diversification se fait par l'acceptation de mouvements non améliorants. Notons que la diversification diminue avec la baisse de la température au fur et à mesure du déroulement de l'algorithme, réduisant par la même la probabilité d'acceptation de mouvements dégradants.

De l'exposé précédent, on peut dégager les principales caractéristiques de la méthode. En ce qui concerne ses *avantages* : on observe que la méthode procure généralement une solution de bonne qualité (minimum absolu ou bon minimum relatif de la fonction objectif). En outre c'est une méthode générale : elle est applicable et facile à programmer pour tous les problèmes qui relèvent des techniques d'optimisation itérative, pourvu que l'on puisse évaluer directement et rapidement, après chaque transformation, la variation correspondante de la fonction objectif. Enfin elle offre une grande souplesse d'emploi, car de nouvelles contraintes peuvent être facilement incorporées après coup dans le programme. Notons que la convergence du recuit simulé vers un optimum global, en choisissant le schéma adapté, a pu être prouvée [AvL85]. Plus précisément le principal résultat établit que, sous certaines conditions, le recuit simulé converge en probabilité vers un optimum global, en ce sens qu'il permet d'obtenir une solution arbitrairement proche de cet optimum, avec une probabilité arbitrairement proche de l'unité. Elle a donné d'excellents résultats pour nombre de problèmes (en robotique, médecine, géologie...), le plus souvent de grande taille.

En ce qui concerne ses *inconvénients*, les utilisateurs peuvent être rebutés par le nombre important de paramètres (température initiale, taux de décroissance de la température, durée des paliers, critères d'arrêt). La loi de décroissance par paliers de la température est souvent empirique, tout comme le critère d'arrêt du programme. Le schéma de refroidissement de la température est une des parties les plus difficiles à régler. Notons, toutefois, que les valeurs standards publiées permettent généralement un fonctionnement satisfaisant de la méthode. Ces schémas sont cruciaux pour l'obtention d'une implémentation efficace. Sans être exhaustif, on rencontre habituellement trois grandes classes de schéma : la réduction par paliers, la réduction continue et la réduction non monotone. Le deuxième défaut de la méthode est le temps de calcul qui peut s'avérer excessif, c'est pourquoi des versions parallèles de l'algorithme ont été proposées.

Quelques variantes

Nous présentons ici deux variantes du recuit simulé : la méthode du *grand déluge* et la méthode du *voyage de record en record*. Il en existe bien d'autres.

La méthode du grand déluge [Dru93], ainsi que la méthode du voyage de record en record, sont des méthodes de maximisation de la fonction objectif. Les différences se situent au niveau des lois d'acceptation des solutions dégradantes. De plus, si la méthode du recuit simulé nécessite, comme nous l'avons vu, le choix délicat d'un certain nombre de paramètres, ces deux méthodes apparaissent plus simples d'utilisation puisqu'elles n'ont que deux paramètres.

La méthode du grand déluge est présentée par l'algorithme 5.

Algorithme 5 Grand Déluge

trouver une solution initiale c
initialiser la quantité de pluie $UP > 0$
initialiser le niveau d'eau $WL > 0$
repeat
    trouver une solution $c' \in \mathcal{N}(c)$
    if $f(c') > WL$ **then**
        $c \leftarrow c'$
        $WL \leftarrow WL + UP$
    end if
until critère d'arrêt satisfait

L'allégorie du grand déluge permet de comprendre le mécanisme intuitif de cette méthode, pour garder les pieds au sec, le randonneur va visiter les points culminants de la région explorée. Alors que le niveau d'eau ne fait que croître, un inconvénient apparaît, celui de la séparation des terres, qui devrait piéger l'algorithme dans des maxima locaux. L'auteur de la méthode présente cependant des résultats tout à fait comparables à ceux obtenus par le recuit simulé.

La méthode du voyage de record en record est présentée sur l'algorithme 6. Dans cette méthode n'importe quelle solution peut être acceptée, du moment qu'elle n'est pas "beaucoup plus mauvaise" que la meilleure valeur record obtenue précédemment.

Algorithme 6 Record en Record

```
trouver une solution initiale c
initialiser l'écart autorisé DEVIATION > 0
évaluer le record initial RECORD = f(c)
repeat
    trouver une solution c' ∈ N(c)
    if f(c') > RECORD − DEVIATION then
        c ← c'
    end if
    if f(c) > RECORD then
        RECORD ← f(c)
    end if
until critère d'arrêt satisfait
```

On retrouve une forte similitude avec la méthode précédente. Il n'y a que deux paramètres à régler, i.e. l'écart *deviation* et le critère d'arrêt. Le choix du premier paramètre est important puisqu'il traduit un compromis entre la vitesse de convergence et la qualité du maximum obtenu.

L'auteur de la méthode mentionne que les résultats de ces deux méthodes sur le problème du voyageur de commerce seraient supérieurs à ceux procurés par le recuit simulé pour des tailles supérieures à 400 villes.

La recherche tabou

Dans un article présenté par Glover [Glo86] en 1986, on voit apparaître pour la première fois le terme *tabu search*. Hansen, présenta à la même époque une méthode similaire, mais dont le nom n'a pas autant marqué. En fait, les balbutiements de la méthode avaient été présentés dès la fin des années 1970 par Glover. Ce sont les travaux de ce dernier, qui vont contribuer de manière importante au succès de la méthode, qui, pour certains chercheurs apparaît même plus satisfaisante sur le plan scientifique que le recuit simulé.

Sa principale particularité tient dans l'utilisation d'une mémoire. La méthode tabou prend, sur ce plan, le contrepied du recuit simulé qui en est totalement dépourvu et s'avère donc incapable de tirer les leçons du passé. En revanche, la modélisation de la mémoire introduit de multiples degrés de liberté qui s'opposent, de l'avis même de Glover, à toute analyse mathématique rigoureuse.

Le principe de base de la méthode tabou est simple : comme le recuit simulé, la méthode fonctionne avec une seule configuration courante à la fois, celle-ci est actualisée au cours des itérations successives. Cependant, contrairement au recuit simulé, qui ne génère qu'une seule solution aléatoire c' dans le voisinage $\mathcal{N}(c)$ de la solution courante c, la méthode tabou, dans sa forme de base, examine le voisinage $\mathcal{N}(c)$ de la solution courante.

A chaque itération, le mécanisme de passage d'une configuration c à la suivante c' comporte deux étapes :

- on construit l'ensemble des voisins de c, i.e. l'ensemble des configurations accessibles en un seul mouvement élémentaire à partir de c. Si cet ensemble $\mathcal{N}(c)$ est trop vaste, on applique une technique de réduction de sa taille, par exemple on a recours à une liste de candidats, ou on extrait aléatoirement un sous-ensemble de voisins de taille fixée. Soit $\mathcal{V}(c)$ l'ensemble ou le sous-ensemble de ces voisins ;
- on évalue la fonction objectif f du problème en chacune des configurations de $\mathcal{V}(c)$. La configuration c' qui succède à c est la configuration de $\mathcal{V}(c)$ qui prend la valeur minimale en f. Notons que cette configuration c' est adoptée même si elle est moins bonne que c. C'est grâce à cette particularité que la méthode tabou permet d'éviter le piégeage dans des optima locaux.

On aura compris que telle quelle, la procédure est inopérante, car il y a un risque important de retourner à une configuration déjà retenue lors d'une itération précédente, ce qui engendre un cycle. Pour pallier ce problème, on tient à jour, à chaque itération, une liste de mouvements interdits, la *liste tabou*. L'algorithme modélise ainsi une forme rudimentaire de mémoire, la *mémoire à court terme* des solutions visitées récemment. Il est courant de ne garder dans la liste tabou que des informations portant soit sur les caractéristiques des solutions, soit sur les mouvements. Avec ce type de mémoire à court terme, il est possible qu'une solution de meilleure qualité que toutes celles déjà rencontrées ait un statut tabou. Accepter malgré tout cette solution revient à outrepasser son statut tabou, c'est l'application du *critère d'aspiration*.

