Ordonnancement d'opérations utilisées pour obtenir des points sur des courbes elliptiques

Mario Flores Gómez[1], Roselyne Chotin[1], Joel Cathebras[1]

LIP6, 4 Place Jussieu, 75005 Paris

mario.flores-gomez,roselyne.chotin,joel.cathebras@lip6.fr

Mots-clés : *Courbes elliptiques, ordonnancement.*

1 Introduction

En cryptographie, les mathématiciens manipulent des objets mathématiques complexes. Certains de ces objets en vogue actuellement sont les courbes elliptiques. Les schémas de chiffrement qui les manipulent sont amenés à faire des opérations sur les points de ces courbes par une successions de calcul de tangentes, de droites et d'intersections avec lesdites courbes elliptiques. En pratique, ces manipulations sont décomposés en opérations de base sur le corps de définition de la courbe[2].

Chaque opération de base est elle-même décomposée en une suite d'opérations élémentaires (addition / soustraction et multiplications). En considérant le nombre d'opérations de courbe à effectuer dans un cas typique d'utilisation (de l'ordre de plusieurs centaines pour une opération de base au niveau du chiffrement), un gain même minime en temps de calcul sur l'ordonnancement d'une opération de base d'une courbe peu s'avérer non négligeable à l'usage. Ceci donne des problématiques d'ordonnancement intéressantes pour nous.

2 Modélisation mathématique

Il s'agit donc d'un problème d'ordonnancement d'opérations algébriques, qui doivent être exécutées dans un certain ordre et dans un nombre limité d'opérateurs disponibles du système. Les instances sur lesquelles nous avons travaillé sont des flux de données représentées par un graphe dirigé acyclique $G(N, A)$, où N est l'ensemble d'opérations qui doivent avoir lieu et A est l'ensemble des opérandes en entrée ou bien les sorties des opérateurs. L'ordonnancement doit prendre en compte le temps d'exécution des opérateurs, qui est différent entre les opérateurs de somme et les opérateurs de multiplication. Ce temps est calculé en nombre de cycles de calcul nécessaires pour les exécuter. De même, nous devons prendre en compte que chaque opérateur de multiplication a la capacité d'exécuter 3 opérations de façon simultanée, pendant que les additionneurs ne peuvent en exécuter qu'une à chaque fois ; cette différence est due à leur structure et à l'algorithme employé par chacun d'entre eux pour exécuter les calculs. Le nombre d'opérations qu'un opérateur peut traiter à la fois, est égale au nombre de voies de calcul présentes sur lui. Notre problème compte donc un ensemble d'opérations à exécuter $o \in [1, nb\_operations]$ numérotées et ayant un type (1 : somme, 2 : soustraction, 3 : multiplication) ; un certain nombre de relations de précédences par opération (0 s'il n'y a pas) du type $o_1 \prec o_2$[3], ce qui veut dire que o_1 doit être exécuté entièrement avant de pouvoir exécuter o_2 ; le nombre d'opérateurs d'addition (soustraction grâce au complément à deux) et de multiplication disponibles sur le système ; le nombre total d'opérateurs ($nb\_operateurs$) est la somme des deux ; un vecteur $c\_min[o], \forall o \in [1, nb\_operations]$ qui indique le premier cycle auquel une opération est éligible pour exécution en fonction des précédences et une borne $cycle\_max$, nombre de cycles disponibles dans le modèle.

Voici le modèle PLNE que nous proposons :

Variables :

— $a_{ovc} \in \{0, 1\}$: l'opération o est active à la voie v au cycle c.

— $b_{oc} \in \{0, 1\}$: l'opération o a déjà été exécutée au cycle c.

— $u_{ov} \in \{0, 1\}$: l'opération o est exécutée sur la voie v.

Nous avons une fonction objectif à maximiser en raison du nombre de cycles sans opération en exécution à la fin de l'ordonnancement, donc maximiser la somme de variables b_{oc}, ceci équivaut à dire que l'ordonnancement finit le plutôt possible.

$$max \sum_o \sum_c b_{oc}$$

s.c.

