B&P algorithms for continuous constraint problems: a survey of branching strategies

Christophe Jermann – Nathalie Revol – Christine Solnon
LS2N, U. Nantes, École Centrale Nantes – INRIA, LIP, ENS de Lyon – INSA, CITI, Lyon
France

SCAN 2025, Oldenburg, Germany, 22-26/09/2025

CSPs: definition

Constraint Satisfaction Problems (CSPs)
defined by a triple (X, D, C) such that

- $X = \{x_1, \ldots, x_n\}$: set of n variables,
- D_i: domain of each variable $x_i \in X$,
  When the D_i are finite sets, the problem is a **discrete CSP**.
  When the D_i are intervals in $\mathbb{R}$, it is a **continuous CSP**.

- $C = \{C_1, \ldots, C_m\}$: set of constraints such that each C_j is a relation defined by $R_j \subseteq D_1 \times D_2 \times \ldots \times D_n$ for a discrete CSP;
- $C = \{C_1, \ldots, C_m\}$: set of constraints such that each C_j is defined by a function f_j over a subset of X for a continuous CSP.

CSPs: solution

Solution: a set of values $\{v_1, \ldots, v_n\}$ such that

- $\forall x_i \in X$, $v_i \in D_i$,
- each constraint in C is satisfied, i.e. $(v_1 \ldots v_n) \in R_j$ or $f_j(v_1, \ldots, v_n) = 0$.

For continuous CSPs, a solution is a set of $\varepsilon-$boxes,
(that is, boxes of – relative or absolute – width $\leq \varepsilon$)
their union containing all solutions.

CSPs: B&P algorithm

$WL := \{D^{\mathtt{init}}\};\ L := \{\}$

\# WL: working list; L: list of solutions

while (WL is not empty) do

    remove D from WL

    propagate the constraints on D to get D_r

    if (D_r is a solution) then

        \# $\forall j,\ R_j(D_r)$ holds (discrete CSP)

        \# $\forall j,\ f_j(D_r) = \{0\}$ or $(f_j(D_r) \ni 0 \wedge D_r$ is small enough)

        insert D_r in L

    else if $D_r \neq \emptyset$ then

        bisect D_r into D^1 and D^2

        insert D^1 and D^2 in WL

    end if

end while

return L

CSPs: B&P algorithm

$WL := \{D^{\texttt{init}}\}$; $L := \{\}$
`while` (WL `is not empty`) `do`
    `remove` D `from` WL
    propagate the constraints on D to get D_r
    `if` (D_r `is a solution`) `then`
        `insert` D_r `in` L
    `else if` $D_r \neq \emptyset$ `then`
        `bisect` D_r `into` D^1 `and` D^2
        `insert` D^1 `and` D^2 `in` WL
    `end if`
`end while`
`return` L

The Propagate or Prune part for continuous CSPs

- forward and backward propagation (Schichl & Neumaier (2005), Jaulin et al. . . .)
- consistency (Lhomme – Benhamou – Rueher – . . . (1993 ff.))
- Newton method (Hansen et al. (1978 ff.))
- affine arithmetic, Taylor expansions, Taylor models (Comba, Stolfi, Figueiredo (1993 ff.), Berz & Makino (1998 ff.)

The Propagate or Prune part for continuous CSPs

- forward and backward propagation (Schichl & Neumaier (2005), Jaulin et al. . . .)
- consistency (Lhomme – Benhamou – Rueher – . . . (1993 ff.))
- Newton method (Hansen et al. (1978 ff.))
- affine arithmetic, Taylor expansions, Taylor models (Comba, Stolfi, Figueiredo (1993 ff.), Berz & Makino (1998 ff.)
- your favorite method, many exist.

CSPs: B&P algorithm

$WL := \{D^{\mathtt{init}}\};\ L := \{\}$
```
while (WL is not empty) do
    remove D from WL
    propagate the constraints on D to get D_r
    if (D_r is a solution) then
        insert D_r in L
    else if D_r ≠ ∅ then
       bisect D_r into D^1 and D^2
       insert D^1 and D^2 in WL
    end if
end while
return L
```

NOT a focus of this talk: constraint ordering

$WL := \{D^{\mathtt{init}}\};\ L := \{\}$
while (WL is not empty) do
    remove D from WL
    propagate the constraints on D to get D_r
    if (D_r is a solution) then
        insert D in L
    else if $D_r \neq \emptyset$ then
        bisect D into D^1 and D^2
        insert D^1 and D^2 in WL
    end if
end while
return L

Focus of this talk: branching strategies in B&P algorithm

- **Christine Solnon** is an expert in discrete CSPs solving;
- **Christophe Jermann** is an expert in continuous CSPs solving.

Question: can ideas from discrete CSPs solving be adapted for continuous CSPs?

