Constructive Interval Disjunction

Gilles Trombettoni[1] and Gilles Chabert[2]

[1] University of Nice-Sophia and COPRIN Project, INRIA, 2004 route des lucioles, 06902 Sophia.Antipolis cedex, B.P. 93, France
[2] ENSIETA, 2 rue François Verny, 29806 Brest cedex 09, France
`trombe@sophia.inria.fr, gilles.chabert@ensieta.com`

Abstract. This paper presents two new filtering operators for numerical CSPs (systems with constraints over the reals) based on *constructive disjunction*, as well as a new splitting heuristic. The fist operator (`CID`) is a generic algorithm enforcing constructive disjunction with intervals. The second one (`3BCID`) is a hybrid algorithm mixing constructive disjunction and *shaving*, another technique already used with numerical CSPs through the algorithm `3B`. Finally, the splitting strategy learns from the CID filtering step the next variable to be split, with no overhead.

Experiments have been conducted with 20 benchmarks. On several benchmarks, `CID` and `3BCID` produce a gain in performance of orders of magnitude over a standard strategy. `CID` compares advantageously to the `3B` operator while being simpler to implement. Experiments suggest to fix the CID-related parameter in `3BCID`, offering thus to the user a promising variant of `3B`.

1 Introduction

We propose in this paper new advances in the use of two refutation principles of constraint programming: shaving and constructive disjunction. We first introduce shaving and then proceed to constructive disjunction that will be considered an improvement of the former.

The shaving principle is used to compute the singleton arc-consistency (SAC) of finite-domain CSPs [3] and the 3B-consistency of numerical CSPs [7]. It is also in the core of the SATZ algorithm [9] proving the satisfiability of boolean formula. Shaving works as follows. A value is temporarily assigned to a variable (the other values are temporarily discarded) and a partial consistency is computed on the remaining subproblem. If an inconsistency is obtained then the value can be safely removed from the domain of the variable. Otherwise, the value is kept in the domain. This principle of refutation has two drawbacks. Contrarily to arc consistency, this consistency is not incremental [2]. Indeed, the work of the underlying refutation algorithm on the *whole* subproblem is the reason why a *single value* can be removed. Thus, obtaining the *singleton arc consistency* on finite-domain CSPs requires an expensive fixed-point propagation algorithm where all the variables must be handled again every time a single value is removed [3]. SAC2 [1] and SAC-optim [2] and other SAC variants obtain better average or worst time complexity by managing heavy data structures for the supports of values (like with AC4) or by duplicating the CSP for every value. However, using these filtering operators inside a backtracking scheme is far from being competitive with the standard MAC algorithm in the current state of research. In its QuickShaving [8], Lhomme uses this shaving principle in a pragmatic way, i.e., with no overhead, by learning the promising variables (i.e., those that can possibly produce gains with shaving in the future) during the search. Researchers and practitioners have also used for a long time the shaving principle in scheduling problems. On numerical CSPs, the 2B-consistency is the refutation algorithm used by 3B-consistency [7]. This property limited to the bounds of intervals explains that 3B-consistency filtering often produces gains in performance.

Example. *Figure 1 (left) shows the first two steps of the 3B-consistency algorithm. Since domains are continuous, shaving does not instantiate a variable to a value but restricts its domain to a sub-interval of fixed size located at one of the endpoints. The subproblems are represented with slices in light gray. The 2B-consistency projects every constraint onto a variable and intersects the result of all projections. In the leftmost slice, 2B-consistency leads to an empty box since the projections of the first and the second constraint onto x_2 are two intervals I_1 and I_2 with empty intersection. On the contrary, the fixed-point of projections in the rightmost slice is a nonempty box (with thick border).*

The second drawback of shaving is that the pruning effort performed by the partial consistency operator to refute a given value is lost, which is not the case with constructive disjunction[1].

Constructive disjunction was proposed by Van Hentenryck et al. in the nineties to handle efficiently disjunctions of constraints, thus tackling a more general model than the standard CSP model [17]. The idea is to propagate independently every term of the disjunction and to perform the union of the different pruned search spaces. In other terms, a value removed by every propagation process (run with one term/constraint of the disjunct) can be safely removed from the ground CSP. This idea is fruitful in several fields such as scheduling, where a common constraint is that two given tasks cannot overlap, or 2D bin packing problems where two rectangles must not overlap.

Constructive disjunction can also be used to handle the classical CSP model (where the problem is viewed as a conjunction of constraints). Indeed, every variable domain can be viewed as a unary disjunctive constraint that imposes one value among the different possible ones ($x = v_1 \vee ... \vee x = v_n$, where x is a variable and $v_1, ..., v_n$ are the different values). In this specific case, similarly to shaving, the constructive disjunction principle can be applied as follows. Every value in the domain of a variable is assigned in turn to this variable (the other values are temporarily discarded), and a partial consistency on the corresponding subproblems is computed. The search space is then replaced by the union of the resulting search spaces. One advantage over shaving is that the

[1] Note that optimized implementations of SAC reuse the domains obtained by subfiltering in subsequent calls to the shaving of a same variable.

(sub)filtering steps performed during constructive disjunction are better reused. This constructive "domain" disjunction is not very much exploited right now while it can sometimes produce impressive gains in performance. In particular, in addition to *all-diff* constraints [14], incorporating constructive domain disjunctions into the famous Sudoku problem (launched, for instance, when the variables/cases have only two remaining possible values/digits) often leads to a backtrack-free solving. The same phenomenon is observed with shaving but at a higher cost [15].

