Optimisation multiobjectif discrète par propagation de contraintes

Tristram Gräbener, Alain Berro
IRIT — Université de Toulouse
tristram.grabener@irit.fr, alain.berro@irit.fr

Abstract

Real-life problems require to optimise contradictive objectives. The solution choosen by a decision-maker will be a compromise depending on a large amount of human parameters varying from one decision-maker to an other and therefore difficult to modelize. Thus an automatized method will not be able to choose one solution from all compromise solutions, but should show them all to allow the decision-maker to make the best choice.

Most algorithms are based on genetic algorithms. Modelling and fine-tuning the parameters are very difficult for anyone who is not an expert of that field. Furthermore those algorithms do not allow to rate the quality of the solutions they offer.

Our approach uses the ease of modeling of *constraint programming* and gives an upper bound of the optimal solutions. Applied to a problem with discrete variables, it allows to obtain the exact Pareto front.

Résumé

La majorité des problèmes réels nécessitent l'optimisation selon des objectifs contradictoires. La solution choisie par un décideur sera un compromis dépendant d'un grand nombre de paramètres variant d'un décideur à un autre et donc difficiles à modéliser. Une méthode automatisée ne pourra pas effectuer un choix parmi toutes les solutions de compromis, mais devra les présenter pour que le décideur effectue le meilleur choix.

Les algorithmes sont généralement basés sur des algorithmes génétiques rendant la modélisation et le réglage des paramètres complexes pour une personne pas spécialisée dans le domaine. De plus évaluer la qualité des solutions proposées est difficile.

Notre approche permet d'utiliser la facilité de modélisation de la programmation par contraintes et permet l'obtention d'une borne supérieure du problème. Sur un problème à variables discrètes, elle permet d'obtenir le front de Pareto exact.

1 L'optimisation multiobjectif

1.1 Enjeux

L'optimisation multiobjectif répond au besoin de satisfaire des besoins contradictoires. Ainsi pour se rendre de Toulouse à Nantes, on cherchera à minimiser le coût, le temps passé, la et maximiser le confort. L'avion sera cher, rapide, et polluant tandis que le vélo sera économique, long et peu polluant. Enfin le train sera une solution de *compromis*.

Comme il n'existe aucune solution meilleure en tout point qu'une autre, un compromis différent selon les personnes doit être choisi. Le choix est donc subjectif, et il est indispensable de proposer l'ensemble des choix possibles afin ne pas exclure une possibilité.

L'optimisation multiobjectif est donc avant tout un outil d'aide à la décision, et c'est une personne qui prendra la décision finale.

1.2 Notations

Les m *variables de décision* sont les valeurs à choisir dans un problème d'optimisation. Ces valeurs sont notées $x_i, i \in \{1, \ldots, m\}$. Le vecteur $\mathbf{x}$ de m variables de décision est représenté par $\mathbf{x} = (x_1, x_2, \ldots, x_m)$. Les vecteurs de variables de décision forme le vecteur $\mathcal{X}$.

Les n *fonctions objectifs* à optimiser sont notées $f_i, i \in \{1, \ldots, n\}$ et le vecteur $\mathbf{f}$ de n objectifs pour $\mathbf{x} \in \mathcal{X}$ est représenté par : $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ Seul le cas de minimisation est traité sans perte de généralité : maximiser f_i revient à minimiser $-f_i$.

Les g *contraintes* limitent les valeurs que peuvent prendre les variables de décision. Elles sont notées $g_i(\mathbf{x}), i \in \{1, \ldots, p\}$.

Un *problème d'optimisation multiobjectif* est donc un triplet $(\mathcal{X}, \mathbf{f}, \mathbf{g})$ qui consiste à minimiser $\mathbf{f}(\mathbf{x})$ pour $\mathbf{x} \in \mathcal{X}$ sachant que $\mathbf{g}(\mathbf{x}) \leq 0$.

