A Probabilistic Learning Approach for Counterexample Guided Abstraction Refinement [⋆]

Fei He[1,2], Xiaoyu Song[3], Ming Gu[2], and Jiaguang Sun[2]

[1] Dept. Computer Science & Technology, Tsinghua University, Beijing, China
hef02@mails.tsinghua.edu.cn
[2] School of Software, Tsinghua University, Beijing, China
[3] Dept. ECE, Portland State University, Oregon, USA

Abstract. The paper presents a novel probabilistic learning approach to state separation problem which occurs in the counterexample guided abstraction refinement. The method is based on the sample learning technique, evolutionary algorithm and effective probabilistic heuristics. Compared with the previous work by the sampling decision tree learning solver, the proposed method outperforms 2 to 4 orders of magnitude faster and the size of the separation set is 76% smaller on average.

1 Introduction

Abstraction is one of the most important techniques when applying model checking to large scale systems. The hypostasis of abstraction is to eliminate the irrelevant information to reduce the system model. The counterexample-guided abstraction refinement (CEGAR) [1] is an effective strategy in application of abstraction. In CEGAR, the verification is performed in an abstract-check-refine fashion, and the refinement is guided by counterexamples. The counterexample contains the critical clues about the cause of the violation. If there exists a real path in the concrete model that simulates the counterexample, one can find a real bug, otherwise the counterexample is spurious and one has to refine the abstract model to eliminate such a spurious path.

Many counterexample-guided abstraction refinement strategies have been proposed [2–7]. Some recent methods on automatic abstraction [6, 8–10] employ the unsatisfiable core saved in the SAT solver, and the abstraction is based on the proofs provided by the SAT solver, but not on refuting the counterexamples. In [3], the abstraction is performed by making a set of state variables invisible. If the counterexample is spurious, we need to refine the abstract model. State separation problem poses the main hurdle during the refinement.

In this paper, we propose a novel probabilistic learning approach to state separation problem (SSP) which occurs in the abstraction refinement. Our approach incorporates sample learning technique, evolutionary algorithm and effective heuristics in a synergistic way. Experimental results demonstrate the

[⋆] This work was supported in part by the Chinese National 973 Plan under grant No. 2004CB719406 and NSF of China under grant No. 60553002.

promising performance of our approach. In comparison with the previous work by the sampling decision tree learning solver [3], the proposed method outperforms 2 to 4 orders of magnitude faster and the size of the separation set is 76% smaller on average.

The remainder of the paper is organized as follows. In Section 2, we introduce some preliminaries. In Section 3, we formally define the problem. In Section 4, we present our probabilistic learning approach. The experimental results are reported in Section 5. Finally, Section 6 concludes the paper.

2 Preliminaries

We use state transition systems to model systems. Given a non-empty set of atomic propositions AP, let $M = \langle S, S_0, R, L\rangle$ be a transition system where

- S is the set of states.
- $S_0 \subseteq S$ is the set of initial states.
- $R \subseteq S \times S$ is the transition relation.
- $L : S \rightarrow 2^{AP}$ is the labeling function.

Let $V = \{v_1, v_2, \dots v_{|V|}\}$ be the universal domain of system variables. We assume that the variables in V range over a finite set D. A valuation for V corresponds to a state in S.

As in [3], we think of V as two parts: the set of *visible* variables (denoted as V_S) and the set of *invisible* variables (denoted as V_N). Invisible variables are those that we will ignore when build the model. For example, consider a digital system with latches. The subset of the latches in which we are interested is considered as visible variables, while the remaining latches are regarded as invisible.

In the original (non-abstracted) model, all system variables are visible. The abstraction process is essentially equivalent to selecting and setting some of the visible variables as invisible. Oppositely, the refinement process is to make some of the invisible variables as visible.

Let M be the original model. We use $\tilde{M} = \langle \tilde{S}, \tilde{S_0}, \tilde{R}, \tilde{L}\rangle$ to denote the abstract model, where the definitions of $\tilde{S}$, $\tilde{S_0}$ $\tilde{R}$ and $\tilde{L}$ follow those in M.

