Performance Modeling of Stochastic Diagnosis Engines

T.P.M. Janssen

Performance Modeling of Stochastic Diagnosis Engines

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

T.P.M. Janssen
born in Nijmegen, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Cover picture: The diagnostics engine of Apollo 13: a group of eight astronauts and flight controllers monitoring the console activity in the Mission Operations Control Room (MOCR) during Apollo 13. When this picture was taken, the Apollo 13 moon landing had already been canceled, and the crew was attempting to bring their crippled spacecraft back home.
The Apollo 13 space craft suffered from a quintuple fault: five simultaneous defects resulting in hard to diagnose sensor readings.
This picture was taken from the Johnson Space Center website at `www.jsc.nasa.gov`.

Performance Modeling of Stochastic Diagnosis Engines

Author: T.P.M. Janssen
Student id: 1053469
Email: tjanssen@scarabee.nl

Abstract

Critical systems are complex, consisting of thousands of components, which can fail at any time. Diagnosing these systems within a certain time is highly desirable. Traditional diagnosis algorithms are mostly deterministic, able to find single faults extremely fast and double faults reasonably quick as well. However, these algorithms fail to find diagnoses fast enough in cases where there are three or more components failing simultaneously. A stochastic algorithm, like SAFARI, is able to diagnose these problems in reasonable time. However, stochastic algorithms are unable to guarantee optimality and completeness of the returned diagnoses. In this thesis we analyze the behavior of the SAFARI algorithm, introducing a characterization of performance. We provide a performance model for this stochastic algorithm and we propose a termination criterion which guarantees a certain level of completeness of the most important set of diagnoses.

Thesis Committee:

Chair:	Prof. dr. ir. A.J.C. van Gemund, ST/ES, Faculty EEMCS, TU Delft
University supervisor:	Prof. dr. ir. A.J.C. van Gemund, ST/ES, Faculty EEMCS, TU Delft
Committee Members:	Dr. ir. H.G. Gross, ST/SE, Faculty EEMCS, TU Delft
	Prof. dr. ir. C. Witteveen, ST/ALG, Faculty EEMCS, TU Delft
	Dr. ir. A.B. Feldman, IICT - Lausanne, Switserland

Preface

Finally, is the word that first comes to mind, when writing these last words of my thesis. It is not that I have failed to enjoy the journey towards its completion. On the contrary, I truly loved the topic, the research, the discussions with my supervisor and finding solutions to difficult problems. However, mostly due to personal circumstances, this journey was very lengthy. Although I wish I have finished it much sooner, I am very proud of what I have accomplished. Maybe even more so, considering the obstacles along the way.

I would like to thank Alex Feldman for providing the foundations for my research, as my topic is the logical follow-up of his fine work on the stochastic search engine SAFARI. Very special thanks go to my supervisor Arjan van Gemund, whom I consider to be a mentor in more than one way. I really appreciate the many conversations on varying topics and his ability to stimulate thinking out of the box. Without his supervision, I am sure the journey would have been even longer.

I would also thank my dear parents, for supporting me along the way. And finally, many thanks go to my wife Wendy for being this patient and to my daughter Livia, just for being there.

T.P.M. Janssen
Delft, the Netherlands
December 28, 2010

Contents

List of Figures

Chapter 1

Introduction

Todays systems are becoming increasingly complex. For example, vehicles, satellites and power plants all consist of thousands of components. A failure of one or more of these components will likely cause the complete system to behave differently. One important reason for locating the fault is to ensure safety or continuous operation of a system. In cases like (aero)space and automotive systems or critical control systems the faults need to be identified as soon as possible in order to be fixed or circumvented. For example, if a nuclear power plant shows some unexpected behavior, like an extremely high reading of one of the sensors, the source of this problem needs to be found as soon as possible. It could be that a part of the installation is overheating and action needs to be taken to cool it down, or it could be a defect sensor giving erroneous information. One error could lead to other errors, which in turn lead to yet more errors, resulting in an *alarm avalanche*, flooding the operators with so many problems that it becomes nearly impossible to extract the root cause in time. Another reason is cost in terms of money and time. There are many complex systems that cost a great amount of money for each minute that it is not in operation. For example, wafer steppers, used in the production of integrated circuits, play a very dominant role in the process, which results in great production losses and thus great costs when unavailable. For companies it is crucial to locate and repair a fault of such systems as soon as possible. Due to the complexity of these systems it is hard to locate and repair a system fault whenever this occurs. Therefore, diagnosis approaches are needed that pair high diagnosis accuracy to low (algorithmic) cost.

1.1 Model Based Diagnosis

Model-based diagnosis (MBD) is a diagnosis approach that requires a model of the system under investigation. The general definition of a diagnosis [12] states that an evaluation of a system model together with observations on the system, results in the possible explanations of the observed behavior:

- A (model of a) system is defined as a set of components and a description of how these components interact. This behavioral model can simulate the workings of the

actual system in nominal mode and it can simulate the system when components malfunction (fault modes). In the general definition this model is typically represented in first order logic.

- Observations on (parts of) the system are defined as additional constraints on the system model. These observations are measurements at specific points of the system. For example, they could be the input and output of a digital circuit or pressure and temperature gauge values of a nuclear power plant. The observations on the actual system will not be in conflict with the model if it is working correctly.

- A diagnosis is defined as the set of faulty components, given a system model together with the constraints of an observation. A diagnosis explains the observed behavior, given that the system model is valid. Typically, a diagnosis engine returns multiple *candidates*, and these diagnoses can be of different fault *cardinality* (number of faulty components). Usually one wants minimal diagnoses (the lowest fault cardinality), as it has higher probability of being the real fault.

The conventional algorithms for diagnosing these faults are deterministic and are fast in finding low cardinality faults, e.g., when there is only one fault in the system. A popular deterministic approach used in model-based diagnosis is the conflict-directed A* (CDA*) algorithm [14], which is built upon the concepts of the general diagnostic engine (GDE) [3]. This method tries to find the best next candidate while continuously decreasing the search space by checking for inconsistencies. A similar conflict-directed best first search algorithm is adopted by the Livingstone kernel [13], which is successfully used in several missions of NASA [1, 11]. Another state-of-the-art approach to model-based diagnosis, called HA* [7], exploits the hierarchy of system models. It consists of an expensive pre-processing step and a very fast diagnosis step. Because the exponential cost of pre-processing is done only once for a model, this is an attractive approach.

However, when multiple components fail at the same time, these deterministic algorithms suffer from poor performance due to the fact that time complexity is exponential in the number of faults. A popular example of a multiple component failure in the field of space exploration is the Apollo 13 crisis, as mentioned in [11]. In this situation no less than five faults occurred in the system simultaneously: three electrical shorts, the bursting of a tank-line, and a pressure jacket. With little sensor information, diagnosing the health state of the system was a huge challenge for ground control.

In todays systems, consisting of thousands of components, a quintuple fault could take weeks to diagnose using conventional algorithms.

1.2 SAFARI

SAFARI (StochAstic Fault diagnosis AlgoRIthm) [6] is developed to overcome the time complexity of the conventional models. Being a stochastic algorithm, the time to solve a problem becomes roughly constant, independent of the number of simultaneous faulty components in the system. It has been shown [5] that SAFARI is able to produce possible

diagnoses for difficult problems that deterministic algorithms are not able to diagnose within a reasonable amount of time (weeks or months).

The downside of applying a stochastic algorithm is the sacrifice of optimality and completeness. A deterministic method starts bottom up, first trying to explain the observations by a single faulty component. After all single fault solutions are visited, then combinations of two faulty components are tried, etcetera. This exhaustive approach will always return the minimal set of faulty components that explain the observed behavior. Therefore, deterministic methods are always optimal. A deterministic method is able to produce all possible diagnoses, making it complete. A stochastic method, however, has the risk of failing to find the diagnosis with the least faulty components, purely by chance. The more runs that a stochastic algorithm executes, the higher the probability of having found this diagnosis. Although possible diagnoses are produced continuously by SAFARI, we can not be certain that all the relevant diagnoses are found, especially that one diagnosis that turns out to be the actual one. The consequence is that SAFARI needs to run for an *unknown* amount of time, unless we can somehow state something about the probability of having found all relevant diagnoses.

Currently, there is no clear model to accurately predict the behavior of SAFARI. Consequently, we are not able to indicate the level of completeness or optimality of the diagnoses found at any given moment by SAFARI. A proper model of the SAFARI algorithm is required to understand the probability of generating a particular diagnosis and to aid in minimizing the uncertainty.

1.3 Problem Statement

We define the *performance* of a stochastic diagnosis algorithm in terms of the probability density of the low cardinality solutions, as the solutions of low cardinality have the highest probability of being the correct diagnosis. If a stochastic diagnosis engine has a high probability of returning a solution of low cardinality, then we consider the performance of the algorithm to be good. For example, for an imaginary problem there exists a pool of possible solutions with a Gaussian-like cardinality distribution. An algorithm with bad performance would randomly pick a solution from this pool. An algorithm with good performance would be able to return low cardinality solutions with higher probability. This is illustrated in Figure 1.1.

From the previous section it becomes clear that the applicability of the SAFARI algorithm is limited due to the uncertainty of completeness and optimality of the results. A correct *performance model* of the algorithm can give an indication of the optimality and completeness during the diagnosis process. This model may lead to a prediction of the behavior and a termination criterion that assures that the probability of finding a new relevant diagnosis is sufficiently small. The problem statement therefore becomes:

> *Can we find a correct performance model for the* SAFARI *algorithm and from this model devise a termination criterion, in order to provide a proper termination condition of the diagnosis process, while ensuring a certain degree of completeness of the returned diagnoses?*

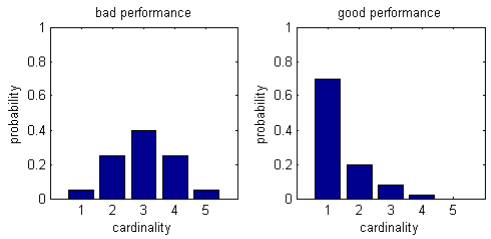

Figure 1.1: (a) Bad performance: probability density highest at medium cardinalities. (b) Good performance: probability density highest at low cardinalities

1.4 Contributions

In this thesis an analysis is made of the problem mentioned above and possible solutions are explored and discussed. The contributions of this thesis are listed below.

- The current performance model of SAFARI as defined in [5], dubbed A_0 in this thesis, is proven to be insufficient to form a basis for a termination criterion, as it disregards the impact of fault cardinalities other than the target cardinality.
- A characterization of performance S is introduced, that will categorize the many solutions possible for a system into a relatively small set, that is used to further analyze and model the SAFARI algorithm.
- The characterization of performance S is shown to be the determining factor of the behavior of SAFARI.
- Different models of the SAFARI algorithms are explored, each model having a different level of abstraction from the actual algorithm. These models, A_1, A_2, and A_3, are compared and one model is selected as best candidate for modeling SAFARI.
- Having analyzed and modeled the SAFARI algorithm, a criterion is proposed to terminate the algorithm while ensuring a certain level of completeness.

1.5 Thesis Outline

The remainder of the thesis is organized as follows. Chapter 2 gives an introduction to model-based diagnosis and explains some basic concepts, like healthy and faulty components, and weak and strong models. Also, in this chapter a running example is introduced that will be used throughout the thesis to illustrate concepts and validate models. Chapter 3 discusses the SAFARI algorithm, the current SAFARI performance model, A_0, and details on the limitation of this model. Chapter 4 and 5 contain the main contributions of this paper. Chapter 4 introduces the characterization S of SAFARI performance and shows that it essentially determines the behavior of SAFARI. Also, in this chapter three new models, A_1, A_2

and A_3, are presented and compared. Chapter 5 discusses termination of the SAFARI algorithm. It shows how the models can be used to predict the characteristic S and it introduces a termination criterion for the SAFARI algorithm. Chapter 6 concludes the thesis with an overview of the results of the work done and the work still remaining on the subject.

Chapter 2

Model Based Diagnosis

In this section the concept of model based diagnosis is explained and a small-sized system is introduced, which serves as a running example used throughout the thesis to illustrate many of the concepts, algorithms and models that will be discussed. Since this example system is thoroughly examined it also serves as a basis for experiments. Furthermore, all relevant terms are explained in this section.

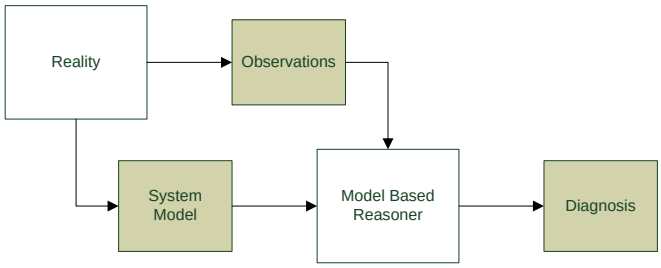

Figure 2.1: Process of Model Based Diagnosis.

Model Based Diagnosis (MBD) [4, 12] is a form of abductive reasoning using a model of the system under investigation. This simplified representation captures the relevant behavior of the real system. Certain points on the actual system can be measured and together with the system model this *observation* on the real system is used as input to a *diagnostic reasoner* in order to obtain possible explanations of (unexpected) behavior of the actual system. This process is depicted in Figure 2.1.

For example, consider MBD on an example digital circuit consisting of three inverter components, as depicted in Figure 2.2. This circuit has one input (x) and two outputs (y and z), which can be observed externally. Internally, it has an unobservable variable w, which serves as the output of inverter 1 and the input of inverters 2 and 3. Let M denote the *size* of a system, or the number of components, which in this case is equal to $M = 3$. Let h be the *health variable* of a component. For a boolean health variable, $h = 1$ or h states that the component is *healthy*, i.e., the component behaves as expected, and $h = 0$ or $\neg h$ denotes an unhealthy or *faulty* component. The nominal behavior, i.e., the case where all

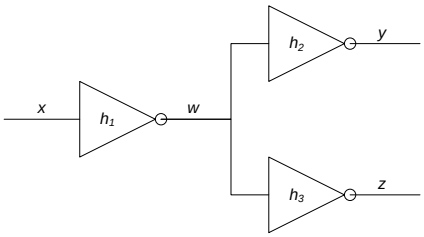

Figure 2.2: Three-inverters digital circuit.

components are healthy ($h_i = 1, \forall i$), would cause the outputs y and z to be equal to the input x. However, we assume that each component in the system can be at fault, leading to 2^M possible fault modes of the system. The aim of MBD is to infer the fault modes that explain an observation, using a model of the system.

Given an observation $x = 1$, $y = 0$, and $z = 1$ on the three-inverter system, the obvious explanation would be $\neg h_2$, i.e., the component leading directly to $y = 0$, inverter 2, is at fault. Another possible explanation would be $\neg h_1 \wedge \neg h_3$, where both inverter 1 and inverter 3 have incorrect behavior. Both diagnoses are correct, given the measured variables and what we know about the nominal system behavior. Knowing the value of the internal variable w would be the key to solve the problem of multiple diagnoses, since w would assume a different value for each diagnosis. However, uncertainty and incomplete knowledge of a system is reality and the set of possible solutions to the problem must be returned by a diagnosis engine.

