RWTH Aachen

Department of Computer Science
Technical Report

Experimental Evaluation of an Independent Set Algorithm

Felix Reidl

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2010-19

RWTH Aachen · Department of Computer Science · December 2010

Experimental Evaluation of an Independent Set Algorithm

Felix Reidl

Dept. of Computer Science
RWTH Aachen University, Germany
Email: felix.reidl@rwth-aachen.de

Abstract. Following up the development of the currently fastest independent set algorithm [13] we present a practical evaluation, both on real-world data taken from common biological problems and on synthetic data. In order to get a clearer picture we measure different aspects of our implementation, especially the different polynomial-time reduction rules. The main purpose of this report is to show that algorithms with provable upper bounds can indeed be used to solve practical instances, though some care must be taken in the transfer of algorithm to code.

1 Introduction

The Maximum Independent Set problem has, regardless of its purely theoretical heritage, found real-world applications in various fields. A common usage is the cleaning of (noisy) data: if pairwise conflicts between measurements are modeled as edges in a graph, an independent set constitutes a feasible subset of those. This approach is widely used in Biology, where protein sequences of related organisms are compared to establish a hierarchy of ancestry. It is therefore not only desirable to find new ways of lowering the worst-cast time bound of algorithms solving it—for which an impressive series of results exists, lowering the trivial bound of $\mathcal{O}(2^n)$ to now $\mathcal{O}(1.2132^n poly(n))$ [13]—but also to find algorithms which work in practice. In the general context of NP-hard problems several lines of research have been conducted to this end: Gramm, Guo, Hüffner and Niedermeier [11] tackled the Clique Cover problem from the angle of parameterized complexity; Alber, Dorn and Nierdermeier experimented with dynamic programming on tree decompositions for Vertex Cover on planar graphs [3]. Dehne, Langston, Luo, Shaw and Zhang studied Cluster Editing [7], which is a different approach to clean up data by dismissing relations between measurements; Langston also worked with Abu-Khzam, Shanbhag and Symons on the parallelization of Vertex Cover [2]. A similar, but earlier, approach can be found in [5] by Cheetham, Dehne, Rau-Chaplin, Stege and Taillon.
The area of fixed parameter tractablity has received attention: problems admitting a parameterization, like Vertex Cover, allow an algorithm with complexity $\mathcal{O}(f(k) poly(n))$ which looks promising for small enough values of k. For example, the above mentioned Vertex Cover admits an algorithms with running time bounded by $\mathcal{O}\left(1.2738^k poly(n)\right)$ [6] where k is the size of the minimum vertex cover. Furthermore, the toolkit of parameterized complexity offers many interesting angles from which problems can be attacked. The technique of kernelization, to name one, produces instances with sizes bounded by some function of the parameter in polynomial time—for Vertex Cover the best known kernelization yields instances of size at most $2k$. However, the premise of the parameter

k being small might not always be realistic for real-world data or the problem at hand—like the considered MAXIMUM INDEPENDENT SET—has been shown not to be fixed parameter tractable at all with respect to the solution size (assuming some complexity theoretical conjectures).
The scope of this paper is to probe the practical boundaries of a MAXIMUM INDEPENDENT SET algorithm and to provide a baseline for further comparisons. As time measurements are influenced by a lot of factors, we also provide the number of recursive calls made by the algorithm. The important polynomial operations, reduction rules, are counted separately to complete the picture.
The implemented algorithm has been shown to have a worst-case running time of $\mathcal{O}(1.2132^n poly(n))$ [13]. Modern methods [9] for the analysis of that algorithm allowed the algorithm itself to make only a minimum number of case distinctions, in contrast to, for example, the VERTEX COVER-algorithm employed in [5], which differentiates more than ten cases [4]. What follows is a brief description of the algorithm and the employed data reductions, some notes on the actual implementation—which differs in some parts from the theoretical algorithm—and finally the experimental results.

2 Search tree algorithm

We present a short excerpt taken from [13] to outline the basic algorithm. The description of the reduction rules is deferred to the next section.
A special notation used in this section is $N[v] := N(v) \cup \{v\}$, the closed neighbourhood of v. Furthermore, $\alpha(G)$ denotes the size of the maximum independent set of G.
Most central is a basic branch on a vertex v with maximum degree: we either include v into the independent set and remove $N[v]$ from the graph or we exclude v. This branching is augmented by the following two rules:

Definition 1. *Let G be a graph $v \in V$. $u \in N(N(v))$ is called* mirror *of v if $N(v) \setminus N(u)$ is a clique.* $M(v) := \{u \in N(N(v)) \mid u \text{ is a mirror of } v\}$.

As shown by Fomin, Grandoni and Kratsch [8], the mirrors of a vertex v can be excluded from an independent set alongside v. For the positive case, i.e. including v into the independent set, the set of so-called satellites is helpful:

Definition 2. *Let G be a graph $v \in V$. $u \in N(N(v))$ is called* satellite *of v, if there is $u' \in N(v)$ such that $N(u') \setminus N(v) = \{u\}$.* $S(v) := \{u \in N(N(v)) \mid u \text{ is a satelite of } v\}$.

As shown in [12], the satellites of a vertex v can be *included* into the independent set alongside v. This implies the assumption that the satellites are pairwise non-adjacent which follows from a reduction rule presented below.
It is important to note that one cannot use both satellites and mirrors at the same time, therefore the algorithms decides before each branch which rule has the greater gain.

Algorithm 1 A fast algorithm for INDEPENDENT SET.

Input: a graph $G = (V, E)$
Output: $\alpha(G)$

```
apply reduction rules to G;
if G is not connected then compute α for each component independently;
if G is cubic then apply algorithm for cubic graphs;
select v ∈ V of maximum degree that yields the best branching number;
if the mirror branch on v is more efficient than the satellite branch then
        return max(α(G \ M[v]), 1 + α(G \ N[v]));
else return max(α(G \ {v}), 1 + |S(v)| + α(G \ N[S[v]]));
```

3 Data reduction

Polynomial-time preprocessing or data reduction reduces the input size of a given instance at 'no cost' from a theoretical standpoint. Reduction rules are usually invoked to rule out certain structures in the graph, thus enabling a finer analysis of the remaining cases. The following rules were used in the analysis of the MAXIMUM INDEPENDENT SET-algorithm and are therefore presented here briefly. Note that some of them are *not* included in the actual implementation, these cases are discussed in the following section.

