Evaluating LTL Satisfiability Solvers

Viktor Schuppan[1] and Luthfi Darmawan[2]

[1] Email: Viktor.Schuppan@gmx.de
[2] Email: luthfi@alumni.itb.ac.id

Abstract. We perform a comprehensive experimental evaluation of off-the-shelf solvers for satisfiability of propositional LTL. We consider a wide range of solvers implementing three major classes of algorithms: reduction to model checking, tableau-based approaches, and temporal resolution. Our set of benchmark families is significantly more comprehensive than those in previous studies. It takes the benchmark families of previous studies, which only have a limited overlap, and adds benchmark families not used for that purpose before.
We find that no solver dominates or solves all instances. Solvers focused on finding models and solvers using temporal resolution or fixed point computation show complementary strengths and weaknesses. This motivates and guides estimation of the potential of a portfolio solver. It turns out that even combining two solvers in a simple fashion significantly increases the share of solved instances while reducing CPU time spent.

1 Introduction

More and more, system specifications are not only used for classical verification of the correctness of a given system, e.g., via model checking [BK08], but they themselves become the subject of investigation (e.g., [Pil+06,Fis+08]). This is justified by observations in industry that many specifications contain errors (e.g., [Bee+01]) as well as by transition to property-based design (e.g., [prosyd]). Propositional Linear Temporal Logic (LTL) [Pnu77,Eme90] is a popular choice for system specifications and many checks on specifications reduce to determining (un)satisfiability (see, e.g., [Pil+06,Fis+08,RV10]). Hence, satisfiability of LTL is of considerable practical relevance.

A broad range of techniques for determining satisfiability of LTL has been developed: tableau-based methods (e.g., [Wol85,Jan99,Sch98]), temporal resolution (e.g., [FDP01]), and reduction to model checking (e.g., [RV10,Wul+08, Cim+07]). Despite the relevance of the problem and the range of techniques, we are not aware of a recent, comprehensive experimental comparison of solvers for satisfiability of propositional LTL on a broad set of benchmarks. In fact, the only line of work containing a representative from each of the above mentioned techniques that we know is the one by Hustadt et al. [HS01,HH01,HS02] (see below), which is somewhat dated.

In this paper we make the following contributions. 1. We perform an experimental evaluation of solvers for satisfiability of propositional LTL using `ALASKA` [alaska,Wul+08], `LWB` [lwb,Heu+95], `NuSMV` [nusmv,Cim+02], `pltl` [pltl], `TRP++`

[trp++,HK04], and `TSPASS` [tspass,LH10]. Both the range of techniques in the solvers we use and the set of benchmarks we collected are significantly more comprehensive than in any previous study we know. We have made our data available for further analysis [www]. 2. We consider number of solved instances, run time, memory usage, and model size. The analysis is greatly helped by using contour/discrete raw data plots, which complement the traditional cactus plots by preserving the relationship between benchmark instances. 3. The analysis shows complementary behavior between some solvers. This motivates estimating the potential of a portfolio solver. We consider portfolio solvers without communication between members of the portfolio for a best case scenario (which is unrealistic) and a reference case scenario (which any portfolio solver should aim to beat). Finally, we show that even a trivially implementable solver that sequentially executes one solver first for a short amount of time and, if necessary, then invokes another solver reduces the number of unsolved instances as well as the average run time.

Related Work Rozier and Vardi compare several explicit state and symbolic BDD-based model checkers for LTL satisfiability checking [RV10]. They find the symbolic tools to be superior in terms of performance and, generally, also in terms of quality. They do not consider SAT-based bounded model checkers, tableau-based solvers, or temporal resolution. While they perform an in-depth comparison of solvers using very similar techniques, our focus is on comparing selected representatives of a broad variety of techniques. We also use more benchmark families and consider memory usage and model size. The same authors compare symbolic constructions of Büchi automata in [RV11] using the BDD-based engine of `Cadence SMV` as backend solver. They show that a portfolio approach to automata construction is advantageous. De Wulf et al. compare `NuSMV` and `ALASKA` [Wul+08]. For a detailed discussion see Sect. 6. Hustadt et al. perform several comparisons [HS01,HH01,HS02] of `TRP`, a version of `LWB`, and a version of `SMV` on the **trp** benchmark set (see Sect. 4). Goré and Widmann perform an experimental comparison of solvers for CTL [GW10]. Goranko et al. [GKS09] compare an implementation of Wolper's tableau construction with `pltl`. For references on solver competitions and on their methodology see App. A.

