Combining Domain Splitting with Network Decomposition for Application in Model-Based Engineering

Rüdiger Lunde

R.O.S.E. Informatik GmbH, Schloßstr. 34, 89518 Heidenheim, Germany email: r.lunde@rose.de

Abstract. This paper discusses a new approach to combine the branch&prune algorithm with constraint network decomposition methods. We present a version of the well-known base algorithm which interleaves pruning, network decomposition and branching and is especially suited for large constraint networks with low connectivity. Special care is taken to support quantitative as well as qualitative and mixed domains. A sufficient criterion for the generation of decompositions called 'independent solvability' is defined, which guarantees maximal domain reduction for finite domains. We present production rules to compute decompositions and give an account of the implementation of the algorithm within a commercial model-based engineering tool.

1 Introduction

In current model-based engineering tools, constraint solving is used to predict the behavior of physical systems, based on component-oriented models. Constraints represent physical laws, whereas variables are used to describe behavioral modes of components as well as physical quantities or observable states of the system. Hybrid behavioral models containing quantitative relations (equations, inequalities, interpolating splines) as well as qualitative ones (tables or boolean formulas) give the modeler flexibility to choose the abstraction level which is best suited to the analysis task at hand.

To support the engineering process, the constraint solver is embedded into a reasoning framework, which uses the behavior prediction as a means to perform its high-level tasks. Typical tasks include predicting the expected behavior of a system over a period of time, estimating the risk that a certain top event occurs, explaining an observed situation, or proposing appropriate test points for candidate discrimination in diagnosis. Especially in diagnosis, models are often underdetermined. For complexity reasons, the constraint solver is required to deliver a compact representation of the solution space instead of point solutions of the network. Consequently, solvers used in this context apply inference techniques rather than search techniques (for a classification, see [5]).

In our approach, domain reduction is used to generate compact representations of solution spaces. The basic algorithm is a variant of the well-known branch&prune algorithm [7], which is extended by a network decomposition step. The paper elaborates on a decomposition strategy based on the so-called 'independent solvability' of subnetworks, which is appropriate to the structure of the constraint networks in model-based engineering.

The paper is organized as follows: We start with some preliminary definitions and a formalization of the investigated constraint problem. In Section 3, our version of the branch&prune algorithm is presented. Independent solvability of subnetworks as a sufficient criterion for constraint network decompositions is discussed in Section 4. We report briefly on the practical exploitation of the algorithm in Section 5, and conclude with outlining related work and promising directions of further research.

2 Basic Definitions

We define a constraint network in accordance with the literature (see e.g. [5]), but take special care to keep relation representations independent from variable domains and to support various kinds of domains:

Definition 1 (constraint network). *A constraint network $CN = \langle V, D, C\rangle$ consists of a finite set of variables $V = \{v_1, \ldots, v_n\}$, a domain composed of the corresponding variable domains $D = D_1 \times \ldots \times D_n$ and a set of constraints $C = \{c_1, \ldots, c_m\}$. All variable domains D_i are subset of a common universe domain U. Constraints are relations defined on sets of variables which constrain legal combinations of values for those variables. Each constraint c is represented by a pair $\langle V_c, R_c\rangle$. $V_c \subseteq V$ is called the scope of c and is accessed by the function* $\mathrm{scope}(c)$. *The relation R_c of c is a subset of $U^{|\mathrm{scope}(c)|}$ and is accessed by* $\mathrm{rel}(c)$.

Discrete variable domains are supported as well as continuous variable domains. If indices are not available, we write D_v to access the variable domain corresponding to variable v. In the algorithms discussed below, domains are represented by vectors of variable domains. If clear without ambiguity, those vectors are identified with the denoted cross product.

The projection function $\pi_i : 2^D \to 2^{D_i}$ is defined as usual. We write π_v instead of π_i where convenient. Obviously, for all domains and variables $\pi_v D = D_v$. The same notation is used to project a subset D' of a domain onto a set of variables instead of a single variable. For example, $\pi_{\{v_3,v_5\}} D = D_3 \times D_5$.

Subnetworks of CN are characterized by the following definition:

Definition 2 (subnetwork of a constraint network). *Let $CN = \langle V, D, C\rangle$ be a constraint network and $\hat{C} \subseteq C$ a set of constraints. The subnetwork of CN defined by $\hat{C}$ is a constraint network $\langle \hat{V}, \hat{D}, \hat{C}\rangle$ with the following properties: $\hat{V} = \bigcup_{c\in C} \mathrm{scope}(c)$, and $\hat{D} = \pi_{\hat{V}} D$. It is denoted by* $\mathrm{sn}(\hat{C}, CN)$.