3.1 Vertice of degree less than 2

Vertices of degree one or zero can always be included in an independent set. For the degree-one vertex v with neighbour u this follows from a simple exchange argument: assume we find an independent set which contains u. Then surely, the same set with u exchanged for v is independent, too. On the other hand, if a independent set does not contain neither u nor v, it cannot be of maximal size: we can add v without conflict.
Therefore, for a vertex v of degree one or zero the following holds:

$$\alpha(G) = \alpha(G - N[v]) + 1$$

3.2 Domination

Definition 3. *Let $G = (V, E)$ be a graph and $u, v \in V$. If $N[u] \subseteq N[v]$, we say v* dominates *u. We call v a* dominating node.

A dominating vertex can only be a worse choice (note that the definition implies that u and v are connected): if it would be included in a independent set we could swap it for the dominated one without any conflict—this is, in essence, a generalized version of the degree-one rule outlined above. It follows that we can remove all dominating vertices from the input graph.
Assume v dominates u, then

$$\alpha(G) = \alpha(G - v)$$

holds.

3.3 Vertices of degree 2

Definition 4. *Let $G = (V, E)$ be a graph and $v \in V$ and $N(v) = \{u_1, u_2\}$. Adding a new node u, connecting u to each node in $N(\{u_1, u_2\})$ and removing $N[v]$ is called* folding.

A vertex v of degree two can present itself in two different ways, namely with either connected or disconnected neighbours. In the former case both neighbours dominate v and the above domination rule works. The latter case can be resolved by folding v: this works because we can restrict ourselves to independent sets which either contain v or u_1, u_2 which is 'modeled' in the reduced instance by either taking u (including u_1 and u_2 in the original graph) or not (taking v in the original graph).
Assume G^* is obtained from G by folding such a vertex v, then the reduction looks as follows:

$$\alpha(G) = \alpha(G^*) + 1$$

3.4 Connected satellites

As mentioned above, the branching with satellites assumes that the satellites are pairwise non-adjacent. The following lemma helps in case they are not:

Lemma 1 ([13]). *Let $G = (V, E)$ be a graph, and $v, u, w \in V$, such that $u, w \in S(v)$ and $\{u, w\} \in E$. Then*

$$\alpha(G) = \alpha(G \setminus \{v\})$$

3.5 Multiple components and Fürers reduction

An independent set can be computed for each component of a graph separately, therefore if the graph G has components $G_1, \ldots, G_l$ the recursion

$$\alpha(G) = \sum_{i=0}^{l} \alpha(G_i)$$

immediately follows. Fürers reduction provides a way of dealing with graphs which *almost* break down into multiple components, i.e. graphs with a separator of size at most two. It has a good number of subcases, we therefore refer to [10] for the details.

4 Implementation

In practice the described algorithm faces some restrictions: polynomial-time reductions are not feasible if the complexity is more than linear in the input size and we cannot evaluate the gain of every single possible branch beforehand. On the other hand, the algorithm can be enhanced by a bounding method which greatly reduces the size of the considered solution space—something which cannot be accounted for in the theoretical analysis. Such Branch&Bound methods are succesfully used in practice, even with apparantly bad theoretical bounds. We implemented a first draft of the algorithm in Java using our in-house graph library and refined the running time by tailoring the general-purpose classes to our needs. For example, the reduction procedure heavily queries the degree of the vertices, which are therefore cached in an array. We feel confident that this course of action—developing in a high-level language with memory management—reduces the development time significantly and eased experimentation with different methods as well as concentrating on a correct implementation rather than correct memory management. Of course, a mature, usable program should finally be written in C++, but a migration is easily made.

4.1 Branch and Bound

We obtain a simple bound by using the well-known 2-approximation of Vertex Cover. Assume the result of this approximation is a vertex cover of size k, then the smallest possible cover has at least the size $k/2$. Accordingly, the largest independent set we can hope for has a size of $n - k/2$. If the largest found independent set so far is of size max, and we have c vertices included so far in the candidate set, we can cut off the next branching step if

$$c + (n - k/2) < max$$

where n denotes the size of the remaining graph. We initialize max by using the same method beforehand, obtaining a vertex cover approximation and using it to calculate an upper bound for the independent set.

4.2 Reduction

In the process of implementing the algorithm we quickly noticed that the domination reduction as well as the satellite reduction described above are too expensive in their most general form: both require consideration of the second neighbourhood of a vertex, which can easily encompass a large portion of the graph. We therefore restrict both methods to vertices with a degree of at most seven which worked well in our experiments.
Both Fürers reduction as well as the branching on components are not implemented: the test for connectivity costs $\mathcal{O}(|E|)$ time, which is infeasible to do in every recursive call. We experimented on how often subgraphs in the lower parts of the search tree (i.e. subgraphs of a constant size) broke down and found that the number was too small to account for the additional polynomial overhead.
Additionally to the above we implemented the following well-known reductions:

Buss reduction Taken from the Buss kernelization algorithms for Vertex Cover this method allows us to get rid of vertices with very high degree. The observation is as follows: if a vertex v has a degree of $n - (max - c)$ or more, where max again denotes the largest currently known solution and c the size of the candidate set, it cannot be contained in an independent set that is larger than max. Thus we can safely remove v from the graph (assuming that $c < max$).

Folding larger degree vertices The operation of folding has been extended by Fomin, Grandoni and Kratsch to higher degree vertices whose neighbourhoods contain no anti-triangle [8]. A drawback of this method is that, in the general case, we first have to check whether the neighbourhood of the considered vertex has no anti-triangles and then introduce new vertices for each anti-edge. As this might actually increase the number of vertices we only considered one special case: if the neighbourhood of a vertex v consists has the form $N(v) = \{C, u\}$ where $G[C]$ forms a clique and $N(u) \cap C = \{v\}$, i.e. u is not connected to C, then v is *foldable* by the following operation:

1. For each $c \in C$, connect c to each vertex in $N(u)$.
2. Remove v and u from the graph

It can be easily verified that this operation is just a special case of the folding operation described in [8]. As this special structure in the neighbourhood of a vertex is increasingly less likely to occur (if one assumes that the different neighbourhoods have are more or less equal frequency in real-world examples) we restrict the search for it to vertices with degree at most ten—the verification of a clique (after exactly one vertex not connected to the rest has been found) otherwise would again cost quadratic time in the number of vertices. Note that this operation is in a sense compatible with the domination reduction: if C contained vertices dominated by others, the additional edges do not destroy this relation as the neighbourhoods of all vertices of C are extended by the same vertices.

