# Clustering the Search Tree for Numerical Constraints

Xuan-Ha Vu, Djamila Sam-Haroud, and Boi Faltings

Artificial Intelligence Laboratory,
Swiss Federal Institute of Technology in Lausanne (EPFL),
CH-1015, Lausanne, Switzerland
{xuan-ha.vu, jamila.sam, boi.faltings}@epfl.ch
http://liawww.epfl.ch

**Abstract.** During search, most interval-based solvers perform splitting when no effective reduction can be made. Some connected regions may be undesirably split into small boxes. This makes the number of branches growing quickly, then impacts the overall performance. Regrouping connected boxes reduces the number of branches of the search tree, thus improves the overall performance. We propose to use clustering techniques based on the connectedness of boxes for the regrouping. This paper introduces new clustering algorithms, that fit in the context of search, and shows experimental results on numerical constraints.

## 1 Introduction

Many practical applications involve solving numerical constraints. A numerical constraint is a constraint on variables taking values from connected domains. In practice, numerical constraints can be equalities or inequalities of arbitrary type, usually expressed using arithmetic and logical expressions. Such constraints are often encountered in real-world engineering applications, where a set of solutions often expresses relevant alternatives that need to be identified as precisely and completely as possible.

Most interval-based solvers take as input a set of numerical constraints that need to be satisfied and repeatedly perform two operations: reduce (e.g. bound or prune) and branch (e.g. split). That usually results into a tree of boxes that *conservatively* encloses the feasible set during search. The solvers perform splitting when no effective reduction can be made. The side effect of splitting is that some connected regions may be undesirably split into small boxes. This makes the number of tree branches growing undesirably and impacts the overall performance of search. Naturally, one may think that regrouping connected boxes reduces the number of branches of the search tree in case the feasible set is complex and disconnected, thus improves the overall performance. We propose to use fast clustering techniques based on the connectedness of boxes to regroup a large number of boxes into a reduced collection of boxes each of which covers one or several connected subsets.

Clustering methods have been studied in statistics, and then expanded to machine learning/neural networks area and the database community. Developed algorithms are able to scan huge databases and extract knowledge patterns within the data. It is known that general computation of an optimal set of $k$ clusters is NP-complete [1]. For this reason, fast clusterings usually can be achieved only by using heuristic algorithms or by restricting to some approximations. Very recently, a fast clustering algorithm named `ANTCLUST` [2] was proposed. This algorithm is close to, but still not suitable for addressing our requirements. Though successful, the general clustering techniques are not suitable for directly applying to our case because they are not guaranteed to be convergent and the homogeneity information in our problems is not available as they require. Therefore, new clustering techniques for tree of boxes are needed.

We first propose a basic clustering algorithm called `COLONIZATION` that takes $O(N^2)$ time to cluster $N$ boxes in fixed-dimensions. The basic method computes homogeneity (i.e. the connectedness in this case) for each pair of boxes. Moreover, we observe that most search techniques implemented in interval-based solvers use *branch-and-prune* or *branch-and-bound* schemes, hence they essentially produce boxes in structure of tree.[1] Taking advantages of tree structure, we propose two more clustering algorithms for trees of boxes. One algorithm, called `MCC`, very quickly provides near-optimal clusterings such that no connected subsets are partitioned. Another algorithm, called `SDC`, generates adaptive clusterings in very short time as a further process of `MCC`. Combinations of the proposed clustering techniques are also discussed in Section 3.

## 2 Background and Definitions

### 2.1 Interval Arithmetic

The finite nature of computers precludes an exact representation of the *reals*. The real set, $\mathbb{R}$, is in practice approximated by a finite set $\mathbb{F}_\infty = \mathbb{F} \cup \{-\infty, +\infty\}$, where $\mathbb{F}$ is a finitely many set of reals usually corresponding to the floating-point numbers. In this paper, we restrict the notion of *interval* to refer to real intervals with bounds in $\mathbb{F}_\infty$ no matter they are open or closed [4]. The set of intervals with bounds in $\mathbb{F}_\infty$, denoted by $\mathbb{I}$, is partially ordered by the relation '$<$' on reals. Our previous results in [5, 6] can be easily extended to accept this notion. An *interval box*, or a *box* for short, $\mathbf{B} = I_1 \times \ldots \times I_n$ is a Cartesian product of $n$ intervals in $\mathbb{I}$. The projection of $\mathbf{B}$ on a subset, $X$, of its axes is denoted by $\mathbf{B}|_X$.

### 2.2 Relations and Approximations

We denotes by $pts(S)$ the set of points represented by object $S$, e.g. $pts(S) = \{x \mid x \in \mathbf{B} \in S\}$ if $S$ is a collection of boxes. A relation can be approximated by a computer-representable superset or subset as defined below. The former is a

[1] Even if the tree structure is not available, we still can construct a good bounding-box tree from a list of boxes in $O(N \log N)$ time [3].