Deux mécanismes d'intensification et de diversification sont souvent mis en oeuvre pour doter la méthode d'une *mémoire à long terme*. Ces processus n'exploitent plus la proximité dans le temps d'évènements particulier, mais la fréquence de leur occurence sur une plus longue période. La mémoire à long terme permet donc d'une part d'éviter de rester dans une seule région de l'espace de recherche et d'autre part d'étendre la recherche vers des zones plus intéressantes. Par exemple, la mémoire à base de fréquence attribue des pénalités à des caractéristiques des solutions plusieurs fois visitées au cours de la recherche. Par ailleurs, les mouvements ayant conduit à de bonnes solutions peuvent être encouragés. On peut ainsi garder en mémoire une liste de solutions élites que l'on utilisera comme nouveau point de départ quand la recherche deviendra improductive pendant plusieurs itérations consécutives.

L'algorithme 7 donne une version simplifiée de la méthode.

Algorithme 7 Recherche Tabou

```
trouver une solution initiale c
tabu ← ∅
repeat
    trouver la meilleure solution non tabou c' ∈ V(c)
    sauvegarder la meilleure solution rencontrée
    ajouter c' → c dans tabu
    c ← c'
until critère d'arrêt satisfait
```

Seules certaines bases de la méthode ont été présentées ci-dessus. D'autres principes permettent d'aboutir à des méthodes plus efficaces et intelligentes. Il est très improbable de pouvoir élaborer une excellente méthode du premier coup, des ajustements, dépendant à la fois du type et de l'exemple de problème traité, devront certainement avoir lieu. Notons que pour certains problèmes d'optimisation, la méthode tabou a donné d'excellents résultats. De plus, sous sa forme de base, la méthode comporte moins de paramètres de réglage que le recuit simulé, ce qui la rend plus simple d'emploi. Cependant, les divers mécanismes annexes ajoutent une complexité notable.

MÉTHODES À BASE DE POPULATIONS Les

méthodes de recherche à population, comme leur nom l'indique, travaillent sur une population de solutions et non pas sur une solution unique. Le principe général de ces méthodes consiste à combiner des solutions entre elles pour en former de nouvelles en essayant d'hériter des "bonnes" caractéristiques des solutions parents. Un tel processus est répété jusqu'à ce qu'un critère d'arrêt soit satisfait (nombre de générations maximum, nombre de génération sans améliorations, temps maximum, borne atteinte, ...). Parmi ces algorithmes à population, on retrouve deux grandes classes qui sont les algorithmes évolutionnaires et les colonies de fourmis. Les algorithmes génétiques ont beaucoup fait parler d'eux et depuis longtemps. Les colonies de fourmis, que nous ne détaillerons pas ici, sont des techniques plus récentes.

Les algorithmes génétiques et les algorithmes évolutionnaires

Les algorithmes évolutionnaires sont des techniques de recherche inspirées par l'évolution biologique des espèces, apparues à la fin des années 1950 [Fra57]. Parmi plusieurs approches, les algorithmes génétiques en constituent certainement l'exemple le plus connu, à la suite de la parution en 1989 du célèbre livre de D.E. Goldberg : *Genetic Algorithms in Search, Optimization and Machine Learning*. Les méthodes évolutionnaires ont d'abord suscité un intérêt limité, du fait de leur important coût d'exécution. Mais elles connaissent, depuis dix ans, un développement important, grâce à l'augmentation de la puissance des calculateurs et suite à l'apparition des architectures parallèles pouvant exploiter le parallélisme intrinsèque de ces méthodes.

Le principe d'un algorithme évolutionnaire se décrit simplement. Un ensemble de N points choisis au hasard dans un espace de recherche constitue la population initiale. Chaque individu c de la population possède une certaine performance, qui mesure son degré d'adaptation à l'objectif visé. Il s'agit alors de faire évoluer, par générations successives, la composition de la population en maintenant sa taille constante. Au cours des générations, l'objectif est d'améliorer globalement la performance des individus, on essaie d'obtenir un tel résultat en mimant les deux principaux mécanismes qui régissent l'évolution des êtres vivants selon la théorie de C. Darwin :

- *la séléction* qui favorise la reproduction et la survie des individus les plus performants,
- *la reproduction* qui permet le brassage, la recombinaison et les variations des caractères héréditaires des parents, pour former des individus nouveaux.

En pratique, une représentation doit être choisie pour les individus d'une population. Classiquement, un individu pourra être une liste d'entiers, un vecteur de nombre réels, une chaîne de nombre binaires... Cette représentation des solutions (codage) est un point critique de la réussite d'un algorithme génétique. Il faut bien sûr qu'il s'adapte le mieux possible au problème et à l'évaluation d'une solution. Le codage phénotypique ou codage direct correspond en général à une représentation de la solution très proche de la réalité. L'évaluation d'une solution représentée ainsi est en général immédiate. On peut aussi utiliser un codage indirect (génotypique) qui est souvent plus éloigné de la réalité et qui nécessite un algorithme de décodage pour constituer une solution valide. Pour ce dernier codage, il existe beaucoup de travaux et notamment des opérateurs de croisement et de mutation en quantité.

Le passage d'une génération à une autre se déroule en quatre phases : une phase de sélection, une phase de reproduction, une phase d'évaluation de performances et une phase de remplacement. La phase de sélection désigne les individus qui participent à la reproduction. Ils sont choisis, éventuellement à plusieurs reprises, d'autant plus souvent qu'ils sont performants. Les individus choisis sont ensuite disponibles pour participer à la phase de reproduction. Celle-ci consiste à appliquer des opérateurs de variation sur des copies des individus sélectionnés pour en engendrer de nouveaux. Les opérateurs les plus utilisés sont le *croisement*, qui produit un ou deux descendants à partir de deux parents, et la *mutation*, qui produit un nouvel individu à partir d'un seul. Les performances des nouveaux individus sont ensuite évaluées à partir des objectifs fixés. Enfin la phase de remplacement, consiste à choisir les membres de la nouvelle génération.

Nous présentons deux versions possibles d'algorithme génétique. L'algorithme 8 propose une version qualifiée en anglais par *population replacement*, tandis que la version 9 est une version incrémentale. La différence réside dans la gestion de la population.

Dans le premier cas, on crée une seconde population en parallèle à la première, puis on insère des individus obtenus par croisement et mutation. Dès qu'une nouvelle population est formée, elle remplace la population précédente et le processus recommence. Il faut prévoir un mécanisme permettant d'éviter de perdre le meilleur individu. Dans cette version, la sélection est faite en effectuant un tirage aléatoire biaisé par la valeur de fitness des individus. Comme dans les systèmes naturels, croisement et mutations se font sous conditions de probabilité.

Algorithme 8 Algorithme Génétique (population replacement)

1: générer une population initiale $\mathcal{P}$ de n individus
2: **repeat**
3: $\quad\mathcal{P}' \leftarrow \emptyset$
4: $\quad$**repeat**
5: $\quad\quad$*séléction* : choisir deux individus c et c' de $\mathcal{P}$ en fonction de leur fitness
6: $\quad\quad$*croisement* : combiner c et c' pour former d et d'
7: $\quad\quad$*mutation* : muter c et c' avec une faible probabilité
8: $\quad\quad$ajouter d et d' dans $\mathcal{P}'$
9: $\quad$**until** $|\mathcal{P}'| = n$
10: $\quad\mathcal{P} \leftarrow \mathcal{P}'$
11: **until** critère d'arrêt satisfait

Dans la seconde version, on crée un nouvel individu par croisement et mutation et il remplace un individu existant de la population courante. Pour cette méthode, on peut utiliser la méthode de tournoi binaire (*roulette wheel*) pour la sélection.