Connected satellites It should be noted that the connected satellites reduction rule hardly ever occurs. Yet because the satellite *branching* is employed regularly, we cannot dismiss it. As we limited the satellite rule to vertices with degree seven, the satellite-augmented branching is used only on vertices with that degree.

4.3 Avoiding copy operations

If one would implement a branching algorithm naively, each branching step would copy the graph and work on it recursively. Such a copy operation takes time $\mathcal{O}(|V| + |E|)$ and is far to expensive to carry out. It is also rather wasteful, as the graph is only modified locally. A reasonable approach is to store the changes made to the graph in a list and reverse these when the recursive call returns. At the same time we restrict the memory usage to $\mathcal{O}(|V| + |E|)$: we store the graph, the modifications (in a depth-first recursion these are a constant times the graph size) and we need space for the calling stack itself. This opens up the possibility of using the additional space to cache results of small graphs, similar to memoization [14][1], a topic which should be explored in the future.

4.4 Randomized branching

Finally, we randomized the order in which the search tree is traversed in order to avoid worst-case behaviors on graphs with special structures. One important question is with which probability the two cases, including a vertex vs. exluding it, should be weighted: including a vertex probably decreases the size of the graph more as excluding it, thus a leaf in the search tree should be reached earlier. On the other hand, the solution size in such a leaf might be quite small (as we

[1] Memoization as presented there is not usable in combination with operations like folding, but in a practical setting caching of small graphs is of course possible by hashing

pretty much 'greedily' included vertices on that branch of the search tree) and in consequence the resulting bound not of much use.
Both to get an idea of how strong the impact of the chosen probability could be as well as to test whether there is a single value suited for all instance we conducted a separate experiment on a subset of the test cases. For all other experiments we used the neutral probability of 50% for both cases.

5 Experimental results

We tested our algorithm on several different data sets in order to establish what constitutes an 'easy' and what a 'difficult' instance. The first data set contains biological data similar to that presented in [5] obtained from the NCBI database (http://www.ncbi.nlm.nih.gov/) and processed with ClustalW [15]. The graphs are constructed as follows: a set of protein sequences is chosen from the NCBI database which then is fed to the ClustalW toolkit, which outputs a scoring for each pair of sequences (between 0 and 100, a high value denotes a high similiarity). From this data we want to obtain a graph of conflicts, i.e. with edges only between sequences where the scoring fell below a chosen threshold. As this threshold is somewhat arbitrary, we tested all values between 0 and 12 and chose the instance that had the worst running time (and did not break into small components). We would have liked to include larger instances, as the running times are very good on the tested ones, but the preprocessing with ClustalW was not feasible for such. The second set was obtained from the second DIMACS implementation challenge (http://dimacs.rutgers.edu/Challenges/). The graphs contained in that challenge were created in order to test CLIQUE-heuristics, we therefore used the complement graphs of the supplied instances.
Finally, we also tested our algorithms on randomly generated graphs and grids (grids of course could be solved in polynomial time, but the algorithm is agnostic to the fact that it works on a bipartite graph).

5.1 Running time

The resulting running times of the algorithms are depicted in Table 1, a comparison with random graphs and grids is displayed in Figure 1. We chose square grids as the number of branches depends solely on the smaller dimension, albeit the number of reductions increases accordingly. The algorithm apparently only uses the clean and folding reduction rules on grids.
The creation of the biological data outlined above entails the choice of the conflict threshold (below which the scoring is considered to low and the protein sequence as conflicting). To illustrate the impact of that threshold on the running time we supply an example in Figure 2.

5.2 Impact of reduction rules

We have inspected the number of times a reduction rule is applied by the algorithm. In order to have a reasonable comparison, these numbers are divided by the size of the search tree, thus they display the average number of times a specific reduction rule has been applied. The operation of removing degree one

Data set	Instance	$\|V\|$	$\|E\|$	MIS size	Calls	Time (seconds)
DIMACS	p_hat300-1	300	33917	8	12122	6
	p_hat300-2	300	22922	25	52358	29
	p_hat300-3	300	11460	36	4613114	772
	p_hat500-1	500	93181	9	67999	44
	p_hat500-2	500	61804	36	1948830	540
	p_hat500-3	500	30950	50	520307757	>1 day
	p_hat700-1	700	183651	11	275696	182
	p_hat700-2	700	122922	44	15311001	6041
	MANN_a9	45	72	16	19	1
	MANN_a27	378	702	126	54158	20
	MANN_a45	1035	1980	345	98974466	13450
	brock200_1	200	5066	21	2779061	764
	brock200_2	200	10024	12	38828	11
	brock200_3	200	7852	15	211349	40
	brock200_4	200	6811	17	464914	108
	brock400_1	400	20077	27	9921590417	>1 day
	brock400_2	400	20014	29	4068393481	>1 day
	brock400_3	400	20119	31	2425541535	>1 day
	san200_0.7_1	200	5970	30	21080	15
	san200_0.7_2	200	5970	18	261183450	>1 day
	san200_0.9_1	200	1990	70	49902	97
	san200_0.9_2	200	1990	60	1204567	671
	san200_0.9_3	200	1990	44	17986522	9386
	san400_0.5_1	400	39900	13	1767204195	>1 day
	sanr200_0.7	200	6032	18	986085	97
	sanr200_0.9	200	2037	42	36089932	7054
	sanr400_0.5	400	39816	13	2274301	279
	sanr400_0.7	400	23931	21	581220736	>1 day
	c-fat200-1	200	18366	12	59	1
	c-fat200-2	200	16665	24	28	1
	c-fat200-5	200	11427	58	13	1
	c-fat500-1	500	120291	14	128	2
	c-fat500-2	500	115611	26	69	1
	c-fat500-5	500	101559	64	22	1
	c-fat500-10	500	78123	126	15	1
	hamming6-2	64	192	32	107	1
	hamming6-4	64	1312	4	280	1
	hamming8-2	256	1024	128	25824	76
	hamming8-4	256	11776	16	1161211	366
	keller4	171	5100	11	63483	15
NCBI	sh2_nematodes	794	30331	324	469986	1524
	somatostatin_vertebrates	747	28496	369	3088	26
	ww_landplants	582	38888	225	1542	9
	thrombin_rodents	715	43547	398	1989	8
	ded_mixed	498	21760	168	4302	23

Table 1. Running times marked with "> 1day" have finished, but not in an acceptable time.

