Proceeding of the 7th International Symposium on Artificial Intelligence, Robotics and Automation in Space: i-SAIRAS 2003, NARA, Japan, May 19-23, 2003

An Advanced Model-Based Diagnosis Engine

Amir Fijany, Farrokh Vatan, Anthony Barrett, Mark James, and Ryan Mackey
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109, USA
{Amir.Fijany, Farrokh.Vatan, Anthony.C.Barrett, Mark.James, Ryan.M.Mackey)@jpl.nasa.gov

keywords Model-Based Diagnosis, Hitting Set Problem, Integer Programming, Boolean Satisfiability

Abstract

The number of Earth orbiters and deep space probes has grown dramatically over the past decade, and this trend is expected to continue. This rate of growth has brought a new focus on autonomous and self-preserving systems, all of which depend on fault diagnosis. Although diagnosis is needed for any autonomous system, current approaches tend to be "ad-hoc," inefficient, and incomplete. Systematic methods of general diagnosis exist in literature but all suffer from two major drawbacks that limit their practical applications. First, they tend to be large and complex and hence difficult to apply. Second and more importantly, they rely on algorithms with exponential computational cost, and hence become impractical as the number of system components increases.

We have developed a new and powerful diagnosis engine that overcomes these limitations through a two-fold approach. First, we propose a novel and compact reconstruction of the General Diagnosis Engine (GDE), one of the most fundamental approaches to model-based diagnosis. We then present a novel algorithmic approach for calculation of minimal diagnosis set. This approach uses a powerful yet simple representation to map minimal diagnosis set calculation onto two well-known problems, namely the Boolean Satisfiability and Integer Programming problems. We report the details of a new and powerful tool, *Diagnosis Engine version 1.0*, that we have developed based on these techniques.

The mapping onto Boolean Satisfiability enables the use of very efficient algorithms with a super-polynomial rather than an exponential complexity for the problem. The mapping onto 0/1 Integer Programming problem enables the use of a variety of algorithms that can efficiently solve the problem for up to several thousand components. These new algorithms significantly improve over the existing ones, enabling efficient diagnosis of large complex systems. In addition, the latter mapping allows, for the first time, determination of the bound on the solution, i.e., the minimum number of faulty components, before solving the problem. This is a powerful insight that can be exploited to develop yet more efficient algorithms for the problem.

At the end, we report the results of validating and benchmarking of our engine based on this technology, and the results of estimating the bounds for specific problems.

1. Introduction

The diagnosis of a system is the task of identifying faulty components that cause the system not to function as it was intended. The diagnosis problem arises when some *symptoms* are observed, that is, when the system's actual behavior is in contradiction with the expected behavior. The solution to the diagnosis problem is then determination of the set of faulty components that fully explains all the observed symptoms. Of course, the meaningful solution should be a *minimal* set of faulty components since the trivial solution, that assumes all components are faulty, always explains all inconsistencies.

The *model-based diagnosis*, first suggested by Reiter [1] and later expanded by de Kleer *et al* [2], is the most disciplined technique for diagnosis of a variety of systems. This technique, which reasons from *first principle*, employs knowledge of how devices work and their connectivity in form of models. In the model-based diagnosis the focus is on the logical relations between components of a complex system. So the function of each component and the interconnection between components all are represented as a logical system, called the *system description SD*. The expected behavior of the system is then a logical consequence of *SD*. This means that the existence of faulty components leads to inconsistency between the observed behavior of the system and *SD*. Therefore, the determination of the faulty components (or, diagnosis) is reduced to finding those components for which assumption of their abnormality could explain all inconsistencies.

In summary, the diagnosis process starts with identifying symptoms that represent inconsistencies (discrepancies) between the system's model (description) and the system's actual behavior. Each symptom identifies a set of *conflicting* components as initial candidates. A *minimal diagnosis* is the smallest set of components that intersects all candidate sets. Therefore, finding the minimal diagnosis set is accomplished in two steps: first generating candidate sets from symptoms, and then calculating minimal set of faulty components. However, there are two major drawbacks in the current model-based diagnosis techniques that severely

limit their practical applications (see also Section 2). First, current model-based techniques tend to be large and complex and hence difficult to apply. Second and more importantly, in order to find the minimal diagnosis set, they rely on algorithms with exponential computational cost and hence are highly impractical for application to many systems of interest.

It should be mentioned that a widely employed approach to overcome limitations of model-based diagnosis techniques is to develop a set of *Fault Protection Modes*, that is, a set of Symptoms-to-Cause Rules. In this approach, the human experts, by relying on their detailed knowledge of the target system, attempt to predict and analyze all possible faults and determine their causes. Obviously, in this approach human knowledge is used to overcome the exponential computational complexity. As a result, this approach is time-consuming, costly and prone to human errors (since it is impossible to predict all possible faults in advance). As an example, the development of Fault Protection Modes for Cassini Spacecraft took more than 20 work years [4].