*complete* approximation but may contain points that are not feasible. Conversely, the latter is a *sound* approximation but may lose certain feasible points. In practice, a relation can be approximated coarsely by the smallest box, called the *(interval) hull* and denoted by $hull(.)$, containing it. A set of objects representing points is called *disjoint* if every two objects have no common points. A set is called a *disconnected set* if it can be partitioned into two nonempty subsets such that each subset has no points in common with the set closure of the other, otherwise it is called a *connected set*. Moreover, in this paper we use the word '*highly disconnected set*' to imply that the distances between its subsets are long, and use '*non-highly disconnected set*' otherwise. Approximating a relation usually requires a decomposition into simple components like convex subsets. In this paper, we are investigating one of decompositions of particular interest that is the partition into disjoint boxes.

## 3 Clustering the Search Tree

During solving a system of numerical constraints with a non-isolated feasible set, interval-based search techniques following the branch-and-prune or branch-and-bound schemes produce a large collection of boxes that can be maintained in a *bounding-box tree*, where child nodes represent for branchings in search. Bounding boxes stored at search nodes are results of pruning phase. In literature, there are some variants of bounding-box tree like *bounding-volume tree*, *interval tree*, *box-tree* and *AABB tree*. We call the boxes produced by the solvers the *primitive boxes*. In this paper, we use the notion of *hull of boxes* to refer to the smallest box that contains all the given boxes, however, for convenience it also refers to the collection of the involved boxes if not confused. A collection of primitive boxes is called *connected* if the union of its boxes is a connected set. If this is the case, we say that the primitive boxes connect to each other. A collection of primitive boxes is called *max-connected* (w.r.t to the set of all the primitive boxes) if it is connected and there is no other primitive boxes that connect to the given boxes. In this paper, we will only focus on clusterings which construct disjoint clusters fully covering all primitive boxes such that each cluster is a hull of its primitive boxes. A clustering is called *max-connected* if each connected collection in a cluster is max-connected. A bounding-box tree is called *orthogonal-separable* if every decomposition of each bounding-box into pairwise-disconnected collections (of primitive boxes) can be performed by sequently using separating hyper-planes that are orthogonal to axes (see Figure 2).

***Goals Of Clustering.*** The non-isolated feasible set of a system of numerical constraints usually consists of one or more connected subsets each of which is a continuum. In many applications, there is a need for enclosing the feasible set by a collection of disjoint bounding boxes such that each of the connected subsets is contained in only one bounding box. To describe the connected subsets as well as possible, the number of bounding boxes should be as big as possible. Interval-based search techniques usually produce approximations that are

either very poor (if being stopped at low precision), or prohibitively verbose (if being stopped at medium/high precision) when solving problems with non-isolated and disconnected feasible sets. When a certain quality is required, we obviously need to take the latter setting and then do a clustering on the large set of boxes. However, the exact feasible set is unknown yet, and only a set of primitive boxes is known. The above-mentioned need is translated into the need for a max-connected clustering. If a max-connected clustering has the maximum number of clusters, we call it an *optimal max-connected clustering*[2]. Optimal max-connected clusterings well characterize highly disconnected feasible sets, where each hull (i.e. a cluster) is expected to contain only one max-connected collection of primitive boxes (see Figure 7-a). In case feasible sets are non-highly disconnected, the optimal max-connected clusterings may not well characterize the feasible set (see Figure 7-b). In this case we may need a further decomposition to characterize the feasible set better (see Figure 8-a).

In the next subsections, we propose four algorithms to address different goals. The first subsection focuses on a basic algorithm that computes the optimal max-connected clustering. It is however not very efficient in practice, we then incrementally propose three alternative algorithms. Each algorithm has two phases, but they have the same first phase. The second subsection describes the common phase. The third subsection describes an algorithm to compute max-connected clusterings which is near-optimal. The fourth subsection gives an algorithm to compute an adaptive clusterings. The last subsection gives discussion about combinations of the proposed clustering algorithms.

### 3.1 Optimal Max-Connected Clustering

As far as we know, there not exists any algorithm that are suitable to find the optimal max-connected clustering. Fortunately, there exists a recent clustering algorithm named `ANTCLUST` [2] that exploits the phenomenon known as *colonial closure* of ants to create homogeneous groups of individuals. The `ANTCLUST` algorithm addresses the general clustering problem in the way that we can borrow a part for addressing our goals. Inspired by Seidel's *invasion* algorithm [7] and some ideas in the `ANTCLUST` algorithm, we propose a simple deterministic algorithm, called `COLONIZATION`, which is given in Figure 1 to compute the optimal max-connected clustering. This basic algorithm tests each pair of boxes to check if they connect to each other. It is easy to see that in the worst-case the lines 02-12 take $d(1 + 2 + \ldots + (N - 1)) = dN(N - 1)/2$ checks for connectedness of two intervals, where $N$ is the number of primitive boxes in $d$-dimensions. As a result, the time complexity of these lines is $O(dN^2)$. At the line 13, each hull contains exact one max-connected collection, but those hulls are not guaranteed to be disjoint. The number of hulls at this line is therefore equal to the maximum number, $p$, of max-connected collections of primitive boxes. Consequently, the lines 14-16 have complexity $O(dp^2)$. We obviously have $p \leq N$, hence the total