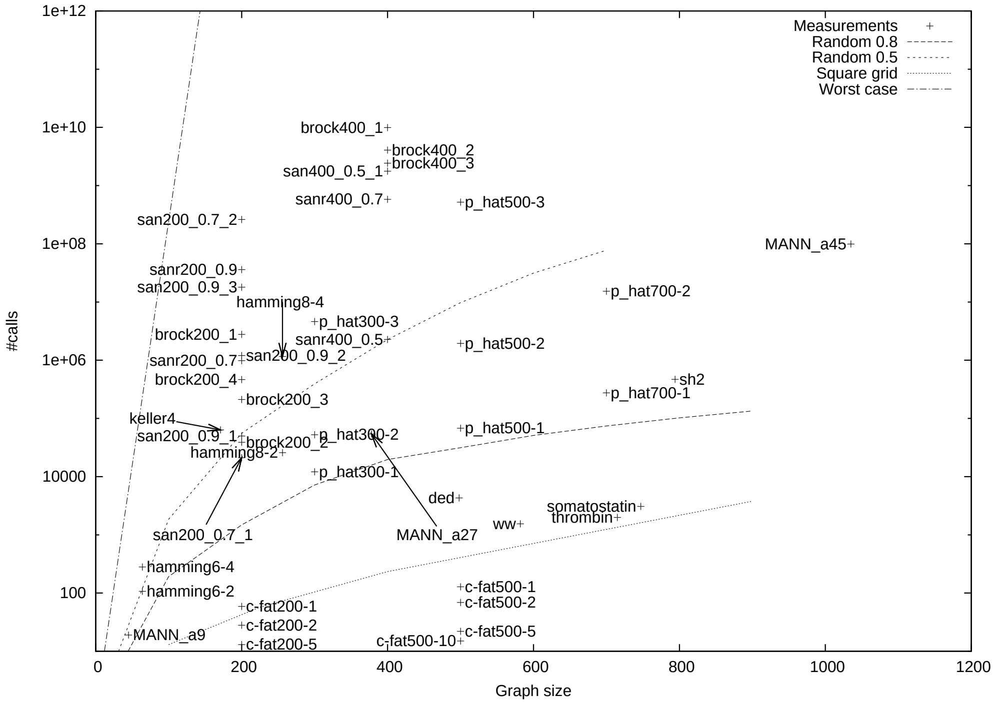

Fig. 1. Overview of all tested instance. As a comparison, the theoretical worst case (without polynomial factors) and the number of recursive calls of the algorithms on generated graphs is shown. The edge probabilities for the random graphs are displayed in the legend.

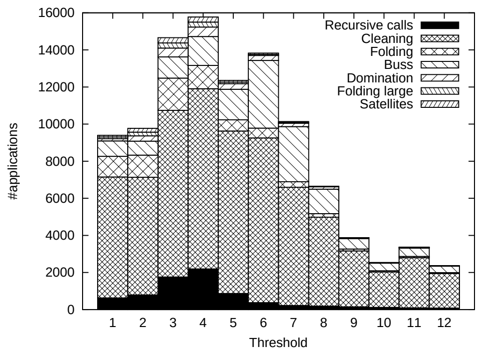

Fig. 2. Impact of changing the threshold for the `thrombin` instance on running time and number of reductions. The displayed measurements took less than a minute each.

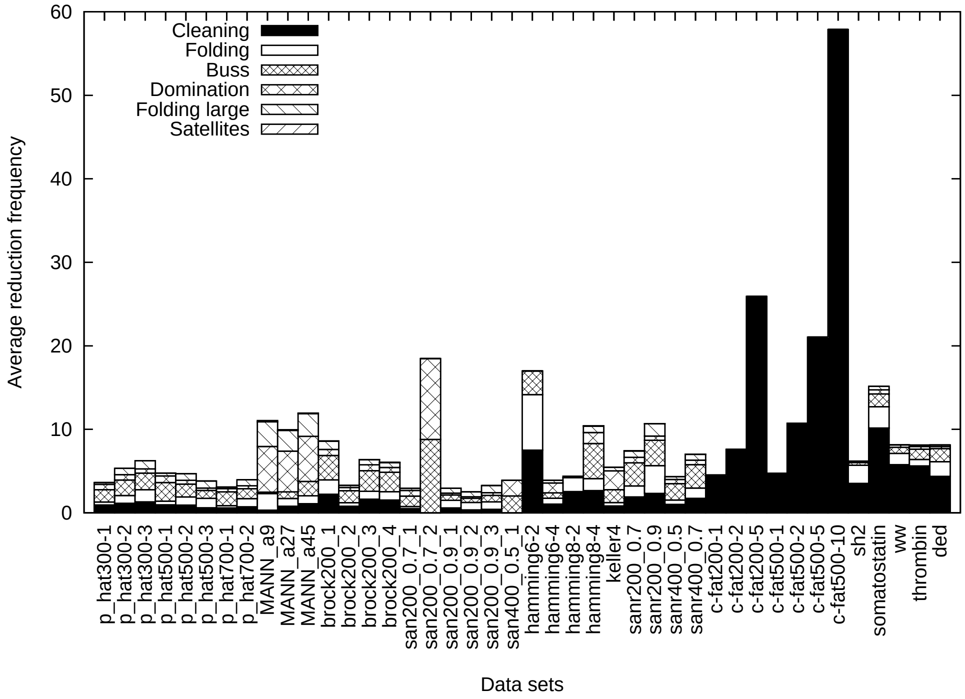

Fig. 3. Application of reduction rules per data set, normalized by the respective number of recursive calls (i.e. average number of times a certain rule has been applied per call).

and degree zero vertices are grouped together into the category *Cleaning*, the folding for vertices of degree larger than two is called *Folding large*.