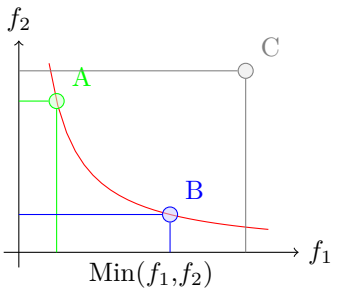

FIG. 1 – Exemple de front de Pareto

1.3 Domination et front de Pareto

Soit $\mathbf{u} = (u_1, u_2, \ldots, u_m)$ et $\mathbf{v} = (v_1, v_2, \ldots, v_m)$ deux vecteurs de décision. $\mathbf{v}$ *domine* $\mathbf{u}$ ($\mathbf{u} \prec \mathbf{v}$) pour un problème de *minimisation* si et seulement si

$$\begin{cases} \forall i \in 1, \ldots, n & f_i(\mathbf{v}) \leq f_i(\mathbf{u}) \\ \exists i \in 1, \ldots, n & f_i(\mathbf{v}) < f_i(\mathbf{u}) \end{cases}$$

Le *front de Pareto* $\mathcal{FP}^*$ est l'ensemble des vecteurs de décision qui ne sont pas dominés.

$$\mathcal{FP}^* = \{\mathbf{x} \in \mathcal{X} \mid \nexists \mathbf{x}' \in \mathcal{X} \quad \mathbf{x} \prec \mathbf{x}'\}$$

Le front de Pareto est l'ensemble des solutions de compromis. Sur la figure 1, les points A et B sont deux points du front de Pareto : A ne domine pas B, B ne domine pas A, mais tous les deux dominent le point C. Le but de l'optimisation multiobjectif est de déterminer le front de Pareto pour un problème donné.

1.4 Méthodes de résolution

Une première approche naïve habituelle consiste à traiter successivement les objectifs. Le résultat favorise alors des solution extrêmes (location d'un Concorde ou marche à pied). Une approche un peu plus évoluée consiste à pondérer les différents objectifs. Cependant cette approche ne laisse pas le choix au décideur. De plus certaines solutions non-dominées ne peuvent pas être obtenues quelque soient les coefficients (lorsque le front de Pareto n'est pas convexe). Enfin il est généralement difficile de faire la somme de certaines grandeurs (p. ex. pollution et niveau de confort).

Il existe un certain nombre de méthodes de résolution spécifiques à un problème permettant de déterminer le front de Pareto. Bien que performantes et permettant parfois d'obtenir l'ensemble du front, ces approches présentent bien évidemment le défaut de ne pas pouvoir être employées pour un problème quelconque.

La majorité des approches génériques sont basées sur des algorithmes génétiques. Les approches réputées les plus performantes sont NSGA-2 et SPEA-2 [7]. Basées sur des algorithmes génétiques, elles permettent de s'adapter à un grand nombre de problèmes. Cependant le réglage est délicat et sans sens pour l'utilisateur de l'algorithme. Ainsi les taux de croisement et de mutation n'ont pas de signification pour le problème à modéliser. À ces paramètres il faut rajouter les paramètres spécifiques aux algorithmes génétiques multiobjectif pour maintenir la diversité et obtenir le front de Pareto le plus large possible.

De plus ces approches peuvent avoir des problèmes de performance lorsque le problème est fortement contraint. Enfin ces algorithmes ne permettent pas d'avoir la moindre information sur le front de Pareto. Il est donc impossible d'avoir une estimation de la qualité des solutions trouvées.

Cependant la tendance des recherches de ces dernières années consiste à améliorer les algorithmes prédominants [6] et non pas à en faciliter l'accès.

2 La programmation par contraintes

Un problème de satisfaction de contraintes (*CSP*) [4] est défini par un ensemble de variables, un ensemble de domaines pour ces variables, et un ensemble de contraintes sur ces variables. Le but est d'assigner à toutes les variables une valeur de son domaine en respectant les contraintes. Une telle affectation est appelée *instanciation*.

La programmation par contraintes permet de définir un problème de manière déclarative sans nécessiter de connaissance sur la méthode de résolution. Le grand nombre de contraintes [5] étudiées permet de modéliser des problèmes complexes.