An *instantiation* e of D is a set of value assignments, noted $\{\langle v_1, a_1\rangle, \ldots, \langle v_n, a_n\rangle\}$ with $v_i \in V$ and $a_i \in D_i$. It is called *complete*, if the set contains assignments for all $v \in V$, and *partial* otherwise. The *scope* of an instantiation e is defined by the set of those variables which occur in an assignment of e. Complete instantiations correspond to elements of D, whereas partial instantiations correspond to elements of projections obtained from D. For example, the instantiation $e = \{\langle v_1, a_1\rangle, \langle v_3, a_3\rangle\}$ corresponds to the value vector $\overline{e} = \langle a_1, a_3\rangle$, which is an element of $\pi_{\{v_1,v_3\}} D$. Value vectors corresponding to instantiations are indicated by overlining.

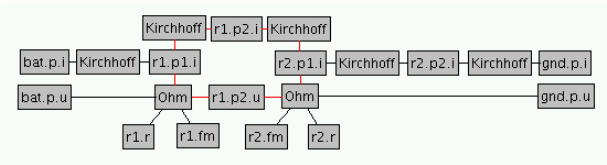

Fig. 1. Constraint net of two serial resistors. Nodes represent variables (dot-notation) and constraints

An instantiation e is said to satisfy a constraint c if $\mathrm{scope}(c) \subseteq \mathrm{scope}(e)$ and $\pi_{\mathrm{scope}(c)}\{\overline{e}\} \subseteq \mathrm{rel}(c)$. Each complete instantiation e of D, which satisfies all constraints $c \in C$ is called a *solution* of CN. The set of all value vectors which correspond to solutions of CN form a subset of D. It is called solution set of CN and noted $\mathrm{sol}(CN)$.

The space of all solutions of a given constraint network can be described by maximally reduced domains based on the following definition:

Definition 3 (maximally reduced domains). *Let $CN = \langle V, D, C\rangle$ be a constraint network and $\hat{D} \subseteq D$ a domain. $\hat{D}$ is called maximally reduced with respect to CN iff $\forall v \in V : \hat{D}_v = \pi_v \mathrm{sol}(CN)$.*

Our aim is to present efficient algorithms which compute maximally reduced domains $\hat{D}$ for given constraint networks. The discussed algorithms try to reduce the range of each variable as far as possible without losing any solution. If the given constraint network has only one solution, a single value remains for each variable. In underdetermined networks (which is common in model-based diagnosis, due to incomplete or fuzzy information) a set of possible values is computed for each variable. For constraint networks with discrete variable domains the domain $\hat{D}$ can be computed exactly. However, when reasoning about continuous domains it is infeasible to compute more than approximations of $\hat{D}$.

3 The Branch and Prune Algorithm

Constraint networks which describe real systems on a physical level are in general hard to solve because of cyclic dependencies, which occur in even the simplest aggregations of components (see Fig. 1 for illustration). The branch&prune algorithm [7] solves cycles by combining incomplete local constraint propagation [8] with recursive domain splitting. Our version of that algorithm, which is shown in Fig. 2, follows the idea of the original but adds a network decomposition step.

The algorithm starts with enforcing local consistency by means of the function `prune`, which implements a local constraint propagation method. If local propagation stops because one of the variable domains turns out to be empty, a domain composed of empty variable domains is returned. Otherwise, the constraint network is checked with regard to subnetworks with insufficiently reduced domains. The constraint sets which represent those parts are computed by the function `decomposeNetwork`. It returns a list of sets of constraints.

```
Domain branchAndPrune(⟨V, D, C⟩) {
    D′ = prune(⟨V, D, C⟩)
    if (¬empty(D′)) {
        L = decomposeNetwork(⟨V, D′, C⟩)
        for Ĉ across L {
            ⟨V̂, D̂, Ĉ⟩ = sn(Ĉ, ⟨V, D′, C⟩)
            ⟨D̂¹, D̂²⟩ = branch(⟨V̂, D̂, Ĉ⟩)
            if (D̂¹ ≠ D̂²) {
                D̂¹′ = branchAndPrune(⟨V̂, D̂¹, Ĉ⟩)
                D̂²′ = branchAndPrune(⟨V̂, D̂², Ĉ⟩)
                for (v ∈ V̂)
                    D′_v = D̂¹′_v ∪ D̂²′_v
                if (empty(D′))
                    break
            }
        }
    }
    return(¬empty(D′)) ? D′ : ⟨∅, ..., ∅⟩
}
```

Fig. 2. The branch&prune Algorithm

For each set of constraints, the corresponding subnetwork is determined and its domain split into two parts by means of the function `branch`. If this operation was successful, the algorithm branch&prune is applied recursively to both partitions, and the set union of the returned variable domains is used to reduce the resulting domain. The loop is aborted if the resulting domain is empty. In this case, a domain composed of empty variable domains is returned.

