Computing Inner Approximations of Numerical MaxCSP

Marc Christie, Jean-Marie Normand, Charlotte Truchet

Laboratoire d'Informatique de Nantes Atlantique
FRE CNRS 2729, Universit de Nantes
2 Rue de la Houssinire, 44322 Nantes Cedex 3
christie,normand,truchet@lina.univ-nantes.fr

Abstract. Classically, inner-approximation techniques guarantee soundness over a conjunction of numerical non-linear constraints. These techniques do not offer obvious extensions to manage such problems as solving numerical MaxCSPs. In this paper, we present an algorithm to compute inner approximations of numerical MaxCSPs. Our approach guarantees a complete and sound exploration of the search space and identifies the subsets that maximize the set of satisfied constraints in a Numerical CSP. The algorithm computes an inner approximation of these subsets given a minimum splitting size. Our approach relies on classical interval-based propagation techniques (AC-3) as well as a n-dimensional inner-extending process. A hybridization with interval adapted local search methods is proposed and tested. Results on toy problems show the gain w.r.t. a naive algorithm, but do not highlight the evidence of contribution from interval-based local search methods.

1 Introduction

Complete solving techniques on Numerical Constraint Satisfaction Problems (CSP defined over real values) are commonly used to tackle well-known hard problems in many domains (conceptual design, molecular conformation, mechanics, robotics, computer graphics, *etc.*). These methods output a set of boxes representing an outer approximation of the real set of solutions, ensuring that no solution is lost during the solving process (completeness property). In the meantime many applications require the computation of reliable solutions, *i.e.,* such that each point belonging to a box is an exact solution of the constraint network. This can be achieved by *inner approximations* of NCSPs techniques [26, 17, 4] or *i-consistency* [11]. However, neither the complete (outer-approximation) nor the correct (inner approximation) approaches are suitable for solving applications where one wants to satisfy as many constraints as possible in a globally inconsistent problem (the so-called Max-NCSP problems). Many applications require to provide information on the subsets of consistent constraints, for incremental model building or debugging purposes. This work is motivated by the fact that it is usually difficult to predict the nature (*i.e.,* over or under constrained) of a

problem beforehand and thus to know which techniques should be used to tackle it.

In this paper, we propose the first results of an inner-extending algorithm for over or under-constrained problems. This algorithm determines a set of interval boxes that satisfies as many constraints as possible for a CSP. The hybrid solving technique consists in weaving interval-based constraint propagation steps with inner-extending procedures based on the choice of a representative point in the current box. The selection of this point will be done by two different techniques, the first one simply returning the middle point of the current box (midPoint) while the second one uses our interval adapted local search methods (LS) to help determining the best possible point in the box. To the authors's knowledge the literature does not provide any other Max-NCSP approaches to compute inner approximations.

The paper is organized as follows. Section 2 presents the classical interval-based constraints resolution techniques as well as a new simple n-ary inner-extending operator (alike classical n-ary contraction operators). Section 3 introduces the definition of Max-NCSP problems and proposes a resolution method based on the use of both contraction and extension operators. We then present in section 4 our theoretical framework for a continuous extension of the classical local search methods before giving some firsts results.

2 Interval Constraint Solving

2.1 Interval Analysis

In this section, we limit the presentation of interval analysis to the concepts needed in the sequel of the paper. A thorough presentation can be found in [23, 1, 14, 24]. One solution advocated by Moore [23] to control the rounding errors that are encountered in floating point arithmetic computations is to use *interval arithmetic*, that is to replace reals ($\in \mathbb{R}$) by intervals containing them, whose bounds are representable numbers. An interval I with representable bounds is called a *floating-point interval*. It is of the form: $I = [a, b] = \{r \in \mathbb{R} \mid a \leq r \leq b$, with $a, b \in \mathbb{F}\}$. A non-empty interval $I = [a, b]$ that contains no more than two floats is said *canonical*. In the same way, a Cartesian product of intervals (or *box*) is said canonical whenever it is canonical in all its dimensions. Let $\mathcal{I}$ be the set of all floating-point intervals. In the sequel, vectors or Cartesian products are written in bold face. Since we will only deal with this kind of intervals in the rest of the paper, we will refer to them as simply *intervals*.

Operations and functions over reals are also replaced by an *interval extension* having the *containment property* (see [23]). Given a real function, $f\colon \mathbb{R}^n \to \mathbb{R}$ and a box $\boldsymbol{B} \in \mathcal{I}^n$, let $f(\boldsymbol{B}) = \{f(\boldsymbol{r}) \mid \boldsymbol{r} \in \boldsymbol{B}\}$, $\mathcal{D}_f$ be the domain of f, and $\mathcal{D}_f^{\mathcal{I}} = \{\boldsymbol{B} \in \mathcal{I}^n \mid \boldsymbol{B} \subseteq \mathcal{D}_f\}$.