Notice that we require our abstraction to be conservative, that is: $\tilde{R}(\tilde{s_1}, \tilde{s_2})$ holds if and only if there exist s_1 in $h^{-1}(\tilde{s_1})$ and s_2 in $h^{-1}(\tilde{s_2})$, such that $R(s_1, s_2)$ holds, where h is the abstract function from S to $\tilde{S}$. Such a conservative translation may introduce additional behaviors into the abstract model. Consider the example shown in Fig. 1, after mapping the concrete states 7, 8, 9 to III, and 10 to IV, respectively, the additional transitions $7 \rightarrow 10$, $8 \rightarrow 10$ are added implicitly to the abstract model.

Given an abstract path $\tilde{P} = \langle \tilde{s_1}, \tilde{s_2}, \dots \tilde{s_m}\rangle$ in $\tilde{M}$ and a concrete path $P = \langle s_1, s_2, \dots s_m\rangle$ in M, we define the simulation relation $\sim$ as follows:

$$P \sim \tilde{P} \iff s_1 \in S_0 \text{ and } s_1 \in h^{-1}(\tilde{s_1}), s_2 \in h^{-1}(\tilde{s_2}), \dots s_m \in h^{-1}(\tilde{s_m}). \quad (1)$$

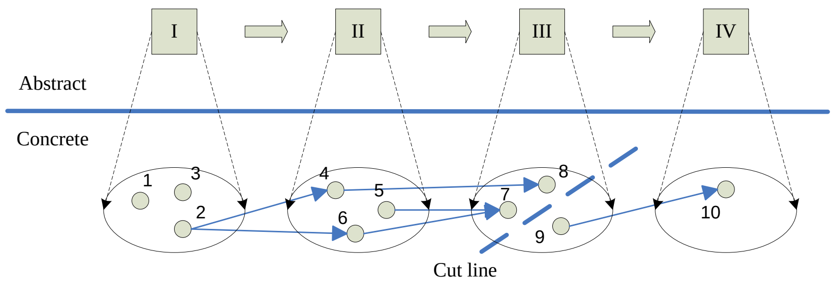

Fig. 1. A spurious counterexample

In the counterexample-guided approach, if we find a counterexample $\tilde{P}$ in the abstract model, we check if there is a concrete path P in M such that $P \sim \tilde{P}$. If it is true, we find a real bug. Otherwise, the counterexample is spurious. In the case of the spurious counterexample, we need to compute the *failure index* i_F, i.e. the maximal index i_F, $i_F < m$, such that there exists a concrete path in M which simulates the i_F prefix of $\tilde{P}$. With the failure index, we define the *failure states* to be the group of concrete states $F = \{s | s \in h^{-1}(\tilde{s_{i_F}})\}$ in M. Consider the example in Fig. 1, the failure index is III, and the failure states are 7, 8 and 9.

The failure states can be partitioned into three sets.

1. the set of deadend states F_d: $s \in F_d$ if and only if
   - $s \in F$;
   - there exists a concrete path to s which simulates the i_F prefix of $\tilde{P}$.
2. the set of bad states F_b: $s \in F_b$ if and only if
   - $s \in F$;
   - there exists no concrete path to s which simulates the i_F prefix of $\tilde{P}$;
   - there exists a transition from s to some states in $h^{-1}(s_{i_F+1})$.
3. $F - F_d - F_b$.

3 State Separation Problem

The State Separation Problem (SSP) [3] is to find a subset Λ of the invisible variables in V_N such that

$$\forall s_i \in F_d, \forall t_j \in F_b, \exists v_r \in \Lambda, s_i(v_r) \neq t_j(v_r). \tag{2}$$

The set Λ is named as *separation set*. We usually want the separation set to be as minimal as possible so that the corresponding refined model is minimal. This problem is known as the *minimal state separation problem* (MSSP).

Consider the abstract counterexample $\hat{P} = \langle \text{I, II, III, IV} \rangle$ shown in Fig. 1. It is spurious since there is no corresponding path in the concrete model. For this instance, the failure states are 7, 8 and 9. To eliminate the counterexample, we need to make some variables visible to distinguish the sets of states $\{7, 8\}$ and $\{9\}$.