We are not aware of previous work on portfolio approaches to LTL satisfiability, except for [RV11]. We use entire solvers as members of a portfolio, while [RV11] uses different frontends for Büchi automata construction all relying on the same BDD-based backend solver. For other problem classes see, e.g., [HLH97] (graph coloring, web browsing), [LB+03] (winner determination problem), [GS01] (constraint satisfaction, mixed integer programming), [Xu+08] (SAT), or [PT09,SM07] (QBF).

Organization In Sect. 2 we introduce notation. In Sect. 3, 4, and 5 we describe solvers, benchmarks, and methodology. Section 6 contains the results of our evaluation. An estimation of the potential of a portfolio solver follows in Sect. 7. Section 8 concludes. Due to space constraints the following parts are in appendices: general concepts and terminology (App. A), details on our benchmark set (App. B), discussion (App. C), and some plots (App. D).

2 Preliminaries

We consider formulas in future time propositional LTL with temporal operators **F**, **G**, **R**, **U**, **X**. We assume familiarity with LTL; otherwise see [Eme90].

The terminology we use is largely standard (e.g., [SB05,Ber+09]); a reader unfamiliar with competition terminology is referred to App. A. A somewhat non-standard term we use is *configuration*, which denotes a tool (solver) with specific option values. A tool is a *state-of-the-art contributor* (sota) if an instance is solved only by configurations of that tool (see also [SS01]). Given a set of configurations C the *virtual best solver* (vbs) is the hypothetical solver using the best configuration in C on any given instance (e.g., [Ber+09]). We use **bold font** for sets of benchmark instances and `teletype` for configurations.

3 Solvers

Choice of Solvers We consider tools to solve satisfiability of propositional LTL from 3 major classes of approaches: 1. reduction to model checking, 2. tableau-based algorithms, and 3. temporal resolution. Tools were chosen as detailed below. To the best of our knowledge this set of solvers is the most diverse considered in an evaluation of solvers for satisfiability of propositional LTL to date.

Reduction to Model Checking We chose `ALASKA` [alaska,Wul+08] and `NuSMV` [nusmv,Cim+02] using BDDs (`NuSMV-BDD`) and SAT (`NuSMV-SBMC`). We ruled out explicit state model checkers, as they did not scale as well as BDD-based symbolic model checkers for LTL satisfiability in [RV10]. The BDD-based engine of `Cadence SMV` [csmv] performed comparable to `NuSMV-BDD` in [RV10]. `sal-smc` [Mou+04] constructs explicit Büchi automata and was found not to scale [RV10]. The BDD-based variant of `VIS` [The96] uses explicit construction of Büchi automata; initial experiments confirmed that this does not scale for satisfiability of LTL. `sal-bmc` [Mou+04] can only prove safety properties [Mou04]. For an alternative using SAT-based symbolic model checking we contacted the VIS group for advice on recommended configurations (the space of configurations is quite large), but have not received an answer yet. Finally, we checked the publicly available versions of the participants of HWMCC'10 [BC10]; as far as we could see, the solvers that are not included in our study only handle safety properties.

Tableau-Based Algorithms We chose `LWB` [lwb,Heu+95] and `pltl` [pltl]. `TWB` [AG07] is superseded by `pltl` [Gor10]. `LTL Tableau` turns out to be inferior to `pltl` [GKS09].

Temporal Resolution We chose `TRP++` [trp++,HK04] and `TSPASS` [tspass,LH10]. An alternative tool is `TeMP` [Hus+04]. `TeMP` was shown to be inferior to `TRP++` on propositional problems in [Hus+04] and comparable to `TSPASS` on monodic problems in [LH10]. Note, that `TSPASS` is fair, while `TeMP` is not [LH09a].

Solver Descriptions Below we briefly describe the tools we consider as well as the set of their options that we take into account. Note that not all combinations of options are valid. Due to space constraints the descriptions have to be kept short, and we refer the reader to the respective tool documentation.

`ALASKA` performs model checking and satisfiability checking of LTL via symbolic computation of fixed points using antichains [alaska,Wul+08]. Relevant options are: `noc/c` dis-/enables model construction, `nos/s` uses a semisymbolic/fully symbolic algorithm, and `nob/b` switches between forward and backward image computation. We use version 0.4 with an additional patch by N. Maquet.

`LWB` [lwb,Heu+95] implements tableau-based algorithms for LTL by Janssen [Jan99] (no model construction) in the function "satisfiable" and by Schwendimann [Sch98] (model construction) in the function "model". Neither has relevant options. We designate the former by `sat` and the latter by `mod`. We use version 1.1.