The method `branch` can be implemented by selecting a so-called split variable and splitting its domain. Important for the correctness of the algorithm is that it returns either a pair of subdomains which partition the original domain $\hat{D}$ or a pair of two equal domains, and that the former is the case if $\hat{D}$ contains a discrete variable domain with at least two elements.

We now discuss some basic features of the branch&prune algorithm. In the following, we assume that the set of different variable domain representations is finite (but not necessarily the domains themselves) and that the implementation of `decomposeNetwork` terminates for all constraint networks.

It can easily be seen that the algorithm branch&prune terminates: Let $<_{\mathrm{dom}}$ be a well-founded ordering on domains which agrees with narrowing, partitioning and subnetwork selection. One obvious realization for $<_{\mathrm{dom}}$ compares the corresponding variable domains with respect to set inclusion. Given such an ordering, in each recursive call, domains become smaller with respect to $<_{\mathrm{dom}}$.

Let $\mathrm{bp}(\langle V, D, C\rangle) = \mathrm{branchAndPrune}(\langle V, D, C\rangle)$. Obviously branch&prune is a narrowing algorithm, which does not add new elements to the given domain at any time:

Lemma 1. *Let $\langle V, D, C\rangle$ be a constraint network. $\forall v \in V : \mathrm{bp}(\langle V, D, C\rangle)_v \subseteq D_v$.*

As an important feature, the branch&prune algorithm does not remove any element from any variable domain which occurs in a solution.

Lemma 2. *Let CN be a constraint network.* $\forall v \in V : \mathrm{bp}(CN)_v \supseteq \pi_v \mathrm{sol}(CN)$.

The quality of the domain reduction, which is obtained by our version of the branch&prune algorithm, strongly depends on the network decomposition strategy. In the next section, we will introduce a sufficient decomposition criterion called 'independent solvability', which guarantees maximally reduced domains for constraint networks over finite domains. For decomposition algorithms based on this criterion, the computed domains are identical to the domains which are computed by the original branch&prune algorithm without network decomposition. The approximation accuracy for constraint networks over continuous or mixed domains depends on the chosen interval bound representation precision, the implementation of the method `branch`, and the used narrowing operators. Let us assume that the method `branch` partitions a given domain whenever possible with respect to the chosen interval bound representation, that all constraints describe continuous functions, and that the type of local consistency obtained by the narrowing operators converges with decreasing variable domain diameters to arc-consistency or a strong approximation like hull-consistency [1]. Then, the resulting domain converges with increasing bound representation precision to the maximally reduced domain.

In practice, the required accuracy depends on the analysis task to solve. Often weak approximations of maximally reduced domains suffice. If that is the case, the key to scale the precision of the result with regard to necessary accuracy and available resources lies in the splitting strategy used by the method `branch`. Our implementation selects continuous variables as split-variables only if the absolute and the relative diameters of their current domain bounds exceed specified thresholds.

Definition 4 (absolute and relative diameter). *Let $I = [a\ b]$ be an interval. The absolute diameter of I is defined by* $\mathrm{diam}(I) = b - a$ *and the relative diameter is defined by*

$$\mathrm{diamRel}(I) = \begin{cases} 0 & : \quad a = 0 = b \\ \infty & : \quad a < 0 \leq b \vee a \leq 0 < b \\ \frac{\mathrm{diam}(I)}{\min(|a|,|b|)} & : \quad \textit{otherwise}. \end{cases} \tag{1}$$

Especially in underdetermined networks a careful selection of appropriate thresholds is crucial for the success of the analysis.

4 Constraint Network Decomposition

When applying the branch&prune algorithm to possibly underdetermined constraint networks of several thousand constraints, the main problem is efficiency. We address this issue with a new structure-based approach.

