Applying Domain Splitting to Behavior Prediction in Model-Based Diagnosis

Rüdiger Lunde [1]

Abstract. This paper discusses the application of constraint solving methods to model-based diagnosis of technical systems. Our approach is based on interval propagation embedded in a branch and prune algorithm. To take into account the large size of constraint nets describing the behavior of real systems, we have taken special care to reduce recursion depth by means of graph decomposition and intelligent branching. An ATMS interface is presented which supports both the behavior prediction and the candidate generation tasks. Finally, we discuss experimental results gained from our implementation.

1 Introduction

Since de Kleer and Williams introduced the GDE concept [5] in the late eighties, lots of diagnostic systems have been presented, which extend the original concept in some way. Explicit knowledge about faulty behavior was added to the models [16], generalizations were presented for the diagnosis of finite state machines [17], and many contributions adressed improvements of efficiency. But still, two fundamental GDE concepts are shared by the majority of current model-based diagnostic systems: The paradigm of component-oriented behavioral modeling and the use of conflicts to direct the search for diagnostic candidates.

The need for conflicts forces developers of inference machines to incorporate some kind of local propagation which allows the solver to track dependencies, especially between diagnostic assumptions and prediction results. Local propagation iterates in some order across the relations of the behavioral model while refining the representation of the analyzed system states until a fixed point is reached. For example, if static behavior is represented by constraints over finite domains, the refinement for a given relation is usually achieved by computing projections for each variable which is constrained by that relation.

The drawbacks of local propagation are well known [14]. In general, this kind of inference is incomplete. Overestimations can and will occur if the relations of the behavioral model contain cyclic dependencies. Such structures occur frequently in systems whose component interfaces are characterized by vectors of flow and potential quantities. Figure 1 shows a widely known example in the domain of analogue circuits: The behavioral model of a series of two resistors, based on Ohm's and Kirchhoff's laws. Fortunately, cycles do not necessarily lead to overestimation. In our simple example, the success of local propagation depends on the relations used. If the ranges of all variables are bounded and the resistance parameters of both resistors are sufficiently different, pure interval propagation will suffice to prune out impossible parts of the given ranges.

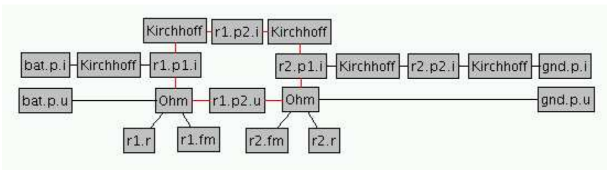


Figure 1. Constraint net of two serial resistors. Nodes represent variables (names in dot-notation) and constraints.

From projects, mostly in the automotive domain, we gained experience which shows that there is a need for stronger constraint solvers to achieve reliable prediction results for models of real-world systems. In this paper, we present an algorithm which combines local propagation with domain splitting techniques. It is well suited for constraint systems over continuous domains, but can be applied to finite and mixed domains as well. The basic idea coincides with the branch and prune algorithm presented in [10]. Our extensions to that approach are motivated by the special structure of the models we built for automotive applications. These models tend to be rather complex and large (containing several thousand variables and constraints), but use mostly very simple constraints. Hence value refinement with respect to a single constraint is cheap, and hull-consistency can often be achieved. At the same time, branching is very expensive.

The motivation of the paper focusses on model-based diagnosis. Nevertheless, the underlying principle of conflict-directed search for states with certain properties is very general. Therefore, the presented algorithms contribute to other application areas as well, including heuristic diagnostic knowledge generation, automated risk analysis and design for diagnosis. We start with some preliminary notions and a survey about the branch and prune algorithm in Section 2. We take a closer look on subnet selection based on graph decomposition techniques, intelligent branching strategies and an ATMS interface in Section 3. In Section 4, we discuss some experiences gained from our implementation of the presented algorithms, which is now part of the commercial model-based analysis tool RODON.

2 The Branch and Prune Algorithm

Before we describe the branch and prune algorithm, we start with some basic definitions to formalize the problem to be solved in the paper.