Definition 1 (Interval extension [24]). *Given a real function* $f\colon \mathbb{R}^n \to \mathbb{R}$*, an* interval extension $F\colon \mathcal{I}^n \to \mathcal{I}$ *of* f *is an interval function verifying:*

$$f(\boldsymbol{r}) = F(\boldsymbol{r}) \qquad \forall \boldsymbol{r} \in \mathcal{D}_f$$
$$\mathsf{Outer}(f(\boldsymbol{B})) \subseteq F(\boldsymbol{B}) \qquad \forall \boldsymbol{B} \in \mathcal{D}_f^{\mathcal{I}}$$

where $\mathsf{Outer}(\rho) = \bigcap\{\boldsymbol{B} \in \mathcal{I}^n \mid \rho \subseteq \boldsymbol{B}\}$ *for any real relation* $\rho \in \mathbb{R}^n$*.*

The extensions of some basic operators are defined as follows:

$$\begin{aligned}
[a,b] \oplus [c,d] &= [a+c, b+d] \\
[a,b] \ominus [c,d] &= [a-d, b-c] \\
[a,b] \otimes [c,d] &= [\min(ac,ad,bc,bd), \max(ac,ad,bc,bd)] \\
\exp([a,b]) &= \quad [\exp(a), \exp(b)]
\end{aligned}$$

Outer Contracting Operators Discarding all inconsistent values from a box of variables' domains is intractable when the constraints are real ones. Consequently, methods such as *hull consistency* [5] and *box consistency* [7] rely on outer contracting operators that discard values according to a given consistency. These methods are instances of the local consistency framework in artificial intelligence [19]. Such operators enjoy the *contractance*, *completeness*, and *monotonicity* properties.

Definition 2 (Outer contracting operator). *Let* c *be an* n*-ary constraint and* ω *a given consistency, an* outer contracting operator *for* c *is a function* $\mathsf{OCO}_c^{\omega}\colon \mathcal{I}^n \to \mathcal{I}^n$ *that discards values according to a consistency* ω*.*

2.2 Computing Interval Inner-approximations

In this section, we will present operators that compute an *inner approximation*, that is, a subset of the relation ρ corresponding to the constraint system. As a consequence, the boxes computed by these operators enjoy the *soundness* property that is any point inside is a solution of the problem [6]. We consider the following definition for the inner-approximation of a relation:

Definition 3 (Inner approximation). *Given an* n*-ary real relation* $\rho \subseteq \mathbb{R}^n$*,*

$$\mathsf{Inner}(\rho) = \{\boldsymbol{r} \in \mathbb{R}^n \mid \mathsf{Outer}(\{\boldsymbol{r}\}) \subseteq \rho\}$$

Definition 4 (Inner approximation operator). *Given an* n*-ary real relation* $\rho \subseteq \mathbb{R}^n$*, the inner approximation operator* $\mathsf{Inner}\colon \mathbb{R}^n \to \mathbb{R}^n$ *is defined by:*

$$\mathsf{Inner}(\rho) = \{\boldsymbol{r} \in \mathbb{R}^n \mid \mathsf{Outer}(\{\boldsymbol{r}\}) \subseteq \rho\}$$

Such an approximation contains all the elements whose smallest enclosing box is included in the relation.

Inner Extending Operator Collavizza *et al.* [11] propose a similar approach to expand consistent domains of a solution of a NCSP yet in one dimension by relying on univariate extrema functions. We extend this approach to a $n-$dimensional extension through an *inner-extending* operator that inputs a consistent box $\boldsymbol{B} \subseteq \rho_c$ as well as a constraint c and that outputs a box $\boldsymbol{B'}$ such that $\boldsymbol{B'} \supseteq \boldsymbol{B}$ in which all values still satisfy the constraint c.

Definition 5 (Inner-extending operator). *Given a n-ary constraint c and its underlying relation ρ_c, an* Inner-extending operator *for c is a function $\mathsf{IEO}_c \colon \mathcal{I}^n \rightarrow \mathcal{I}^n$ verifying for all boxes $\boldsymbol{B} \subseteq \rho_c$:*

$$\rho_c \supseteq \mathsf{IEO}_c(\boldsymbol{B}) \supseteq \boldsymbol{B}$$

We now define an Optimal Inner-extending operator, *i.e.,* such that it is impossible to provide a larger box (w.r.t. inclusion) such that each point satisfies the constraint c.

Definition 6 (Optimal Inner-extending operator). *Given a n-ary constraint c, an* Optimal Inner-extending operator *for c is a function $\mathsf{OptimalIEO}_c \colon \mathcal{I}^n \rightarrow \mathcal{I}^n$ that verifies for all $\boldsymbol{B} \subseteq \rho_c$:*

$$\nexists \boldsymbol{B'} \in \mathcal{I}^n | \boldsymbol{B'} \supset \mathsf{OptimalIEO}_c(\boldsymbol{B}) \wedge \boldsymbol{B'} \supseteq \rho_c$$