In the field of continuous CSP solving, strong graph decomposition algorithms are known [2] which are based on the Dulmage and Mendelsohn decomposition. The result of this decomposition is a directed acyclic graph of blocks[1]. Unfortunately, those algorithms cannot be applied here, due to the hybrid structure of our constraint networks.

[1] Blocks correspond to subnetworks in our approach and generally include cycles. They are connected by directed arcs which represent causal dependencies.

While conditions might be handled by a two step simulation approach, which separates mode identification from continuous behavior prediction, inequalities are definitely not covered by the Dulmage and Mendelsohn decomposition technique. Solvers based on this decomposition techniques compute series of point-solutions, but fail to determine complete ranges for variables in underdetermined networks.[2]

The decomposition criteria proposed here are much weaker, but do not impose restrictions on the type of relations used. Their aim is to identify parts of the constraint network which can be solved sequentially, one after the other. Besides the graph information obtained from the constraint network structure, we integrate another source of information, namely, the current value ranges of the variables. Since the ranges are subject to change during the recursive constraint solving process, graph decomposition is not applied as a preprocessing step, but is performed anew after each pruning step.[3]

4.1 Graph Notions

The static dependency structure of a constraint network $CN = \langle V, D, C \rangle$ is represented by an undirected bipartite graph G called *constraint graph of* CN. The set of vertices is defined by the union of all variables and all constraints in CN. Undirected edges connect each constraint with the variables of its scope. We represent undirected edges by sets. So we get

$$G = \langle V \cup C, \{\{c, v\} \mid c \in C \wedge v \in V \wedge v \in \mathrm{scope}(c)\}\rangle.$$

For the computation of maximally reduced domains, dependencies between variables whose domains are not uniquely restricted are of special interest. We call a variable v *relevant* iff $|D_v| > 1$. Replacing V by the set of all relevant variables $V_r \subseteq V$ in the definition above, a subgraph of G is obtained which we call *relevant subgraph of* CN.

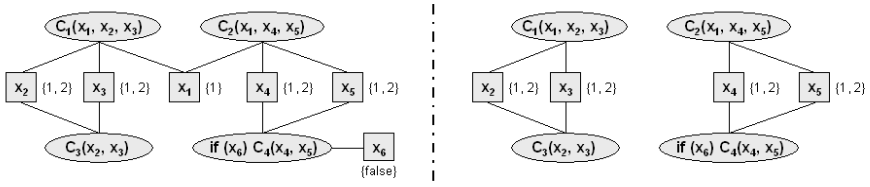

Fig. 3. A relevant subgraph

The complexity of the task to compute maximally reduced domains for a given constraint system strongly depends on the structure of its relevant subgraph. If the graph is acyclic pure local propagation is sufficient, as the following theorem states.

[2] One reason for this lies in hidden cycles between different blocks. While causality between different blocks is cycle-free, dependencies are not. In general, the corresponding undirected version of the block graph is cyclic.

[3] Implementations can profit from the monotonic refinement of ranges by reusing graph analysis results gained one recursion level up.

Theorem 1. *Let $CN = \langle V, D, C\rangle$ be a constraint network with a non-empty domain. If CN is arc-consistent and its relevant subgraph acyclic, then D is maximally reduced with respect to CN.*

Proof. Let CN be arc-consistent and its relevant subgraph G be acyclic. Obviously $\forall v \in V : D_v \supseteq \pi_v \mathrm{sol}(CN)$. We show that $\forall v \in V : D_v \subseteq \pi_v \mathrm{sol}(CN)$.

Let $C_r \subseteq C$ be the set of all constraints, whose scope contains at least one relevant variable. Since CN is arc-consistent, each complete instantiation of D which satisfies all $c \in C_r$ also satisfies all $c \in C \setminus C_r$. Hence $\mathrm{sol}(CN) = \mathrm{sol}(\langle V, D, C_r\rangle)$ and it suffices to show that $\forall v \in V : D_v \subseteq \pi_v \mathrm{sol}(\langle V, D, C_r\rangle)$.

By precondition, each variable domain contains at least one value. If C_r is empty, all complete instantiations of D are solutions of $\langle V, D, C_r\rangle$ and there exists at least one. Hence $\forall v \in V : D_v \subseteq \pi_v \mathrm{sol}(\langle V, D, C_r\rangle)$.

Otherwise let $v \in V$ and $a \in D_v$ be chosen arbitrarily. We have to prove that there exists a solution of $\langle V, D, C_r\rangle$ which contains the assignment $\langle v, a\rangle$. Let G_r be the relevant subgraph of $\mathrm{sn}(C_r, CN)$. G_r contains at least one constraint and a relevant variable which is connected to it.

