A Novel Model-Based Diagnosis Engine:
Theory and Applications

Amir Fijany, Farrokh Vatan, Anthony Barrett, Mark James, Colin Williams, and Ryan Mackey
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-5342
{Amir.Fijany, Farrokh.Vatan, Anthony.C.Barrett, Mark.James, Ryan.M.Mackey}@jpl.nasa.gov, cpw@cs.stanford.edu

Abstract—Over the past decade, the number of Earth orbiters and deep space probes has grown dramatically and is expected to continue in the future as miniaturization technologies drive spacecraft to become more numerous and more complex. This rate of growth has brought a new focus on autonomous and self-preserving systems that depend on fault diagnosis. Although diagnosis is needed for any autonomous system, current approaches are almost uniformly “ad-hoc,” inefficient, and incomplete. Systematic methods of general diagnosis exist in literature, but they all suffer from two major drawbacks that severely limit their practical applications. First, they tend to be large and complex and hence difficult to apply. Second and more importantly, in order to find the minimal diagnosis set, i.e., the minimal set of faulty components, they rely on algorithms with exponential computational cost and hence are highly impractical for application to many systems of interest.

In this paper, we propose a two-fold approach to overcome these two limitations. Then we report the details of a new and powerful tool, Diagnosis Engine version 1.0, we have developed based on these techniques. First, we propose a novel and compact reconstruction of General Diagnosis Engine (GDE), as one of the most fundamental approaches to model-based diagnosis. We then present a novel algorithmic approach for calculation of minimal diagnosis set. Using a powerful yet simple representation of the calculation of minimal diagnosis set, we map the problem 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 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.

TABLE OF CONTENTS

1. INTRODUCTION
2. MODEL-BASED DIAGNOSIS: GENERAL DIAGNOSIS ENGINE (GDE)
3. HITTING SET PROBLEM
4. MAPPING ONTO BOOLEAN SATISFIABILITY PROBLEM
5. MAPPING ONTO 0/1 INTEGER PROGRAMMING PROBLEM
6. THE TOOL AND BENCHMARKING: MODEL-BASED DIAGNOSIS ENGINE, VERSION 1.0
7. LOWER BOUNDS
8. CONCLUSION
REFERENCES

1. INTRODUCTION
The diagnosis of a system is the task of identifying faulty components that cause the system to not function as it was intended. The diagnosis problem arises when some symptoms
is 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 explain 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 more by de Kleer, Mackworth and Reiter [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. As an example, Figure 1 illustrates a model of a hydrazine propulsion subsystem — in this case for Cassini’s [3] attitude thrusters. This model consists of a set of connected components, where the connectors are pipes. To diagnose this device the pressure sensors are monitored for discrepancies while valves open and close during normal operations.

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 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 which 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 a well-known problem, the Hitting Set Problem. In Section 4, we discuss a simple representation of both the Hitting Set and calculation of minimal diagnosis set problems which 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 about 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 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. For instance, consider Figure 1 illustrating a model of a hydrazine propulsion subsystem for Cassini's attitude thrusters. Computationally, each connector corresponds to a set of variables, and each component corresponds to a set of rules for computing variable values of incident connectors from variable values of other incident connectors. In this propulsion example, each pipe corresponds to a pair of variables for pressure and flow. Valve components correspond to rules that conditionally make these variables equivalent depending on whether or not a valve was commanded open or closed, assuming that the valve is functioning.

GDE 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. For instance, consider the two open valves between pressure sensors $P_1$ and $P_2$ in Figure 1. Assuming that both valves work gives us identical pressures at each sensor. If the sensors are measuring very different pressures, then at least one sensor or one valve must be faulty. In general, whenever GDE computes two incompatible values for the same variable, the union of the two supporting assumption sets is incompatible. In this case assumptions that $P_1$ and the two valves work imply a pressure value that is incompatible with the pressure value supported by the assumption that $P_2$ works. Thus, the set of four assumptions leads to contradictory computations.