[2] It is easy to prove that the optimal max-connected clustering exists uniquely.

```
01: algorithm COLONIZATION
02:     L := ∅;
03:     for each box, B, not in any collection in L do
04:         C := {B};
05:         for each C_i ∈ L do
06:             if {B} ∪ C_i is connected then
07:                 C := C ∪ C_i;
08:                 L := L \ {C_i};
09:             end-if
10:         end-for
11:         L := L ∪ {C};
12:     end-for
13:     Replace each collection by the hull of its primitive boxes.
14:     while there exist two hulls that have nonempty intersection do
15:         Combine the two hulls into a single hull.
16:     end-while
17: end
```

**Fig. 1.** The `COLONIZATION` algorithm

time complexity of the `COLONIZATION` algorithm is $O(dN^2)$, or $O(N^2)$ if $d$ is fixed. In practice, we have $p \ll N$ and $p$ is bounded for fixed problems.

### 3.2 Separator Computation

In order to cluster the primitive boxes into a number of disjoint hulls, we may use hyper-planes that are orthogonal to axes to separate the primitive boxes. We then define new notations for computing such separating hyper-planes as follows.

**Definition 1 (Separator, `SPT`).** *Given a hull,* $\mathbf{H}$*, of primitive boxes, an axis* $x \in \mathbb{N}$*, and an interval,* $I \in \mathbb{I}$*. The couple* $(x, I)$ *is called a* separator *of* $\mathbf{H}$ *if* $I$ *is a maximal (w.r.t. the set inclusion) interval that satisfies the following condition for all primitive box,* $\mathbf{B}$*, in* $\mathbf{H}$*:* $I \subseteq \mathbf{H}|_x \wedge I \cap \mathbf{B}|_x = \emptyset$*. When this holds,* $I$ *is called a* separating interval *and* $x$ *is called a* separating axis*. The set of all separators of* $\mathbf{H}$ *and the set of all separators on axis* $x$ *is denoted by* $\texttt{SPT}(\mathbf{H})$ *and* $\texttt{SPT}(\mathbf{H}, x)$ *respectively.*

**Definition 2 (Extension, `EXT`).** *Given a hull,* $\mathbf{H}$*, of primitive boxes, an axis* $x \in \mathbb{N}$*, and a box* $\mathbf{B} \subseteq \mathbf{H}$*. A couple* $(x, I)$ *is called an* extension *of* $\mathbf{B}$ *w.r.t.* $\mathbf{H}$ *(on axis* $x$*) if* $I$ *is an interval that is maximal (w.r.t. the set inclusion) in* $\mathbf{H}|_x \setminus \mathbf{B}|_x$*. When this holds,* $I$ *is called an* extending interval *and* $x$ *is called an* extending axis*. We denote by* $\texttt{EXT}(\mathbf{B}, x)$ *the set of extensions of* $\mathbf{B}$ *(w.r.t.* $\mathbf{H}$*) on axis* $x$*.*

Figure 2-a gives an illustration of the two notions. It is easy to see that, on each axis, a hull has at most two extensions w.r.t. its parent hull and that all separators of the hull lie between the two extensions. In a bounding-box tree of primitive boxes, we do a process in a bottom-up manner (e.g. in post-order) to

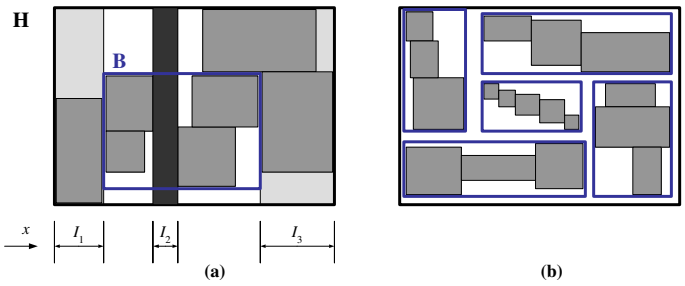


**Fig. 2.** Dark grey boxes are primitive. (a) Orthogonal-separable tree: $(x, I_2)$ is a separator of $\mathbf{H}$, $(x, I_1)$ and $(x, I_3)$ are extensions of $\mathbf{B}$ to $\mathbf{H}$. (b) The tree is not orthogonal-separable.

make each bounding box to be the hull of its child bounding boxes. Hence, each node's box becomes the hull of primitive boxes contained in it. A bounding-box tree whose bounding boxes are the hulls of primitive boxes contained in the bounding boxes is called a *fitted tree*.

**Definition 3 (Separating Set, SE).** *In a fitted tree, given an axis $x \in \mathbb{N}$ and a hull, $\mathbf{H}$, at a node. The union of the intervals of all separators and extensions*[3] *of $\mathbf{H}$ on axis $x$ is called the* separating set *of $\mathbf{H}$ on axis $x$ and denoted by* $\texttt{SE}(\mathbf{H}, x)$.

For simplicity, we use the same notions SPT, EXT, SE for the tree node corresponding to the hull $\mathbf{H}$. The computation of extensions of a hull w.r.t. its parent hull is trivial and takes $O(d)$ time. We observe that, in a fitted tree, a couple $(x, I)$ is a separator of a hull if and only if $I$ is a maximal (w.r.t. the set inclusion) interval that can be contained in all the separating sets on axis $x$ of all children of the hull. It means that all the separators of the hull can be computed from separators and extensions of its children. Moreover, this is true even during the above-mentioned bottom-up process. The ordering relation among separators and extensions is given by the following proposition.