Case 1: v is a vertex of G_r. Since G is acyclic, G_r is acyclic too. Hence, we can arrange its nodes as a forest, in which one of the trees is rooted by v. Nodes with even depth are variables and with odd depth constraints. Let $<_{C_r}$ be a total ordering on the constraint vertices which agrees with depth, and for each instantiation e let $\mathrm{mc}(e)$ denote the minimal constraint c with respect to $<_{C_r}$ which is not satisfied.

Let us now assume that there is no solution of $\langle V, D, C_r\rangle$ which contains the assignment $\langle v, a\rangle$. Among the complete instantiations of D which contain $\langle v, a\rangle$, there must be an instantiation e with the property that $c = \mathrm{mc}(e)$ is maximal with respect to $<_{C_r}$. Since CN is arc-consistent, regardless of the value assignment which is used in e for the parent variable v_p of c in G_r, there must exist value assignments e_s for all other variables in $\mathrm{scope}(c)$ such that c is satisfied. By replacing in e the assignments for all variables in $\mathrm{scope}(c) \setminus \{v_p\}$ by e_s, we obtain a new instantiation e' with differs from e in some assignments for the child variables of c in G_r. Since it satisfies c and (like e) all smaller constraints with respect to $<_{C_r}$, we get $\mathrm{mc}(e') >_{C_r} \mathrm{mc}(e)$, which is a contradiction to our assumption. So there must be a solution of $\langle V, D, C_r\rangle$ which contains the assignment $\langle v, a\rangle$.

Case 2: v is not a vertex of G_r. Let v' be a variable vertex of G_r and $a' \in D_{v'}$ chosen arbitrarily. As proved in case 1 there exists a solution e of $\langle V, D, C_r\rangle$ which contains $\langle v', a'\rangle$. Since v is irrelevant, $D_v = \{a\}$. We conclude that e must contain the assignment $\langle v, a\rangle$. □

4.2 Independent Solvability

When searching for realizations of `decomposeNetwork`, the major motivation is the reduction of computational costs to compute maximally reduced domains. The smaller the resulting subnetworks, the better. But we have to be careful not to loose too much narrowing quality, compared to the branch&prune variant without network decomposition. We definitely want to guarantee maximally reduced domains for constraint networks over finite domains.

We start by defining a class of decompositions called independently solvable decompositions. Based on the definition, a sufficient but not necessary criterion for the realization of `decomposeNetwork` is given. The problem of computing independently solvable decompositions is addressed by providing a set of production rules.

Definition 5 (decomposition of a constraint network). *Let $CN = \langle V, D, C\rangle$ be a constraint network, L a set of mutually disjoint subsets of C. The set of subnetworks $\Lambda = \bigcup_{\hat{C}\in L}\{\mathrm{sn}(\hat{C}, CN)\}$ is called a decomposition of CN. $\Lambda_v \subseteq \Lambda$ denotes the set of all constraint networks in Λ which contain the variable v.*

Definition 6 (independent solvability). *Let $CN = \langle V, D, C\rangle$ be a constraint network, Λ a decomposition of CN. Let $D^{\mathrm{is}} \subseteq D$ be a domain whose variable domains are computed from the maximally reduced domains for the corresponding constraint networks in Λ by intersection: $\forall v \in V : D_v^{\mathrm{is}} = \bigcap_{cn\in\Lambda_v} \pi_v\mathrm{sol}(cn) \cap D_v$*

Λ is called independently solvable, iff D^{is} is empty or maximally reduced with respect to CN.

An implementation of `decomposeNetwork` can be based on independent solvability. In this case, for any constraint network CN over finite domains, `decomposeNetwork` computes a vector $\langle C_1, \ldots, C_n\rangle$ with the property that $\bigcup_{1\leq i\leq n}\{\mathrm{sn}(C_i, CN)\}$ is an independently solvable decomposition of CN. In the following $\mathrm{bp}^{\mathrm{is}}(CN)$ denotes the result of a variant of the branch&prune algorithm which uses an independent solvability based `decomposeNetwork` implementation.