2.1 Constraint System

System behavior is described in the form of constraints:

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

Definition 1 (constraint system) *A constraint system $CS = \langle V, D, C\rangle$ 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 $C = \{c_1, \ldots, c_m\}$ over V, restricting the values that may be assigned to variables. Subsets of C are also called subnets.*

We support discrete variable domains as well as continuous variable domains. 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.

Let $\mathrm{var} : C \to 2^V$ be a function returning the set of variables affected by a given constraint C, and $\mathrm{rel} : C \to 2^D$ a mapping from a constraint to its corresponding relation which is a subset of D. Relations can be defined extensionally (by tables) or intentionally, for example by (in)equalities. The projection function $\pi_i : 2^D \to 2^{D_i}$ maps each subset D' of the domain D to a subset of the variable domain D_i:

$$\begin{aligned}\pi_i(D') \quad = \quad & \{e_i \mid \exists e_1 \ldots \exists e_{i-1} \exists e_{i+1} \ldots \exists e_n : \\ & \langle e_1, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_n\rangle \in D'\}\end{aligned} \tag{1}$$

Elements of D satisfying all constraints are called solutions, and the set of all solutions can be characterized by

$$\mathrm{sol}(CS) = \bigcap_{1 \le j \le m} \mathrm{rel}(c_j). \tag{2}$$

The set of possible values for a single variable v_i is given by $\pi_i(\mathrm{sol}(CS))$.

In the context of model-based diagnosis, we use variables to model behavioral modes of components (e.g. switch open/closed, wire ok/defective) as well as observable or internal states (e.g. engine stopped/running) and other quantities (e.g. current through a connection). Conditional constraints of the form `if <condition> then <relation>` are used to express mode-dependent behavior. To guarantee decidability of conditions, we expect all variables which are tested in a condition to have a finite domain.

It seems to be a key requirement to allow different levels of abstraction in models, as well as the explicit representation of uncertain knowledge. Therefore, we are especially interested in hybrid behavioral models containing quantitative relations (equations, inequalities, interpolating splines) as well as qualitative ones (tables, formulas with qualitative operators).

2.2 Constraint Solving

For certain diagnostic problems it suffices to determine whether a solution exists or not [2], for instance to check a candidate by steady state analysis in a consistency-based framework. More information about possible values of variables is needed for abductive diagnosis, or for predicting the evolution of states over time. Unfortunately, the set of all solutions cannot be computed exactly because of the potentially infinite set size, and the limited precision of machine number representations. That is why interval analysis algorithms focus on hull representations of solution spaces, based on the notion of global consistency.

Definition 2 (global consistency) *A constraint system $CS = \langle V, D, C\rangle$ is called globally consistent iff $\forall i : D_i = \pi_i(\mathrm{sol}(CS))$. In the context of fixed sets V and C we also speak of globally consistent variable domains.*

[2] Even this problem is NP-complete for constraint systems with finite domains.

Let $CS = \langle V, D, C\rangle$ be a constraint system. The aim of this paper is to present efficient algorithms to compute approximations of a domain D' with the following properties:

- $D' \subseteq D$,
- $\mathrm{sol}(CS) = \mathrm{sol}(\langle V, D', C\rangle)$ and
- $\langle V, D', C\rangle$ is globally consistent.

The discussed algorithms try to reduce the range of each variable as far as possible without losing any solution. Ideally, a single value remains for each variable. In underdetermined systems (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 systems with discrete variable domains the domain D' can be computed exactly. However, when reasoning about continuous domains we can compute only supersets of D'.

2.3 The Base Algorithm

In [10] a recursive branch and prune algorithm is presented to solve polynomial systems. Our version of that algorithm is shown in Fig. 2. It follows the general idea of the original but extends it by a graph decomposition step.

