A Low-Cost Approximate Minimal Hitting Set Algorithm and Its Application to Model-Based Diagnosis*

Rui Abreu and **Arjan J. C. van Gemund**

Embedded Software Lab
Delft University of Technology
The Netherlands
{r.f.abreu,a.j.c.vangemund}@tudelft.nl

Abstract

Generating minimal hitting sets of a collection of sets is known to be NP-hard, necessitating heuristic approaches to handle large problems. In this paper a low-cost, approximate minimal hitting set (MHS) algorithm, coined STACCATO, is presented. STACCATO uses a heuristic function, borrowed from a lightweight, statistics-based software fault localization approach, to guide the MHS search. Given the nature of the heuristic function, STACCATO is specially tailored to model-based diagnosis problems (where each MHS solution is a diagnosis to the problem), although well-suited for other application domains as well. We apply STACCATO in the context of model-based diagnosis and show that even for small problems our approach is orders of magnitude faster than the brute-force approach, while still capturing all important solutions. Furthermore, due to its low cost complexity, we also show that STACCATO is amenable to large problems including millions of variables.

Introduction

Identifying minimal hitting sets (MHS) of a collection of sets is an important problem in many domains, such as in model-based diagnosis (MBD) where the MHS are the solutions for the diagnostic problem. Known to be a NP-hard problem (Garey and Johnson 1979), one (1) desires focusing heuristics to increase the search efficiency and/or (2) limits the size of the return set. Such strategies have the potential to reduce the MHS problem to a polynomial time complexity at the cost of completeness.

In this paper, we present an algorithm, coined STACCATO[1], to derive an approximate collection of MHS. STACCATO uses a heuristic borrowed from a statistics-based software fault diagnosis approach, called spectrum-based fault localization (SFL). SFL uses sets of component involvement in nominal and failing program executions to yield a ranking of components in order of likelihood to be at fault. We show that this ranking heuristic is suitable in the MBD domain as the search is focused by visiting solutions in best-first order (aiming to capture the most relevant probability mass in the shortest amount of time). Although the heuristic originates from the MBD domain, it is also useful in other domains. We also introduce a search pruning parameter λ and a search truncation parameter L. λ specifies the percentage of top components in the ranking that should be considered in the search for true MHS solutions. Taking advantage of the fact that most relevant solutions are visited first, the search can be truncated after L solutions are found, avoiding the generation of a myriad of solutions.

In particular, this paper makes the following contributions:

- We present a new algorithm STACCATO, and derive its time and space complexity;
- We compare STACCATO with a brute-force approach using synthetic data as well as data collected from a real software program;
- We investigate the impact of λ and L on STACCATO's cost/completeness trade-off.

To the best of knowledge this heuristic approach has not been presented before and has proven to have a significant effect on MBD complexity in practice (Abreu, Zoeteweij, and Van Gemund 2009).

The reminder of this paper is organized as follows. We start by introducing the MHS problem. Subsequently, STACCATO is outlined, and a derivation of its time/space complexity is given. The experimental results using synthetic data and data collected from a real software system are then presented, followed by a discussion of related work. Finally, we close this paper with some concluding remarks and directions for future work.

Minimal Hitting Set Problem

In this section we describe the minimal hitting set (MHS) problem, and the concepts used throughout this paper.

Let S be a collection of N non-empty sets $S = \{s_1, \ldots s_N\}$. Each set $s_i \in S$ is a finite set of elements (components from now on), where each of the M elements is represented by a number $j \in \{1, \ldots, M\}$. A minimal

*This work has been carried out as part of the TRADER project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the BSIK03021 program.

[1] STACCATO is an acronym for STAtistiCs-direCted minimAl hiTing set algOrithm.

hitting set of S is a set d such that

$$\forall s_i \in S, s_i \cap d \neq \emptyset \wedge \nexists d' \subset d : s_i \cap d' \neq \emptyset$$

i.e., there is at least a component of d that is member of all sets in S, and no proper subset of d is a hitting set. There may be several minimal hitting sets for S, which constitutes a collection of minimal hitting sets $D = \{d_1, \ldots, d_k, \ldots, d_{|D|}\}$. The computation of this collection D is known to be a NP-hard problem (Garey and Johnson 1979).