Project CONFIANTE: CONstraint programming using Floating-point Interval Arithmetic : New heuristics, Tests, Experiments (2025-2027)
supported by the Fédération Informatique de Lyon.

Agenda

Classical approaches

For discrete CSPs: depth-first search (DFS), breadth-first search (BFS) (*actually, no BFS in real implementations*), best-first search (if available).

For continuous CSPs:
to ensure convergence: oldest box first&oldest side first, or largest box first and bisect largest side: similar to BFS (Neumaier (survey 2003)).

BFS compared to DFS:

- ☺ diversity, convergence
- ☹ memory usage

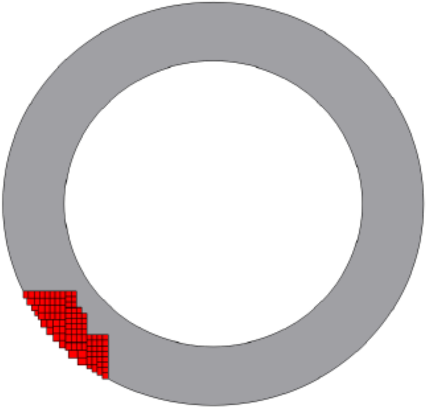
Classical approaches

For discrete CSPs: depth-first search (DFS), breadth-first search (BFS) (*actually, no BFS in real implementations*), best-first search (if available).

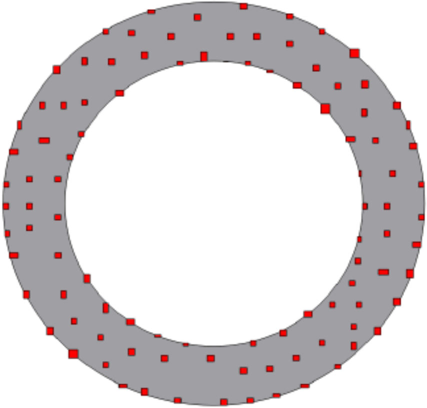
For continuous CSPs:
to ensure convergence: oldest box first&oldest side first, or largest box first and bisect largest side: similar to BFS (Neumaier (survey 2003)).

BFS compared to DFS:

- ☺ diversity, convergence
- ☹ memory usage
- ☺ anytime



(a) DFS



(b) MDFS

(From Richard de la Tour (PhD 2023).)

Classical approach: trade-off

Easy fix: explore one element from BFS every α node, and the rest is DFS.

Interleaved DFS:

- **discrete CSPs:** mix DFS and BFS, depending whether DFS is helpful or not (Meseguer (1997))
- **continuous CSPs:** mix DFS and BFS, depending whether DFS is helpful or not (Chenouard, Goldsztejn and Jermann (2009) – La Tour, Chenouard and Granvilliers (2024))

Corresponding algorithm

```
while (WL is not empty) do
   remove D from WL
   propagate the constraints on D to get D_r
   if (D_r is a solution) then
       insert D_r in L
       sort WL according to criterion ρ
   else if D_r ≠ ∅ then
      bisect D_r into D^1 and D^2
      insert D^1 and D^2 in front of WL
   end if
end while
return L
```

Criterion ρ leads to maximize the distance between the boxes in L and the unexplored part of the search space.

Another view of this algorithm

```
WL := {D^init}; L := {}
while (WL is not empty) do
    remove D from WL
    explore D and its neighbour nodes/boxes
    until one solution is found (insert it in L)
    reorder WL in order to explore another region
end while
return L
```

Agenda

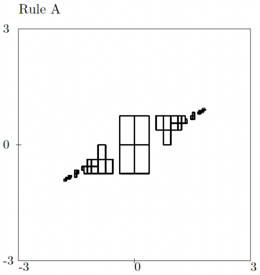
CSPs: B&P algorithm

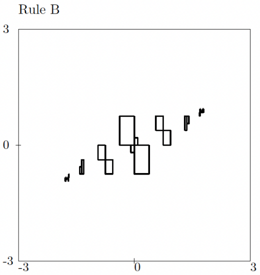
Principle: fail-first.

- **Discrete CSPs:** fail-first: min domain.
- **Continuous CSPs:** smear:

$$\mathrm{smear}(x_i, f_j) = \left|\frac{\partial \boldsymbol{f}_j}{\partial x_i}(\boldsymbol{x})\right| \cdot w(\boldsymbol{x}_i)$$

Choose i that maximises $\max_j(\mathrm{smear}(x_i, f_j))$ or
$\sum_j \mathrm{smear}(x_i, f_j)$
or some variant.
(Kearfott – Csendes, Kreinovich – Araya, Neveu, Trombettoni, Chabert et al. (1987 ff.))





after 250 iteration steps
for the Three-Hump-Camel-Back,
either largest width (Rule A)
or largest smear (Rule B)
(From Csendes (2008).)