In this paper, we present a novel and two-fold approach for model-based diagnosis to overcome the two above-mentioned limitations and to achieve a powerful engine that can be used for fast diagnosis of large and complex systems. This approach starts by a novel and compact reconstruction of General Diagnosis Engine (GDE), as one of the most fundamental approaches to model-based diagnosis. More importantly, we present a novel algorithmic approach for calculation of minimal diagnosis set. We first discuss the relationship between this calculation and solution of the well-known Hitting Set Problem. We then discuss a powerful yet simple representation of the calculation of minimal diagnosis set. This representation enables the mapping onto two well-known problems, that is, the Boolean Satisfiability and 0/1 Integer Programming problems. The mapping onto Boolean Satisfiability enables the use of very efficient algorithms with a super-polynomial complexity for the problem (see Section 4). The mapping onto 0/1 Integer Programming problem enables the use of variety of algorithms that can efficiently solve the problem for up to several thousand components. Therefore, these new algorithms significantly improve over the existing ones, enabling efficient diagnosis of large complex systems. In addition, the latter mapping allows, for the first time, determination of the bound on the solution, i.e., the minimum number of faulty components, before solving the problem. This powerful insight can potentially lead to yet more powerful algorithms for the problem.

This paper is organized as follows. In Section 2, we review the main notions and concepts of model-based diagnosis by considering GDE, as applied to the hydrazine propulsion subsystem of Cassini spacecraft. We also briefly discuss a novel and compact reconstruction of GDE and its advantages. In Section 3, we discuss the relationship between the calculation of minimal diagnosis set and the Hitting Set Problem. In Section 4, we discuss a simple representation of both the Hitting Set and calculation of minimal diagnosis set problems that allows simple mapping onto Boolean Satisfiability and Integer Programming, discussed in details in Section 4 and 5. In Section 6 we describe our tool, Diagnosis Engine version 1.0, which we have developed based on techniques described in this paper. We also present the results of benchmarking of this tool, based on experiments on circuits with up to 100 components. In Section 7, we show that the mapping onto Integer Programming can be used for *a priori* determination of bound on the solution, i.e., on the number of faulty components. We show that these bounds could provide sharp results for specific cases. Finally, some concluding remarks and discussion of future works are presented in Section 8.

2. Model-based diagnosis: general diagnosis engine (GDE)

The *model-based diagnosis* approach was first suggested by Reiter [1] and later expanded more by de Kleer *et al* [2]. The General Diagnostic Engine (GDE) [5] is one of the most fundamental approaches to model-based diagnosis. GDE combines a model of a device with observations of its actual behavior to detect discrepancies and diagnose root causes. Here a system is represented as

$$\mathbf{S} = (SD, COMPONENTS, OBS),$$

where *COMPONENTS* is a finite set of components constituting the system (they are the constants of the logical theory describing the system), and *SD* and *OBS* are finite sets of first-order sentences describing the system description and observations, respectively. There is a distinguished unary predicate *AB*, where *AB*(*c*) is interpreted as "the component *c* is abnormal or faulty." For example, consider the system described in the following diagram:

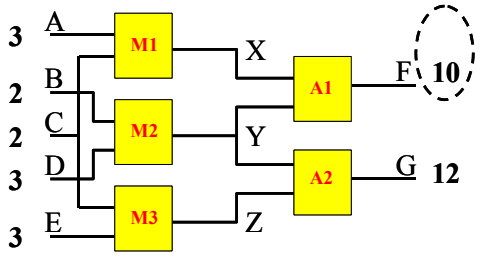

Figure 1 An Adder-Multiplier network.

Here the system has five components, represented by boxes. There are two different types of gates: the *adder* and the *multiplier*. We use the predicates *ADDER* and *MULTIPLIER* to denote these two types. In this example, *COMPONENTS* consists of five gates: M1, M2, M3, A1, and A2. The following sentences determine the type of each gate:

ADDER(A1), *ADDER*(A2), *MULTIPLIER*(M1), *MULTIPLIER*(M2), *MULTIPLIER*(M3).