`NuSMV-BDD` In this evaluation we treat `NuSMV` [nusmv,Cim+02] as two tools `NuSMV-BDD` and `NuSMV-SBMC`. `NuSMV-BDD` performs symbolic model checking of LTL using symbolic fixed point computation with BDDs [CGH97]. Experience with `NuSMV-BDD` allows us to restrict experiments to the following options. `nodcx/dcx` en-/disables model construction, `nofflt/fflt` dis-/enables forward computation of reachable states in the model and tableau for the LTL formula, `nodyn/dyn` dis-/enables dynamic reordering, and `elbwd/elfwd` switches between backward and forward image computation in the Emerson-Lei algorithm [EL86, HKQ03]. We use version 2.5.0.

`NuSMV-SBMC` performs incremental simple bounded model checking [HJL05] of LTL using MiniSat [minisat]. Options considered are `nodcx/dcx` to en-/disable model construction and `noc/c` to dis-/enable checking completeness. With the latter disabled `NuSMV-SBMC` cannot solve *unsat* instances. We use version 2.5.0.

`pltl` [pltl] implements tableau-based algorithms for LTL along the lines of [GW09] via the command line argument "graph" and by Schwendimann [Sch98] via the command line argument "tree". Neither has model construction or relevant options. We designate the former by `graph` and the latter by `tree`. We use version r1424.

`TRP++` [trp++,HK04] uses temporal resolution for LTL [FDP01]. Relevant options: `nos/s` to dis-/enable simplification, `nor/r` to dis-/enable rewriting, `noal/al` to ex-/include an order statement, `dfs/bfs` to choose dfs/bfs in loop search, `nop/p` to dis-/enable pre-test for sometime resolution, and `nofsr/fsr` to dis-/enable forward subsumption resolution. `TRP++` cannot construct models. We use v. 2.x.

`TSPASS` [tspass,LH10] is a temporal resolution solver for monodic first-order temporal logic with model construction for propositional LTL [LH09b]. We consider `noext/ext` to dis-/enable extended step clauses, `nogrp/grp` to dis-/enable regrouping of $\mathbf{X}$, `nosev/sev` to dis-/enable transforming multiple eventualities into a single one, `log/sub` to select logical equivalence or subsumption in loop tests, `nosls/sls` to dis-/enable sequential loop search, `norfmrr/rfmrr` (resp. `norbmrr/rbmrr`) to dis-/enable forward (resp. backward) matching replacement resolution, `nomod/mod` to dis-/enable model construction, and `mur/mor` to select unordered or ordered resolution in model construction. We use version 0.94-0.16.

4 Benchmarks

In Tab. 1 we give an overview of the benchmark families we use. To our knowledge this set of benchmarks is the most comprehensive used for evaluating propositional LTL satisfiability solvers so far. [RV10] used **rozier_counter**, **rozier_pattern**, and **rozier_formulas**. [Wul+08] used **alaska_lift**, **alaska_szymanski**, and subsets of **rozier_counter** and **rozier_formulas**. [HS02] used **trp**. Note that there is little overlap. [RV10, Wul+08] and [HS02] represent separate communities. We added the following benchmark families that, to our knowledge, had not been used to evaluate solvers for propositional LTL satisfiability before: **acacia**, **anzu**, and **forobots**.[3] To provide more challenging instances we scaled up some families. Moreover, for the families **acacia_demo-v3**, **anzu_amba**, and **anzu_genbuf**, which consist of a set of assumptions and a set of guarantees, we not only used the form $(\bigwedge_i a_i) \rightarrow (\bigwedge_i g_i)$ but also $(\bigwedge_i a_i) \wedge (\bigwedge_i g_i)$ (marked by "c" in the family name). For **acacia_demo-v3**, **alaska_lift**, **anzu_amba**, and **anzu_genbuf** we added variants with liveness conditions to trigger nontrivial behavior (marked by "l" in the family name). For **alaska_lift** we also use a fixed [Sch10] variant (marked by "f" in the family name). Finally, we added the families **schuppan_O1formula**, **schuppan_O2formula**, and **schuppan_phltl**. Our set of benchmarks contains 3723 instances. All benchmarks are available from [www].

5 Methodology

Hardware and Software We used machines with Intel Xeon 3.0 GHz processors and 4 GB memory running Red Hat Linux 5.4 with 64 bit kernel 2.6.18-164.2.1.el5. Run time and memory usage were measured with `run` [run].