In realistic systems, the size of failure states is usually very large. Moreover, since the state separation problem is embedded in the abstract-check-refine iteration, each time a spurious counterexample is found, a solution to the SSP needs to be provided. Thus, there is a strong demand for the effectiveness of SSP solvers in terms of time and memory.

In [3], an integer linear programming (ILP) model for the minimal state separation problem (MSSP) has been presented, and both an ILP solver and a decision tree learning (DTL) solver are employed for solving this problem. The general ILP solver attempts to enumerate the solution space to find the optimal solution for the state separation problem. However, since the minimal state separation problem is NP-hard, it is infeasible for the ILP solver to find the solution when the problem size is large. Note that we do not necessarily need the solution to be minimal. An approximate optimum may still good enough for the refinement process, nevertheless, the resulting refined model may be slightly bigger. In [3], an improved solver was proposed, which is based on the decision tree learning. The DTL algorithm trains the decision tree based on the input examples. It utilizes the well-trained decision tree to classify data. With some adjustments on the parameters, the DTL algorithm is used to solve the state separation problem, and the structure of its decision tree just gives a possible solution. Obviously, the DTL approach is an approximate method. Its solution precision relies on the number of input examples. If there are a sufficient number of examples, the solution could be guaranteed. However, if the input examples are too many, the time cost is extremely high. Thus, there is a trade-off between the solution precision and the solving cost. Furthermore, in coping with the large problem size, an efficient sampling technique has been applied to the DTL solver. Experimental results show that DTL solver with efficient sampling technique (for short, SDTL) outperforms the ordinary DTL solver [2].

3.1 Problem Formulation

In this section, we first prove the NP-hardness of MSSP by reducing it to the set covering problem. Then we present a new mathematical model for MSSP.

Definition 1. *Given a pair of states $\langle s, t\rangle$, $s \in F_d$, $t \in F_b$, if there exists a variable v, such that $s(v) \neq t(v)$, we say that the state pair $\langle s, t\rangle$ is* covered *by the variable v.*

Proposition 1. *The MSSP is reducible to the set covering problem.*

Proof. Consider $F_d \times F_b$ as the universal set. Obviously, each variable in V_N covers a subset of elements in $F_d \times F_b$, i.e. each variable corresponds to a subset of $F_d \times F_b$. Then according to the definition, the MSSP is essentially to find a minimal collection of subsets of $F_d \times F_b$ such to cover all the elements in $F_d \times F_b$. Obviously, it is a set covering problem. □

Given a MSSP instance, assume there are n invisible variables and m state pairs. For simplicity, we use $p_j, 1 \leq j \leq m$, to denote a state pair in $F_d \times F_b$, i.e. $F_d \times F_b = \{p_1, p_2, \ldots, p_m\}$. We define the decision variables as follows:

$$x_i = \begin{cases} 1, \text{ if } v_i \in \Lambda, \\ 0, \text{ else.} \end{cases}$$

Assume $p_j = \langle s^{p_j}, t^{p_j} \rangle$, where

$$s^{p_j} = \langle s_1^{p_j}, s_2^{p_j}, \ldots, s_n^{p_j} \rangle$$

and

$$t^{p_j} = \langle t_1^{p_j}, t_2^{p_j}, \ldots, t_n^{p_j} \rangle.$$

According to (2), p_j must be covered by certain variable in the separation set, i.e.

$$\exists i \in \Lambda, s_i^{p_j} \neq t_i^{p_j}.$$

It is equivalent to:

$$\sum_{i=1 \text{ to } n} (s_i^{p_j} \oplus t_i^{p_j}) \cdot x_i \geq 1, \tag{3}$$

where $\oplus$ is the *exclusive or* operator, and x_i the decision variable of v_i.