The operation of each type of gates can be described by the following (first order) logical sentences:

$$\forall x\left[ADDER(x)\wedge\neg AB(x)\Rightarrow out(x)=in1+in2(x)\right],$$
$$\forall x\left[MULTIPLIER(x)\wedge\neg AB(x)\Rightarrow out(x)=in1\times in2(x)\right].$$

The predicates *in*1, *in*2, and *out* are used for describing the interconnections of the gates. For example,

out(M1)=*in*1(A1), *out*(M2)=*in*2(A1), …

Since this system is dealing with the numbers, we should also add the axioms of the theory of numbers to the set *SD*. Thus in this example, the set *SD* consists of all the above sentences. The set *OBS* describes the inputs and their corresponding outputs. In our example, the following sentence describes the inputs and the corresponding outputs:

$$(in1(\mathrm{M1})=3)\wedge(in2(\mathrm{M1})=2)\wedge(in1(\mathrm{M2})=2)\wedge$$
$$(in2(\mathrm{M2})=3)\wedge(in1(\mathrm{M3})=2)\wedge(in2(\mathrm{M3})=3)\wedge$$
$$(out(\mathrm{A1})=10)\wedge(out(\mathrm{A2})=12)$$

A *conflict* of the system ***S*** is a *disjunction* of the following form:

$$\Gamma=F_1(c_1)\vee F_2(c_2)\vee\cdots\vee F_k(c_k)$$

where $c_j\in COMPONENTS$ and $F_j=AB$ or $F_j=\neg AB$ such that

$$SD,OBS\Rightarrow\Gamma.$$

i.e., Γ is a logical consequence of *SD* and *OBS*. The conflict Γ is *positive* if $F_j=AB$ for every *j*=1,…,*k*. The conflict Γ is a *minimal conflict* if no subclause of it is a conflict.

A *diagnosis* of the system ***S*** is a *conjunction* of the following form determined by a subset *D* of *COMPONENTS* of the form

$$\Delta=\Delta(D)=\left[\wedge_{c\in D}\ AB(c)\right]\wedge\left[\wedge_{c\notin D}\ \neg AB(c)\right],$$

such that the set $\{SD,\ OBS,\ \Delta(D)\}$ is consistent. The diagnosis $\Delta(D)$ is a *minimal diagnosis* if for every proper subset *F* of *D* the conjunction $\Delta(F)$ is not a diagnosis.

The central issue of the diagnosis problem is to find small minimal diagnosis. So we formally formulate the diagnosis problem as follow.

Diagnosis

Instance: A system **S** = (SD, COMPONENTS, OBS) with n components and a number $0<c<1$.

Question: Is there a diagnosis $\Delta(D)$ for **S** such that $|D|\le c\,n$?

In [6] we have proved the following theorem that shows the problem of finding minimal diagnosis, in general, is intractable.

Theorem (see [6]): The problem Diagnosis is NP-hard, even if c is a constant in the interval $(0,0.5]$.

For solving the diagnosis problem, the General Diagnostic Engine (GDE) first performs a causal simulation by taking variable observations and using rules to compute the values of other variables in the network. Since computations have underlying assumptions, GDE tags each value with the assumptions that contribute to its computation. A discrepancy arises when two incompatible values are assigned to the same variable. Typically in the course of causal simulation, no discrepancies are found, but when failures occur, multiple incompatible assumption sets appear. This process continues to determine new incompatible sets until the causal simulation is completed. The next step after causal simulation is to the find minimal set of assumptions that intersects with all detected incompatible sets. This set contains the actual diagnoses of the root causes for contradictory measurements. However, GDE also suffers from the two main limitations of other model-based diagnosis approach; that is, the complexity of software makes its application difficult, and there is an exponential computational cost for finding the minimal set.

In order to overcome the first limitation, we have developed a novel and compact reconstruction of GDE. Traditionally GDE has been implemented using an inference engine to reason about a device model combined with an Assumption-based Truth Maintenance System (ATMS) to keep track of assumptions. A surprising result that arose from our rational reconstruction of GDE involves merging the ATMS with the inference engine. It turns out that the ATMS and the inference engine have many similarities, and combining the two dramatically simplifies the algorithm. The resulting system was completely implemented in under 150 lines of LISP code! This reconstruction also has some valuable properties for improving reasoning performance. Directly linking the reasoning about a device with reasoning about underlying assumptions facilitates the use of computation reduction heuristics.

The second limitation is, however, by far more challenging, as it was stated in the above theorem. However, in the following we show that, by mapping the diagnosis problem onto Boolean Satisfiability and Integer Programming problems, algorithms with much better performance and hence with a much wider range of applicability can be devised.

3. Hitting set problem

The *Hitting Set Problem*, also known as *the Transversal Problem*, is one of the key problems in the combinatorics of finite sets (see [7]) and the theory of diagnosis (see [1,2]).

The problem is simply described as follows. A collection $\boldsymbol{S} = \{S_1, \ldots, S_m\}$ of nonempty subsets of a set M is given. A hitting set (or transversal) of $\boldsymbol{S}$ is a subset H of M that meets every set in the collection $\boldsymbol{S}$; i.e., $S_j \cap H \neq \{\}$, for every $j = 1,\ldots,m$. Of course, there are always trivial hitting sets, for example the background set M is always a hitting set. Actually we are interested in *minimal hitting sets with minimal cardinality*: a hitting set H is minimal if no proper subset of H is a hitting set.