This observation has precisely motivated the research described in this paper devoted to the application of constructive domain disjunction to numerical CSPs. The continuous nature of interval domains is particularly well-suited for constructive domain disjunction. By splitting an interval into several smaller intervals (called *slices*), constructive domain disjunction leads in a straightforward way to the *constructive interval disjunction* (`CID`) filtering operator introduced in this paper.

After useful notations and definitions introduced in Section 2, Sections 3 and 4 describe the *CID* partial consistency and the corresponding filtering operator. A hybrid algorithm mixing shaving and CID is described in Section 5. Section 6 presents a new CID-based splitting strategy. Finally, experiments are presented in Section 7.

2 Definitions

The algorithms presented in this paper aim at solving systems of equations or, more generally, numerical CSPs.

Definition 1. *A* **numerical CSP** *(NCSP)* $P = (X, C, B)$ *contains a set of constraints* C *and a set* X *of* n *variables. Every variable* $x_i \in X$ *can take a real value in the interval* $\mathbf{x_i}$ *and* $\mathbf{B}$ *is the cartesian product (called a* **box***)* $\mathbf{x_1} \times ... \times \mathbf{x_n}$*. A solution of* P *is an assignment of the variables in* X *satisfying all the constraints in* C*.*

Remark 1. Since real numbers cannot be represented in computer architectures, the bounds of an interval $\mathbf{x_i}$ should actually be defined as floating-point numbers.

CID filtering performs a union operation between two boxes.

Definition 2. *Let* B_l *and* B_r *be two boxes corresponding to a same set* V *of variables.*
We call **hull** *of* B_l *and* B_r*, denoted by* `Hull`(B_l, B_r)*, the minimal box including* B_l *and* B_r*.*

To compute a bisection point based on a new (splitting) strategy, we need to calculate the *size* of a box. In this paper, the size of a box is given by its perimeter.

Definition 3. *Let* $B = \mathbf{x_1} \times ... \times \mathbf{x_n}$ *be a box. The* **size** *of* B *is* $\sum_{i=1}^{n}(\overline{\mathbf{x_i}} - \underline{\mathbf{x_i}})$*, where* $\overline{\mathbf{x_i}}$ *and* $\underline{\mathbf{x_i}}$ *are respectively the upper and lower bounds of the interval* $\mathbf{x_i}$*.*

3 CID-consistency

The CID-consistency is a new partial consistency that can be obtained on numerical CSPs. Following the principle given in introduction (i.e., combining interval splitting and constructive disjunction), the CID(2)-consistency can be formally defined as follows (see Figure 1).

Definition 4. (CID(2)-consistency)
Let $P = (X, C, B)$ be an NCSP. Let F be a partial consistency.

Let B_i^l be the sub-box of B in which $\mathbf{x_i}$ is replaced by $[\underline{\mathbf{x_i}}, \check{\mathbf{x_i}}]$ (where $\check{\mathbf{x_i}}$ is the midpoint of $\mathbf{x_i}$). Let B_i^r be the sub-box of B in which $\mathbf{x_i}$ is replaced by $[\check{\mathbf{x_i}}, \overline{\mathbf{x_i}}]$.
A variable x_i in X is CID(2)-consistent w.r.t. P and F if
$B = Hull(F(X, C, B_i^l), F(X, C, B_i^r))$. The NCSP P is CID(2)-consistent if all the variables in X are CID(2)-consistent.

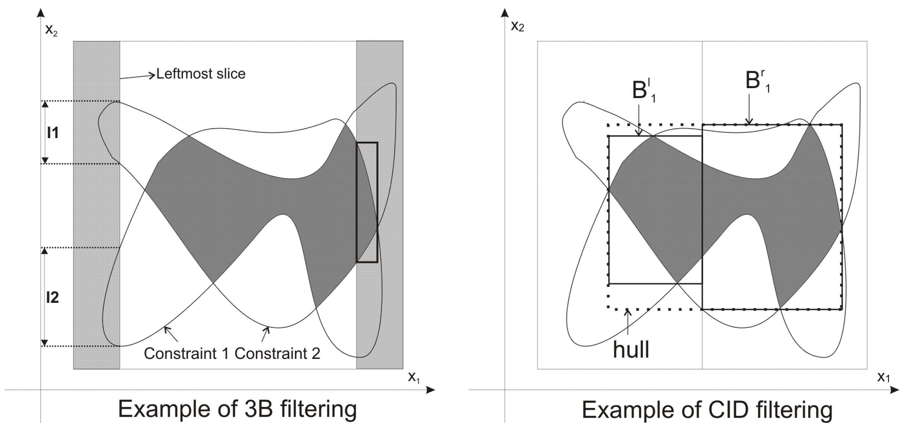

Fig. 1. Shaving and CID-consistency on a simple example with two constraints. The constraints are represented with their intersection in dark gray. **Left:** The first two steps of 3B. **Right:** the result of `VarCID` on variable x_1 (with 2 slices). The subboxes B_1^l and B_1^r are represented with thick borders; the resulting box appears in dotted lines.

For every dimension, the number of slices considered in the CID(2)-consistency is equal to 2. The definition can be generalized to the CID(s)-consistency in which every variable is split into s slices.

In practice, like 3B-w-consistency, the CID-consistency is obtained with a precision that avoids a slow convergence onto the fixed-point. We will consider that a variable is CID(2,w)-consistent if the hull of the corresponding left and right boxes resulting from subfiltering reduces no variable more than w.