Let $A = \{a_{ij}\}_{m \times n}$ be a coefficient matrix where

$$a_{ij} = s_i^{p_j} \oplus t_i^{p_j}, \text{ for } 1 \leq i \leq n, 1 \leq j \leq m.$$

Obviously, a_{ij} equals 1 if and only if the state pair p_j is covered by the variable v_i. Then the MSSP can be formulated as:

$$\min \sum_{i=1}^{n} x_i, \text{ where}$$

$$\sum_{i=1}^{n} a_{ij} x_i \geq 1, \qquad j = 1, \ldots, m \tag{4}$$

$$x_i = \{0, 1\}, \qquad i = 1, \ldots, n \tag{5}$$

where equations (4) and (5) characterize the feasible solutions.

4 Our Approach

In the verification process, note that we do not necessarily need the solutions of SSP to be minimal. Thus, it is possible for us to use some approximate method to solve this problem. In [3], a decision tree learning solver is proposed. In this paper, we present a novel learning approach based on the sample learning technique, evolutionary algorithm and efficient heuristics. Experimental results show the better performance of our solver.

4.1 Sample Learning Technique

In practice, the number of failure states of SSP is very large. It is not easy to determine the separation set for large scale systems. In [3], an idea of inferring the separation set by learning from some selected samples, instead of the entire sets, was introduced.

The main procedure of our Sample Learning Approach (SLA) is shown in Alg. 1. The method avoids the complexity of SSP by considering only samples of the set of state pairs. This algorithm is iterative. By adjusting the parameters *MAX_ITER* and *MAX_SAM*, we set the maximal number of iterations and the maximal number of samples picked in every iteration. A sample here is a pair of states $\langle s, t\rangle \in F_d \times F_b$. The algorithm randomly picks *MAX_SAM* samples in every iteration, among which only those that are not covered by the present separation set (we call them efficient samples) can be added into the set *SAMPLE*. The *REQ_SIZE* is a preassigned parameter. When there are enough efficient samples generated, the set of samples will be renewed, and then the separation set is computed.

Note that we use the covering concept to judge the validity of the given samples. The samples that are already covered by the present separation set will be directly discarded. Given an appropriate value to the *REQ_SIZE*, many samples will be discarded directly according to their coverage to the present separation set, and thus the number of invoking EA solver will be greatly reduced.

Let $A_j = \langle a_{0j}, a_{1j}, \ldots, a_{nj}\rangle$ be the coefficient vector corresponding to p_j. According to (3), it is not difficult to determine the coverage of p_j to the present separation set Λ. It is equivalent to testing true value of the following formula:

$$\sum_{i=1 \text{ to } n} a_{ij} \cdot x_i \geq 1.$$

4.2 Probabilistic Evolutionary Algorithm

Evolutionary algorithm (EA) [11] is a powerful search and optimization paradigm. It utilizes the principles of natural evolution and "survival of the fittest". The EA elaborates on many solutions at the same time. The main characteristic of an evolutionary algorithm is population-based. Starting with a set of initial solutions, evolutionary algorithms explore the solution space through the simulated evolution. Solutions are evaluated by their fitness. The more suitable they are, the more chances they have to survive and be reproduced.

There are many studies on applying evolutionary algorithms to the set covering problem [12–15]. The experimental results listed in the above literatures show the good performance of applying EA to the set covering problem. However, we cannot apply EA directly to the SSP, since the huge number of failure states. Essentially, SSP is a special case of set covering problem, where the number of constraints is much more than the number of variables. By applying the sample learning technique, we avoid the complexity of such huge number of constraints.

Algorithm 1 Outline of the sample learning algorithm

```
$\Lambda := \phi$
SAMPLE := $\phi$
NEWSAMPLE := $\phi$
for $i$ := 1 to MAX_ITER do
  for $i$ := 1 to MAX_SAM do
    randomly pick $\langle s, t\rangle$ from $F_d \times F_b$
    if $\langle s, t\rangle$ cannot be covered by $\Lambda$ then
      NEWSAMPLE := NEWSAMPLE $\cup\langle s, t\rangle$
    end if
  end for
  if sizeof(NEWSAMPLE) $\geq$ REQ_SIZE then
    SAMPLE := SAMPLE $\cup$ NEWSAMPLE
    call solver to compute $\Lambda$ based on SAMPLE
    NEWSAMPLE := $\phi$
  end if
end for
```

Algorithm 2 Evolutionary algorithm

```
1: Generate a initial population
2: while not (terminal condition) do
3:   Update the chromosomes by crossover and mutation operations
4:   Evaluate the fitness of each chromosome
5:   Select chromosomes to form a new population
6: end while
```

We use EA as the central solver embedded in the learning structure and used for computing the separation set. The EA procedure is shown in Alg. 2.