Theorem 2 (correctness of branchAndPrune). *The branch&prune algorithm returns maximally reduced domains for all constraint networks over finite domains if the implementation of* `decomposeNetwork` *is based on independent solvability.*

Proof. (Sketch) Thanks to Lemma 2, it suffices to show, that for all domains D which are composed of finite variable domains, the following proposition holds:

$$\forall C\ \forall V\ \forall v \in V : \mathrm{bp}^{\mathrm{is}}(\langle V, D, C\rangle)_v \subseteq \pi_v\mathrm{sol}(\langle V, D, C\rangle). \tag{2}$$

The proof is a strong induction over the domain size. The central loop invariant is

$$\mathrm{empty}(D')\ \vee\ \forall v \in V : D'_v \subseteq \bigcap_{cn\in\Lambda_v^i} \pi_v\mathrm{sol}(cn) \cap D_v^{\mathrm{pr}},$$

where i is the number of previously performed loop body evaluations, D^{pr} the contents of the algorithm variable D' after pruning, $\Lambda^i = \bigcup_{j\leq i}\{\mathrm{sn}(C_j, \langle V, D^{\mathrm{pr}}, C\rangle)\}$, and C_j the components of the vector which was returned by `decomposeNetwork`. □

Consequently, independent solvability is a sufficient criterion to guarantee maximally reduced domains for constraint networks over finite domains. We now face the task how to compute such decompositions. The Lemmas 3 – 6 are called production rules for independently solvable decompositions. They suggest to start with a trivial decomposition, and to refine it subsequently by removing or splitting contained subnetworks.

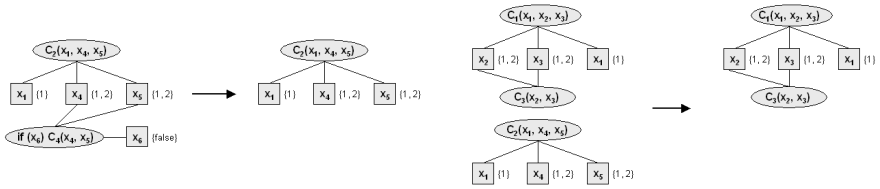

Fig. 4. Constraint removal and subnetwork removal

Lemma 3 (trivial decomposition). *Let $CN = \langle V, D, C\rangle$ be a constraint network. The set $\{\mathrm{sn}(C, CN)\}$ is an independently solvable decomposition of CN.*

Proof. (Sketch) If D is empty, $\{\mathrm{sn}(C, CN)\}$ is an independently solvable decomposition of CN. Otherwise let $\langle \hat{V}, \hat{D}, C\rangle = \mathrm{sn}(C, CN)$. Since D_v is not empty for all $v \in V \setminus \hat{V}$, every solution of $\langle \hat{V}, \hat{D}, C\rangle$ which contains the assignment $\langle v, a\rangle$ can be converted into a solution of $\langle V, D, C\rangle$ with the same assignment and vice versa. So we get for all $v \in \hat{V}$, $\pi_v \mathrm{sol}(\langle \hat{V}, \hat{D}, C\rangle) = \pi_v \mathrm{sol}(\langle V, D, C\rangle)$, which implies the lemma. □

Many interactions between different components depend on the current operational or fault state of the system. An open switch disconnects two parts of a circuit, the same is true for a closed valve in a hydraulic system or a broken belt in an engine. Using conditional constraints to express state-dependent relations allows to disconnect parts of the networks, which are not independent in general, but with respect to the currently analyzed state.

Lemma 4 (constraint removal). *Let $CN = \langle V, D, C\rangle$ be a constraint network, $\Lambda \uplus \{\mathrm{sn}(\hat{C}, CN)\}$ an independently solvable decomposition of CN and $c \in \hat{C}$ a constraint, whose condition cannot be fulfilled by the instantiations of D. Then $\Lambda \cup \{\mathrm{sn}(\hat{C} \setminus \{c\}, CN)\}$ is an independently solvable decomposition of CN.*

Proof. Since c is satisfied by any instantiation e with $\mathrm{scope}(e) \supseteq \mathrm{scope}(c)$, we get $\mathrm{sol}(\mathrm{sn}(\hat{C} \setminus \{c\}, CN)) = \mathrm{sol}(\mathrm{sn}(\hat{C}, CN))$ which implies the lemma. □

As stated by Theorem 1, domains of arc-consistent acyclic networks are maximally reduced. Therefore, such subnetworks can be removed from an independently solvable decomposition. Since the branch&prune algorithm performs local propagation before splitting the network, it can provide arc-consistency for certain subnetworks (e.g. subnetworks over finite domains).