In the remainder of this paper, the collection of sets S is encoded into a $N \times M$ (binary) matrix A. An element a_{ij} is equal to 1 if component j is a member of set i, and 0 otherwise. For $j \leq M$, the row A_{i*} indicates whether a component is a member of set i, whereas the column A_{*j} indicates which sets component j is a member. As an example, consider the set $S = \{\{1,3\},\{2,3\}\}$ for $M = 3$, represented by the matrix

1	2	3	
1	0	1	first set
0	1	1	second set

A naïve, brute-force approach to compute the collection D of minimal hitting sets for S would be to iterate through all possible component combinations to (1) check whether it is a hitting set, and (2) and (if it is a hitting set) whether it is minimal, i.e., not subsumed by any other set of lower cardinality (cardinality of a set d_k, $|d_k|$, is the number of elements in the set). As all possible combinations are checked, the complexity of such an approach is $O(2^M)$. For the example above, the following sets would be checked: $\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}$ to find out that only $\{3\}$ and $\{1,2\}$ are minimal hitting sets of S.

STACCATO

As explained in the previous section, brute-force algorithms have a cost that is exponential in the number of components. Since many of the potential solution candidates turn out to be no minimal hitting set, a heuristic that *focuses* the search towards high-potentials will yield significant efficiency gains. In addition, many of the computed minimal hitting sets may potentially be of little value for the problem that is being solved. Therefore, one would like the solutions to be *ordered* in terms of relevance, possibly terminating the search once a particular number of minimal hitting sets have been found, again boosting efficiency. In this section we present our approximate, statistics-directed minimal hitting set algorithm, coined STACCATO, aimed to increase search efficiency.

The key idea behind our approach is the fact, that components that are members of more sets than other components, may be an indication that there is a minimal hitting set containing such component. The trivial case are those components that are *involved* in all sets, which constitute a minimal hitting set of cardinality 1. A simple search heuristic is to exploit a ranking based on the number of set involvements such as

$$\mathcal{H}(j) = \sum_{i=1}^{N} a_{ij}$$

To illustrate, consider again the example above. Using the heuristic function $\mathcal{H}(j)$, it follows that $\mathcal{H}(1) = 1$, $\mathcal{H}(2) = 1$, and $\mathcal{H}(3) = 2$, yielding the ranking $< 3, 1, 2 >$. This ranking is exploited to guide the search. Starting with component 3, it appears that it is involved in the two sets, and therefore is a minimal hitting set of minimal cardinality. Next in the ranking comes component 1. As it is not involved in all sets, it is combined with those components that are involved in all sets except the ones already covered by 1 (note, that combinations involving 3 are no longer considered due to subsumption). This would lead us to find $\{1,2\}$ as a second minimal hitting set.

Although this heuristic avoids having to iterate through all possible component combinations ($O(2^M)$), it may still be the case that many combinations have to be considered. For instance, using the heuristic one has to check 3 sets, whereas the brute-force approach iterates over 8 sets. Consequently, we introduce a search pruning parameter λ that contains the fraction of the ranking that will be considered. The reasoning behind this parameter is that the components that are involved in most sets (ranked high by $\mathcal{H}$) are more likely to be a minimal hitting set. Clearly, λ cannot be too small. In the previous example, if λ would be set to $\lambda = 1/3$, only element 3 would be considered, and therefore we would miss the solution set $\{1,2\}$. Hence, such a parameter trades efficiency for completeness.

Approximation

While the above heuristic increases search efficiency, the number of minimal hitting sets can be prohibitive, while often only a subset need be considered that are most relevant with respect to the application context. Typically, approaches to compute the minimal hitting set are applied in the context of (cost) optimization problems. In such case, one is often interested in finding the minimal hitting set of minimal cardinality. For example, suppose one is responsible for assigning courses to teachers. In particular, one proposes to minimize the number of teachers that need to be hired. Hence, one would like to find the minimal number of teachers that can teach all courses, which can be solved by formulating the problem as a minimal hitting set problem. For this example, solutions with low cardinality (i.e., number of teachers) are more attractive than those with higher cardinality. The brute-force approach, as well as the above heuristic approach are examples of approaches that find minimal hitting sets with lower cardinality first.

In many situations, however, obtaining MHS solutions in order of just cardinality does not suffice. An example is model-based diagnosis (MBD) where the minimal hitting sets represent fault diagnosis candidates, each of which has a certain probability of being the actual diagnosis. The most cost-efficient approach is to generate the MHS solutions in decreasing order of probability (minimizing average fault localization cost). Although probability typically decreases with increasing MHS cardinality, cardinality is not sufficient, as, e.g., there may be a significant probability difference between diagnosis (MHS) solutions of equal cardinality (of which there may be many). Consequently, a heuristic that predicts probability rather than just cardinality makes

the difference. The fact that the MHS solutions are now generated in decreasing order of probability allows us to truncate the number of solutions, where, e.g., one only considers the MHS subset of L solutions that covers .99 probability mass, ignoring the (many) improbable solutions. This *approximation* trades limited cost penalty (completeness) for significant efficiency gains.