Input Format and No Shuffling We converted all instances into `NuSMV` format and from there to the input formats of the other tools. We did not syntactically alter instances as there was no risk of cheating by syntactic recognition of benchmarks (e.g., [BS03]) and we, too, think that syntactic information should be preserved for the benefit of solvers (e.g., [SB05]).

Stages The valid option combinations of the options in Sect. 3 yield the following number of configurations (model construction dis-/enabled): `ALASKA` 4/2, `LWB` 1/1, `NuSMV-BDD` 6/4, `NuSMV-SBMC` 2/2, `pltl` 2/-, `TRP++` 64/-, `TSPASS` 128/128.

The number of configurations of `TRP++` and `TSPASS` is too large to include all of them in the main stage of our evaluation. We therefore performed a preliminary stage with a time limit of 10 seconds and a memory limit of 2 GB on a representative subset of instances. In that stage we used all 64 combinations of `TRP++`. For `TSPASS` we considered the following subset of configurations: all options at their default value (sometimes implied by other options) as well as a single option switched to its non-default value. This resulted in 24/24 configurations. We then fixed options that either had a clear benefit one way or

[3] While the full version of [RV11] uses **acacia** and **anzu**, these were included based on a previous submission of this paper that we made available to the authors of [RV11].

family	max. size	num. *sat*	num. *unsat*	num. *unkn.*	source	description
application						
acacia_demo-v22	76	10	–	–	[acacia,FJR09, JB06]	window screens
acacia_demo-v3	426	36	–	–	[acacia,FJR09, JB06]	arbiters (scaled up, added variants)
acacia_example	144	25	–	–	[acacia,FJR09, JB06]	mostly arbiters and traffic light controllers
alaska_lift	4450	102	34	–	[alaska,Wul+08, Har05]	lifts (scaled up, added variants, added fixes [Sch10])
alaska_szymanski	183	4	–	–	[alaska,Wul+08, STV05]	mutual exclusion protocols
anzu_amba	6173	43	–	8	[anzu,Blo+07a]	microcontroller buses (scaled up, added variants)
anzu_genbuf	5805	48	–	12	[anzu,Blo+07b]	generalized buffers (scaled up, added variants)
forobots	636	14	25	–	[BDF09]	foraging robots
crafted						
rozier_counter	751	78	–	–	[rozier,RV10]	serial counters (long models)
rozier_pattern	7992	244	–	–	[rozier,RV10, GH06]	patterns to test explicit state model checkers (scaled up)
schuppan_O1formula	4007	–	27	–	(new)	patterns that trigger exponential behavior in some solvers
schuppan_O2formula	6001	–	15	12		
schuppan_phltl	40501	–	10	8	(new)	temporal formulation of pigeonhole principle [Bie+09]
random						
rozier_formulas	185	1943	57	–	[rozier,RV10]	random formulas as in [DGV99] (subset of original family)
trp	1422	573	397	–	[trp,HS02]	random formulas from fixed conjunctive normal form templates (subset of original family)

Table 1. Overview of benchmark families, grouped by benchmark categories. The first column lists the name of the family. Columns 2 – 5 show the size (see App. A) of the largest instance and the number of *sat*, *unsat*, and *unknown* instances, respectively, in that family. The 6th column provides references to the source and the 7th column gives a brief description.

the other or clearly had little effect to the corresponding values and kept the remaining configurations for the main stage (see Sect. 6). In the main stage all configurations of `ALASKA`, `LWB`, `NuSMV-BDD`, `NuSMV-SBMC`, and `pltl` as well as the remaining configurations of `TRP++` and `TSPASS` were run with a time limit of 60 seconds and a memory limit of 2 GB.

In each stage, each configuration was run only once on each instance. While performing more than one run would provide more accurate information about run time distributions [Nik10] performing only a single run allows to use more configurations, more instances, or higher time bounds with equal resources.

Tracks We have two tracks: one for configurations with model construction disor enabled (e.g., `LWB` using `mod` constructs models but is superior to `sat` that doesn't) and one for configurations with model construction enabled. The former considers all instances; the latter is restricted to *sat* instances.

Correctness of Solvers is a recurring issue in tool competitions and comparisons (e.g., [RV10]). Besides obvious cross checking of the *sat*/*unsat* results reported by different configurations for the same instance we used the fact that `NuSMV-SBMC` produces shortest (possibly plus one) models as an additional correctness check. We did not perform further validation of generated models.

Scoring We essentially use scoring based on a higher number of solved instances and lower time taken on solved instances (see Sect. 2) as it preserves and clearly shows what we consider two important performance indicators.