Definition 5. (CID(2,w)-consistency)
With the notations of Definition 4, put $B' = Hull(F(X, C, B_i^l), F(X, C, B_i^r))$.

A variable x_i in X is CID(2,w)-consistent if $|\mathbf{x_i}| - |\mathbf{x'_i}| \leq w$, where $\mathbf{x_i}$ and $\mathbf{x'_i}$ are the domain of x_i in resp. B and B'.
The NCSP is CID(2,w)-consistent if all the variables in X are CID(2,w)-consistent.

Algorithm CID details the CID(s,w)-consistency filtering algorithm. Like the 3B-consistency algorithm, CID iterates on all the variables until a stop criterion, depending on w, is reached (see Definition 5): the Repeat loop is interrupted when no variable interval has been reduced more than w [2].

Every variable x_i is "*varcided*", i.e., handled by the procedure VarCID. The domain of x_i is split into s slices of size $\frac{|\mathbf{x_i}|}{s}$ each by the procedure SubBox. The partial consistency operator F (e.g., 2B or a variant of the latter called Box-consistency [16]) filters the corresponding sub-boxes, and the union of the resulting boxes is computed by the Hull operator. Note that if the subfiltering operator F applied to a given sub-box sliceBox detects an inconsistency, then the result sliceBox' is empty, so that there is no use of performing the union of sliceBox' with the current box in construction.

```
Algorithm CID (s: number of slices, w: precision, in-out $P = (X, C, B)$: an NCSP,
$F$: subfiltering operator and its parameters)
    repeat
        $P_{old} \leftarrow P$
        LoopCID ($X$, $s$, $P$, $F$)
    until StopCriterion($w$, $P$, $P_{old}$)
end.
Procedure LoopCID ($X$, $s$, in-out $P$, $F$)
    for every variable $xi \in X$ do
        VarCID ($xi$, $s$, $P$, $F$)
    end
end.
Procedure VarCID ($xi$, $s$, ($X$, $C$, in-out $B$), $F$)
    $B' \leftarrow$ empty box
    for $j \leftarrow 1$ to $s$ do
        sliceBox $\leftarrow$ SubBox ($j$, $s$, $xi$, $B$)   /* the $j^{th}$ sub-box of $B$ on $xi$ */
        sliceBox' $\leftarrow F(X, C,$ sliceBox)  /* perform a partial consistency */
        $B' \leftarrow$ Hull($B'$, sliceBox')          /* Union with previous sub-boxes */
    end
    $B \leftarrow B'$
end.
```

4 A CID-Based Solving Strategy

To find all the solutions of a numerical CSP, we propose a strategy including a bisection operation, CID filtering and an interval Newton [10]. Between two bisections, two operations are run in sequence:

1. a call to CID (how to fix the parameters is discussed just below),
2. a call to an interval Newton operator.

[2] From a theoretical point of view, in order that the stop criterion leads to a (non unique) fixed-point, it is necessary to return the box obtained just before the last filtering, i.e., the box in P_{old} in Algorithm CID.

The Right Set of Parameters for CID Filtering

First of all, the subfiltering operators we chose for `CID` are `2B` and `Box`. The classical user-defined parameter `w-hc4` (percentage of the interval width) is used by the subfiltering operator (`Box` or `2B`) inside CID to control the propagation loop. The parameter s is the number of slices used by the `VarCID` procedure.

Although useful to compute the CID-consistency property, the fixed-point repeat loop used by Algorithm `CID` does not pay off in practice (see below). In other words, running more than once `LoopCID` on all the variables is always counterproductive, even if the value of w is finely tuned. Thus, we have set the parameter w to ∞, that is, we have discarded w from the user-defined parameters.

Endowed with the parameters s and `w-hc4`, CID filtering appears to be an efficient general-purpose operator that has the potential to replace 2B/Box (alone) or 3B. For some (rare) benchmarks however, the use of only 2B/Box appears to be more efficient.

To offer a compromise between pure 2B/Box and `CID`, we introduce a third parameter n' defined as the number of variables that are varcided between two bisections in a round-robin strategy: given a predefined order between variables, a `VarCID` operation is called on n' variables (modulo n), starting at the lattest varcided variable plus one in the order. (This information is transmitted through the different nodes of the tree search.) Thus, if n' is set to 0, then only 2B/Box filtering is called between two bisections; if $n' = n$ (n is the number of variables in the system), then `CID` is called between two bisections.

We shall henceforth consider that `CID` has three parameters: the number of slices `s`, the propagation criterion `w-hc4` of `2B/Box` used for subfiltering and the number `n'` of varcided variables.

Default Values for CID Parameters

So far, we have proposed an efficient general-purpose combination of CID, interval Newton and bisection. However, such a strategy is meaningful only if default values for `CID` are available. In our solver, the default values provide the following CID operator: `CID(s=4, w-hc4=10%, n'=n)`.

Experiments have led us to select $s = 4$ (see Section 7.2):

– The optimal number of slices lies between 2 and 8.
– Selecting $s = 4$ generally leads to the best performance. In the other cases, the performance is not so far from the one obtained with a tuned number of slices.

Discarded Variants

Several variants of CID or combinations of operators have been discarded by our experiments. Mentioning these discarded algorithms may be useful.

First, different combinations of bisection, CID and interval Newton have been tried. The proposed combination is the best one, but adding an interval Newton to the 2B/Box subfiltering (i.e., inside CID filtering) produces interesting results as well.