We reinforce the basic EA in a way such that problem-specific knowledge is incorporated. We observe following properties in SSP, which derive the effective heuristics:

1. For a state pair, there may be multiple variables that can cover it.
2. If the variables in a separation set cover all state pairs in $F_d \times F_b$, then the corresponding solution is already a feasible solution.

In order to get a feasible solution more quickly, an effective strategy is to assign larger probabilities to the variables which cover more state pairs. Denote $EV(v)$ as the number of state pairs covered by variable v. Based on the statistic analysis on the sets of states F_d and F_b, the $EV(v)$ values for all variables can be evaluated easily in advance of the execution of our algorithm.

Probabilistic initialization We use a n-bit binary string as the chromosome structure where n is the number of invisible variables. A value of 1 for the i-th bit implies that the variable v_i is selected into the separation set.

We generate *pop_size* chromosomes to initialize the population. To obtain a random chromosome, the involved method acts as follows:

1. randomly generate an integer e $(0 \leq e \leq n)$, and use it as the size of the separation set.
2. randomly select e variables into the separation set.
3. the probability of each variable to be selected is proportional to the number of state pairs it covers.

Probabilistic mutation Let P_m be the probability of mutation. We adopt the two-point mutation. For a traditional two-point mutation, it randomly selects two points r_1 and r_2 in the chromosome, and then replaces the value of every character between sites r_1 and r_2 with a random value (0 or 1).

In our probabilistic two-point mutation, the mutation sites r_1 and r_2 are selected similarly, however, the value of each character between site r_1 and r_2 are replaced in a heuristic way as follows:

1. randomly generate a integer e $(0 \leq e < r_2 - r_1)$.
2. randomly select e genes between sites r_1 and r_2 into the separation set.
3. the probability of each gene between sites r_1 and r_2 to be chosen is proportional to the number of state pairs it covers.

Probabilistic crossover We let P_c be the probability of crossover. We adopt the uniform crossover operator. It is claimed that the uniform crossover has a better recombination potential to combine smaller building blocks into larger ones [16,17]. The uniform crossover works with a crossover mask which is created at random. The mask has the same length as the chromosome structure, and the parity of the bits indicates the corresponding parent.

We follow the probabilistic crossover operator defined in [12]. Empirical studies show that this crossover operator is suitable for the set covering problem. Probabilistic crossover is derived from the standard uniform crossover. For the probabilistic crossover operator, the probability of a parent to be chosen for contributing its variable to the offspring is proportional to its fitness value. Formally, given parents $P = \langle P_1 P_2 \ldots P_n \rangle$ and $Q = \langle Q_1 Q_2 \ldots Q_n \rangle$, the crossover mask $M = \langle M_1 M_2 \ldots M_n \rangle$ is generated as follows:

$$M_i = 0 \text{ with the probability } p = \frac{fitness(Q)}{fitness(P) + fitness(Q)}$$
$$M_i = 1 \text{ with the probability } 1 - p$$

Solution improvement When applying evolutionary operators to the chromosomes, the resulting solutions are no longer guaranteed to be feasible. We implemented two strategies to deal with infeasible solutions.