Typically in the course of causal simulation, no discrepancies are found, but when failures occur, multiple incompatible assumption sets appear. For instance, continuing our causal simulation to the right from $P_2$ eventually leads to computing two values at $P_3$ with different assumption tags. Depending on the value measured by $P_3$, $P_3$ combines with one (or both) of these tags to form incompatible assumption set(s). This process continues to determine new incompatible sets until the causal simulation completes. The next step after causal simulation is to 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 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 resultant 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. For instance, tagging each assumption with a probability results in being able to focus the system on making inferences about high probability situations that match the current observations.

The second limitation is, however, by far more challenging. For, while it is easy to state the minimal set determination step, the actual computation solves the provably NP-Complete prime implicants problem. In fact, one of the authors [6] has recently shown, for the first time, that the diagnosis problem is NP-complete. 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.

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

3. HITTING SET PROBLEM

Our primary interest to Hitting Set Problem is due to its connection with the problem of diagnosis. The Hitting Set Problem, also known as the Transversal Problem, is one of 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 \( S = \{S_1, \ldots, S_n\} \) of nonempty subsets of a set \( M \) is given. A hitting set (or transversal) of \( S \) is a subset \( H \) of \( M \) that meets every set in the collection; i.e., \( S_j \cap H \neq \emptyset \), 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.

Note that for any system \( S \) of subsets of the set \( M = \{m_1, m_2, \ldots, m_n\} \), finding one minimal hitting set is easy. We define the sequence \( M_0, M_1, \ldots, M_{n+1} \) recursively as follows: let \( M_0 = M \); suppose that the set \( M_i \) is defined; let \( H = M_i - \{m_j\} \), i.e., remove the member \( m_j \) from \( M_i \), check whether \( H \) is a hitting set, if it is then let \( M_{i+1} = H \), otherwise let \( M_{i+1} = M_i \). Then it is easy to see that each set of the sequence \( M_0, M_1, \ldots, M_{n+1} \) is a hitting set and the set \( M_{n+1} \) is in fact a minimal hitting set. 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 \( S = \{S_1, \ldots, S_n\} \) of subsets of the set \( M \) and a constant \( \frac{1}{2} \leq c < 1 \).

**Question:** Is there a hitting set \( H \) such that \( |H| \leq c |M| \)?

We should mention that it is well known that the above problem is NP-complete if the condition is replaced by \( |H| \leq K \), where \( K \leq |M| \) (see [8]). It is also known that, in this latter form, the problem remains NP-complete even if \( |S_j| \leq 2 \), 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 the need of 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).

Our approach for solving the Hitting Set Problem and thus calculation of minimal diagnosis set is two-fold. 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 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 problems, 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:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_i m_1 m_2 ... m_n</td>
</tr>
<tr>
<td>S_1</td>
</tr>
<tr>
<td>S_2</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>S_m</td>
</tr>
</tbody>
</table>

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

\[
F_i = x_{i1} \lor x_{i2} \lor \cdots \lor x_{im}
\]

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 \( F_i = x_2 \lor x_3 \lor x_4 \). Now the formula

\[
F_S = F_1 \land F_2 \land \cdots \land F_m
\]

represents the mapping of the Hitting Set Problem associated with the system \( S \) onto the Boolean Satisfiability Problem in the sense that every hitting set of the system \( S \) in a natural way, corresponds with a satisfying truth-assignment for the formula \( F_S \), and vice versa. Let \( (s_1, s_2, ..., s_n) \) be a Boolean vector that satisfies the formula \( F_S \), and let the subset \( S \) be the corresponding set. Then the formula (2) guarantees that \( S \) intersects the set \( S_s \) and (3) guarantees that \( S \) intersects all sets \( S_1, S_2, ..., S_m \). 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 slightly be 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 (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 \( \leq cn \), for \( \frac{1}{2} \leq c < 1 \), is NP-complete. Also, the problem of finding minimal hitting sets of the system \( S \) reduces to the problem of finding prime implicants of the monotone function \( F_S \).