The algorithm starts with enforcing local consistency by means of the function `prune`, which implements a local constraint propagation method. For numeric constraints, interval propagation [11] is applied. If local propagation stops because one of the variable domains turns out to be empty, an empty domain is returned. Otherwise, the constraint system is checked whether some of its parts are insufficiently solved. The constraint sets which represent those parts are computed by the function `getSubnetsForBranching`. It returns a set of subnets. A very simple implementation of that function could look for domains D_i which are not yet small enough (with respect to some measure), and return the empty set if there are no such

```
Domain branchAndPrune($\langle V, D = \langle D_1, \ldots, D_n\rangle, C\rangle$) {
    $D'$ = prune($\langle V, D, C\rangle$)
    if ($\exists i : D'_i = \emptyset$) {
        return $\langle \emptyset, \ldots, \emptyset\rangle$
    } else {
        $L$ = getSubnetsForBranching($\langle V, D', C\rangle$)
        for ($C' \in L$) {
            $\langle D^1, D^2\rangle$ = branch($\langle V, D', C'\rangle$)
            $D^{1'}$ = branchAndPrune($\langle V, D^1, C'\rangle$)
            $D^{2'}$ = branchAndPrune($\langle V, D^2, C'\rangle$)
            $D' = \langle D_1^{1'} \cup D_1^{2'}, \ldots, D_n^{1'} \cup D_n^{2'}\rangle$
        }
        return $D'$
    }
}
```

Figure 2. The Branch & Prune Algorithm

```
$\langle$Domain, Domain$\rangle$ branch($\langle V, D, C\rangle$) {
    $v_i$ = select a decision variable from $\bigcup_{c \in C} \mathrm{var}(c)$
    $\langle D_i^1, D_i^2\rangle = D_i$.split()
    return $\langle\langle D_1, \ldots, D_i^1, \ldots, D_n\rangle, \langle D_1, \ldots, D_i^2, \ldots, D_n\rangle\rangle$
}
```

Figure 3. Decision variable selection in algorithm `branch`

domains, and otherwise, a set containing the set of all constraints as a single subnet. For instance, in discrete domains, a suitable measure is size$(D_i) > 1$.

For each subnet, the domain is split into two parts by means of the function `branch`. To both partitions, the algorithm `branchAndPrune` is applied recursively. Afterwards, the set union of the variable domains of both partitions is used to define the resulting domain.

Fig. 3 describes a possible realization of domain partitioning by selecting a so-called decision variable and splitting its domain. Suitable selection criteria will be discussed in Section 3.2.

Given the simple implementation of `getSubnetsFor-Branching` discussed above and the finite domains for each variable, it can easily be seen that the algorithm `branchAndPrune` terminates. The total size of all variable domains provides an upper bound for the recursion depth. Let $D^{\mathrm{bp}} = \mathrm{branchAndPrune}(\langle V, D, C\rangle)$. We show that $\mathrm{sol}(\langle V, D, C\rangle) \subseteq \mathrm{sol}(\langle V, D^{\mathrm{bp}}, C\rangle)$, i. e. `branchAndPrune` does not lose a solution, by induction over the number of elements in D:

1. Let D contain just one element e and let $e \in \mathrm{sol}(\langle V, D, C\rangle)$. We show that $e \in D^{\mathrm{bp}}$ by stepping through the statements of the algorithm. Since e satisfies all constraints, it is part of D' after pruning. No subnet for branching is found, so D' (containing e) is returned.
2. Let D contain n elements, and let one of those elements e satisfy all constraints. The solution e is not lost during pruning, hence $e \in D'$. If D' contains just e, D' is returned.
   Otherwise, C is selected for branching and the domain is partitioned. So there exists a partition D^i with $e \in D^i$ and size(D^i) $< n$. The induction assumption implies that $e \in D^{i'}$. So after uniting the results of both partitions, e is contained in D', which is finally returned.

To show that $\langle V, D^{\mathrm{bp}}, C\rangle$ is globally consistent, it can be proven in a similar way, by induction over the size of D, that

$$\forall i \le n\, \forall e_j : e_j \notin \pi_i(\mathrm{sol}(\langle V, D, C\rangle)) \rightarrow e_j \notin D_i^{\mathrm{bp}}. \qquad (3)$$

Hence, using contraposition and the fact that no solution is lost,

$$\forall i \le n\, \forall e_j : e_j \in D_i^{\mathrm{bp}} \rightarrow e_j \in \pi_i(\mathrm{sol}(\langle V, D^{\mathrm{bp}}, C\rangle)). \qquad (4)$$

In general, global consistency is lost if continuous domains are used. But approximations of globally consistent domains can still be obtained. When specializing the generic branch and prune algorithm for interval domains, besides accuracy of the approximation a major concern is efficiency.