As it reveals impossible to implement an optimal inner-extending operator (due to rounding of floating-point arithmetic operators and impossibility to implement operators enforcing arc consistency in a general case), we propose to implement an operator that will approximate the optimal operator. It's implementation can be found in algorithm 1. It is parameterized by a constraint c, an initial search box $\boldsymbol{B}$, a consistent box $\boldsymbol{P} \subseteq \rho_c$ and a precision ε. A trivial implementation would consist in iteratively extending $\boldsymbol{P}$ by ε in all its dimensions until $\boldsymbol{P}$ does not satisfies c anymore (or until $\boldsymbol{P}$ becomes bigger than $\boldsymbol{B}$) and then to return the next to last computed box. We propose a more efficient implementation based on a dichotomic extension of $\boldsymbol{P}$. A first iteration will extend $\boldsymbol{P}$ in all its dimensions by half the size of the domain between $\boldsymbol{P}$ and $\boldsymbol{B}$ until $\boldsymbol{P}$ is only partially consistent. The following step will then reduce $\boldsymbol{P}$ in a similar way until $\boldsymbol{P}$ certainly satisfies c. The process is then applied recursively until the desired precision is reached. In algorithm Alg. 1, $\boldsymbol{P}|_{v_i}$ denotes the domain of variable v_i in box $\boldsymbol{P}$, $\lfloor I$ stands for the left bound of interval I while $\lceil I$ represents its right bound.

This approach can be seen as a n-dimensional extension of the implementation of the box-consistency operator [28], however guarantying the soundness of the algorithm instead of its completeness.

Figure 1 shows the extension of a consistent box $\boldsymbol{P}$, given a constraint c in a search space limited by box $\boldsymbol{B}$.

Computing an Inner Extension for a set of constraints $\{c_1, \ldots, c_n\}$. Now that the inner-extension has been presented for a single constraint c, we

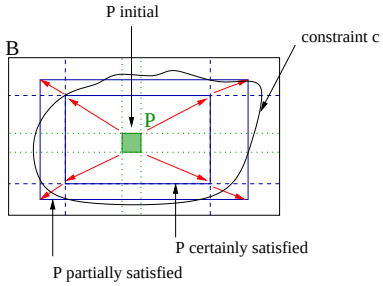

Fig. 1. 2D Extension of a box $\boldsymbol{P}$ w.r.t. a constraint c in a limited search space.

Alg. 1. Inner-extending algorithm given a constraint c (AlgoIEO).

AlgoIEO$_c$(**in**: $\boldsymbol{P} \in \mathcal{I}^n, \boldsymbol{B} \in \mathcal{I}^n, \varepsilon \in \mathbb{R}$): **out**: $\boldsymbol{K} \in \mathcal{I}^n$
begin
if (smallestDifference$(\boldsymbol{B}, \boldsymbol{P}) \leq \varepsilon$) **then**
  return $\boldsymbol{P}$
else
  % Extension loop of $\boldsymbol{P}$
  while certainlySatisfied$(\boldsymbol{P}, c)$
    $\boldsymbol{P'} \leftarrow \boldsymbol{P}$
    for each $v_i \in c$ **do**
      $$\boldsymbol{P}|_{v_i} \leftarrow \left[\frac{\lfloor \boldsymbol{B}|_{v_i} + \lfloor \boldsymbol{P}|_{v_i}}{2}, \frac{\lceil \boldsymbol{B}|_{v_i} + \lceil \boldsymbol{P}|_{v_i}}{2}\right]$$
    endforeach
  end
  % Reduction loop of $\boldsymbol{P}$
  while not certainlySatisfied$(\boldsymbol{P}, c)$
    $\boldsymbol{P''} \leftarrow \boldsymbol{P}$
    for each $v_i \in c$ **do**
      $$\boldsymbol{P}|_{v_i} \leftarrow \left[\frac{\lfloor \boldsymbol{P}|_{v_i} + \lfloor \boldsymbol{P'}|_{v_i}}{2}, \frac{\lceil \boldsymbol{P}|_{v_i} + \lceil \boldsymbol{P'}|_{v_i}}{2}\right]$$
    endforeach
  end
  % Recursion
  return AlgoIEO$_c(\boldsymbol{P}, \boldsymbol{P}$”$, \varepsilon)$
end
end

consider the computation of an inner-extending approximation for a set of constraints $\{c_1, \ldots, c_n\}$, in a box $\boldsymbol{B}$ given an initially consistent box $\boldsymbol{P}$. The principle follows the idea presented in [11]: for each constraint c_i we compute all the inner-extended boxes with operator IEO and then intersect them. The obtained algorithm IEOA is presented in table Alg. 2. It is parameterized by a set of constraints $C = \{c1, \ldots, c_n\}$, a box $\boldsymbol{B}$ and a consistent box $\boldsymbol{P}$.

Alg. 2. Inner-extending algorithm for a set C of n constraints (AlgoIEA).

```
1 AlgoIEA(in: C; P ∈ I^n, B ∈ I^n, ε ∈ ℝ)
            out: K ∈ I^n
2 begin
3     return ⋂_{c_i ∈ C}^n AlgoIEO_{c_i}(P, B, ε)
4 end
```

3 Tackling Max-NCSP problems

We propose a method to compute an inner-approximation of the subsets of the search space which maximizes the number of satisfied constraints of a numerical CSP. First let us introduce a few definitions.

Definition 7 (A Max-NCSP Problem). *A Max Numerical CSP is defined by $\mathcal{S} = \langle C, V, \boldsymbol{D} \rangle$ where C is a set of numerical constraints, V a set of variables and $\boldsymbol{D}$ a cartesian product of floating point bounded intervals.*

A solution $\boldsymbol{B} \subseteq \boldsymbol{D}$ of the Max-NCSP $\mathcal{S}$ is an assignment of values of V in $\boldsymbol{D}$ such that $\nexists \boldsymbol{B'} \subseteq \boldsymbol{D}$ *s.t.* $\mathsf{satis}(\boldsymbol{B'}) > \mathsf{satis}(\boldsymbol{B})$, where $\mathsf{satis}(\boldsymbol{B})$ provides the number of possibly satisfied constraints of C by box $\boldsymbol{B}$.