2.1 Consistance des contraintes

Une contrainte est dite *consistante* si elle ne permet pas de réduire le domaine des variables sur lesquelles elle porte. L'*arc-consistance* est employée lorsque le domaine des variables est une énumération et qu'il est possible de retirer une valeur du domaine. L'arc-consistance ne peut bien évidemment n'être utilisée que s'il s'agit de variables discrètes. La *consistance aux bornes* ne porte que sur les bornes du domaine d'une variable. Le filtrage est moins efficace, mais plus rapide, moins gourmand en mémoire et utilisable avec des variables continues.

Par exemple pour les variables discrètes $v_1 \in \{2, 3, 4\}$ et $v_2 \in \{1, 2, 3\}$ et la contrainte d'infériorité $v_1 \leq v_2$ on déduit que $v_1 \in \{2, 3\}$ et $v_2 \in \{2, 3\}$.

Pour les variables continues $v_1 \in [1; 3]$ et $v_2 \in [-1; 4]$ et la contrainte $v_2 = {v_1}^2$ on déduit que $v_1 \in [1; 2]$ et $v_2 \in [1; 2]$.

2.2 Résolution et optimisation d'un CSP

La résolution d'un *CSP* consiste à réduire les domaines selon les contraintes et éliminer les valeurs impossibles. Lorsqu'un point fixe est atteint (toutes les contraintes sont consistantes) un choix arbitraire est effectué sur une variable, quitte à revenir sur ce choix.

La programmation par contraintes n'est pas une technique d'optimisation en soi. Pour cela la fonction objectif est modélisée par une variable reliée aux autres variables par une contrainte. Pour minimiser l'objectif, tant qu'il existe une solution réalisable, la borne supérieure du domaine de l'objectif est successivement réduite dans le cas d'une minimisation.

Les *Weighted-CSP* (*WCSP*) attribuent un valeur à chaque contrainte pouvant être violée. L'objectif est de minimiser la somme des valeurs des contraintes violées. Similairement les *Max-CSP* tentent de maximiser le nombre de contraintes satisfaites. Il s'agit donc de problèmes d'optimisation.

3 Programmation par contraintes et optimisation multiobjectif

D'une part, les techniques d'optimisation multiobjectif présentent des problèmes de modélisation, un paramétrage délicat, une mauvaise convergence si le problème est fortement contraint et ne permettent pas d'estimer une borne du front de Pareto. D'autre part la programmation par contraintes excelle par la simplicité de sa modélisation et exploite les contraintes pour trouver une solution plus rapidement.

Malgré cela peu de tentatives ont été faites pour exploiter les possibilités de la programmation par contraintes pour l'optimisation multiobjectif.

L'approche la plus remarcable est la méthode *mini-bucket elimination* (MBE) qui permet d'obtenir une borne inférieure du front de Pareto (en minimisation) [8]. Une recherche arborescente de type *branch-and-bound* permet alors d'obtenir le front de Pareto. À chaque nœud de l'arbre, une solution réalisable est comparée à la borne inférieure pour déterminer s'il est nécessaire de descendre plus bas dans l'arbre ou pas.

Cette approche a des résultats prometteurs lorsque le vecteur objectif est la somme de fonctions portant sur peu de variables. L'utilisation de contraintes globales donnera donc généralement une borne de mauvaise qualité.

De plus cette approche reste limitée au cadre des *WCSP* qui consistent à définir une pénalité pour chaque contrainte violée et minimiser la somme des contraintes violées. La modélisation d'une fonction objectif qui ne soit pas associée à des contraintes n'est donc pas naturelle.

3.1 PICPA

Vincent Barichard propose une nouvelle approche basée sur une population d'individus pour encadrer le front de Pareto [3] : *Population and Interval Constraint Propagation Algorithm (PICPA)*. Chaque individu est représenté par un hypercube de l'espace des m variables, c'est-à-dire un sous-ensemble de $\mathbb{R}^m$.

Ces individus sont construits tels que deux individus soient disjoints dans l'espace des objectifs. Pour cela il applique des coupes successives dans l'espace des objectifs pour obtenir deux individus disjoints. Effectuer ces coupes est possible grâce à la propagation des contraintes de l'espace des objectifs vers l'espace des variables.