We should mention here a new result [11,14] which 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 a time of \( O(n + n^{O(\log n)}) \), 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 superpolinormally 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 INTEGER PROGRAMMING PROBLEM

In order to describe the mapping onto 0/1 Integer Programming Problem, define the \( n \times m \) matrix \( A = (a_{ij})_{i\leq n, j\leq m} \) associated with the system \( S \), 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 operation as follows. Let the vector \( A_m \) for \( i = 1, ..., m \), denotes the \( i \)th row of the matrix \( A \). Also, define a binary vector \( x = (x_1, x_2, ..., x_m) \), wherein \( x_1 = 1 \) if the member \( m_1 \) belongs to the minimal hitting set, otherwise \( x_1 = 0 \). Since at least one member should belong to every \( S_i \) for every \( i = 1, ..., m \), we then have

\[
A_i \cdot x \geq 1.
\]

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

\[
\text{minimize} \quad w(x)
\]

subject to

\[
Ax \geq b, \quad x_j = 0 \text{ or } 1
\]

where \( b = (1,1,1,...,1) \) is the all-one vector, and we denote the Hamming weight, i.e., the number of one-components of the binary vector \( x \), by \( w(x) \). With this setting, identification of the minimal hitting set is then equivalent to solution for the binary vector \( x \) 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 elements 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.

Example

As an example we consider the small problem solved in [9,10]. Here $M = \{a, b, c, d\}$ and $S = \{(a, b), (b, c), (a, c), (b, d), (b)\}$. In the case the incidence matrix (1) is of the following form:

$$
\begin{bmatrix}
1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0
\end{bmatrix}
$$

The corresponding 0/1 optimization problem is then formulated as:

minimize $x_1 + x_2 + x_3 + x_4,$
subject to $x_1 + x_2 \geq 1, x_3 + x_4 \geq 1, x_1 + x_3 \geq 1, x_2 + x_4 \geq 1,$

This problem has two solutions

$$\begin{align*}
\{x_1 = x_2 = 1, x_3 = x_4 = 0\}, \\
\{x_1 = x_3 = 1, x_2 = x_4 = 0\},
\end{align*}$$

which correspond to the minimal hitting sets $\{a, b\}$ and $\{b, d\}$.

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:

1. 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.

2. 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).

3. 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 case 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 (Figure 3 shows one such circuit). The following tables show the results of these experiments.

<table>
<thead>
<tr>
<th>Number of components</th>
<th>Size of the matrix</th>
<th>Number of faults</th>
<th>Time: Engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>105</td>
<td>10x105</td>
<td>2</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>105</td>
<td>12x105</td>
<td>3</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>105</td>
<td>12x105</td>
<td>4</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>105</td>
<td>17x105</td>
<td>5</td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>105</td>
<td>23x105</td>
<td>6</td>
<td>2 sec.</td>
</tr>
</tbody>
</table>
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 $a_1$ of (4), a new integer programming problem is defined by simply adding a new row to the matrix $A$ to form matrix $A_I$: the new row is the complement of $a_1$ (note that, for example, if $a_1=\langle 1,0,0,1,0 \rangle$ then its complement is $\langle 0,1,1,0,1 \rangle$). So the new problem is defined by substituting $A$ by $A_I$ in (4). It is easy to check that $a_1$ is not the solution of the new system, and all solutions of the new system are all solutions of the old one except $a_1$.

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 $x$. This method, of course, is extremely inefficient, as the number of such vectors $x$ 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 $a$ is a solution and $a \leq b$, then $b$ is also a solution. Since the objective function of this optimization problem is the Hamming weight of the vector $x$, it follows that once a solution $a$ of weight $wt(a) = k$ 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 4).

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 to check

$$\sum_{j=0}^{t} \binom{n}{j}$$

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 4. 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 \( x = (x_1, x_2, \ldots, x_n) \) and \( y = (y_1, y_2, \ldots, y_n) \)
in \( \mathbb{R}^n \), we write \( y \geq x \) 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

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

For the vector \( b \) in (4), we then have \( \|b\|_1 = m \) and

\[
\|b\|_2 = \sqrt{m}.
\]

Since the elements of both vectors \( Ax \) and \( b \) in (4) are positive, we can then drive the following two inequalities:

\[
\begin{align*}
\|Ax\|_1 \geq m & \Rightarrow \|x\|_1 \geq m/\|A\|_1, \\
\|Ax\|_2 \geq \sqrt{m} & \Rightarrow \|x\|_2 \geq \sqrt{m}/\|A\|_2.
\end{align*}
\]