Algorithme 9 Algorithme Génétique (incremental replacement)

1: générer une population initiale $\mathcal{P}$ de n individus
2: **repeat**
3: *séléction* : choisir deux individus c et c' de $\mathcal{P}$ par la méthode de tournoi
4: *croisement* : combiner c et c' pour former d
5: *mutation* : muter d avec une faible probabilité
6: choisir un individu d' de la population
7: remplacer d' par d
8: **until** critère d'arrêt satisfait

L'intensification est amenée à la fois par la séléction et le croisement. Le facteur de diversification principal est la mutation.

Parce qu'ils manipulent une population d'instances de solutions, les algorithmes évolutionnaires sont particulièrement indiqués pour proposer un jeu de solutions diverses, quand une fonction objectif comporte plusieurs optima globaux. Ainsi, ils peuvent fournir un échantillon de solutions de compromis, lors de la résolution de problèmes comportant plusieurs objectifs.

Les algorithmes évolutionnaires ont également trouver à s'appliquer dans de nombreux domaines. Il faut toutefois reconnaître qu'ils sont relativement gourmands en temps de calcul. Ils nécessitent d'autre part une bonne connaissance des techniques afin de bien adapter le codage et les opérateurs au problème considéré. Bien maitrisés, ils s'avèrent toutefois très intéressants.

RÉGLAGES ET COMPARAISON DES MÉTHODES La mise au point d'une méthode heuristique pour résoudre un problème d'optimisation difficile demande de procéder à divers choix. Certains de ceux-ci peuvent être relativement faciles à justifier, mais d'autres comme le calibrage des paramètres numériques ou le choix d'un voisinage plutôt qu'un autre peuvent être beaucoup plus délicats. Lorsque la théorie ou l'intuition ne sont pas à même de nous guider vers un choix, il ne reste plus qu'à effectuer des expériences empiriques pour obtenir des réponses. Or il est difficile d'étayer ces choix empiriques par des bases scientifiques propres. Rappelons que les méthodes que nous avons présentées sont stochastiques, de ce fait afin de conduire un calibrage rigoureux, sans pour autant faire des batteries gigantesques de tests numériques, il est important de faire usage de tests statistiques.

Une première question concerne la comparaison des taux de succès de deux méthodes $\mathcal{A}$ et $\mathcal{B}$. En pratique, nous avons les informations suivantes : lorsque la méthode $\mathcal{A}$ est exécutée n_a fois, elle réussit à résoudre le problème a fois ; lorsque la méthode $\mathcal{B}$ est exécutée n_b fois, elle réussit à résoudre le problème b fois. On se pose alors la question suivante : est-ce qu'un taux de succès $\frac{a}{n_a}$ est significativement meilleur que le taux $\frac{b}{n_b}$? Evidemment un expérimentateur consciencieux effectuera un très grand nombre de tests et effectuera un test statistique standard se reposant sur le théorème central limite. On peut lire dans la littérature [Tai03] qu'un taux de succès de 5/5 est significativement, avec un seuil de confiance de 95%, plus élevé qu'un taux de 1/4. Il existe des tableaux permettant de déterminer les couples de valeur donnant des seuils de confiance équivalent.

Avant d'utiliser ces tableaux, il faut bien évidemment définir ce que l'on entend par succès. Cela peut être le fait de trouver la solution optimale ou une solution approchée de qualité donnée pour un problème. Il faut également comparer les méthodes avec le même problème ou tout du moins des problèmes de difficulté identique.

En plus de l'optimisation de la qualité des solutions, on cherche à minimiser l'effort de calcul nécessaire pour y parvenir. Or ce dernier paramètre peut être à la discrétion de l'utilisateur à travers un nombre maximal d'itérations ou d'un nombre limite de générations d'un algorithme évolutionnaire. De plus ces méthodes étant non déterministes, deux exécutions successives sur un même exemple donnent, en général, des solutions différentes.

Traditionnellement, la mesure de la qualité d'une méthode est la valeur moyenne des solutions qu'elle produit. La mesure de l'effort de calcul est exprimé en secondes sur un calculateur donné. Ces deux mesures présentent des inconvénients. Si l'on fixe l'effort de calcul pour les méthodes $\mathcal{A}$ et $\mathcal{B}$, et que l'on désire comparer leur qualité, il faudra exécuter ces méthodes plusieurs fois et procéder à un test statistique portant sur les moyennes des solutions obtenues. Malheureusement, la répartition des solutions produites n'est en général pas gaussienne, et il est nécessaire de répéter les expériences numériques un grand nombre de fois, en pratique cela peut être plusieurs centaines de fois.

Si l'on se tourne vers d'autres mesures de qualité que la moyenne, on peut obtenir des comparaisons intéressantes avec très peu d'exécutions. Une de ces méthodes consiste à classer par qualité décroissante l'ensemble des solutions produites par les méthodes $\mathcal{A}$ et $\mathcal{B}$ et à se préoccuper de la somme des rangs des solutions obtenues par une méthode. Si cette somme est inférieure à une valeur, dépendante du seuil de confiance désirée et que l'on peut trouver dans des tables numériques, on peut en conclure qu'une exécution de cette méthode a une probabilité significativement supérieure à 1/2 d'obtenir une solution meilleure que l'autre.

Bien évidemment, si l'on compare des méthodes itératives, ce test doit être répété pour chaque effort de calcul fixé. Il est préférable d'éviter de mesurer l'effort par le temps d'exécution sur une machine donnée, par trop dépendant du matériel utilisé, du système d'exploitation, du compilateur et autres. Il est préférable d'utiliser une mesure absolue de l'effort, telle que le nombre d'évaluation de solutions voisines. On exprime alors l'effort non pas en secondes mais en nombre d'itérations dont on peut spécifier la complexité unitaire.

En appliquant ces principes, il est possible de générer deux diagrammes sur le même schéma, en fonction de l'effort de calcul : le premier donnant la valeur moyenne des solutions obtenues et le second donnant la probabilité qu'une méthode soit meilleure que l'autre.

Nous comprenons donc qu'il est loin d'être trivial de régler les paramètres des métaheuristiques ainsi que de les comparer.

BILAN Nous avons fait un tour d'horizon des principales métaheuristiques d'optimisation en tentant de faire ressortir leurs avantages, leurs inconvénients ainsi que leurs points communs. Bien évidemment, il existe un grand nombre d'autres méthodes que nous n'avons pu développer telles que les *colonies de fourmis*, les *essaims particulaires*, *GRASP*, les *recherches à voisinage variable* et bien d'autres. Nous avons également du passer sous silence un grand

nombre de raffinements et variantes propres à chaque méthode. C'est pourquoi nous invitons le lecteur désireux de creuser ces questions à consulter les références que nous avons mentionnées.

Que ressort-il de cette étude ? Les métaheuristiques sont des méthodes d'optimisation efficaces qui ont fait leurs preuves en pratique. Elles sont toutefois fortement tributaires de leurs paramètres qui sont difficiles à régler. Il est, d'autre part, difficile de choisir une technique plutôt qu'une autre ainsi que de les comparer. Leur temps de calcul peut s'avérer également assez conséquent pour des problèmes de taille importante. Elles se prêtent cependant bien à la parallélisation. Notons que les résultats théoriques du domaine restent encore assez limités et que des études sont en cours pour tenter de résoudre ces limitations. On tend également de plus en plus vers l'utilisation de méthodes hybrides combinant une ou plusieurs métaheuristiques et des techniques locales chargées d'affiner les solutions.

