Applying Simulated Annealing to Problems in Model-based Diagnosis

Alexander Diedrich[1] and **Alexander Feldman**[2] and **Alejandro Perdomo-Ortiz**[3] and **Rui Abreu**[2] and **Oliver Niggemann**[4] and **Johan de Kleer**[2]

[1]Fraunhofer IOSB-INA, Lemgo, Germany
alexander.diedrich@iosb-ina.fraunhofer.de
[2]Palo Alto Research Center, Palo Alto, USA
{afeldman, rui, dekleer}@parc.com

[3]Quantum Artificial Intelligence Laboratory
NASA Ames Research Center, Moffet Field, USA
alejandro.perdomoortiz@nasa.gov
[4]Institute Industrial IT, Lemgo, Germany
oliver.niggemann@hs-owl.de

Abstract

Generating all diagnoses is computationally intractable. Therefore, many state-of-the-art approaches are incomplete. Quantum computers may however offer a solution. The first commercially available quantum computer is being used to minimize polynomials that are difficult for classical simulated annealing but easy for quantum annealing. All problems in Model-based Diagnosis (MBD) can be transformed into a polynomial minimization problem, allowing one to apply a quantum algorithm called quantum annealing to solve MBD problems. To better understand the need for this quantum approach, we designed two simulated annealing diagnostic algorithms tailored to run on a polynomial representation of MBD. These algorithms differ on their policy for random neighborhood variable selection. In addition, enhanced metrics were devised to provide more diagnostic coverage. Finally, these two simulated annealing algorithms were analyzed, empirically evaluated, and compared against state-of-the-art probabilistic methods for MBD such as SAFARI using ISCAS-85.

1 Introduction

Traditional algorithms for circuit diagnosis rely on structural descriptions such as propositional logic formulae of a circuit. In this paper we investigate whether an alternative formulation of a circuit in the form of a polynomial is more suitable for performing diagnostic tasks. Our conjecture is the following: If we translate a propositional logic formula that fully describes a boolean circuit, into an equivalent polynomial and we use this polynomial as the input to a standard minimization algorithm, is this going to be faster or more accurate than traditional diagnosis algorithms?

This problem is relevant in two ways. First, we explore a different approach to diagnosis which, if faster or more accurate, can help to advance the field of diagnostics. Second, when carrying diagnosis tasks into the quantum domain such as the D-Wave quantum computer [Perdomo-Ortiz *et al.*, 2015] we are dealing with a quantum annealing (QA)[Kadowaki and Nishimori, 1998; Farhi *et al.*, 2001] algorithm. QA received its name in relation to its classical and more famous relative: simulated annealing (SA) [Kirkpatrick *et al.*, 1983]. The only relation of these two algorithms is that both of them are sequential algorithms and use an annealing schedule as a heuristic strategy to solve combinatorial optimization problems. In SA, thermal fluctuations are used to explore the complex energy landscape, while in QA, quantum tunneling and other quantum mechanical effects are used as a computational resource. Since QA requires a polynomial instead of a propositional formula we can establish a base line to compare future quantum applications against polynomial-based diagnostic algorithms on standard PCs. When we look at traditional diagnosis algorithms such as SAFARI [Feldman *et al.*, 2010], we see that these algorithms take the model structure of a circuit into account. Therefore, these algorithms are very accurate, but require a significant amount of runtime. Thus, we think it is important to try alternative approaches in order to discover whether traditional PCs can be sped up by evaluating polynomials instead of propositional logic formulae.

With the approach described in this paper we translate a model of a boolean circuit into a polynomial that can be evaluated by standard optimization algorithms such as SA or Hill climbing. This paper makes two contributions: First, building on work in [Feldman *et al.*, 2010] we show how to translate a model of a boolean circuit on the gate-level into a propositional logic formula. From there we show how to translate the propositional logic formula into a polynomial. Second, we show how different implementations of Simulated Annealing and Random Search that use a polynomial formula compare to SAFARI that uses a propositional logic formula. Results are obtained using the ISCAS85 benchmark.

2 Related Work

Several approaches to compute a set of diagnosis candidates in the context of model-based diagnosis have been proposed in the past. In [Reiter, 1987], the au-

thors proposed a breadth-first search algorithm that uses the so called HS-trees; in [Wotawa, 2001], some improvements over the base algorithm have been suggested. All of the above algorithms make use of a constraint solver to check whether or not a set d is minimal diagnosis candidate, thus not requiring an explicit conflict set availability. While sound and complete, such algorithms do not gracefully scale to large real-world problems.