Model-Based Diagnosis

In this section we extend our above heuristic for use in MBD. In the context of MBD a diagnosis is derived by computing the MHS of a set of so-called conflicts (de Kleer and Williams 1987). A conflict would be a sequence of probable faulty components that explain the observed failure (this set explains the differences between the model and the observation). For instance, in a logic circuit a conflict may be the sub-circuit (cone) activity that results in an output failure. In software a conflict is the sequence of software component activity (e.g., statements) that results in a particular faulty return value (Abreu, Zoeteweij, and Gemund 2008).

In MBD the MHS solutions d_k are ranked in order of probability of being the true fault explanation $\Pr(d_k)$, which is computed using Bayes' update according to

$$\Pr(d_k|obs_i) = \frac{\Pr(obs_i|d_k)}{\Pr(obs_i)} \cdot \Pr(d_k|obs_{i-1}) \qquad (1)$$

where obs_i denotes observation i. In the context of the presentation in this paper, an observation obs_i stands for a conflict set s_i that results from a particular observation. The denominator $\Pr(obs_i)$ is a normalizing term that is identical for all d_k and thus needs not be computed directly. $\Pr(d_k|obs_{i-1})$ is the prior probability of d_k, before incorporating the new evidence obs_i. For $i = 1$ $\Pr(d_k)$ is typically defined in a way such that it ranks solutions (diagnosis candidates) of lower cardinality higher in absence of any observation. $\Pr(obs_i|d_k)$ is defined as

$$\Pr(obs_i|d_k) = \begin{cases} 0 & \text{if } obs_i \wedge d_k \models \bot \\ 1 & \text{if } d_k \rightarrow obs_i \\ \varepsilon & \text{otherwise} \end{cases}$$

In MBD, many policies exist for ε based on the chosen modeling strategy. See (Abreu, Zoeteweij, and Gemund 2008; de Kleer 2006; 2007) for details.

Given a sequence of observations (conflicts), the MHS solutions should be ordered in terms of Eq. (1). However, using Eq. (1) as heuristic is computationally prohibitive. Clearly, a low-cost heuristic that still provides a good prediction of Eq. (1) is crucial if STACCATO is to be useful in MBD.

An MBD Heuristic

A low-cost, statistics-based technique that is known to be a good predictor for ranking (software) faults in order of likelihood is spectrum-based fault localization (SFL) (Abreu, Zoeteweij, and Van Gemund 2007). SFL takes the set of conflicts S (corresponding to erroneous system behavior) as well as information collected during nominal system behavior (again, set of components involved), and produces a ranking the components in order of fault likelihood. The component ranking is computed using a statistical similarity coefficient that measures the statistical correlation between component involvement and erroneous/nominal system behavior. To comply with SFL, we extend A into a pair (A, e) (see Figure 1), where e is a binary array which indicates whether the A_{i*} corresponds to erroneous system behavior ($e = 1$) or nominal behavior ($e = 0$).

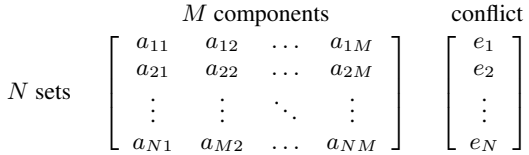

Figure 1: Encoding for a collection of sets

Many similarity coefficients exist for SFL, the best one currently being the Ochiai coefficient known from molecular biology and introduced to SFL in (Abreu, Zoeteweij, and Van Gemund 2007). It is defined as follows

$$s(j) = \frac{n_{11}(j)}{\sqrt{(n_{11}(j) + n_{01}(j)) * (n_{11}(j) + n_{10}(j))}} \qquad (2)$$

where

$$n_{pq}(j) = |\{i \mid a_{ij} = p \wedge e_i = q\}|$$

A similarity coefficient indicts components using $n_{11}(j)$, and exonerates components using $n_{10}(j)$ and $n_{01}(j)$. In (Abreu, Zoeteweij, and Van Gemund 2007) it has been shown that similarity coefficients provide an ordering of components that yields good diagnostic accuracy, i.e., components that rank high are usually faulty. This diagnostic performance, combined with the very low complexity of $s(j)$ is the key motivation to use the Ochiai coefficient $s(j)$ for $\mathcal{H}$. If (A, e) only contains conflicts (i.e., $\nexists e_i = 0$), the ranking returned by this heuristic function reduces to the original one

$$\mathcal{H}(j) = \sum_{i=1}^{N} a_{ij} = n_{11}(j)$$

and, therefore, classic MHS problems are also adequately handled by this MBD heuristic.