The above diagnosis of the three-inverter digital circuit can be formalized using the logic model

$$h_1 \rightarrow w \Leftrightarrow \neg x$$
$$h_2 \rightarrow y \Leftrightarrow \neg w$$
$$h_3 \rightarrow z \Leftrightarrow \neg w$$

where each inverter is modeled by stating that if the component is healthy, the output equals the inverted input. Applying the observation $x = 1$, $y = 0$, and $z = 1$ on this model, it follows

$$(\neg h_1 \vee \neg w) \wedge (\neg h_2 \vee w) \wedge (\neg h_3 \vee \neg w) \tag{2.1}$$

Converting this CNF (conjunctive normal form) expression to DNF (disjunctive normal form) yields

$$(\neg h_1 \wedge \neg h_2 \wedge \neg h_3) \vee (\neg h_1 \wedge \neg h_2 \wedge \neg w) \vee (\neg h_1 \wedge \neg h_3 \wedge w) \vee (\neg h_2 \wedge \neg h_3 \wedge \neg w) \vee (\neg h_2 \wedge \neg w) \tag{2.2}$$

which reduces to the minimal DNF expression

$$(\neg h_2 \wedge \neg w) \vee (\neg h_1 \wedge \neg h_3 \wedge w) \tag{2.3}$$

This results in the same explanations as given earlier, with the addition that the internal variable w is also assigned a value with each solution. Thus, given the observation of x, y, and z, either inverter 2 is faulty and the internal variable $w = 0$, or inverters 1 and 3 are faulty and $w = 1$.

In the formal theory of diagnosis of the three-inverters system, the DNF (2.2) shows all five consistent solutions to the problem. One of these solutions is the trivial solution of all components being unhealthy. This is consistent with the model, since the components are modeled *weakly*, that is, only the behavior of a *healthy* component is described, leaving the behavior of an faulty component undefined. On the other hand, *strong* models would also describe the behavior of an unhealthy component, such as the component being stuck-at-zero, which means that a faulty component always outputs a low signal. The difference between a weak model and a strong model is shown in Table 2.1 where a logic and-gate is modeled in both ways. Together with the fact that internal variables exist that can not be observed, weakly modeled components cause supersets of solutions to exist. Considering the DNF result (2.2) of the three-inverters problem, three of the five solutions are subsumed by $(\neg h_2 \wedge \neg w)$. The reduced DNF (2.3) contains the *minimal solutions* to the problem, which are not subsumed by any other solution.

weak			strong (stuck-at-zero)		
h	$\rightarrow$	$o \Leftrightarrow (i_1 \wedge i_2)$	h	$\rightarrow$	$o \Leftrightarrow (i_1 \wedge i_2)$
			$\neg h$	$\rightarrow$	$\neg o$

Table 2.1: Weak and strong model of a logic and-gate

The three-inverters example results in two minimal solutions, which differ in the number of components that are faulty. When assuming equal probability of components being unhealthy, the number of faulty components (*fault cardinality C*) is used to determine the most probable solution. In this case, the $C = 1$ solution $(\neg h_2 \wedge \neg w)$ is the most probable, since the probability that only one inverter (h_2) is broken, is greater than the probability of the two other inverters being broken (assuming components fail independently, a standard assumption in diagnosis). Therefore, in diagnostic problems, we are usually interested in the *minimal cardinality* (*MC*) solutions.

A diagnostic reasoner employs an algorithm to get from a CNF representation of the problem (2.1), which is relatively easy to extract from a model description and observations, to the list of solutions (2.2) or at least a partial list containing the lower cardinality solutions. Using exhaustive search would imply trying all 2^M possible fault modes, which is prohibitive for a realistic system. As mentioned earlier, diagnostic reasoning algorithms can be divided in deterministic, stochastic and hybrid algorithms. Deterministic algorithms, like GDE (General Diagnostic Engine) and CDA* (Conflict Directed A*), are expensive but complete. Stochastic algorithms, like the SAFARI algorithm, are cheaper, but incomplete. Completeness is important for diagnostic applications, especially completeness of lower cardinality solutions, since finding all reasons of failure that are most probable is key to repairing the system. If a stochastic reasoner fails to return one of the most probable solutions, which in fact turns out to be the true cause of the system failure, this method is considered unreliable and will therefore not be used by diagnosticians. However, a second requirement for most diagnostic applications is that the diagnosis is returned within a reasonable amount of time. If the minimal cardinality *MC* of a problem is large, deterministic methods will take an extremely large amount of time to return even one of the possible so-

lutions and might even be unable to return a solution given a time or space constraint. A stochastic method, however, does not guarantee completeness, but can guarantee relatively quick solutions at roughly regular intervals, independent of *MC*.

2.1 Running Example: 74180

In this thesis the 74180 logic circuit serves as a running example and is the basis for experimental research, because it is a relatively small, but real system, which is thoroughly analyzed. The 74180 circuit is an 8-bit parity generator/checker, of which the logic diagram is given in Figure 2.3. This integrated circuit is typically used to generate and check parity on data being communicated over a lossy connection.

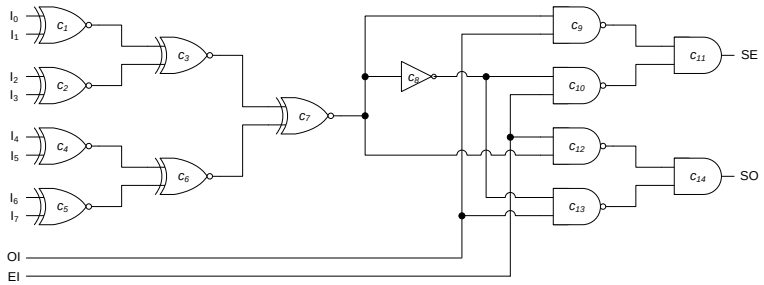

Figure 2.3: Diagram of the 74180 system

The system has an 8-bit data input ($I_0 - I_7$), an odd input (*OI*) and an even input (*EI*). The outputs (*SE* and *SO*) signify the parity (even or odd) of the data input and parity control input. This system can be cascaded to allow an unlimited word length for the data input. The intended behavior of the system is given in a truth table (Table 2.2).

sum of 1's in I_0 thru I_7	*IE*	*IO*	*SE*	*SO*
EVEN	1	0	1	0
ODD	1	0	0	1
EVEN	0	1	0	1
ODD	0	1	1	0
irrelevant	1	1	0	0
irrelevant	0	0	1	1

Table 2.2: Truth table for the 74180 system

The example 74180 circuit is modeled by modeling each logic gate as a separate component and by connecting them as depicted in Figure 2.3. As with the three-inverters example, each component can be healthy, behaving as expected, or it can be faulty, where the behavior of the component may be inconsistent with its modeled nominal behavior. Each logic gate is modeled weakly and lets the behavior of an unhealthy component be undefined. The

example system contains $M = 14$ logic gates, denoted $c_1 - c_{14}$ in Figure 2.3, each of which can break and may cause unexpected behavior of the total system. The system can only be observed at the twelve variables at its interface. If some input results in unexpected output, then it can be assumed that one or more internal components are at fault. Using the observation on the system, in this case the inputs and outputs, and a model of the system, the components that are the most probable candidates to be broken can be inferred. For example, consider the following observation on the system:

$$o_1 = \{I = 11111111, EI = 0, OI = 0, SE = 0, SO = 1\}$$

It states that all data input lines ($I_0 - I_7$) are high, which results in an even number of 1's, EI and OI are both low. According to the truth table of the system (Table 2.2), both SE and SO should be set to high. However, with o_1 it is observed that only SO is high and that SE is low. If the system would be diagnosed with this observation it would turn out that either one of the components c_9, c_{10} or c_{11} is broken, (i.e., three single faults). Internally, from the point of view of a diagnostic reasoner, this is represented by a *health vector*. This is an array of health variables, 1's and 0's each relating to a different component, where a 1 signifies a healthy component and a 0 signifies that that component is faulty. The health vectors of the resulting solutions which explain the example observation would be

$$d_1 : h = (11111111011111)$$

$$d_2 : h = (11111111101111)$$

$$d_3 : h = (11111111110111)$$

where all components are healthy except for c_9, c_{10}, c_{11}, respectively.

A health vector of all 1's would denote nominal modeled behavior, where each component is healthy. Since a weak model of a component does not model the faulty behavior, any behavior is consistent with a faulty component. This implies that the trivial health vector of all 0's, d_t, where all components are faulty, explains every possible behavior of the system. The solutions of interest, however, are the minimal solutions, as discussed earlier. Given the following health vectors of existing solutions

$$d_t : h = \{00000000000000\}$$

$$d_0 : h = \{11111111001111\}$$

$$d_1 : h = \{11111111011111\}$$

$$d_2 : h = \{11111111101111\}$$

solution d_0 is subsumed by the minimal solutions d_1 and d_2, and the trivial solution d_t is subsumed by all other solutions. In this thesis a *solution* is denoted by its health vector and we define the *solution variables* to be the faulty health variables of the solution. In the above example, the solution variable of d_2 is part of the solution variables of d_0.

The *diagnosis* of the example problem consists of three minimal solutions and can be written as

$$D = \neg h_9 \vee \neg h_{10} \vee \neg h_{11}$$

where $\neg h_9$ denotes a faulty (not healthy) component c_9. Each of these minimal solutions contains only one faulty component. In other words, the fault cardinality of each solution is $C = 1$.

With the explained concept of model-based diagnosis and the terminology as discussed in this chapter we are able to describe and analyze the SAFARI stochastic diagnosis algorithm. In the remaining chapters *performance models* are discussed of the SAFARI algorithm. This concept of a model should not be confused with a system model, which is discussed previously and is used as input to the diagnosis algorithm.

Chapter 3

SAFARI Algorithm

SAFARI is a greedy stochastic algorithm, which finds multiple cardinality-minimal diagnoses. The pseudo-code of the algorithm is given in Algorithm 1. In this pseudo-code, the following notations are used. The system description SD is a propositional theory describing the behavior of the system. COMPS is the set of assumable variables in SD, which, in practice, are the M health variables of the system. OBS is the set of observable variables in SD. Together they form the diagnosable system DS.

The input to the algorithm consists of the diagnosable system DS together with an observation α, an integer R and an integer N. α is an observation on the system, in other words, known values for the variables in OBS. R is a value between 1 and M which determines the maximum number of retries per run. N denotes the number of runs, where each run adds a new solution to the resulting set of possible solutions to the problem. C_r is the cardinality of a resulting solution of a single run.

At the beginning of each run a random, consistent, diagnosis is generated as a starting point for the stochastic hill climbing. Then, this diagnosis is improved upon in the sense that it is transformed to a more probable solution, i.e. a solution with less faulty variables. The IMPROVEDIAGNOSIS function, in fact, selects a faulty health variable and makes it healthy by flipping it. This can result in either an improved and consistent solution or an inconsistent solution. In the case of an inconsistent solution another health variable is selected and consistency is checked again. This can be done a maximum of R times before the current solution is viewed as a local optimum and is added to the set of solutions. The selection of variables to flip is done *without repetition*, so that multiple checks of the same solution are avoided. A $C = C_r$ solution is added to the set only if it is not subsumed by other solutions in the set and all solutions in the set that are subsumed by the added solution are removed so that the resulting set contains only minimal solutions with regards to the encountered solutions. Deterministic methods start with a all-healthy solution, which is inconsistent in most cases, and find the minimal solutions by changing variables to unhealthy. SAFARI, on the other hand, starts with an inconsistent but high fault cardinality solution and tries to reduce fault cardinality while keeping consistency.

For example, consider the algorithm applied on an observation of the 74180 system. The DS is composed of a weak model of the logical system SD as depicted in Figure 2.3 in the previous chapter, COMPS, which are the $M = 14$ health variables, and OBS, the twelve

```
1:  function HILLCLIMB(DS, α, R, N) returns a set
        inputs: DS = ⟨SD, COMPS, OBS⟩, diag. system
                α, term, observation
                R, integer, climb restart limit
                N, integer, number of runs
2:      n ← 0
3:      while n < N do
4:          ω ← RANDOMDIAGNOSIS(SD, α)
5:          r ← 0
6:          while r < R do
7:              ω′ ← IMPROVEDIAGNOSIS(ω, ρ)
8:              if SD ∧ α ∧ ω′ ⊭ ⊥ then
9:                  ω ← ω′
10:                 r ← 0
11:             else
12:                 r ← r + 1
13:             end if
14:         end while
15:         unless ISSUBSUMED(S, ω) then
16:             ADDTOSET(S, ω)
17:             REMOVESUBSUMED(S, ω)
18:         end unless
19:         n ← n + 1
20:     end while
21:     return S
22: end function
```

Algorithm 1: SAFARI: A greedy stochastic hill climbing algorithm for approximating the set of minimal diagnoses.

observable variables (I_{0-7},*EI*,*OI*,*SE*,*SO*). Let an observation o_1 be

$$o_1 = \{I = 11111111, EI = 0, OI = 0, SE = 0, SO = 1\}$$

and let the input to SAFARI be $\alpha = o_1$, $N = 3$ and $R = 2$, so there will be three runs and two possible retries. Since a weak model of the system is used, RANDOMDIAGNOSIS may return an all faulty solution, which is always consistent. We know, by exhaustively diagnosing the system, that the true minimal diagnoses of this observation are

$$d_1 : h = (11111111011111)$$

$$d_2 : h = (11111111101111)$$

$$d_3 : h = (11111111110111)$$

Deterministic methods would usually try each health variable and would quickly come to the correct minimal diagnosis. SAFARI, in this case, starts with the all-faulty but consistent solution. A partial trace of the algorithm is shown in Table 3.1. It starts with setting n and r to zero and using the trivial all-faulty solution of $C = 14$. At the second step, this solution is transformed to a consistent $C = 13$ solution by flipping one random health variable to

healthy. At the fourth step, health variable h_9 is flipped to healthy, which makes it unable to reach the existing minimal solution d_1 in the rest of this run. However, solutions d_2 and d_3 can still be reached, so the current solution is still consistent. Further along, at step seven, h_{11} is flipped to healthy, eliminating the chance of reaching d_3 in this run, letting d_2 be the only possible true minimal solution outcome of this run. At step eleven h_{10} is flipped to healthy as well, resulting in a $C = 4$ solution which is inconsistent. At this point r is increased and another variable is flipped, which results in a consistent $C = 4$ solution. At $C = 1$ h_{10} is indeed the only remaining faulty health variable and further flipping of variables result in the inconsistent all-healthy solution. Thus, d_2 is added to the set of resulting solutions and a new run is started by increasing n and by reinitializing the current solution to the trivial all-faulty solution.

n	r	health vector	C	consistent
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	14	yes
0	0	0 0 0 1 0 0 0 0 0 0 0 0 0 0	13	yes
0	0	0 1 0 1 0 0 0 0 0 0 0 0 0 0	12	yes
0	0	0 1 0 1 0 0 0 0 1 0 0 0 0 0	11	yes
0	0	0 1 0 1 0 0 1 0 1 0 0 0 0 0	10	yes
0	0	0 1 0 1 0 1 1 0 1 0 0 0 0 0	9	yes
0	0	0 1 0 1 0 1 1 0 1 0 1 0 0 0	8	yes
0	0	0 1 0 1 0 1 1 0 1 0 1 1 0 0	7	yes
0	0	0 1 0 1 0 1 1 0 1 0 1 1 0 1	6	yes
0	0	1 1 0 1 0 1 1 0 1 0 1 1 0 1	5	yes
0	0	1 1 0 1 0 1 1 0 1 1 1 1 0 1	4	no
0	1	1 1 0 1 0 1 1 1 1 0 1 1 0 1	4	yes
0	0	1 1 0 1 0 1 1 1 1 0 1 1 1 1	3	yes
0	0	1 1 1 1 0 1 1 1 1 0 1 1 1 1	2	yes
0	0	1 1 1 1 1 1 1 1 1 0 1 1 1 1	1	yes
0	0	1 1 1 1 1 1 1 1 1 1 1 1 1 1	0	no $\rightarrow d_2$
1	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	14	yes
1	0	0 0 0 0 0 0 0 0 0 1 0 0 0 0	13	yes
		⋮		

Table 3.1: Trace of SAFARI with $N = 3$ and $R = 2$

The SAFARI algorithm returns a set of solutions which is an approximation of the actual set of minimal diagnoses in different aspects. First, because of the stochastic behavior of the algorithm there is always a chance that some path towards a certain existing minimal solution is not traversed. For example, if one minimal solution exists of which a health variable h_i is faulty, but by chance each run sets h_i to healthy early in the run without causing inconsistency, then this minimal solution is never found. Because no inconsistency has occurred, the number of retries R is ineffective. This problem can be overcome by increasing N to increase the probability of climbing towards that solution. Secondly, the returned set is an approximation because the number of retries is limited to R. This causes uncertainty in reaching the actual local minimum. For example, if at a certain point only five

faulty health variables remain (h_1 to h_5) and $R = 4$, then only four of the possible five health variables are tried to be changed to healthy. If none of these tries resulted in a consistent solution, the algorithm adds the $C_r = 5$ solution to the resulting set. However, it could be that the solution with the fifth variable changed to healthy resulted in a consistent $C_r = 4$ solution. Increasing the value of R to try all variables would solve this issue, however, this will make the algorithm slower.