Definition 8 (Inner-approx. of a Max-NCSP). *The inner approximation of a Numerical Max-NCSP $\mathcal{S} = \langle C, V, \boldsymbol{D} \rangle$ represents all the distinct sub-regions $\boldsymbol{B} \subseteq \boldsymbol{D}$ such that $\nexists \boldsymbol{B'} \subseteq \boldsymbol{D}$* s.t. *$\mathsf{csatis}(\boldsymbol{B'}) > \mathsf{csatis}(\boldsymbol{B})$, where $\mathsf{csatis}(\boldsymbol{B})$ provides the number of certainly satisfied constraints of C by box $\boldsymbol{B}$.*

Therefore computing an inner approximation of a Max-NCSP consists in providing inner-approximations that maximize the number of consistent constraints. If we consider a Numerical Max-NCSP $\langle \mathcal{S}, \mathcal{V}, \boldsymbol{D} \rangle$, each area $\boldsymbol{B}$ of the search space (*i.e.,* $\boldsymbol{B} \subseteq \boldsymbol{D}$) can be characterized by the disjoint sets of constraints $\mathcal{C} \subseteq \mathcal{S}, \mathcal{P} \subseteq \mathcal{S}, \mathcal{N} \subseteq \mathcal{S}$ that respectively are $\mathcal{C}$ertainly satisfied, $\mathcal{P}$ossibly satisfied and $\mathcal{N}$ot satisfied by box $\boldsymbol{B}$. An illustration is provided in Figure 2.

In the following, we define three operators CS, PS and NS that compute the sub-set of constraints of a Numerical Max-NCSP $\mathcal{S}$ that respectively are certainly satisfied, possibly satisfied and not satisfied by a box $\boldsymbol{B}$.

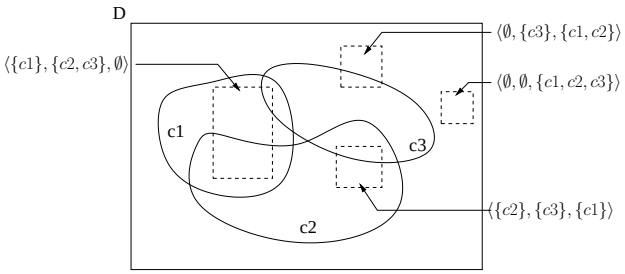

Fig. 2. $\mathcal{CPN}$-characterization of some sub-regions of a box $\boldsymbol{B}$ along constraints $c1, c2, c3$.

Definition 9 (Certainly satisfied operator). *Let $\boldsymbol{B}$ be a box and $\mathcal{S}$a set of constraints, the operator $\mathsf{CS}(\mathcal{S}, \boldsymbol{B})$ provides a set $\mathcal{S}'$ such that $\mathcal{S}' \subseteq \mathcal{S}$ and $\forall c \in \mathcal{S}', \boldsymbol{B} \subseteq \rho_c$*

Definition 10 (Possibly satisfied operator). *Let $\boldsymbol{B}$ be a box and $\mathcal{S}$a set of constraints, the operator $\mathsf{PS}(\mathcal{S}, \boldsymbol{B})$ provides a set $\mathcal{S}'$ such that $\mathcal{S}' \subseteq \mathcal{S}$ and $\forall c \in \mathcal{S}', \boldsymbol{B} \nsubseteq \rho_c \wedge \boldsymbol{B} \cap \rho_c \neq \emptyset$*

Definition 11 (Not satisfied operator). *Let $\boldsymbol{B}$ be a box and $\mathcal{S}$a set of constraints, the operator $\mathsf{NS}(\mathcal{S}, \boldsymbol{B})$ provides a set $\mathcal{S}'$ such that $\mathcal{S}' \subseteq \mathcal{S}$ and $\forall c \in \mathcal{S}', \boldsymbol{B} \cap \rho_c = \emptyset$*

We now define a CPN-characterized Box that associates an interval box $\boldsymbol{B}$ with three sets of constraints representing the set of certainly satisfied constraints, possibly satisfied constraints and non satisfied constraints by box $\boldsymbol{B}$.

Definition 12 (CPN-characterized Box). *A CPN-characterized box $K^{\mathcal{S}}_{\boldsymbol{B}}$ w.r.t. a set of constraints $\mathcal{S}$ and a box $\boldsymbol{B}$ is a 4-tuple $\langle \boldsymbol{B}, \mathcal{C}, \mathcal{P}, \mathcal{N} \rangle$* s.t. *$\mathcal{C} = \mathsf{CS}(\mathcal{S}, \boldsymbol{B})$, $\mathcal{P} = \mathsf{PS}(\mathcal{S}, \boldsymbol{B})$ and $\mathcal{N} = \mathsf{NS}(\mathcal{S}, \boldsymbol{B})$.*

3.1 A naive algorithm

In order to compute an inner-approximation of the solutions of a Max-NCSP problem, we propose a first approach based on *evaluation-bisection* like Sam-Haroud and Jaulin [18, 16] work on inner approximation of a NCSP. The evaluation step consists in determining the consistency of each constraint w.r.t. the operators $\mathcal{CS}, \mathcal{PS}$ and $\mathcal{NS}$ for a given box. Exploring a branch of the search tree generated by the evaluate and bisect process ends as soon as the current box is smaller than a threshold or has not any partially satisfied constraints left. The Max-NCSP area is the conjunction of all the nodes with the greatest $\mathcal{CS}$. The whole search space is explored by this method.