Our primary interest to Hitting Set Problem is its connection with the problem of diagnosis. The main theorem in the theory of model-based diagnosis states that the minimal diagnoses of the system are exactly the minimal hitting sets of the sets of conflicts

Note that for any system $\boldsymbol{S}$ of subsets of the set $M = \{m_1, m_2, \ldots, m_n\}$, finding *one* minimal hitting set is easy. The more challenging, and more interesting both from practical and theoretical point of view, is the problem of finding hitting sets of small size. It turns out that this is a hard problem. First let formalize the problem.

Hitting Set

Instance: A system $\boldsymbol{S} = \{S_1, \ldots, S_m\}$ of subsets of the set M and a constant $\frac{1}{2} \le c < 1$.

Question: Is there a hitting set H such that $|H| \le c|M|$?

We should mention that it is well known that the above problem is NP-complete if the condition is replaced by $|H| \le K$, where $K \le |M|$ (see [8]). It is also known that, in this latter form, the problem remains NP-complete even if $|S_j| \le 2$, for every $j = 1,\ldots,m$. Utilizing our results on the complexity of the diagnosis problem, it is possible to show that this stronger form of the problem is NP-complete. In [13] the complexity of several other problems related to hitting sets is investigated.

As mentioned before, we are interested in the Hitting Set Problem because of its connection with the problem of diagnosis. In fact, as it was discussed, each symptom identifies a set of conflicting components as initial candidates and minimal diagnoses are then the smallest sets of components that intersect all candidate sets. The main theorem in the theory of model-based diagnosis [1,2] also states that the minimal diagnoses of the system are exactly the minimal hitting sets of the conflict sets (see Figure 2).

The Reiter's hitting set algorithm [1] is one of the major algorithms for finding minimal hitting sets. The correction of this algorithm is presented in [9] and a modified and more efficient version in [10]. The original algorithm and its modifications are based on generating the lattice of the subsets of the background set M and then extracting a sublattice of it that provides the minimal hitting sets. If the goal is to find a minimal hitting set with minimal cardinality, then this algorithm is not efficient by any means; because it requires to save the whole sublattice which leads (in the worst case) to an exponential size memory to save the sublattice. We will show that it is possible to find a minimal hitting set with minimal cardinality with an algorithm that requires a linear size memory (while it still may needs an exponential time to complete the computation).

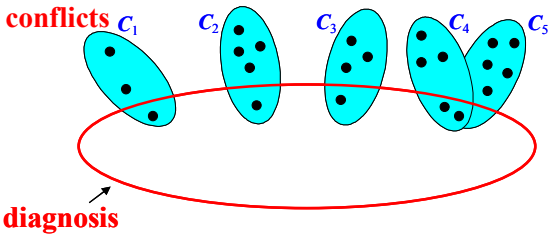

Figure 2 Diagnosis as the hitting set of the conflicts.

Our approach for solving the Hitting Set Problem and thus calculation of minimal diagnosis set is two-folded. On one hand, we map the problem onto the *Monotone* Boolean Satisfiability Problem. This provides the opportunity of utilizing the super-polynomial algorithms for finding the prime implicants of monotone functions (see [11,14]) and thus minimal diagnosis set. Also, this mapping makes it possible to better understand the complexity of the Hitting Set Problem, by comparing it with the well-studied Boolean function problems. On the other hand, we map the problem onto an Integer Programming Optimization Problem. This simple mapping gives us access to a vast repertoire of Integer Programming techniques that in some cases can effectively solve problems with several thousands variables. We would like to mention that mapping of the problem of finding prime implicants (not necessarily prime implicants of monotone formulas) onto the Integer Programming has already been introduced; see, e.g., [16,17]. The mappings of the hitting set problem onto monotone satisfiability and Integer Programming, which is introduced in this paper, provides a new mapping of the problem of finding prime implicants of *monotone* formulas onto the Integer Programming.

4. Mapping onto Boolean satisfiability problem

In order to describe mapping of the Hitting Set Problem onto Boolean Satisfiability and 0/1 Integer Programming, consider a different representation of the problem by describing the attribution of the members (or, components) to subsets (or, initial candidate sets) as given by the following matrix:

	m_1	m_2	$\cdots$	m_n
S_1	1	0	$\cdots$	0
S_2	0	1	$\cdots$	1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
S_m	1	1	$\cdots$	0