Lemma 5 (subnetwork removal). *Let CN be a constraint network and $\Lambda \uplus \{\hat{CN}\}$ an independently solvable decomposition of CN. If $\hat{CN}$ is arc-consistent and its relevant subgraph acyclic, then Λ is an independently solvable decomposition of CN.*

Finally, subnetworks whose relevant subgraphs do not overlap can be split, as stated in the last production rule.

Lemma 6 (subnetwork splitting). *Let CN be a constraint network, $\hat{CN} = \mathrm{sn}(\hat{C}, CN)$ a subnetwork of CN and $\Lambda \uplus \{\hat{CN}\}$ an independently solvable decomposition of CN. Let further $C^1 \uplus C^2 = \hat{C}$ be a partitioning of $\hat{C}$ and $CN^i = \mathrm{sn}(C^i, CN)$ ($1 \leq i \leq 2$) the*

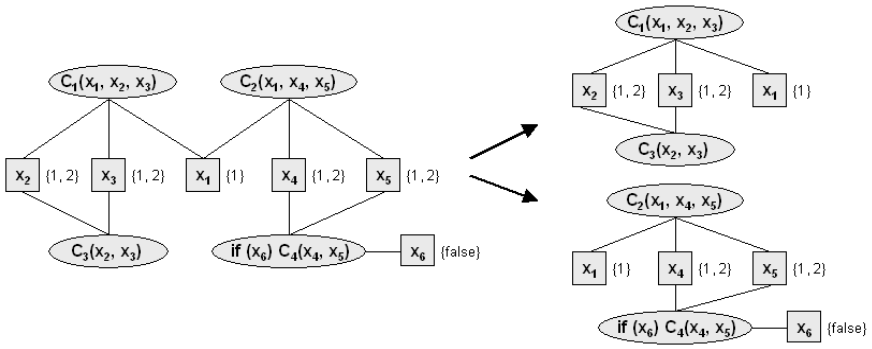

Fig. 5. Subnetwork splitting

corresponding subnetworks. If the relevant subgraphs of CN^1 and CN^2 do not overlap, then $\Lambda \cup \{CN^1, CN^2\}$ is an independently solvable decomposition of CN.

Proof. (Sketch) Let $\langle V, D, C\rangle = CN$ and $\langle V^i, D^i, C^i\rangle = CN^i$ for $1 \leq i \leq 2$. Since $D^i_v = D_v$ for all variables $v \in V^i$, and $|D_v| = 1$ for all common variables $v \in V^1 \cap V^2$, the union of all value assignments contained in any pair of solutions $e^i \in \mathrm{sol}(CN^i)$ ($1 \leq i \leq 2$) forms a solution e of $\hat{CN}$.

The rest of the proof instantiates the definition of independent solvability for $\Lambda \cup \{CN^1, CN^2\}$ and $\Lambda \uplus \{\hat{CN}\}$ and makes a case differentiation about whether $\mathrm{sol}(\hat{CN})$ is empty or not. □

5 Experimental Results

We have integrated a Java implementation of the presented concepts into the commercial model-based engineering tool RODON.

To demonstrate the usage of the branch and prune approach in our application focus, we have chosen a diagnostic problem. Given a model of a real system and a symptom describing some abnormal state, the diagnostic module of RODON performs a conflict directed search for diagnostic candidates. The results presented in Table 1 are based on a quantitative model of an automotive exterior lighting system (see Fig. 6). It consists of three electrical control units, 10 drivers providing 20 diagnostic trouble codes, 4 switches, 12 actuators and several connector blocks, wires, fuses, diodes, resistors and relays. Altogether 86 physical components are included, defining 143 single faults including unknown fault modes for all connector boxes. The behavior model is built out of 1816 variables (most of them quantitative and unbounded) and 1565 constraints.

The first diagnosed symptom ($\triangle$) describes the observation that two rear lights are dimmed which can be explained for example by a disconnected ground node. RODON checks 15 diagnostic candidates and returns five of them which are consistent with the given symptom. The second symptom ($\diamond$) represents a partial system state in which two side marker fault codes indicate a short to ground failure. Here 18 diagnostic candidates are checked and two returned, including a multiple fault. For both symptoms behavior

