Combining the Lazy Label Evaluation with Focusing Techniques in an ATMS

Mugur M. Tatar[1]

Abstract. For large problems the ATMS often becomes the main resource consumer in any reasoning system. We propose an architecture (the 2vATMS) that combines the advantages of two techniques aiming to reduce the complexity of the ATMS tasks: *(i)* focusing on a few possible worlds, which abandons the requirement of global label completeness; and *(ii)* the lazy label evaluation, which avoids the label update work as long as there is no evidence that it would be relevant for the problem solver. The 2vATMS integrates two logically dependent views on data: The first view stores by which focus environments a node is supported. This information is easy to maintain and corresponds to the information provided by a set of simple monotonic TMSs. The second view provides more detailed information, i.e. about the dependence on the assumptions. The labels of the detailed view are similar to the ones computed by a focused ATMS, but their maintenance is ensured only by request. The content of the first view helps to achieve a very tight control of the environment propagation in the detailed view. Moreover, the problem solver has the opportunity to decide when the information offered by the monotonic TMSs is sufficient, and when the switch to the more detailed view is valuable.

1 INTRODUCTION

The assumption based truth maintenance systems [1] are instruments used to record the dependence of inferred data on a set of hypotheses. The ATMS is to be used in conjunction with a *problem-solver* which is responsible for the inferences communicated to the ATMS. The ATMS records the inferences as purely propositional material implications, also called *justifications*, denoted by e.g. $P_1 \wedge P_2 \wedge ... P_k \rightarrow Q$. The ATMS also assigns a *node* to every proposition the problem-solver reasons about. There are distinguished nodes called *assumptions*, specified by the problem-solver. A set of assumptions is an *environment*. The ATMS *labels* all nodes with the complete set of minimal (w.r.t. set inclusion) and consistent environments from which they are derivable. There is a distinguished ATMS node ($\perp$) denoting contradiction. The environments supporting $\perp$ are called *nogoods* and are removed from all node labels.

The success of the ATMS is due, among other motives, to providing a very efficient way of searching in multiple belief spaces. However, the attempt of the basic ATMS to maintain the *complete, minimal* and *consistent* supports for *all* it's nodes at *all* times leads to prohibitive computational costs [8]. Relaxing these requirements may reduce the computational costs. Two successful approaches aiming to improve the performance of the ATMS have been reported[2]:

(1) Focus the ATMS [6,8]. This approach relies on the ability of the problem-solver to identify a set of more ”plausible” possible worlds. These most plausible possible worlds can be communicated to the ATMS as a set of *focus environments*. The focusing can be done both at the problem-solver's level and within the ATMS. At the problem-solver's level, only those inferences are performed which hold in at least one focus environment. Within the ATMS, only those sets of assumptions which are contained in at least one focus environment are propagated along the justifications. The propagation of those sets of assumptions outside the current focus is temporarily blocked. Focusing the label propagation, the global label completeness and consistency is (temporarily) abandoned, but it will be guaranteed in a weaker sense, i.e. with respect to the set of focus environments:

Consistency w.r.t. the focus environments: Every environment which is part of a node's label and which is a subset of at least one focus environment is consistent.

Completeness w.r.t. the focus environments: Every set of assumptions which supports the derivation of a node, and which is a subset of at least one focus environment is either in the node's label or is a superset of some environment from the node's label.

(2) Give up the idea that *all* the nodes should have the labels updated at *any* time. This second approach assumes that, at any time, there is a significant number of nodes whose labels the problem-solver is not interested in. The LazyRMS [9,10] computes a node label only by request. The addition of a justification no longer triggers the label update, but the LazyRMS marks those nodes whose labels might be affected by the addition of new justifications. So, if a node is marked, then also all its followers in the network of justifications will get marked. The mark indicates that the node's label must be recomputed before usage. Unmarked nodes have complete labels. When the problem-solver asks for the label of a marked node, the following process starts: first, the marked antecedent nodes of the queried node in the justification net are detected and their labels are recursively computed; then the changes in the antecedents are propagated to the queried node.