MÉTAHEURISTIQUES & CLUSTERING

De nombreuses méthodes de clustering et d'apprentissage automatique peuvent se ramener à l'optimisation d'une fonction objectif. D'autre part, il est clair que sur de grands ensembles de données, décrites dans un espace de dimension élevée, il est impossible d'explorer exhaustivement l'espace de recherche pour trouver la solution optimale au sens de la fonction à optimiser. Comme nous l'avons vu, les algorithmes classiques tels que *EM*, les *k-means* et autres peuvent converger vers un optimum local éloigné de l'optimum global. Compte tenu de la description que nous avons faite des métaheuristiques d'optimisation, il semble tout à fait naturel d'utiliser ces techniques pour pallier les limitations des algorithmes mentionnés. Nous allons donc étudier plus précisément comment appliquer les métaheuristiques d'optimisation à l'apprentissage non-supervisé. Pour ce faire, nous présenterons un bref état de l'art du domaine émaillé par une analyse plus détaillée de quelques articles traitant de la question.

ÉTAT DE L'ART

Recherche locale

Nous avons présenté quelques métaheuristiques de recherche locale. Il convient maintenant d'étudier comment il a été proposé de les exploiter dans le cadre de l'apprentissage non-supervisé.

Un certain nombre de tentatives ont été mises en oeuvre pour utiliser le recuit simulé dans des tâches de clustering. Citons [Dub89], ainsi que l'algorithme SINICC (Simulation of Near-optima for Internal Clustering Criteria) [BH90]. Ce dernier article est relativement simple. Il propose tout d'abord une formalisation générale du problème de clustering. La fonction objectif est extrêmement simple puisqu'il est proposé d'utiliser la somme des distances au sein de chaque cluster. Cela conduit à générer un cluster par instance donnée. Ce problème est partiellement résolu en utilisant la somme des distances à laquelle est retranchée un seuil pour chaque point d'un cluster. Ainsi les clusters de grandes tailles sont pénalisés par le nombre de distance ajouté alors que les clusters trop réduits souffrent d'avoir une moins grande valeur retranchée. Il présente succintement les fondements et principes du recuit simulé. Il discute brièvement du voisinage utilisé et du réglage des paramètres du recuit simulé. L'opérateur de perturbation consiste ici à réallouer un point du cluster courant à un autre choisi aléatoirement. Il présente également une application de l'algorithme pour un problème de surveillance d'entités terrestres par des senseurs. Enfin, il conclut que le recuit simulé, bien que coûteux, s'avère adapté en pratique.

En ce qui concerne l'utilisation de la recherche tabou, la littérature ne manque pas d'exemples. Nous allons nous focalisé ici sur l'étude de quelques articles représentatifs ainsi que sur l'algorithme PRESS.

[NW02] cherche à traiter le problème du clustering flou en utilisant la recherche tabou. Le problème est dans un premier temps formulé sous forme de problème d'optimisation. La fonction à optimiser n'étant généralement pas convexe il est rappelé que l'algorithme fuzzy k-mean converge généralement vers un optimum local. Un rappel de l'algorithme fuzzy k-modes ainsi que de fuzzy k-prototypes est donné, il n'appelle pas de commentaires particuliers. Après un rappel des grands principes de la méthode tabou, l'article aborde précisément leur extension de fuzzy k-modes utilisant la recherche tabou. L'ensemble des voisins d'un état courant s'obtient en modifiant aléatoirement les attributs d'un des centres choisis aléatoirement. Les attributs numériques et ordinaux sont modifiés en leur ajoutant 0, 1 ou -1 alors que l'on s'autorise à modifier d attributs catégoriels non ordinaux aléatoirement. Les jeux de données ainsi que les modalités de test sont ensuite décrits. La mesure de qualité repose simplement sur le rapport entre le nombre d'instances bien classées et le nombre total d'instances. La méthode avec recherche tabou atteint 99% de précision moyenne sur 100 exécutions contre 79% pour le fuzzy clustering classique. La méthode avec recherche tabou trouve la bonne classification 67 fois sur 100 contre 20 pour l'algorithme classique. Différents tests empiriques sont menés pour le réglage des paramètres. Ces tests ont été effectués avec des classes connues a priori afin de vérifier la pertinence des résultats, toutefois il apparait que comme plus la valeur de la fonction objectif est faible plus la précision du clustering est bonne il est possible de se fier à la simple valeur de la fonction objectif. Une comparaison similaire est faite entre recherche tabou et fuzzy k-prototypes. Là encore, la recherche tabou fait mieux que la méthode classique. Il faut toutefois noter que la méthode tabou est beaucoup plus gourmande en temps d'exécution que les méthodes classiques.

[SJ00] s'intéresse au clustering classique et non plus au fuzzy clustering comme l'article précédent. Le problème s'exprime simplement comme un problème d'optimisation sous contraintes en variables 0-1. La fonction objectif étant non convexe et non linéaire, il apparaît là encore difficile d'aborder le problème par une approche analytique. L'algorithme repose sur trois procédures, la procédure de *packing*, la procédure de *releasing* et la recherche tabou. Le packing permet de traiter un sous-ensemble d'éléments très proches comme un seul élément. Ainsi, les éléments très proches sont regroupés en un seul élément et assignés au même cluster. Pour ce faire, on considère toutes les paires d'éléments que l'on classe par ordre croissant de distance. On réunit les premiers éléments jusqu'à ce que l'on obtienne le nombre désiré de paquets en regroupant certains paquets lorsque cela s'avère nécessaire. Les auteurs dis-

cutent du nombre de paquets à créer. La procédure de *releasing* est l'opération inverse permettant de libérer les paires les moins proches. L'opérateur de voisinage déplace aléatoirement tous les éléments d'un même paquet. L'algorithme utilise deux listes tabou, une liste classique pour éviter les cycles directs et une seconde liste tabou pour éviter les cycles liés aux différentes permutations des clusters qui représentent en fait une partition identique des données. La pertinence des jeux de données de test est ensuite discutée, ils ont essayé de tester le recuit simulé, la recherche tabou, k-means et leur algorithme sur des données dont les nombre d'instances, d'attributs et de cluster soient suffisamment variés pour dégager les propriétés de chaque méthode. La recherche tabou semble meilleure que le recuit simulé et l'algorithme k-means mais moins bonne que l'algorithme proposé.

Intéressons nous enfin à l'algorithme PRESS proposé dans [Vel05], [VG05]. Cette algorithme vise à extraire des stéréotypes en se fondant sur la logique des défauts et sur des techniques de recherche locale. Il vise à traiter des données fortement lacunaires. Il s'inscrit dans la famille des algorithmes de clustering conceptuel, i.e. la description des clusters est intelligible et permet de raisonner sur les catégories formées. Pour ce faire, il utilise une extension de la subsomption classique : la subsomption par défaut. Une mesure de similarité basée sur la subsomption par défaut ainsi que sur le nombre d'attributs communs entre une description d'un stéréotype et une donnée est utilisée. Chaque instance se voit associée au stéréotype dont elle est le plus proche selon cette mesure de similarité. L'algorithme construit une partition des données en utilisant une recherche tabou pour optimiser la fonction objectif définie comme la somme des distances entre chaque exemple et le stéréotype associé selon la mesure de similarité précédemment décrite. La recherche tabou a été utilisé pour parcourir l'espace de recherche. Les opérateurs de voisinage utilisés sont ajout ou retrait d'un descripteur d'un stéréotype (mouvement de faible influence) ou ajout ou suppression d'un stéréotype (forte influence). L'un des avantages de cette technique est qu'il n'est pas nécessaire de fixer à l'avance le nombre de classes que l'on veut obtenir. L'algorithme a été testé sur des jeux de données artificiels sur lesquels il fait mieux que EM, COBWEB et K-MODES sur des données lacunaires. Notons également que les stéréotypes construits semblent porteur de sens.