In [Feldman and Provan, 2008], the authors propose a stochastic search algorithm, that starts with a candidate d for the system and set of observations and iteratively removes elements from d while guaranteeing that the resulting set still is a candidate. In [Staal and Øhrn, 2000; Huang *et al.*, 1994] several genetic algorithms to compute diagnosis candidates are proposed. In [Abreu and van Gemund, 2009], a low-cost approximate minimal hitting set algorithm is discussed in the context of model-based diagnosis. This sequential algorithm has been improved to leverage distributed computation capabilities [Cardoso and Abreu, 2013b; 2013a]. While scalable to large problems, these algorithms do not guarantee soundness nor completeness.

The research community has considerably extended the body of recent benchmarking studies in domains ranging from space to machine learning and diagnostics [Smelyanskiy *et al.*, 2012; Rieffel *et al.*, 2015; Perdomo-Ortiz *et al.*, 2015; Benedetti *et al.*, 2016], of quantum annealers based on superconducting qubits [Johnson *et al.*, 2011; Harris *et al.*, 2010]. This paper extends this body of benchmarks by studying MBD problems and compare it to the classical/probabilistic approaches.

3 Definitions

This section provides the basic framework for the algorithms we design and analyze.

Definition 1 (Basis)**.** A basis $\mathcal{B}$ is a set of single-output Boolean functions $\{B_1, B_2, \ldots, B_n\}$.

A basis can be constructed from the common logic-gate types AND, OR, NAND, NOR, XOR, inverter, and buffer.

In this article we use the common propositional logic connectives: $\wedge$, $\vee$, $\oplus$, $\neg$, $\rightarrow$, and $\leftrightarrow$. The logical connectives implication ($\rightarrow$) and equivalence ($\leftrightarrow$) do not appear in the circuits we are diagnosing and are not part of the basis. The Boolean formula that is equivalent to an AND gate, for example, is $a \wedge b$. The semantics of all this is the usual one and is explained in any introductory logic or VLSI design book.

Logic circuits are designed by drawing gates from the basis and connecting them with wires. Wires are represented as variables. A well-formed logic-circuit design is a Direct Acyclic Graph (DAG). There are no hanging edges in the DAG (that would be a violation of the common definition for a DAG). We work with connected DAGs only. If somewhere in our algorithms a DAG gets disconnected, we remove the orphan or treat the two disconnected DAGs separately.

Definition 2 (Boolean Circuit)**.** Given a basis $\mathcal{B}$, a Boolean circuit $\mathrm{M}(\mathcal{B}) = \langle V \cup \{I^\star, O^\star\}, E\rangle$ is a DAG in which each edge $e \in E$ is a variable, each node $v \in V$ is a Boolean function drawn from $\mathcal{B}$, $I^\star$ is a primary input source, and $O^\star$ is a primary output sink.

The special primary input source and primary output sink nodes are not normally drawn in a circuit diagram. The edges that are adjacent to $I^\star$ are the primary inputs and the edges that are adjacent to $O^\star$ are the primary outputs.

A Boolean circuit, as defined, has no provisions for failure. We can allow parts of a circuit to fail by introducing extra fault variables to each logic-gate $\mathcal{B}$ and extra elementary failure functions (constraints).

Definition 3 (Fault-Augmented Model)**.** Given $\mathcal{B}$, a Boolean circuit $\mathrm{M}(\mathcal{B})$ and a second fault-augmented basis $\mathcal{B}^\star$, a fault-augmented model $\mathrm{SD}(\mathcal{B}, \mathcal{B}^\star)$ is defined as the ordered triple $\langle \mathrm{COMPS}, V, E, F\rangle$ where $\mathrm{COMPS} = \{f_1, f_2, \ldots, f_n\}$, $n = |V|$, and F is a mapping $F : \mathcal{B} \rightarrow \mathcal{B}^\star$.

Definition 3 allows us to add a fault variable to each logic gate, and depending on the value of the fault variable to allow either the original (intended) behavior of the gate or some specified faulty one.

Definition 4 (Observation)**.** An observation α is an assignment to some or all primary inputs and primary outputs of a Boolean circuit SD.

In a circuit with two 2-bit input vectors a and b and an output vector c an observation constitutes exactly one assignment to each value of a, b, and c. Observations are generated either using the fault-augmented model or are taken from real-world sensor data.

Definition 5 (Fault-Injection)**.** Given SD with fault variables COMPS, a fault-injection ϕ is an assignment to all fault variables in COMPS.

With fault-injection we take all assumable variables in a model and assign some value to them. As each assumable $f_i \in F$ implies the function of a gate, we can use F to model faulty behavior for a propositional term. For an AND gate this implication becomes: $h \rightarrow (o \leftrightarrow i_1 \wedge i_2)$. As ϕ is the set of all fault variables, we count the amount of $True$ values with-ing $|phi|$ with $|phi| = \sum_i \phi_i$, where$\phi_i = True$. We speak of a single fault, when $|phi| = 1$. A double-fault is given by $|\phi| = 2$ etc.