The focusing technique and the lazy label evaluation reduce the computational effort in two distinct ways: The focused ATMS computes *shorter labels*; the lazy ATMS computes *fewer labels*. A natural idea would be to combine the characteristics of

[1] Computer Science Department, Technical University of Cluj-Napoca, Romania. Current mailing address: Daimler-Benz, Systems Technology Research, Alt-Moabit 91b, 10559 Berlin, Germany. E-mail: *tatar@dbresearch–berlin.de*.

[2] It is known that the algorithms underlying the basic ATMS have an exponential worst case complexity. The techniques presented in this paper aim at improving the average case complexity only.

© 1994 M. M. Tatar
ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

the lazy and of the focused ATMS. As also pointed out in [10], the combination of these two techniques is not straightforward, mainly because the focusing at the problem-solver's level makes the lazy label evaluation pointless : The problem-solver is interested only in the data which hold under the current focus, but if it had to *query all* the labels in order to find out this, then the assumption underlying the application of the lazy ATMS would be refuted.

In order to solve this apparent incompatibility we propose an ATMS architecture which maintains two views on data (we will call it 2vATMS). The views share the nodes and the justifications, but attach distinct labels to the nodes. The first view is used only to find out whether a node holds in the current focus or not. For each focus environment it contains a mark indicating if the node holds in the context defined by that focus environment. The computations in this view are performed immediately, when a justification is added. The labels of the second view are similar to the labels computed by the focused ATMS, but they are computed only by request. We will call the first view as the *focus view* and the second one as the *detailed view.* Based on the the information stored in the focus view a very tight control of the environment propagation in the detailed view is achieved.

The focus view is analogous to a set of single-context monotonic TMSs[3], one TMS for each focus environment. In many cases this view provides all the information that is relevant for the problem-solver. When this is not the case, the 2vATMS gives the problem-solver the opportunity to switch to the more detailed view.

2 THE 2vATMS

There are two basic suppositions underlying the ideas of the 2vATMS: *(a)* that, at any moment, the problem-solver is interested only in a *small* set of possible worlds, communicated as a set of focus environments to the ATMS; and *(b)* that the problem-solver is not interested all the time in the node's detailed dependence on the assumptions, but is interested to know which nodes are supported by the current focus. At any time, however, the problem-solver may express the interest in the detailed label of a node.

The 2vATMS attaches two labels to the nodes: the *f-label*, and the *d-label*. The d-labels logically correspond to the labels computed by the focused ATMS, but their maintenance is delayed as long as there is no evidence that they could be relevant for the problem-solver. The f-labels have no correspondent in the basic ATMS. The f-labels trace the dependence of the nodes on the *focus contexts*. Each focus environment has an associated identifier in the focus view. The f-labels contain the identifiers of those focus environments which support the node derivation (in the following we will say that an f-label contains a certain focus environment, meaning that the f-label contains in fact the *identifier* of that focus environment). So, if f_i is a focus environment, then the f-label of a node contains f_i iff that node holds in the context defined by f_i. The views are logically related, i.e. the focus view

[3] A monotonic TMS labels with IN the nodes which hold in the current context.

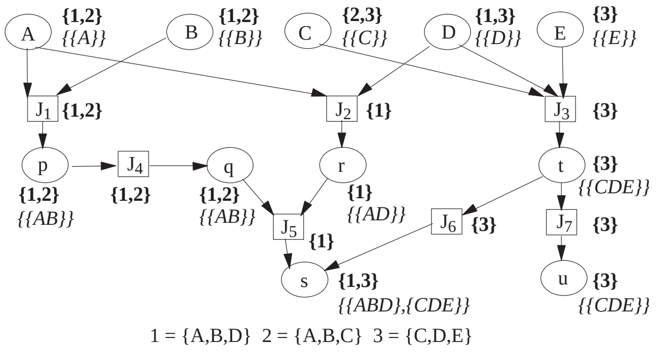

Figure 1.

can be regarded as an abstraction of the detailed view. One consequence of this is that the computation of the focus view is also less expensive than the computation of the detailed view. The 2vATMS maintains by default only the focus view. However, if the problem solver explicitly queries the d-label of a node, the detailed view must be *partially* updated, but, as we shall see, the information from the focus view can be used to control the computations required. In the following, we describe how the f-labels are computed and how the lazy computation in the detailed view is performed.