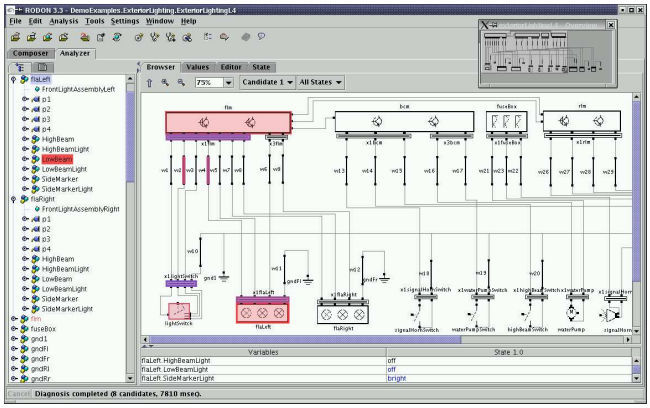

Fig. 6. The model-based engineering tool RODON

prediction based on pure local propagation fails due to a large number of algebraic cycles.

Table 1 shows that while the number of successful network decompositions (#splits) is limited in the presented example runs, the impact on the number of recursive solver calls (#propagations) as well as the overall time to compute the diagnosis (t, measured on a Pentium 4 with 1.1GHz) is significant. We have tested the algorithm in different analysis tasks with various models including much larger ones. As one would expect, the performance gains differ from case to case, but the presented results seem to be typical for the average case. Network decomposition has shown to reduce execution time significantly when underconstraint states are investigated. The smaller the diameters used to select split variables are, the more computation time can be saved.

Task Characterization			Without Decomposition		With Decomposition		
Symptom	diam	diamRel	#propagations	t [sec]	#splits	#propagations	t [sec]
△	1	5	100	5.4	7	80	1.9
△	0.01	1	954	11	7	204	3
△	0.01	0.1	99056	1209	7	4768	34
◇	1	5	8453	66	5	261	3.5
◇	0.01	1	> 55000	> 1000	5	21821	190

Table 1. Performance gained by network decomposition during diagnostic runs

6 Conclusion and Related Work

Reducing domains in large cyclic constraint structures is generally a hard task, especially if the system includes continuous variables. The currently known inference methods can be divided into algebraic and numeric methods.

Several contributions have addressed the problem of solving cyclic constraint structures by algebraic transformation algorithms (see e. g. [6] for a survey). By linearization,

a linear system of equations is obtained which can be solved by the Gauss-Seidel iterative method. Polynomial systems can be transformed by Gröbner basis computation. A common framework for transformations based on variable elimination called "Bucket Elimination" is presented in [4].

Numeric methods reduce variable domains by combining local consistency criteria with some kind of recursive trial-and-error strategy. The branch&prune algorithm discussed in this paper splits domains and tests local consistency of the resulting boxes. Other numeric methods shift the bounds of variable domains until inconsistency is detected (see e.g. [3]).

While algebraic methods can only be applied to more or less restricted types of constraints, numeric methods suffer from their worst-case complexity, especially when applied to large underconstraint networks. In this paper, we have presented a structure-based approach to improve the performance of the branch&prune algorithm by defining a class of state dependent network decompositions called 'independently solvable decompositions' and providing production rules to compute instances of it. The decompositions allow to solve independent parts sequentially instead of recursively, and thereby reduce recursion depth significantly. Parts which are already maximally reduced are identified by cycle analysis and removed from the network. These optimizations increase significantly the size of models whose solution is feasible in practice.

The class of independently solvable decompositions is very general and can improve other numeric domain reduction methods as well. Nevertheless, one obvious limitation of all decompositions which can be derived by the provided production rules is, that the contained subnetworks do not share any common relevant variable. That is partly because the production rules do not span the complete class of independently solvable decompositions. But the required notion of independence also limits the available decompositions more than necessary. We are currently working on relaxations of this criterion.

References

1. F. Benhamou, D. McAllester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Proceedings Int. Logic Programming Symposium*, 1994.
2. Christian Bliek, Bertrand Neveu, and Gilles Trombettoni. Using graph decomposition for solving continuous CSPs. *Lecture Notes in Computer Science*, 1520:102+, 1998.
3. Lucas Bordeaux, Eric Monfroy, and Frederic Benhamou. Improved bounds on the complexity of kB-consistency. In *IJCAI*, pages 303–308, 2001.
4. Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.
5. Rina Dechter. *Constraint Processing*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, May 2003.
6. L. Granvilliers, E. Monfroy, and F. Benhamou. Symbolic-interval cooperation in constraint programming. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 150–166, 2001.
7. P. Van Hentenryck, D. McAllester, and D. Kapur. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis*, 34(2):797–827, April 1997.
8. A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25, 1985.