This process is illustrated in IAMaxNCSP-SplitEval (*cf.* Alg.3), the *soundness* property of the solutions of this algorithm is ensured. The algorithm takes as parameters a set of constraints C and a box $\boldsymbol{B}$ representing the original search space. It outputs a lists of boxes that represent the inner approximations of the solutions of the Max-NCSP.

Alg. 3. Naive inner approximation algorithm for a Max-NCSP.

```
1 IAMaxNCSP-SplitEval(in: C; B ∈ I^n)
      out: a list ResultL of CPN boxes
2 begin
3   ResultL ← ∅
4   L ← {⟨B, CS(C), PS(C), NS(C)⟩}
5   while not empty(L)
6     ⟨B, cs, ps, ns⟩ ← chooseOneFrom(L)
7     if (|ps| = 0) then
8       ResultL ← ResultL ∪ ⟨B, cs, ps, ns⟩
9     else
        %still some partially satisfied constraints
10      L ← L ∪ split(B)
11    end
12  endwhile
13  return best(ResultL)
14 end
```

3.2 An inner extension approach

We propose to rely on a *branch and bound* approach to improve the computation; by maintaining an optimum value m of the number of constraints certainly satisfied. Alike a classical optimization process, we prune as soon as possible the dead branches of the search tree; *i.e.,* such that $|\mathsf{PS}(C, \boldsymbol{B})| + |\mathsf{CS}(C, \boldsymbol{B})| < m$. Our extension of the classical algorithm is improved by considering :

- evaluate a canonical box $\boldsymbol{P}$ chosen in $\boldsymbol{B}$ in order to update the optimum m more often. The canonical box possibly satisfies more constraints than $\boldsymbol{B}$ (being canonical indeed favors the fact that she might have less possibly satisfied constraints).
- we propose to compute a box $\boldsymbol{B'}$ by applying the inner extending operator in $\boldsymbol{B}$ starting from $\boldsymbol{P}$ in order to reduce the amount of search in $\boldsymbol{B}$.
- finally, we propose to use an outer contracting operator on box $\boldsymbol{B'}$ considering only the still partially satisfied constraints.

The algorithm is presented in Alg. 4. Let us consider a $\mathcal{CPN}$ $\langle \boldsymbol{B}, cs, ps, ns \rangle$ box at a given step in the solving process (line 6). First, we retrieve a point (*i.e.,* a canonical box) by applying the chooseConfiguration procedure, which returns a point in the current search box. If the point is better (*i.e.,* it satisfies more constraints than the current optimum), the optimum is updated. Then an inner extension is performed (*i.e.,* applying IEOA algorithm) in order to maximize the size of this box. An outer contraction propagation step is then enforced on the extended box.

It is natural to think that the performance of this algorithm is strongly dependent of when the optimal box (w.r.t. the CSP constraints satisfaction)

Alg. 4. Inner approximation algorithm for a Max-NCSP.

```
1  IAMaxNCSP(in: C; B ∈ I^n)
       out: a list ResultL of CPN boxes.
2  begin
3  maxSat ← 0
4  L ← {⟨B, CS(C), PS(C), NS(C)⟩}
5  while ¬empty(L)
6    ⟨B, cs, ps, ns⟩ ← chooseOneFrom(L)
7    if (|ps| = 0) then
8      ResultL ← ResultL ∪ ⟨B, cs, ps, ns⟩
9    else
       % searching for a canonical in B
10     ⟨P, cs', ps', ns'⟩ ← chooseConfiguration(B)
11     if (|cs'| > maxSat) then
         % updating the optimum
12       maxSat ← |cs'|
13     end
       % is this branch interesting ?
14     if (|cs' + ps'| > maxSat) then
15       B' ← AlgoIEA(cs', P, B, ε)
16       B'' ← propagation(B', ps')
17       L ← L ∪ split(B'')
18       L ← L ∪ {B'\B''}
19       L ← L ∪ {B\B'}
20     end
21   end
22 endwhile
23 return best(ResultL)
24 end
```

is determined. The quicker the optimum is found the sooner branches can be pruned. In the sequel we propose to instantiate the chooseConfiguration method by a local search method in order to find the optimum as soon as possible.

4 A Continuous Extension of the Local Search Framework

Incomplete resolution methods have been applied to a large set of discrete CSPs, and shown to be very efficient in practice (see for instance [15] or more recently [8]). Among them, local search algorithms rely on an iterative improvement of an initial random configuration. The notion of improvement refers to some penalty (or cost, or error) function f, which, within a CSP framework, usually represents the number of violated constraints in the current configuration (Min-Conflict [21]).