Second, we recommend to forget the fixed-point parameter w in `CID`. Indeed, a lot of experiments confirmed that relaunching `LoopCID` several times in a row between two bisections is nearly *never* useful. The same phenomenon has been observed if the relaunch criterion concerns several dimensions, i.e., if the reduction of box size (perimeter or volume) is sufficiently large. A third experiment running `LoopCID` twice between two splits leads to the same conclusion. Finally, the same kind of experiments have been conducted with no more success to determine which specific variables should be varcided again in an adaptive way. All these experiments give us the strong belief that one `LoopCID`, i.e., varciding all the variables once, is rather a *maximal* power of filtering.

Third, in a previous workshop version of this paper, we had proposed a `CID246` variant of CID, in which the number s of slices was modified between two splits: it was alternatively 2, 4 or 6 : $s = ((i \, modulo \, 3) + 1) \times 2)$, where i indicates the i^{th} call to `LoopCID`. The comparison with the standard CID was not fair because the parameter s in CID was set to 2. It turns out that CID with $s = 4$ yields nearly the same results as `CID246` while being simpler.

Finally, we also investigated the option of a reentrant algorithm, under the acronym k-CID. As k-B-consistency [7] generalizes the 3-B-consistency, it is possible to use $(k-1)$`-CID` as subfiltering operator inside a `k-CID` algorithm. Like 4-B-consistency, 2-CID-consistency remains a theoretical partial consistency that is not really useful in practice. Indeed, the pruning power is significant (hence a small number of required splits), but the computational time required to obtain the solutions with `2-CID` plus bisection is often not competitive with the time required by `1-CID` plus bisection.

5 `3B`, `CID` and a `3BCID` Hybrid Version

As mentioned in the introduction, the CID partial consistency has several points in common with the well-known 3B-consistency partial consistency [7].

Definition 6. (3B(s)-consistency)
Let $P = (X, C, B)$ be an NCSP. Let B_i^l be the sub-box of B in which $\mathbf{x_i}$ is replaced by $[\underline{\mathbf{x_i}}, \underline{\mathbf{x_i}} + \frac{|\mathbf{x_i}|}{s}]$. Let B_i^r be the sub-box of B in which $\mathbf{x_i}$ is replaced by $[\overline{\mathbf{x_i}} - \frac{|\mathbf{x_i}|}{s}, \overline{\mathbf{x_i}}]$.

Variable x_i is 3B(s)-consistent w.r.t. P if $2B(X, C, B_i^l) \neq \emptyset$ and $2B(X, C, B_i^r) \neq \emptyset$. The NCSP P is 3B(s)-consistent if all the variables in X are 3B(s)-consistent.

For practical considerations, and contrarily to finite-domain CSPs, a partial consistency of an NCSP is generally obtained with a precision w [7]. This precision avoids a slow convergence to obtain the property. Hence, as in `CID`, a parameter w is also required in the outer loop of `3B`.

When the subfiltering operator is performed by *Box consistency*, instead of 2B-consistency, we obtain the so-called *Bound consistency* property [16].

The `3B` algorithm follows a principle similar to `CID`, in which `VarCID` is replaced by a shaving process, called `VarShaving` in this paper. In particular, both algorithms are not incremental, hence the outside repeat loop possibly reruns the treatment of all the variables, as shown in Algorithm 3B.

```
Algorithm 3B (w: stop criterion precision, s : shaving precision, in-out
P = (X, C, B): an NCSP, F: subfiltering operator and its parameters)
    repeat
        for every variable xi ∈ X do
            VarShaving (xi, s, P, F)
        end
    until StopCriterion(w, P)
end.
```

The procedure `VarShaving` reduces the left and right bounds of variable x_i by trying to refute intervals with a width at least equal to $\frac{|x_i|}{s}$. The following proposition highlights the difference between 3B filtering and CID filtering.

Proposition 1. *Let $P = (X, C, B)$ be an NCSP. Consider the box B' obtained by* `CID(s,w)` *w.r.t.* 2B *and the box B'' obtained by* `3B(s,w)` [3]. *Then,* `CID(s,w)` *filtering is stronger than* `3B(s,w)` *filtering, i.e., B' is included in or equal to B''.*

This theoretical property is based on the fact that, due to the hull operation in `VarCID`, the whole box B can be reduced on several, possibly all, dimensions. With `VarShaving`, the pruning effort can impact only x_i, losing all the temporary reductions obtained on the other variables by the different calls to F.

Proposition 1 states that the pruning capacity of CID is greater than the one of 3B. In the general case however, 3B-consistency and CID-consistency are not comparable because s is the exact number of calls to subfiltering F inside `VarCID` (i.e., the upper bound is reached), while s is an upper bound of the number of calls to 2B by `VarShaving`. Experiments will confirm that a rough work on a given variable with CID (e.g., setting $s = 4$) yields better results than a more costly work with 3B (e.g., setting $s = 10$).

The 3BCID Filtering Algorithm

As mentioned above, the 3B and CID filtering operators follow the same scheme, so that several hybrid algorithms have been imagined. The most promising version, called 3BCID, is presented in this paper.

3BCID manages two parameters: the numbers $s_{\mathtt{CID}}$ and $s_{\mathtt{3B}}$ of slices for the CID part and the shaving part. Every variable x_i is handled by a shaving and a `VarCID` process as follows.