Definition 6 (Diagnosis)**.** Given a fault-augmented model SD with fault variables COMPS and an observation α, a diagnosis ω is defined as an assignment to all fault variables in COMPS such that $\omega \models \mathrm{SD} \wedge \alpha$.

Diagnoses are found with respect to the assumables. Given an observation α and a fault-free model of a boolean circuit in propositional logic it is possible to look at each assumable and determine the probability whether or not the associated component is faulty. Many algorithms for finding diagnoses exist. The next section shows an experimental setup to compare different algorithms by means of an example circuit.

Definition 7 (Health Estimation)**.** Given SD with fault variables COMPS, a Boolean circuit health estimation H is defined as the set of probabilities $H = \{\Pr(f = \bot)\}$ of each fault variable $f \in \text{COMPS}$ assuming the value of $\bot$.

Notice that $\Pr(f = \bot) = 1 - \Pr(f = \top)$ and that each element of H defines a binomial probability distribution function. In a Boolean circuit the health-estimation gives a value for the probability of each single component being faulty.

4 Circuit Diagnosis and Polynomial Minimization

As a running example throughout this paper we will use the two-bit adder depicted in figure 1. The adder is a combination of three AND-Gates, three XOR-Gates and one OR-Gate. a and b are the input vectors, respectively. Σ describe the output sum, while z are intermediate values. c_0 is the carry bit. The circuit is fully described through the following propositional logic formula:

$$\left|\begin{array}{l} z_0 \leftrightarrow a_0 \wedge b_0 \\ \Sigma_0 \leftrightarrow a_0 \oplus b_0 \\ z_1 \leftrightarrow a_1 \wedge b_1 \\ z_2 \leftrightarrow a_1 \oplus b_1 \\ z_3 \leftrightarrow z_2 \wedge z_0 \\ \Sigma_1 \leftrightarrow z_2 \oplus z_0 \\ c_o \leftrightarrow z_1 \vee z_3 \end{array}\right. \tag{1}$$

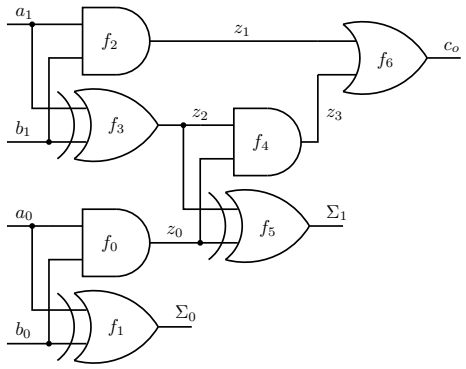

Figure 1: 2-bit adder

In order to use the described adder for SA three steps are necessary:

1. Adding fault modes to formula 1 to enable diagnostic tasks
2. Creating observations for the adder to create hypotheses for value propagation
3. Integrating the observations with the adder model to obtain a polynomial that can be used for optimization.

After obtaining a model of the boolean circuit in form of a propositional logic formula the model has to be augmented to become a fault model (see definition 3). The fault-model allows us to have a fault variable imply each boolean term $M(\mathcal{B})$. Through this it is possible to specify which gates are functioning normal and which exhibit faulty behavior. Three different fault models exist, the weak-fault models and the strong fault models in form of stuck-at-one and stuck-at-zero models, respectively.

We typically use "standard" ways in order to augment bases in order to allow faults. The simplest ones in Boolean (propositional logic) are "weak-fault-models" also known as models with ignorance of abnormal behavior. Consider, for example, the Boolean model of an AND-gate as placed in a circuit: $o \leftrightarrow i_1 \wedge i_2$. The weak-fault model of the same AND-gate would look like $h \rightarrow (o \leftrightarrow i_1 \wedge i_2)$.

A different type of fault-models are "strong-fault-models". In these cases the basis $\mathcal{B}^\star$ contains only one function, the constant $\top$ or $\bot$. In this simple case, when the output of a gate assumes a stuck-at value, we talk about stuck-at-zero and stuck-at-one models.

The propositional logic formula that fully models the 2-bit adder in a stuck-at-one model is shown in figure 1:

$$\left|\begin{array}{l} [\neg f_0 \rightarrow (z_0 \leftrightarrow a_0 \wedge b_0)] \wedge (f_0 \rightarrow z_0) \\ [\neg f_1 \rightarrow (\Sigma_0 \leftrightarrow a_0 \oplus b_0)] \wedge (f_1 \rightarrow \Sigma_0) \\ [\neg f_2 \rightarrow (z_1 \leftrightarrow a_1 \wedge b_1)] \wedge (f_2 \rightarrow z_1) \\ [\neg f_3 \rightarrow (z_2 \leftrightarrow a_1 \oplus b_1)] \wedge (f_3 \rightarrow z_2) \\ [\neg f_4 \rightarrow (z_3 \leftrightarrow z_2 \wedge z_0)] \wedge (f_4 \rightarrow z_3) \\ [\neg f_5 \rightarrow (\Sigma_1 \leftrightarrow z_2 \oplus z_0)] \wedge (f_5 \rightarrow \sigma_1) \\ [\neg f_6 \rightarrow (c_o \leftrightarrow z_1 \vee z_3)] \wedge (f_6 \rightarrow c_o) \end{array}\right. \tag{2}$$

