The Branch & Move algorithm: Improving Global Constraints Support by Local Search

Thierry Benoist

Bouygues e-lab, 1 av. Eugène Freyssinet,
78061 St Quentin en Yvelines Cedex, France
tbenoist@bouygues.com

Abstract. Most global constraints maintain a support for their filtering algorithm, namely a tuple consistent with both the constraint and current domains. However, this highly valuable information is rarely used outside of the constraint. In this paper we propose a generic hybridization scheme that we tested on a real-world application in the field of TV advertisement. The principle of this *Branch and Move* approach is to use the support of the main global constraint of the problem as a guide for the branching strategy. The accuracy of this oracle is enhanced by local search improvements of this support tuple at each node of the search tree.

1. Introduction

Constraint propagation is based on *infeasible* values. The filtering algorithm of a constraint aims at removing infeasible values from the domain of its variables, i.e. values belonging to no tuple of the relation consistent with current domains. When all such inconsistent values are detected, the algorithm is said to be complete. In the past few years, in order to perform more accurate filtering on rich n-ary constraints, several global filtering algorithms have been developed usually based on OR polynomial algorithms. Most of these global constraints maintain a *feasible* tuple consistent with current domains of variables. When no such support tuple exists, the constraint fails (detects a contradiction), otherwise it is used by the filtering algorithm. For instance the reference matching of the *AllDifferent* constraints (Régin 1994) ensure the feasibility of the constraint, and the strongly connected component decomposition based on this matching offers complete filtering. As more and more CP models are centred on a few number of global constraints (often one), the central role played by global constraints points out the relevancy of considering their support. For instance constrained TSP, constrained Knapsack problems and constrained flow problems can be respectively modelled with (in addition to problem specific constraints) TSP con-

straints (Beldiceanu and E. Contejean 1994), Knapsack constraints (Trick 2001, T. Fahle, M. Sellmann 2002) or flow constraints (Bockmayr et al. 2001, Benoist et al. 2002).

For such problems where a principal global constraint can be identified, we propose a support-guided branching scheme and an associated cooperation between Constraint Programming and Local Search. We name this procedure *Branch and Move*. and expose it in section 3, after the introduction of some definitions in section 2.

2. Definitions

Given K an ordered set and $D = D_1 \times D_2 \times \dots D_n$ with $D_i \subseteq K$ for all $i \in [1,n]$ (domains of variables), we define the following objects:

1. **Constraint[1]:** a *constraint* R is a relation on K^n ($R \subseteq K^n$)**.**
2. **Support set:** the *support set* of relation R on D is $supp(R,D) = R \cap D$. Omitting D, we will note the support set of a constraint R as $supp(R)=supp(R,D)$ with D equal to current domains.
3. **Support tuple:** a support of R is a tuple $x \hat{I} supp(R)$
4. **Constraint Satisfaction Problem:** A constraint satisfaction problem on K is a triplet (n,D,P) where P is a collection of constraints $P=\{R_1,R_{2\dots}R_m\}$ on K^n.
5. **Solutions:** Solutions of P are $sol(P)= R_1 \cap R_2 \cap .. R_m \cap D$.
6. **Potential:** With d_K a distance on K, we define the potential $\Delta(R,x)$ of constraint R with respect to tuple $x \hat{I} K^n$ as the L_1 distance from x to the current support set of R.

$$\Delta(R,x) = \min_{y \in supp(R)} \sum_{i \le n} d_K(x_i, y_i) \qquad (1)$$

   Note that $\Delta(R,x)$ equals 0 if $x \hat{I} supp(R)$ and $+\infty$ if $supp(R)=\varnothing$. This potential literally measures the distance to feasibility of a tuple x for constraint R. For a binary constraint R involving two variables of domains D_1 and D_2, $\Delta(R,x)$ can be computed by enumeration in complexity $O(|D_1| \times |D_2|)$. Moreover, the potential of a linear constraint like $\sum X_i \ge X_0$, merely equals $max(0, x_0 - \sum x_i)$ (slack variable). When a exact computation of $\Delta(R,x)$ would be too costly, the distance to a feasible tuple of R ("empirically close") is an upper bound of $\Delta(R,x)$.
7. **Corrective decision:** A corrective decision for a conflicting pair R,x (a pair with $\Delta(R,x)>0$) is a constraint A such that $x \notin A$ and $A \cap supp(R,D) \neq \varnothing$. In other words it is a constraint discarding x whilst consistent with R. A non-empty support for R is sufficient to ensure its existence.
8. **Neighbourhood:** A neighbourhood structure for R is a function $N_{R,D}$ associating to each support tuple x of R a subset $N_{R,D}(x) \subset supp(R,D)$ such that $x \in N_{R,D}(x)$.
9. **Selection:** A function *move* defined on $supp(R,D)$ is a selection function for neighbourhood $N_{R,D}$ if and only if $move(x) \in N_{R,D}(x)$.
10. **Improvement:** Given a strict order $<$ on K^n, *move* is an *improving* selection function if and only if $\forall x$, $move(x) = x \vee move(x) < x$. For instance the *potential order* $<_P$