**Proposition 1.** *In a fitted tree, given a hull, $\mathbf{H}$, of primitive boxes. All the separators and extensions of $\mathbf{H}$ can be* totally ordered *by the partial order '$<$' on $\mathbb{I}$ and the total order on $\mathbb{N}$. That is, for any two separators/extentions, $s_1$ and $s_2$, of $\mathbf{H}$, either $s_1 = s_2$, $s_1 < s_2$ or $s_2 < s_1$ holds. Where the ordering relations are defined as follows.*

$$(i)\ (x_1, I_1) = (x_2, I_2) \Leftrightarrow x_1 = x_2 \wedge I_1 = I_2$$
$$(ii)\ (x_1, I_1) < (x_2, I_2) \Leftrightarrow x_1 < x_2 \vee (x_1 = x_2 \wedge I_1 < I_2)$$

*Proof.* By Definition 1 and Definition 2, one can easily see that every two distinct separators/extensions of $\mathbf{H}$ on the same axis have empty intersection, otherwise either the separators/extensions are not maximal w.r.t to the set inclusion or $\mathbf{H}$

[3] If not specified, the extensions of a hull are taken w.r.t. the parent hull in the tree.

```
function FittingProcess(in/out : T_0)        /* T_0 is a bounding-box tree */
    for each node P of T_0 in a post-order visit do
        if P is leaf then
            for each axis x do SPT(P, x) := ∅;
        else
            pullSeparators(P);
        end-if
    end-for
end

function pullSeparators(in/out : P)          /* P is a tree node */
    if P is leaf then return;
    C := children(P);
    Set the box at P to hull({B ∈ I^d | B is a box at a node in C}).
    for each C ∈ C and each axis x do
        Compute EXT(C, x) and denote it by {E_l, E_u}.        /* may be empty */
        SE(C, x) := {E_l, SPT(C, x), E_u};                    /* ordered set */
    end-for
    for each axis x do SPT(P, x) := ∩_{C∈C} pts(SE(C, x));    /* ordered intersection*/
    SPT(P) := {SPT(P, 1), ..., SPT(P, d)};                    /* ordered set */
end
```

**Fig. 3.** The functions for the fitting process

is not the hull of its children. As a result, the set of all separating and extending intervals on an axis is totally ordered. This results in what we have to prove.

To compute separators of all nodes in a bounding-box tree, we only need to do a single bottom-up process, called the *fitting process*. In summary, in the fitting process, operations at each node consist of *(i)* making the box at the current node to be the hull of its children; and *(ii)* computing the ordered set of separators by taking the ordered intersection of ordered separating sets of its children. The details of the fitting process is given in Figure 3. We denote by $m$ the maximum number of children of a tree node. The operation *(i)* is trivial and takes $O(md)$ time. The leaf nodes of the tree have no separators. Each node in the tree has at most two extensions that straddle the box stored at the node, hence straddle the set of separators. Moreover, the bottom-up process starts at leaf nodes, therefore the separating sets can be computed by the intersection in the operation *(ii)* and maintained sorted by the total order in Proposition 1. At the current node, we denote by $q_{i,j}$ the number of elements in the separating set on axis $i \in \mathbb{N}$ of the $j$-th child hull, where $1 \leq i \leq d, 1 \leq j \leq m$. We denote $\bar{q}_j = \sum_i q_{i,j}$ and $\bar{q} = \max_j\{\bar{q}_j\}$, then $\bar{q}$ is the maximum number of separators/extensions that a child of the current node can have. Noting that each separating set on an axis is totally ordered, we have time complexity of computing the intersection of at most $m$ sorted separating sets on axis $i$ is $O(\sum_j q_{i,j})$. As a result, time complexity of the operation *(ii)* is $O(\sum_i \sum_j q_{i,j}) = O(\sum_j \sum_i q_{i,j}) = O(\sum_j \bar{q}_j)$, i.e. not greater than $O(m\bar{q})$. Because the number of nodes in the tree is $O(N)$,

the total time complexity of the fitting process is $O(mdN + Q)$, where $Q$ is the total number of separators/extensions in the tree except in the root. This can not exceed $O((md+q)N)$, where $q$ is the maximum number of separators/extensions that each node in the tree has. In most existing solvers, $m$ is not greater than $2d+1$, and it is usually small in comparison with $2d+1$ (e.g. $m = 2$ if bisection is used). We conjecture that $q$ is bounded for fixed problems, particularly it is bounded by $O(p)$ ($p$ in Section 3.1). If this conjecture is true, as our experiments show, the time complexity of the fitting process is $O(N)$ for fixed problems.

### 3.3 Max-Connected Clustering

The second phase is called the *separating process*. We now compute a nearly optimal max-connected clustering based on next observations. During the separating process, we maintain max-connected collections of primitive boxes in form of fitted trees.

