Placement optimisé d'opérateurs arithmétiques

Mario Flores Gómez[1], Roselyne Chotin[1], Lilia Zaourar[2]

[1] LIP6, 4 Place Jussieu, 75005 Paris
mario.flores-gomez,roselyne.chotin@lip6.fr
[2] CEA LIST, Paris-Saclay Campus - Nano-INNOV, F-91191 Gif-sur-Yvette Cedex,France
Lilia.ZAOURAR@cea.fr

Mots-clés : *Heuristique, représentation graphique, algorithme génétique.*

1 Introduction

L'affectation d'opérateurs pour la réalisation d'opérations arithmétiques de base présente des problèmes d'optimisation riches. On cherche à déterminer le meilleur circuit possible par rapport au temps, réalisant une somme entre plusieurs entrées. Il s'agit d'un problème facile du point de vue arithmétique. Il semble même trivial de dire que $a + b + c + d + e + f$ peut être factorisé sous la forme de $(a + b) + ((c + d) + (e + f))$ de façon optimale. Afin de trouver le circuit optimal, il suffit de remplacer chaque paire de parenthèses par un opérateur, dans ce cas, le plus rapide disponible pour optimiser la solution. Or, pour un problème plus général, dans lequel on retrouvera à la fois des opérateurs booléens et d'autres opérateurs arithmétiques nous aurons besoin de nous baser sur un modèle mathématique exploitable. Nous présentons dans ce travail un premier cas d'étude simple avec 5 types d'opérateurs d'addition possibles en entrée et une expression arithmétique à optimiser. Les travaux de [3] présentent 4 opérateurs d'addition parmi les plus performants du marché. Nous les avons choisis comme dictionnaire de référence pour le problème, chacun avec des points positifs et négatifs sur la performance en fonction du temps et de la taille. Nous cherchons une représentation optimisée de l'expression sous forme d'un circuit logique à l'aide des opérateurs. Le premier objectif est de minimiser le temps d'exécution, le second étant la surface.

2 Modélisation mathématique

Nous nous sommes inspirés du papier de Coello & Hernandez [2] pour la modélisation. Une solution pour notre problème consiste en une matrice dont le nombre de lignes est égal au nombre d'opérandes en entrée, et dont le nombre de colonnes est la taille de la pire solution connue pour le problème comme représenté dans la Figure 1. Dans notre cas, ce nombre est 6. Ceci revient à ajouter de manière successive les opérandes, donc à utiliser 5 opérateurs, l'un après l'autre. La première colonne présente juste les opérandes. Ceci est donc une borne supérieure. Dans le cas d'une expression plus longue et avec d'autres opérations que les sommes, un algorithme glouton est utilisé afin de trouver une solution réalisable dont la taille est une borne.

Chaque case de la matrice représente un opérateur, avec trois informations : le type, et les deux entrées. Chaque opérateur de chaque colonne autre que la première prend en entrée les éléments de la colonne précédente. Par exemple, si dans la deuxième colonne il y a un opérateur qui prend en entrée les opérandes 1 et 3, alors les indices correspondront aux lignes respectives de ces deux opérandes dans la première colonne, si dans la i-ème colonne un opérateur prend en entrée les sorties des opérateurs de la colonne précédente se trouvant à la ligne 5 et 6 de la matrice, les deux indices dans la case seront 5 et 6.

3 Résolution

Une fois le type des solutions établi, nous avons développé un algorithme basé sur le schéma des algorithmes génétiques inspiré également des travaux de [2] les principales étapes sont décrites ci-dessous :

1. La *population initiale* est générée aléatoirement avec un nombre d'individus (solutions possibles) suffisamment grand, puis nous la répartissons en sous-populations.
2. Pour chaque individu de la sous-population i nous vérifions à la i-ème colonne de cet individu que chaque élément de la colonne précédente a été pris en compte soit par un opérateur de somme, ou bien par une interconnexion dans un premier temps. Si ce n'est pas le cas on lui donne une évaluation de 0.