In this formula faults can be injected by making use of different values for f. This is done through fault-injection (see definition 5). It is important that a fault-injection assigns values to all assumable variables in a model. A fault injection for the adder example in figure 1 would be $\phi = \neg f_2$, a single-fault. As definition 5 requires that all fault variables are given values, the ϕ example in the previous sentence assumes that all missing variables f_0, f_1, f_3, f_4, f_5, and f_6 are assigned the constant $\bot$.

The purpose of a diagnostic algorithm is to find fault-injections from a model and an observation. Due to the Boolean nature of our models and the limited number of observation variables there are often competing hypotheses for a fault injection. These individual hypotheses are called diagnoses.

To perform diagnosis on the fault-augmented model observations have to be made (see definition 4). In the case of the two-bit adder one observation that describes assignments to all primary inputs and outputs might be: $a_0 = 0$, $a_1 = 0$, $b_0 = 0$, $b_1 = 1$, $\Sigma_0 = 0$, $\Sigma_1 = 1$, and $c_0 = 1$. Obviously, the observation does not match the expectation of the output of a functioning two-bit adder, as the input of $2 + 0 \neq 6$.

Analytically this implies that one or more components of the circuit must be faulty. By looking at the binary representation of the output we can infer that either f_0, f_2, f_3, f_4, or f_6 is not working correctly. Having the knowledge about our fault injection from

the previous paragraph we can expect that a diagnosis algorithm should indicate a high probability that $\phi = \neg f_2$ was the injected fault and therefore that f_2 is the faulty component. Further we can expect that circuits connected to f_2 will also show a higher probability of being faulty. Thus we will end up with multiple diagnoses for a single fault.

Augmenting the fault-model (SD and COMPS) with observations gives a scenario. To create a scenario the observations are propagated through the circuit using the fault-model. The resulting propositional formula, however, has assignments for all variables with one specific observation. To perform diagnosis the scenario must be matched with the original circuit model so that one unified polynomial results.

Converting the Boolean circuit to a propositional formula, adding the observation to the formula, and finding all satisfiable solutions is not a very practical way to compute circuit diagnoses. Sometimes it is beneficiary in terms of speed to add extra constraints or to temporarily remove parts of the formula.

Another approach is to convert the Boolean circuit or the propositional formula to a polynomial expression. Table 1 shows the common propositional operators and their corresponding polynomial expressions. If, as is customary, we set the Boolean constant $\bot$ to correspond to the real number 0 and the Boolean constant $\top$ to 1, then the valuation of each polynomial in table 1 is equivalent to the valuation of its corresponding propositional operator.

Propositional Logic Operator	Polyn. Equivalent
$x \leftrightarrow y$	$1 - x - y + 2xy$
$x \rightarrow y$	$1 - x + xy$
$x \wedge y$	xy
$x \vee y$	$x + y - xy$
$\neg x$	$1 - x$

Table 1: Propositional operators to polynomial conversion table

Table 1 shows the polynomial equivalence of the most commonly used propositional operators and all other propositional operators can be reduced to these. Conversions of Boolean circuits, however, result in conjunction of small propositional expressions such as the ones in table 1. A typical system description SD looks like this:

$$\mathrm{SD} = \bigwedge_{v \in V} v \tag{3}$$

where $|V|$ is sufficiently large.

Replacing this outermost conjunction operator with a product gives a polynomial whose valuation is 0 for the $\bot$ valuation of the propositional formula and 1 for the $\top$ valuation of the corresponding formula which will work for the purpose of computing satisfiable and non-satisfiable assignments through polynomial optimization. Large products, however, lead to numerical problems and we can take the sum instead. The polynomial equivalent of Eq. 3 becomes:

$$\mathrm{SD}_{\mathrm{poly}} = \sum_{v \in V} \mathrm{poly}(v) \tag{4}$$

where $\mathrm{poly}(v)$ is the result of recursively applying the equivalent formulas shown in table 1. For example, translating the first row of equation 1 from propositional logic into a polynomial gives the following: Starting from the propositional logic formula $z_0 \leftrightarrow a_0 \wedge b_0$ we first substitute the conjunction $a_0 \wedge b_0$ to xy and then substitute the equivalence operator, resulting in the polynomial

$$1 - z_0 - a_0 b_0 + 2 a_0 b_0 z_0 \tag{5}$$

This is done for all other rows as well, finally giving the summation $\mathrm{SD}_{\mathrm{poly}}$. The value of $\mathrm{SD}_{\mathrm{poly}}$ is minimal if and only if it is a non-satisfiable solution of SD. This shows that polynomial minimization is an NP-complete problem. Polynomial minimization itself can be defined as finding the minimum of a polynomial by substituting different values for each variable.