A stochastic algorithm like SAFARI leads to incompleteness and non-optimality, which is essential in the area of diagnostics. But ensuring completeness and optimality requires a deterministic method which is complex in terms of time and often in terms of space as well. Most diagnostic problems have a solutions set with low *MC*, however, this is not always the case, especially when systems have limited observability. Time and space can be an important requirement for diagnostic applications, which can relate directly to money, e.g., when the cost of a failing machine rises with each minute it is off-line for diagnosis, or to safety, e.g., when an aircraft needs to diagnose and repair some system failure while flying. For higher MC diagnoses stochastic algorithms like SAFARI are the only feasible solution.

When discussing the performance model for the SAFARI algorithm some basic assumptions of the diagnosis problem are made:

1. The system models are assumed to be *boolean*, since this simplifies the flipping of health variables, i.e. either faulty or healthy. Also, any system model created in the integer domain can be rewritten into a boolean model.

2. Secondly, a system is assumed to be *weakly* modeled. Because the plain SAFARI algorithm as discussed is not capable of solving every non-weak model an extension of the algorithm is needed, which makes modeling the algorithm even harder. It must be noted that every system can be modeled weakly, which means that this assumption only excludes certain system models, rather than the systems themselves. A property of weak system models is that given a consistent solution s, then the solution, created by flipping any of the healthy variables to faulty, is also consistent.

3. Finally, it is assumed that SAFARI initializes with the *trivial solution* of all components being unhealthy. This removes any bias to the algorithm, since the method of determining the starting health vector could favor some health variables being healthy. Also, any initialization other than the trivial initialization is only required for non-weak models, which could yield an inconsistency when all components are set to unhealthy.

3.1 SAFARI Performance

As can be seen in Algorithm 1, the result of the SAFARI algorithm is a set which contains minimal solutions to the diagnostic problem. These solutions are minimal with respect to all solutions that are found during the N runs. Due to the fact that SAFARI is stochastic and thus incomplete and non-optimal, the solutions could be non-minimal in reality, since more solutions may exist which subsume a subset of the resulting set of solutions.

Since we are interested in the resulting cardinalities of solutions, as explained in the previous chapter, we can capture the distribution of cardinalities of solutions in the resulting set. Let f_s (s for solution space) be the pdf of cardinalities of the minimal solution space of the diagnostic problem as returned by SAFARI. For example, if for a given diagnostic problem, SAFARI returns a set which consists of the following six minimal solutions:

	h	C
d_1	0111111	$C = 1$
d_2	1011111	$C = 1$
d_3	1100111	$C = 2$
d_4	1101011	$C = 2$
d_5	1110011	$C = 2$
d_6	1101100	$C = 3$

then the resulting f_s contains a probability of $\frac{1}{3}$ for a $C = 1$ solution, $\frac{1}{2}$ for a $C = 2$ solution and $\frac{1}{6}$ for a $C = 3$ solution. This is illustrated in Figure 3.1, where f_s shows the per-

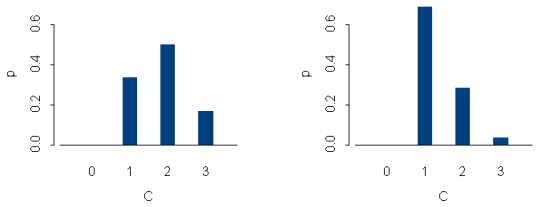

Figure 3.1: left: f_s and right: f_a of a system of size $M = 7$.

formance of SAFARI in terms of optimality and completeness when compared to the true solution space, another distribution, f_a (a for algorithm), shows more clearly the behavior of SAFARI. f_s is based on unique minimal solutions returned by SAFARI. Let f_a be the distribution of cardinalities C_r of solutions actually returned by each SAFARI run. Different runs can result in the same solution. f_a captures the frequency per resulting cardinality. As will be explained in the next section, an $M > 0$ will result in a shift of probability mass towards lower C. An example of ten consecutive SAFARI runs, for the diagnostic problem discussed above, is shown in the following table.

run	1	2	3	4	5	6	7	8	9	10
solution	d_1	d_2	d_3	d_2	d_2	d_6	d_4	d_1	d_3	d_2
cardinality	1	1	2	1	1	3	2	1	2	1

Because of the high probability of returning solutions of cardinality $C = 1$, solutions d_1 and d_2 are returned more often than the rest, although the number of $C = 2$ solutions is greater than the number of $C = 1$ solutions.

Figure 3.2 illustrates the transformation of the solution space by the SAFARI algorithm. It shows that f_s and f_a are two views of the same SAFARI output. One view is the unique

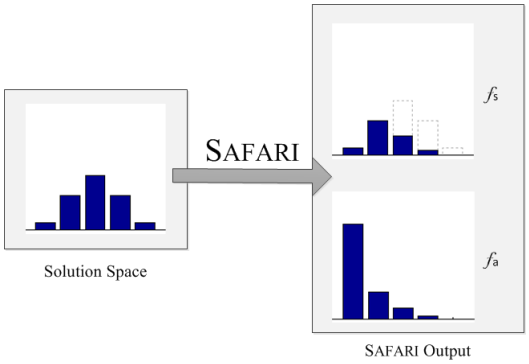

Figure 3.2: SAFARI performance.

(minimal) solutions returned by the algorithm (f_s) after N runs. The dashed outline shows that the returned solution space is incomplete at higher cardinalities. The reason is shown in the f_a view of the SAFARI output, which shows the relative frequencies of returned cardinalities. The lower cardinality solutions are more likely to be returned, which accounts for the success of SAFARI. If a stochastic algorithm like SAFARI would run for an infinite amount of times ($N \to \infty$), f_s would be equal to the true solution space, i.e., every solution would be visited at least once, even the solutions that are least probable (the higher cardinality solutions, in the case of SAFARI). Since f_a is independent of the number of runs N, we will define the *performance* of SAFARI in terms of the probability density of the solution cardinalities, and in specific, the low cardinalities, since these are more relevant. This performance could differ per input (system model and observations). For some problems SAFARI will return solutions of lowest cardinality with high probability, while for other problems the number of higher cardinality solutions is so great, that they decrease the probability of finding the *MC* solutions, and thus decrease the performance of SAFARI.

3.2 Current Model A_0

Currently, a basic model of the SAFARI algorithm exists that models the pdf of the solution cardinalities as returned by SAFARI. This model, A_0, is based on a SAFARI run which leads to a *target solution* of a given cardinality C_t. Ideally, a run would result in $C_r = C_t$.

Let k be the current fault cardinality, starting with $k = M$. A successful variable flip from unhealthy to healthy results in a transition to $k-1$, reducing the fault cardinality of the current, intermediate, solution. Let $p(k)$ be the probability that a variable flip on a solution of $C = k$ results in a consistent improved solution of $C = k-1$. Then $q(k) = 1 - p(k)$ signifies the probability that a variable flip fails to result in a consistent solution.

The solution at k has $M-k$ healthy and k faulty health variables. In this model, a successful variable flip is defined to be a flip of a faulty health variable which is not part of the faulty health variables of the target solution. Since there are k variables which are faulty

of which C_t variables are part of the target solution, the probability $p(k)$ becomes

$$p(k) = \frac{k - C_t}{k} = 1 - \frac{C_t}{k}$$

This process can be modeled in terms of a Markov chain, which is depicted in Figure 3.3, where k signifies the state of the algorithm. A transition to state *fail* occurs when a variable flip results in an inconsistent solution.

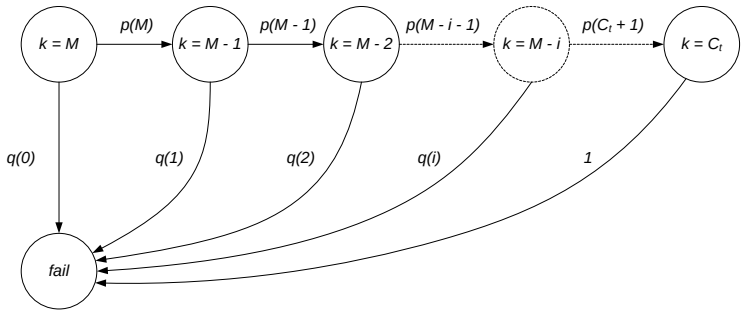

Figure 3.3: Model A_0 of a SAFARI run with $R = 1$ and a single diagnosis of cardinality c

Retries are modeled by adding an axis to the linear Markov chain, using k and r as state. An unsuccessful variable flip from state (k, r) results in a transition to state $(k, r+1)$, whereas a successful variable flip results in a transition to state $(k-1, 0)$. This can also be modeled by generalizing the calculation of probability p by stating

$$p(k) = 1 - \frac{\binom{C_t}{R}}{\binom{k}{R}}$$

Figure 3.4 shows the effect of R on the performance of SAFARI. A larger R results in a drastic increase of the probability mass in f_a near the target cardinality C_t. This is explained by the fact that a retry increases the probability of reaching the C_t.

Note that this model assumes a selection *with repetition* of a variable to flip, which is in contradiction with the actual SAFARI algorithm. SAFARI would show an even greater increase of probability mass near the lower cardinalities, because the probability of flipping a variable twice is zero.

If there is more than one solution of the same cardinality C_t, the chance of returning a C_t solution increases. Let the number of solutions of $C = C_t$ be N, then this *solution multiplicity* is modeled by taking the minimum of the resulting cardinalities of N independent runs. In terms of model A_0, the resulting C_r would be the minimum C_r of N independent runs:

$$C_r = \min_{i=1}^{N} C_r(i)$$

Figure 3.5 shows that the existence of multiple solutions of $C = C_t$ cause a increase of the probability mass towards C_t. This increase is not as profound as with a larger R, but

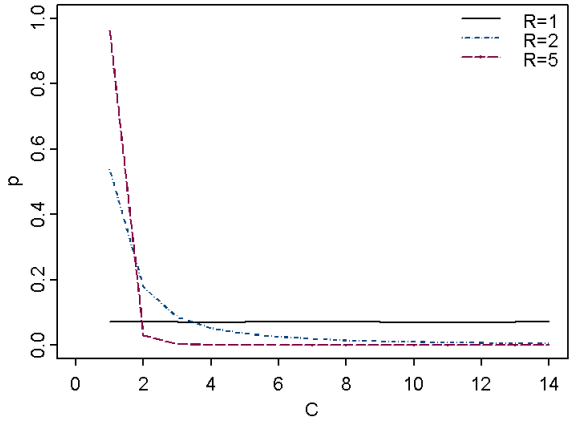

Figure 3.4: f_a of Model A_0 for different R on a system with $M = 14$ and $C_t = 1$.

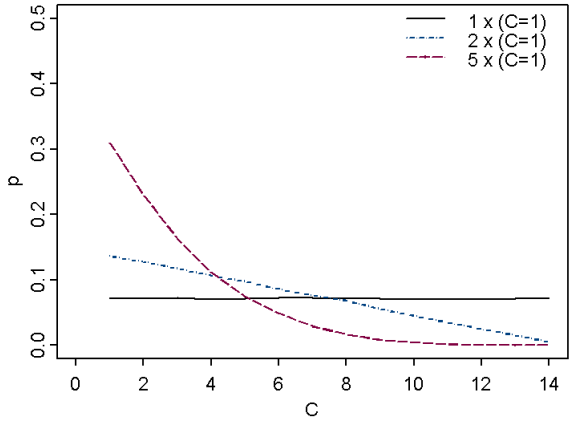

Figure 3.5: f_a of Model A_0 for different solution multiplicities on a system with $M = 14$ and $C_t = 1$ and $R = 1$.

it is significant. It is explained by the fact that if there are more solutions of $C = C_t$, the probability of reaching any of them increases. This, of course, is not an algorithm parameter, like R, but this shows that the effect of multiple solutions on the performance of SAFARI is large.

3.3 Limitations of Model A_0

Figure 3.6 shows the effect of higher cardinality solutions on the distribution of solution cardinalities of SAFARI. In the figure $MC = 1$, which is also used as the target cardinality of model A_0. There is only a single *MC* solution of $C = 1$. The resulting distribution of model A_0 is the same as the solution distribution of SAFARI when run on random observations with a single $C = 1$ solution and $R = 1$. When no other solutions exist, the behavior of SAFARI is the same as modeled by model A_0. However, when SAFARI is run on random observations with again a single $C = 1$ solution, but with solutions of $C = 2$ as well, the resulting solution distribution is very different. Where one single *MC* solution results in an equal probability of returning any cardinality, including higher cardinality solutions results in an increase of probability mass near that cardinality. Also, including higher cardinality solutions decreases the probability of inconsistency at higher k.

Model A_0 is accurate if the true diagnosis of the problem contains only *MC* solutions. It is clear that model A_0 is unable to capture the influences of higher cardinality solutions. To base a model on *MC* and its multiplicity results in an incomplete model. We need more information to capture the behavior of SAFARI.

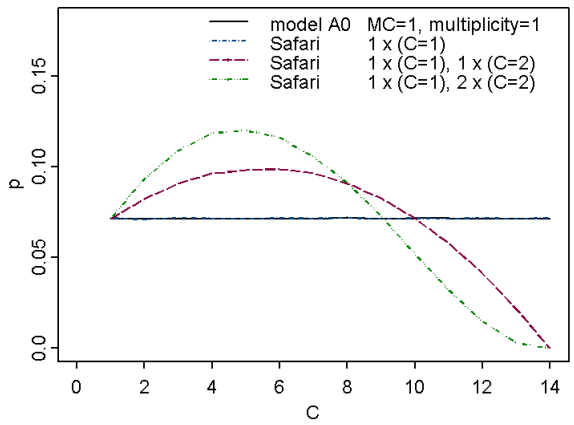

Figure 3.6: pdf of Model A_0 compared to empirical results of SAFARI on random observations both without and with higher cardinality solutions on the 74180 circuit.

Since the *MC* of the solutions and its multiplicity are insufficient information to base a model upon, it becomes clear that more information of existing solutions of the problem is required. This notion is supported by Figure 3.7, which shows the distribution of solution cardinalities of three actual observations of the example system with $MC = 1$ and *MC* solution multiplicity of 3. One of these observations results in a diagnosis of only *MC* solutions. This is accurately modeled by model A_0. However the remaining two observations result in very different behavior of SAFARI, even though the *MC* and its multiplicity remain the same, the difference being the presence of higher cardinality solutions.

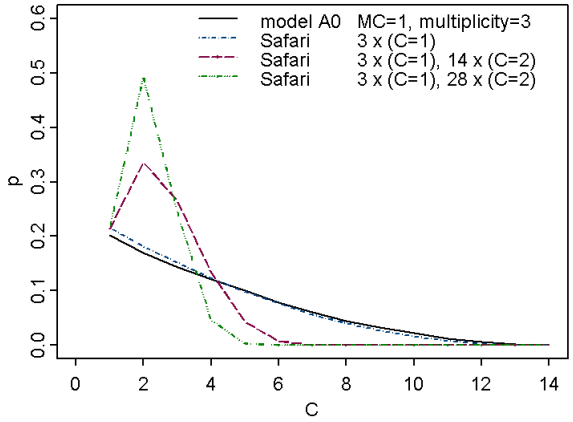

Figure 3.7: pdf of Model A_0 compared to empirical results of SAFARI on three existing observations without and with higher cardinality solutions

This leads to the following theorem on SAFARI.