$$\frac{1}{nb\_cycles} \sum_c a_{ovc} \leq u_{ov} \quad \forall o, \forall v \quad (1)$$

$$\sum_v u_{ov} = 1 \quad \forall o \quad (2)$$

$$\sum_v \sum_c a_{ovc} = latence[o] \quad \forall o \quad (3)$$

$$a_{ovc} = 0 \quad \forall o, \forall v | type[o] \neq type[v], \forall c \quad (4)$$

$$a_{ovc} = 0 \quad \forall o, \forall v, \forall c | c\_min[o] > c \quad (5)$$

$$\sum_o a_{ovc} \leq 1 \quad \forall v, \forall c \quad (6)$$

$$\sum_{v \in i} \sum_o a_{ovc} \leq 1 \quad \forall i \in [1, nb\_operateurs], c = 1 \quad (7)$$

$$1 \geq \sum_{v \in i} \sum_o (a_{ovc-1} - a_{ovc}) \geq -1 \quad \forall i \in [1, nb\_operateurs], \forall c \geq 2 \quad (8)$$

$$b_{oc} = 0 \quad \forall o, \forall c | c\_min[o] + latence[o] > c \quad (9)$$

$$b_{oc} \leq \sum_v \sum_{c'=0}^{c} a_{ovc'} \quad \forall o, \forall c \geq 2 \quad (10)$$

$$\sum_v a_{ovc} \leq 1 - b_{oc} \quad \forall o, \forall c \quad (11)$$

$$b_{oc-1} \leq b_{oc} \quad \forall o, \forall c \geq 2 \quad (12)$$

$$\sum_v a_{o_2vc} \leq b_{o_1c-1} \quad \forall o_1 \prec o_2, \forall c \geq 2 \quad (13)$$

$$b_{o_2c} \leq b_{o_1c-1} \quad \forall o_1 \prec o_2, \forall c \geq 2 \quad (14)$$

Les contraintes (1) ci-dessus servent à donner les valeurs aux variables u_{ov} en fonction de a_{ovc}. La contrainte (2) force les opérations à être exécutées sur une voie uniquement. Le nombre de cycles sur lequel un opérateur doit réaliser une opération est égal à sa cadence, et c'est le nombre de cycles qu'une opération doit être active sur une voie (3). Une opération d'addition ne peut pas être exécutée sur une voie correspondante à un opérateur de multiplication et vice versa (4). Afin de réduire l'espace de recherche, nous interdisons pour chaque opération o d'être activée avant $c\_min[o]$ (5). La contrainte (6) assure qu'il n'y ait qu'une seule opération active par voie à chaque cycle. Finalement, (7) et (8) introduisent une contrainte particulière pour assurer le fonctionnement correct du système, car nous ne pouvons pas commencer l'exécution de deux ou plus opérations différentes au même cycle sur les voies appartenant au même opérateur, ceci est redondant pour les opérateurs d'addition qui ne possèdent qu'une seule voie, mais indispensable pour les multiplieurs car le fait d'occuper les voies simultanément pourrait ralentir leur temps d'exécution. Nous avons besoin de (8) pour $c = 1$ car les variables a_{ovc} ne sont pas définies pour $c = -1$.

Comme nous avons indiqué précédemment, les variables b_{oc} sont binaires, elles doivent valoir 1 à partir du moment où l'opération o a été entièrement exécutée. Pour les borner afin de pouvoir nous en servir nous commençons par réduire l'espace de recherche, nous forçons ces variables à être nulles quand c'est impossible que l'opération soit finie, d'une façon similaire à la contrainte (6), (9) ne permet pas aux variables b correspondantes à chaque opération de valoir 1 pour tout cycle c avant $c\_min[o] + latence[o]$ (la latence est un abus de langage car elle correspond à un opérateur et non pas à une opération), ce qui est une sorte de borne inférieure pour la fin de l'exécution de l'opération. Pour sa part, les contraintes (10) et (11) bornent les variables b par rapport aux variables a_{ovc} pour les lier entre elles et savoir à quel moment les opérations finissent leur exécution. (12) provoque que toutes les variables $b_{\hat{o}c}$ prennent une valeur de 1 du cycle où commence l'exécution de $\hat{o}$ jusqu'à $cycle\_max$. Les contraintes (13) et (14) assurent que les relations de précédence sont respectées, pour chaque relation $o_1 \prec o_2$, on assure que o_2 ne peut pas commencer (ni finir) avant la fin de o_1.

3 Résolution

Nous avons effectué plusieurs test sur CPLEX[1], sur des instances allant de 10 à 30 opérateurs. Nous avons réussi à obtenir des résultats avec des temps de calcul raisonnables (< 2 minutes). Nous présenterons l'ensemble des résultats. Par la suite une résolution gloutonne a été mise en place pour simplifier la manipulation des entrées et des résultats de l'ordonnancement pour les gens sans formation en Recherche Opérationnelle.

Références

[1] IBM. Optimization Studio CPLEX User's Manual. Version 12 Release 6.

[2] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen et Frederik Vercauteren. Handbook of elliptic and hyperelliptic curve cryptography. Chapman & Hall/CRC, 2006 (ISBN 9781584885184, OCLC 58546549)

[3] Floyd,R.W. Juillet. Syntactic Analysis and Operator Precedence. *Journal of the ACM.* 1963