Figure 3 shows the relative frequency of the different reduction rules. As mentioned before, the satellites rule is hardly ever applied. Cleaning is, as could be expected, applied regularly. Surprisingly, the Buss rule—which, as should be stressed, has no *theoretical* gain—is applied even more frequently (it should be noted that vertices of degree zero, one or two are always removed by the other reductions in our implementation). This happens mainly in the lower parts of the search tree, where already a good portion of the vertices have been selected for the independent set and a good bound on the solutions size is known.

5.3 Impact of branching probability

Biasing the branching decision, i.e. which subcase should be explored first, has a quite interesting impact on the observed running time of the algorithm. In order to explore this issue further, we sampled the running time with respect to the branching probability in increments of 1/8, where a probability of 0.0 results in always exploring the inclusion case and 1.0 the exclusion case first. Figure 4 shows the results for a selected set of instances, displaying the different trends observed. Note the huge difference in running time for the `sh2` dataset: the worst and best case exhibit a gap of 80%! As we feared that these discrepancies might be due to the vertex ordering—although the branching is randomized, the overall search tree look identical safe for branches which might be cut off differently—we repeated the experiment again for the `sh2` dataset but shuffled the vertices

beforehand. The result is displayed in Figure 4 on the right hand side: the overall trend is still clearly visible, the vertex order seems no to be the culprit.
Other instances, like the `hamming` dataset, feature hardly any variation at all. We found that this is also true for the `johnson` dataset (both where obtained from problems relevant to coding theory), which suggests that highly symmetric graphs—a well-known worst case for independent set algorithms—cannot be solved faster by chosing a certain branching probability.

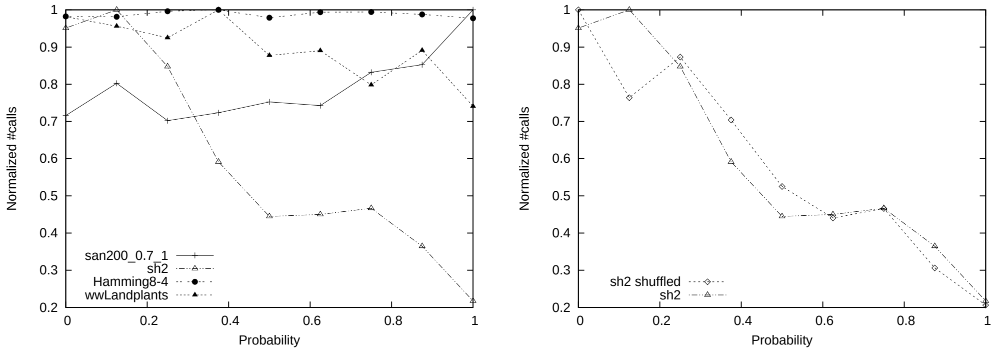

Fig. 4. Selected measurements on the impact of a branching bias. The `sh2` measurement, being the most extreme example, was tested again with permutations applied to the input graph in order to rule out the possibility that the vertex order might be responsible for the observed discrepancies. Averages over ten measurements where taken in both measurements.

Nevertheless, knowing the right probability could, at least for some instances, greatly reduce the running time. The problem, of course, is to determine it beforehand without too much effort. One practical approach we tested works by checking the bias on small samples of the graph; the sampling method employed is displayed in algorithm 2: we start out by putting a random vertex in the sample set S. While S is smaller than the desired sample size, we choose a random vertex from $N(S)$—the vertices connected to any vertex already contained in the sample—but weight the respective probabilities by the vertex degree restricted to S. For example, assume S has neighbouring vertices u, v, w and u has 1, v has 3 and w has 4 neighbours in S. Then u is chosen with probability 1/8, v with 3/8 and w with 4/8. The reasoning behind this approach is that the overall structure of graph is better reflected by the sample than by simply choosing a random connected set of vertices (which might, for example, induce a sparse graph even if the original graph is dense).
This sampling should preserve some structural details of the graph, as only neighbours to already selected vertices may in turn be added to the sample. The results for instances used before are plotted in Figure 5. One immediately notices a globally skewed tendency favoring the exclusion branch. While the instances favoring the exclusion case show a similar trend in sampling (`ww` and `sh2`), the instance selected because of its reversed trend (`san200_0.7_1`) is poorly reflected by it. This might be due to the overall small size of the instance: the larger sampling of 50% shows the general trend of `san200_0.7_1`, albeit quite weakly. The most rigid instance (`hamming8-4`) shows a trend in the sampling tests not found in the original measurement. This false positive does not pose a problem for the

Algorithm 2 The sampling algorithm.

Input: a graph $G = (V, E)$ and an integer $0 < k \leq |V|$
Output: a subgraph G' of G with $|G'| = k$

Choose random $v \in V$
$S := \{v\}$
while $|S| < k$
  $C := N[S] = \{c_1, \ldots, c_k\}$
  for i **in** $1 \ldots k$
    $p_i := |\{(c_i, v) | v \in S\}| / |\{(u, v) | u \text{ or } v \in S\}|$
  Choose one c_j with probability p_j
  $S := S \cup \{c_j\}$
return $G[S]$

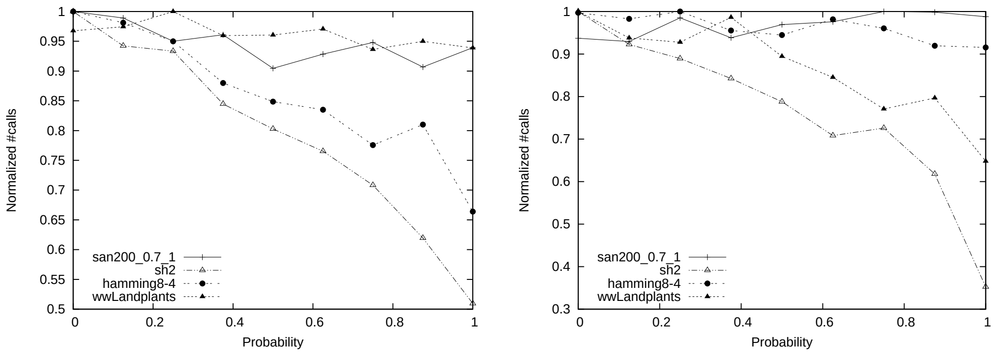

Fig. 5. Probing the resulting running time for different branching probabilities. The left diagram shows the result for a sample size of 25%, the right for 50%. In both plots averages over ten repetitions where used. In comparison to Figure 4 the overall trends are visible.

method, as choosing a certain probability for these cases has no big influence on the running time, anyway.