(1)

where $\boldsymbol{S} = \{S_1, \ldots, S_m\}$ and $M = \{m_1, m_2, \ldots, m_n\}$ denote the set of nonempty subsets and the set of members (elements), respectively. The $(i,j)^{\text{th}}$ entry in this matrix is denoted as a_{ij} and we have $a_{ij} = 1$ if m_j belongs to S_j, otherwise $a_{ij} = 0$. To map the problem onto Boolean Satisfiability, we introduce the Boolean variables $x_1, x_2, \ldots, x_n$,, where each variable x_j represents the member m_j. Then to each subset $S_i = \{m_{i1}, m_{i2}, \ldots, m_{ini}\}$ (i.e., each row of matrix (1)) we correspond the disjunction

$$F_i = x_{i1} \vee x_{i2} \vee \cdots \vee x_{i\,ni} \qquad (2)$$

i.e., for each "1" in the i^{th} row of matrix (1) the corresponding Boolean variable appears in the disjunction (2). For example, if the i^{th} row of matrix (1) is (0,1,1,0,0,1,0) then $F_i = x_2 \vee x_3 \vee x_6$. Now the formula

$$F_{\boldsymbol{S}} = F_1 \wedge F_2 \wedge \cdots \wedge F_m \qquad (3)$$

represents the mapping of the Hitting Set Problem associated with the system $\boldsymbol{S}$ onto the Boolean Satisfiability Problem in the sense that every hitting set of the system $\boldsymbol{S}$, in a natural way, corresponds with a satisfying truth-assignment for the formula $F_{\boldsymbol{S}}$, and vice versa. Let $(s_1, s_2, \ldots, s_n)$ be a Boolean vector that satisfies the formula $F_{\boldsymbol{S}}$, and let the subset S be the corresponding set. Then the formula (2) guarantees that S intersects the set S_i, and (3) guarantees that S intersects all sets $S_1, S_2, \ldots, S_m$. Thus S is a hitting set.

We should notice that the Boolean formula (3) is in fact *monotone*. In the case of monotone formulas, the standard form of the Satisfiability Problem should be slightly modified to avoid the trivial cases. Note that, in the case of the monotone formulas, the all-one vector (1,1,…,1) is always a satisfying truth-assignment (or equivalently, the background set M is always a hitting set). Here, the correct formulation of the problem is to find the assignments with bounded weight, or in the hitting set setting, the problem is to find hitting sets with bounded number of members. We have shown that the problem of finding truth-assignments for monotone formulas with weight $\le cn$, for $\frac{1}{2} \le c < 1$, is NP-complete [6]. Also, the problem of finding minimal hitting sets of the system $\boldsymbol{S}$ reduces to the problem of finding prime implicants of the monotone function $F_{\boldsymbol{S}}$.

We should mention here a new result [11,14] that suggests a major breakthrough regarding finding hitting sets in the most general case of the problem. They show that there is an algorithm that produces the list of prime implicants of a monotone Boolean function such that each prime implicant is produced in the time $O\left(nt + n^{O(\log n)}\right)$, where t is the time needed to determine the value of the Boolean function at any point. Also the list that produced by this algorithm has no repetitions. Practical implication of this result for hitting set problem is that for the systems that do not have large number of minimal hitting sets (i.e., there are at most superpolinomailly many minimal hitting sets), it is possible to solve the hitting set problem in superpolynomial time, instead of exponential time of a typical NP-complete problem.

5. Mapping onto 0/1 programming problem

In order to describe the mapping onto 0/1 Integer Programming Problem, define the $n \times m$ matrix $A = \left(a_{ij}\right)_{1 \le i \le m,\, 1 \le j \le n}$ associated with the system $\boldsymbol{S}$, as defined in matrix (1). Note that, by this definition, each row of A corresponds to a subset and each column to a member. The mapping onto 0/1 Integer Programming Problem is simply obtained by considering an operator application of A as follows. Identification of a minimal subset of members, representing a minimal hitting set, is equivalent to finding a minimal subset of columns of the matrix A whose summation results in a vector with elements equal to or greater than 1. This can be better described in terms of matrix-vector operations as follows. Let the vector A_i, for $i = 1, \ldots, m$, denotes the i^{th} row of the matrix A. Also, define a binary vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, wherein $x_j = 1$ if the member m_j belongs to the minimal hitting set, otherwise $x_j = 0$. Since at least one member should belong to every S_i, for every $i = 1,\ldots,m$, we then have

$$A_i \cdot \boldsymbol{x} \ge 1.$$

Since, by the definition of the minimal subset, the above equation should be simultaneously satisfied for all $i = 1,\ldots,m$, we then have the following formulation of the problem as an 0/1 integer programming problem

$$\begin{array}{ll} \text{minimize} & wt(\boldsymbol{x}) \\ \text{subject to} & A\,\boldsymbol{x}^T \ge \boldsymbol{b}^T, \quad x_j = 0 \text{ or } 1 \end{array} \qquad (4)$$

where $\boldsymbol{b} = (1,1,..,1)$ is the all-one vector, and we denote the *Hamming weight*, i.e., the number of one-components of the binary vector $\boldsymbol{x}$, by $wt(\boldsymbol{x})$. With this setting, identification of the minimal hitting set is then equivalent to solution for the binary vector $\boldsymbol{x}$ from (4), which corresponds to the solution of the 0/1 Integer Programming Problem.