The *f-label of a justification* is the set intersection among the f-labels of justification's antecedents. It is easy to see that the consequent of a justification is supported by each focus environment from the justification's f-label. The f-labels are computed as follows: *(i)* the f-label of a *premise* is the total set of focus environments; *(ii)* the f-label of an *assumption* contains those focus environments which mention that assumption; *(iii)* for a different node *n*, the f-label is the union of the f-labels of the justifications that have *n* as consequent.

Example 1. figure 1. contains a small network of dependencies with fully determined f-labels and d-labels. *A, B, C, D, E* are assumptions and *p, q, r, s, t, u,* are derived nodes. There are 7 justifications depicted, e.g. $J_1 = A, B \rightarrow p;$ $J_5 = q, r \rightarrow s;$ an so on. The set of focus environments used here is: *1: {A, B, D}, 2: {A, B, C}, 3: {C, D, E}*. The d-labels are shown in italics, the f-labels in bold face. The f-label of *s* is the union of J_5's and J_6's f-labels.

The size of the f-labels can be kept bound if the problem-solver uses at any time a limited set of focus environments, e.g. two bytes are sufficient for working with a focus of size 16. Except the assumptions and the premises, which have a complete d-label at any time, the label completeness in the detailed view depends on the sequence of justifications added and on the queries posed. In order to express the relationship between the f-label and the d-label of a node *at a given moment*, the structure of a node includes another field – the *f-status*: It contains the set of focus environments w.r.t. which the node's d-label *might not be complete*. The f-status is always a subset of the f-label, and plays a similar role as the mark which the LazyRMS attaches to its nodes. A node with an empty f-status has a completely determined d-label w.r.t. the current focus. Two processes affect the content of the f-status fields: *(a)* the addition of a new justification; *(b)* a d-label query. The addition of a justification causes the addition of new elements to the f-labels and to the f-statuses, while the d-label queries cause the removal of some elements from the f-statuses.

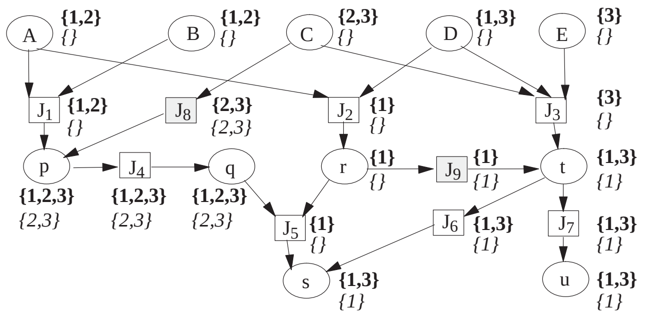

Figure 2.

It is useful to define also the f-status propagated by a certain justification: A *justification's f-status* contains the set of focus environments propagated by the justification w.r.t. which the d-label completeness for the consequent node cannot be guaranteed. If the d-label of an antecedent node of the justification *J* is not complete w.r.t. a focus environment f_i, and if f_i *is propagated* by *J*, then the d-label completeness for *J*'s consequent cannot be guaranteed w.r.t. f_i. Also, when a justification is newly added, the d-label completeness w.r.t. the focus environments propagated by the justification cannot be guaranteed for the consequent node, since the addition of a justification does not trigger the d-label update. Thus *(1)* when a justification is added its f-status is set equal with its f-label; *(2)* the f-status of a node is the set union among the f-statuses of the justifications having that node as consequent; *(3)* If the f-status of an antecedent node of a justification *J* is modified such that the set *F* is incrementally added to it, *J*'s f-status is set to the union of its old value and the intersection between *F* and *J*'s f-label (i.e. $f\text{-}status(J) := f\text{-}status(J) \cup (F \cap f\text{-}label(J))$). The update of the f–statuses and of the f–labels is performed simultaneously and in an incremental manner.