6 Conclusion

We have implemented a variant of the, from a theoretical point of view, currently fastest independent set-solver. The different aspects of the algorithm have been critically examined with respect to their practical usefulness and the resulting algorithm has been tested on various instances. First and foremost, we conclude that this algorithm works very satisfying for the chosen practical scenario. In fact, the generation of the data through string alignment poses the greater bottleneck in the workflow: it was impossible for us to obtain instances of size 1000 or more[2]. Another important conclusion concerns the Buss reduction rule: its surprisingly high impact coupled with the low effort needed to apply it (simply checking the degree) makes it a very valuable addition to any independent set algorithm. This begs the question: could other kernelization algorithms be used in this manner? After all, the guaranteed bounds of the respective Vertex Cover-kernels are far better—but the question is whether this additional gain can compensate for the additional overhead: compared to the simple degree-lookup of the Buss rule, those kernelizations are quite complicated.

[2] There are online services which compute sequence alignments on clusters, but the input size is restricted to about 500 proteins. ClustalW crashes for large inputs on our local machines.

The branching bias—a very simple and easy to implement addition—can in some cases reduce the running time dramatically, notably in all our real-world instances. The sampling method yields a good bias to start with, but more complicated "branching strategies" are imaginable: techniques employed in AI to traverse large search spaces might be usable to that end. Observing the large gain in some instances this seems like an interesting and fruitful area for further research; especially if complemented by an approach to solve independent set on very symmetric graphs. Whether the observed impact of a branch bias on some instances is only an artifact visible in small instances or whether it is rooted in structural properties of graphs of arbitrary size is up for debate. We would like to note, however, that even if the effect vanishes asymptotically, it is nonetheless important for the instance sizes currently feasible.
We have shown that solving independent set is feasible for certain real-world instances and that the approach of polynomial-time reduction is a great addition for the practical toolkit. The measurements suggest that optimizing the implementation towards fast recognition and execution of reductions and not necessarily towards a quick branching could lead to faster solvers. In concordance with results obtained with regard to kernelization methods [1, 7], preprocessing—not only data reduction but also probing how the instance reacts to the algorithm—seems of grave importance for fast, real-world algorithms.

References

1. F. N. Abu-Khzam, R. L. Collins, M. R. Fellows, M. A. Langston, W. H. Suters, and C. T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In Lars Arge, Giuseppe F. Italiano, and Robert Sedgewick, editors, *ALENEX/ANALC*, pages 62–69. SIAM, 2004.
2. F. N. Abu-Khzam, M. A. Langston, P. Shanbhag, and C.T. Symons. Scalable parallel algorithms for fpt problems. *Algorithmica*, 45:269–284, 2006. 10.1007/s00453-006-1214-1.
3. J. Alber, F. Dorn, and R. Niedermeier. Experimental evaluation of a tree decomposition-based algorithm for vertex cover on planar graphs. *Discrete Applied Mathematics*, 145(2):219 – 231, 2005. Structural Decompositions, Width Parameters, and Graph Labelings.
4. R. Balasubramanian, M. R. Fellows, and V. Raman. An improved fixed parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.
5. J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, and P. J. Taillon. Solving large fpt problems on coarse-grained parallel machines. *Journal of Computer and System Sciences*, 67:691–706, 2003.
6. J. Chen, I.A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In R. Kralovic and P. Urzyczyn, editors, *Proceedings of the 31st Conference on Mathematical Foundations of Computer Science (MFCS)*, number 4162 in Lecture Notes in Computer Science, pages 238–249. Springer, 2006.
7. F. K. H. A. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw, and Y. Zhang. The cluster editing problem: Implementations and experiments. In Hans Bodlaender and Michael Langston, editors, *Parameterized and Exact Computation*, number 4169 in Lecture Notes in Computer Science, pages 13–24. Springer Berlin / Heidelberg, 2006. 10.1007/11847250_2.
8. F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 18–25, 2006.
9. F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. Technical Report 359, Department of Informatics, University of Bergen, July 2007.
10. M. Fürer. A faster algorithm for finding maximum independent sets in sparse graphs. In *Proceedings of the 7th Symposium on Latin American Theoretical Informatics (LATIN)*, number 3887 in Lecture Notes in Computer Science, pages 491–501. Springer, 2006.

11. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Data reduction and exact algorithms for clique cover. *ACM Journal of Experimental Algorithmics*, 13, 2008.
12. J. Kneis and A. Langer. Satellites and mirrors for solving independent set on sparse graphs. Technical Report AIB-2009-08, RWTH Aachen University, April 2009.
13. J. Kneis, A. Langer, and P. Rossmanith. A fine-grained analysis of a simple independent set algorithm. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009)*, volume 4 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 287–298, 2009.
14. J. M. Robson. Algorithms for maximum independent sets. *J. Algorithms*, 7:425–440, 1986.
15. J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

Aachener Informatik-Berichte

This list contains all technical reports published during the past five years. A complete list of reports dating back to 1987 is available from `http://aib.informatik.rwth-aachen.de/`. To obtain copies consult the above URL or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