Theorem 1. *The behavior of the* SAFARI *algorithm depends on all minimal solutions present in the system, not only the MC solution(s).*

Proof. With $R = 1$, SAFARI returns when the consistency check fails. With larger values for R, SAFARI tries to flip another variable when the consistency check fails. This consistency check is equivalent to checking if one or more minimal solutions of the problem is still reachable. In other words, at least one solution should consist of variables which are still unflipped. Since all minimal solutions are by definition unsubsumed and SAFARI flips variables to unhealthy until an inconsistency is reached, each of these solutions has a probability of being the last remaining solution of one SAFARI run. The fact that each of the existing minimal solutions can be the last remaining solution implies that the distribution of probabilities of solution cardinalities is influenced by the complete set of minimal solutions. □

Table 3.2 shows an example of a SAFARI run as a sequence of health vectors. In this example $M = 6$, $R = 1$, and only one $C = 1$ solution exists, which is denoted by an underlined health variable. Since only one solution exists, an inconsistency arises whenever one of its variables is flipped. At the start, $k = M = 6$ variables can be flipped, of which one variable belongs to the last remaining solution. There is a $\frac{1}{6}$ chance of reaching an inconsistency at the beginning of the run.

Given that one variable was successfully flipped, $k = M - 1 = 5$ variables can be flipped at this point, which makes the probability of reaching an inconsistency $\frac{1}{5}$ at this point. The table shows the probabilities of flipping the solution variable at each k, given that each k is reached (cf. Model A_0).

health vector	p_{fail}
0 0 0 0 0 0	0.17
0 0 0 0 0 1	0.20
0 0 1 0 0 1	0.25
0 0 1 0 1 1	0.33
0 0 1 1 1 1	0.50
0 1 1 1 1 1	1.00

Table 3.2: Trace of SAFARI with one existing solution and the probability of reaching an inconsistency

Table 3.3 shows two runs of SAFARI on the same system of size $M = 6$, but for another observation which has an extra solution of fault cardinality $C = 2$ next to the $C = 1$ solution. There is already a difference compared to the previous example at the start of the run. In this example there is no chance of flipping a first variable which would make all solutions impossible to reach. If the variable of the $C = 1$ solution is flipped, the $C = 2$ solution would still be possible, and vice versa. In the first of the two example runs in Table 3.3 the $C = 2$ solution is made infeasible at $k = 4$. From this point on the $C = 1$ solution is the only remaining solution and the probabilities of reaching inconsistency are the same as in Table 3.2. The second example run shows that the probabilities are very different when the variable of the single fault solution is flipped, although the chance of that happening is smaller.

health vector	p_{fail}	health vector	p_{fail}
0 0 0 0 0 0	0.00	0 0 0 0 0 0	0.00
0 0 0 0 0 1	0.00	0 0 0 0 0 1	0.00
0 0 1 0 0 1	0.25	0 0 0 0 1 1	0.00
0 0 1 0 1 1	0.33	1 0 0 0 1 1	0.67
0 0 1 1 1 1	0.50	1 0 0 1 1 1	1.00
0 1 1 1 1 1	1.00		

Table 3.3: Influence of multiple solutions on the probability of reaching an inconsistency

This explains how model A_0 is insufficient to model SAFARI behavior, since this model only works when the set of existing solutions consists only of solutions of the same cardinality. Higher cardinality solutions, and, in fact, all minimal solutions of the problem influence the probability of returning a certain cardinality.

Chapter 4

Performance Modeling

In this section the performance of SAFARI is analyzed in more detail and three new performance models are studied. The names of the models are based on the level of abstraction. Model A_0, the existing model, is furthest away from reality as every way of checking for inconsistency is abstracted. Model A_3 is closest to the actual algorithm and the abstraction from reality is kept at a minimum. The models are compared regarding their accuracy of modeling actual SAFARI behavior, and their complexity. All models are based on a particular way of characterizing SAFARI performance, which is presented in the next section.

4.1 Characterizing SAFARI Performance

Considering only boolean systems, the number of different observations of any system with O observable variables is 2^O. The running example has ten inputs and two outputs which can be observed, which results in $2^{12} = 4096$ possible different observations of which many are observations corresponding to faulty system behavior.

Many of these observations can lead to the same diagnosis. For example, in the running example there are 1024 different observations which lead to the same nominal diagnosis, where no component is faulty.

4.1.1 Characteristic S

The previous chapter shows that the pdf of cardinalities returned by SAFARI is dependent on all existing minimal solutions of the problem. We have also seen that the SAFARI algorithm does not exploit the system model in any way for optimizing the way it searches for solutions. The system model is only used as an oracle for determining the consistency of the solutions, which are obtained by flipping variables randomly. The same holds for the observation, which is simply a constraint on the problem. Because the algorithm focuses on flipping variables randomly, not based on the system model and observation, the behavior of SAFARI while reaching one solution of cardinality $C = 3$ is similar to its behavior when reaching another $C = 3$ solution. It seems that the behavior of SAFARI is influenced more by the cardinalities and the number of solutions existing in the solution space. This leads us to the following hypothesis:

Hypothesis 1. *The actual existing minimal solutions in the health vector of a diagnosis problem is less influential on the performance of the* SAFARI *algorithm than the* fault cardinality *is. Consequently, let S_i be the number of minimal solutions of fault cardinality i, then the vector $S = \{S_1, S_2, \ldots, S_M\}$ is assumed to be* characteristic *for the performance of* SAFARI.

Observations can be grouped further by making use of Theorem 1 and the additional Hypothesis 1. The assumption is made that the actual diagnosis does not determine the behavior of SAFARI, but the composition of minimal solutions in terms of cardinalities. As indicated in Table 3.3, it is the fault cardinalities of the remaining solutions that determine the probability of reaching an inconsistency.

To illustrate S, let the true diagnosis for an observation o on a system of $M = 14$ components consist of one $C = 1$ solution, four $C = 2$ solutions and twenty-one $C = 3$ solutions. There are no solutions of higher cardinalities. Then for this observation S equals

$$S(o) = \{1,4,21,0,0,0,0,0,0,0,0,0,0,0\}$$

For simplicity, we write

$$S(o) = \{1,4,21\}$$

The 4096 different observations on the example system were used as input to exhaustively[1] obtain the true set of minimal solutions for each observation. It turns out that the example system contains only 13 different values of S. Table 4.1 shows these values together with the number of observations leading to each S. That would mean that the performance of SAFARI is limited to only thirteen variations for this system.

MC	*S*					count
0	{0,	0,	0,	0,	0}	1024
1	{1,	4,	21,	0,	0}	128
	{2,	7,	21,	0,	0}	256
	{2,	10,	0,	0,	0}	256
	{2,	21,	0,	0,	0}	256
	{3,	0,	0,	0,	0}	512
	{3,	14,	0,	0,	0}	256
	{3,	28,	0,	0,	0}	256
	{4,	21,	0,	0,	0}	256
	{7,	8,	0,	0,	0}	256
	{7,	9,	0,	0,	0}	256
2	{0,	9,	0,	0,	0}	256
	{0,	11,	21,	0,	0}	128

Table 4.1: The thirteen unique values for S of the 74180 system

The resulting distributions f_a of SAFARI of all observations of the *same* S were compared and demonstrated surprisingly little variation, supporting the hypothesis that the cardinalities of all existing solutions are the most important influence on the behavior of the

[1] Exhaustive search was done by trying all health vectors, resulting in all possible solutions of the problem, of which the minimal set was taken.

SAFARI algorithm, rather than the exact positioning of the solutions in the health vector. For example, in Figure 4.1 the resulting boxplots of all observations of $S = \{1,4,21\}$ and all observations of $S = \{2,10,0\}$, when used as input for SAFARI with $R = 5$, are shown. It shows that the variation of f_a over resulting cardinalities is very small. This is seen in experiments on the observations of the remaining values of S as well. This shows that S is indeed characteristic for the behavior of SAFARI.

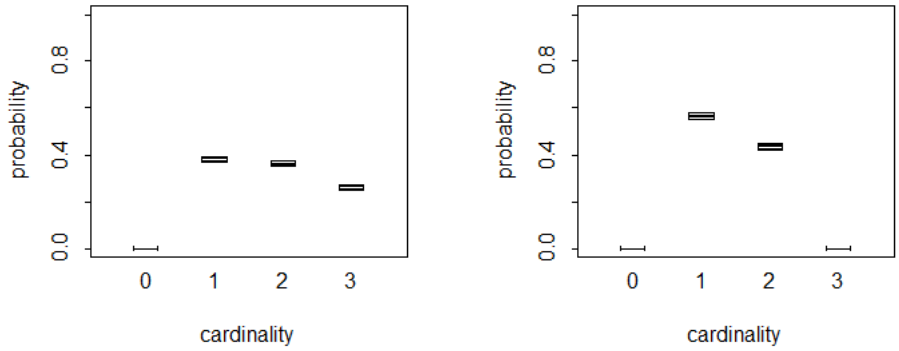

Figure 4.1: Boxplots of SAFARI output ($R = 5$) of all observations with $S = \{1,4,21\}$ and $S = \{2,10\}$

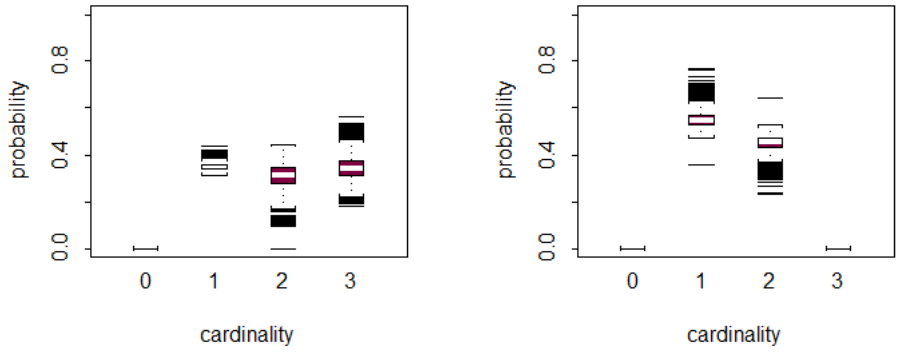

Figure 4.2: Boxplots of SAFARI output ($R = 5$) of random observations with $S = \{1,4,21\}$ and $S = \{2,10\}$

Figure 4.1 shows only results of the observations of a single system, the example system. It is likely that for actual systems there is a certain pattern of how solutions of groups of observations are distributed in the health vector. This can result in a optimistic view of S as a characterization of resulting cardinality distribution. To generalize the experiments which show that S is characteristic for the behavior of SAFARI, a solution generator is used to create random solutions of a given S and a health vector of size M. These random solutions can be seen as the diagnoses of a random system and random observations or as a random observation leading to solutions of the cardinalities given in S. The resulting boxplots when given the same S as used in Figure 4.1 are shown in Figure 4.2. Some 50% of the observations reside within a limited range of resulting probabilities, supporting the hypothesis that S is the main characteristic with respect to the behavior of the algorithm. However, from the plots it is also clear that there are outliers, exceptions to the general case, some improbable distribution of solutions in the health vector that results in a different distribution of cardinalities.

4.1.2 Overlapping Solutions

One theory that could explain the different SAFARI behavior when returning solutions of the same S is that the solutions of the resulting diagnosis can overlap each other in different degrees. For example, if there are two solutions of cardinality $C = 3$ in a health vector of size $M = 6$, their variables could be distributed disjunctively:

C=3	0	0	0	1	1	1
C=3	1	1	1	0	0	0

Another way the variables could be distributed is by overlapping the solutions:

C=3	0	0	0	1	1	1
C=3	1	0	0	0	1	1

In the latter case two variables are shared among two solutions, increasing the probability that one variable flip of SAFARI will make the two solutions impossible to reach.

To have an indication of how solutions are overlapped we consider the variance σ^2 of the number of solutions that share a variable. For example, consider the sets of minimal solutions given in Table 4.2. Both sets contain five solutions, of which one is of cardinality one, three solutions are of cardinality two and the last solution is of fault cardinality three. In the first set the solutions are spread out as much as possible in the nine available health variables, therefore the number of overlaps of solutions is kept to a minimum, although variable h_7 is still shared among two solutions. The second set contains solutions which are overlapped as much as possible. Variables h_2 and h_4 are shared by three solutions, while variables h_7, h_8 and h_9 are not used by any solution.

The calculation of the mean μ of the number of sharing solution per variable is equal to the average fault cardinality of each solution. Because the two solution sets of the same $S = \{1,3,1\}$ are compared, μ is the same for the two examples. However, because the overlap of variables differs greatly, the assumption is that the variance of the number of

$C=1$	0	1	1	1	1	1	1	1	1
$C=2$	1	0	0	1	1	1	1	1	1
$C=2$	1	1	1	0	0	1	1	1	1
$C=2$	1	1	1	1	1	0	0	1	1
$C=3$	1	1	1	1	1	1	0	0	0
$\lvert shared \rvert$	1	1	1	1	1	1	2	1	1
μ					1.111				
σ^2					0.099				

$C=1$	0	1	1	1	1	1	1	1	1
$C=2$	1	0	0	1	1	1	1	1	1
$C=2$	1	0	1	0	1	1	1	1	1
$C=2$	1	1	1	0	0	1	1	1	1
$C=3$	1	0	1	0	1	0	1	1	1
$\lvert shared \rvert$	1	3	1	3	1	1	0	0	0
μ					1.111				
σ^2					1.210				

Table 4.2: Variance of the number of sharing solutions per variable

sharing solutions per variable will be a measure of how extreme the overlapping of solutions is.

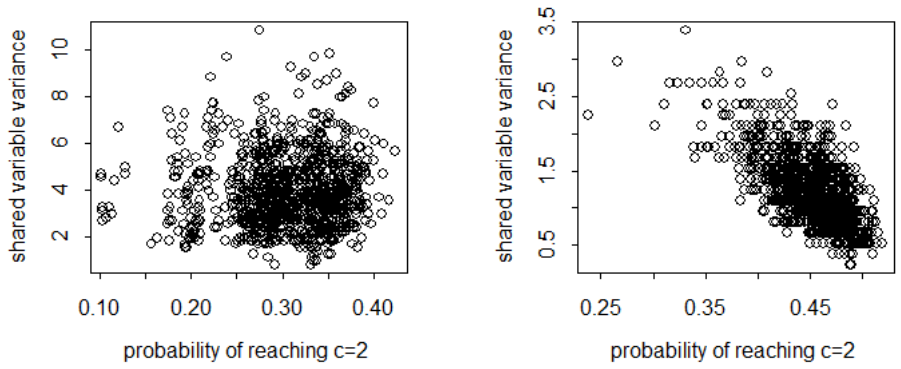

Figure 4.3: Scatterplot of the probability of reaching $c=2$ solutions versus the variance of the number of sharing solutions of random observations with $S=\{1,4,21\}$ and $S=\{2,10\}$

If the assumption that the degree of overlapping solutions explains the variation in the boxplots of Figure 4.2 is correct, then the calculated variance of each of the solutions used in the experiment should correlate with the probabilities of reaching a $C=2$ solution of the two tested S, which show a large variation. The scatterplot of the probabilities versus

the variance of the number of sharing solutions as seen in Figure 4.3 in fact show the very opposite for the $S = \{1,4,21\}$, but show a correlation for the $S = \{2,10\}$ case.