Example 2. Consider the network of dependencies depicted in fig. 1., and assume that the d-label of every node is completely determined (this is a rather unrealistic state for the 2vATMS since it assumes that all the node d-labels have been queried). Consequently, the f-status sets, not depicted in fig. 1., are empty for all the nodes and justifications. Suppose we add two new justifications: $J_8 = c \rightarrow p$, and $J_9 = r \rightarrow t$. This will change the f-labels and the f-statuses, but will not affect the detailed view. The resulting state of the network is depicted in fig. 2. (the f-labels have a bold face and the f-statuses are shown in italics; the d-labels are not shown). The f-statuses of J_8 and J_9 are identical with their f-label. The addition of J_8 caused the update of the f-label and of the f-status of *p*, J_4 and *q*. The f-status *{2,3}* did not propagate from *q* through J_5 because the intersection with J_5's f-label is empty. The addition of J_9 caused the update of the f-status at: *t*, J_6, *s*, J_7 and *u*. Note that when a new justification is added, the LazyRMS marks *all* the followers of the consequent – meaning that their label could have been modified by the operation. The 2vATMS, however, determines much more precisely the followers where the label completeness w.r.t. the focus might be violated. For instance, *s* is a follower of *p* in the network of dependencies but the label completeness w.r.t. the focus cannot be affected by the addition of J_8 (only the addition of J_9 caused the update of the f-status of *s*).

When the problem-solver queries the d-label of a node, a process very much alike to that from the LazyRMS [9,10] is initiated, but the information from the focus view can be used to reduce the amount of computation required. The focus view is used to selectively determine only those justifications that are *relevant for the current query.* This significantly reduces the number of nodes whose d-labels must be updated. In order to ensure the completeness of the d-label for a node *n* w.r.t. a focus environment f_i mentioned in *n*'s f-status, we consider only those justifications for *n* whose f-status contain f_i; for each such justification the completeness w.r.t. f_i for the antecedent nodes is recursively ensured[4]. Finally, the incremental change from the antecedents' d-labels is propagated through the relevant justifications. Thus, a very tight control of the propagation in the detailed view is achieved: The propagation of those environments outside the current focus is temporarily blocked by storing them in a "blocked label" as in the focused ATMS. Moreover, the propagation of those environments which *are implied* by the current focus is blocked at the level of those individual justifications which *are not needed* for answering the current query. After a d-label is updated such that it is complete w.r.t. a focus environment, the focus environment is removed from the f-status.

The 2vATMS maintains an empty d-label and f-label for the contradiction node *(* $\perp$ *)*. Any time when something propagates to the f-label of the contradiction node, at least one focus environment is inconsistent, and the d-label of the contradiction node is automatically computed in order to find the *minimal* nogoods. Afterwards, the focus and the detailed views are updated in order to restore consistency. This implies that even if the d-label *completeness* w.r.t. the focus is not guaranteed, at least not until the node is queried, the *consistency* of the d-labels w.r.t. the focus is at all times guaranteed.

Example 3. Consider fig. 2., and assume that the problem-solver asks for the d-label of *s*. Since the intersection between the f-status of *s* and J_5's f-status is empty, J_5 is not relevant for the query. Only J_6 affects d-label's completeness w.r.t. the focus environment 1, so the d-label completeness w.r.t. the set {1} is recursively ensured for J_6's antecedents (i.e. for *t*). Finally the environments which were not propagated through J_9 are restarted, but the propagation will be restricted to the justifications involved in the current query: The incremental change from *t*'s d-label will be propagated through J_6, but not through J_7. After this process the set {1} is removed from the f-status of J_9, *t*, J_6, and *s*. Note that, without the information from the focus view, a purely lazy ATMS [10], would have recurred to update also the label for *q* and *p*, and would have also had to consider the justifications J_5, J_4, and J_8 which were not relevant for this query.