Theorem 1. Polynomial minimization is NP-hard.

Proof. Sketch: Consider a 3-CNF formula:

$$\gamma = \bigwedge_{i=1}^{n} x_{i,1} \vee x_{i,2} \vee x_{i,3}$$

where $x_{i,1}, x_{i,1}$, and $x_{i,3}$ are positive or negative literals. There exists a transformation τ that takes any formula γ and produces an equivalent polynomial γ':

$$\gamma' = \prod_{i=1}^{n} 1 - x_{i,1} + x_{i,2} + x_{i,3} - \\ - x_{i,1}x_{i,2} - x_{i,1}x_{i,3} - x_{i,2}x_{i,3}$$

The transformation procedure from γ to γ' takes polynomial time and polynomial space.

Assume that polynomial minimization is easy. We can take a formula γ in 3-CNF and translate it to γ' as per the equations above. A valuation of γ' would be minimal iff the corresponding γ valuation is satisfiable. According to our assumption it means that it is easy to find a satisfiable solution of a 3-CNF formula, and 3-CNF satisfiability is known to be NP-hard [Woeginger, 2003] which leads to a contradiction: polynomial minimization cannot be easy. □

5 Algorithm

In this section we first describe the SA algorithm, as well as different neighborhood selection criteria. We analyze the theoretical properties of SA and its convergence behavior. We then briefly describe SAFARI and Random Search.

5.1 Simulated Annealing

Simulated Annealing (SA) models physical annealing processes that occur when a material cools down.

When the temperature of a molten block of iron is decreased, the atoms' movement decreases proportionally until some solid state is reached. If the temperature decreases too rapidly, atoms can get stuck in a non-optimal configuration and the material becomes brittle. A too slow decrease of temperature, however, would lengthen the cooling process unnecessarily. The objective is to find a rate of temperature reduction that enables the material to enter its energetically optimal configuration, while limiting the maximum length of the cooling process.

The physical annealing process can be transferred into the domain of optimization algorithms. For this an iterative approach is used which starts from an initial random state with high temperature and gradually approaches states of lower temperature. The objective is to enter a low-energy state that has an optimal configuration of its parameters. We use SA in order to find a minimal diagnosis given the observation α and the propositional logic description of a boolean circuit. Therefore, the global optimum of the polynomial, consisting of α and SD, describes the minimal diagnosis and is a subset of higher cardinality diagnoses that form local optima.

The starting point is formed by an initially random position in the search-space (i.e. a random diagnosis). From this position a neighborhood solution is calculated. If the neighborhood solution is better than the current solution it is selected for the next iteration. If, the neighborhood solution inferior to the current solution, the new solution is only accepted by a certain probability p. p is calculated by the difference between the initial and the neighborhood solution, divided by the current temperature. This approach facilitates that contrary to Hill-climbing or gradient-descent methods, SA can overcome local optima by "jumping" over them (i.e. temporarily assuming a worse solution). With decreasing temperature the likelihood of accepting an inferior solution becomes smaller. As the likelihood for accepting inferior solutions goes against zero, the overall solution can only become better until an optimum or the maximum amount of steps is reached.

The algorithm is presented in listing 1. The search-space is represented as a directed acyclic graph $G = (V, E)$ with V being the operators and E being the variables of the boolean expression . The initial state is constructed through a random assignment of real values $[0, 1]$ to all variables. All subsequent solutions are attained by randomly increasing or decreasing a value by 0.05 in the range $[0, 1]$. $f(.)$ constitutes the objective function that evaluates the polynomial. A neighborhood solution is defined by either taking parent-child nodes from a graph, or randomly selecting variables from the polynomial.

Graph-based Neighborhood

Select one node $v \in V$ and randomly increase or decrease their value by a predefined amount. Then do the same for each child node $child(v)$. Here we assume that connected nodes have a higher influence on the overall solution than disconnected nodes in the graph. As adjacent nodes in a graph make up a local neighborhood, changing their value corresponds to an overall neighborhood solution.

Random Neighborhood

Another approach is to randomize the neighbor selection. Therefore only the value of a predefined percentage $r \in [0, 1]$ of randomly selected nodes v is changed. Connections between nodes and parent-child relations are disregarded.

