

# Heuristiques de branchement pour les problèmes continus de satisfaction de contraintes

Projet CONFIANTE  
CONstraint programming using Floating-point Interval Arithmetic :  
New heuristics, Tests, Experiments

Proposition de stage de M2 informatique 2026-2027

## Cadre

- encadrantes : Christine Solnon et Nathalie Revol, en collaboration avec Christophe Jermann
- lieu : au CITI ou au LIP
- dates : de février à juin ou juillet 2027
- contact : Christine.Solnon@insa-lyon.fr et Nathalie.Revol@ens-lyon.fr

Christine Solnon, Nathalie Revol et Christophe Jermann, qui est également associé à ce stage, collaborent sur ce sujet dans le cadre du projet CONFIANTE, qui finance le stage.

## Motivations

Les problèmes de satisfaction de contraintes, ou CSP (Constraint Satisfaction Problems) par la suite, constituent une famille de problèmes importante en informatique fondamentale avec des applications variées, notamment en aide à la décision. Ces problèmes sont définis par un ensemble de variables (les inconnues), chacune de ces variables ayant un domaine (l'ensemble des valeurs qu'elle peut prendre) et un ensemble de contraintes que ces variables doivent satisfaire. Nous distinguerons les *CSP discrets*, pour lesquels les domaines sont discrets (booléens, entiers par exemple), et les *CSP continus*, pour lesquels les domaines sont continus, et nous nous focaliserons sur les domaines qui sont des intervalles de l'ensemble des réels.

Très schématiquement, pour les CSP discrets comme continus, les méthodes de résolution les plus utilisées sont de type « branch-and-prune », aussi appelées « branch-and-propagate ». L'idée est de résoudre le problème en décomposant le problème initial en sous-problèmes (c'est la partie "branch", généralement réalisée en choisissant une variable dont le domaine contient plusieurs valeurs, et en partitionnant ce domaine en sous-domaines), et de propager les contraintes à chaque sous-problème afin de réduire les domaines (c'est la partie "prune" ou "propagate"). Les CSP continus se distinguent peut-être par une grande attention portée à l'étape d'élagage/propagation.

A contrario, dans ce projet nous nous intéresserons à l'étape de branchement, visant à choisir une variable et à partitionner son domaine, l'angle choisi étant de transposer, d'adapter et d'évaluer les heuristiques connues pour être efficaces pour les CSP discrets, au cas des CSP continus.

## Travail attendu

Une première étape du projet consistera à mettre en commun la connaissance des heuristiques existantes pour les CSP discrets et à transposer ces heuristiques pour les CSP continus. On peut penser à des heuristiques sur la façon de choisir les variables en s’inspirant de celles utilisées pour résoudre les CSP discrets. On pourra notamment utiliser une forme d’apprentissage par renforcement. Pour les CSP discrets, l’idée est d’associer des poids aux contraintes et d’augmenter le poids d’une contrainte quand sa propagation déclenche un échec. Au moment de choisir une variable, on privilégie celles qui interviennent dans les contraintes ayant les poids les plus importants. De nombreuses variantes ont été développées. Côté continu, les travaux de Christophe Jermann et ses co-auteurs indiquent que c’est une piste intéressante et à creuser.

Pour la résolution de CSP discrets comme pour celle de CSP continus, l’ordre choisi sur les contraintes a également de l’importance. Cependant, leur utilisation heuristique pour les CSP continus n’a pas été aussi étudiée et testée que pour les CSP discrets.

On pourra aussi utiliser des approches permettant d’accélérer la recherche de solutions. On peut penser tout simplement au redémarrage aléatoire (« random restart ») : il est établi, pour les CSP discrets, qu’avec une heuristique comportant un peu d’aléatoire, si on lance plusieurs exécutions avec des graines différentes, les temps d’exécution sont très différents. Souvent, il y a beaucoup d’exécutions avec des temps petits et seulement quelques-unes avec des temps vraiment longs, il s’agit du phénomène de « heavy tail ». Dans ce cas, une stratégie simple pour améliorer les performances consiste à arrêter l’exécution au bout d’un certain temps et à relancer l’exécution avec une nouvelle graine. Cette stratégie n’est, à notre connaissance, pas employée pour les CSP continus ; cependant des stratégies de diversification contenant une partie d’aléatoire ont été proposées par C. Jermann et ses co-auteurs. Nous souhaitons déterminer dans un premier temps à quel point ce phénomène de « heavy tail » se produit pour les CSP continus également et tester de façon plus approfondie cette stratégie.