- *(Observation 1).* Any separator stored at the root of a fitted tree representing a max-connected collection can be used to partition this tree into two fitted subtrees each of which represents a max-connected collection.
- *(Observation 2).* If a set of primitive boxes represented by a fitted tree can be partitioned by a hyper-plane that is orthogonal to axis $x$ into two collections whose projections on $x$ are disjoint, then the root of the fitted tree has some separating interval that contains the projection of the hyper-plane on the coupled separating axis.

Proof of these observations is trivial due to Definition 1 and the definition of fitted tree. By these observations, we can compute a max-connected clustering from the separators computed in Section 3.2. The process can be done in bottom-up manner, or simply by a recursive call at root. For simplicity, we describe the process in recursive mode. A recursive call starts from a root node of a fitted

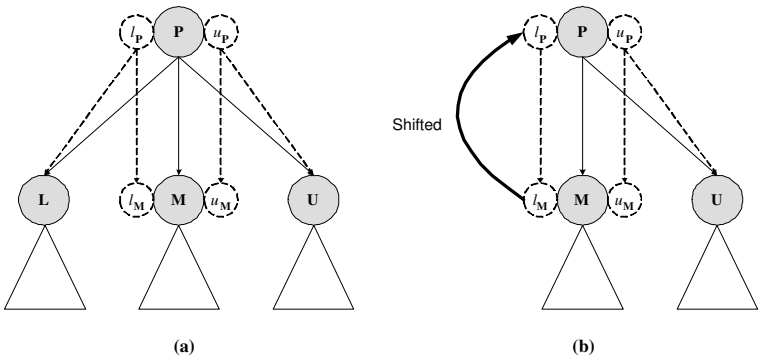


**Fig. 4.** The separating process: separate the primitive boxes based on separators

/* Input: $\mathbf{P}$ is a node of a fitted tree, separator $S \in \texttt{SPT}(\mathbf{P})$ */
**function** *separateSubtree*($\mathbf{in} : \mathbf{P}, S$) $\rightarrow$ $\mathbf{out} : \{l_{\mathbf{P}}, u_{\mathbf{P}}\}$
    Remove the separator $S$ from $\texttt{SPT}(\mathbf{P})$.
    Create two trees, $l_{\mathbf{P}}$ and $u_{\mathbf{P}}$, each has a single node copied from $\mathbf{P}$.
    **for** each $\mathbf{C} \in$ *children*($\mathbf{P}$) **do**
        **if** $\mathbf{C}$ lies on lower side of $S$ **then**
            Move the subtree rooted at $\mathbf{C}$ to a new subtree of the root of $l_{\mathbf{P}}$.
        **else if** $\mathbf{C}$ lies on upper side of $S$ **then**
            Move the subtree rooted at $\mathbf{C}$ to a new subtree of the root of $u_{\mathbf{P}}$.
        **else**
            Find the separator $S' \in \texttt{SPT}(\mathbf{C}) : S \subseteq S'$.
            $\{l_{\mathbf{C}}, u_{\mathbf{C}}\} :=$ *separateSubtree*($\mathbf{C}, S'$);
            Attach $l_{\mathbf{C}}$ (resp. $u_{\mathbf{C}}$) as a new subtree of the root of $l_{\mathbf{P}}$ ($u_{\mathbf{P}}$ resp.).
        **end-if**
    **end-for**
    If any root in trees $l_{\mathbf{P}}$ or $u_{\mathbf{P}}$ has only one child, shift this child to the root.
    *pullSeparators*(*root*($l_{\mathbf{P}}$)); *pullSeparators*(*root*($u_{\mathbf{P}}$));
**end**

**Fig. 5.** The separating process of a node/subtree

/* $\mathcal{T}_0$ is the initial bounding-box tree, $\mathcal{L}$ is a list of fitted tree to be return */
**algorithm** MCC($\mathbf{in} : \mathcal{T}_0$) $\rightarrow$ $\mathbf{out} : \mathcal{L}$
    *FittingProcess*($\mathcal{T}_0$); $\mathcal{L} := \{\mathcal{T}_0\}$;
    **while** $\exists \mathcal{T} \in \mathcal{L}$: $\mathcal{T}$ has at least one separator, $S$, at root **do**
        $\mathcal{L} := (\mathcal{L} \setminus \mathcal{T}) \cup$ *separateSubtree*(*root*($\mathcal{T}$), $S$);
    **end-while**
**end**

