Gestion des erreurs numériques dans un solveur d'optimisation : un art entre théorie et pratique

Simon Boulmier

LocalSolver, 24 avenue Hoche, Paris, France
sboulmier@localsolver.com

Mots-clés : *optimisation globale, arithmétique flottante, erreurs numériques, C++*

1 Introduction

Pour une théorie mathématique donnée, la performance d'un outil d'optimisation numérique basé sur celle-ci se mesure selon deux critères. Premièrement la vitesse de d'exécution - et donc de résolution, qui est déterminée par de nombreux facteurs. La vitesse de convergence théorique des algorithmes utilisés, la capacité d'une implémentation de ces algorithmes à atteindre effectivement cette vitesse de convergence, les structures de données utilisées, la complexité des algorithmes qui les utilisent, et, enfin, la qualité d'ingénierie logicielle qui met l'ensemble en musique sont les critères majeurs qui déterminent la vitesse de résolution d'une instance.

L'autre critère important pour évaluer les performances d'un solveur est la fiabilité des résultats obtenus. En effet, si un solveur renvoie un résultat faux (par exemple une solution violant des contraintes ou déclarée optimale alors qu'une autre solution de meilleure valeur est connue de l'utilisateur), alors quelque soit sa vitesse d'exécution, celui-ci ne sera pas retenu pour résoudre le problème.

Ces deux axes, souvent antagoniques, constituent les principales difficultés d'impleméntation d'un solveur. La quantité d'ingénierie dédiée à leur amélioration est souvent ce qui différencie une implémentation basique d'un solveur industriel. On peut citer par exemple les algorithmes du simplexe dual ou des points intérieurs pour la programmation linéaire. Ces algorithmes se décrivent sur papier de façon concise, mais la différence avec des implémentations industrielles fortes de plusieurs décennies de peaufinage est criante.

2 Philosophie de la présentation

Pendant cette présentation, nous nous intéresserons à l'étude de la robustesse numérique du solveur d'optimisation globale présent dans LocalSolver [1]. L'objectif est de détailler quelques cas d'étude sur des composants précis d'implémentation.

Plus précisément, le contexte est celui de l'optimisation globale des programmes non linéaires mixtes (MINLP). Un solveur global rassemble des outils issus de plusieurs domaines, comme la programmation par contraintes, l'analyse convexe, la programmation mathématique (linéaire, quadratique, non linéaire, etc.) ou encore la combinatoire. Le moteur principale de tels solveurs est le calcul de bornes duales pour le problème d'optimisation à résoudre, ainsi qu'un processus de globalisation de la convergence *via* l'utilisation d'une certaine variante du principe de *branch-and-bound*.

Le logiciel ainsi obtenu est déterministe, mais chaotique : une différence minime (par exemple une permutation) du modèle initial donne une séquence d'actions dans le solveur qui change totalement. En présence d'erreurs numériques, ceci peut induire une solution optimale et un temps de résolution très variables sur des modèles en théorie isomorphes. Ainsi, la fiabilité des bornes calculées et la robustesse de chaque rouage numérique du solveur sont des éléments centraux d'une implémentation.

3 Cas d'étude

Arithmétique d'intervalle La source des erreurs numériques est le caractère approché de l'arithmétique flottante utilisée par les processeurs des ordinateurs. Ces erreurs pouvant se cumuler, certains outils, comme ceux de programmation par contraintes, ne sont pas viables en l'état, et doivent utiliser d'autres arithmétiques pour effectuer leurs calculs. La plus utilisée est l'arithmétique d'intervalle, qui exploite les propriétés garanties par la norme IEEE754 [3] de l'arithmétique flottante pour rendre les calculs exacts. Chaque nombre est représenté par un intervalle, et un résultat numérique est remplacé par un intervalle contenant (avec certitude) sa vraie valeur. Les difficultés d'implémentation d'une telle arithmétique, les interactions avec les compilateurs C++ ainsi que quelques exemples de problèmes imprévus seront présentés.

Direction de descente et recherche linéaire À chaque nœud d'un arbre de branchement, une relaxation convexe du MINLP est résolue. Sa valeur optimale fournit la borne duale du sous-problème associé au nœud. La capacité d'un solveur à résoudre toutes ces relaxations, en particulier dans des cas inconsistants ou mal conditionnés, est ainsi nécessaire au bon déroulement de la recherche arborescence. Dans le cas où les relaxations sont non linéaires, les solveurs utilisés sont le plus souvent basés sur le calcul d'une direction de recherche, ainsi que sur une recherche linéaire dans cette direction [5]. Dans ce cas, le solveur échoue si et seulement si à une certaine itération la recherche linéaire échoue. Si la théorie garantit que cette dernière doit nécessairement réussir, des erreurs numériques peuvent s'interposer et rendre certaines hypothèses nécessaires à sa convergence invalides. Nous nous intéresserons ainsi aux causes principales des échecs des solveurs non linéaires, puis à quelques éléments qui permettent de rendre une recherche linéaire plus robuste aux erreurs numériques.

Conditions de KKT approchées et bornes duales Pour une relaxation convexe d'un MINLP, la condition de KKT de stationnarité du lagrangien est celle qui permet d'obtenir des bornes duales. Cette condition est une égalité, et n'est donc jamais exactement vérifiée en arithmétique flottante. Nous analyserons ce que la stationnarité approchée (celle qui utilise des tolérances, comme dans presque tous les solveurs) implique pour la borne duale associée, et nous présenterons quelques approches possibles pour rendre cette borne robuste [2, 4].

Références

[1] Thierry Benoist et al. « Localsolver 1. x : a black-box local-search solver for 0-1 programming ». In : *4OR* 9.3 (2011), p. 299.

[2] Christian Jansson. « A rigorous lower bound for the optimal value of convex optimization problems ». In : *journal of global optimization* 28.1 (2004), p. 121-137.

[3] William Kahan. « IEEE standard 754 for binary floating-point arithmetic ». In : *Lecture Notes on the Status of IEEE* 754.94720-1776 (1996), p. 11.

[4] Yahia Lebbah, Claude Michel et Michel Rueher. « Using constraint techniques for a safe and fast implementation of optimality-based reduction ». In : *Proceedings of the 2007 ACM symposium on Applied computing*. ACM. 2007, p. 326-331.

[5] Jorge Nocedal et Stephen J Wright. *Numerical Optimization*. Springer, 2006.