Approches évolutionnaires

Nous avons étudié les approches évolutionnaires, voyons comment il a été proposé de les utiliser dans le cadre du clustering. L'algorithme 10 présente la méthode générale dans ce cas.

Algorithme 10 Approche évolutionnaire

1: Choisir une population aléatoire correspondant à une partition valide des données. On associe à chaque solution une valeur de *fitness*, par exemple l'inverse de l'erreur quadratique.
2: **repeat**
3: Utiliser les opérateurs (sélection, croisement, mutation) pour générer la population suivante. Evaluer la *fitness* des nouveaux individus.
4: **until** critère d'arrêt satisfait

L'un des premiers articles sur le sujet est sans doute celui de Raghavan et Birchand [RB79], où les algorithmes génétiques ont été utilisés pour minimiser l'erreur quadratique du clustering. Ici, chaque chromosome représente une partition de N objets dans K clusters et est représenté par une chaîne de longueur N sur K symboles. Par exemple, pour les instances A, B, C, D, E, F la chaîne 101001 représente l'assignation suivante sur deux clusters : {A, C, F} et {B, D, E}. Avec cette représentation, lorsqu'il y a K clusters il y a $K!$ chromosomes différents correspondant à chaque K-partition des données. Des travaux ont été effectués pour améliorer la représentation des chromosomes.

Dans Bhuyan et al. [BRE91], la représentation suivante a été proposée : un symbole de séparation, noté $*$ est utilisé. Ainsi le chromosome ACF$*$BDE correspond à la 2-partition {A, C, F} et {B, D, E}. Cette représentation permet d'assimiler le problème de clustering à un problème de permutation du type voyageur de commerce ce qui permet d'utiliser les opérateurs classiques qui ont faire leur preuve sur ce problème. Il est clair que cette représentation souffre toujours d'une redondance de représentation dans l'expression d'une même partition.

Une approche hybride a été proposé dans Babu et Murty [BM93], un algorithme génétique est ici utilisé pour trouver de bons centroïdes initiaux et l'algorithme k-means est ensuite appliqué pour trouver la partition finale.

[HÖB99] propose d'utiliser une approche guidée génétiquement (*Genetically Guided Algorithm* - GGA) afin d'optimiser l'algorithme k-means dur et flou. Le but est d'éviter les optima locaux et de limiter la sensibilité à l'initialisation de ces méthodes. La sélection se fait en sélectionnant aléatoirement deux individus et en conservant celui qui présente la meilleure fitness. Des croisements sont effectués en deux points et ce pour toutes les paires de centroïdes de chaque parent. La mutation modifie chaque bit suivant une probabilité. Une partie est consacrée au réglage des paramètres à savoir : le taux de mutation, la probabilité de croisement, le critère d'arrêt ou encore la taille de la population. Le reste du papier est consacré à des tests empiriques sur différents jeux de données. Les conclusions sont que les algorithmes génétiques permettent d'obtenir des solutions au moins aussi bonnes que les méthodes classiques.

Notons enfin l'utilisation d'algorithmes génétiques avec des génomes de longueur variable [LA00] pour à la fois améliorer les centroïdes des k-means mais également la valeur de k qui s'avère plus efficace que de lancer plusieurs fois la méthode avec des k différents.

Les principales défauts de ces méthodes sont leur sensibilité aux réglages de leurs paramètres, le choix d'une méthode de représentation appropriée qui soit rapide et sans redondance et leurs temps de calcul.

Comparaison des techniques

Une étude empirique des performances du recuit simulé, de la recherche tabou et des algorithmes génétiques appliqués au clustering a été proposé dans [MR94]. Il en ressort que les algorithmes génétiques sont performants pour les données unidimensionnelles tandis qu'ils s'avèrent moyens pour les données en plus grandes dimensions. Le recuit simulé se révèle lent et il semble que la recherche tabou s'en sort le mieux. Notons cependant qu'aucune méthode semble vraiment se dégager.

Une autre étude empirique des k-means, du recuit simulé, de la recherche tabou et des algorithmes génétiques a été proposé dans [ASK96]. La recherche tabou, le recuit simulé et les algorithmes génétiques ont été jugés similaires en terme de qualité de la solution renvoyée qui est meilleure que celle de k-means. Cependant, la méthode des k-means est beaucoup plus rapide et moins coûteuse que les méthodes citées par un facteur de 500 à 2500 et ce pour partitionner un jeux de données de 60 instances en 5 clusters. Notons que les tests ont été effectués sur des données de petites tailles de l'ordre de moins de 200 instances.

En résumé, les algorithmes employant le recuit simulé, la recherche tabou ou les algorithmes génétiques ne semblent pas avoir été testés sur des données de grande taille. Cela s'explique par la difficulté de réglage des paramètres de ces méthodes ainsi que par leur temps d'exécution important.

NOS EXPÉRIMENTATIONS

Nous avons souhaité mettre en pratique les connaissances acquises dans le cadre d'implémentations concrètes. Il s'agissait ainsi de mieux appréhender les problématiques susceptible de se poser ainsi que de pouvoir mener à bien quelques comparaisons empiriques des différentes méthodes. Nous relatons donc ici le travail pratique qui a été accompli sur l'implémentation des métaheuristiques ainsi que nos expérimentations relatives à un algorithme d'apprentissage non-supervisé les exploitant.

IMPLÉMENTATION DES MÉTAHEURISTIQUES
Pour nos expérimentations, nous avons implémenté les métaheuristiques suivantes : méthodes de descente, recuit simulé, grand déluge, record en record, recherche tabou et algorithmes génétiques.

Commentaires

Bien que l'implémentation en elle-même ne requiert pas de commentaires particuliers car relativement simple, nous avons souhaité mettre l'accent sur la généricité et la réutilisation. Le diagramme de la figure 2 présente l'architecture générale adoptée. Nous avons défini une classe abstraite *MetaHeuristic* qui contient tout ce qui peut être généralisé à savoir un nombre d'itérations maximal, la fixation d'un coût acceptable pour une solution et autres. Elle demande à ses filles d'implémenter une méthode *optimize*. De cette classe abstraite dérive une classe pour chaque métaheuristique qui contient les paramètres et implémentations propres à chaque méthode. La classe *MetaHeuristic* contient un attribut qui n'est autre que la définition du problème à optimiser. Chaque problème hérite de la classe abstraite *OptimizationProblem* qui demande d'implémenter les méthodes suivantes : *getRandomState* pour obtenir un état aléatoire, *getNextStates* pour obtenir un voisinage à partir d'un état courant, *objective* qui n'est autre que la fonction objectif à optimiser. Si une solution intéressante est déjà connue, il est possible de la spécifier par *setInitialSolution*, l'optimisation se fera alors à partir de là. La classe *OptimizationProblem* manipule des états qui sont évidemment dépendant du problème traité, cela se fait par l'intermédiaire d'une classe abstraite *State* qui définit la représentation d'un état et nécessite simplement de définir une méthode permettant de déterminer si deux états sont égaux. Notons que pour utiliser les algorithmes génétiques il est nécessaire de faire dériver le problème de la classe *GeneticOptimizationProblem* qui demande en plus d'implémenter les méthodes de croisement et de mutation qui là encore sont propres à chaque problème.

Cette approche permet donc très facilement de décrire un problème et de l'optimiser avec la méthode souhaitée. Il est également très simple d'étendre les fonctionnalités et d'ajouter de nouvelles métaheuristiques.

L'ensemble a été implémenté en Java afin de pouvoir éventuellement s'intégrer dans Press mais également afin de pouvoir être utilisé avec Weka, nous y reviendrons. Nous avons également pris soin de documenté le code au format javadoc afin de pouvoir éventuellement proposer notre travail aux développeurs de Weka.