Pour chaque individu une solution *réalisable* est cherchée grâce à une recherche locale ou grâce à un algorithme évolutionnaire. Les individus sont éliminés si toutes les instanciations de cet individu sont dominées par une instanciation d'un autre individu. Ainsi les coupes et éliminations successives réduisent l'espace en encadrant de plus en plus près le front de Pareto.

Cette approche a été testée avec succès dans le cadre de variables continues. Nous proposons de l'adapter dans le cadre de variables discrètes avec cependant quelques variations :

- les variables discrètes permettent d'obtenir le front de Pareto *in extenso* ;
- le catalogue de contraintes sur les variables discrètes est beaucoup plus riche et permettra la modélisation de problèmes plus compliqués ;
- il est possible de savoir si un individu possède ou non une instanciation réalisable.

3.2 PS-Dominance

Vincent Barichard introduit la notion de *Point-Set dominance* (*PS-dominance*) qui étend la notion de domination pour comparer un point à un ensemble de points.

Soit un point $\mathbf{x}$ et un ensemble de points $\mathcal{Y}$, on dit

- $\mathbf{x}$ PS-domine $\mathcal{Y}$ ssi $\forall \mathbf{y} \in \mathcal{Y}, \quad \mathbf{y} \prec \mathbf{x}$
- $\mathbf{x}$ est PS-dominé par $\mathcal{Y}$ ssi $\forall \mathbf{y} \in \mathcal{Y}, \quad \mathbf{x} \prec \mathbf{y}$
- $\mathbf{x}$ est PS-égal à $\mathcal{Y}$ ssi $\forall \mathbf{y} \in \mathcal{Y}, \quad \mathbf{x} = \mathbf{y}$
- $\mathbf{x}$ est PS-non-dominé par $\mathcal{Y}$, réciproquement sinon

Une condition nécessaire pour avoir la PS-dominance est que $\mathbf{x}$ domine l'hypercube défini par les valeurs extrèmes de $\mathcal{Y} = \mathcal{Y}_1 \times \ldots \times \mathcal{Y}_m$

- $\mathbf{x}$ PS-domine $\mathcal{Y}$ si $\forall i \in 1, \ldots, m \quad x_i \leq \inf(\mathcal{Y}_i)$
- $\mathbf{x}$ est PS-dominé par $\mathcal{Y}$ si $\forall i \in 1, \ldots, m \quad x_i \geq \sup(\mathcal{Y}_i)$
- la PS-dominance est indéterminée sinon

Par la suite nous nous contenterons de cette conditions pour déterminer la dominance pour réduire le nombre de test à effectuer.

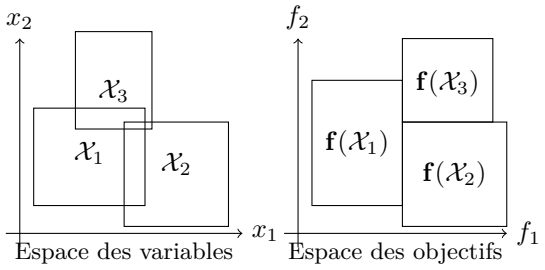

FIG. 2 – Individus dans les deux espaces

3.3 Encadrement du front de Pareto

Soit f une fonction de $\mathbb{R}^m$ dans $\mathbb{R}^n$ à optimiser. Un individu I est un CSP tel que :

- les variables $\mathbf{x}$ sont dans $\mathcal{X} \subset \mathbb{R}^m$
- les objectifs $\mathbf{f}$ sont dans $f(\mathcal{X}) \subset \mathbb{R}^n$
- la contrainte $\mathbf{f} = f(\mathbf{x})$ lie les objectifs aux variables

Soit $\mathcal{P}$ une population d'individus. On impose que pour deux individus I et I' $f(\mathcal{X}) \cap f(\mathcal{X}') = \varnothing$: les individus sont disjoints dans l'espace des objectifs. Ils peuvent par contre se chevaucher dans l'espace des variables (fig 2).