The basic algorithm starts from a random configuration, explores its neighborhood and eventually modifies it, under specific conditions : combination of improvements of the cost with a metaheuristic. Those steps are repeated until a stopping condition is satisfied (either a solution is found, or a maximum number of iterations is reached). The metaheuristic part of the algorithm consists in guiding the search at a higher level of abstraction than the elementary iteration, thus preventing short term cycling and guarantying some properties of the algorithm w.r.t. the search space (intensification, diversification). One of the most famous and most efficient metaheuristic is the well-known Tabu Search proposed separately by Glover [12] and Hansen [25].

Local search algorithms (and metaheuristics in general) rely on both the idea of the search being guided by the penalty function, and conversely a partly random exploration on the search space. These two notions are called diversification, intuitively defined as a way to explore the search space as widely as possible (random restarts are added after a number of iterations without improvement, for instance, as proposed in [13]), and intensification or aspiration, which is intuitively defined as a way to enforce search around supposedly promising areas of the search space. This is made, in the case of Tabu Search, by playing on the Tabu list T. Of course, a good balance between both has to be found, see [9] for more details.

By now, there have only been a few applications of metaheuristics techniques to continuous CSPs. Several population based methods have been proposed, such as genetic algorithms with the works of Michalewicz (see [20] for instance). Work from Chelouah and Siarry have extended numerous discrete metaheuristics to continuous domains, especially tabu search with ECTS [10]. A difficulty encountered is to deal with domains far bigger than in the discrete case. ECTS uses a metric on those domains in order to define concentric rectangles around the current configuration within which random neighbors configurations are chosen. Another continuous tabu search has been proposed by Battiti and Tecchiolli [3] in a hybrid algorithm in which continuous domains are treated like a grid of points covering the search space $\mathbb{R}^n$. Anglada *et al.* [2] propose all the same a continuous extension of an algorithm relatively close to tabu search, the adap-

tative search. In their case, the neighborhood depends on the penalty function *e.g.* by cutting for a given variable the subsets of its domain that improve the cost function.

In all these works, studied configurations are, like in the discrete case, continuous instantiations of the variables, *i.e.,* points in $\mathbb{R}^n$. This leads in reconsidering the neighborhood and the heuristic parts of the algorithm in order to handle the huge size of the domains taken into account ($\mathbb{R}$ discretized). Our approach is somehow orthogonal : we consider intervals of $\mathcal{I}^n$ instead of points in $\mathbb{R}^n$. Our algorithm is then centered on managing boxes-configurations and the continuous nature of the constraints we encounter. The metaheuristic part or tabu search of our algorithm is quite simple for now. This method is closer to work from Barichard and Hao [27] on PICPA, a hybrid method which metaheuristic part is a genetic algorithm. Individuals are boxes selected among a multi-objective criterion. In our algorithm, configurations are boxes selected w.r.t. a min-conflict criterion.

4.1 Interval configurations

In the discrete case, a local search method is usually described by three data.

- a neighborhood function, which assigns a set of neighbors to each configuration of the search space,
- the penalty function to optimize;
- a heuristic in order to prevent cycling.

But beneath, local search algorithms use some other operators which are very often neither defined nor identified, because their instantiation is obvious in the discrete case. In the first place, the notion of configuration is straightforward, a configuration trivially being an instantiation of the variables with discrete values. From this are derived the comparison between two configurations w.r.t. equality of the values of their variables. A solution being trivially identified as a configuration with a null penalty. Continuous applications may demand to precise those basic facts as these definitions are obviously not sufficient for dealing with intervals, for instance.

As said before, our application requires to handle continuous variables in a wider way, their domains being defined by intervals with floating points bounds. In order to consider a continuous local search method, we need to redefine the following elementary operations : a configuration, the penalty function and the neighborhood.

As a first choice, we need to define the set S of possible configurations. It could simply be taken as $\mathbb{F}$ as in the discrete case. But this would lead to loose the continuous properties of the domains as presented in section 2.1. Moreover computing penalties would become hazardous as for constraints evaluation for complete methods : rounding errors, equality constraints have a close to null probability to be fulfilled ($\mathbb{F}$ being a denumerable subset of $\mathbb{R}$), *etc.* Finally, when considering the inequalities, we want to compute a box of solutions (*i.e.,* an inner approximation of the solution set). The only way to achieve this is to define S as the set of floating point bounded intervals $\mathcal{I}^n$.

4.2 Choosing the LS operators

Once S has been chosen, we still have to give coherent definitions for the following operators : neighborhood function, penalty function and penalty comparison.

Neighborhood Function The local search neighborhood function has to return candidate configurations for the next iteration of the algorithm. We consider neighbor configuration both in geometrical (close to the current configuration) and computational (neighbors should be obtained quickly) ways.

The neighborhood of a configuration p is defined in quite the same way as in usual local search algorithms, as a box $\boldsymbol{B}_p$ centered on the current configuration. The problem in continuous domains is that if we want to explore this neighborhood, whatever its size, we would have to compute a huge set of boxes (with an order of magnitude of the power set of the $\boldsymbol{B}_p \cap \mathbb{F}$). Thus we introduce another operator that will generate nb_{neigh} new random configurations inside $\boldsymbol{B}_p$, those configurations creating the neighborhood to explore.