The experiments show that S is characteristic for the behavior of the SAFARI algorithm, but that it abstracts the way the variables of solutions is positioned in the health vector. This positioning of solutions should explain the outliers of random solutions of the same S, but the variance calculation that is used to capture the overlapping of solutions is unable to explain the variation of behavior of SAFARI on solutions with the same chararcteristic S. This would imply that either the variance calculation is inaccurate or too general for all cardinalities, or that there is another factor than overlapping solutions.

4.1.3 Summary

The number of MC solutions of a diagnosis problem, used in model A_0, is unable to explain the behavior of SAFARI, when in addition to the MC solutions, the total set of existing minimal solutions also contains solutions of higher cardinality. S contains the number of existing minimal solutions for each cardinality C, and is shown to be the main characteristic with respect to the behavior of SAFARI. Figure 4.4 illustrates this. The observations are grouped by different performance characterizations. Each S results in a unique SAFARI performance pdf f_a. It is also shown that S does not explain the behavior of SAFARI exactly, since, for example, the degree in which solutions overlap each other in the health vector is a factor which is abstracted by S. Although using S as a basis for developing models for SAFARI behavior could increase model complexity, the accuracy can be greatly enhanced.

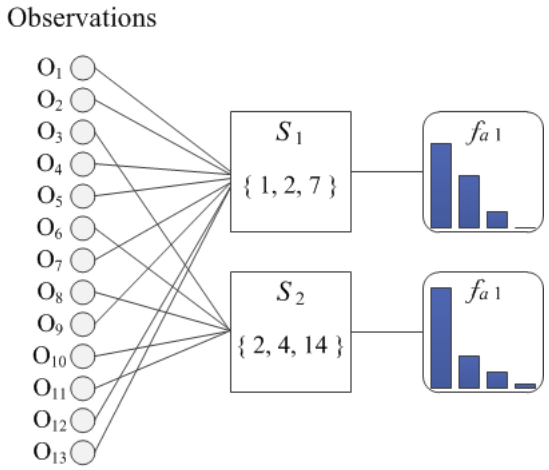

Figure 4.4: Characterizing SAFARI performance

4.2 Model A_3

In the previous section we have introduced S and we have confirmed that it is sufficiently characteristic for the performance of the SAFARI algorithm. In this section we introduce model A_3, which is based on S and simulates the algorithm as closely as possible, abstracting the information on the positioning of solutions.

In this model S is used as state in the Markov Chain. Its starting state is a state with the given S and $k = M$. The chain is then constructed by adding a traversal from each $S(k)$ to every possible $S(k-1)$ that can be a result of a variable flip. Figure 4.5 shows the Markov chain of this model applied to a simple example of system of size $M = 6$ and an observation that can be characterized by $S = \{0,1,2\}$.

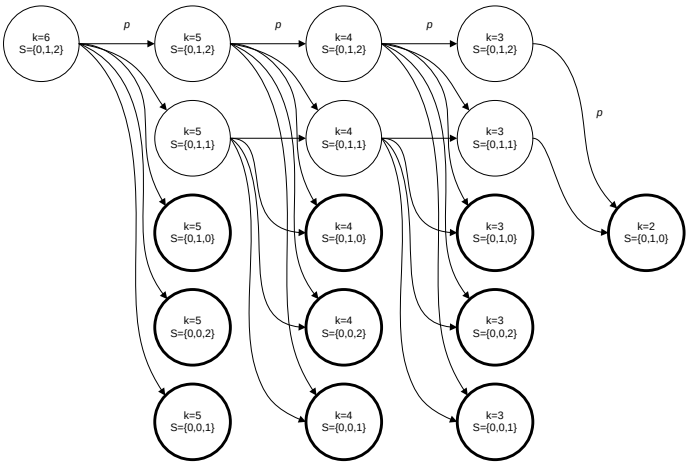

Figure 4.5: Model A_3 of a SAFARI run on a system of size $M = 6$ and an observation of $S = \{0,1,2\}$ (terminating states in bold)

The value of R is left out of this model. Since SAFARI picks variables at random *without repetition*, an R greater or equal to k (the number of unflipped variables), ensures that all variables are tried. In reality, this could mean a worse time performance of SAFARI, but since completeness of the algorithm is valued more than speed, R is assumed to be large enough to guarantee minimal solutions. Furthermore, the effect of an insufficient R is greater at lower k near the end of a run, when the probability of reaching an inconsistency with a variable flip is larger, making the performance drop limited to a number of retries equal to a small k.

In this model we assume that the solutions that are present in $S(k)$ are randomly positioned in the health vector. The probability of each traversal from state $S(k)$ to state $S(k-1)$ depends on the number of ways the new state can be reached, the probability that certain solutions are made infeasible, and the probability that the remaining solutions have no variables flipped. For example, suppose that the actual solutions to a problem of a system of $M = 6$ are

	h	C
s_1	001111	$C = 2$
s_2	010011	$C = 3$
s_3	101100	$C = 3$

then the characteristic S for this problem is $S = \{0,1,2\}$. The starting state, therefore, is $S(k = 6) = \{0,1,2\}$. Given this system and its starting state of Model A_3, consider the traversal to state $S(k = 5) = \{0,1,1\}$. This is a decrease of S_3 by one, i.e. one of the $C = 3$ solutions, s_2 or s_3 is made infeasible by flipping one of its variables. The number of solutions per cardinality that are made infeasible is given by

$$S(k) - S(k-1) = \{0,0,1\}$$

In this case, either s_2 or s_3 is made infeasible, making the number of combinations equal to 2. In general, the number of ways one can choose the solutions that are made infeasible out of the complete set of solutions is given by

$$\prod_i \binom{S_i(k)}{S_i(k-1)}$$

where S_i is the number of solutions in S for cardinality $C = i$.

In order to make a $C = 3$ solution infeasible, either a variable of s_2 is flipped, or a variable belonging to s_3. The probability of making any $C = 3$ solution infeasible given $k = 6$ is equal to $\frac{3}{6}$ since only one of its variables needs to be flipped in order for the solution to be impossible to be reached. In the case of the example, either h_1, h_3 or h_4 needs to be flipped in order to make solution s_2 infeasible.

The probability that the remaining solutions (s_1 and the remaining $C = 3$ solution) have none of their variables flipped is equal to

$$\left(1 - \frac{2}{6}\right) \cdot \left(1 - \frac{3}{6}\right) = \frac{1}{3}$$

Combining it all results in a probability $p = \frac{1}{3}$ that a variable flip results in reaching a state where one of the two $C = 3$ solutions is made infeasible, while the other $C = 3$ solution and the $C = 2$ solution remain feasible. In general this probability is equal to

$$p = \prod_i \left(\binom{S_i(k)}{S_i(k-1)} \cdot \left(\frac{i}{k}\right)^{S_i(k)-S_i(k-1)} \cdot \left(1 - \frac{i}{k}\right)^{S_i(k-1)} \right)$$

In total, there are six possible transitions from state $S(k = 6) = \{0,1,2\}$ as shown in Table 4.3. State $S(k = 5) = \{0,0,0\}$ is an inconsistency state, where none of the cardinalities have solutions. Since the assumption is made that R is large enough, an inconsistency always results in trying to flip another variable until all variables are tried. In practice, reaching an inconsistency results in a transition to the same state with the same probabilities for reaching another state. The probabilities of traversing to the consistent states can therefore be normalized to have the same effect. In the example, the probabilities of the transitions to any state other than $S(k = 5) = \{0,0,0\}$ should be multiplied by $\frac{12}{11}$.

k	S	p	C_r
5	$\{0,1,2\}$	$\frac{1}{6}$	N/A
5	$\{0,1,1\}$	$\frac{1}{3}$	N/A
5	$\{0,1,0\}$	$\frac{1}{6}$	2
5	$\{0,0,2\}$	$\frac{1}{12}$	3
5	$\{0,0,1\}$	$\frac{1}{6}$	3
5	$\{0,0,0\}$	$\frac{1}{12}$	N/A

Table 4.3: Possible transitions from state $S(k=6) = \{0,1,2\}$

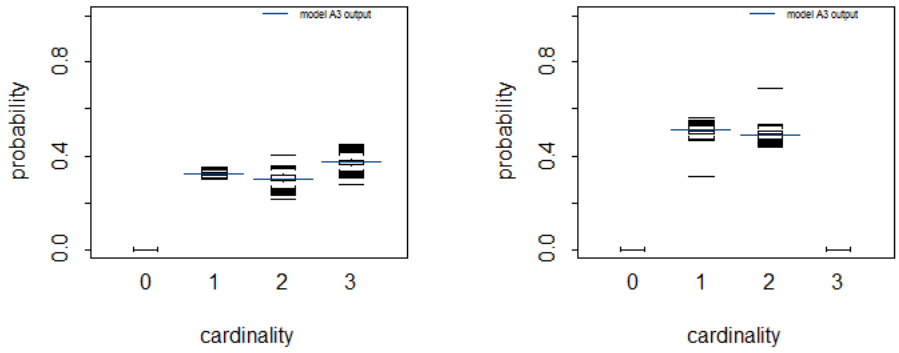

Figure 4.6: Accuracy of Model A_3 when compared to SAFARI when run with random generated solutions.

The end state of the chain is any state that contains an S which contains only one cardinality. In Figure 4.5 these states are emphasized. Table 4.3 shows the resulting cardinality C_r whenever this is clear.

Model A_3 serves as a reference for other models, since this is the best achievable model without including exact solution positioning in the health vector. We performed an experiment where SAFARI was run 10,000 times on a randomly generated solution set of a certain S to obtain a single pdf f_a of its resulting C_r. This was done 10,000 times to create f_a for many randomly created sets of the same S. The total experiment was performed for all different S of the example system, among others. All results demonstrated the same accuracy of this model. The result for two different S are shown in Figure 4.6, where the outcome of model A_3 lies exactly at the median of outcomes of SAFARI when run on random observations of the same S, which indicates that model A_3 is a very accurate model for the behavior of SAFARI on random solutions.

However, this model proves to be too complex to be used in practice. Depending on S, the number of transitions from each state can be very large. The number of transitions is

equal to

$$|S(k-1)| = \prod_i (S_i(k)+1)$$

For a relatively small system as the 74180 system with 14 components, a first state of $S = \{1,4,21\}$ has a total number of 220 transitions. Each of these states has again 220 transitions, making the total number of states in the Markov chain very large. For a more complex and realistic system of hundreds of components and an S which contains many solutions the size of the model is exponential prohibitive.

4.3 Model A_2

Based on model A_3 a model is created to minimize model complexity while abstracting as little as possible. The approach used in this model (A_2) is to simulate a SAFARI run while keeping track of the *mean* intermediate S at each step of the algorithm. In this approach, the C-values within $S(k)$ are mean values, averaged over all possible trajectories leading to $S(k)$. The intermediate $S(k)$ is expressed in terms of the intermediate $S(k-1)$.

For example, consider a system of $M = 6$ components and a characterization of performance $S = \{0,1,2\}$, where the actual solutions are

	h	C
s_1	001111	$C = 2$
s_2	010011	$C = 3$
s_3	101100	$C = 3$

As with the real algorithm, the model starts with $k = M$. $S(6)$ contains one $C = 2$ solution and two $C = 3$ solutions. Flipping a variable which causes s_1 to be infeasible has a probability of $\frac{2}{6}$. If this is done a large number of times, on average, $\frac{1}{3}$ of the $C = 2$ solution would remain. The average number of $C = 2$ solutions, which is made impossible to reach by flipping one of the six variables, is equal to

$$S_2 \cdot \frac{2}{k} = 1 \cdot \frac{2}{6} = \frac{1}{3}$$

In other words, on average, a third of the number of $C = 2$ solutions is made infeasible, because the variable that was flipped belonged to a $C = 2$ solution. The same is done for $C = 3$, where the average number of solutions that is made impossible to reach is equal to

$$S_3 \cdot \frac{3}{k} = 2 \cdot \frac{3}{6} = 1$$

These average values are then subtracted from the previous S to obtain the average S at the new k, making $S(k-1) = S(k) - \{0, \frac{1}{3}, 1\} = \{0, \frac{2}{3}, 1\}$.

The average $S(k-2)$ is calculated in the same way, but it is based on the generated average $S(k-1)$. The generalized function for this model is written as

$$S_i(k-1) = S_i(k) - S_i(k) \cdot \frac{i}{k}$$

When the model is applied on a starting value of S a complete trace of averages of intermediate S is acquired. This equals the average number of solutions of each cardinality that would still be feasible at each step k of the actual SAFARI algorithm. The complete trace of the example becomes

k	S		
6	{0,	1.00,	2.00}
5	{0,	0.67,	1.00}
4	{0,	0.40,	0.40}
3	{0,	0.20,	0.10}
2	{0,	0.07,	0.00}
1	{0,	0.00,	0.00}

Note, that, compared to model A_3, the Markov chain has reduced to a 1-dimensional chain, greatly reducing model complexity. Table 4.4 shows example traces of the example system of size $M = 14$ and an observation with a characteristic $S = \{1,4,21\}$. Both traces are acquired by running SAFARI multiple times and taking the average number of solutions at each k. The trace of model A_2 is very accurate when compared with the actual SAFARI algorithm. Unfortunately, however, this is only a valid model for $R = 1$, which is displayed on the left of Table 4.4. At the right hand side of the table is the trace of SAFARI when run with $R = 5$ on the same observation, averaged over 10,000 runs. Both traces show that at $k = 14$ the intermediate S is still equal to the original. The values of S at each of the cardinalities decrease with every next k, but we see that the value of S_3 decreases significantly faster than the others, which is because these relatively larger solutions are made infeasible with a larger probability. The difference in the two traces is that with $R = 1$ the numbers decrease to zero, while with a large enough R the values decrease to some value higher than zero at a k equal to the cardinality.

k	S for $R = 1$			S for $R = 5$		
14	{1.00,	4.00,	21.00}	{1.00,	4.00,	21.00}
13	{0.93,	3.43,	16.50}	{0.93,	3.43,	16.49}
12	{0.86,	2.90,	12.70}	{0.86,	2.90,	12.70}
11	{0.79,	2.42,	9.52}	{0.78,	2.42,	9.53}
10	{0.71,	1.98,	6.92}	{0.71,	1.98,	6.94}
9	{0.64,	1.58,	4.85}	{0.65,	1.60,	4.89}
8	{0.57,	1.23,	3.23}	{0.58,	1.26,	3.32}
7	{0.50,	0.92,	2.02}	{0.52,	0.97,	2.17}
6	{0.43,	0.66,	1.15}	{0.47,	0.75,	1.37}
5	{0.36,	0.44,	0.58}	{0.43,	0.58,	0.84}
4	{0.29,	0.26,	0.23}	{0.40,	0.46,	0.48}
3	{0.21,	0.13,	0.06}	{0.38,	0.39,	0.26}
2	{0.14,	0.04,	0.00}	{0.38,	0.36,	0.00}
1	{0.07,	0.00,	0.00}	{0.38,	0.00,	0.00}

Table 4.4: Averaged S over 10,000 runs of SAFARI on a 74180 observation with $S = \{1,4,21\}$. Left: $R = 1$. Right: $R = 5$.

The effect of the average number of solutions decreasing less when a larger value for R is chosen is explained by the fact that a variable flip is retried after an inconsistency. The algorithm is able to reach lower k, adding to the average number of solutions at that k. For $R = 5$ (i.e., exhaustive retrying) it turns out that the values of the average number of solutions at k equal to the cardinality, which are underlined in Table 4.4, are exactly equal to the distribution of these cardinalities of the actual SAFARI algorithm. This leads us to the following theorem.