The interval of x_i is first split into $s_{\mathtt{3B}}$ slices handled by shaving. Using a subfiltering operator F, a simple shaving procedure tries to refute these slices to

[3] An additional assumption related to floating-point numbers and to the fixed-point criterion is required in theory to allow a fair comparison between algorithms: the 2B/Box subfiltering operator must work with a subdivision of the slices managed by 3B and CID.

the left and to the right (no dichotomic process is performed). Let s_{left} (resp. s_{right}) be the leftmost (resp. rightmost) slice of x_i that has not been refuted by subfiltering, if any. Let $\mathbf{x_i'}$ be the remaining interval of x_i, i.e., $\overline{s_{left}} \leq \underline{\mathbf{x_i'}} \leq \overline{\mathbf{x_i'}} \leq \underline{s_{right}}$.

Then, if $\mathbf{x_i'}$ is not empty, it is split into $s_{\mathtt{CID}}$ slices and handled by `CID`. One performs the hull of the (at most) $s_{\mathtt{CID}} + 2$ boxes handled by the subfiltering operator F: s_{left}, s_{right} and the $s_{\mathtt{CID}}$ slices between s_{left} and s_{right}.

It is straightforward to prove that the obtained partial consistency is stronger than 3B($s_{\mathtt{3B}}$)-consistency.

The experiments will show that `3BCID` with $s_{\mathtt{CID}} = 1$ can be viewed as an improved version of `3B` where constructive disjunction produces an additional pruning effect with a low overhead.

6 A New CID-Based Splitting Strategy

There are three main splitting strategies (i.e., variable choice heuristics) used for solving numerical CSPs. The simplest one follows a *round-robin* strategy and loops on all the variables. Another heuristic selects the variable with the largest interval. A third one, based on the *smear function* [10], selects a variable x_i implied in equations whose derivative w.r.t. x_i is large.

The round-robin strategy ensures that all the variables are split in a branch of the search tree. Indeed, as opposed to finite-domain CSPs, note that a variable interval is generally split (i.e., instantiated) several times before finding a solution (i.e., obtaining a small interval of width less than the precision). The largest interval strategy also leads the solving process to not always select a same variable as long as its domain size decreases. The strategy based on the smear function sometimes splits always the same variables so that an interleaved schema with round-robin, or a preconditionning phase, is sometimes necessary to make it effective in practice.

We introduce in this section a new CID-based splitting strategy. Let us first consider different box sizes related to (and learnt during) the `VarCID` procedure applied to a given variable x_i :

– Let $\mathtt{OldBox}_i$ be the box B just before the call to `VarCID` on x_i. Let $\mathtt{NewBox}_i$ be the box obtained after the call to `VarCID` on x_i.
– Let $B_i^{l'}$ and $B_i^{r'}$ be the left and right boxes computed in `VarCID`, after a reduction by the F filtering operator, and before the `Hull` operation.

The ratio `ratioBis` leads to an "intelligent" splitting strategy. The ratio `ratioBis`$= \frac{f(Size(B_i^{l'}),Size(B_i^{r'}))}{Size(NewBox)}$, where f is any function that aggregates the size of two boxes (e.g., sum), in a sense computes the size lost by the `Hull` operation of `VarCID`. In other words, $B_i^{l'}$ and $B_i^{r'}$ represent precisely the boxes one would obtain if one splits the variable x_i (instead of performing the hulloperation) immediately after the call to `VarCID`; `NewBox` is the box obtained by the `Hull` operation used by `CID` to avoid a combinatorial explosion due to a choice point.

Thus, after a call to `LoopCID`, the CID principle allows us to learn about a good variable interval to be split: *one selects the variable having led to the lowest* `ratioBis`. Although not related to constructive disjunction, similar strategies have been applied to finite-domain CSPs [4,13].

Experiments, not reported here, have compared a large number of variants of `ratioBis` with different functions f. The best variant is `ratioBis` $= \frac{Size(B_i^{l'})+Size(B_i^{r'})}{Size(NewBox)}$.

7 Experiments

We have performed a lot of comparisons and tests on a sample of 20 instances. These tests have helped us to design efficient variants of CID filtering.

7.1 Benchmarks and Interval-Based Solver

Twenty benchmarks are briefly presented in this section. Five of them are sparse systems found in [11]: `Hourglass`, `Tetra`, `Tangent`, `Ponts`, `Mechanism`. They are challenging for general-purpose interval-based techniques, but the algorithm `IBB` can efficiently exploit a preliminary decomposition of the systems into small subsystems [11]. The other benchmarks have been found in the Web page of the COPRIN research team or in the COCONUT Web page where the reader can find more details about them [12]. The precision of the solutions. i.e., the size of interval under which a variable interval is not split, is $1e-08$ for all the benchmarks, and $5e-06$ for `Mechanism`. `2B` is used for all the benchmarks but one because it is the most efficient local consistency filtering when used alone inside `3B` or `CID`. `Box+2B` is more adequate for `Yamamura8`. All the selected instances can be solved in an acceptable amount of time by a standard algorithm in order to make possible comparisons between numerous variants. No selected benchmark has been discarded for any other reason!

All the tests have been performed on a `Pentium IV 2.66 Ghz` using the interval-based library in `C++` developed by the second author. This new solver provides the main standard interval operators such as `Box`, `2B`, interval Newton [10]. The solver provides round-robin, largest-interval and CID-based splitting strategies. Although recent and under developement, the library seems competitive with up-to-date solvers like `RealPaver` [5]. For all the presented solving techniques, including `3B` and `3BCID`, an interval Newton is called just before a splitting operation iff the width of the largest variable interval is less than $1e-2$.