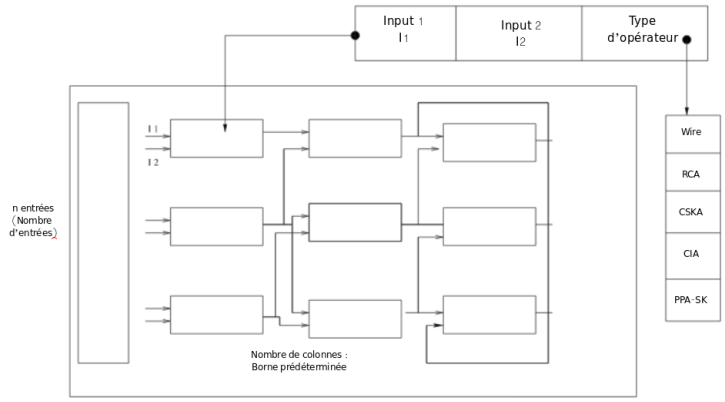

FIG. 1 – Matrice représentative d'une solution pour notre problème. Chaque case n'appartenant pas à la première colonne contient trois informations : l'indice de la ligne dont nous prenons la sortie dans la colonne précédente pour les deux entrées, et le type d'opérateur utilisé[2].

3. Pour chaque individu de la sous-population i n'ayant pas une évaluation de 0, on vérifie qu'il n'y ait pas des éléments répétés. Si c'est le cas on lui donne une évaluation négative correspondant au nombre d'opérateurs manquants par rapport au nombre d'opérandes/2 (-1, -2, -3) et de -3 si il y a ce nombre d'opérandes, sinon on lui donne une évaluation de -0.5. Ce processus est arbitraire, et **ne sert qu'à avoir une référence**. Ceci permet de réduire considérablement l'espace de recherche.
4. Pour chaque élément de la sous-population i dont l'évaluation est inférieure ou égale à -1 on vérifie qu'une solution réalisable existe en échangeant la première et i-ème colonne (une mutation).
5. La réalisabilité d'une solution est évaluée en fonction du temps d'exécution de **chaque opérateur**, pondéré par un coefficient correspondant à la colonne dans laquelle il se trouve. Plus la colonne est à droite plus ce coefficient est grand afin de valoriser les solutions les plus rapides, dont les opérateurs utilisés sont le plus à gauche de la matrice.
6. Les opérations de *crossover* et *mutation* sont ensuite appliquées afin de générer une nouvelle population. Ces étapes sont fondamentales [1] et permettent de faire évoluer la population. L'évaluation des solutions est importante pour déterminer les meilleures solutions. Pour la mutation nous procédons à un échange de colonnes de la matrice.
7. Les étapes de 2 à 7 sont répétées un certain nombre de fois (nombre d'itérations fixé par l'utilisateur) ou bien jusqu'à ne plus avoir d'amélioration par rapport à la meilleure solution trouvée.

Nous avons implémenté et testé cet algorithme sur différentes instances types. Nous présenterons les détails de notre implémentation ainsi que quelques résultats numériques obtenus. La mise en place de cette modélisation nous a permis de mieux comprendre la structure du problème et de proposer une autre modélisation du circuit sous la forme d'un graphe dirigé acyclique et d'un Programme Linéaire à Nombres Entiers pour sa résolution exacte.

Références

[1] M.C.Bhuvaneswari. Application of Evolutionary Algorithms for Multi-objective Optimization in VLSI and Embeded Systems. Springer, India :1–46,2015

[2] C.A.Coello and A.Hernandez Aguirre. Design of combinatorial logic circuits through an evolutionary multiobjective optimization approach. In *Artificial Intelligence for Engineering Design, Analysis and Manufacturing.* 2002

[3] R.Zimmermann. Lecture notes on Computer Arithmetic : Principles, Architectures, and VLSI Design. In *IEEE International Symposium on Computer Arithmetic.* 1999