**Fig. 6.** The Max-Connected Clustering (MCC) algorithm

tree created during the separating process. Recursively, for each node, $\mathbf{P}$, and the current separator, $S \in \texttt{SPT}(\mathbf{P})$, we construct two fitted trees from the subtree rooted at $\mathbf{P}$. One tree is called *lower-bound tree* (denoted by $l_{\mathbf{P}}$), the other is *upper-bound tree* (denoted by $u_{\mathbf{P}}$) with single roots being copied from $\mathbf{P}$ at first. If a child node's box lies on lower-bound (respectively upper-bound) side of $S$, the subtree rooted at the child node is moved to $l_{\mathbf{P}}$ ($u_{\mathbf{P}}$, respectively). Otherwise the child node, called $\mathbf{M}$, is processed similarly to construct its lower-bound and upper-bound trees, $l_{\mathbf{M}}$ and $u_{\mathbf{M}}$ respectively. The trees $l_{\mathbf{M}}$ and $u_{\mathbf{M}}$ are then attached to $l_{\mathbf{P}}$ and $u_{\mathbf{P}}$ respectively. If any tree root in this process has only one child, this child is shifted to the root. Additionally, we make roots of lower-bound and upper-bound trees to be hulls of their child nodes. The separators of the root nodes of new lower- and upper-bound trees are updated from its children. Figure 4 gives an illustration of the process. Figure 5 gives the details of this process for a subtree rooted at a node in a fitted tree. Figure 6 gives the *Max-Connected Clustering* (MCC) algorithm that performs the separating process to get a max-connected clustering. Without difficult, we can prove that

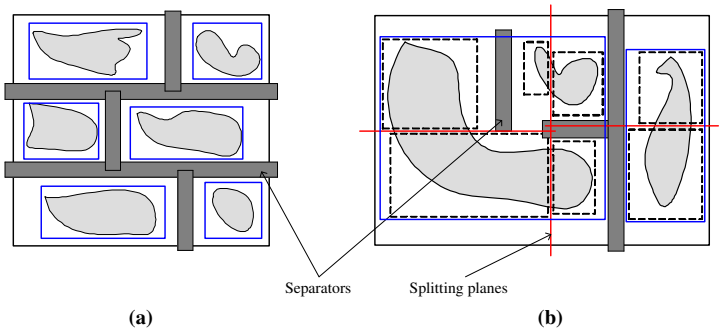


**Fig. 7.** MCC: The set of blue boxes is the optimal max-connected clustering of: (a) highly disconnected feasible set; (b) non-highly disconnected feasible set

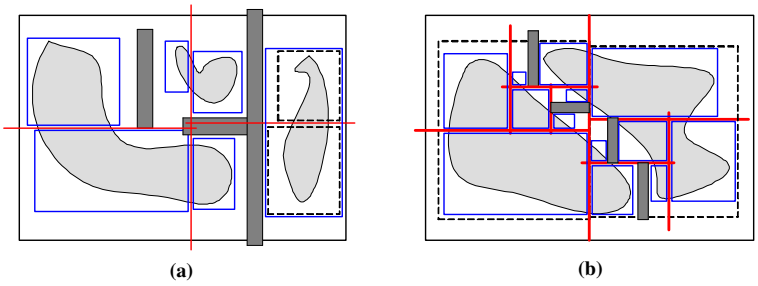


**Fig. 8.** SDC: Separators are considered as hints for decomposition. The set of blue boxes is an adaptive clustering of non-highly disconnected feasible set

the obtained max-connected clustering is optimal if the bounding-box tree is orthogonal-separable. For simplicity, in the MCC algorithm we use only one list, $\mathcal{L}$, however in implementation it should be maintained as two lists: one list for the trees that have no separators at root and the other list for the rest. Figure 7 gives illustrations on the clusters obtained by the MCC algorithm. It is easy to see that the time complexity of the MCC algorithm is $O(mdN + Q)$, that is practically shown to be linear in $N$ for fixed problems.

### 3.4 Separator-Driven Clustering

In some applications, the clustering obtained by the MCC algorithm characterize feasible sets not well enough (see Figure 7-b). A remedy for this problem is given as an additional process for MCC so that applications can choose to whether to run it. We observe that the separators that still exist in output fitted trees of the MCC algorithm (e.g. two separators in the left of Figure 7-b are not at roots of output trees) can be used as a guide for further separations. If we consider

```
/* T_0 is the initial bounding-box tree, L is a list of fitted tree to be return */
algorithm SDC(in : T_0) → out : L
    L := ∅; L_0 := MCC(T_0);        /* call to the MCC algorithm */
    for each fitted tree T ∈ L_0 do
        L := L ∪ separateByAllSeparators(T);
    end-for
end

function separateByAllSeparators(in : T) → out : L
    Search in post-order, from left to right, for a node N_0 such that SPT(N_0) ≠ ∅.
    if not found then return L := {T};
    L := ∅; N := N_0; P := parent(N);        /* P may be null */
    while P ≠ ∅ do
        for each C ∈ children(P) do
            if C on the left of N then Move the subtree rooted at C to L;
            if C on the right of N then
                Detach the subtree rooted at C and make a new tree T_C.
                L := L ∪ separateByAllSeparators(T_C);
            end-if
            if C = N ≠ N_0 then Erase the node C;
            if C = N = N_0 then
                Detach the subtree rooted at C and make a new tree T_C.
                Find a separator S ∈ SPT(C);
                L := L ∪ separateSubtree(root(T_C), S);        /* In Figure 5 */
            end-if
        end-for
        N := P; P := parent(N);        /* P may be null */
    end-while
end
```

**Fig. 9.** The Separator-Driven Clustering (SDC) algorithm

the remaining separators (after performing MCC) as hints for decomposing the collection of boxes into smaller groups that well characterize the feasible set, we will need to use all the splits (i.e. the branchings) that exist in the path from the current node upward to the root for separating the corresponding child boxes, though they may not make the child boxes disconnected. That is, all siblings of the nodes from the current node upward to the root are to be separated into different groups/collections in the clustering. Figure 8-a gives the result of further process for the problem in Figure 7-b. Six boxes obtained in Figure 8-a describe the feasible set better than two boxes in Figure 7-b. Figure 8-b gives another illustration on the SDC algorithm where two close connected subsets are well covered by 13 boxes each of which is a hull of primitive boxes. Figure 9 gives the main steps of the algorithm, called *Separator-Driven Clustering* (SDC), which performs the above idea as an additional process for the MCC algorithm. The SDC algorithm quickly provides an adaptive clustering as shown in our experiments.

