

Rapport sur l'Optimisation Combinatoire



$P = NP$
 $\Leftrightarrow N = 1$

Julien GIRAUD, Luna NADJARE
G2S4
UNIVERSITÉ LYON 1

Introduction

L'optimisation combinatoire est un concept mathématique qui occupe une place forte dans la résolution de problèmes par informatique. À l'origine il s'agit de prendre un problème combinatoire et de le résoudre. De façon générale, n'importe quel problème combinatoire peut être résolu de la façon suivante : on énumère toutes les possibilités du problème et on ne garde que la ou les meilleures solutions. Le problème est que la combinatoire est gigantesque, voire incalculable à l'échelle de la vie de l'univers tout entier ! L'objectif est alors de trouver un moyen de réduire fortement les possibilités pour résoudre le problème en un temps raisonnable. Autrement dit, l'optimisation combinatoire c'est la maximisation de la qualité de la résolution du problème.

Le problème de la complexité

Pour bien comprendre le problème de l'optimisation combinatoire il faut parler un peu de la complexité des algorithmes de résolution des problèmes combinatoires. On va supposer que notre problème dépend d'une variable N , un entier positif non nul. Le temps de résolution du problème varie de différentes façons, et en fonction de N (à moins que le temps de résolution soit constant mais ce n'est pas le cas ici).

Dans le pire des cas, la complexité est de $\exp(N)$. Il est alors impossible de tester toutes les combinaisons dès que N dépasse quelques dizaines. Ensuite il y a la complexité polynomiale : NP. Certains problèmes qui semblent être $\exp(N)$ peuvent être résolus avec des algorithmes NP. Il devient alors possible de faire varier N vers de plus grands nombres, mais dans les limites du calculable.

Enfin il y a la complexité égale à N . Et la meilleure complexité est du type $\log(N)$, à ce moment là, plus la valeur de N est grande, moins il faut de temps pour résoudre le problème (par rapport au nombre de combinaisons et au temps qu'il faut pour les traiter toutes).

Le concept de la programmation par contrainte

Comme nous avons vu précédemment, la complexité des algorithmes de résolution des problèmes combinatoires pose plusieurs problèmes. D'une part nous savons qu'il existe plusieurs algorithmes pour résoudre ces problèmes et certains d'entre eux ont une complexité plus faible que les autres. D'autre part, peu importe la complexité de l'algorithme, la combinatoire peut être gigantesque et il nous faut des moyens de réduire les possibilités pour accélérer la recherche de solutions. C'est là qu'intervient la programmation par contrainte.

Son principe est plutôt simple, on sait que certaines combinaisons sont impossibles, il n'est donc pas nécessaire de les tester. Ainsi on va raisonner sur les contraintes du problème afin d'éliminer toutes les possibilités inutiles. Pour cela on va exprimer le problème selon des variables dont on connaît les contraintes. Résoudre le problème va donc se résumer à trouver les valeurs des variables pour satisfaire les contraintes.

Modélisation du problème

On appelle « modélisation du problème » la façon dont on va écrire les contraintes de notre problème, il s'agit finalement de l'étape la plus difficile de la programmation par contrainte. Encore faut-il savoir ce qu'est une contrainte, et de quelle façon fonctionne une variable.

Une variable est un élément qui peut prendre une valeur d'un ensemble connu (0, 3 ou 4 par exemple). Ensuite une contrainte est une relation mathématique avec une ou plusieurs variables ($x > y$ ou $x < 3$ par exemple).

Une fois notre problème modélisé, il ne reste plus qu'à le donner à un solveur de problème à contraintes. En fait on passe d'un problème d'optimisation combinatoire à un problème de satisfaction de variables.

