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.