Theorem 2. *If R is chosen to guarantee solution minimality, then the trace of an ideal model A_2 contains the probability distribution of cardinalities on its diagonal.*

Proof. The diagonal contains the average values of S for which the solution cardinality is equal to the remaining number of unflipped variables ($k = c$). If SAFARI is run with a sufficiently large R, the intermediate set of feasible solutions contains only one solution when a cardinality is returned. Because a maximum of one feasible solution can remain at $k = C$, and minimality of solutions is guaranteed, the sum of the average number of solutions remaining at $k = C$ for all C is equal to one. Therefore the diagonal contains the probability distribution of cardinalities which are returned by SAFARI. □

4.3.1 Improving A_2

The difference between model A_2 as discussed so far and the SAFARI algorithm is the calculation of the probability of reaching an inconsistency. Reaching an inconsistency is caused by flipping a variable which is shared among the remaining solutions. Flipping this variable makes all solutions impossible to reach. This probability grows as the number of variables decreases and is overcome by using a larger value for R in SAFARI, which results in other variables being tried. For example, if the remaining feasible solutions are

C=3	0	0	0	1	1	1
C=3	1	0	0	0	1	1

then there are two shared variables among the remaining solutions. If either one of these variables is flipped, the system becomes inconsistent and no minimal solution is reached.

If R is chosen large enough to guarantee solution minimality, in effect the variables that are shared among all solutions are taken out of the equation, because a retry occurs whenever these are flipped. In the above example, variables h_5 and h_6 are shared by both solutions. Therefore, only four variables remain that can be flipped resulting in a consistent situation. The probability of flipping a $C = 3$ solution becomes $\frac{1}{4}$, instead of $\frac{3}{6}$. The original calculation of the next intermediate S

$$S_i(k-1) = S_i(k) - S_i(k) \cdot \frac{i}{k}$$

should therefore become

$$S_i(k-1) = S_i(k) - S_i(k) \cdot \frac{i - |shared|}{k - |shared|}$$

where $|shared|$ is the average number of shared variables among the remaining solutions.

If the solution variables are randomly distributed in the health vector, the probability of a variable being shared by all solutions is equal to

$$p = \prod_{i=1}^{k} \left(\frac{i}{k}\right)^{S_i}$$

The mean number of shared variables, thus the mean number of variables which are left out of the equation is

$$|shared| = p \cdot k$$

k	S		
14	{1.000000	4.000000	21.000000}
13	{0.928571	3.428571	16.500000}
12	{0.857143	2.901099	12.692308}
11	{0.785714	2.417582	9.519231}
10	{0.714286	1.978022	6.923077}
9	{0.642858	1.582420	4.846161}
8	{0.571481	1.230882	3.231064}
7	{0.501055	0.925025	2.023492}
6	{0.437926	0.673732	1.179030}
5	{0.400915	0.493434	0.647632}
4	{0.400915	0.383199	0.335299}
3	{0.400915	0.308474	0.134957}
2	{0.400915	0.223213	0.000000}
1	{0.400915	0.000000	0.000000}

Table 4.5: Trace of model A_2 on a 74180 observation with $S = \{1,4,21\}$.

Table 4.5 shows the trace of model A_2 improved to model SAFARI with a large R. This version is clearly different from the initial version and resembles the true SAFARI trace in Table 4.4 in that the values of S_i do not drop to zero but slow down their decrease due to the effect of the shared variables which influence the algorithm at lower values of k.

It must be noted, however, that the resulting probability distribution of cardinalities is still not perfect. This is because variables of the solutions are assumed to be randomly distributed in the health vector, while in reality this cannot be the case, since subsumption of solutions would occur when they were placed randomly. In addition, the values on the diagonal can not serve as a probability distribution as the values do not sum up to one. There are two options to deal with this problem and to be able to use the outcome in practical applications:

- Assume that the remaining probability of solution cardinality is distributed over larger cardinalities.
- Normalize the values.

The latter option would be the more logical solution, since, due to the approximation, the probabilities might add up to more than one. This ensures that the cardinalities in S are the only cardinalities in the resulting probability distribution. This would be correct, compared to reality, since solution minimality ensures that only these cardinalities are returned.

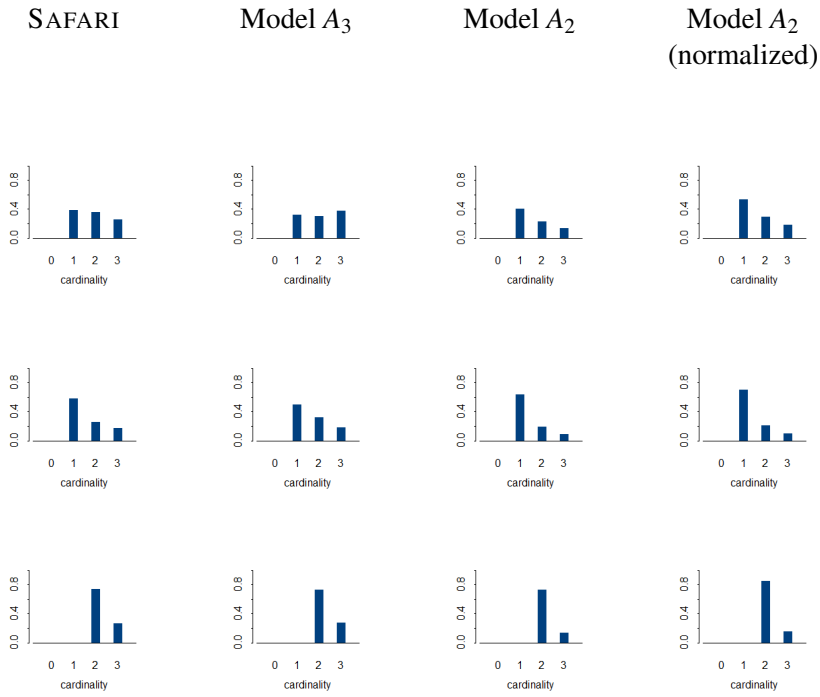

Figure 4.7: Comparison of SAFARI output with models A_3, A_2 and the normalized model A_2.

Figure 4.7 shows the pdf results of model A_2 on three different observations on the example system, compared to the actual SAFARI output and to the output of model A_3. The last column contains the normalized values of model A_2. When compared to the SAFARI output, model A_2 seems to model SAFARI more accurately that model A_3, which serves as a simulator which should resemble SAFARI closest. This can be explained by the fact that model A_3 assumes random observations of S, while SAFARI is run on real observations on the system, which are considered outliers compared to the random observations. The output of model A_2 differs from the output of model A_3 because of the assumption that the variables of the solutions are assumed to be randomly distributed in the health vector and therefore does not account for solution subsumption. By chance, this difference transforms the resulting pdf to resemble the SAFARI outcome more that model A_3. It would be more accurate to compare the model to the reference model A_3, which stands as close as possible to the actual algorithm, but which excludes the information of the positioning of solution variables in the health vector. Looking at the output of the normalized version of model A_2 we see that the lower cardinalities have a slightly larger probability mass when compared to

the output of model A_3. Nonetheless, the difference between model A_3 and the normalized model A_2 is small, indicating that model A_2 is usable as a model for the behavior of the SAFARI algorithm. Both model A_3 and A_2 result in outputs with no probability mass at cardinalities of which there are no existing solutions.

An advantage of this model is that it is computationally cheap. The number of computations is determined only by the number of components M. In the worst case, S contains significant values for all cardinalities, which results in M^2 computations of new values for each cardinality in S for all k. Half of these values are known to be zero, since the number of solutions of cardinality C is zero if $k < C$. In addition, the lower cardinalities in S are usually dominant, which makes it possible to exclude the higher cardinalities with only little difference in behavior and limit the number of computations even further.

4.4 Model A_1

Model A_3 and, less so, model A_2 try to follow the SAFARI algorithm step by step, calculating intermediate probabilities of each cardinality. The low level of abstraction of model A_3 results in a computationally complex model, with the number of possible states exponentially growing for larger M. Model A_2 is more abstract than A_3 resulting in lower computational cost. Only $\frac{1}{2}M^2$ calculations are required to compute the estimated pdf. With the next model, A_1, we will further decrease the computational complexity and have an even greater abstraction.

Where the models A_3 and A_2 follow the steps of the SAFARI algorithm, lowering k at each step, model A_1 computes the probability of each cardinality directly from S. Again, we assume that R is chosen large enough to guarantee that minimal solutions are always reached. Model A_1 is defined as

$$n_i = \frac{S_i}{\binom{M}{i}}$$

$$p_i = \frac{n_i}{sum(n)}$$

where i is the cardinality and p is the resulting pdf, containing the probabilities of SAFARI returning a certain cardinality. The probability of returning a solution of cardinality i is determined by the the number of solutions of this cardinality, the number of ways such a solution can be reached, independently of other solutions. This probability is decreased if more solutions exist in the solution space, however, an extra $C = 1$ solution has more impact on other solutions than an extra $C = 3$ solution.

To understand the reasoning of the model we will investigate some imaginary cases. First, the case that a single cardinality exists in the solution space. For example, let $M = 6$ and consider the following solution space

011111

There is just one cardinality possible as outcome, and because we guarantee minimal solutions, SAFARI will always return this solution. Therefore the pdf will have a probability of

1 for $C = 1$. The same holds for the following solution space

000001

000010

where there are two solutions, but both having five faulty components. In this case, the resulting pdf will have all probability mass at $C = 5$. The single cardinality case is handled by model A_0 by having a non-zero n_i only if S_i is non-zero. The 100% probability mass at that cardinality is the result of normalizing all n_i to get p.

Another interesting case is a high cardinality solution together with a low cardinality solution. For the same system of $M = 6$ let there be a solution of $C = 5$ and $C = 1$. The solution space would be

000001

111110

When the SAFARI algorithm processes this system and first flips the variable belonging to the $C = 1$ solution, it will return the $C = 5$ solution. In all other cases it will return the $C = 1$ solution. The probability that the $C = 5$ solution is returned is equal to $\frac{1}{\binom{6}{5}} = \frac{1}{6}$, leaving a probability of $\frac{5}{6}$ that the $C = 1$ solution is reached. This is verified by running SAFARI for this imaginary system. Unfortunately, model A_1 is unable to correctly calculate this, since both n_5 and n_1 are equal and thus normalizing this results in $p = \frac{1}{2}$ for both solutions. If this was a realistic scenario, one would dismiss this model because of this result. However, a $C = 5$ solution for a system with $M = 6$ is a diagnosis where almost all components are faulty. One may assume that this is not the case in realistic systems, since it would mean that the only possible way to solve the problem is to replace the complete system. For this model, let us assume that the maximum number of simultaneous faults is less than half the number of components.

With this in mind, we consider the following case of the same $M = 6$ system and three solutions of cardinality $C = 3$, $C = 2$ and $C = 1$, respectively. There are two variations of solution spaces. One is a disjunct set of solutions:

000111

111001

111110

And the other contains a shared variable:

000111

011011

111101

The behavior of SAFARI for these different solution spaces is very different and hard to model, since the probability mass of a solution without a shared variable is increased drastically when the shared variable is flipped. Model A_1 works at a more abstract level, dismissing this notion of shared variables among solutions. The basis for this model is the probability of reaching a solution of a given cardinality. In this example, the probability of reaching the $C = 3$ solution, independently of other solutions, is equal to $\frac{3}{6} \cdot \frac{2}{5} \cdot \frac{1}{4}$ or $\frac{1}{\binom{6}{3}} = \frac{1}{20}$. In the same manner, reaching the $C = 2$ solution has $p = \frac{1}{15}$ and the $C = 1$ solution has $p = \frac{1}{6}$. The preference of SAFARI for lower cardinality solutions is reflected in these probabilities. Because these are all possible solutions in the solution space, the next step is to normalize these probabilities to maintain the relative proportions. The resulting pdf of model A_1 for this example is shown in Table 4.6. The columns represent the pdf for

C	SAFARI	A_1	A_2	A_3
1	0.612	0.588	0.429	0.816
2	0.267	0.235	0.312	0.182
3	0.121	0.176	0.260	0.001

Table 4.6: pdf of model A_1 compared to SAFARI for $S = \{1,1,1\}$ and $M = 6$

SAFARI and models A_1, A_2 and A_3. It shows that model A_1 resembles SAFARI quite well. In fact, for this example model A_1 performs better than models A_2 and A_3.

When, for the same example, an extra solution of $C = 1$ is added, n_1 is doubled. The probability of returning a $C = 1$ solutions, independently of other solution, is twice as large. However, when normalized, including all other solutions in the calculation, the actual increase of probability is not doubled. One could say that other existing solutions prevent the probability to increase as much. As with the previous example, the pdf of all models is given in Table 4.7. Like the actual SAFARI algorithm, model A_1 has a clear reaction on

C	SAFARI	A_1	A_2	A_3
1	0.800	0.741	0.501	0.944
2	0.144	0.148	0.273	0.056
3	0.055	0.111	0.227	0.000

Table 4.7: pdf of model A_1 compared to SAFARI for $S = \{2,1,1\}$ and $M = 6$

the extra $C = 1$ solution. It also shows that the probability of returning the $C = 3$ solution according to model A_1 is twice the actual probability of SAFARI. But again, A_1 seems to be better at modeling SAFARI than the other models.

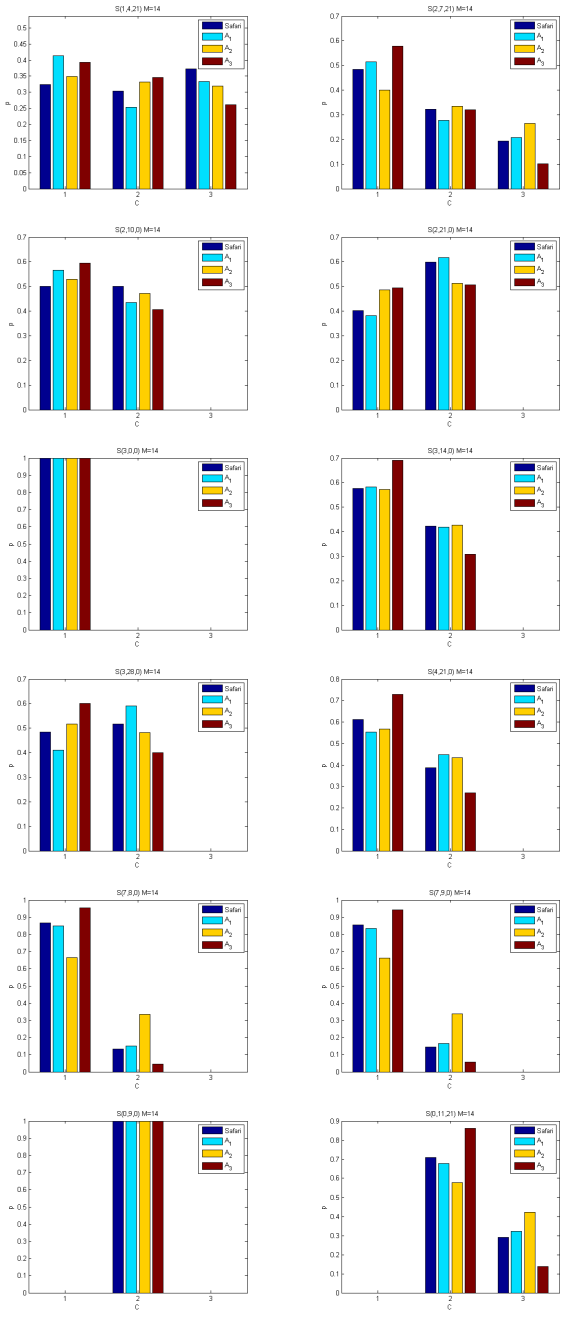

Figure 4.8: Model A_1 of a SAFARI run and a diagnosis of cardinality c

4.5 Summary

In this chapter we have explored three new performance models for the SAFARI algorithm, as an improvement on the original model, A_0. All three models use S as input and return the

pdf of solution cardinalities f_a.