Une dernière étape de ce projet consiste à expérimenter sur un nombre significatif de problèmes tests ces différentes approches, afin d’évaluer leur efficacité respective. Il nous faudra choisir un, ou un petit nombre, de logiciels pour implanter ces heuristiques ; nous pensons naturellement à IBEX, développé à Nantes, et nous profiterons de l’ensemble des jeux de tests qu’il comporte.

## Compétences

- en informatique et en particulier programmer en C++
- aimer les mathématiques

# Branching heuristics for continuous constraint programming

Project CONFIANTE  
CONstraint programming using Floating-point Interval Arithmetic :  
New heuristics, Tests, Experiments

Internship proposal, computer science master M2, 2026-2027

## Context

- supervision : Christine Solnon and Nathalie Revol, in collaboration with Christophe Jer-
- mann
- where : at CITI or at LIP
- when : from February to June or July 2027
- contact : Christine.Solnon@insa-lyon.fr and Nathalie.Revol@ens-lyon.fr

Together with Christophe Jermann, Christine Solnon and Nathalie Revol collaborate within the CONFIANTE project, which funds this internship.

## Motivations

Constraint satisfaction problems, denoted by CSP in what follows, are an important set of problems in computer science and they correspond to diverse applications, notably in decision making. These problems are defined by a set of variables (whose values are to be determined), each of these variables belonging to a predetermined domain (the set of admissible values), and a set of constraints that must be satisfied by the variables. When variable domains are finite sets, the problem is called a *discrete CSP*; when they are intervals of the set of real numbers, it is called a *continuous CSP*.

The principle of algorithms solving a CSP, be it discrete or continuous, is *Branch-and-Propagate* or *Branch-and-Prune* : Branch involves selecting a variable and partitioning its domain to create subproblems that are recursively solved; Propagate/Prune involves exploiting constraints to reduce domains by removing inconsistent values. Whenever constraint propagation wipes out a domain, the current subproblem has no solution so that it is pruned and we backtrack to the next subproblem. Continuous CSP solving often focuses on the Propagate/Prune step.

A contrario, in the CONFIANTE project we will focus on the Branch part, that is to the choice of the variable whose domain is partitioned. The idea is to transpose, adapt for continuous CSP, and assess heuristics that are efficient to solve discrete CSP.

## Description of the work

A first step is to collect state-of-the-art heuristics that are widely used for discrete CSP and to determine if and how they can be adapted for continuous CSP. A first direction of research is to study how the principles that lead to the choice of the branching variable in discrete CSP solving apply to the continuous case. Reinforcement learning can be employed : for discrete CSP, constraints are given weights and the weight of a constraint increases when the constraint is involved in a failure (and thus the corresponding domain is wiped out/no more explored). The variable

that appears in the constraints with the largest weight is chosen for branching. Numerous variants of this approach exist. For continuous CSP, preliminary work by C. Jerermann and his co-authors is promising and we will explore this approach further.

Another direction is to explore how the order in which constraints are used impacts performances. For continuous CSP, this order has not been explored as thoroughly as for discrete CSP and this direction will be deepened.

Another principle widely adopted to solve discrete CSPs is the random restart, so that the search is not stuck in a long and inefficient branch. Indeed, for discrete CSP, it has been established that launching several executions of the solving process that contains some randomization yields executions times that can vary greatly : frequently, most executions take a small time and some executions need much longer. This phenomenon is known as "heavy-tail" and a simple remedy is to stop the execution after a while and to restart it with a different seed. We are not aware of many similar approaches for continuous CSPs ; however, partly-random diversification search strategies for continuous CSPs were proposed by C. Jerermann and his co-authors. We would like to observe in more details how the "heavy-tail" phenomenon shows up for continuous CSP.

Last but not least, the proposed heuristics must be assessed and experiments must be led on a significative set of benchmark problems. The software testbed for these experiments is very likely to be IBEX, developed in Nantes, and its set of benchmark tests.

## **Skills**

- skills in computer science and in particular C++ programming
- a taste for mathematics