Test des algorithmes

Afin de valider nos méthodes et l'architecture adoptée, nous les avons testées sur deux problèmes classiques pour lesquels les solutions étaient connues à savoir un problème de placement de composants électroniques et le problème de recherche du cycle hamiltonien de coût minimal i.e. le problème du voyageur de commerce. Dans le problème du placement de composants électroniques, nous avons N composants électroniques à placer afin de réduire la longueur en L entre des composants définis comme voisins dès le départ. Il y a $N!$ solutions possibles. Un voisin s'obtient en permutant deux composants, ce qui nous donne $\frac{N(N-1)}{2}$ voisins à chaque état. On obtient les figures 3 et 4 pour 25 composants.

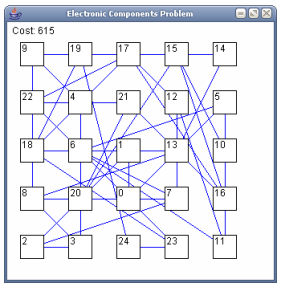

FIG. 3 – Solution initiale

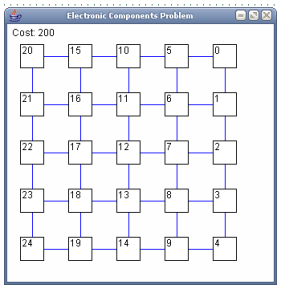

FIG. 4 – Optimum globale

Nous avons obtenu les résultats de la figure 6 à partir de la configuration initiale de la figure 5 pour le problème du voyageur de commerce avec 89 villes. Nous avons testé avec succès des problèmes de 1000 villes.

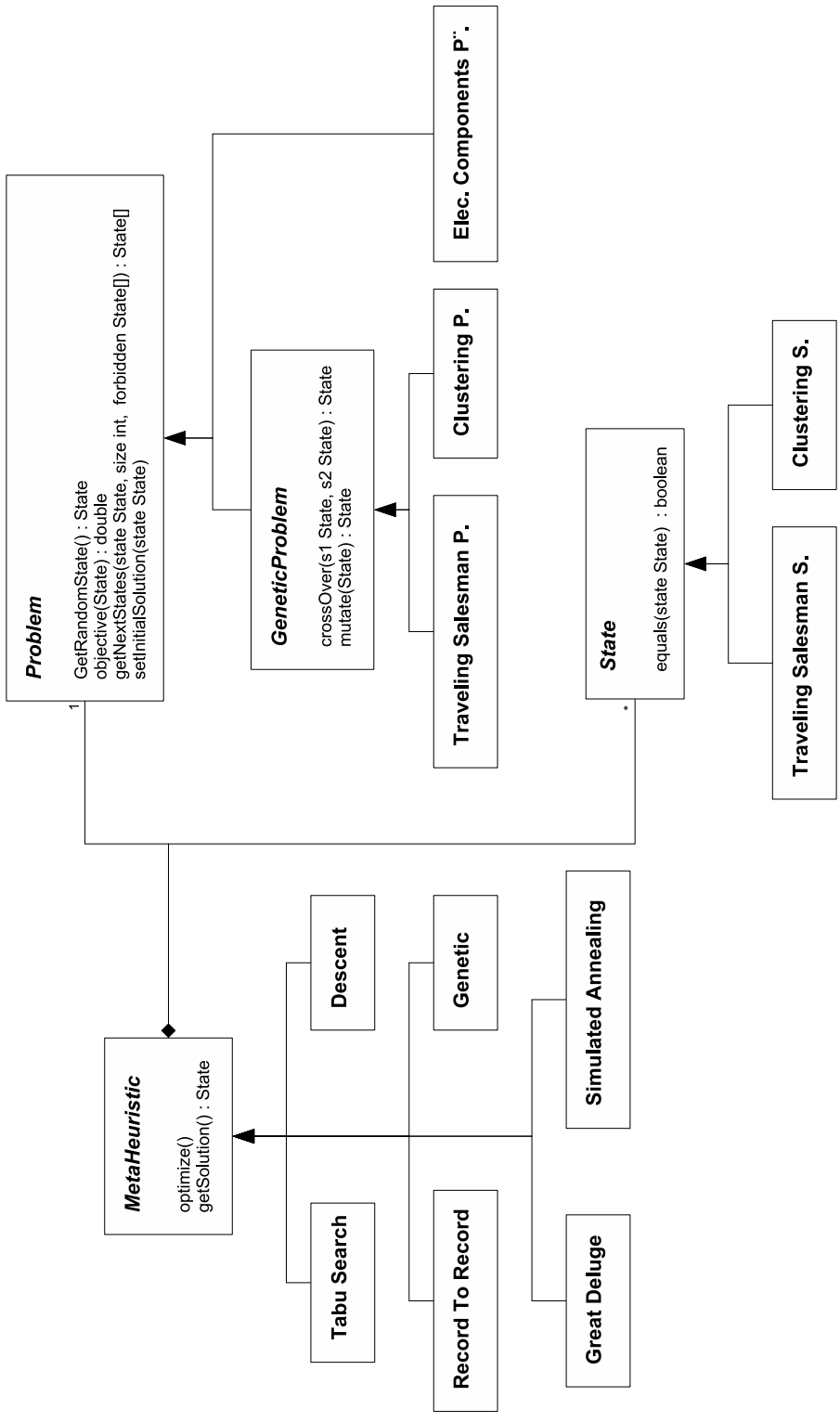

FIG. 2 – Diagramme UML de l'implémentation

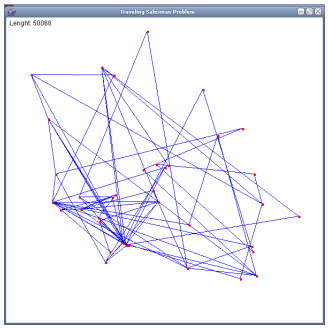

FIG. 5 – Solution initiale

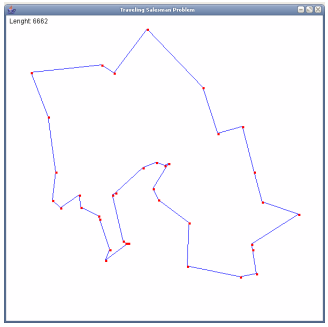

FIG. 6 – Optimum globale

INTÉGRATION DANS WEKA Weka [WF05], est un ensemble d'algorithmes d'apprentissage regroupés dans une application commune visant à traiter différentes tâches sur des données. Cet outil a été développé par l'université de Waikato et repose sur le langage Java. L'application fournit notamment des outils de prétraitement des données, de classification, de régression, de clustering, d'extraction de règles d'associations et de visualisation. L'ensemble est libre et sous licence GPL.

Nous avons choisi d'intégrer nos travaux dans Weka pour différentes raisons. Tout d'abord, il nous est apparu pertinent d'exploiter l'interface graphique offerte par Weka, notamment pour l'importation et l'exportation des fichiers ainsi que pour le prétraitement et la visualisation des données. D'autre part, le format arff, que nous avons utilisé, est le fruit des développeurs de Weka, c'est pourquoi, plutôt que de refaire un parser pour ce format et afin de pouvoir bénéficier directement de ses éventuelles évolutions il nous est apparu comme une bonne chose de s'insérer dans Weka. Enfin, il peut être intéressant de partager nos développements avec la communauté des utilisateurs de Weka.

Dans cet esprit, nous avons donc ajouté un module *optimizers* au code source de Weka qui permet d'utiliser très simplement n'importe quelle métaheuristique d'optimisation implémentée. Il est d'autre part très simple d'ajouter à loisirs de nouvelles méthodes. Nous avons également ajouté l'algorithme *PRESS* dans le module *clusterers* comme en témoigne la figure 7.