However, there are fairly big differences in the number of instances in our benchmark families. Still, we would like to consider many benchmarks rather than only sampling the larger families. Hence, we modify the above scoring method as follows. We consider the benchmark families as a tree. We then compute the share of solved instances and the average run time on solved instances for each leaf (here all instances have equal weight). Then, for each non-leaf node, aggregate values are computed as averages with equal weights for all children of that node. For the tree of families see App. B.2.

6 Results

For more plots and data see App. D and the website [www].

Preliminary Stage For `TRP++` configurations with `s_nor` proved inferior so that only `s_r`, `nos_r`, and `nos_nor` were kept. The effects of `noal`/`al`, `dfs`/`bfs`, and `nofsr`/`fsr` are unclear; hence all combinations were kept. `nop`/`p` had little effect so that we set it to its default `nop`. All in all this left us with 24 configurations.

For `TSPASS` `ext`, `nosev`, `sub`, and `mor` turned out to be advantageous. The effects of `nogrp`/`grp`, `norfmrr`/`rfmrr`, and `norbmrr`/`rbmrr` are unclear and we kept all. `nosls`/`sls` had little effect so that we disabled it as is default. This resulted in 8 configurations each with model construction disabled and enabled.

We now move to the main stage.

Correctness of Solvers We found no bug in `pltl` but 1 or 2 bugs in each of `NuSMV`, `ALASKA`, `TRP++`, and `TSPASS`. All of them were kindly fixed by the respective tool authors. As of now we are not aware of wrong results or bugs triggered in the above tools by our benchmark set. In `LWB` we found several bugs. We emailed our findings to the developers but have not received a response. There are currently 187 out of 7446 (non-negated and negated) instances known to us that trigger bugs in `LWB`; 13 are wrong results. Hence, `LWB` is hors-concours. Some large instances failed in `ALASKA` and `TSPASS` due to certain built-in limits. These instances were rerun with increased limits.

Selecting Winning Configurations per Tool To focus the subsequent comparison we select one winning configuration per tool to be used for the comparisons between tools in the remainder of this section. We choose the configuration with the highest weighted share of solved instances (see Sect. 5) for each tool. We distinguish between model construction dis- or enabled and model construction enabled as model construction is not available for some tools or options.

Table 2 provides a summary. For all tools except `NuSMV-BDD` and `LWB` the weighted share of instances solved by the winning configurations is close to that of the vbs of all configurations of that tool (Tab. 2). Below we mostly restrict the analysis to the winning configurations. We use the tool name to identify the respective winning configurations.

Track Model Construction Disabled In Fig. 1 we show *contour/discrete raw data plots* of the run time for the winning configurations with model construction dis- or enabled. The name is taken from [Sta]. A somewhat related way to display results of a solver competition was used in Pseudo-Boolean Competitions [pbc10].

	model construction dis- or enabled (all instances)				model construction enabled (*sat* instances)			
tool	winning configuration	max	min	vbs	winning configuration	max	min	vbs
ALASKA	noc_nos_nob	0.581	0.322	0.595	c_nos_nob	0.595	0.318	0.595
LWB	mod	0.740	0.656	0.800	mod	0.795	0.795	0.795
NuSMV-BDD	dcx_fflt_dyn_elbwd	0.743	0.607	0.823	nodcx_fflt_dyn_elbwd	0.754	0.625	0.771
NuSMV-SBMC	nodcx_c	0.723	0.651	0.726	nodcx_noc	0.860	0.857	0.861
pltl	tree	0.694	0.687	0.702	—	—	—	—
TRP++	s_r_noal_bfs_nop_fsr	0.752	0.593	0.776	—	—	—	—
TSPASS	ext_nogrp_nosev_sub_nosls_rfmrr-_norbmrr_nomod_mor	0.667	0.479	0.670	ext_grp_sev_sub_nosls_rfmrr-_rbmrr_mod_mor	0.531	0.495	0.538

Table 2. Selecting a winning configuration per tool (separately for tracks). The leftmost column lists the tool name. Next come 2 groups of 4 columns. The 1st group is for configurations with model construction dis- or enabled, the 2nd with model construction enabled. In each group the 1st column shows the winning configuration per tool. The 2nd column shows its score, the 3rd column shows the worst score, and the 4th column shows the score of the vbs of all configurations of that tool.