Theoretical Properties

Given infinite time it has been proven that SA will converge to a global optimumn [Rajasekaran, 1990]. Using SA with timing constraints, however, only guarantees that one solution is found. It is not known if this solution is a local or a global optimum. Each local optimum is a diagnosis, while each global optimum is a minimum cardinality diagnosis. From this follows that the algorithm is not sound, as local optima may contain conflicts that are not a valid diagnosis. The notion of completeness is violated, since SA generates exactly one solution per run, it cannot be guaranteed that it will eventually find all valid solutions.

Convergence of the algorithm cannot be guaranteed when using a time-constrained version of SA, as the run might be aborted before a global optimum of the objective function was encountered. Running the algorithm time-bounded is necessary, however, because SA has an exponential runtime dependent on the number of input parameters.

Convergence

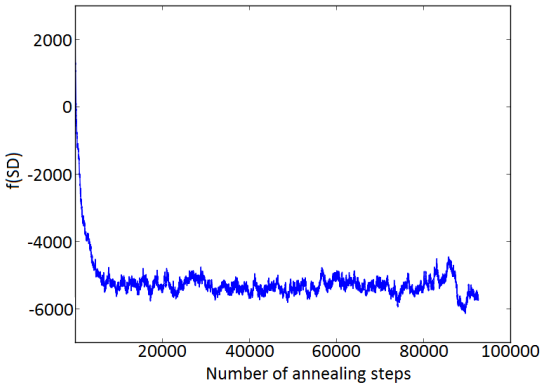

Figure 2: Output of the SA algorithm graph-based

Figure 2 depicts the results of SA using a graph-based neighborhood selection. It shows that the output value progresses in an inverted exponential manner until it converges to a certain value. At approximately 90000 steps a global optimum was found. It is not known, however, whether this is the only optimum (i.e. lowest energy state) or if there would be another optimum at a later time.

Algorithm 1 Simulated Annealing

```
function doSimulatedAnnealing
    current ← initial()
    best ← current
    while temp ≥ 1 do
        if msteps = 0 then
            T ← decreaseT()
            msteps ← mstepsInit
        end if
        new ← neighbor(current)
        accept = e^((f(current)−f(new))/T)
        if f(new) < f(current) then
            current ← new
            if f(new) < f(best) then
                best ← new
            end if
        else if accept > random(0, 1) then
            current ← new
        end if
    end while
    return best
end function
function neighbor(state)
    node ← selectRandomNode(state)
    child ← selectRandomChild(node)
    state ← changeValue(node)
    if exists(child) then
        state ← changeValue(child)
    end if
    return state
end function
```

5.2 Comparison Algorithms

This section introduces a random search algorithm and SAFARI. These were compared to SA in order to determine the advantages and disadvantages between the algorithms. Random search naively guesses new solutions, while SAFARI uses a stochastic approach.

Random Search

The random search algorithm takes as an input a graph-based representation of the canonical formula. It executes in a loop until a maximum amount of time has elapsed. In each iteration the algorithm takes one node from the graph and changes the node's value as well as the value of all children of the node. If the generated solution is better than the previous one it is kept, otherwise the algorithm continues with the next iteration. Since the random search algorithm is simply guessing a solution we use it as a baseline to compare against.

In Figure 3 it is evident that the random search algorithm does not converge but instead oscillates between different values. This results from the algorithms' design as it continuously generates random solutions. As only a small fraction of nodes are changed in each step, the output value cannot perform large jumps. Furthermore, as the algorithm uses no convergence criterion by definition, the output value will continue oscillating until the algorithm is stopped.

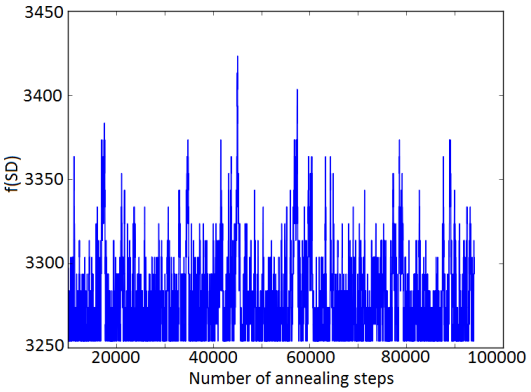

Figure 3: Output of the random search algorithm

SAFARI

SAFARI is a greedy stochastic hill climbing algorithm for approximating the set of minimal diagnoses in a diagnostic system. Its input consists of a diagnostic system description given as a fault-augmented propositional logic formula and an observation. SAFARI starts with a random diagnosis. Taking the random diagnosis as a baseline SAFARI iteratively improves the diagnosis by randomly "flipping" fault literals in the propositional logic formula. This is a one-way process so that literals that were flipped from faulty to healthy will not be flipped back. Each outcome is consistency checked, which results in diagnoses with strongly monotonically decreasing cardinality. This process is restarted N times to start from N different random initial diagnoses. The number of "flips" and the number of restarts N can be adjusted to optimally cover the search space to find minimal cardinality diagnoses.