Since \( x \) is a binary vector, then both norms in (5) give the bound on the size of the solution, that is, the number of nonzero elements of vector \( x \) 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 (5). 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 lower 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**

<table>
<thead>
<tr>
<th>Number of components</th>
<th>Size of the matrix</th>
<th>The actual size of the diagnosis</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>20x15</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>39</td>
<td>10x39</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>39</td>
<td>15x39</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>17x39</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>39</td>
<td>26x39</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>39</td>
<td>48x39</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>52</td>
<td>44x52</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>52</td>
<td>45x52</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>52</td>
<td>62x52</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>

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 algorithm.

**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 allows 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, i.e., with 1 or 0 elements only, and the vector \( b \) 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


Amir Fijany is a Principal Research Scientist in and technical group supervisor of the Ultracomputing Technology Research Group at the Jet Propulsion Laboratory, California Institute of Technology. He has been with JPL since 1987. His research interests include new computing devices and paradigms, advanced algorithms for scientific and grand challenge computational problems, and quantum information processing. He is currently leading the State Diagnosis Task funded by NASA. He is the author of over 70 scientific and technical papers, and research monographs.

Farrokh Vatan is a senior scientist in the Ultracomputing Technology Research Group at JPL. He received his Ph.D. in Computer Science from University of Illinois at Chicago. He joined JPL in February of 2001 and before that he worked at Electrical Engineering Department of UCLA as a member of research staff. His research interests include
computation theory, analog computation, discrete mathematics, quantum computation and information.

**Anthony Barrett** is a senior member of the Artificial Intelligence Group at the Jet Propulsion Laboratory, California Institute of Technology where his R&D activities involve planning, scheduling, and diagnosis applied to controlling constellations of spacecraft. He holds a B.S. in Physics, Computer Science, and Applied Mathematics from James Madison University and both an M.S. and PhD in Computer Science from the University of Washington. His research interests are in the areas of planning, scheduling, executives, and multi-agent systems.

**Mark James** is a senior R&D researcher of the Ultracomputing Technologies Research Group in the Information and Computing Technologies Research Section within NASA's Jet Propulsion Laboratory, Pasadena CA. At JPL he is Principle Investigator and Task Manager in real-time inference and knowledge-based systems. His primary research focus is on high-speed inference systems and their applications. His expertise includes core artificial intelligence technology, high-speed real-time inference systems, flight system software architectures and software design and implementation for those systems. He has received a number of NASA awards which includes: NASA Software of the year Award Nominee, JPL Team Excellence Award, JPL JET Productivity Award, Major NASA Monetary Award, NASA certificate of Recognition, NASA Achievement Award, NASA Exceptional Service Medal and NASA Exceptional Service Plaque for 1989. He is author of two inventions, 36 scientific and technical papers and 10 technical manuals.

**Colin Williams** is a Senior Research Scientist at JPL. He earned his Ph.D. in Artificial Intelligence from the University of Edinburgh in 1989, and from 1990-1995 was a Research Scientist at Xerox PARC. He was a Visiting Scholar in the Knowledge Systems Laboratory at Stanford University from 1995-1996, and an Acting Associate Professor of Computer Science at Stanford University from 2000-2002. In 1997 he joined the Jet Propulsion Laboratory, Caltech to start the Quantum Computing Technologies Group. He has published several papers in peer reviewed journals on topics ranging from computational phase transitions in constraint satisfaction problems, quantum algorithms, quantum computing hardware, circuit design software tools, and topological effects in physics. He has also written five books and been an editor of a special issue of A.I. Journal devoted to computational phase transition.

**Ryan Mackey** received his B.A. degree from the University of California at Santa Cruz (1993) for Mathematics and Physics, and went on to an M.S. (1994) and Eng. (1997) in Aeronautics at Caltech. He is presently a senior researcher and charter member of the Ultracomputing Technologies Research Group at the Jet Propulsion Laboratory. His research centers upon revolutionary computing methods and technologies for advanced machine autonomy, specifically deep space missions, UAVs and maintainable aerospace vehicles. His interests also include quantum- and biologically-inspired computing.