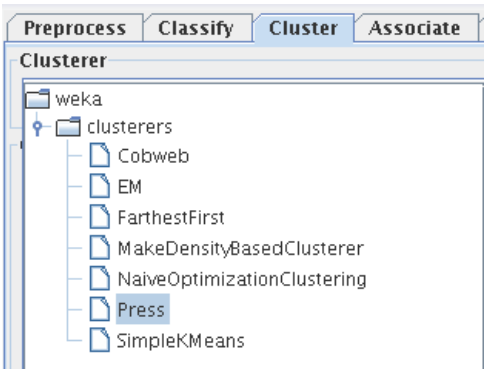

FIG. 7 – Choix d'un algorithme de clustering dans Weka

UN ALGORITHME DE CLUSTERING Nous avons décidé d'implémenter un algorithme de clustering très simple, nommé *NaiveOptimizationClustering* afin de comparer les différentes métaheuristiques dans le contexte de l'apprentissage non-supervisé. Ceci impliquait évidemment une modélisation particulière du problème de clustering sous forme d'états et d'une fonction de transition permettant de passer d'un état au suivant. Pour la modélisation du voisinage, deux choix se sont offerts à nous, soit déplacer aléatoirement un exemple assigné d'un cluster à un autre, soit déplacer aléatoirement un des centroïdes dans l'espace des attributs. Nous avons opté pour la deuxième solution qui permet de faire des déplacements plus radicaux dans l'espace de recherche. Les exemples sont ensuite assignés aux centroïdes dont ils sont les plus proches à la manière des k-means.

Les méthodes de métaheuristique servant à la minimisation d'une fonction objectif, il nous à fallu définir une fonction bien adaptée au problème. Afin de remplir sa mission, cette fonction doit renvoyer un indice représentatif à la fois de la qualité des classes effectivement construites, mais également de la pertinence des représentants des classes. Comme nous l'avons vu, de nombreuses méthodes répondant à cette problématique existent. Dans notre travail, nous avons utilisé la mesure de compacité-séparation sous la forme : $\beta * compacite + (1-\beta) * separation$ qui traduit l'objectif principal d un clustering réussi, le coefficient β traduisant un compromis entre les deux critères principaux, à savoir :

- l'homogénéité interne des exemples à l intérieur d une même classe (homogénéité intraclasse)
- la séparation entre les exemples de classes différentes (hétérogénéité interclasse)

Cette mesure se base essentiellement sur la notion de distance entre deux exemples (un centroïde étant considéré comme un exemple particulier). Là aussi de nombreuses méthodes existent (Russel et Rao, Jaccard, Kendall) et l'utilisation de l'une d'elle plutôt qu'une autre repose fortement sur la nature du problème et des exemples considérés. C'est pourquoi nous avons choisi de toutes les implémenter et de laisser la possibilité à l'utilisateur de choisir celle qu'il souhaite utiliser au moment du lancement de l'algorithme. Les métriques utilisées permettent de traiter les données catégorielles, numériques ainsi que les données cachées.

Notons que le choix de la métaheuristique à employer, des paramètres généraux, ainsi que des paramètres propres

à chaque technique sont également modifiables directement depuis l'interface de Weka comme le montre la figure 8. Le nombre de clusters à générer doit être défini a priori.

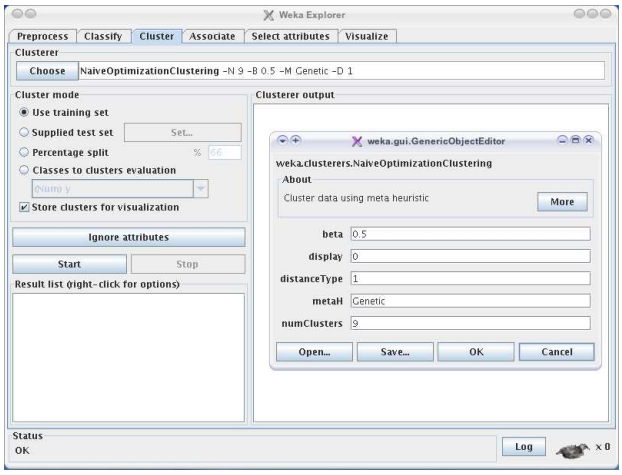

FIG. 8 – Interface de l'algorithme intégrée à Weka

En ce qui concerne les méthodes particulières liées aux algorithmes génétiques, la mutation d'un état en un autre suit le même processus que la fonction de transition décrite ci-dessus. En revanche, l'opérateur de croisement est un tout petit moins naïf : un appariement des centroïdes entre 2 états est effectué de façon à regrouper les centroïdes qui sont les plus ressemblants entre les 2 états, ensuite, le croisement consiste simplement à prendre X centroïdes de l état 1 et Y de l état 2 pour réaliser un nouvel état à X+Y centroïdes.

La figure 9 présente les résultats obtenus sur un exemple très simple de test avec 1230 instances décrites sur seulement 2 attributs. Nous reviendrons sur ces données lors de la présentation de nos résultats.

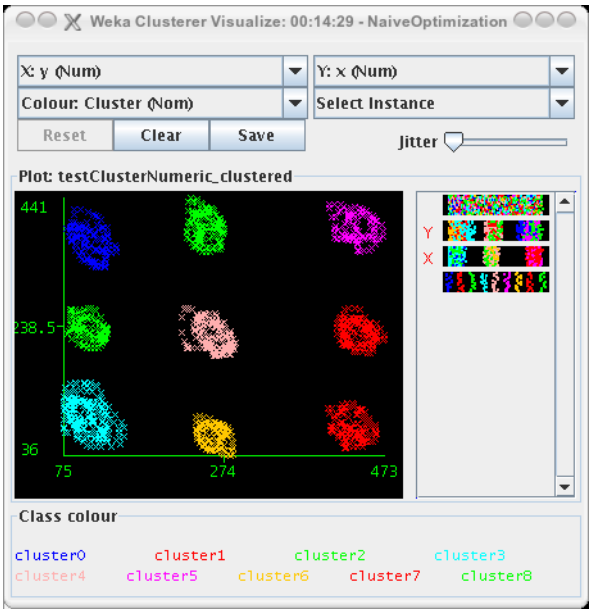

FIG. 9 – Exemple de clustering

RÉSULTATS

Afin de comparer les métaheuristiques entre elles, nous avons effectué un premier test sur 1230 instances décrites sur 2 attributs numériques. Ces données présentent 9 nuages de points identifiables immédiatement à l'oeil humain. Pour recueillir les résultats décrits dans le tableau 1, nous avons lancé chaque algorithme 100 fois sur les données en ayant pris soin au préalable de régler les paramètres de chaque méthode. Les chiffres présentés constituent les moyennes obtenues.

	Etats	Temps (s)	Coût
D. Simple	23251	50,046	0.073
D. Grande	55025	188,908	0.071
Recuit Simulé	16521	45,713	0.075
Grand Deluge	5001	10,969	0.095
Record	17901	42,488	0.073
Recherche Tabou	12432	28,203	0.077
AG Population	15680	41,400	0.094
AG Incremental	17907	45,504	0.083

TAB. 1 – Résultats du test 1

On en déduit que toutes les méthodes parviennent à obtenir un coût intéressant et à classer correctement les données. Evidemment globalement plus le nombre d'états visités est important plus le temps d'exécution est important et plus le résultat est bon. Notons toutefois que les algorithmes génétiques semblent en retrait sur ce test.

Nous avons effectué un second test sur le fichier *zoo.arff* qui décrit une classification du règne animal. Il est constitué de 101 instances décrites sur 18 attributs dont 17 sont symboliques, 7 classes sont fournies a priori. Cela nous permet donc de tester comment notre algorithme se comporte par rapport aux algorithmes classiques du domaine et si il parvient à retrouver ces classes. Les résultats sont présentés dans le tableau 2.