Our *neighborhood function* will be parameterized by the number nb_{neigh} of neighbors generated at each step. Hence our definition of *neighborhood function* is quite close to the notion of *Variable Neighborhood Search* (VNS) proposed by Hansen and Mladenovi'c [22].

Here is the reason why we can afford not adding a Tabu list. In a discrete case, the neighborhood is usually entirely explored, and it is of a reasonable size (w.r.t. the number of variables / size of the domains). Then a configuration is likely to be chosen twice within a small number of iterations. In our case, the definition of the neighborhood exploration makes it very unlikely to choose exactly the same configuration. This prevents of mere cycling. Of course, there is still a chance to have the search narrowed around a local minimum. The notions of diversification and intensification still need to be defined more in details. In the actual state of the algorithm, diversification is ensured by performing random restarts.

Penalty function In our application, the constraints are defined by usual arithmetic operators over real values, thus they are functions from $\mathbb{R} \rightarrow \{0, 1\}$. Here our goal is to optimize configurations w.r.t. constraints satisfaction by taking into account the different fulfillment possibilities of continuous constraints (inner and outer approximations of the solution). We thus have to perform different steps for our penalty function.

The only constraints we handle are inequalities of the form $f(V) \leq g(V)$ where f and g are real functions defined on subsets of variables. Let us take as an example a circle centered in 0 within XY plane, then we will have $f(x, y) = \sqrt{x^2 + y^2}$ and g will be a constant value (the radius of the circle). In order to perform the evaluation of the penalties on boxes instead of points, we have to rely on the power of interval extensions on real functions (*cf.* section 2.1). Indeed, we know how to compute the intervals images for f and g via their respective extensions F and G. Finally, penalty evaluation is done in a discrete Min-Conflict fashion : given a constraint c, we can state if the computed resulting

boxes certainly, partially or do not satisfy c. Intuitively, the idea is to count the satisfied constraints by extending the discrete Min-Conflict to intervals. Each constraint give us three different data as illustrated in figure 2. By adding these values, we obtain the number of certainly, possibly and non-satisfied constraints given a box. In the actual version of the algorithm, the penalty function is represented by the number of certainly satisfied constraints.

The continuous LS algorithm We can now provide the pseudo-code of the continuous LS algorithm. This algorithm has three parameters :

1. the number of random restarts (maxTries),
2. the number of iterations for each starting point (maxSteps),
3. the number of neighbors nb_{neigh} generated at each iteration.

Alg. 5. Continuous Local Search.

```
1 CLS(in: C; P ∈ I^n;
        in: maxTries,maxSteps,nb_neigh ∈ ℤ;)
        out: B ∈ I^n
2 begin
3   best ← initialConfiguration(C, P)
4   current ← best
4   nbTries ← 0
    % Diversification Loop
5   while nbTries < maxTries
6     nbSteps ← 0
      % Intensification Loop
7     while nbSteps < maxSteps
8       neigh ← genNeighbors(nb_neigh, current, C)
9       current ← getBestNeighbor(neigh, C)
10      if (cost(current) ≺_I cost(best)) then
11        best ← current
12      end
13      nbSteps + +
14    endwhile
15    nbTries + +
16  endwhile
17  return best
18 end
```

Our algorithm is presented in Alg. 5. It inputs a set of constraints $C = c_1, \ldots, c_n$, a search box $\boldsymbol{P}$ as well as the three parameters listed above.

5 Results

We compare the execution times of algorithms IAMaxNCSP-SplitEval, IAMaxNCSP-Mid et IAMaxNCSP-CLS for the same set of problems. Algorithm IAMaxNCSP-Mid is an instantiation of IAMaxNCSP where chooseConfiguration() simply returns the middle of the current box. IAMaxNCSP-CLS instantiates IAMaxNCSP in which chooseConfiguration() calls our continuous local search algorithm (CLS). The toy problems used, Circle-n, defined n circles with random centers and radiuses. The search space is restricted in interval $[-3, 3]^2$. Problems Sphere-n identically define spheres randomly distributed in $[-3, 3]^3$. Local search algorithm parameters are : maxTries=10, maxSteps=10, nb_{neigh}=10.

Results clearly show that this local search methods can not compete with mid point method, despite different parameterizing (number of neighbors, steps and random restarts). Although in most cases the optimum value (maximum number of satisfied constraints) is found first by the local search algorithm (very often this number is found at first iteration), this method looses the whole remaining time in evaluating non-promising areas of the space (typically all the areas on the borders of the relations). We have tried a few adaptative methods (give preference to local search in the beginning of the process before using mid point) but their performances are uneven on the different benchmarks.

Table 1. Results of SplitEval, Mid and CLS approaches in seconds on a Pentium M 1.7 GHz, 512Mo.