7.2 Results Obtained by `CID`

Table 1 reports the results obtained by `CID`(s, `w-hc4`, n'), as defined in Section 4.

The drastic reduction in the number of required bisections (often several orders of magnitude) clearly underlines the filtering power of `CID`. In addition,

Table 1. Comparison between [`CID` + interval Newton + round-robin bisection strategy] and [a standard strategy]: 2B/Box + interval Newton + round-robin splitting. n is the number of variables. The column #s yields the number of solutions. The first column `w-hc4` is the user-defined parameter `w-hc4` used by `2B` or `Box`. The last 3 columns s, `w-hc4` and n' indicate the values of parameters that have been tuned for `CID` (first `CID` column). The second `CID` column reports the results of `CID` when $s = 4$ and `w-hc4`$= 10\%$, i.e., when only n' is tuned. The third `CID` column reports the results of `CID` when $s = 4$ and $n' = n$, i.e., when only `w-hc4` is tuned. The fourth `CID` column reports the results of `CID` with the default values for parameters, i.e., $s = 4$, `w-hc4`$= 10\%$, $n' = n$. Every cell contains two values: the CPU time in seconds to compute all the solutions (top), and the number of required bisections (bottom). For every benchmark, the best CPU time is bold-faced.

Name	n	#s	`w-hc4`	`2B/Box` `+ Newton`	`CID` $(s,\texttt{whc4},n')$	`CID` $(4,10\%,n')$	`CID` $(4,\texttt{whc4},n)$	`CID` $(4,10\%,n)$	s	`w-hc4`	n'
`BroydenTri`	32	2	15%	758	**0.12**	0.28	0.19	0.45	4	80%	40
				2e+07	46	44	65	50			
`Hourglass`	29	8	5%	24	**0.44**	0.44	0.52	0.52	4	10%	17
				1e+05	109	109	80	80			
`Tetra`	30	256	0.02%	401	**10.1**	11.6	11.7	14.5	4	30%	20
				1e+06	2116	1558	1690	1320			
`Tangent`	28	128	15%	32	**3.7**	3.7	4.9	5.1	4	50%	28
				1e+05	692	692	447	450			
`Reactors`	20	38	5%	156	**15.6**	16.4	16.7	17.7	4	15%	18
				1e+06	2588	2803	2381	2156			
`Trigexp1`	30	1	20%	371(3.4)	**0.12**	0.15	0.14	0.14	8	2%	30
				5025	1	3	2	3			
`Discrete25`	27	1	0.01%	5.2	**0.62**	1.08	0.84	2.13	8	0.5%	35
				1741	2	3	12	99			
`I5`	10	30	2%	692	**126**	147	150	157	6	2%	5
				3e+06	23105	60800	20874	32309			
`Transistor`	12	1	10%	179	**66**	79.4	91.4	91.4	8	10%	6
				1e+06	11008	31426	16333	16333			
`Ponts`	30	128	5%	10.8	**2.7**	2.9	2.9	3.1	4	30%	25
				34994	388	338	380	304			
`Yamamura8`	8	7	1%	13	**7.5**	9.5	9.5	9.5	4	10%	4
				1032	104	60	60	60			
`Design`	9	1	10%	395	**275**	278	313	313	5	10%	2
				3e+06	200272	256000	76633	76633			
`D1`	12	16	5%	4.1	**1.7**	1.7	1.7	1.7	4	10%	12
				35670	464	464	464	464			
`Mechanism`	98	448	0.5%	TO(111)	**43.1**	45.2	46.6	47.8	4	2%	50
				24538	3419	3300	2100	2420			
`Hayes`	8	1	0.01%	155	**75.8**	77	111	147	4	80%	2
				3e+05	1e+05	1e+05	81750	58234			
`Kin1`	6	16	10%	84	**76.8**	76.8	83.5	87.4	4	10%	3
				70368	6892	6892	4837	4100			
`Eco9`	8	16	10%	26	**18**	19.4	26.6	26.6	3	10%	1
				2e+05	55657	46902	10064	10064			
`Bellido`	9	8	10%	**80**	94.4	94.4	106	106	4	10%	3
				7e+05	1e+05	1e+05	45377	45377			
`Trigexp2-9`	9	1	20%	61.8	**50.4**	65.14	62.4	68.4	6	10%	9
				3e+05	4887	14528	11574	9541			
`Trigexp2-5`	5	1	20%	**3.0**	3.8	4.6	6.2	6.6	2	20%	1
				13614	10221	4631	2293	1887			
`Caprasse`	4	18	30%	**2.6**	2.73	3.0	4.7	5.1	2	5%	1
				37788	18176	12052	5308	5624			

impressive gains in running time are obtained by CID for the benchmarks on the top of the table, as compared to standard strategy using 2B or 2B+Box, an interval Newton and a round-robin splitting policy[4].

CID often obtains better running times than the standard strategy, except on Bellido, Trigexp2-5 and Caprasse, for which the loss in performance is small. However, it is not reasonable to propose to the user an operator for which three parameters must be tuned by hand. That is why we have also reported the last three CID columns where only 0 or 1 parameter has been tuned. The default values (fourth CID column with $s = 4$, w-hc4= 10%, $n' = n$) yield very good results: it outperforms the standard strategy in 15 of the 21 instances. In particular, setting $s = 4$ provides the best results in 12 of the 21 instances.