[1] Without loss of generality we extend constraints of smaller arity to relations on K^n.

defined by $x <_P y \Leftrightarrow \sum_{R\in P}\Delta(R,x) < \sum_{R\in P}\Delta(R,y)$ is a strict order on K^n. For optimization problems a second criteria based on the objective function can be added.

11. **Descent:** A descent from $x \in supp(R,D)$ is the iteration of an improving selection function *move* until a fix point is reached[2]:

    descent(x) → **while** (*move*(x) ≠ x) x := *move*(x), **return** x

Example with relations on $K^n=\{0,1,2\}^2$:

- $D_1=D_2=\{0,1\}$, $R=\{(0,0),(0,1),(1,0),(2,2)\}$,.
- The *support* set of R is $supp(R,D)=R\cap(D_1\times D_2) = \{(0,0),(0,1),(1,0)\}$
- (0,1) is a *support* of R (among the three possible ones).
- With $R'=\{(0,0),(1,1),(2,2)\}$ a second relation on $\{0,1,2\}^2$, solutions of problem $P=\{R,R'\}$ are $sol(P)= R\cap R'\cap(D_1\times D_2)=\{(0,0)\}$.
- The potential of R with respect to tuple (1,1) is $\Delta(R,(1,1))=1$ because the closest support is $(0,1)\in supp(R,D)$ and $\delta((0,1),(1,1))=1$.
- A possible corrective decision for pair R,(1,1) is $A=\{(0,x)\ \forall\ x\}$ since $(1,1)\notin A$ and $A\cap supp(R,D)=\{(0,0),(0,1)\}\neq\emptyset$.
- A possible neighbourhood for R is $N_{R,D}$: $(x,y) \rightarrow \{(x,0),(0,y),(x,y)\}$.
- For this simple neighbourhood, *move:* $(x,y) \rightarrow (0,0)$ is a possible selection function since (0,0) belongs to the neighbourhood of any tuple of $supp(R,D)$.
- This selection function is strictly improving with respect to the potential order $<_P$, since (0,0) is the only solution of P.

3. Branching strategy and local search improvements

Let R_0 be the main constraint of a problem P and x an element of its support. If the potential of x with respect to all other constraints is null ($\forall R\in P, \Delta(R,x)=0$) then x is a solution of P. Otherwise we decide to select among all additional constraints the one with highest potential and to branch on an associated corrective decision[3]. More precisely we select R maximizing $\Delta(R,x)$ and A a corrective decision for R,x. Then we successively consider sub problems $P\cup A$ and $P\cup A$, with $A=K^n-A$ (complementary decision). In order to make this branching strategy more efficient, the global appropriateness of the support tuple x is improved by local search before each choice: according to a neighbourhood structure suitable for constraint R_0, a descent procedure is applied on x. The resulting *Branch and Move* algorithm reads as follows:

[2] such a fix point is necessarily reached since K^n is finite and $<_P$ is a strict order.

[3] This approach is directly inspired from what can be done in a MIP branch and bound when a continuous solution is found: a violated integrality constraint is selected, and a decision is taken to bring the LP solver to modify this best continuous solution accordingly.

```
solve(P) →                                   // returns true if P is feasible, false otherwise
  if not (propagate(P)) return false,        // constraints propagation, returns false in case of inconsistency
  else
    let x := descent(getSupport(R0)),        // improvement of the support tuple of R0 by local search
        Rmax := argmax{Δ(R,x), R∈ P}  in     // selection of the most unhappy constraint w.r.t. x
      (if (Δ(Rmax,x) = 0) return true        // if x satisfies all constraints it is a solution of P
       else
         let A = selectCorrective(Rmax,x) in // selection of a corrective decision for Rmax
           return (solve(P∪A) or solve(P∪A)) // recursive exploration
```