3 CYCLIC DEPENDENCIES

The current implementation of the 2vATMS does not employ the most parsimonious control technique when cycles in the justifications that *are relevant for the current query* are encountered. In such a case the completeness w.r.t. the queried set of focus environments is ensured for *all* the nodes connected by the relevant justifications (a similar approach is followed also by the LazyRMS [10]). In networks with high connectivity this can in-

[4] In fact, special care must be taken when cycles in the dependencies are encountered.

crease significantly the work of the 2vATMS. Hopefully there is in fact no need to ensure the label completeness w.r.t. the queried focus environments for all members of such cycles. It is always the case that the cycles in the dependencies can be "broken" when we are interested only in a specific node's label.

The justification J_5 from fig. 3 is not relevant when comput-

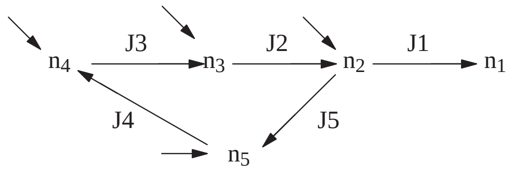

Figure 3.

ing n_1's or n_2's d-label. It is always the case that, when computing the label of a node *n*, all the justifications having *n* as antecedent can be ignored (thus J_5 is "ignored" during the recursive computation of n_2's d-label). Breaking the above cycle at J_5 has the effect that the d-label of n_1 or of n_2 can be correctly computed without enforcing the label completeness for n_3, n_4 and n_5.

4 CONTEXT CHANGING

Changing the context within the focus worlds is performed at no cost in the 2vATMS. However the addition of new focus environments entails the addition of new elements to the f–labels. Although the computations in the focus view are fairly cheap and, usually, this overhead is balanced by the more restrictive control of the computations in the detailed view, it can become significant in the cases which require intensive context changing. Based on the observation that usually the focus environments do not differ dramatically in between, the following heuristics can help reduce the costs of context changing:

(*i*) Incremental changing of the focus: When a focus environment *f* becomes inconsistent, do not remove immediately its associated identifier from nodes' f-labels. Instead just mark that identifier as "invalid". When a new environment *g* is added to the focus assign it the "invalid" identifier of *f* and update only the followers of the assumptions from the set: $(f \setminus g) \cup (g \setminus f)$. Since we expect that *f, g* do not differ dramatically, a significant amount of work can be saved because there is no need to relabel the nodes which depended only on the assumptions from the set $f \cap g$.

(*ii*) If the problem-solver decides to add several focus environments into the focus at a given time, then it is better to add the focus environments in batches. The nodes depending only on the common assumptions are processed only once this way.

The above two heuristics can be used in conjunction, of course.

5 THE APPLICATION OF THE 2vATMS IN MODEL BASED DIAGNOSIS

Model based diagnosis [2,3] seeks to identify the malfunctioning components of a device. The only information to be used is how the components are connected to each other, how they behave, and a set of observations about the real device. The diagnosis must find those assignments of modes of behavior to the components which are consistent with the observations. The diagnosis is driven by the contradictions identified between the assumed modes of behavior and the observations. Such a diagnostic engine can be implemented using a value-propagating engine and an ATMS. The ATMS is used to keep track of the predicted values' dependence on the assumptions, and to identify the minimal nogoods supporting the contradictions.

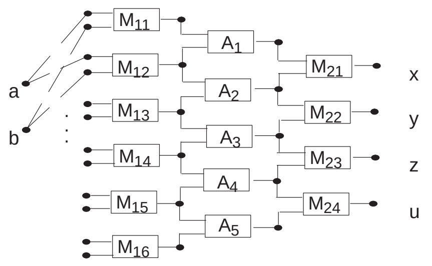

Figure 4.