Algorithm

STACCATO uses the SFL heuristic Eq. (2) to focus the search of the minimal hitting set computation (see Algorithm 1). To illustrate how STACCATO works, consider the following (A, e), comprising two (conflict) sets originating from erroneous system behavior and one set corresponding to component involvement in nominal system behavior.

1	2	3	e_i	
1	0	1	1	first set (error)
0	1	1	1	second set (error)
1	0	1	0	third set (nominal)

From (A, e) it follows $\mathcal{H}(1) = 0.5$, $\mathcal{H}(2) = 0.7$, and $\mathcal{H}(3) = 1$, yielding the following ranking $< 3, 2, 1 >$.

As component 3 is involved in all failed sets, it is added to the minimal hitting set and removed from A using function STRIP_COMPONENT, avoiding solutions subsumed by $\{3\}$ to be considered (lines 5–12). After this phase, the (A, e) is as follows

1	2	e_i
1	0	1
0	1	1
1	0	0

Next component to be checked is component 2, which is not involved in one failed set. Thus, the column for that component as well as all conflict sets in which it is involved are removed from (A, e), using the STRIP function, yielding the following

1	e_i
1	1
1	0

Running STACCATO with the newly generated (A, e) yields a ranking with component 1 only (line 15 – 18), which is a MHS for the current (A, e). For each MHS d returned by this invocation of STACCATO, the union of d and component 2 is checked ($\{1, 2\}$), and because this set is involved in all failed sets, and is minimal, it is also added to the list of solutions D (lines 18–24). The same would be done for component 1, the last in the ranking, but no minimal set would be found. Thus, STACCATO would return the following minimal hitting sets $\{\{3\}, \{1, 2\}\}$. Note that this heuristic ranks component 2 on top of component 1, whereas the previous heuristic ranked component 1 and 2 at the same place (because they both explained the same number of conflicts).

In summary, STACCATO comprises the following steps

- Initialization phase, where a ranking of components using the heuristic function borrowed from SFL is computed (lines 1–4 in Algorithm 1);
- Components that are involved in all failed sets are added to D (lines 5–12);
- While $|D| < L$, for the first top λ components in the ranking (including also the ones added to D, lines 13-25) do the following: (1) remove the component j and all A_{i*} for which $e_i = 1 \wedge a_{ij} = 1$ holds from (A, e) (line 17), (2) run STACCATO with the new (A, e), and (3) combine the solutions returned with the component and verify whether it is a minimal hitting set (lines 17–24).

Complexity Analysis

To find a minimal hitting set of cardinality C STACCATO has to be (recursively) invoked C times. Each time it (1) updates the four counters per component ($O(N \cdot M)$), (2) ranks components in fault likelihood ($O(M \cdot \log M)$), (3) traverse λ components in the ranking ($O(M)$), and (4) check whether it is a minimal hitting set ($O(N)$). Hence, the overall time complexity of STACCATO is merely $O((M \cdot (N + \log M))^C)$. In practice, however, due to the search focusing heuristic the time complexity is merely $O(C \cdot M \cdot (N + \log M))$ (confirmed by measurements in Section).

Algorithm 1 STACCATO

Inputs: Matrix (A, e), number of components M, stop criteria λ, L

Output: Minimal Hitting set D

```
1  T_F ← {A_i* | e_i = 1}          ▷ Collection of conflict sets
2  R ← RANK(H, A, e)
3  D ← ∅
4  seen ← 0
5  for all j ∈ {1..M} do
6      if n_11(j) = |T_F| then
7          PUSH(D, {j})
8          A ← STRIP_COMPONENT(A, j)
9          R ← R\{j}
10         seen ← seen + 1/M
11     end if
12 end for
13 while R ≠ ∅ ∧ seen ≤ λ ∧ |D| ≤ L do
14     j ← POP(R)
15     seen ← seen + 1/M
16     (A', e') ← STRIP(A, e, j)
17     D' ← STACCATO (A', e', λ)
18     while D' ≠ ∅ do
19         j' ← POP(D')
20         j' ← {c} ∪ j'
21         if IS_NOT_SUBSUMED(D, j') then
22             PUSH(D, j')
23         end if
24     end while
25 end while
26 return D
```

With respect to space complexity, for each invocation of STACCATO, it has to store four counters per component to create the SFL-based ranking $(n_{11}, n_{10}, n_{01}, n_{00})$. As the recursion depth is C to find a solution of the same cardinality, STACCATO has a space complexity of $O(C \cdot M)$.