The first strategy is to apply penalty function to deteriorate the optimality of an infeasible solution by adding a penalty cost to its objective function. In our approach, after the penalty function applied, the optimization model becomes:

$$\text{Minimize } \sum_{i=1}^{n} x_i + \sum_{j=1}^{m} f(\sum_{i=1}^{n} a_{ij} x_j \geq 1),$$

where $f(\cdot)$ is the penalty function for unsatisfying the constraints (4). The penalty function has a strong influence on the performance of the whole algorithm. In our approach, we implement a simple and efficient penalty function as follows:

$$f(x) = \begin{cases} 0, & \text{if } x \text{ is true,} \\ \text{BIGVALUE}, & \text{otherwise.} \end{cases}$$

The second strategy way is to apply a heuristic operator to transform the infeasible solution into feasible solution. We implemented the heuristic feasibility operator proposed in [12] with minor modifications. By applying this heuristic operator, not only can the infeasible solutions be transformed into feasible solutions, but also the feasible solutions can be improved by eliminating the redundant variables. Algorithm 3 gives the framework of the operator.

Algorithm 3 Heuristic feasibility operator

```
1: for each $A_j$, compute the number of variables that are in the separation set and
   can cover this row, i.e.
       $$n_j = \sum_{i=1}^{n} a_{ij} x_i, \text{ for } 1 \le j \le m.$$
2: while $(\exists j \in [1, m], n_j = 0)$ do
3:     find the best variable $v^*$ which is not in the separation set and can cover maximal
       number of uncovered rows, i.e.
           $$\max_{i=1}^{n} \left\{ \sum_{j=1}^{m} (n_i = 0) \wedge (x_j = 0) \wedge (a_{ij} = 1) \right\}.$$
4:     add $v^*$ into the solution and renew $n_j$ for each $A_j$.
5:     eliminate the redundant variables, i.e. the variables satisfying:
           $$\forall j \in [1, m], (a_{ij} = 1) \wedge (x_i = 1) \rightarrow n_j \ge 2.$$
6: end while
```

5 Experimental Results

To validate our approach, we implemented our probabilistic learning approach using C^{++} language and ran on a PC with Intel$^{®}$ Celeron$^{®}$ 2.4GHz CPU and 512M RAM. All benchmarks are created using a random generator. The parameters are set as: *pop_size* = 40, *MaxIter* = 1000, $P_m = 0.25$, $P_c = 0.5$, where *pop_size* is the size of the population, *MaxIter* is the maximal number of generations, P_m and P_c are the probabilities of mutation and crossover, respectively.

The experiment compares the performance of our solver to the latest published sampling decision tree learning (SDTL) solver [2,3]. The results are listed in Table 1. *Benchmark* is the name of the tested benchmark. The benchmark's name implies the parameters. For example, the name "$ran\_k30\_m500\_n300$" indicates that the number of invisible variables is 30, the number of deadend states is 500, and the number of bad states is 300, respectively. The *time* column lists the runtime in seconds, and the |SepSet| column gives the size of the resulting separation set. We evaluate the efficiency of the solver by its runtime, and the solution quality by the size of the separation set. In order to force termination, we impose a limit of two hours on the running time. We denote by 'timeout' in the *time* column the examples that could not be solved in this time limit.

The results in Table 1 are arranged into six groups. In the former two groups, we let the numbers of deadend states and bad states be fixed, and let the number of invisible variables increase, we observed that all the solvers' run times increase in most of cases. In the latter four groups, we fixed the number of invisible variables, and let the number of deadend states and bad states increase, we observed that the SDTL solver quickly blows up, whereas our solver still works well. Even for the benchmarks that are solvable by both the solvers, the runtime of our solver are 2 to 4 orders of magnitude smaller than that of the SDTL solver. Regarding the separation set size, the separation set found by our solver is smaller 76% than that by the SDTL solver on average.

6 Conclusion

We investigated the state separation problem in this paper. A novel probabilistic learning approach was presented for solving this problem. Experimental results showed the efficiency and power of our approach. Compared with the latest work using the sampling decision tree learning (SDTL) solver, the proposed approach outperforms 2 to 4 orders of magnitude faster and the size of the separation set is 76% smaller on average.