3 Extensions for Application in Model-Based Diagnosis

3.1 Subnet Selection

In the field of continuous CSP solving, strong graph decomposition algorithms are known [1], which are based on the Dulmage and Mendelsohn decomposition. In those algorithms, a system of equations is viewed as a bipartite graph, whose vertices represent the variables and equations. Edges connect each equation with the variables occuring in it. The result of the decomposition is a directed acyclic graph of blocks[3]. Unfortunately, those algorithms cannot be applied here, due to the hybrid structure of our constraint systems. While conditions might be handled by a two step simulation approach, which separates mode identification from continuous behavior prediction, inequalities are definitly not covered by the Dulmage and Mendelsohn decomposition technique.

The decomposition criteria proposed here are much weaker, but do not impose restrictions on the kinds of relations used. Their aim is to identify parts of the constraint system which can be solved independently from each other. Besides the graph information obtained from the constraint system 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 solving process, graph decomposition is not applied as a preprocessing step. Instead, it is performed anew after each pruning step.[4]

If there exists a partitioning of the graph with the property that there is no edge between nodes of different partitions, it is obvious that the solutions of the whole constraint system can be computed by solving each partition independently. The decomposition can be computed in linear time with respect to the number of vertices. But real systems do not often consist of completely independent parts. In the following, we try to separate parts of the graph by stepwise removal of irrelevant vertices and edges. In Figures 4–6, each step is illustrated for a small constraint system with four constraints (ellipses) and six variables (rectangles). Value ranges are noted in curly brackets.

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 distinguish between relations which are relevant for the current context and those which are currently irrelevant. Constraints whose conditions have been evaluated to false are irrelevant for dependency analysis, and can be removed from the graph. By focussing on relevant constraints, partitions can be computed which are not independent in general, but with respect to the currently analyzed state.

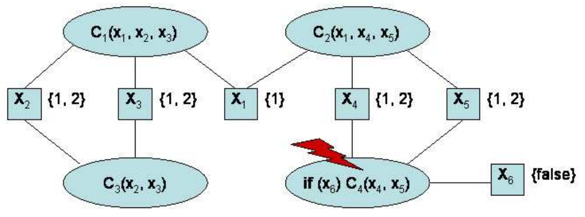


Figure 4. Step 1: Removing irrelevant constraints

Further subnet decomposition can be based on variables whose range has already been determined to be a unique value. Pruning stops at variables whose current range cannot be narrowed. Hence, the effects of recursive pruning after splitting the domain of a decision variable is restricted to non-unique values of variables in the neighborhood of the decision variable. Consequently, variables with unique values are irrelevant for subnet dependency analysis, and can be removed from the analyzed constraint graph.

Unfortunately, unique value domains are never reached when splitting interval domains. So we have to define an approximation of suffi-

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

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

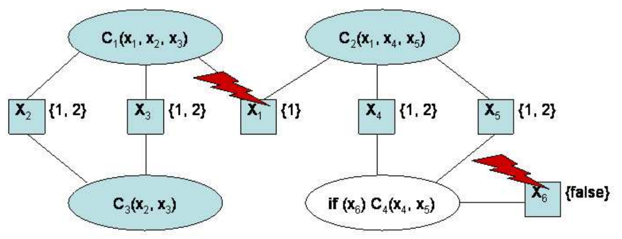


Figure 5. Step 2: Removing irrelevant variables

cient uniqueness. The notion we use is based on absolute and relative interval diameters:

Definition 3 (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*

$$\mathrm{diamRel}(I) = \begin{cases} 0 & : \quad a = 0 = b \\ \infty & : \quad a < 0 \leq b \vee a \leq 0 < b \\ \frac{\mathrm{diam}(I)}{\min(|a|,|b|)} & : \quad \textit{otherwise}. \end{cases} \tag{5}$$

Definition 4 (uniqueness with respect to maximal diameters) *An interval* I *is called unique with respect to the diameters* d_{abs} *and* d_{rel} *iff* $\mathrm{diam}(I) \leq d_{\mathrm{abs}}$ *or* $\mathrm{diamRel}(I) \leq d_{\mathrm{rel}}$.