The second and third CID columns report very good results obtained by a filtering algorithm with only one parameter to be tuned (like with 2B or Box). Hence, since CID with only parameter n' to be tuned (second CID column) allows a continuum between pure 2B and CID ($n' = 0$ amounts to a call to 2B), a first recommendation is to propose this CID variant in interval-based solvers.

The second recommendation comes from a combinatorial consideration. In a sense, constructive interval disjunction can be viewed as a "polynomial-time splitting" since VarCID performs a polynomial-time hull after having handled the different slices. However, the exponential aspect of (classical) bisection becomes time-consuming only when the number of variables becomes high. This would explain why CID cannot pay off on Trigexp2-5 and Caprasse which have a very small number of variables and lead to no combinatorial explosion due to bisection. (The intuition is confirmed by the better behavior of CID on the (scalable) variant of Trigexp2 with 9 variables.) Thus, the second recommendation would be to use CID on systems having a minimal number of variables, e.g., 5 or 8.

7.3 Comparing CID, 3B and 3BCID

Table 2 reports the results obtained by CID, 3B and 3BCID (see Section 5). All the results have been obtained with a parameter w-hc4 set to 5%.

Several trials have been performed for every algorithm, and the best result is reported in the table. For 3B, seven values have been tried for the parameter $s_{\texttt{3B}}$: 4, 5, 7, 10, 20, 50, 100.

For CID, nine values of the parameters have been tried: five values for the number of slices $s_{\texttt{CID}}$ (2, 3, 4, 6, 8; fixing $n' = n$), and four values for the parameter n' (1, $0.5n$, $0.75n$, $1.2n$; fixing $s_{\texttt{CID}} = 4$). For 3BCID, eight combinations of parameters have been tried: four values for $s_{\texttt{CID}}$ (1,2,3,4) combined with two values for $s_{\texttt{3B}}$ (10, 20).

The main conclusions drawn from Table 2 are the following:

- 3BCID and CID always outperform 3B. Even the standard strategy outperforms 3B for 9 benchmarks in the bottom of the table.

[4] Note that this strategy is inefficient for Trigexp1 (solved in 371 seconds) and for Mechanism (that is not solved after a timeout (TO) of several hours). However, a reasonable running time can be obtained with a variant of 2B that push *all* the constraints of the NCSP in the propagation queue after every bisection.

Table 2. Comparison between CID, 3B and 3BCID. The first three columns recall resp. the name of the benchmark, its number of variables and the CPU time in seconds required by the strategy 2B/Box + Newton + bisection with round-robin. The other columns report the CPU time required to solve the benchmarks with CID, 3B and 3BCID. The best CPU time result is bold-faced.

Name	n	2B/Box	CID	3B	3BCID($s_{\texttt{CID}} = 1$)	3BCID($s_{\texttt{CID}} = 2$)
BroydenTri	32	758	0.23	0.22	**0.18**	0.19
Hourglass	29	24	0.45	0.73	**0.43**	0.50
Tetra	30	401	**13.6**	20.7	17.1	18.8
Tangent	28	32	4.13	8.67	**3.18**	4.13
Reactors	20	156	18.2	24.2	**15.5**	16.9
Trigexp1	30	3.4	**0.10**	0.26	0.12	0.11
Discrete25	27	5.2	1.37	2.19	1.26	**1.13**
I5	10	692	139	144	**115**	123
Transistor	12	179	71.5	77.9	49.3	**46.9**
Ponts	30	10.8	**3.07**	5.75	4.19	4.43
Yamamura8	8	13	**9.0**	9.1	10.3	10.7
Design	9	395	300	403	**228**	256
D1	12	4.1	1.78	2.99	**1.64**	1.76
Mechanism	98	111	**79**	185	176	173
Hayes	8	155	**99**	188	102	110
Kinematics1	6	84	**76.1**	136	76.6	81.4
Eco9	8	26	**19.3**	40.1	27.0	30.3
Bellido	9	**80**	95	143	**93**	102
Trigexp2-9	9	61.8	52.2	74.5	**39.9**	45.1
Caprasse	4	**2.6**	**3.1**	9.38	4.84	5.35

– 3BCID is competitive with CID. 3BCID is better than CID concerning 11 benchmarks. CID remains even better for Mechanism. We wonder if it is related to its large number of variables.

The good news is that the best value for $s_{\texttt{CID}}$ in 3BCID is often $s_{\texttt{CID}} = 1$. In only four cases, the best value is greater, but the value $s_{\texttt{CID}} = 1$ also provides very good results. This suggests to propose 3BCID with $s_{\texttt{CID}} = 1$ as an alternative of 3B. In other words, 3BCID with $s_{\texttt{CID}} = 1$ can be viewed as a promising implementation of 3B. This is transparent for a user who has to specify the same parameter $s_{\texttt{3B}}$, the management of constructive disjunction being hidden inside 3BCID (that performs a Hull operation of at most 3 boxes since $s_{\texttt{CID}} = 1$).

7.4 Comparing Splitting Strategies

Table 3 applies the three available splitting strategies to CID with $n' = n$ and $s = 4$. We underline some observations.

The new CID-based splitting strategy is better than the other strategies on 13 of the 20 instances, especially on Design. The largest interval strategy is the best on only one instance. The round-robin strategy is the best on 6 instances.