Note that (4) represents a rather special case of the 0/1 Integer Programming Problem since the matrix A is a binary matrix, i.e., with 1 or 0 entries only. Interestingly, our above derivation also establishes a mapping of the Monotone Boolean Satisfiability Problem onto this special case of 0/1 Integer Programming Problem. To see this, note that any Monotone Boolean Satisfiability Problem, given by the formula (3), can be equivalently represented by a matrix similar to (1), from which the mapping onto this special case of 0/1 Integer Programming Problem follows immediately.

6. The tool and benchmarking: model-based diagnosis engine, version 1.0

We have developed a tool that combines all the techniques we have discussed in this paper. This tool, Diagnosis Engine, version 1.0, constitutes several components so that it has the capability to an end-to-end diagnosis. The key components of the tool are:

- **Description of the system.** The first step of diagnosis process of a specific system is to define the system. This step itself involves two stages. (i) We have to specify the functionality of all possible components used by the system. Right now, the code we have developed in LISP is capable of handling components in arithmetical (adder-multiplier gates) and Boolean (logical gates) settings. Extending the LISP code to other discrete settings is straightforward. (ii) We have to define the interconnections between the components of the system. We have adapted a natural way to specify these interconnections in the LISP code. The inputs and the observations of the system are treated as part of system description, and are provided by this component.
- **Conflict finding process.** The conflict finding routine of version 1.0 of Engine is based on a reconstruction of GDE method [5]. The output of this component is a matrix A of the form of matrix (1).
- **Integer programming solver.** The conflict matrix A, that is the output of the conflict finding routine, will be used to solve the integer programming problem (4). To solve (4), the version 1.0 of Engine has two methods in its disposal. One is an enhanced version of the brute force method. This method is very successful in the cases of small number of faults. We will describe it later. The other method for solving (4), is the GLPK (GNU Linear Programming Kit), version 3.2. This is a set of routines in the ANSI C programming language. The integer programming routine of GLPK (actually it is much more powerful routine and is capable of solving mixed integer programming problems) applies a variant of branch-and-bound method for the problem. The GLPK, like other available integer programming solver tools, provides only one solution of (4). The version 1.0 of Engine has capability of finding all possible solution of (4). The routine that provides this feature is described later in this section.

Benchmarking

As the first stage of benchmarking and validation of the tool, we applied our engine on test cases of circuits with adder and multiplier gates. The Table 1 shows the results of these experiments.

Table 1 Some of the results of benchmarking

Number of components	Size of the matrix	Number of faults	Time: Engine
105	10×105	2	< 1 SEC.
105	12×105	3	< 1 SEC.
105	12×105	4	< 1 SEC.
105	17×105	5	< 1 SEC.
105	23×105	6	2 SEC.
39	48×39	9	< 1 sec.
39	48×39	9	< 1 sec.
52	62×52	10	< 1 sec.
65	33×65	13	1 sec.
78	94×78	16	5 sec.

Finding all solutions of integer programming problem

The GLPK routine finds only one solution of the integer programming problem (4). To find all possible solutions, the version 1.0 of Engine applies the following procedure. After finding the first solution $\boldsymbol{a}_1$ of (4), a new integer programming problem is defined by simply adding a new row to the matrix A to form matrix A_1: the new row is the complement of $\boldsymbol{a}_1$ (note that, for example, if $\boldsymbol{a}_1$=(1,0,0,1,0) then its complement is (0,1,1,0,1)). So the new problem is defined by substituting A by A_1 in (4). It is easy to check that $\boldsymbol{a}_1$ is not the solution of the new system, and all solutions of the new system are all solutions of the old one except $\boldsymbol{a}_1$. This method will be continued till the new system becomes infeasible.

Enhanced brute force algorithm

There is a trivial exhaustive search method for finding solutions of the integer programming problem (4): check all possible binary vectors $\boldsymbol{x}$. This method, of course, is extremely inefficient, as the number of such vectors $\boldsymbol{x}$ is 2^n, where n is the number of components of the system. But a closer examination of the problem (4), shows that there is a much better way to carry out such search. The problem (4) is *monotone*, in the sense that if $\boldsymbol{a}$ is a solution and $\boldsymbol{a} \le \boldsymbol{b}$, then $\boldsymbol{b}$ is also a solution. Since the objective function of this optimization problem is the Hamming weight of the vector $\boldsymbol{x}$, it follows that once a solution $\boldsymbol{a}$ of the weight $wt(\boldsymbol{a}) = k$ is found, we do not have to look for solutions among binary vectors of weight larger than k. Therefore, the enhanced search algorithm looks for the solutions of (4) by systematically checking the vectors of weight 1, 2, 3, … in the increasing order of their weights; i.e., the algorithm does not starts checking vectors of weight k before it examines all vectors of smaller weights (see Figure 3).