Experimental Results

In this section we present our experimental results using synthetic data and data collected from a real software program.

Synthetic Diagnosis Experiments

In order to assess the performance of our algorithm we use synthetic sets generated for diagnostic algorithm research, based on random (A, e) generated for various values of N, M, and the number of injected faults C (cardinality). Component activity a_{ij} is sampled from a Bernoulli distribution with parameter r, i.e., the probability a component is involved in a row of A equals r. For the C faulty components c_j (without loss of generality we select the first C components, i.e., $c_1, \ldots, c_C$ are faulty). We also set the probability a faulty component behaves as expected h_j. Thus, the probability of a component j being involved *and* generating a failure equals $r \cdot (1 - h_j)$. A row i in A generates an error ($e_i = 1$) if at least 1 of the C components generates a failure (or-model). Measurements for a specific scenario are

			h	0.1		0.9	
			C	1	5	1	5
B-F			$\|D\|$	355	508	115	286
			T (s)	25.5	54.3	0.27	5.72
			W (%)	0.0	13	14	21
STACCATO	$\lambda = 0.1$		$\|D\|$	63	127	10	46
			T (s)	0.006	0.007	0.001	0.003
			ρ (%)	0/0/0/65/87/0	0/0/41/95/60/82	0/44/100/0/0/0	0/0/42/88/0/0
			W (%)	0.0	10.7	0.0	12.9
	$\lambda = 0.2$		$\|D\|$	86	181	16	63
			T (s)	0.008	0.009	0.02	0.003
			ρ (%)	0/0/0/54/87/0	0/0/30/87/59/70	0/31/100/0/0/0	0/0/36/88/0/0
			W (%)	0.0	9.2	0.0	13.9
	$\lambda = 0.3$		$\|D\|$	112	232	26	75
			T (s)	0.009	0.010	0.003	0.004
			ρ (%)	0/0/0/30/74/0	0/0/21/87/53/65	0/25/73/0/0/0	0/0/26/75/0/0
			W (%)	0.0	9.1	0.0	14.4
	$\lambda = 0.4$		$\|D\|$	175	276	47	83
			T (s)	0.011	0.012	0.004	0.004
			ρ (%)	0/0/0/26/75/0	0/0/21/87/18/64	0/6/72/0/0/0	0/0/10/63/0/0
			W (%)	0.0	9.2	0.0	13.8
	$\lambda = 0.5$		$\|D\|$	218	300	67	146
			T (s)	0.013	0.018	0.004	0.007
			ρ (%)	0/0/0/23/66/0	0/0/10/87/6/65	0/0/64/0/0/0	0/0/5/56/0/0
			W (%)	0.0	9.0	0.0	14.3
	$\lambda = 0.6$		$\|D\|$	253	372	83	180
			T (s)	0.015	0.019	0.004	0.008
			ρ (%)	0/0/0/20/61/0	0/0/0.08/65/0/65	0/0/39/0/0/0	0/0/0/46/0/0
			W (%)	0.0	8.8	0.0	14.7
	$\lambda = 0.7$		$\|D\|$	293	425	87	199
			T (s)	0.019	0.025	0.005	0.008
			ρ (%)	0/0/0/11/50/0	0/0/0.06/54/0/55	0/0/39/0/0/0	0/0/0/44/0/0
			W (%)	0.0	8.6	0.0	14.4
	$\lambda = 0.8$		$\|D\|$	343	449	109	228
			T (s)	0.023	0.028	0.06	0.009
			ρ (%)	0/0/0/7/26/0	0/0/0.02/38/0/24	0/0/24/0/0/0	0/0/0/32/0/0
			W (%)	0.0	8.8	0.0	14.8
	$\lambda = 0.9$		$\|D\|$	355	508	115	270
			T (s)	0.024	0.034	0.08	0.012
			ρ (%)	0/0/0/0/0/0	0/0/0/15/0/10	0/0/0/0/0/0	0/0/0/13/0/0
			W (%)	0.0	13	14	21
	$\lambda = 1$		$\|D\|$	355	508	115	286
			T (s)	0.025	0.041	0.010	0.016
			ρ (%)	0/0/0/0/0/0	0/0/0/0/0/0	0/0/0/0/0/0	0/0/0/0/0/0
			W (%)	0.0	13	14	21

Table 1: Results for the synthetic matrices

averaged over 500 sample matrices.