Usually, there are too many diagnoses which can account for the observations. In focused diagnosis, we are interested in finding only the most *plausible* ones, according to some criteria [4,7,11]. Without the use of focusing, an exponential number of predictions can be computed because combinations of predictions from different modes need be computed. Even with focused value propagation, in a basic ATMS, the number of environments propagated grows exponentially. The use of a focused ATMS [6,8] significantly alleviates these problems, but such an ATMS attaches labels for all the values propagated, although usually only a small part of these labels might be interesting for diagnosis. The 2vATMS will not spend effort to compute/update the d-labels of those predictions not involved in any conflict relevant for the current focus.

Example 4. Suppose we are diagnosing a pyramid of multipliers and adders like that from fig. 4. Assume we supply the values of the inputs of the multipliers M_{1i}. Values will be predicted and propagated for all the lines of the circuit. As far as no conflict is discovered, the 2vATMS does not attach any d-label to them. Now suppose the value of *x* is measured, and it is found incorrect. The 2vATMS discovers the same conflict as the basic ATMS would have discovered, i.e. $\{ ok(M_{11}), ok(M_{12}), ok(M_{13}), ok(A_1), ok(A_2), ok(M_{21}) \}$, but in the 2vATMS only the values predicted at the output of the components from the above conflict will have their d-label computed. The number of labeled values would remain constant in the 2vATMS, while it would always increase in the focused ATMS, even if the base of the three-level pyramid of components from fig. 4 had 10,000 components, instead of 6.

We have implemented a prototype of the 2vATMS and we have run a series of preliminary tests in model based diagnostic tasks. Our diagnostic engine never queried a 2vATMS node. The discovery of the minimal conflicts is sufficient for finding the diagnoses, while the information from the f-labels is sufficient for focused value propagation.

Table 1. summarizes some of the results obtained while diagnosing the circuit from Fig. 4. Every component was characterized by 7 modes of behavior: *ok* / probability 0.75 - the correct behavior; *s1* / 0.1 - output stuck at 1; *s0* / 0.05 - output stuck at 0; *l* / 0.04 - output equal with the left input; *r* / 0.04 - output equal with the right input; *s* / 0.018 - output equal with the correct result shifted with one bit to the left; *u* / 0.002- unknown failure. In all the cases described in Tab. 1. the diagnostic engine focused only on the first most probable diagnoses, but several diagnoses were added to focus if they were equally probable. The table compares the performance of using the focused ATMS vs. the 2vATMS (see [6,8] for comparisons between the focused ATMS and the basic one). Column 3 compares the total size of the environment database at the end of diagnosis (the nogoods were also counted in this number). Column 4 compares the average length of the labels, computed as the total label length divided by the total number of ATMS nodes (the use of the 2vATMS does not reduce the number of the nodes and the number of the conflicts discovered vs. the use of the focused ATMS). Column 5 compares the total amount of time[5] *spent within the ATMS* during the diagnosis process. *a* and *b* are linked to the left respectively the right input of the multipliers M_{1i}.

6 DISCUSSION

The use of the simple basic ATMS leads to combinatorial explosion very soon. Additional control mechanisms need to be found before trying to scale up to larger tasks. Relaxing the requirement of global label completeness and the requirement that all the nodes should have the label updated at any time, the amount of work done within the ATMS can be reduced. We have presented an architecture which combines the lazy label evaluation [10] with the focusing techniques [6,8]. We can also regard the 2vATMS architecture as *tightly integrating* a set of single–context TMSs with a focused-lazy ATMS. Our preliminary tests, with an unoptimized prototype, show significant reductions of the time and memory consumed by the ATMS in diagnostic tasks, even on small examples.