Agenda

Classical approaches

Variable selection heuristics

Randomized algorithms
- Principle
- No-good recording

Adaptive strategies

Optimization

Randomized algorithms for discrete CSPs

Heavy-tail phenomenon for discrete CSPs:
large number of runs with very short execution time,
and large number of runs with very long execution time.
(Gomes, Selman, Kautz et al. (1998 ff.))

- **Randomized choice for the next node/variable to explore:**
  choose the best up to $\alpha\%$ (e.g. $\alpha = 20\%$ or $\alpha = 30\%$)
- **Restarts:** after c explored nodes, restart
  Increase progressively – or learn – the value of the cutoff c.

(early 1990s, Walsh – many authors)

Randomized algorithms for continuous CSPs

Heavy-tail phenomenon?
not established for continuous CSPs.

- **Randomized choice:**
  e.g. (Chenouard, Goldsztejn and Jermann (2009))
- **Restarts:** after c explored nodes, restart
  not adopted for continuous CSPs?

Randomized algorithms for discrete CSPs no-good recording

Goal:

avoid re-visiting already explored parts of the search space
and simultaneously avoid storing an exponential amount of data.

Principle:

store the clauses and variables' assignments explored so far.

Implementation:

Reduce the size of no-goods (=reduce memory).
Exploit, propagate the no-goods.
Too many strategies and authors to be reported here!

Randomized algorithms for continuous CSPs no-good recording

Principle:

create a (partial) order among variables to express, for instance:
"if $x_1 = v_1$ and $x_2 = v_2$ then y cannot be assigned to v",
as "$x_1 < y$ and $x_2 < y$".
This (partial) order among variables is used to guide the choice of the next explored variable. (Bliek (1998))

Implementation: decompose the set of constraints into separate DAGS (i.e. smaller subproblems)
then determine such a partial order among variables.
(Bliek, Neveu and Trombettoni (1998), Jermann (2002))

Not so frequent for continuous CSPs.

Randomized algorithms for discrete CSPs no-good recording

Many ideas to borrow for continuous CSPs?

- always explore first $x_i = v_i$ then $x_i \neq v_i$:
  transpose into "always explore first $x_i \leq v_i$ then $x_i > v_i$"?
- summary of no-good: store only the negative part (= rightmost path)
- store a small number of no-goods
- be able to use the no-goods
- prove some properties of such no-goods to reduce the storage and be able to recover useful past decisions after a restart
- none of the above?

Agenda

Adaptive strategies for discrete CSPs

Principle: attach a weight to each constraint
increase this weight when the corresponding constraint leads to a wipe-out of some domain
(Boussemart, Hemery, Lecoutre, Sais et al. (2004 ff.))

Use to branch: for each variable, sum the weights of the constraints in which it appears
choose the variable with the largest sum.

Decaying strategy: decrease the weights with time
as constraints are no more used
(e.g. Li, Yin, Li (2021))

Adaptive strategies for CSPs
(machine) learning

Difficulty: many strategies, many parameters to be tuned
integrate strategies
(Kotthoff (2013))

Learn a strategy: from a set of strategies
learn from the beginning of the search, deduce which strategy to employ to pursue

- multi-armed bandit (e.g. Wattez et al.(2019))
- overview (e.g. Popescu et al.(2021))
- using Bayesian optimization (e.g. Haddad et al.(2024))
- for MINLP (e.g. Tang et al.(2025))
- for global optimization (e.g. Bertsimas et al.(2025))
- ...

Adaptive strategies for continuous CSPs

Reinforcement learning in Goualard - Jermann (2008)
but for the Prune part.

Many directions to explore:

- reinforcement learning for the Branch?
- large and expanding literature on this topic for discrete CSPs: a source of inspiration?

Agenda

Optimization problems

More explored topic for continuous CSPS:
many strategies!

- best-first strategies
- fail-first strategies: smear...
- ...

Optimization problems

More explored topic for continuous CSPS:
many strategies!

- best-first strategies
- fail-first strategies: smear...
- ...

for the **Prune** part?

Adaptive strategies can probably be adopted:
weights, reinforcement, learning...

Optimization problems

More explored topic for continuous CSPS:
many strategies!

- best-first strategies
- fail-first strategies: smear...
- ...

for the **Prune** part?

Adaptive strategies can probably be adopted:
weights, reinforcement, learning...
see you at next SCAN with some experiments!

Optimization problems

More explored topic for continuous CSPS:
many strategies!

- best-first strategies
- fail-first strategies: smear...
- ...

for the **Prune** part?

Adaptive strategies can probably be adopted:
weights, reinforcement, learning...
see you at next SCAN with some experiments!

WANTED: a master student to help us with this project in 2026!