Solveur de problème à contraintes (CSP)

L'objectif d'un CSP est très simple, il s'agit de trouver pour chaque variable, la ou les valeurs de son domaine telles que les contraintes soient satisfaites. Celui-ci va commencer par enlever les valeurs inutiles dans les domaines. Ensuite il va tester des combinaisons de valeurs parmi celles qui restent, en faisant des ajustement intelligents (pour éviter d'ajuster la valeur de la mauvaise variable par exemple) jusqu'à pouvoir conclure sur la ou les solutions s'il en a trouvé. Il s'agit là d'une grosse vulgarisation mais dans l'idée c'est à peu près ça.

L'avantage est qu'il s'agit de programmation déclarative, il n'est donc pas nécessaire de savoir développer le programme de résolution pour résoudre un problème d'optimisation combinatoire de cette façon. Cependant il faut se conformer aux normes de formalisation du CSP.

Il existe de nombreux CSP, les deux gratuits les plus connus sont choco-solver.org et gecode.org.

Exemple d'implémentation

Pour l'implémentation nous avons étudié l'exemple disponible sur la page d'accueil de choco-solver. Cet exemple permet de résoudre le problème musical des séries dodécaphoniques. Une série dodécaphonique est une succession des douze sons du total chromatique sans que l'un d'entre eux ne soit répété. Autrement dit ça revient à générer des séquences musicales atonales (contraire de l'harmonie tonale) à la suite sans avoir deux fois la même séquence.

Exemple de série dodécaphonique d'ordre 4.

Au final le programme va générer N séquences de 12 nombres compris entre 0 et 11 (correspondant aux 12 notes possibles).

```

1 // http://www.csplib.org/Problems/prob007/
2
3 int N = 100;
4 // 1. Modelling part
5 Model model = new Model("all-interval series of size "+ N);
6 // 1.a declare the variables
7 IntVar[] S = model.intVarArray("s", N, 0, N - 1, false);
8 IntVar[] V = model.intVarArray("V", N - 1, 1, N - 1, false);
9 // 1.b post the constraints
10 for (int i = 0; i < N - 1; i++) {
11     model.distance(S[i + 1], S[i], "=", V[i]).post();
12 }
13 model.allDifferent(S).post();
14 model.allDifferent(V).post();
15 S[1].gt(S[0]).post();
16 V[1].gt(V[N - 2]).post();
17
18 // 2. Solving part
19 Solver solver = model.getSolver();
20 // 2.a define a search strategy
21 solver.setSearch(Search.minDomLBSearch(S));
22 if(solver.solve()){
23     System.out.printf("All interval series of size %d\n", N);
24     for (int i = 0; i < N - 1; i++) {
25         System.out.printf("%d < %d> ",
26             S[i].getValue(),
27             V[i].getValue());
28     }
29     System.out.printf("%d", S[N - 1].getValue());
30 }

```

Lignes 3 à 8 on déclare N, puis le modèle correspondant au CSP et on lui ajoute les variables S et V (des tableaux int particuliers pour les notes).

Lignes 10 à 12 on ajoute la contrainte sur v permettant de vérifier que chaque série est dodécaphonique.

Puis lignes 13 à 16 on force toutes les séquences à être uniques.

Enfin lignes 22 on fait résoudre les solutions, puis on affiche les séquences à la suite lignes 24 à 29.

Nous avons voulu lancer ce code, nous avons donc installé chocosolver mais nous n'avons pas réussi à régler les bugs de compilation, le programme n'a donc pas fonctionné chez nous.

```

Exception in thread "main" java.lang.Error: Unresolved compilation problems:
    The import gnu cannot be resolved
    The import org.chocosolver.memory cannot be resolved
    The import org.chocosolver.memory cannot be resolved
    The hierarchy of the type Model is inconsistent
    The type Model must implement the inherited abstract method IIntConstraintFactory.cumulative(Task[])
    The type Model must implement the inherited abstract method IVariableFactory.taskVarArray(IntVar[]),
    The type Model must implement the inherited abstract method IViewFactory.intMinusView(IntVar)
    The type Model must implement the inherited abstract method IIntConstraintFactory.cumulative(IntVar)
    The type Model must implement the inherited abstract method ISetConstraintFactory.union(IntVar[], S)
    The type Model must implement the inherited abstract method IViewFactory.realIntView(IntVar, double)
    The type Model must implement the inherited abstract method ISetConstraintFactory.union(SetVar[], S)
    The type Model must implement the inherited abstract method ISatFactory.addClausesBoolIsLeVar(BoolV,