The definition can be generalized in a straightforward way for sets of intervals and also discrete sets, by defining that sets with size greater than one are not unique regardless of the specified diameters. In our implementation, all variables whose current range is unique with respect to the specified diameters are removed from the graph. Termination is guaranteed if the domains of all variables are bounded.

If pruning is based on arc-consistency or a strong approximation like hull-consistency[5], branching is not needed unless there are algebraic loops in the constraint net. Hence, cycle analysis can be used to estimate the success of further branching, and subnets without any cycles can be removed from the graph.

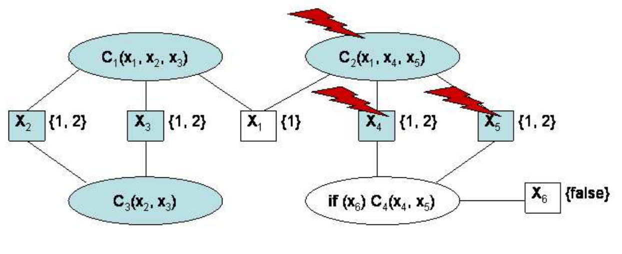


Figure 6. Step 3: Removing cycle-free subgraphs

3.2 Intelligent Branching

Given a certain subnet, branching strategies have to face two problems:

- What is the best decision variable?
- Which is the best way to split the domain of that variable?

[5] See e.g. [10] for a definition.

The latter question is the easier one to answer. It suggests itself to split bounded domains "in the middle". More crucial for efficiency is the selection of the decision variable. We use a combination of several heuristics based on the following criteria (among others):

- Variable types: prefer discrete variables, especially those who occur in not yet decided conditions;
- Value set size: prefer sets with two elements;
- Interval diameter: prefer large diameters.[6]

Another rather helpful criterion can be defined using the results of cycle analysis. Recall that the relevance of a variable depends on uniqueness with respect to the diameters d_{abs} and d_{rel}. Hence, splitting a variable's domain can make this variable irrelevant for recursive subnet selection. Therefore, a cycle can be cut by splitting the domain of one of the involved variables, until the specified maximal diameter is reached. We call a constraint net node cyclic if it is involved in a cycle. For each cyclic variable, the number of cyclic neighboring constraints in the net can be used to estimate the number of cycles potentially cut when choosing that variable as a decision variable. The more cycles a variable potentially cuts, the more preferable it is. This criterion helps to reduce the needed recursion depth perceivably. In our experience, it is worth the additional computation effort which grows linearly with respect to the net size.

3.3 An ATMS Interface

To integrate the branch and prune algorithm into a conflict directed search for diagnostic candidates, some kind of dependency tracking is needed. Several dependency tracking strategies have been published to improve the efficiency of solvers for constraint satisfaction problems (CSP). *Conflict-based Backjumping* [15] prunes out parts of the search space based on conflicts. In addition, *Dynamic Backtracking* [7] supports prediction sharing across different points of the search space. Dependency tracking for combinations of local propagation techniques and dichotomic search strategies has been investigated by Jussien (see [12] and [13]).

The branch and prune algorithm differs from the CSP solving strategies mentioned above in two points: It does not stop after a solution has been found[7], and it computes unions of variable ranges obtained in different branches. Those ranges denote hulls around all solutions of the constraint system. But how can we justify those hulls with respect to the context assumptions of the corresponding recursion depth? In the following, we sketch an interface to a basic ATMS [3] extended by disjunction handling techniques. Interfaces to more efficient TMS can be implemented analogously.

As usual, ATMS nodes are composed of three parts $\langle$data, label, justification$\rangle$. The label consists of a set of assumption sets (called environments) which represents a disjunction of conjunctions. In our context, data is used to store range assignments to variables and assumptions. ATMS assumptions are introduced for those range assignments which characterize the diagnostic candidate. Horn justification statements (noted $\alpha_1, \ldots, \alpha_n \Rightarrow \beta$) suffice to inform the ATMS about new range assignments resulting from local propagation steps during pruning. If an empty range is deduced during pruning, a nogood (noted $\mathrm{nogood}\{\alpha_1, \ldots, \alpha_n\}$) is

[6] To cut cycles as fast as possible, one should prefer small diameters. However, if the constraint system contains directed elements (constraints, whose projections cannot be computed for every variable), the proposed preference improves the reliability of simulation results.