La population est initialisée avec un individu I_0 engloband l'espace des variables initial. L'algorithme choisit un individu *père* et effectuera une bissection selon un objectif. Ainsi un nouvel individu *fils* est généré et les individus père et fils sont disjoints dans l'espace des objectifs.

Ce nouvel individu est rajouté à la population. Tous les individus sont instanciés et tous les individus dominés par l'instanciation d'un autre individu et les individus n'ayant pas de solution réalisable sont supprimés.

À chaque itération l'espace des objectifs est réduit en garantissant :

- aucune élément du front de Pareto n'est éliminée ;
- la convergence vers le front de Pareto en un nombre fini d'itérations.

Ainsi l'égalité $\mathcal{FP}^* = \bigcap_i \mathbf{f}(\mathcal{X}_i)$ est toujours vraie.

La manière la plus naturelle de sélectionner un individu est de choisir celui dont les objectifs dessinent le plus grand espace, et d'effectuer la coupe selon l'objectif ayant le plus grand domaine.

La condition d'arrêt peut être :

- tous les individus sont réduits à un point dans l'espace des fonctions. Les individus représentent alors l'ensemble du front de Pareto ;
- un certain nombre d'itération ou temps ;
- un certain nombre d'individus a été généré [3] ;
- le front est encadré à un intervalle donné.

Dans les trois derniers cas, seul encadrement du front de Pareto est obtenu.

Algorithme 1 : Encadrement du front de Pareto

```
P ← {I_0}
tant que ¬ condition_arrêt faire
    I ← sélectionner_individu(P) ;
    i ← sélectionner_objectif(I) ;
    I' ← cloner(I) ;
    milieu ← (sup(f_i)+inf(f_i))/2 ;
    contraindre(f_i ≤ ⌊milieu⌋) ;
    contraindre(f'_i ≥ ⌈milieu⌉) ;
    P ← P ∪ I';
    pour I ∈ P faire
        propager_contraintes(I) ;
        instancier(I) ;
    pour I, I' ∈ P² faire
        si instanciation(I) PS-domine I' alors
            P ← P − I';
```

3.4 Limites

Même si la convergence est garantie, et que le nombre de coupes à effectuer est de l'ordre du nombre de fonctions à optimiser, chaque itération de l'algorithme nécessite la résolution d'un *CSP* pour chaque individu qui est NP-difficile. La performance de l'algorithme est donc directement liée à la vitesse de résolution du *CSP*.

Cependant si le problème est trop grand pour être résolu dans un temps acceptable, l'algorithme propose à tout instant un encadrement du front de Pareto pouvant donc fournir des solutions suffisantes.

Un autre facteur potentiellement limitant est la nature des fonctions objectifs. En effet si les contraintes associées à ces objectifs ne permettent pas de réduire le domaine des variables, l'espace des variables à explorer est significativement plus grand.

4 Premiers tests

4.1 Problème

Nous avons testé notre algorithme sur le problème du sac-à-dos multiobjectif 0-1 (MOKP01). Le problème consiste à choisir un sous-ensemble de m objets maximisant les n profits en respectant les contraintes de poids. Formellement :

$$\begin{cases} \max & f_j(x) = \sum_{i=0}^{m} x_i \cdot p_i^j & j = 1, \ldots, n \\ \text{t. q.} & \sum_{i=0}^{m} x_i \cdot w_i^j \leq w_j & j = 1, \ldots, n \\ & x_i \in \{0, 1\} & i = 1, \ldots, m \end{cases}$$

L'algorithme a été implémenté autour de la bibliothèque de programmation par contraintes java *Choco* [1]. Le programme a été éxécuté sur un Power Mac 2,66 Ghz et 4 Go de mémoire.