Contrary to cactus plots contour/discrete raw data plots retain the relationship between instances (one x-coordinate corresponds to the same rather than different instances) but are more legible than line plots. They allow to see the performance of the solvers on benchmark families that are a subfamily of the one comprising a plot. A particular advantage is that they permit identification of similar and complementary behavior in performance. They also allow to see how difficult a particular instance or subfamily is. However, these plots make it harder to determine a ranking of solvers by higher number of solved instances with ties broken by lower average time taken on solved instances. Due to space constraints we cannot show both kinds of plots for the same data. We chose to use the contour/discrete raw data plots here to demonstrate their utility. For corresponding cactus plots see Fig. 10–13 in App. D.2.

Overall Picture In this paragraph we refer to all configurations used in the main stage. No configuration solves all instances. 8–12 instances in **anzu_amba**, **anzu_genbuf**, **schuppan_O2formula**, and **schuppan_phltl** remain unsolved. The instances in the former two families are expected to be *sat*, in the latter *unsat*. The smallest unsolved instance is **O2formula50** (size 301). `NuSMV-BDD` is a sota on a number of (*unsat*) instances in **alaska_lift** and **schuppan_O2formula**; `NuSMV-SBMC` on instances in **alaska_lift**, **anzu_amba**, and **anzu_genbuf** (all *sat*); `TRP++` on instances in **rozier_counter** (*sat*); `LWB` on instances in **schuppan_phltl** (*unsat*). See also Fig. 8 in App. D.1.

Families The majority of benchmark families contain instances that are challenging for some solver. In category **application** the 3 families with larger instances, **alaska_lift**, **anzu_amba**, and **anzu_genbuf**, are the more difficult ones. Among them the variants that were modified to trigger meaningful behavior are the hardest. In category **crafted** the (*unsat*) families **schuppan_O2formula** and **schuppan_phltl** are the most difficult. **rozier_counter** is hard for most solvers, except for `TRP++` and `TSPASS` (and `NuSMV-BDD` in a configuration using only backward fixed point computation). The two families in category **random** show very different pictures. Family **rozier_random** is solved well by non-resolution-based tools but somewhat more difficult for `TRP++` and `TSPASS`; roles are reversed in family **trp**. Note that **trp** comes from the temporal resolution community, while **rozier_random** is taken from the model checking community.