First, we have discussed model A_3, the least abstract model, which defines all possible transitions from one state (intermediate S) to all possible next states. Any state consisting of a single cardinality serves as ending state for the model. By calculating the probabilities of all state transitions, the probability of reaching each cardinality can be determined. However, this is a very expensive method and unrealistic for large systems, since the number of possible states grow exponentially for larger M. The Monte Carlo approach, traversing the model multiple times to achieve a pdf of solution cardinalities, is another possibility, however, this method consumes time and could suffer from lack of numerical precision.

Second, the more abstract model A_2 is introduced, which starts with a given S, and, with each step k, calculates the average S for that step in the algorithm. The resulting pdf is constructed by using the values on the diagonal of the trace of average S of the model. Since this computational complexity of this model is small ($O(M^2)$), and because this model seems to model the actual algorithm more accurately, this is a far more attractive model to use, compared to model A_3.

Last, we have our most abstract model, A_1, which computes the pdf directly (in a two-step approach) from the characteristic S. It calculates the probability of returning each cardinality, given only the number of components and the number of solutions of the same cardinality. After calculating this for all cardinalities, the pdf is calculated by normalizing the values, which models the influence that solutions of different cardinality have on each other. This is the computationally cheapest model and experiments show that it seems to model SAFARI more accurately than model A_2. However, more extensive testing should be performed, with different systems and observations, to be able to decide which of the performance models is best at modeling SAFARI.

Chapter 5

Termination

Next to understanding the behavior of the algorithm, the reason for modeling SAFARI is to be able to state something about the level of completion. At this moment, there is no way to determine the percentage of solutions already returned by SAFARI, and more importantly, the level of completion of the *MC* solutions. The following sections are focused on determining an ending condition for the stochastic algorithm, both by making use of the discussed performance models and by making use of other methods.

5.1 Prediction of S

An important application for the performance models would be to predict the S based on the distribution of solution cardinalities that SAFARI obtained thus far. In practical diagnostic situations the minimal cardinality solutions are the most desirable ones. Because SAFARI is a completely stochastic algorithm, it is never known for certain if the resulting minimal cardinality of its returned solutions is in fact the true minimal cardinality. Although it can be shown that if SAFARI returns a zero cardinality solution (the nominal solution) or a single cardinality solution, this *is* the minimal cardinality, the SAFARI algorithm will typically be used if a diagnosis is expected to contain a relatively high minimal cardinality, in which case it is unknown if *MC* is reached during algorithm execution.

In addition to knowing the minimal cardinality, it is highly desirable to know the total number of solutions of the minimal cardinality. Since SAFARI does not include failure probabilities that are present in the model to return the most probable solutions, as is done with most conventional algorithms, every solution of the same cardinality is considered to be equally probable to be the true solution to the problem. Because in this view higher cardinality solutions are less probable, all minimal cardinality solutions should be known and used as candidates for examining the system.

A database of solution cardinality distributions of a certain system can be obtained either by calculating it for a large number of possible S or by storing the resulting distribution of cardinalities of each actual run when SAFARI is used in practice. Using this database a new diagnosis can be categorized by comparing the data of known S with the results of the

algorithm early on in its number of runs. Because computational effort is small, a large database can easily be made in a pre-diagnosis stage.

In theory a prediction of the S can be made, which contains the minimal cardinality and the number of solutions of that cardinality. However, this still a topic of research and experimental results are therefore not included in this thesis.

5.2 Algorithm Termination

The SAFARI algorithm can result in a solution which is already present in the set of previously returned solutions. It can also return a new solution, possibly of a lower cardinality than any of the solutions currently known. One can be never sure if there is still a solution with a lower cardinality. It is also uncertain if all minimal cardinality solutions are returned. With an increasing number of runs the probability of having reached the true minimal cardinality and the probability of having reached all minimal cardinality solutions increase as well. However, the number of runs necessary, to be certain that no important solution is missed, is unknown.

Using the model to predict the S belonging to the observation currently under diagnosis it is possible to calculate the probability of having reached the minimal cardinality and even the probability of having returned all solutions of minimal cardinality. This is essential for determining a safe termination point for the algorithm. Different observations can require a different number of runs to have the same level of certainty of optimality and completeness of minimal solutions.

Ideally, the algorithm can be adopted to provide the user with the current minimal cardinality and the probability that this is the true minimal cardinality at run time. The same can be done for the number of minimal cardinality solutions.

5.2.1 The $n(k)$ Curve

Now that we have a model that predicts the probability density function f_a from S, how do we determine that SAFARI is 'done'? That is, when do we know that we have seen, for example, 99.99% of the posterior probability mass? Using the pdf from our model we are able to predict the total number of unique minimal solutions.

First, consider the following example. We have an S of $\{N,0,0,...\}$, having a solution space of just one cardinality. Let $S' = \{n,0,0,...\}$ be the collection of solutions seen by SAFARI thus far. Then

$$p(k) = \frac{N - n(k)}{N} = 1 - \frac{n(k)}{N}$$

is the probability of seeing a new sample at run k, since each run results in a minimal solution (given that R is chosen big enough) and the probability of getting a solution that we have seen already becomes greater with every new unique solution. As explained throughout this thesis, same-cardinality solutions are equally probable to reach using the SAFARI algorithm.

Let $N = 6$ at run 1, when no solutions are yet seen, then $p = \frac{6}{6}$. In other words, the first run will always result in one of the N solutions. During the second run, there is a probability

of $\frac{1}{6}$ that the first solution is found again, leaving $p = \frac{5}{6}$ that a new solution will be found. With every run, the probability of finding a new solution decreases. From this follows:

$$n(k) - n(k-1) = p(k)$$

Solving this, we get:

$$n(k) = N(1 - (1 - \frac{1}{N})^k)$$

From this $1 - e^{-x}$ type of curve, we are able to estimate N. In our example, at run 10 we have reached $n(10) = 5.03$. We have probably found five of the six unique solutions, or a probability mass of $\frac{n(k)}{N} = 0.84$. To have a posterior probability mass of 99.99% that we have found all six solutions, we need to have run for at least $k = 51$ times.

We can apply this to multiple bins as well. Let the characteristic $S = \{N_1, N_2, N_3, ...\}$ and the set of found solutions thus far $S' = \{n_1, n_2, n_3, ...\}$. Furthermore, we have the pdf $f = \{p_1, p_2, p_3, ...\}$, as returned by a model of SAFARI. Every bin has its own $n(k)$ curve, however, each of these curves is normalized according to its corresponding p_i. This is because the probability of having found a new unique solution of cardinality i is now multiplied by the probability of having actually found a solution belonging to the cardinality-i bin, which is given by the pdf. The formula now becomes:

$$n_i(k) = N(1 - (1 - \frac{1}{N})^{kp_i})$$

The result is that the individual cardinality curves are 'stretched' on the k-axis. In other words, we need to have executed more runs to get to an acceptable probability of having found all *MC* solutions. Fortunately, the behavior of SAFARI is such that most of the probability mass is located in the most important lower cardinalities, as shown in the previous chapters. Therefore, the extra number of runs caused by the stretching of the k-axis is minimized by the nature of SAFARI.

Figure 5.1 shows an actual SAFARI progression of unique solutions. The running example is used with an observation of category $S = \{2, 7, 21\}$. For each cardinality a different figure is used, because of the difference in scale of the k-axis. It shows the cumulative number of unique solutions (n) at each run (k) of the algorithm. This is the 'stairs-curve' plotted as a solid blue line. Additionally, the $n(k)$ curve of empirical SAFARI data is plotted as a solid red line. This is calculated from the pdf of 3000 executions of the SAFARI algorithm with 3000 runs each.

It is interesting to note that the $n(k)$ curve is located at the 'bottom' of the stairs-curve. It fits through the values of n at the latest k. In other words, in runs through the points of transition to the next n.

On top of each figure, the $n(k)$-curve of the resulting pdf of model A_1 to A_3 is plotted. This shows the validity of the respective models. The closer the plot resembles the empirical data, the better the 'fit' of the model.

5.2.2 Estimation of S

In the previous section we reasoned from a known S to a $n(k)$ curve for each cardinality. We saw that the influence of one cardinality on another is determined only by their respective

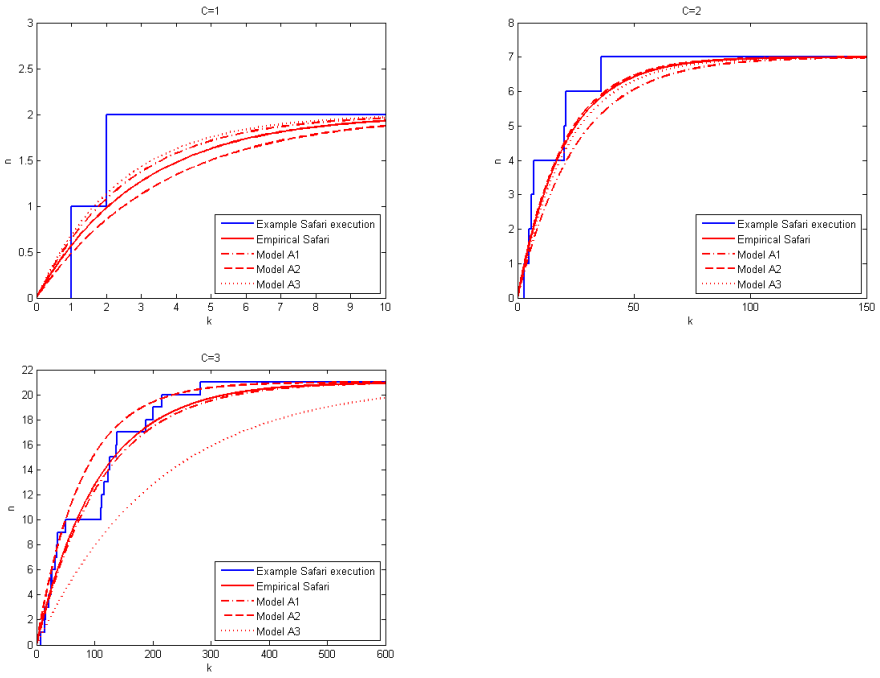

Figure 5.1: n_k curve of the models compared to an actual SAFARI execution.

probability of being found, which is captured by the pdf. In practice, however, we know nothing of the actual S, but we are interested in an accurate estimation of S based on the solutions returned by SAFARI. For a practical application of the theory discussed thus far, we need to 'fit' a $n(k)$ curve onto set $S' = \{n_1, n_2, n_3, ...\}$ of solutions found thus far by SAFARI.

Consider again the running example with an observation of $S = \{2, 7, 21\}$. In this example we will focus on cardinality $C = 2$. At specific points in the execution of the SAFARI algorithm, a fit is made onto the the S' known at that moment. The function used for fitting is

$$y = N(1 - (1 - \frac{1}{N})^{px})$$

where N and p are the coefficients to be estimated[1]. Table 5.1 lists the number of found solutions of $C = 2$ at a selection of runs. For each run, the estimated N' and p' are also listed. We see that the estimated N'_2 quickly approaches the true value of $N_2 = 7$. This is also depicted in Figure 5.2.

Figure 5.3 shows the 'stairs' plot of the progressing n_2 over k, together with the plotted $n(k)$ curves of the estimated N'_2. The dotted curves are calculated at $k = \{10, 20, ..., 90\}$ and

[1] For fitting, the *Curve Fitting Tool* of Matlab was used, with *non-linear least squares* as fitting method. The function for fitting is $y = N(1 - (1 - \frac{1}{N})^{px})$. The coefficients to be fitted are N, bounded by $(1, 100)$, having starting value 1, and p, bounded by $(0, 1)$, having starting value 0.5.

k	$n_2(k)$	N_2'	p_2'	SSE
10	4	100.0000	0.4496	3.0822
20	5	4.8189	0.5875	6.5553
30	6	6.8028	0.4624	10.5931
40	7	7.1741	0.4398	12.2605
50	7	7.3507	0.4299	12.4588
60	7	7.2680	0.4358	12.5501
70	7	7.1951	0.4419	12.7114
80	7	7.1467	0.4466	12.8597
90	7	7.1149	0.4499	12.9754
100	7	7.0933	0.4522	13.0619

Table 5.1: Least squares fitting of N at a selection of runs.

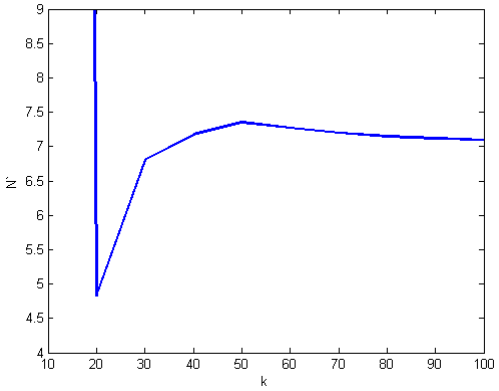

Figure 5.2: Increasing accuracy of estimation N' at runs $k = \{10, 20, ..., 100\}$.

the solid curve is calculated at $k = 100$, when it becomes less likely that there is no new unique sample of $C = 2$.

The value of p is quite different from the actual value, as given from the empirical data. The true value of p should approximately be 0.32. The curves also show this, as they are located more to the center of the stairs-curve. The larger value of p stretches the curves less in the axis of k than was seen in Figure 5.1. This behavior is explained by the fact that we are trying to estimate a real valued curve through a discrete function. An added effect of this, and also because we are increasing the sample space with each run, is that the SSE keeps increasing, instead of decreasing, meaning that we can not use this as a criterion for terminating SAFARI. A more elegant solution would be to have a discrete estimation. However, this requires some more investigation, which is outside the scope of this thesis.

Another way to improve the estimation of the true $n(k)$ curve is to consider only part of the data of the 'stairs'-plot. As we have noticed in the previous section, the $n(k)$ curve

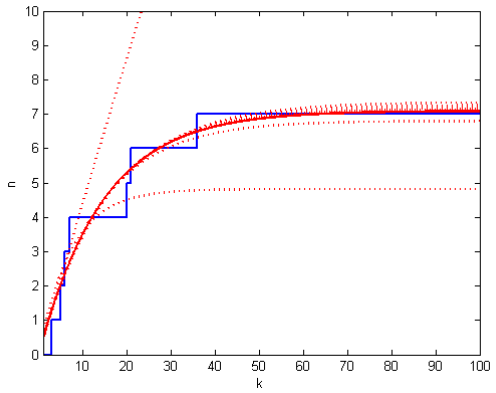

Figure 5.3: $n(k)$ curves of N' fitted at runs $k = \{10, 20, ..., 100\}$.

touches the lower part of the stairs. If we would only fit the function to those points that have the highest value of k for each value of n, then we should obtain a better estimation of p as well. For example, consider the $S = \{2, 7, 21\}$ case, where we have seen, in one SAFARI execution, new unique solutions of cardinality $C = 2$ at runs k = 3,5,6,7,20,21 and 36. Table 5.2 shows the data points used for fitting the curve at different values of k. We only use the 'bottom-right' points of the stairs data. Notice that the last coordinate is not fixed, but changes with increasing k, since we the last known value of k for the current n is the current run. The resulting values are shown in Table 5.3. It is clear that the estimated value of N approaches the true value 7 after about 60 runs according to this model. A line for run $k = 1000$ is added as well to show at which values the estimated N and p settle. As predicted, the value of p' estimated with this method comes closer to the true value of 0.32 than with the previously discussed method.

k	data points (k,n)
1	$(1,0)$
5	$(2,0),(4,1),(5,2)$
10	$(2,0),(4,1),(5,2),(6,3),(10,4)$
20	$(2,0),(4,1),(5,2),(6,3),(19,4),(20,5)$
50	$(2,0),(4,1),(5,2),(6,3),(19,4),(20,5),(35,6),(50,7)$
100	$(2,0),(4,1),(5,2),(6,3),(19,4),(20,5),(35,6),(100,7)$

Table 5.2: Data points used in the estimation of N'_2 and p'_2.