[7] The algorithm can easily be modified to do so, if only a consistency check is needed.

asserted to the ATMS which represents the clause $\alpha_1 \wedge \ldots \wedge \alpha_n \rightarrow \bot$. Range assignments resulting from a split operation do not have a Horn justification. But for dependency tracking, the relationship between the original range assignment $x = D$ and the two split assignments $x = D^1$ and $x = D^2$ is important. The logical formula representing the relationship is an implied disjunction $x = D \rightarrow x = D^1 \vee x = D^2$. In [2] several encodings for complex implications are proposed. To that purpose, a new statement (noted choose$\{\alpha_1, \ldots, \alpha_n\}$) is introduced which can be asserted to the ATMS at any time, and which represents a primitive disjunction $\alpha_1 \vee \ldots \vee \alpha_n$. All encodings of complex implications use a combination of assumptions, justifications, nogoods, and primitive disjunctions.

In our context, the encoding "with no assumption disjuncts" is the most efficient, because literals resulting from splitting will rarely occur in different disjunctions. Following that encoding, for each clause $x = D \rightarrow x = D^1 \vee x = D^2$ resulting from a split operation, two new assumptions A^i are introduced and the following expressions are asserted to the ATMS:

$$\begin{aligned}&\text{choose}\{A^1, A^2\}\\ &x = D, A^i \Rightarrow x = D^i \quad \text{for all } i \in \{1, 2\}\end{aligned}$$

If no conflict is found in either of the branches, for each variable y the union D of both result ranges D^i for y is computed and the following clauses asserted to the ATMS:

$$y = D^i \Rightarrow y = D \quad \text{for all } i \in \{1, 2\}$$

This knowledge allows the ATMS to backtrack the dependencies between the assumptions which define the context with respect to the current recursion level, and the ranges obtained. For this purpose, the standard label update has to be extended by a hyperresolution rule:

$$\frac{\begin{array}{l}\text{choose}\{A^1, A^2\}\\ \langle \text{data}, \lambda, \text{justification}\rangle\\ \text{nogood}[\{A^i\} \cup \alpha_i] \text{ or } \{A^i\} \cup \alpha_i \in \lambda \text{ and } A^{3-i} \notin \alpha_i \quad \text{for all } i\end{array}}{\langle \text{data}, \{\alpha_1 \cup \alpha_2\} \cup \lambda^*, \text{justification}\rangle}$$

λ^* is obtained from λ by removing all supersets of $\{\alpha_1 \cup \alpha_2\}$.

Finally, another hyperresolution rule is needed to backtrack nogoods in case both branches resulted in a conflict:

$$\frac{\begin{array}{l}\text{choose}\{A^1, A^2\}\\ \text{nogood}[\{A^i\} \cup \alpha_i] \text{ and } A^{3-i} \notin \alpha_i \quad \text{for all } i\end{array}}{\text{nogood}[\alpha_1 \cup \alpha_2]}$$

The diagnostic system can benefit from dependency tracking in two ways: First, the candidate generation process is focussed by conflicts obtained from candidates which failed to explain the given situation. Second, the efficiency of prediction is increased by sharing prediction results between different contexts.

Results can even be shared between different branches during one constraint solving task. To this end, the branch and prune algorithm accesses the ATMS after pruning between the first and second recursive call:

- If the first branch resulted in a conflict, the current context should be checked for consistency. If the conflict is independent of the split assumptions introduced in the first branch, the second branch need not be investigated. This improvement corresponds to Conflict-based Backjumping.
- Otherwise, for each refined range assignment, the label should be checked. If the label contains an environment which itself does not contain any of the newly introduced split assumptions, the corresponding range assignment can be reused for the second branch. This improvement corresponds to some aspects of Dynamic Backtracking.