Objets m	Dimensions n	Coupes	Temps (s)
10	1	9	0,5
20	1	10	1,7
30	1	9	41
40	1	9	$4,14 \cdot 10^3$
10	2	65	2,3
20	2	248	28
30	2	424	$4,68 \cdot 10^3$
40	2	631	$5,54 \cdot 10^5$
10	3	224	5,5
20	3	1588	336
30	3	4160	$7,64 \cdot 10^4$

TAB. 1 – Résultats expérimentaux pour le MOKP

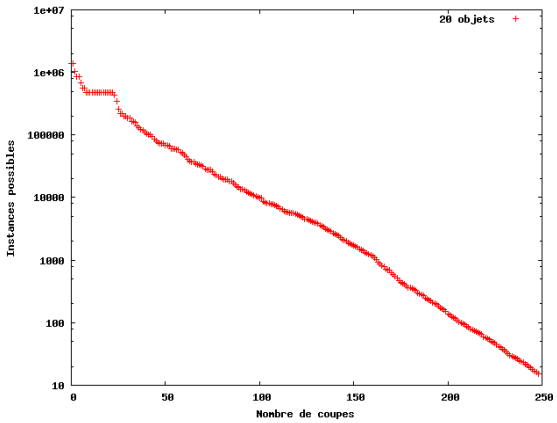

FIG. 3 – Évolution de la taille du domaine

4.2 Résultats

Le problème a été testé avec plusieurs instances basées sur celles proposées par [2] dont la liste d'objets a été tronquée pour obtenir le nombre voulu. La capacité a été adaptée en fonction du nombre d'objets avec une règle de trois.

La figure 3 montre le nombre d'instanciations possibles en fonction du nombre de coupes effectuées pour 20 objets et 2 objectifs. Elle démontre la convergence vers la solution optimale.

Le tableau 1 présente le résultat des expériences. Le nombre de coupes à effectuer évolue peu en fonction du nombre de variables, cependant le temps d'exécution est exponentiel. Cela confirme de l'importance de pouvoir instancier efficacement le CSP.

Pour un même CSP, gérer plus d'objectifs n'augmente pas significativement le nombre de coupes à effectuer ni le temps d'éxecution. Nous supposons que le nombre de coupes à effectuer est une fonction polynomiale du nombre de dimensions.

Considérant un espace des variables de n dimensions dont le plus grand cardinal est c, l'espace comporte au plus c^n éléments. Le front de Pareto comporte au moins un élément (cas où le plus de coupes sont nécessaires). Soit k le nombre maximal de coupes à effectuer. Chaque coupe divise l'espace des variables en deux. On a donc $\frac{c^n}{2^k} = 1$. On en déduit $n\log(c) - k\log(2) = \log(1)$ soit $k = \mathcal{O}(n)$.

Si cela est prouvé par plus d'expérimentations, ce serait un avantage significatif par rapport aux autres approches qui ont généralement des problèmes pour gérer un grand nombre d'objectifs.

5 Perspectives

Nous avons démontré la faisabilité d'adapter *PICPA* à des problèmes à variables discrètes. Il reste cependant de nombreuses pistes à explorer. Il faut en effet tester l'approche sur d'autres problèmes et en particulier des problèmes du monde réel.

Le facteur limitant étant l'instanciation des individus, tandis que la selection des individus Pareto-optimaux nécessite un échange très faible d'informations, cette approche est adaptée afin d'être parallélisée.

Enfin, afin de faciliter la modélisation de nouveaux problèmes une meilleure intégration avec le solveur *CSP* (*Choco* dans notre cas) est indispensable.

References

[1] `http://choco-solver.net`.

[2] `http://www.tik.ee.ethz.ch/sop/download/supplementary/testProblemSuite/`.

[3] V. Barichard and J.K. Hao. A population and interval constraint propagation algorithm. *Lecture Notes in Computer Science*, 2632:88–101, 2003.

[4] R. Bartak. Constraint Programming: In Pursuit of the Holy Grail. 1999.

[5] N. Beldiceanu, M. Carlsson, and J.X. Rampon. Global constraint catalog. *Research Report T*, 2005.

[6] C.A.C. Coello, S. de Computacion, and C.S.P. Zacatenco. 20 Years of Evolutionary Multi-Objective Optimization: What Has Been Done and What Remains To Be Done. 2006.

[7] C.A.C. Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2007.

[8] E. Rollón and J. Larrosa. Bucket elimination for multiobjective optimization problems. *Journal of Heuristics*, 12(4):307–328, 2006.