```

Domaines d'application et limites

Les domaines d'applications sont très vastes, l'optimisation combinatoire touche aux problèmes combinatoires, aux mathématiques discrètes ainsi qu'à la théorie des graphes. Leurs applications concernent toutes sortes de problèmes informatiques, d'optimisation de production à grande échelle, de gestion d'opérations militaires, de problèmes économiques, même de composition de bouquets !

Les limites sont plus faciles à définir, il y a en premier lieu la modélisation du problème qui peut vite devenir très compliquée, et ensuite l'efficacité des CSP qui est aussi limitée par la puissance de calcul.

<https://www.emse.fr/~delorme/Papiers/MemoireDEA/memoire003.html>

<http://prolland.free.fr/Cours/Cycle1/MIAS/MIAS2/Optimisation/Optimisation.html#complexite>

http://www.unit.eu/cours/EnsROtice/module_me/co/Mon_Module_2.html

https://fr.wikipedia.org/wiki/Optimisation_combinatoire

<http://www.g-scop.grenoble-inp.fr/fr/recherche/qu-est-ce-que-l-optimisation-combinatoire>

<http://binaire.blog.lemonde.fr/2015/11/20/la-programmation-par-contraintes-expliquee-a-ma-garagiste-ou-a-mon-fleuriste/>

<http://www.i3s.unice.fr/~malapert/thesis/split/chapitre2.pdf>

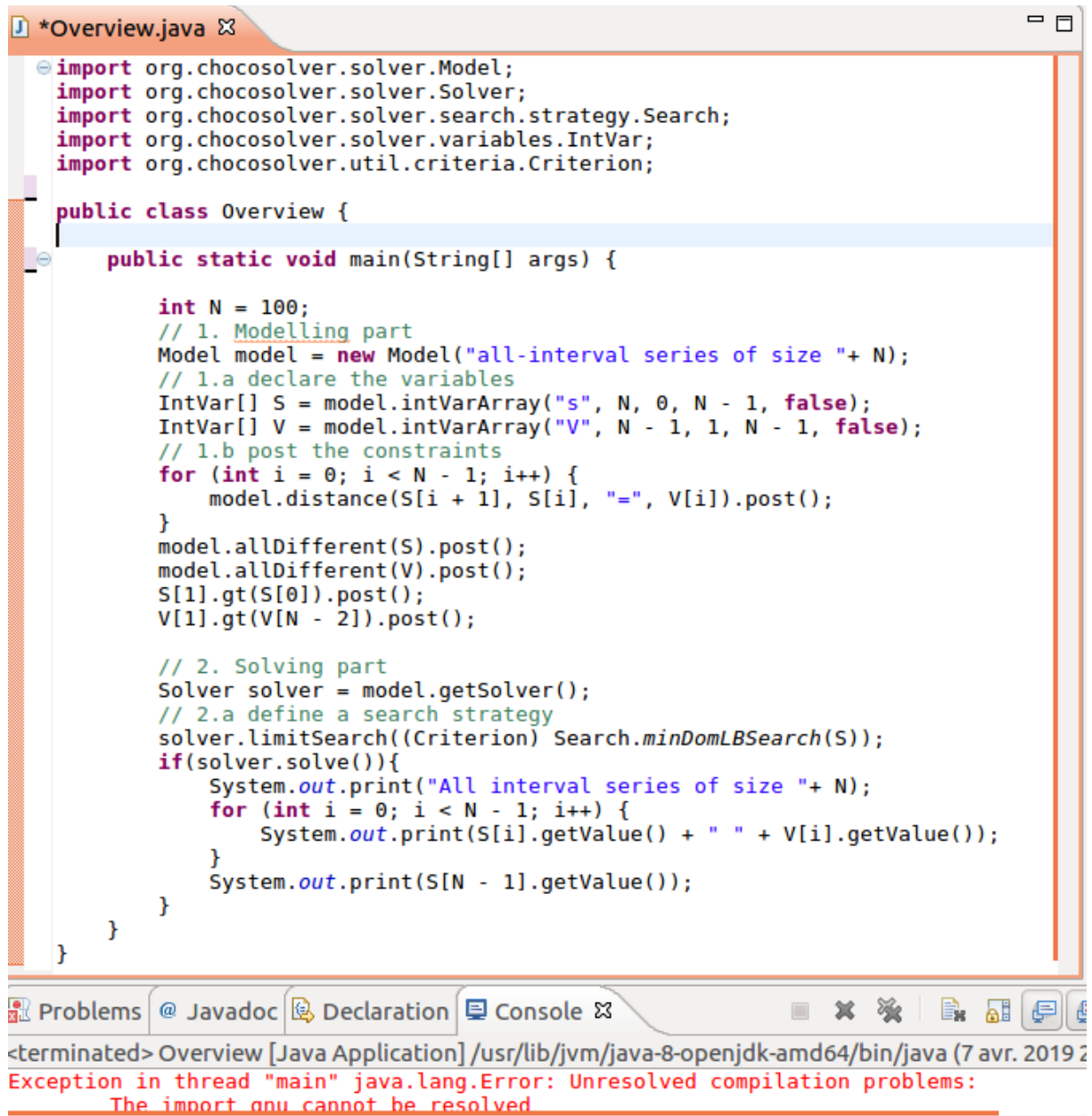
https://www.lix.polytechnique.fr/~liberti/teaching/isic/isc612-07/sadykov-CP_slides.pdf