To compare the performance of this enhanced search algorithm with the trivial exhaustive search, note that if the size of the minimal diagnosis is t, then the enhanced algorithm requires checking

$$\sum_{j=1}^{t}\binom{n}{j} \qquad (5)$$

vectors. For small values of t, the number defined by (5) is proportional to n^t. This means that, for small values of the number of faulty components t, the complexity of the enhanced algorithm grows polynomially. For example, if we want to run the enhanced algorithm for 1 minute, then the algorithm can handle the cases of 2, 3, and 4 faulty components for circuits of size (about) 1200, 160, and 70, respectively

```
for j = 1 to n do
        {
        test all vectors x as sum of j columns of the
        matrix A;

        break, if a solution is found;
        }
```

Figure 3 The enhanced brute force algorithm.

7. Lower bounds

As stated before, the Integer Programming is known to be an intractable problem (see [8]), though there are several reasonably good algorithms that can solve the problem either exactly for certain size or approximately for any size. However, our recent discovery of the bounds on the size of the solution of (4) opens a new direction for improving the efficiency of existing algorithms and/or devising new and more efficient algorithms. Here, we briefly describe these new results.

For two vectors $\boldsymbol{x}=(x_1,x_2,\ldots,x_n)$ and $\boldsymbol{y}=(y_1,y_2,\ldots,y_n)$ in $\mathbf{R}^n$, we write $\boldsymbol{y}\geq\boldsymbol{x}$ if and only if $y_j \geq x_j$, for every $j = 1,\ldots,n$. Also, we consider the 1–norm and 2–norm of vectors defined as

$$\|\boldsymbol{x}\|_1=\sum_{j=1}^{n}|x_j|, \quad \|\boldsymbol{x}\|_2=\sqrt{\sum_{j=1}^{n}x_j^{\,2}}.$$

For the vector $\boldsymbol{b}$ in (4), we then have $\|\boldsymbol{b}\|_1 = m$ and $\|\boldsymbol{b}\|_2 = \sqrt{m}$. Since the elements of both vectors $A\boldsymbol{x}^T$ and $\boldsymbol{b}$ in (4) are positive, we can then drive the following two inequalities:

$$\left.\begin{array}{rcl} \|A\|_1\times\|\boldsymbol{x}\|_1\geq m & \Rightarrow & \|\boldsymbol{x}\|_1\geq m/\|A\|_1 \\ \|A\|_2\times\|\boldsymbol{x}\|_2\geq\sqrt{m} & \Rightarrow & \|\boldsymbol{x}\|_2\geq\sqrt{m}/\|A\|_2 \end{array}\right\} \qquad (6)$$

Since $\boldsymbol{x}$ is a binary vector, then both norms in (6) give the bound on the size of the solution, that is, the number of nonzero elements of vector $\boldsymbol{x}$ which, indeed, corresponds to the minimal diagnosis set. Note that, depending on the structure of the problem, i.e., the 1– and 2–norm of the matrix A and m, a sharper bound can be derived from either of (6). To our knowledge, this is the first time that such bounds on the solution of the problem have been derived without any need to explicitly solve the problem. Such a priori knowledge on the size of solution will be used for developing much more efficient algorithms for the problem.

The following table shows that results of application of these bounds on several instances of the problem. As this table shows, these bounds provide non-trivial estimates, and in some cases exact values, for the size of the minimal diagnoses. Here we applied the above lower bound method on circuits of the type we used in our benchmarking.

Table 2 Lower bound on the size of the diagnosis

Number of components	Size of the matrix	The actual size of the diagnosis	Lower bound
15	20×15	3	2
39	10×39	3	3
39	15×39	4	4
39	17×39	5	5
39	26×39	6	4
39	48×39	9	4
52	44×52	4	4
52	45×52	5	5
52	62×52	10	6

We would like to mention several applications of these lower bounds. First of all, the *a priori* lower bound, before starting to solve the hard problem of finding the minimal hitting sets, allows us to separate the cases where the high number of faulty components requires another course of action instead of usual identification of faulty components. Second, a good lower bound could determine whether the enhanced brute-force algorithm (discussed in Section 6) can provide a solution efficiently. Finally, these lower bounds can be used for finding bounds for subproblems in branch-and-bound method for solving integer programming problem.