Table 3. Comparison on CID with three splitting strategies: Round-robin, Largest interval and the new CID-based strategy.

Filtering	CID	CID	CID
Splitting	Round-robin	Largest Int.	CID-based
BroydenTri	0.21	0.18	**0.17**
Hourglass	0.52	0.51	**0.37**
Tetra	**12.1**	28.2	16.4
Tangent	**3.7**	21.7	5.2
Reactors	17.0	13.2	**12.7**
Trigexp1	0.15	0.19	**0.14**
Discrete25	**0.84**	1.49	1.06
I5	**151**	421	179
Transistor	93	**36**	41
Ponts	2.92	5.51	**2.31**
Yamamura8	9.5	6.9	**5.1**
Design	318	334	**178**
D1	**1.72**	2.96	2.50
Mechanism	47	49	**46**
Hayes	**115**	564	318
Kinematics1	83	70	**63**
Eco9	26.7	31.4	**26.1**
Bellido	107	102	**99**
Trigexp2-9	62	55	**53**
Caprasse	5.16	5.43	**5.04**

On the 7 instances for which the CID-based strategy is not the best, the loss in performance is significant on Hayes.

The behavior of the CID-based strategy with $s = 6$ (not reported here) is even better, the round-robin strategy being the best on only 3 instances. This would suggest that the ratioCID learned during a VarCID operation is more accurate with a higher number of slices.

8 Conclusion

This paper has introduced two new filtering operators based on the constructive disjunction principle exploited in combinatorial problems. The first experimental results are very promising and we believe that CID and 3BCID have the potential to become standard operators in interval constraint solvers. The CID operator also opens the door to a new splitting strategy learning from the work of CID filtering.

The experiments lead to clear recommendations concerning the use of these new filtering operators. First, CID can be used with fixed values of parameters s and w-hc4, letting the user only tune the third parameter n' (i.e., the number of variables that are varcided between 2 bisections). This allows the user to select in a sense a rate of CID filtering, $n' = 0$ producing the pure 2B/Box. Used this way, CID could maybe subsume existing filtering operators. Second, 3BCID with $s = 1$ can be provided as a promising alternative of a 3B operator.

Several questions remain open. It seems that, due to combinatorial considerations, CID is not convenient for small problems while it seems more interesting for large-scale systems. A more complete experimental study should confirm or contradict this claim. Also, 3BCID should be compared to the weak-3B operator implemented in RealPaver [5][5]. A comparison with filtering algorithms based on linearization, like Quad [6], will be performed as well.

Moreover, the CID-based splitting strategy merits a deeper experimental study. In particular, a comparison with the *smear* function will be performed once the latter is implemented in our solver.

An interesting future work is to propose *adaptive* variants of CID that can choose which specific variable should be varcided or bisected next.

Acknowledgements

Special thanks to Olivier Lhomme for useful discussions about this research. Also thanks to Bertrand Neveu, Arnold Neumaier and the anonymous reviewers.

References

1. Barták, R., Erben, R.: A new Algorithm for Singleton Arc Consistency. In: Proc. FLAIRS (2004)
2. Bessière, C., Debruyne, R.: Optimal and Suboptimal Singleton Arc Consistency Algorithms. In: Proc. IJCAI, pp. 54–59 (2005)
3. Debruyne, R., Bessière, C.: Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In: Proc. IJCAI, pp. 412–417 (1997)
4. Geelen, P.A.: Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In: Proc. ECAI'92, pp. 31–35 (1992)
5. Granvilliers, L., Benhamou, F.: RealPaver: An Interval Solver using Constraint Satisfaction Techniques. ACM Trans. on Mathematical Software 32(1), 138–156 (2006)
6. Lebbah, Y., Michel, C., Rueher, M.: A Rigorous Global Filtering Algorithm for Quadratic Constraints. Constraints Journal 10(1), 47–65 (2005)
7. Lhomme, O.: Consistency Tech. for Numeric CSPs. In: IJCAI, pp. 232–238 (1993)
8. Lhomme, O.: Quick Shaving. In: Proc. AAAI, pp. 411–415 (2005)
9. Min Li, C., Anbulagan: Heuristics Based on Unit Propagation for Satisfiability Problems. In: Proc. IJCAI, pp. 366–371 (1997)
10. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, Cambridge (1990)
11. Neveu, B., Chabert, G., Trombettoni, G.: When Interval Analysis helps Interblock Backtracking. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 390–405. Springer, Heidelberg (2006)
12. Web page of COPRIN: www-sop.inria.fr/coprin/logiciels/ALIAS/Benches/benches.html
    COCONUT benchs:
    www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html

[5] An implementation of both operators in a same solver would lead to a fair comparison.

13. Refalo, P.: Impact-Based Search Strategies for Constraint Programming. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 557–571. Springer, Heidelberg (2004)
14. Régin, J.C.: A Filtering Algorithm for Constraints of Difference in CSPs. In: Proc. AAAI, pp. 362–367 (1994)
15. Simonis, H.: Sudoku as a Constraint Problem. In: CP Workshop on Modeling and Reformulating Constraint Satisfaction Problems, pp. 13–27 (2005)
16. Van Hentenryck, P., Michel, L., Deville, Y.: Numerica: A Modeling Language for Global Optimization. MIT Press, Cambridge (1997)
17. Van Hentenryck, P., Saraswat, V., Deville, Y.: Design, Implementation, and Evaluation of the Constraint Language CC(FD). J. Logic Programming 37(1–3), 139–164 (1994)