2005-01 *	Fachgruppe Informatik: Jahresbericht 2004
2005-02	Maximillian Dornseif, Felix C. Gärtner, Thorsten Holz, Martin Mink: An Offensive Approach to Teaching Information Security: “Aachen Summer School Applied IT Security”
2005-03	Jürgen Giesl, René Thiemann, Peter Schneider-Kamp: Proving and Disproving Termination of Higher-Order Functions
2005-04	Daniel Mölle, Stefan Richter, Peter Rossmanith: A Faster Algorithm for the Steiner Tree Problem
2005-05	Fabien Pouget, Thorsten Holz: A Pointillist Approach for Comparing Honeypots
2005-06	Simon Fischer, Berthold Vöcking: Adaptive Routing with Stale Information
2005-07	Felix C. Freiling, Thorsten Holz, Georg Wicherski: Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks
2005-08	Joachim Kneis, Peter Rossmanith: A New Satisfiability Algorithm With Applications To Max-Cut
2005-09	Klaus Kursawe, Felix C. Freiling: Byzantine Fault Tolerance on General Hybrid Adversary Structures
2005-10	Benedikt Bollig: Automata and Logics for Message Sequence Charts
2005-11	Simon Fischer, Berthold Vöcking: A Counterexample to the Fully Mixed Nash Equilibrium Conjecture
2005-12	Neeraj Mittal, Felix Freiling, S. Venkatesan, Lucia Draque Penso: Efficient Reductions for Wait-Free Termination Detection in Faulty Distributed Systems
2005-13	Carole Delporte-Gallet, Hugues Fauconnier, Felix C. Freiling: Revisiting Failure Detection and Consensus in Omission Failure Environments
2005-14	Felix C. Freiling, Sukumar Ghosh: Code Stabilization
2005-15	Uwe Naumann: The Complexity of Derivative Computation
2005-16	Uwe Naumann: Syntax-Directed Derivative Code (Part I: Tangent-Linear Code)
2005-17	Uwe Naumann: Syntax-directed Derivative Code (Part II: Intraprocedural Adjoint Code)
2005-18	Thomas von der Maßen, Klaus Müller, John MacGregor, Eva Geisberger, Jörg Dörr, Frank Houdek, Harbhajan Singh, Holger Wußmann, Hans-Veit Bacher, Barbara Paech: Einsatz von Features im Software-Entwicklungsprozess - Abschlußbericht des GI-Arbeitskreises “Features”
2005-19	Uwe Naumann, Andre Vehreschild: Tangent-Linear Code by Augmented LL-Parsers

2005-20 Felix C. Freiling, Martin Mink: Bericht über den Workshop zur Ausbildung im Bereich IT-Sicherheit Hochschulausbildung, berufliche Weiterbildung, Zertifizierung von Ausbildungsangeboten am 11. und 12. August 2005 in Köln organisiert von RWTH Aachen in Kooperation mit BITKOM, BSI, DLR und Gesellschaft fuer Informatik (GI) e.V.

2005-21 Thomas Noll, Stefan Rieger: Optimization of Straight-Line Code Revisited

2005-22 Felix Freiling, Maurice Herlihy, Lucia Draque Penso: Optimal Randomized Fair Exchange with Secret Shared Coins

2005-23 Heiner Ackermann, Alantha Newman, Heiko Röglin, Berthold Vöcking: Decision Making Based on Approximate and Smoothed Pareto Curves

2005-24 Alexander Becher, Zinaida Benenson, Maximillian Dornseif: Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

2006-01 * Fachgruppe Informatik: Jahresbericht 2005

2006-02 Michael Weber: Parallel Algorithms for Verification of Large Systems

2006-03 Michael Maier, Uwe Naumann: Intraprocedural Adjoint Code Generated by the Differentiation-Enabled NAGWare Fortran Compiler

2006-04 Ebadollah Varnik, Uwe Naumann, Andrew Lyons: Toward Low Static Memory Jacobian Accumulation

2006-05 Uwe Naumann, Jean Utke, Patrick Heimbach, Chris Hill, Derya Ozyurt, Carl Wunsch, Mike Fagan, Nathan Tallent, Michelle Strout: Adjoint Code by Source Transformation with OpenAD/F

2006-06 Joachim Kneis, Daniel Mölle, Stefan Richter, Peter Rossmanith: Divide-and-Color

2006-07 Thomas Colcombet, Christof Löding: Transforming structures by set interpretations

2006-08 Uwe Naumann, Yuxiao Hu: Optimal Vertex Elimination in Single-Expression-Use Graphs

2006-09 Tingting Han, Joost-Pieter Katoen: Counterexamples in Probabilistic Model Checking

2006-10 Mesut Günes, Alexander Zimmermann, Martin Wenig, Jan Ritzerfeld, Ulrich Meis: From Simulations to Testbeds - Architecture of the Hybrid MCG-Mesh Testbed

2006-11 Bastian Schlich, Michael Rohrbach, Michael Weber, Stefan Kowalewski: Model Checking Software for Microcontrollers

2006-12 Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker: Replaying Play in and Play out: Synthesis of Design Models from Scenarios by Learning

2006-13 Wong Karianto, Christof Löding: Unranked Tree Automata with Sibling Equalities and Disequalities

2006-14 Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen, Andreas Wolfram: Report of the GI Work Group "Requirements Management Tools for Product Line Engineering"

2006-15 Sebastian Ullrich, Jakob T. Valvoda, Torsten Kuhlen: Utilizing optical sensors from mice for new input devices

2006-16 Rafael Ballagas, Jan Borchers: Selexels: a Conceptual Framework for Pointing Devices with Low Expressiveness

2006-17 Eric Lee, Henning Kiel, Jan Borchers: Scrolling Through Time: Improving Interfaces for Searching and Navigating Continuous Audio Timelines

2007-01 * Fachgruppe Informatik: Jahresbericht 2006

2007-02 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl: SAT Solving for Termination Analysis with Polynomial Interpretations

2007-03 Jürgen Giesl, René Thiemann, Stephan Swiderski, and Peter Schneider-Kamp: Proving Termination by Bounded Increase

2007-04 Jan Buchholz, Eric Lee, Jonathan Klein, and Jan Borchers: coJIVE: A System to Support Collaborative Jazz Improvisation

2007-05 Uwe Naumann: On Optimal DAG Reversal

2007-06 Joost-Pieter Katoen, Thomas Noll, and Stefan Rieger: Verifying Concurrent List-Manipulating Programs by LTL Model Checking

2007-07 Alexander Nyßen, Horst Lichter: MeDUSA - MethoD for UML2-based Design of Embedded Software Applications

2007-08 Falk Salewski and Stefan Kowalewski: Achieving Highly Reliable Embedded Software: An empirical evaluation of different approaches

2007-09 Tina Kraußer, Heiko Mantel, and Henning Sudbrock: A Probabilistic Justification of the Combining Calculus under the Uniform Scheduler Assumption

2007-10 Martin Neuhäußer, Joost-Pieter Katoen: Bisimulation and Logical Preservation for Continuous-Time Markov Decision Processes

2007-11 Klaus Wehrle (editor): 6. Fachgespräch Sensornetzwerke

2007-12 Uwe Naumann: An L-Attributed Grammar for Adjoint Code

2007-13 Uwe Naumann, Michael Maier, Jan Riehme, and Bruce Christianson: Second-Order Adjoints by Source Code Manipulation of Numerical Programs