8. Conclusion

We proposed a two-folded approach to overcome the two major limitations of the current model-based diagnosis techniques, that is, the complexity of the tools and the exponential complexity of calculation of minimal diagnosis set. To overcome the first limitation, we have developed a novel and compact reconstruction of GDE. To overcome the second and more challenging limitation, we have proposed a novel algorithmic approach for calculation of minimal diagnosis set. Starting with the relationship between the calculation of minimal diagnosis set and the celebrated Hitting Set problem, we have proposed a new method for solving the Hitting Set Problem, and consequently the diagnosis problem. This method is based on a powerful yet

simple representation of the problem that enables its mapping onto two other well-known problems, that is, the Boolean Satisfiability and 0/1 Integer Programming problems. The mapping onto Boolean Satisfiability enables the use of very efficient algorithms with a super-polynomial rather than an exponential complexity for the problem.

The mapping onto 0/1 Integer Programming problem enables the use of variety of algorithms that can efficiently solve the problem for up to several thousand components. These new algorithms significantly improve over the existing ones, enabling efficient diagnosis of large complex systems. In addition, this mapping allows, for the first time, *a priori* determination of the bound on the solution, i.e., the minimum number of faulty components, before solving the problem. This is a powerful insight that can potentially lead to yet more powerful algorithms for the problem. It should be mentioned, however, that (4) represents a rather special case of the 0/1 Integer Programming Problem, by being specific to the calculation of minimal diagnosis set, since the matrix A is a binary matrix, and the vector $\boldsymbol{b}$ is the all-one vector. We are currently devising new techniques to exploit this special structure of this mapping to develop yet more efficient algorithms, optimized for calculation of the minimal diagnosis set.

We described the tool, Diagnosis Engine version 1.0, that we have developed based on the above technologies. This tool is capable of performing an end-to-end diagnosis process. We reported the results of benchmarking of the Engine on systems with up to 100 components with different number of faults. Also, by applying our lower bounds on some specific cases, we demonstrated that these bounds could provide sharp estimates on the number of faulty components.

Our current effort on developing a more powerful and practical model-based diagnosis engine builds upon the unique and compact reconstruction of GDE. In addition, the integration of these novel efficient algorithms within this reconstruction of GDE enables the development of new tools that can efficiently diagnose large systems.

Acknowledgement

The research described in this paper was performed at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under contract with National Aeronautics and Space Administration (NASA). This work is supported by JPL InterPlanetary Network and Information Systems Directorate under the State Diagnosis task.

References

[1] R. Reiter, "A theory of diagnosis from first principles," *Artificial Intelligence* **32**, 57–95, 1987.

[2] J. de Kleer, A. K. Mackworth and R. Reiter, "Characterizing diagnoses and systems," *Artificial Intelligence* **56,** 197–222, 1992.

[3] Cassini Program Environmental Impact Statement Supporting Study, Volume 3: Cassini Earth Swingby Plan, JPL D–10178–3, November 18,1993.

[4] Robert Rasmussen, Personal Communication, 2001.

[5] J. de Kleer and B. Williams, "Diagnosing Multiple Faults," *Readings in Model-Based Diagnosis*, Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[6] F. Vatan, "The complexity of diagnosis and monotone satisfiability," submitted to *Discrete Applied Mathematics*.

[7] C. Berge, *Hypergraphs: Combinatorics of Finite Sets*, Elsevier-North Holland, Amsterdam, The Netherlands, 1989.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[9] R. Greiner, B. A. Smith, and R. W. Wilkerson, "A correction to the algorithm in Reiter's theory of diagnosis," *Artificial Intelligence* **41**, 79–88, 1989.

[10] F. Wotawa, "A variant of Reiter's hitting-set algorithm," *Information Processing Letters* **79**, 45–51, 2001.

[11] V. Gurvich and L. Khachiyan, "On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean Functions," *Discrete Appl. Math*. **96–97**, 363–373, 1999.

[12] T. Bylander, D. Allemang, M. C. Tanner and J. R. Josephson, "The computational complexity of abduction," *Artificial Intelligence* **49**, 25–60, 1991.

[13] T. Eiter and G. Gottlob, "Identifying the minimal transversals of a hypergraph and related problems," *SIAM J. Comput*. **24**, 1278–1304, 1995.

[14] M. Fredman and L. Khachiyan, "On the complexity of dualization of monotone disjunctive normal forms," *J. of Algorithms* **21**, 618–628, 1996.

[15] G. Friedrich, G. Gottlob and N. Nejdl, "Physical impossibility instead of fault models," *Proceedings of Eighth National Conference on Artificial Intelligence AAAI-90*, vol. 1, MIT Press, Cambridge, 331–336, 1990.

[16] V. M. Manquinho, P. F. Flores, J. P. M. Silva and A. L. Oliveira, "Prime implicant computation using satisfiability algorithms," *Proceedings Ninth IEEE International Conference on Tools with Artificial Intelligence*, IEEE Computer Society, Los Alamitos, CA, 232–239, 1997.

[17] L. Palopoli, F. Pirri and C. Pizzuti, "Algorithms for selective enumeration of prime implicants," *Artificial Intelligence* **111**, 41–72, 1999.