Table 1 summarizes the results of our study for $r = 0.6$ (typical value for software), $M = 20$ and $N = 300$, which is the limit for which a brute-force approach is feasible. Per scenario, we measure the number of MHS solutions ($|D|$), the computation CPU time (T, measured on a 2.3 GHz Intel Pentium-6 PC with 4 GB of memory), the miss ratio ρ per C (the ratio of solutions with cardinality C found using STACCATO and the brute-force approach, and the diagnostic performance (W) for the brute-force approach (B-F) and STACCATO with several λ parameter values. The miss ratio ρ values presented are per cardinality - separated by a '/' - where the last value is the percentage of solutions missed with $C \geq 6$. Diagnostic performance is measured in terms of a diagnostic performance metric W that measures the percentage of excess *work* incurred in finding the actual components at fault, a typical metric in software debugging (Abreu, Zoeteweij, and Van Gemund 2007), after ranking the MHS solutions using the Bayesian policy described in (Abreu, Zoeteweij, and Gemund 2008). For in-

stance, consider a $M = 5$ component program with the following diagnostic report $D =< \{4, 5\}, \{1, 2\} >$, while components 1 and 2 are actually faulty. The first diagnosis candidate leads the developer to inspect components 4 and 5. As both components are healthy, W is increased with $\frac{2}{5}$. The next components to be inspected are components 1 and 2. As they are both faulty, no more wasted effort is incurred. After repairing these two components, the program would be re-run to verify that all test cases pass. Otherwise, the debugging process would start again until no more test cases fail.

As expected, $|D|$ and the time needed to compute D decrease with λ. Although some solutions are missed for low values of λ, they are not important for the diagnostic problem as W does not increase. This suggests that our heuristic function captures the most probable solutions to be faulty. An important observation is that for $\lambda = 1$, the results are essentially the same as an exhaustive search but with several orders of magnitude speed-up. Furthermore, we also truncated $|D|$ to 100 to investigate the impact of this parameter in the diagnostic accuracy for $\lambda = 1$. Although it has a small negative impact on W, it reduces the time needed to compute W by more than half. For instance, for $C = 5$ and $h = 0.1$ it takes 0.008 s to generate D, requiring the developer to waste more effort to find the faulty components, $W = 10\%$.

We have not presented results for other settings of M, N because the brute-force approach does not scale. However, as an example, for $M = 1,000,000$, $N = 1,000$, and $C = 1,000$, the candidate generation time rate with STACCATO is still only 88.6 ms on average (22.1 ms for $C = 100$).

Results with matrices containing only conflict sets show that the performance of STACCATO is similar to the one just reported, meaning that the general, classic MHS problems are properly handled by our algorithm.

			tcas		
	C		1	2	5
	#matrices		41	100	100
B-F	$\|D\|$		76	59	68
	T (s)		0.98	2.1	11.2
	W (%)		16.7	23.7	29.7
STACCATO	$\lambda = 0.1$	$\|D\|$	30	35	61
		T (s)	0.11	0.16	0.22
		W	15.2	29.3	37.1
	$\lambda = 0.2$	$\|D\|$	34	39	62
		T (s)	0.15	0.17	0.25
		W	15.2	29.3	37.0
	$\lambda = 0.3$	$\|D\|$	50	44	63
		T (s)	0.18	0.18	0.26
		W	16.2	28.8	37.1
	$\lambda = 0.4$	$\|D\|$	51	58	65
		T (s)	0.19	0.19	0.27
		W	16.3	23.7	32.1
	$\lambda = 0.5$	$\|D\|$	76	58	66
		T (s)	0.20	0.20	0.30
		W	16.7	23.7	30.1
	$\lambda = 0.6$	$\|D\|$	76	59	67
		T (s)	0.22	0.22	0.31
		W	16.7	23.7	29.7
	$\lambda = 0.7$	$\|D\|$	76	59	68
		T (s)	0.23	0.25	0.34
		W	16.7	23.7	29.7
	$\lambda = 0.8$	$\|D\|$	76	59	68
		T (s)	0.24	0.26	0.35
		W	16.7	23.7	29.7
	$\lambda = 0.9$	$\|D\|$	76	59	68
		T (s)	0.27	0.27	0.37
		W	16.7	23.7	29.7
	$\lambda = 1$	$\|D\|$	76	59	68
		T (s)	0.30	0.28	0.48
		W	16.7	23.7	29.7