2007-14 Jean Utke, Uwe Naumann, Mike Fagan, Nathan Tallent, Michelle Strout, Patrick Heimbach, Chris Hill, and Carl Wunsch: OpenAD/F: A Modular, Open-Source Tool for Automatic Differentiation of Fortran Codes

2007-15 Volker Stolz: Temporal assertions for sequential and concurrent programs

2007-16 Sadeq Ali Makram, Mesut Güneç, Martin Wenig, Alexander Zimmermann: Adaptive Channel Assignment to Support QoS and Load Balancing for Wireless Mesh Networks

2007-17 René Thiemann: The DP Framework for Proving Termination of Term Rewriting

2007-18 Uwe Naumann: Call Tree Reversal is NP-Complete

2007-19 Jan Riehme, Andrea Walther, Jörg Stiller, Uwe Naumann: Adjoints for Time-Dependent Optimal Control

2007-20 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, and Verena Wolf: Three-Valued Abstraction for Probabilistic Systems

2007-21 Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre: Compositional Modeling and Minimization of Time-Inhomogeneous Markov Chains

2007-22 Heiner Ackermann, Paul W. Goldberg, Vahab S. Mirrokni, Heiko Röglin, and Berthold Vöcking: Uncoordinated Two-Sided Markets

2008-01 * Fachgruppe Informatik: Jahresbericht 2007
2008-02 Henrik Bohnenkamp, Marielle Stoelinga: Quantitative Testing
2008-03 Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, Harald Zankl: Maximal Termination
2008-04 Uwe Naumann, Jan Riehme: Sensitivity Analysis in Sisyphe with the AD-Enabled NAGWare Fortran Compiler
2008-05 Frank G. Radmacher: An Automata Theoretic Approach to the Theory of Rational Tree Relations
2008-06 Uwe Naumann, Laurent Hascoet, Chris Hill, Paul Hovland, Jan Riehme, Jean Utke: A Framework for Proving Correctness of Adjoint Message Passing Programs
2008-07 Alexander Nyßen, Horst Lichter: The MeDUSA Reference Manual, Second Edition
2008-08 George B. Mertzios, Stavros D. Nikolopoulos: The λ-cluster Problem on Parameterized Interval Graphs
2008-09 George B. Mertzios, Walter Unger: An optimal algorithm for the k-fixed-endpoint path cover on proper interval graphs
2008-10 George B. Mertzios, Walter Unger: Preemptive Scheduling of Equal-Length Jobs in Polynomial Time
2008-11 George B. Mertzios: Fast Convergence of Routing Games with Splittable Flows
2008-12 Joost-Pieter Katoen, Daniel Klink, Martin Leucker, Verena Wolf: Abstraction for stochastic systems by Erlang's method of stages
2008-13 Beatriz Alarcón, Fabian Emmes, Carsten Fuhs, Jürgen Giesl, Raúl Gutiérrez, Salvador Lucas, Peter Schneider-Kamp, René Thiemann: Improving Context-Sensitive Dependency Pairs
2008-14 Bastian Schlich: Model Checking of Software for Microcontrollers
2008-15 Joachim Kneis, Alexander Langer, Peter Rossmanith: A New Algorithm for Finding Trees with Many Leaves
2008-16 Hendrik vom Lehn, Elias Weingärtner and Klaus Wehrle: Comparing recent network simulators: A performance evaluation study
2008-17 Peter Schneider-Kamp: Static Termination Analysis for Prolog using Term Rewriting and SAT Solving
2008-18 Falk Salewski: Empirical Evaluations of Safety-Critical Embedded Systems
2008-19 Dirk Wilking: Empirical Studies for the Application of Agile Methods to Embedded Systems
2009-02 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Quantitative Model Checking of Continuous-Time Markov Chains Against Timed Automata Specifications
2009-03 Alexander Nyßen: Model-Based Construction of Embedded Real-Time Software - A Methodology for Small Devices
2009-04 Daniel Klünder: Entwurf eingebetteter Software mit abstrakten Zustandsmaschinen und Business Object Notation
2009-05 George B. Mertzios, Ignasi Sau, Shmuel Zaks: A New Intersection Model and Improved Algorithms for Tolerance Graphs
2009-06 George B. Mertzios, Ignasi Sau, Shmuel Zaks: The Recognition of Tolerance and Bounded Tolerance Graphs is NP-complete

2009-07 Joachim Kneis, Alexander Langer, Peter Rossmanith: Derandomizing Non-uniform Color-Coding I

2009-08 Joachim Kneis, Alexander Langer: Satellites and Mirrors for Solving Independent Set on Sparse Graphs

2009-09 Michael Nett: Implementation of an Automated Proof for an Algorithm Solving the Maximum Independent Set Problem

2009-10 Felix Reidl, Fernando Sánchez Villaamil: Automatic Verification of the Correctness of the Upper Bound of a Maximum Independent Set Algorithm

2009-11 Kyriaki Ioannidou, George B. Mertzios, Stavros D. Nikolopoulos: The Longest Path Problem is Polynomial on Interval Graphs

2009-12 Martin Neuhäußer, Lijun Zhang: Time-Bounded Reachability in Continuous-Time Markov Decision Processes

2009-13 Martin Zimmermann: Time-optimal Winning Strategies for Poset Games

2009-14 Ralf Huuck, Gerwin Klein, Bastian Schlich (eds.): Doctoral Symposium on Systems Software Verification (DS SSV'09)

2009-15 Joost-Pieter Katoen, Daniel Klink, Martin Neuhäußer: Compositional Abstraction for Stochastic Systems

2009-16 George B. Mertzios, Derek G. Corneil: Vertex Splitting and the Recognition of Trapezoid Graphs

2009-17 Carsten Kern: Learning Communicating and Nondeterministic Automata

2009-18 Paul Hänsch, Michaela Slaats, Wolfgang Thomas: Parametrized Regular Infinite Games and Higher-Order Pushdown Strategies

2010-02 Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time

2010-03 Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering

2010-04 René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme

2010-05 Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme

2010-06 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata

2010-07 George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms

2010-08 Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting

2010-09 George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs

2010-10 Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut

2010-11 Martin Zimmermann: Parametric LTL Games

2010-12 Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut

2010-13 Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems

2010-14 Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination

2010-15 Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode

2010-16 Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles

2010-17 Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten

2010-18 Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit

* These reports are only available as a printed version.

Please contact biblio@informatik.rwth-aachen.de to obtain copies.