	Erreurs	% d'erreurs	Coût
EM	33	32.67	-
FarthestFirst	28	27.72	-
Kmeans	28	27.72	-
D. Simple	25	24.75	0.60
D. Grande	25	24.75	0.60
Recuit Simulé	24	23.76	0.62
Grand Deluge	18	17.82	0.63
Record	22	21.78	0.62
Recherche Tabou	22	21.78	0.61
AG Population	33	32.67	0.53
AG Incremental	28	27.72	0.48

TAB. 2 – Résultats du test 2

Il ressort tout d'abord que toutes les méthodes à base de métaheuristiques parviennent à faire mieux que les algorithmes classiques. Notons toutefois que la fonction objectif ne semble pas complètement révélatrice d'un clustering réussi puisque la méthode du grand déluge obtient la moins bonne minimisation du coût et pourtant c'est elle qui obtient la classification la plus fidèle. Il ressort d'autre part qu'en

terme d'optimisation pure de la fonction objectif les algorithmes génétiques se montrent extrêmement performant. D'après nos tests sur d'autres données, leur utilisation est très intéressante car ils s'avèrent assez rapides, stables et obtiennent les coûts les plus faibles.

PERSPECTIVES & CONCLUSIONS

Les algorithmes classiques de clustering présentent un certain nombre de défauts, ils peuvent rester emprisonner dans des optima locaux (EM, k-means), il est nécessaire de leur fournir le nombre de classes à générer a priori (k-means et ses dérivés), ils sont parfois sensibles à l'ordre des données d'entrée (COBWEB).

La problématique du clustering se ramenant fréquemment à une optimisation de fonction objectif, il est naturel de vouloir tenter d'exploiter les avantages des métaheuristiques d'optimisation qui sont génériques, fournissent une solution approchée de l'optimum global en parcourant intelligemment des espaces de recherche combinatoires ne pouvant être explorés exhaustivement.

Nous avons vu par l'étude d'un certain nombre d'articles du domaine ainsi que par nos expériences que ces méthodes parviennent effectivement à trouver de très bonnes solutions. Elles ne sont toutefois pas exemptes de défauts puisqu'elles partagent évidemment les mêmes problèmes que les métaheuristiques sur lesquelles elles s'appuient : elles sont coûteuses en temps de calcul et sensibles au réglage de leurs nombreux paramètres ainsi qu'à la définition des opérateurs de voisinage. Il est difficile de les utiliser car ces paramètres doivent être choisis avec soin et sont généralement réglés de manière empirique. Il est également malaisé de choisir la métaheuristique qui s'adaptera le mieux au problème.

Bien que nous n'ayons pas pu faire de comparaisons très approfondies entre ces méthodes, il semble qu'elles se valent plus ou moins sur les données relativement simples. En revanche, nous avons été agréablement surpris par les performances des algorithmes génétiques en mode incrémental qui s'en sortent très honorablement sur des entrées plus volumineuses et font montre d'une grande stabilité dans leurs résultats. Il conviendrait toutefois de faire de plus amples analyses sur des problèmes plus variés pour se faire une idée plus précise.

Il serait intéressant d'améliorer notre algorithme naïf afin qu'il découvre automatiquement le nombre le plus adéquat de clusters à générer. Il serait également souhaitable de tester d'autres métaheuristiques que la recherche tabou avec l'algorithme PRESS. Une autre piste de recherche serait d'investiguer des méthodes hybrides combinant une phase d'initialisation exploitant les métaheuristiques puis faisant intervenir des algorithmes classiques de clustering tels que les k-means.

REMERCIEMENTS

Nous remercions chaleureusement Jean-Gabriel Ganascia ainsi que Julien Velcin pour leur disponibilité, leur gentillesse et sans qui ce travail n'aurait pu être effectué.

RÉFÉRENCES

*

[ASK96] Khaled S. Al-Sultan and M. Maroof Khan. Computational experience on four algorithms for the hard clustering problem. *Pattern Recogn. Lett.*, 17(3) :295–308, 1996.

[AvL85] E. H. L. Aarts and P. J. M. van Laarhoven. Statistical cooling : A general approach to combinatorial optimization problems. Technical report, Philips J. Research, 1985.

[Ber02] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[BH90] D. E. Brown and C. L. Huntley. A practical application of simulated annealing to clustering. Technical Report IPC-91-03, 1990.

[Bil98] Jeff A. Bilmes. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Technical report, ICSI, 1998.

[BM93] G. Phanendra Babu and M. Narasimha Murty. A near-optimal initial seed value selection in k-means algorithm using a genetic algorithm. *Pattern Recogn. Lett.*, 14(10) :763–769, 1993.

[BRE91] J.N. Bhuyan, V.V. Raghavan, and V.K. Elayavalli. Genetic algorithm for clustering with an ordered representation. In *Proc. International Conference on Genetic Algorithms'91*, pages 408–415, 1991.

[Did71] E. Diday. La méthode des nuées dynamiques. *Revue de Statistique Appliquée*, 19(2) :19–34, 1971.

[DPST03] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Métaheuristiques pour l'optimisation difficile*. Algorithmes. Eyrolles, 2003.

[Dru93] G. Drueck. New optimization heuristics, the great deluge and the record to record travel. *Journal of Computational Physics*, 104 :86–92, 1993.

[Dub89] Raymond W. Klein Richard C. Dubes. Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, 22(2) :213–220, 1989.

[Fra57] A. S. Fraser. Simulation of genetic systems by automatic digital computers. i. introduction. *Aust. J.Biool. Sci.*, 10 :484–491, 1957.

[Glo86] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.*, 13(5) :533–549, 1986.

[HBV01] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3) :107–145, 2001.

[HÖB99] Lawrence O. Hall, Ibrahim Burak Özyurt, and James C. Bezdek. Clustering with a genetically optimized approach. *IEEE Trans. on Evolutionary Computation*, 3(2) :103–112, 1999.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering : a review. *ACM Computing Surveys*, 31(3) :264–323, 1999.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598 :671–680, 1983.

[LA00] C. Lee and E. Antonsson. Dynamic partitional clustering using evolution strategies. In *In Proceedings of the Third Asia Pacific Conference on Simulated Evolution and Learning.*, 2000.

[Min98] Thomas P. Minka. Expectation-maximization as lower bound maximization. Technical report, 1998.

[MR94] S.K. Mishra and V.V. Raghavan. An empirical study of the performance of heuristic methods for clustering. *Pattern Recognition in Practice*, pages 425–436, 1994.

[MRR+53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. of Chem. Physics*, 21 :1087–1092, 1953.

[NW02] Michael K. Ng and Joyce C. Wong. Clustering categorical data sets using tabu search techniques. *Pattern Recognition*, 35(12) :2783–2790, 2002.

[RB79] Vijay V. Raghavan and Kim Birchard. A clustering strategy based on a formalism of the reproductive process in natural systems. In *SIGIR '79 : Proceedings of the 2nd annual international ACM SIGIR conference on Information storage and retrieval*, pages 10–22, New York, NY, USA, 1979. ACM Press.

[Sev04] Marc Sevaux. Métaheuristiques stratégies pour l'optimisation de la production de biens et de services. Technical report, UVHC, 2004.

[SJ00] Chang Sup Sung and Hyun Woong Jin. A tabu-search-based heuristic for clustering. *Pattern Recognition*, 33(5) :849–858, 2000.

[Tai03] E. Taillard. A statistical test for comparing success rates. In *Metaheuristic international conference MIC'03, Kyoto, Japan*, August 2003.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[Vel05] Julien Velcin. *Extraction automatique de stéréotypes à partir de données symboliques et lacunaires*. PhD thesis, UPMC, 2005.

[VG05] J. Velcin and J.G. Ganascia. Stereotype extraction with default clustering. In *In Proceedings of the 19th International Joint Conference on Artificial Intelligence. Edinburgh, Scotland*, 2005.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining : Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

[XW05] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3) :645–678, 2005.