Fig. 1. The Branch and Move algorithm

It is important to note that the improvement of the support by local search is completely non-destructive: it has no impact at all on current domains since only the support tuple is modified. Instead of jumping to another node of equal depth, modifying already instantiated variables as in *Incremental Local Optimization* [Caseau & Laburthe 1999], we modify the values taken by *remaining* variables in the *support tuple*. This absence of impact on current domains keeps the search tree complete whilst the embedded problem-specific local search algorithm avoids wasting time in a costly sub tree searching for a solution laying just a few moves far from current support tuple, or trying the ensure the satisfaction of constraints that are consistent with a nearby support tuple. The latter case points out that the improvement process helps identifying critical points: constraints remaining in conflict with the obtained support ("resisting to local moves") may be the most critical ones. The *Branch and Move* algorithm can be compared to the *Branch and Greed* [Sourd & Chrétienne 1999] and *Local Probing* techniques [Kamarainen & El Sakkout 2002] that are also based on heuristic computations at each node, searching for possible extension of the current partial assignement. From a more general point of view these algorithms belong to the family of CP/LS hybrids [Focacci et al 2003], [Focacci & Shaw 2002].

From Local Search point of view, the whole process can be seen as the systematic exploration of the support set of the main constraint. In this context, the initial improvement process is a standard greedy descent starting from the support tuple. Now the local optimum escaping strategy significantly differs from tabu or simulated annealing paradigms. Instead of changing the solution by non-improving moves, the landscape itself is modified thanks to a corrective decision. The propagation of this decision removes many tuples from the landscape, including the current one. The filtering algorithm computes a new support tuple close to the discarded one. Then a new greedy descent is applied from this new starting point to reach the closest local optimum in this *new filtered landscape*. Since these decisions are embedded in a CP search tree, a complete enumeration of possible landscapes is performed yielding to an *exact* local search algorithm. When such a complete enumeration would be too costly, the CP framework gives us the opportunity to use well-tried partial tree search techniques developed in the last decade. Restricted Candidate Lists, discrepancy based search, credit search, barrier limit, etc… are likely to guide the enumeration through a diversified set of promising landscapes [Focacci et al 2003].

For a more comprehensive description of *Branch and Move* principles, practice (on the so-called TV-Break Packing Problem) and related work, we refer the reader to [Benoist & Bourreau, 2003].

References

N. Beldiceanu and E. Contejean, 1994. *Introducing global constraints in CHIP*. Mathematical and Computer Modelling, 20(12):p 97-123.

T. Benoist, E. Bourreau, 2003. *Improving Global Constraints Support by Local Search*. In COSOLV'03.

T. Benoist, E. Gaudin, B. Rottembourg, 2002. *Constraint Programming Contribution to Benders Decomposition: A Case Study*. In CP-02, pages 603-617.

A. Bockmayr, N. Pisaruk, A. Aggoun, 2001. *Network Flow Problems in Constraint Programming*. In CP-01, pages 196-210.

Y. Caseau, F. Laburthe, 1999. *Heuristics for Large Constrained Vehicle Routing Problems*, Journal of Heuristics 5 (3), Kluwer.

T. Fahle, M. Sellmann, 2002. *Cost Based Filtering for the Constrained Knapsack Problem*. In Annals of Operations Research 115:73-94.

F. Focacci, F. Laburthe, A. Lodi, 2003. *Local search and constraint programming*. In Handbook of Metaheuristics pages 369-403, Kluwer.

F. Focacci and P. Shaw, 2002. *Pruning sub-optimal seach branches using local search*. In Proceedings of CP-AI-OR-02 pages 181-189.

O. Kamarainen, H. El Sakkout, 2002. *Local Probing Applied to Scheduling*. In CP02: 155-171.

U. Montanari, 1974. *Networks of constraints: fundamental properties and applications to picture processing*. Information Science, 7: 95-132.

J-C. Régin, 1994. *A filtering algorithm for constraints of difference in CSPs*. In AAAI 94, Twelth National Conference on Artificial Intelligence, pages 362-367, Seattle, Washington.

F. Sourd, P. Chrétienne, 1999, *Fiber-to-Object Assignment Heuristics*. In European Journal of Operational Research, 117.

M. Trick, 2001. A dynamic programming approach for consistency and propagation for knapsack constraints. Proceedings of CPAIOR-01 pages 113-124.