<https://perso.liris.cnrs.fr/christine.solnon/Site-PPC/session1/e-miage-ppc-sess1.htm>

https://fr.wikipedia.org/wiki/Programmation_par_contraintes

Annexe

Version du code modifiée lors de l'implémentation pour résoudre les erreurs de compilation (enfin pour essayer en tout cas).



```
*Overview.java
import org.chocosolver.solver.Model;
import org.chocosolver.solver.Solver;
import org.chocosolver.solver.search.strategy.Search;
import org.chocosolver.solver.variables.IntVar;
import org.chocosolver.util.criteria.Criterion;

public class Overview {

    public static void main(String[] args) {

        int N = 100;
        // 1. Modelling part
        Model model = new Model("all-interval series of size "+ N);
        // 1.a declare the variables
        IntVar[] S = model.intVarArray("s", N, 0, N - 1, false);
        IntVar[] V = model.intVarArray("V", N - 1, 1, N - 1, false);
        // 1.b post the constraints
        for (int i = 0; i < N - 1; i++) {
            model.distance(S[i + 1], S[i], "=", V[i]).post();
        }
        model.allDifferent(S).post();
        model.allDifferent(V).post();
        S[1].gt(S[0]).post();
        V[1].gt(V[N - 2]).post();

        // 2. Solving part
        Solver solver = model.getSolver();
        // 2.a define a search strategy
        solver.limitSearch((Criterion) Search.minDomLBSearch(S));
        if(solver.solve()){
            System.out.print("All interval series of size "+ N);
            for (int i = 0; i < N - 1; i++) {
                System.out.print(S[i].getValue() + " " + V[i].getValue());
            }
            System.out.print(S[N - 1].getValue());
        }
    }
}
```

Problems @ Javadoc Declaration Console

<terminated> Overview [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (7 avr. 2019 2
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
The import org cannot be resolved