6 Experiment

The experiment was performed on a single Intel Xeon CPU with 3.3 GHz and 1.5 TB RAM, running a 64-Bit Ubuntu 14.04. As test instances the ISCAS85 benchmark is used.

6.1 Benchmark

We used the ISCAS85 benchmark [Brglez, 1985] as the input to the algorithms. The ISCAS85 benchmark consists several combinatorial circuits that can be used to measure the performance of testing and diagnosis algorithms. Included in the benchmark are circuits such as decoders, multipliers, cryptography circuits, and arithmetic logic units with the number of gates per circuit varying between 160 and 3512.

6.2 Results

Although the propositional logic formula that is used with SAFARI was converted into a polynomial, the results show that the algorithms specifically designed to work with propositional logic achieve far better results than traditional optimization techniques using polynomials. For all algorithms a fixed runtime was defined.

		Run 1	Run 2	Run 3	Run 4	Run 5
Parameters	Temperature	20	20	30	10	10
	Steps	10	4	20	20	5
	Runtime	20	20	60	20	20
	Iterations	5	5	5	50	50
SA Graph	Steps completed	900	1800	95000	47000	45000
	Isolation acc.	32.4	32.3	31.4	32.2	32.8
SA Random	Steps completed	800	1700	85000	42000	42000
	Isolation acc.	20.1	19.9	19.6	20.0	20.0

Table 2: Runs with different parameters for SA using either the graph-based neighborhood selection (SA Graph) or a random selection of nodes (SA Random).

Upon exceeding the time limit the by then achieved parameters were taken as the solution.

We define the following metric to denote the quality of a solution: *Isolation accuracy* M_{ia} describes the quality of a solution given the health state $H = \{\Pr(f = \perp)\}$ (Definition 7) and a fault injection ϕ. We calculate the Euclidean distance between H and ϕ to determine how close the output of an algorithm is to the original fault injection [Stern *et al.*, 2015].

$$M_{ia} = \sqrt{\sum_{i=1}^{|H|} (\Pr(f_i = \perp) - \Pr(p_i = \perp))^2} \quad (6)$$

With $f_i \in H$ and $p_i \in \phi$ and $M_{ia} \in \mathcal{R}$. Smaller values of M_{ia} indicate that the calculated health state is closer to the fault injection. From this we can see how good the diagnosis algorithm was able to determine the actual injected fault. Bigger values on the other hand indicate that the algorithm found more root-causes for faults than were in the fault-injection ϕ, which means that false-positive diagnoses are generated.

Table 2 shows the results the two approaches to SA. The initial temperature was chosen in order to achieve a reasonable probability for accepting worse states during cool-down of the annealing algorithm. The number of steps indicates how many annealing steps the algorithm performs until the temperature is decreased. The number of iterations specifies how often the algorithm was started for a specific observation α and a given polynomial p. The outcome of the SA algorithm is the average value of all iterations. From the table it can be concluded that the more iterations are performed, the better the isolation accuracy becomes. This can be explained through statistical analysis. As the average value for determining the accuracy becomes closer to the real value, the more iterations are performed.

Contrary to the assumption neither implementation of simulated annealing is able to beat SAFARI in terms of speed or accuracy. While isolation accuracy for SA is around 32 and 20, respectively, SAFARI achieves an isolation accuracy of approximately 3. For these instances Random Search achieves an accuracy of approximately 10. The low value of isolation accuracy for Random Search can be explained through the algorithms "lucky" guessing and finding a good solution given a sufficient amount of runs. For SA, we can guarantee that it will hit a global optimum eventually.

7 Conclusion

Before MBD problems can be mapped to the D-Wave quantum annealers or to any realization of QA, they have to be mapped to a polynomial representation. Since the field of MBD operates more naturally in a propositional logic representation, a natural question is to evaluate the performance of canonical algorithms such as SA working directly at the polynomial representation. The focus of our paper is to provide such a baseline study, comparing the performance of algorithms working in these two representation paradigms. Here, we demonstrate the methodology for solving MBD circuit diagnostics with SA under this new polynomial encoding. This contribution compares SA to state-of-the-art diagnostic solvers such as SAFARI and discusses performance and optimality differences. We also compare random search to SA to validate the use of an annealing scheme. The isolation accuracy is tested on a popular benchmark, but it was found that the SA accuracy is far inferior to a targeted SAFARI search. The difference in isolation accuracy between SA and the near-optimum (i.e.SAFARI) makes us believe that QA would be a valid approach for solving MBD problems and showing quantum speed-up.

Quantum optimization opens an entirely new area for research in MBD. Quantum computers based on this principle may provide some valuable insights and may offer answer to fundamental questions in MBD, AI, and computational complexity theory. Currently, we are exploring whether QA, e.g., with D-Wave-based circuit diagnostics, can exhibit quantum speed-up for benchmarks based on combinational circuits, as the ones studied here.