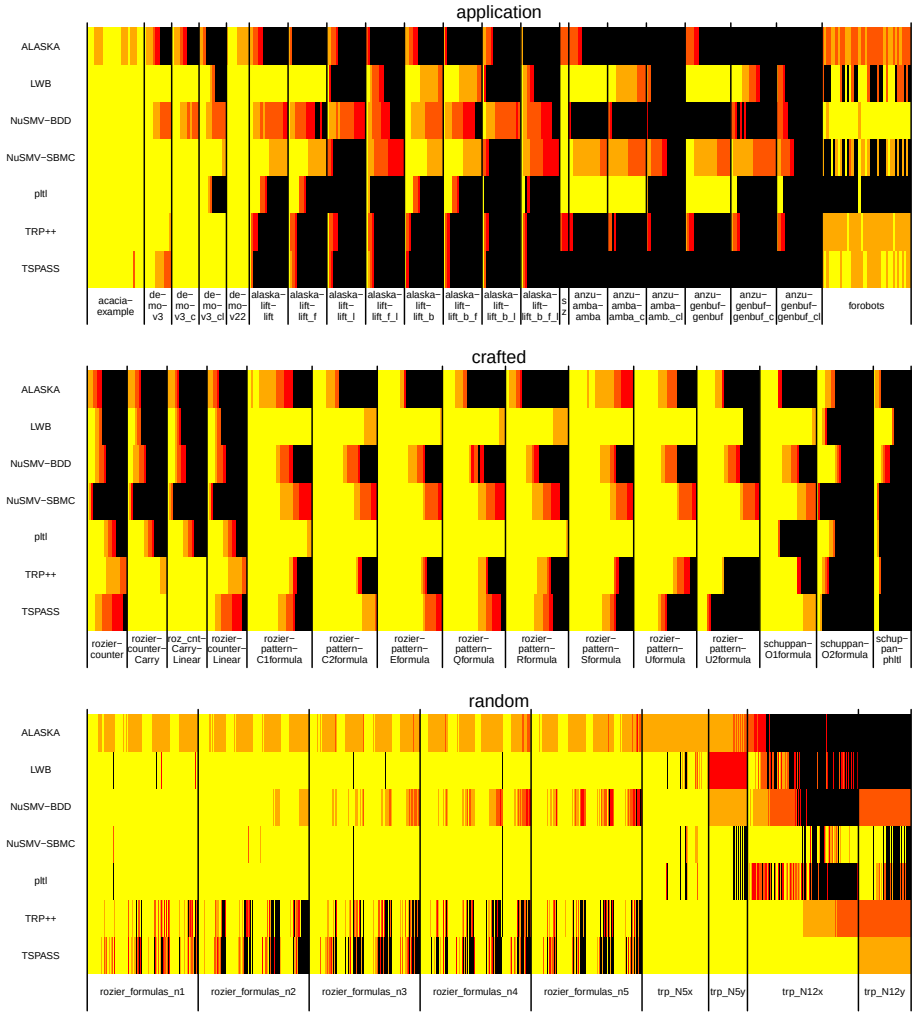

Fig. 1. Contour/discrete raw data plots of run time for winning configurations with model construction dis- or enabled (all instances). Instances are on the x-axes (only identified by their families), configurations on the y-axes. Each rectangle represents the run time of one configuration on one instance. **sz** abbreviates **alaska_szymanski**, **roz_cnt** abbreviates **rozier_counter**, and **demo** stands for **acacia_demo**. Run times are encoded using the following colors:
≤ 0.1 sec; > 0.1 sec, ≤ 1 sec; > 1 sec, ≤ 10 sec; > 10 sec, ≤ 60 sec; unsolved.

Solvers: Similarities and Differences Figure 1 shows that `TRP++` and `TSPASS`, which both use temporal resolution, have similar strengths and weaknesses. `TSPASS` tends to improve over `TRP++` on **trp**, while `TRP++` tends to be faster on most of the remaining families. Between the two tools using symbolic fixed

point computation NuSMV-BDD mostly dominates ALASKA; the latter has a higher start up time than the other tools. The strengths and weaknesses of NuSMV-BDD mostly resemble those of TRP++ and TSPASS. Intuitively, symbolic fixed point computation [EL86] is closer in spirit to temporal resolution as performed in TRP++ [HK04] than to searching models (stating a more formal relationship is left as future work). LWB, NuSMV-SBMC, and pltl display similar characteristics. Note that these solvers essentially try to find models, although NuSMV-SBMC uses a fairly different technique than pltl and LWB. It is important to note that the strengths and weaknesses of NuSMV-BDD, TRP++, and TSPASS are somewhat complementary to those of LWB, NuSMV-SBMC, and pltl.

Sat versus Unsat Instances NuSMV-SBMC exhibits the largest difference in its behavior between *sat* and *unsat* instances. NuSMV-SBMC solves most *sat* instances among the solvers. A notable exception is **rozier_counter**, which has shortest models of exponential size; few shortest models outside **rozier_counter** have size larger than 3 (see below). On the contrary, NuSMV-BDD and ALASKA, which are based on symbolic fixed computation, are hardly affected. For plots see Fig. 14–17 in App. D.3 and Fig. 18–21 in App. D.4.

Instance Size The two tools based on symbolic fixed point computation, ALASKA and NuSMV-BDD, show a fairly clear influence of the size of an instance on their run time. At the other end of the spectrum are LWB and pltl, trying to find models. They solve some large instances in almost no time. For plots see Fig. 22–25 in App. D.5.

Non-negated versus Negated Instances The relevance of negated versions of instances is questionable. We have not included negated versions of instances in any part of this paper, except where stated explicitly. However, we briefly comment on one aspect because of the size of the observed effect. On the **rozier_formulas** family — where negation should not change any relevant characteristic of the benchmark set — the variation in performance between the non-negated and the negated version of an instance is considerably higher for TSPASS and TRP++ than for NuSMV-BDD and ALASKA. For scatter plots see Fig. 26 in App. D.6.

Memory Memory usage turned out to be less of a problem than time taken, therefore we do not report detailed results. In fact, very rarely a configuration used more than 300 MB when it solved an instance. ALASKA typically used most memory. For plots see App. D.

VBS rather than Winning Configurations While the findings above were mostly stated for the winning configurations of each tool, the picture does not change significantly when comparing the vbs of each tool (for plots see App. D). As suggested by Tab. 2 notable improvements only happen for NuSMV-BDD, LWB, and, to a lesser extent, TRP++.

Track Model Construction Enabled We focus on model size. Figure 2 shows a cactus plot for the winning configurations with model construction enabled (*sat* instances). A vbs of all configurations with model construction enabled solves all but the largest instances of **anzu_amba**, **anzu_genbuf**, and **rozier_counter**.