Table 2: Results for `tcas`

Real Software Diagnosis

In this section we apply the STACCATO algorithm in the context of model-based software fault diagnosis, namely to derive the set of valid diagnoses given a set of observations (test cases). We use the `tcas` program which can be obtained from the software infrastructure repository (SIR, (Do, Elbaum, and Rothermel 2005)). `TCAS` (Traffic Alert and Collision Avoidance System) is an aircraft conflict detection and resolution system used by all US commercial aircraft. The SIR version of `tcas` includes 41 faulty versions of ANSI-C code for the resolution advisory component of the TCAS system. In addition, it also provides a correct version of the program and a pool containing $N = 1,608$ test cases. `tcas` has $M = 178$ lines of code, which, in the context of the following experiments, are the number of components. In our experiments, we randomly injected C faults in one program. All measurements are averages over 100 versions, except for the single fault programs which are averages over the 41 available faults. The activity matrices are obtained using the GNU `gcov`[2] profiling tool and a script to translate its output into a matrix. As each program suite includes a correct version, we use the output of the correct version as reference. We characterize a run/computation as failed if its output differs from the corresponding output of the correct version, and as passed otherwise.

Table 2 presents a summary of the results obtained using a brute-force approach (B-F) and STACCATO with different λ parameter values. Again, we report the size of the minimal hitting set ($|D|$), the time T required to generate D, and the diagnostic performance incurred by the different settings. As expected, the brute-force approach is the most expensive of them all. The best trade-off between complexity and the diagnostic cost W is for $\lambda \approx 0.5$, since STACCATO does not miss important candidates - judging by the fact that W is essentially the same as the brute-force approach - and it is faster than for other, higher λ.

Although STACCATO was applied to other, bigger software programs (Abreu, Zoeteweij, and Van Gemund 2009), no comparison is given as the brute-force algorithm does not scale. As an indication, for a given program, `space` (Do, Elbaum, and Rothermel 2005), with $M = 9,564$ lines of code and $N = 132$ test cases, STACCATO required roughly 1 s to compute the relevant MHS solutions (for $\lambda = 0.5$

[2]http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

and $L = 100$). In these experiments, using the well-known Siemens benchmark set of software faults and the `space` program, $L = 100$ was proven to already yield comparable results to those obtained for $L = \infty$ (i.e., generating all solutions).

Related Work

Several exhaustive algorithms have been presented to solve the MHS problem. Since Reiter (1987) showed that diagnoses are MHSs of conflict sets, many approaches to solve this problem in the model-based diagnosis context have been presented. In (Greiner, Smith, and Wilkerson 1989; de Kleer and Williams 1987; Reiter 1987; Wotawa 2001) the hitting set problem is solved using so-called hit-set trees. In (Fijany and Vatan 2004; 2005) the MHS problem is mapped onto an 1/0-integer programming problem. Contrary to our work does not use any other information but the conflict sets. The integer programming approach has the potential so solve problems with thousands of variables but no complexity results are presented. In contrast, our low-cost approach can easily handle much larger problems. In (Zhao and Ouyang 2007) a method using set-enumeration trees to derive all minimal conflict sets in the context of model-based diagnosis is presented. The authors just conclude that this method has an exponential time complexity in the number of elements in the sets (components). The Quine-McCluskey algorithm (Quine 1955; Mccluskey 1956), originating from logic optimization, is a method for deriving the prime implicants of a monotone boolean function (a dual problem of the MHS problem). This algorithm is, however, of limited use due to its exponential complexity, which has prompted the development of heuristics such as Espresso (discussed later on).

Many heuristic approaches have been proposed to render MHS computation amenable to large systems. In (Lin and Jiang 2002) an approximate method to compute MHSs using genetic algorithms is described. The fitness function used aims at finding solutions of minimal cardinality, which is not always sufficient for MBD as even solutions with similar cardinality have different probabilities of being the true fault explanation. Their paper does not present a time complexity analysis, but we suspect the cost/completeness trade-off to be worse than for STACCATO. Stochastic algorithms, as discussed in the framework of constraint satisfaction (Freuder et al. 1995) and propositional satisfiability (Qasem and Prügel-Bennett 2008), are examples of domain independent approaches to compute MHS. Stochastic algorithms are more efficient than exhaustive methods. The Espresso algorithm (Brayton et al. 1984), primarily used to minimize logic circuits, uses a heuristics to guide the circuit minimization that is inspired by this domain. Originating from logic circuits, it uses a heuristic to guide the circuit minimization that is specific for this domain. Due to its efficiency, this algorithm still forms the basis of every logic synthesis tool. Dual to the MHS problem, no prime implicants cost/completeness data is available to allow comparison with STACCATO.

To our knowledge the statistics-based heuristic to guide the search for computing MHS solutions has not been presented before. Although the heuristic function used in our approach comes from a fault diagnosis approach, there is no reason to believe that STACCATO will not work well in other domains.