8 Acknowledgments

We thank Nora Boettcher for her valuable input in making this paper more readable. Alejandro Perdomo-Ortiz acknowledges support by the Intelligence Advanced Research Projects Activity (IARPA), the Office of the Director of National Intelligence (ODNI),

via IAA 145483. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

References

[Abreu and van Gemund, 2009] R Abreu and AJC van Gemund. A low-cost approximate minimal hitting set algorithm and its application to model-based diagnosis. In *SARA*, volume 9, pages 2–9, 2009.

[Benedetti *et al.*, 2016] M Benedetti, J Realpe-Gómez, R Biswas, and A Perdomo-Ortiz. Estimation of effective temperatures in quantum annealers for sampling applications: A case study with possible applications in deep learning. *Physical Review A*, 94:022308, Aug 2016.

[Brglez, 1985] F. Brglez. A neutral netlist of 10 combinational benchmark circuits and a target translation in fortran. *ISCAS-85*, 1985.

[Cardoso and Abreu, 2013a] N Cardoso and R Abreu. A distributed approach to diagnosis candidate generation. In *Portuguese Conference on Artificial Intelligence*, pages 175–186. Springer, 2013.

[Cardoso and Abreu, 2013b] N Cardoso and R Abreu. Mhs2: A map-reduce heuristic-driven minimal hitting set search algorithm. In *Proceedings of the International Conference on Multicore Software Engineering, Performance, and Tools-Volume 8063*, pages 25–36. Springer-Verlag, 2013.

[Farhi *et al.*, 2001] E Farhi, J Goldstone, S Gutmann, J Lapan, A Lundgren, and D Preda. A quantum adiabatic evolution algorithm applied to random instances of an NP-Complete problem. *Science*, 292(5516):472–475, April 2001.

[Feldman and Provan, 2008] A Feldman and G Provan. Computing minimal diagnoses by greedy stochastic search. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, AAAI'08, pages 911–918, 2008.

[Feldman *et al.*, 2010] A Feldman, G Provan, and A Van Gemund. Approximate model-based diagnosis using greedy stochastic search. *Journal of Artificial Intelligence Research*, 38:371–413, 2010.

[Harris *et al.*, 2010] R Harris, MW Johnson, T Lanting, AJ Berkley, J Johansson, P Bunyk, E Tolkacheva, E Ladizinsky, N Ladizinsky, T Oh, et al. Experimental investigation of an eight-qubit unit cell in a superconducting optimization processor. *Physical Review B*, 82(2):024511, 2010.

[Huang *et al.*, 1994] WC Huang, CY Kao, and JT Horng. A genetic algorithm approach for set covering problems. In *International Conference on Evolutionary Computation*, 1994.

[Johnson *et al.*, 2011] MW Johnson, MHS Amin, S Gildert, T Lanting, F Hamze, N Dickson, R Harris, AJ Berkley, J Johansson, P Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, May 2011.

[Kadowaki and Nishimori, 1998] T Kadowaki and H Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355, November 1998.

[Kirkpatrick *et al.*, 1983] S Kirkpatrick, CD Gelatt, and MP Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[Perdomo-Ortiz *et al.*, 2015] A Perdomo-Ortiz, J Fluegemann, S Narasimhan, R Biswas, and VN Smelyanskiy. A quantum annealing approach for fault detection and diagnosis of graph-based systems. *The European Physical Journal Special Topics*, 224(1):131–148, 2015.

[Rajasekaran, 1990] S Rajasekaran. On the convergence time of simulated annealing. 1990.

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

[Rieffel *et al.*, 2015] EG. Rieffel, D Venturelli, B O'Gorman, MB. Do, EM. Prystay, and VN. Smelyanskiy. A case study in programming a quantum annealer for hard operational planning problems. *Quantum Information Processing*, 14(1):1–36, 2015.

[Smelyanskiy *et al.*, 2012] VN Smelyanskiy, EG Rieffel, SI Knysh, CP Williams, MW Johnson, MC Thom, W G Macready, and KL Pudenz. A near-term quantum computing approach for hard computational problems in space exploration. *arXiv:1204.2821*, 2012.

[Staal and Øhrn, 2000] AV Staal and Aleksander Øhrn. Minimal approximate hitting sets and rule templates. *International Journal of approximate Reasoning*, 2000.

[Stern *et al.*, 2015] RT Stern, M Kalech, S Rogov, and A Feldman. How many diagnoses do we need? In *AAAI*, pages 1618–1624, 2015.

[Woeginger, 2003] GJ Woeginger. Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization: Eureka, You Shrink!*, pages 185–207. Springer, 2003.

[Wotawa, 2001] F Wotawa. A variant of Reiter's hitting-set algorithm. *Information Processing Letters*, 79(1):45–51, 2001.