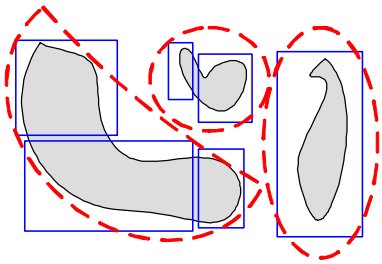

**Fig. 10.** Three clusters are represented by red slashed lines.

### 3.5 Combination of Clustering Techniques

We observe that if we apply the `COLONIZATION` algorithm to each fitted tree produced by the `MCC` algorithm we will get the optimal max-connected clustering. Therefore, the algorithm obtained from the `MCC` algorithm by applying the `COLONIZATION` algorithm in this way is called the *Optimal Max-Connected Clustering* (`OMCC`) algorithm. We can see that the time complexity of the `OMCC` algorithm is $O(mdN+Q+d\sum_i N_i^2)$, where $N_i(1 \leq i \leq p)$ is the number of primitive boxes in the fitted trees produced by the `MCC` algorithm. It is easy to see that $Q$ does not exceed $O(dN^2)$. We also have $\sum_i N_i = N$ and $N^2/p \leq \sum_i N_i^2 \leq N^2$, then $O(mdN + Q + d\sum_i N_i^2)$ does not exceed $O(dN^2)$. In practice we often see that $Q$ is much smaller than $O(dN^2)$ and the actual running time of `OMCC` is better than of `COLONIZATION` for problems with disconnected feasible set ($p > 1$). Note that we can also do a similar process for the `SDC` algorithm. Combining `SDC` and the first phase of `COLONIZATION` (lines 01-12) results into a more adaptive clustering, however, some bounding boxes of the clusters may overlap. Figure 10 gives the result of applying this combination to the problem in Figure 8-a.

## 4 Experiments

To evaluate the proposed clustering techniques, we now present experimental results on 18 non-linear constraint satisfaction problems which are selected to reflect different topologies of feasible set and categorized into three groups: *(i)* problems with only one connected subset; *(ii)* problems with several connected subsets; and *(iii)* problems with more than ten connected subsets. Our experiments show the similarity in running time of each algorithm in each group. Therefore, due to lack of space we only give the average running time for individual groups in Table 1. Figure 11 shows the graph of average running time in each group and all groups. The results show that the running time of `COLONIZATION` is quadratic in $N$ (the number of boxes) while the running time of `MCC` and `SDC` is very small (the average time of clustering 5000 boxes is less than 50ms, and the running time is always less than 120ms). The running time of the selected constraint satisfaction solver is linear in $N$. The solver

**Table 1.** Average running time (ms) for three groups

| $N =$ | 100 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 | 4000 | 4500 | 5000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *(i)* Solver | 45 | 226 | 461 | 728 | 978 | 1261 | 1556 | 1849 | 2096 | 2440 | 2736 |
| *(i)* COLO. | 2 | 49 | 194 | 432 | 767 | 1196 | 1831 | 2529 | 3564 | 4809 | 6053 |
| *(i)* MCC | 1 | 2 | 5 | 8 | 10 | 13 | 16 | 18 | 20 | 25 | 27 |
| *(i)* SDC | 1 | 2 | 6 | 9 | 11 | 15 | 17 | 20 | 23 | 28 | 31 |
| *(ii)* Solver | 69 | 285 | 525 | 774 | 1024 | 1263 | 1538 | 1788 | 2002 | 2288 | 2601 |
| *(ii)* COLO. | 2 | 25 | 78 | 172 | 290 | 447 | 636 | 863 | 1120 | 1453 | 1850 |
| *(ii)* MCC | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 15 | 16 | 19 | 21 |
| *(ii)* SDC | 0 | 2 | 4 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 22 |
| *(iii)* Solver | 66 | 415 | 860 | 1253 | 1683 | 2112 | 2557 | 2935 | 3371 | 3883 | 4271 |
| *(iii)* COLO. | 1 | 19 | 46 | 83 | 148 | 229 | 310 | 420 | 579 | 740 | 949 |
| *(iii)* MCC | 1 | 6 | 12 | 18 | 24 | 31 | 38 | 41 | 42 | 46 | 49 |
| *(iii)* SDC | 1 | 7 | 12 | 18 | 24 | 32 | 39 | 41 | 43 | 46 | 50 |