This curve can be used for a termination criterion. For example, the SAFARI algorithm could stop acquiring new diagnoses if the relative delta of N' is below some defined threshold. Also, the estimated p' of each cardinality can be checked with the true pdf as obtained thus far. If the estimated p' differs too much from the true frequency of finding solutions of

k	$n_2(k)$	N_2'	p_2'	SSE
10	4	100.0000	0.4018	1.3791
20	5	4.7609	0.4226	1.8245
30	6	7.0333	0.3765	2.2795
40	7	7.6470	0.3545	2.5638
50	7	7.3107	0.3596	2.3331
60	7	7.1252	0.3644	2.2640
70	7	7.0321	0.3673	2.2385
80	7	6.9844	0.3689	2.2274
90	7	6.9592	0.3697	2.2221
100	7	6.9457	0.3702	2.2194
1000	7	6.9293	0.3707	2.2162

Table 5.3: Least squares fitting of N at a selection of runs on a selected set of data points.

the corresponding cardinality, then this could be an indication that more unique solutions are expected. However, this is a very fuzzy condition, and probably not applicable in practice, until a method of estimation is found that is more exact. We could benefit more from a termination condition that takes into account the probability of having not found all relevant solutions.

5.2.3 Coupon Collectors Problem

The problem of expecting a unique solution of some cardinality resembles *the coupon collectors problem* described in probability theory [8, 9]. This problem is defined as: If a collector tries to find all n coupons for his collection and the coupons are selected with replacement, then what is the probability that more than t sample trials are needed to collect all n unique coupons? For each cardinality we can ask the same question: If we have seven unique solutions of cardinality two, but every time we get a $C = 2$ solution it could be any of the seven, with equal probability, then what is the probability that more than k runs are needed to get all of the unique solutions?

The expectation of the number of runs (K) to get all n unique solutions is:

$$E(K) = n \cdot H_n = n \ln n + \gamma n + \frac{1}{2}$$

where H_n is called the harmonic number and $\gamma \approx 0.577$ (the Euler-Mascheroni constant).

For the example case of the seven unique solutions of cardinality two, the expected number of runs is equal to 18.16. However, this is only considering the cardinality two solutions. In fact, we are drawing from the total set of solutions of all cardinalities. The expected number of runs for the seventh unique solution of $C = 2$ therefore needs to be divided by the probability that a solution is of cardinality two. This is given by the pdf of cardinalities. In the case of the example of $S = \{2, 7, 21\}$ the probability of getting any of the $C = 2$ solutions at a given run is equal to 0.32. This results in an expected number of

runs equal to $\frac{18.16}{0.32} = 56.75$. According to this theory we will need, on average, 57 runs before all seven unique solutions are found, which is consistent with our previous results.

However, the most interesting question is: If we have seen only six solutions at run $k = 57$, how many runs should we execute after this point, before concluding that six is the true number of $C = 2$ solutions? The coupon collectors problem provides us with an upper bound by using the Markov inequality:

$$P(K \geq cnH_n) \leq \frac{1}{c}$$

If we solve this inequality for $n = 7$ and a target probability of $\frac{1}{100}$, then we find that the probability that more than 1816 runs are required is less or equal to $\frac{1}{100}$. In fact, this number of runs only considers the runs at which a $C = 2$ solution is returned by SAFARI.

We can also look at the probability bound when we are at a certain run. For example, if $k = 100$, what can we say about the probability P that we have not found all solutions yet? Solving the same inequality, we get $P \leq 0.18$. That is, the probability that we have not found the seventh unique solution of $C = 2$ at $k = 100$ is at most 0.18. Again, the run number actually is the number of runs in which we have found a $C = 2$ solution. The calculated bounds are not very strict as the number of runs to get a decent probability ranges in the thousands, while the correct number of solutions in our example tracing is already found at run $k = 35$.

A stricter bound exists, using the Chebyshev inequality, which is defined as:

$$P(K - nH_n \geq cn) \leq \frac{2}{c^2}$$

If we solve this for the same situation as above, we find that the probability of k being greater than 117 is less or equal to $\frac{1}{100}$. And the probability that we have not found the seventh unique solution of $C = 2$ at $k = 100$ is at most 0.015. This is a great improvement on the previous bound inequality. However, considering that this calculation only applies to the solutions of the same cardinality, the number of runs should be at least $\frac{117}{0.32} = 366$ to guarantee a maximum probability of $\frac{1}{100}$ that the next solution of the same cardinality is a unique solution.

The coupon collectors problem gives us a method for calculating bounds on the probability of having found all solutions of a selected cardinality. However, the quality of this bound is still unclear. When applying this to the SAFARI algorithm as a termination criterion, the probability can be communicated continuously to the user. The fact that this probability can be expressed will improve the applicability of the SAFARI algorithm, because there is a defined ending to the diagnosis process after which a summation can be given of the probabilities of missing a diagnosis for each of the cardinalities in the solution space.

5.3 Termination in Practice

In this chapter several techniques are discussed which could aid in deciding when to stop the execution of the SAFARI algorithm. Before stopping the generation of new solutions,

we should have enough certainty that there are no more unseen solutions of the relevant lowest cardinalities, since most of the posterior probability mass of the problem resides in the *MC* (and possibly in $MC+1$, if the number of *MC* solutions is very small). One or more of the following steps can be implemented as a termination procedure.

- If a database of multiple performance model executions (and previous diagnoses) exists, then the pdf f_a of the algorithm during the diagnosis process can be matched to a corresponding S. If the database contains data that is distinctive enough and this match is clear enough, then from the obtained S the number of *MC* solutions can be read and used as stopping criterion. However, these required conditions are not very likely to be met.

- The performance model can be executed at every run of the SAFARI algorithm using the S as obtained thus far from the actual runs. The predicted pdf can then be compared to the actual f_a to see if this matches. This method assumes that the performance model accurately models the SAFARI behavior.

- During the execution of the algorithm, the *Coupon Collectors* method can be used, which gives a defined bound for the number of runs needed after which the probability of finding a new solution of *MC* (or another cardinality) is, for example, less than a percent.

- The previous two methods can be combined, where the performance model predicts f_a early in the execution of the algorithm. Based on this pdf, the *Coupon Collectors* method can determine the number of runs needed before stopping the algorithm. The same formula can be used to give the (bounded) probability that the next run results in a new *MC* solution. This probability will decrease to an acceptable value, which can even be determined by the user at runtime. Having an estimation, or progress indication, this early in the execution may be a very attractive feature for the SAFARI algorithm, as stochastic methods usually provide no information to the user about their progress.

Chapter 6

Conclusions

A stochastic diagnostic engine like SAFARI can be used to diagnose problems with an expected higher *MC*, where deterministic methods fail due to space and time complexity. Because of the stochastic behavior, a model is required to determine the optimality and completeness of these algorithms. However, the previous model of SAFARI is insufficient to be used as a general model for all inputs to the algorithm. In this thesis we set out to find a performance model for the SAFARI algorithm and to devise a termination procedure for the diagnosis progress. To achieve this, we have introduced a characterization of performance, dubbed *S*, and showed that this is an important factor for the behavior of SAFARI. Three new performance models are introduced which are based on this *S*. Finally, termination of the algorithm is discussed and a process for terminating SAFARI is proposed.

6.1 Results

Our conclusions are as follows. We have established that *S* largely determines the behavior of SAFARI. This characteristic set *S* contains the number of existing minimal solutions of each cardinality of a certain group of problem. Experiments have shown that the number of components *M* together with the characteristic *S* are the dominant influence on the behavior of the SAFARI algorithm. Since the observation on a system is the only variable factor in a diagnosis and because each observation leads to a certain set of minimal solutions, *S* serves as a characterization of observations as well. Although experiments have shown that the characteristic *S* is not the only factor in the behavior of SAFARI, it is the most influential factor. Another factor which explains the behavior of SAFARI is the way solution variables are distributed in the health vector. Solutions can overlap, causing some variables being shared by many solutions, which increases the possibility that a single variable flip makes many solutions infeasible. This effect is researched, however it is shown that there must be another factor that causes SAFARI to behave differently other than the characteristic *S* and the distribution of solution variables. Although this other factor is still an unknown, the effect is relatively small and it can be concluded that *S* is sufficient to model SAFARI behavior. This characteristic *S* is used to create new models of the behavior of SAFARI.

A model A_3 is constructed, which resembles the SAFARI algorithm as much as possible,

and only abstracts the exact location of solution variables in the health vector. It is a model which is based on the traversals from one intermediate S to all other possible S. It assumes a large enough number of retries R of SAFARI to guarantee solution minimality. Mode A_3 is computationally expensive, because for a realistic system of many components and many existing minimal solutions, the number of states in the model explodes exponentially. Because the computational complexity of model A_3 it is impractical to be used in reality. This computational complexity also seems to be expressed in the inaccuracy of the model output, making it even less applicable.

Model A_2 is a computationally efficient model which has the potential of being used in practice. It is based on calculating the mean of intermediate values of S at each step k in the algorithm. The output of an ideal version of model A_2, created by statistical information of multiple SAFARI runs, show that the true distribution of solution cardinalities is shown in the trace of the intermediate values of the mean of S. A transformation formula to calculate S for the next k is researched to model the SAFARI algorithm with an R chosen to guarantee solution minimality. Model A_2 is efficient and can be used in practice at run time.

Model A_1 was created as an attempt to even further decrease complexity. It is a higher level of abstraction than the previous models. It is able to efficiently calculate the estimated pdf of the problem with one calculation per cardinality. Propagation of calculation errors is avoided using this model. Results have shown that this model outperforms models A_2 and A_3 for a number of observations on the running example system, also for an observation resulting in higher cardinality solutions. This has led us to conclude that of the models discussed in this thesis, model A_1 is the best model to explain the behavior of SAFARI.

The models can be used to predict the characteristic S for a problem, based on the distribution of solution cardinalities that are obtained during the execution of the algorithm. The characteristic S provides information about the cardinalities existing in the solution space, where the minimal cardinality is of special interest to the user. Using the models for predicting S is not further explored in this thesis.

To be able to define a termination criterion for the SAFARI algorithm, we have introduced the $n(k) - curve$, which is constructed using the runs at which the algorithm returns a new unique solution. This curve is used to estimate the total number of unique solutions of selected cardinalities, effectively estimating S. The estimated value will approach the true number of unique solutions. The decreasing difference between the estimations of each consecutive run can be used for defining a threshold to be used as termination criterion.

Finally, the coupon collectors problem from probability theory is applied to this problem to provide a termination criterion which is based on probability bounds. With these bounds, a defined level of certainty can be given to the end user about the diagnostic quality of the SAFARI algorithm. Although the explored bounds are not very strict, it provides a good starting point for the termination of the diagnosis process.

6.2 Future Work

The work done in this thesis is mostly on the *Proof of Concept* level. Many follow-up experiments need to be conducted to build upon the work done in this thesis. Future work

includes the following directions:

- In this thesis, the algorithm and models are tested against a well analyzed running example, which is not too small having fourteen components, and of which all possible observations and diagnoses are known. However, to have a better validation of the models introduced in this thesis, they should be tested against a great number of systems and observations. A well-known set of systems, used in many model-based diagnosis papers, is the ISCAS85 benchmark [2]. Consisting of ten systems, the number of components ranging from 160 to 3512, this will give a good overview of the applicability of the models. Next to this benchmark, the four medium-sized circuits of the 74XXX discussed in [10], with 19 to 65 components, could be extensively analyzed, as done with the running example system in this thesis, to form a more firm base for the models.

- The models discussed in this paper assumed that the systems are weakly modeled, that is, there is a description of the healthy behavior of the system components. On the other hand, systems can also be modeled strongly, where different fault-modes can be described, resulting in more detailed and less diagnoses. The current SAFARI algorithm depends on systems being modeled weakly, because a weakly modeled component that is set as unhealthy by the algorithm can also behave as described by its healthy state. In this case, the all-components-unhealthy initial state will also explain nominal behavior. In contrast, strong models have explicit behavior descriptions of faulty components. A system with all components being at fault will not explain nominal behavior in this case. These strong models could be translated into weak(er) models, enabling SAFARI to diagnose the system. The effect of the degree of weakness on the algorithm can be analyzed with the models discussed in this thesis. Also, the models might aid in adopting the algorithm to handle stronger models.

- There are some other possible improvements on the SAFARI algorithm that might be researched using the models discussed in the thesis. For example, one improvement could be to make SAFARI hybrid. In this case SAFARI will switch from being stochastic to deterministic at a certain point in the search. The great benefit would be that all solutions are visited near a local optimum of the search space, instead of depending on chance to get to a nearby solution. The models in this thesis could aid in determining the best point at which to make this switch to deterministic search. However, the termination criterion, as currently defined, will be incorrect, because the resulting pdf will be very different as multiple diagnoses are found during one run. The termination criterion will most likely be modified to account for a partially deterministic algorithm.

- Every model in this thesis has assumed an R great enough to assure optimality of the algorithm. A smaller R is no problem if solutions exist with cardinality equal or lower than this value, since the selection of new variables to flip is done without repetition. However, if the minimal cardinality is higher, then the value of R will play a role. A correct model of SAFARI should also consider this number of retries per run, since

it is a parameter to the algorithm. This will result in more complex models of the algorithm and the impact of dismissing this parameter should be more thoroughly investigated.

- The termination criterion is now based on probability bounds that are not very strict. Further investigation into better bounds is required for terminating the SAFARI algorithm earlier, without decreasing the certainty of having found all minimum cardinality solutions.

Bibliography

[1] A. Bajwa and A. Sweet. The Livingstone Model of a Main Propulsion System. In *Proceedings of the IEEE Aerospace Conference*, 2003.

[2] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Proc. of International Symposium on Circuits and Systems*, volume 663, page 698, 1985.

[3] J. De Kleer and B.C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[4] K. de Kleer Alan et al. Characterizing Diagnoses and Systems. *Artificial Intelligence*, 56(2-3):197–222, 1992.

[5] A. Feldman, G. Provan, and A. Van Gemund. Computing Minimal Diagnoses by Greedy Stochastic Search. *Proc. AAAI08*, pages 919–924, 2008.

[6] 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.

[7] A. Feldman and A. van Gemund. A Two-step Hierarchical Algorithm for Model-based Diagnosis. In *Proceeding of the National Conference on Artificial Intelligence*, volume 21, page 827. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[8] William Feller. *An Introduction to Probability Theory and Its Applications, Vol. 1, 3rd Edition*. Wiley, 3 edition, January 1968.

[9] P. Flajolet, D. Gardy, and L. Thimonier. Birthday Paradox, Coupon Collectors, Caching Algorithms and Self-organizing Search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

[10] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering. *IEEE Design & Test*, 16(3):72–80, 1999.

[11] N. Muscettola, P.P. Nayak, B. Pell, and B.C. Williams. Remote Agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, 1998.

[12] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[13] B.C. Williams and P.P. Nayak. A Model-based Approach to Reactive Self-configuring Systems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 971–978, 1996.

[14] B.C. Williams and R.J. Ragno. Conflict-directed A* and its Role in Model-based Embedded Systems. *Discrete Applied Mathematics*, 155(12):1562–1595, 2007.

Appendix A

List of Symbols

A_0	original single target cardinality model
A_1	multiple cardinality model
A_2	mean of intermediate S model
A_3	Markov chain with S as state
C	cardinality
d_c	single diagnosis of cardinality c
f_a	probability density function of cardinalities of the SAFARI algorithm outcome
f_s	cardinality distribution of the minimal solution space of the problem
k	current number of variables flipped
M	number of components
MC	minimal fault cardinality of the problem
R	number of retries (SAFARI algorithm parameter)
S	set of minimal solutions
s_i	number of hits of solutions of cardinality i