4 Experimental Results

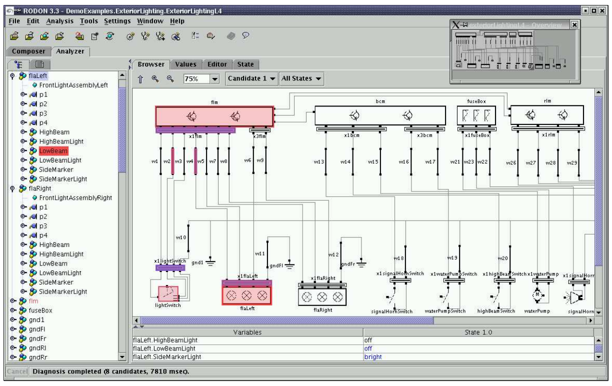
We have integrated a Java implementation of the concepts presented in this paper, into the commercial model-based analysis tool RODON. The first version of the branch and prune algorithm was available about one year ago. Since then, lots of improvements have been added, especially with respect to performance, triggered by the experience gained from modeling real-life systems. Today, the algorithms are in daily use to automate the generation of diagnostic knowledge based on construction data. For example, several car series produced by Daimler Chrysler are equipped with onboard diagnostic knowledge bases generated by RODON.

To demonstrate the usage of the branch and prune approach in our application focus, we have chosen an anonymized model of an automotive exterior lighting system (see Fig. 7). 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 relais. Altogether 86 physical components are included, defining 143 single faults. The behavior model is built out of 1816 variables and 1565 constraints.

To cover the self diagnostic capabilities of todays electronic control units, the driver models themselves contain several algebraic loops including diodes, and nonlinear resistors. Those loops overlap with loops built of sensors, fuses and consumers. With pure local propagation, not even a conflict can be found when checking the candidate "System OK" with respect to the symptom "diagnostic trouble code xy set". The time needed to compute a steady state for the given system strongly depends on the operational state and the analyzed fault. The average time is about half a second on a 700 MHz Pentium. 82% of the time is spent on local propagation, 6% on constraint net analysis and 12% on storage management. Computing a diagnosis takes between three and ten seconds.

Among the improvements we added to the base algorithm, the most significant is the graph decomposition step. If this step is disabled, the diagnostic algorithm fails to return a satisfactory result in reasonable time for the described model. The performance of the algorithm also strongly depends on the criteria used for decision variable selection. For example, if decision variable selection is not restricted to cyclic variables, the average time to compute a diagnosis in our example rises to several minutes. We have tested lots of selection strategies with models of very different size and structure. All criteria proposed in Section 3.2 proved to be useful. However, the optimal combination of them depends on the characteristics of the model.

The benefits from prediction sharing between different branches depend on the success of the graph decomposition step. The smaller the identified subnets, the less the gains from prediction sharing. In our example, the benefits are not significant. In some diagnoses the gains are completely consumed by the range initialization overhead, in other diagnoses about 10 percent of the propagation time can be spared.


RODON 3.3 - DemoExamples.ExteriorLighting.ExteriorLightingL4
File Edit Analysis Tools Settings Window Help
Composer Analyzer
Browser Values Editor State
75%
Candidate 1
All States

Variables	State 1.0
flaLeft.HighBeamLight	off
flaLeft.LowBeamLight	off
flaLeft.SideMarkerLight	bright

Diagnosis completed (8 candidates, 7810 msec).

Figure 7. The model-based analysis tool RODON

5 Conclusion and Related Work

The limitations of pure local propagation have shown to be a significant drawback of current model-based diagnostic systems. The incompleteness of this kind of inference has prevented a wide spread success of model-based technology in industrial applications. Often, overestimation resulting from algebraic cycles can be limited by tuning the model manually, but this kind of tuning increases the production costs of models and counteracts the “no function in structure” principle.

Methods combining local propagation on quantitative constraints with dependency tracking have a long tradition in the application area of analog circuit diagnosis. Early work concentrated on search space reduction and efficiency aspects. In [4] a diagnostic system called SOPHIE III is presented, which is specialized on single faults in DC circuits. It uses conflicts as well as so called corrobations to direct the search for diagnostic candidates. Further search space reduction is achieved by distinguishing primary and secondary assumptions during dependency tracking. The incompleteness of local propagation is identified as a serious problem. As a pragmatic solution, de Kleer and Brown propose to add circuit specific knowledge in the form of heuristic rules to the knowledge base.