Other papers also observed that the focused ATMS performs sometimes non-relevant work in diagnosis (cf. [5,7]). In [5] two distinct TMSs are used in conjunction: a (single-context) LTMS, and a focused ATMS. As opposed to our architecture, the LTMS and the focused ATMS are loosely coupled in [5]. In the 2vATMS the content of the focus view is used to control the environment propagation. Also, by maintaining *a set* of single-context TMSs, the 2vATMS performs the context switching within the focus worlds at no cost.

[7] suggests to use special *all-ok* assumptions, representing a conjunction of *ok* assumptions, and argue that this help reduce the effort of ATMS node labelling during diagnosis. The identifiers attached to the focus environments in the 2vATMS can be regarded as single assumptions replacing the sets of assumptions representing the focus environments.

Our architecture still guarantees the consistency of the d-labels w.r.t. the current focus at any moment, since the whole environment database is checked for consistency immediately after the discovery of a new minimal nogood. One alternative, which we have not tested, is to delay also the d-label consistency check until the d-label is needed, as the LazyRMS [10] does.

ACKNOWLEDGEMENTS

The paper owes much to the discussions with Gerry Kelleher. Sebastian Iwanowski provided extensive comments on earlier drafts of this paper. I further want to thank the anonymous referees and my thesis supervisor, Ioan A. Letia.

[5] These tests were run on a PC286 under DOS. The 2vATMS and the diagnosis engine were implemented using C++. The prototype did not embed all the ideas presented in this paper in the optimal way.

Table 1. □ focused ATMS ◊ 2vATMS

Input (a = 2 b = 3)				Diagnosis	Environments			Avg. label length			Time in seconds		
x	y	z	u		□	◊	*ratio*	□	◊	*ratio*	□	◊	*ratio*
144	36	27	108	{r(M_{14}), r(A_3)}	382	166	*2.3*	1.91	1.26	*1.6*	8.18	2.36	*3.5*
144	108	81	108	{r(M_{14})}	263	42	*6.3*	3	0.8	*3.8*	4.23	0.5	*8.5*
12	12	12	12	{s1(A_2), s1(A_4)}	206	85	*2.4*	1.69	1.18	*1.4*	3	1	*3*
84	144	96	64	{s1(M_{11}), l(M_{15})}	342	67	*5.1*	2.4	0.85	*2.8*	6.8	0.77	*8.8*

REFERENCES

[1] de Kleer, J., *An Assumption Based Truth Maintenance System*, Artificial Intelligence 28, (1986), p. 127–162.

[2] de Kleer, J., Williams, B., *Diagnosing Multiple Faults*, Artificial Intelligence 32 (1987), p.97–1307.

[3] de Kleer, J., Williams, B., *Diagnosis with Behavioral Modes*, Proc.IJCAI 1989, p. 1324–1330.

[4] de Kleer, J., *Focusing on Probable Diagnoses*, Proc. AAAI 1991, (1991)

[5] de Kleer, J., *Optimizing Focusing Model-Based Diagnoses*, Proc. 3rd International Workshop on Principles of Diagnosis, 1992.

[6] Dressler, O., Farquhar, A., *Putting the Problem Solver Back in the Driver's Seat: Contextual Control of the ATMS*, ECAI TMS Workshop, 1990.

[7] Dressler, O., Struss, P., *Model–based Diagnosis with the Default–based Diagnosis Engine: Effective Control Strategies that work in practice*, Siemens Report ARM-1-94, 1994.

[8] Forbus, K., de Kleer, J., *Focusing the ATMS*, Proc. AAAI 1988, p. 193–198.

[9] Kelleher, G., van Rij, T., *The Application of Lazy RMS in Automated Diagnosis*, Proc. ECAI 1992 Workshop on Applications of RMS.

[10] Kelleher, G., Gaag, L., *The LazyRMS: Avoiding work in the ATMS*, Computational Intelligence: An Int. Journal, Aug. 1993, vol. 9., no. 3., p. 239–253.

[11] Tatar, M., Iwanowski, S., *Efficient Candidate Generation in a Model-Based Diagnostic Engine*, Daimler-Benz Technical Report, 1994.