Conclusions and Future Work

In this paper we presented a low-cost approximate hitting set algorithm, coined STACCATO, which uses a heuristic borrowed from a low-cost, statistics fault diagnosis tool, making it especially suitable to the model-based diagnosis domain. Moreover, the very low time/space complexity of the algorithm allows dealing with large-size problems with millions of variables.

Our experiments have demonstrated that even for small problems our heuristic approach is orders of magnitude faster than exhaustive approaches, even when the algorithm is set to be complete ($\lambda = 1$). Furthermore, the experiments have shown that the search can be further focused using λ, where completeness is hardly sacrificed for $\lambda \approx 0.5$ (i.e., reduce search space). Compared to λ, the potential impact of truncating the number of solutions L in the set on cost is much greater. As most relevant solutions are visited first, the number of solutions returned to the user can be suitably truncated (e.g., only returning 100 candidates in the context of our model-based diagnosis experiments). Hence, a very attractive cost/completeness trade-off is reached by setting $\lambda = 1$ while limiting L.

Future work includes extending the parameter range for our experiments (e.g., investigate the impact of truncating the number of returned solutions). Furthermore, we plan to quantitatively compare the efficiency of STACCATO to other approaches to compute minimal hitting sets. In particular, we intend to compare the efficiency of our approach to SAT approaches, which also handle problems with millions of variables.

Acknowledgments

We gratefully acknowledge the fruitful discussions with our TRADER project partners from NXP Research, NXP Semiconductors, Philips TASS, Philips Consumer Electronics, Embedded Systems Institute, Design Technology Institute, IMEC, Leiden University, and Twente University.

References

Abreu, R.; Zoeteweij, P.; and Gemund, A. J. C. V. 2008. A dynamic modeling approach to software multiple-fault localization. In *Proceedings of the Internation Workshop on Principles of Diagnosis (DX'08)*.

Abreu, R.; Zoeteweij, P.; and Van Gemund, A. J. C. 2007. On the accuracy of spectrum-based fault localization. In *Proceedings of the Testing: Academia and Industry Conference - Practice And Research Techniques (TAIC PART'07)*.

Abreu, R.; Zoeteweij, P.; and Van Gemund, A. J. C. 2009. A new Bayesian approach to multiple intermittent fault diagnosis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'09)*. To appear.

Brayton, R. K.; Sangiovanni-Vincentelli, A. L.; McMullen, C. T.; and Hachtel, G. D. 1984. *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers.

de Kleer, J., and Williams, B. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130.

de Kleer, J. 2006. Getting the probabilities right for measurement selection. In *Proceedings of the Internation Workshop on Principles of Diagnosis (DX'06)*.

de Kleer, J. 2007. Diagnosing intermittent faults. In *Proceedings of the Internation Workshop on Principles of Diagnosis (DX'07)*.

Do, H.; Elbaum, S. G.; and Rothermel, G. 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering: An International Journal* 10(4):405–435.

Fijany, A., and Vatan, F. 2004. New approaches for efficient solution of hitting set problem. In *Proceedings of the winter international synposium on Information and communication technologies (WISICT'04)*. Cancun, Mexico: Trinity College Dublin.

Fijany, A., and Vatan, F. 2005. New high performance algorithmic solution for diagnosis problem. In *Proceedings of the IEEE Aerospace Conference (IEEEAC'05)*.

Freuder, E. C.; Dechter, R.; Ginsberg, M. L.; Selman, B.; and Tsang, E. P. K. 1995. Systematic versus stochastic constraint satisfaction. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*, 2027–2032.

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

Greiner, R.; Smith, B. A.; and Wilkerson, R. W. 1989. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence* 41(1):79–88.

Lin, L., and Jiang, Y. 2002. Computing minimal hitting sets with genetic algorithms. In *Proceedings of the Internation Workshop on Principles of Diagnosis (DX'02)*.

Mccluskey, E. J. 1956. Minimization of boolean functions. *The Bell System Technical Journal* 35(5):1417–1444.

Qasem, M., and Prügel-Bennett, A. 2008. Complexity of max-sat using stochastic algorithms. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'08)*, 615–616.

Quine, W. 1955. A way to simplify truth functions. *Amer.Math.Monthly* 62:627 – 631.

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

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

Zhao, X., and Ouyang, D. 2007. Improved algorithms for deriving all minimal conflict sets in model-based diagnosis. In Huang, D.-S.; Heutte, L.; and Loog, M., eds., *Proceedings of the 3rd International Conference on Intelligent Computing (ICIC'07)*, volume 4681 of *Lecture Notes in Computer Science*, 157–166. Springer.