Bench. IAMaxCSP	-SplitEval time	-Mid time	-CLS time
$\varepsilon = 0.1$			
Circle5 (2-sat)	0.49	0.09	0.22
Circle25 (9-sat)	0.96	0.13	0.05
Circle100 (21-sat)	3.99	1.3	0.9
Sphere10 (3-sat)	10.19	1.21	1.21
Sphere100 (9-sat)	157.74	9.29	2.39
$\varepsilon = 0.01$			
Circle5 (2-sat)	3.87	1.9	1.8
Circle25 (9-sat)	16.14	0.54	0.51
Circle100 (21-sat)	133.04	1.02	1.62
Sphere10 (3-sat)	107.76	75.6	77.31
Sphere100 (9-sat)	$>$ 10mn	3.64	3.63
$\varepsilon = 0.001$			
Circle5 (2-sat)	24.87	3.92	7.56
Circle25 (9-sat)	127.64	2.0	1.72
Circle100 (21-sat)	$>$ 10mn	4.31	3.80
Sphere100 (9-sat)	$>$ 10mn	71.53	72.52

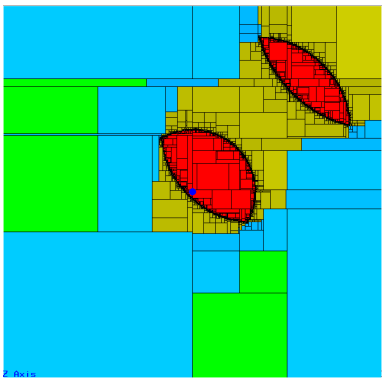

Fig. 3. Our approach applied on the Circle5 problem ($\varepsilon = 0.01$). Red areas represent boxes that maximize CSP satisfaction (2 constraints out of 5).

6 Conclusion

We have presented a method that can handle Max-NCSP problems by introducing an inner extending operator that is used conjointly with usual outer contracting operators. Results are presented on toy problems. A method mixing continuous local search has been tested without obtaining the expected results. A new framework has been defined for a continuous extension of local search, orthogonal to classical approaches that use points as configurations. Our framework is centered on the notion of intervals, being the only way to tackle the questions of inner and outer approximations of solutions and to manage the intrinsic complexity of continuous problems. Major counterpart lays in the difficulty to explain metaheuristics which still has to be developed.

References

1. G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press Inc., New York, USA, 1983.
2. Alexis Anglada, Philippe Codognet, and Laurent Zimmer. An adaptive search for the NSCSPs. In *CSCLP 2004*, Lausanne, 2004.
3. R. Battiti and G. Tecchiolli. The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization. *Annals of Operations Research*, 62, 1996.
4. F. Benhamou and F. Goualard. Universally quantified interval constraints. In *Procs. of CP'2000*, pages 67–82, 2000.
5. Frederic Benhamou. Interval constraint logic programming. In Andreas Podelski, editor, *Constraint programming: basics and trends, Chatillon Spring School, Chatillon-sur-Seine, France, May 1994*, volume 910, pages 1–21, 1995.
6. Frédéric Benhamou, Frédéric Goualard, Éric Languénou, and Marc Christie. Interval constraint solving for camera control and motion planning. *ACM Transactions on Computational Logic*, 5(4), October 2004.

7. Frédéric Benhamou, David McAllester, and Pascal Van Hentenryck. CLP(Intervals) revisited. In *Procs. of the International Symposium on Logic Programming (ILPS'94)*, pages 124–138, Ithaca, NY, November 1994. MIT Press.
8. C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268–308, 2003.
9. Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
10. R. Chelouah and P. Siarry. Tabu search applied to global optimization. *European Journal on Operational Research*, 2000.
11. Hélène Collavizza, François Delobel, and Michel Rueher. Extending consistent domains of numeric CSP. In *Procs. of the 16th IJCAI*, volume 1, pages 406–411, Stockholm, Sweden, July 1999.
12. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1997.
13. Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. *AAAI'98*, 2000.
14. Eldon Robert Hansen. *Global Optimization Using Interval Analysis*. Pure and Applied Mathematics. Marcel Dekker Inc., 1992.
15. Jin-Kao Hao, Philippe Galinier, and Michel Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Journal of Heuristics*, 1998.
16. D. Haroud and B. Faltings. Global consistency for continuous constraints. *Lecture Notes in Computer Science*, 874:40–50, 1994.
17. L. Jaulin and E. Walter. Guaranteed tuning, with application to robust control and motion planning. *Automatica*, 32(8):1217–1221, 1996.
18. Luc Jaulin and Éric Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993.
19. Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 1(8):99–118, 1977.
20. Z. Michalewicz. Genetic Algorithms, Numerical Optimization and Constraints. *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 151–158, 1995.
21. S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.
22. N. Mladenović and P. Hansen. Variable Neighborhood Search. *Comps. in Opns. Res.*, 24:1097–1100, 1997.
23. R. E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.
24. Arnold Neumaier. *Interval methods for systems of equations*, volume 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge Unversity Press, 1990.
25. Hansen P. The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on Numerical Methods in Combinatorial Optimization*, 1986.
26. Jamila Sam-Haroud and Boi V. Faltings. Consistency techniques for continuous constraints. *Constraints*, 1:85–118, 1996.
27. Barichard V. and Jin-Kao Hao. A population and interval constraint propagation algorithm. *LNCS*, 2632:88–101, 2003.
28. Pascal Van Hentenryck, Laurent Michel, and Yves Deville. *Numerica: A Modeling Language for Global Optimization*. The MIT Press, 1997.