References

1. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Computer Aided Verification. (2000) 154–169
2. Clarke, E.M., Gupta, A., Kukula, J.H., Strichman, O.: SAT based abstraction-refinement using ILP and machine learning techniques. In: CAV. (2002) 265–279
3. Clarke, E., Gupta, A., Strichman, O.: SAT based counterexample-guided abstraction-refinement. IEEE Transactions on Computer Aided Design **23**(7) (2004) 1113–1123
4. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: Symposium on Principles of Programming Languages. (2002) 58–70
5. Glusman, M., Kamhi, G., Mador-Haim, S., Fraer, R., Vardi, M.Y.: Multiple-counterexample guided iterative abstraction refinement: an industrial evaluation. In: TACAS. (2003) 176–191
6. Gupta, A., Strichman, O.: Abstraction refinement for bounded model checking. In: Computer Aided Verification. (2005) 112–124

Table 1. Our solver vs. SDTL

Benchmark	Our solver		SDTL	
	time	|SepSet|	time	|SepSet|
ran_k10_m150_n120	0.141	7	15.954	10
ran_k20_m150_n120	0.422	7	30.594	20
ran_k30_m150_n120	0.594	6	35.891	28
ran_k40_m150_n120	0.656	7	51.813	31
ran_k50_m150_n120	2.953	6	53.312	38
ran_k20_m500_n300	0.515	7	1197.562	20
ran_k30_m500_n300	1.281	7	1836.203	30
ran_k40_m500_n300	1.454	7	2664.453	40
ran_k50_m500_n300	3.907	7	3476.797	46
ran_k60_m500_n300	2.89	7	4360.484	55
ran_k30_m150_n200	0.719	6	78.485	29
ran_k30_m500_n1000	1.985	8	4028.937	30
ran_k30_m3000_n4000	3.813	8	timeout	
ran_k30_m5000_n4000	3.922	8	timeout	
ran_k40_m200_n250	1.735	7	470.078	36
ran_k40_m1000_n2000	4.703	8	timeout	
ran_k40_m2000_n5000	5.734	8	timeout	
ran_k40_m8000_n7000	8.859	8	timeout	
ran_k50_m200_n300	1.891	7	1026.672	43
ran_k50_m1000_n2000	8.672	7	timeout	
ran_k50_m2000_n5000	12.687	8	timeout	
ran_k50_m8000_n7000	23.515	9	timeout	
ran_k60_m200_n300	1.125	6	1595.594	47
ran_k60_m1000_n2000	7.578	8	timeout	
ran_k60_m2000_n5000	11.625	8	timeout	

7. Govindaraju, S.G., Dill, D.L.: Counterexample-guided choice of projections in approximate symbolic model checking. In: ICCAD. (2000) 115–119
8. McMillan, K.L., Amla, N.: Automatic abstraction without counterexamples. In: TACAS. (2003) 2–17
9. Gupta, A., Ganai, M.K., Yang, Z., Ashar, P.: Iterative abstraction using SAT-based BMC with proof analysis. In: ICCAD. (2003) 416–423
10. Wang, C., Jin, H., Hachtel, G.D., Somenzi, F.: Refining the SAT decision ordering for bounded model checking. In: DAC. (2004) 535–538
11. Dumitrescu, D., Lazzerini, B., Jain, L., Dumitrescu, A.: Evolutionary Computation. CRC Press (2000)
12. Beasley, J., Chu, P.: A genetic algorithm for the set covering problem. European Journal of Operational Research **94** (1996) 392–404
13. Sen, S.: Minimal cost set covering using probabilistic methods. In: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing, Indianapolis, Indiana, United States, ACM Press (1993) 157–164
14. Aickelin, U.: An indirect genetic algorithm for set covering problems. Journal of the Operational Research Society **53**(10) (2002) 1118–1126

15. Marchiori, E., Steenbeek, A.: An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In: EvoWorkshops. Volume 1803 of Lecture Notes in Computer Science., Springer (2000) 367–381
16. Syswerda, G.: Uniform crossover in genetic algorithms. In: Proceedings of the 3rd International Conference on Genetic Algorithms, San Mateo, California, USA, Morgan Kaufmann Publishers Inc. (1989) 2–9
17. Spears, W.M., De Jong, K.A.: On the virtues of parameterized uniform crossover. In Belew, R., Booker, L., eds.: Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufman (1991) 230–236