Another diagnostic system called Skordos is discussed in [8]. It addresses the difficulty of managing the tremendous number of possible predictions in quantitative value set propagation algorithms. Value sets are represented by inequalities. A process called hibernation delays propagation of useless combinations of inequalities, until they become relevant for diagnostic reasoning.

In recent years, several contributions have addressed the problem of solving cyclic constraint structures by net transformation algorithms (see e. g. [9] 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 [6]. The problem with those transformations is that efficient algorithms exist only for several more or less restricted classes of constraints.

For cyclic constraint systems in which transformation algorithms fail (at least partially), the branch and prune algorithm provides an interesting alternative. It is not restricted to a special domain or constraint language. The relations forming the behavioral model are not required to have any further properties (such as continuity or differentiability) besides the evaluation function. It provides for all model variables complete ranges which approximate globally consistent domains.

For application in model-based diagnosis, we have shown how an ATMS can be supported.

In our experience, the simulation time is acceptable in most cases, at least for offboard use. However, there are some situations, where the performance could be further improved:

- In underdetermined systems, the resulting number of partitions can grow exponentially with the recursion depth, causing high simulation effort without significant narrowing. Additional termination criteria seem to help in this case.
- Cyclic structures in ill-conditioned systems can slow down convergence of local propagation. Since the branch and prune algorithm uses local propagation for pruning, this effect directly reduces the simulation performance. For such cases, we think that the cooperation of different solvers, possibly involving even computer algebra methods, might improve performance considerably.

This will be subject to future work.

REFERENCES

[1] Christian Bliek, Bertrand Neveu, and Gilles Trombettoni, ‘Using graph decomposition for solving continuous CSPs’, *Lecture Notes in Computer Science*, **1520**, 102+, (1998).

[2] J. de Kleer, ‘Extending the ATMS’, *Artificial Intelligence*, **28**(2).

[3] J. de Kleer, ‘An assumption-based TMS’, *Artificial Intelligence*, **28**, 127–162, (1986).

[4] J. de Kleer and J.S. Brown, ‘Model-based diagnosis in SOPHIE III’, (1992).

[5] J. de Kleer and B. C. Williams, ‘Diagnosing multiple faults’, *Artificial Intelligence*, **32**, 97–130, (1987).

[6] Rina Dechter, ‘Bucket elimination: A unifying framework for reasoning’, *Artificial Intelligence*, **113**(1-2), 41–85, (1999).

[7] Matthew L. Ginsberg, ‘Dynamic backtracking’, *Journal of Artificial Intelligence Research*, **1**, (1993).

[8] David Jerald Goldstone, ‘Controlling inequality reasoning in a TMS-based analog diagnosis system’, 206–211, (1992).

[9] 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*, pp. 150–166, (2001).

[10] 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).

[11] E. Hyvönen, *Constraint Reasoning with Incomplete Knowledge*, Ph.D. dissertation, Helsinki University, 1991.

[12] Narendra Jussien and Patrice Boizumault, ‘Dynamic backtracking with constraint propagation - application to static and dynamic CSPs’, in *CP97 Workshop on the Theory and practice of dynamic constraint satisfaction*, (1997).

[13] Narendra Jussien and Olivier Lhomme, ‘Dynamic domain splitting for numeric CSPs’, in *Proceedings of the 13th European Conference on Artificial Intelligence*, (1998).

[14] J. Mauss, V. May, and M. Tatar, ‘Towards model-based engineering: Failure analysis with MDS’. Lecture at the ECAI 2000 (Berlin), 2000.

[15] Patrick Posser, ‘Hybrid algorithms for the constraint satisfaction problem’, *Computational Intelligence*, **9**(3), 268–299, (August 1993).

[16] P. Struss and O. Dressler, ‘Physical negation: Integrating fault models into the general diagnostic engine.’, in *Proceedings 11th Int. Joint Conference on Artificial Intelligence (IJCAI ’89)*, pp. 1318–1323, (1989).

[17] M. Tatar, ‘Diagnosis with cascading defects’, in *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI’96), Budapest (Hungary)*, pp. 511–518, (1996).