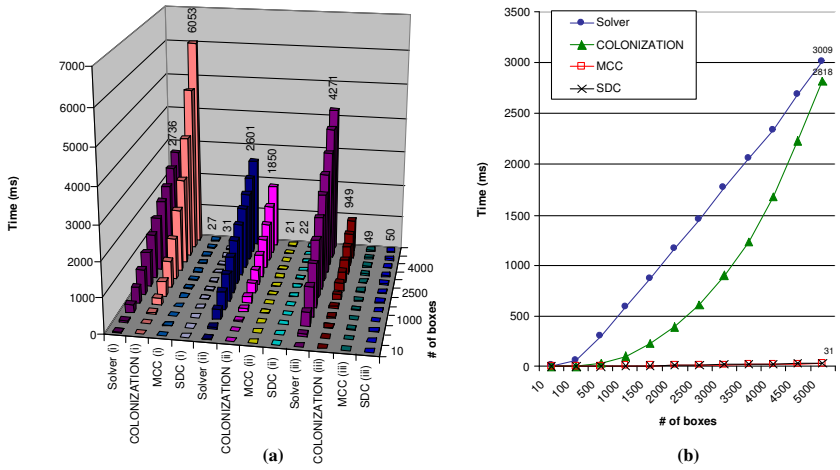


**Fig. 11.** Average running time in (a) 3 groups; (b) all the problems

is used to generate the tree of boxes that cover the feasible set. For all problems, the running time of MCC and SDC much less than that of COLONIZATION. In the first group, COLONIZATION, MCC and SDC provide the same clustering which consists of a single cluster. COLONIZATION and MCC provide the same clustering for all the problems. Moreover, for most problems the three algorithms produce the similar clusterings. They only show significant differences in the following problems: $F2.4 = \{\sin(x \sin y) \geq \cos(y \cos x); -4 \leq x, y \leq 4\}$ (Sinusoid); $G1.4 = \{x_1^2 + 0.5x_2 + 2(x_3 - 3) \geq 0, x_1^2 + x_2^2 + x_3^2 \leq 25, \forall i : -8 \leq x_i \leq 8\}$; $H1.1 = \{x_1^2 + x_2^2 + x_3^2 \leq 9, (x_1 - 0.5)^2 + (x_2 - 1)^2 + x_3^2 \geq 4, x_1^2 + (x_2 - 0.2)^2 \geq$

$x_3, \forall i : -4 \leq x_i \leq 4\}$. For example, when $N = 1000$, `COLONIZATION` and `MCC` provide six boxes for the problem $F2.4$, the volume ratio of the six boxes to the hull of primitive boxes is 0.761 while the ratio obtained by `MCC` is 0.385 with 30 boxes. When $N = 1000$ for problem $G1.4$ (and $H1.1$ respectively), `COLONIZATION` and `MCC` produce no reduction while `SDC` produces 4 (5, respectively) boxes with the volume ratio is 0.225 (0.191, respectively). The experiments show that even in case `COLONIZATION` and `MCC` cannot characterize the feasible set well (e.g. the problems $G1.4$ and $H1.1$), `SDC` still can provide an adaptive clustering that are more suitable for applications.

## 5 Conclusion

Four algorithms addressing different goals have been proposed to regroup tree of boxes into a fewer number of boxes. We have done experiments with three of them. In our experiments, `SDC` shows to be the best among the techniques we proposed and tested. Moreover, `MCC` and `SDC` are very cheap (in running time) postprocessing to get useful information for further process. They also preserve the tree structure of the input tree. In the COCONUT project, we are investigating a cooperation of optimization and numerical constraint satisfaction where we could exploit these fast postprocessing to make the interval-based constraint satisfaction solvers more useful in the context of optimization. Obviously, these techniques can also be applied to ad-hoc discrete constraints if we translate each integer into an unit box and use the technique in [3] to construct a good bounding-box tree from a list of unit boxes.

## 6 Acknowledgments

This research was funded by the European Commission and the Swiss Federal Education and Science Office through the COCONUT project (IST-2000-26063).

## References

1. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
2. Labroche, N., Monmarché, N., Venturini, G.: A New Clustering Algorithm Based on the Chemical Recognition System of Ants. In: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'2002), France, IOS Press (2002) 345–349
3. Agarwal, P., de Berg, M., Gudmundsson, J., Hammar, M., Haverkort, H.: Box-Trees and R-Trees with Near-Optimal Query Time. In: Proceedings of the 17th ACM Symposium on Computational Geometry, ACM Press (2001) 124–133
4. Benhamou, F., Goualard, F.: Universally Quantified Interval Constraints. In: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'2000). (2000) 67–82

5. Vu, X.H., Sam-Haroud, D., Silaghi, M.C.: Approximation Techniques for Nonlinear Problems with Continuum of Solutions. In: Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation (SARA'2002), (Canada) 224–241
6. Vu, X.H., Sam-Haroud, D., Silaghi, M.C.: Numerical Constraint Satisfaction Problems with Non-isolated Solutions. In: Post-Proceedings of the 1st International Workshop on Global Constrained Optimization and Constraint Satisfaction (COCOS'2002), France (2002) To appear.
7. Tsang, E.: Foundations of Constraint Satisfaction. Academic Press (1993)