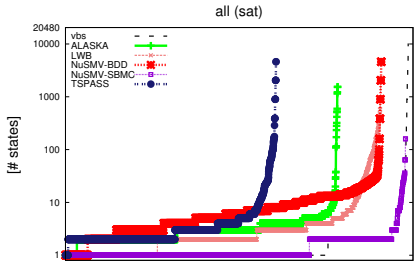

Fig. 2. Cactus plot of model size for the winning configurations with model construction enabled (*sat* instances).

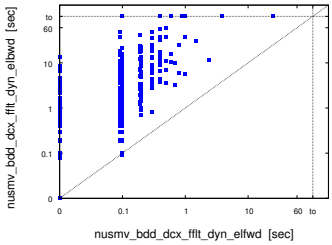

Fig. 3. Scatter plot comparing run time for the forward and the backward version of the Emerson-Lei algorithm in `NuSMV-BDD` on the **rozier_formulas** family. "to" marks *time-out*.

`NuSMV-BDD` is a sota based on instances in **rozier_counter**; `NuSMV-SBMC` on instances in **alaska_lift**, **anzu_amba**, **anzu_genbuf**, and **rozier_pattern**; `LWB` on instances in **rozier_pattern**.

95 % of the satisfiable instances have shortest models of size 3 or less. Instances with shortest models of size larger than 11 are either from **rozier_counter** or from the variants in **application** modified to trigger meaningful behavior.

`NuSMV-SBMC` mostly produces shortest models, while `NuSMV-BDD` produces the longest ones. On the other hand, `NuSMV-BDD` solves more instances of the **rozier_counter** family (which has very long models) than the other tools.

A Performance Advantage of `ALASKA` over `NuSMV-BDD`? In [Wul+08] De Wulf et al. perform a comparison between `ALASKA` and `NuSMV-BDD` for satisfiability and model checking of LTL. For LTL satisfiability they find that `ALASKA` outperforms `NuSMV-BDD` on **alaska_lift**, **alaska_szymanski**, and a subfamily of **rozier_formulas**, while `NuSMV-BDD` performs better on **rozier_counter**.

A comparison of the antichain-based algorithm in `ALASKA` [Wul+08] and the Emerson-Lei algorithm [EL86] used in `NuSMV-BDD` shows that the algorithm in [Wul+08] computes fixed points using forward image computation, while `NuSMV-BDD` up to version 2.4.3 only uses (as is common) backward image computations for [EL86]. This triggered us to implement a forward version (e.g., [HKQ03]) of the Emerson-Lei algorithm in `NuSMV-BDD`. Figure 3 shows that the forward version performs considerably better than the backward version on the **rozier_formulas** family. Using forward image computation `NuSMV-BDD` outperforms `ALASKA` on **rozier_formulas**. Note also that `ALASKA` can be switched to perform backward image computation in which case its performance degrades considerably.

Our evaluation shows that `NuSMV-BDD` can solve the **alaska_lift** and **alaska_szymanski** families easily (and faster than `ALASKA`) by restricting computation to reachable states (`fflt`) and enabling dynamic reordering (`dyn`).

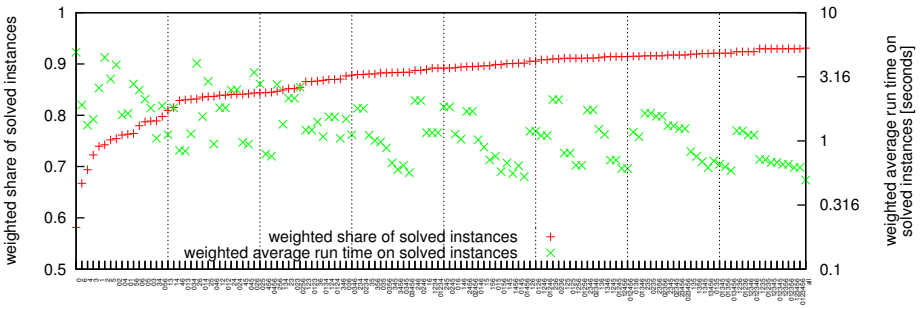

Fig. 4. Potential of a portfolio solver consisting of subsets of the winning configurations with model construction dis- or enabled using a perfect oracle. Portfolios are identified by their constituent configurations: 0: `ALASKA`; 1: `LWB`; 2: `NuSMV-BDD`; 3: `NuSMV-SBMC`; 4: `pltl`; 5: `TRP++`; 6: `TSPASS`. On the x-axis are the portfolios sorted in increasing order of weighted share of solved instances; ties are broken by decreasing order of weighted average run time on solved instances. For each portfolio the weighted share of solved instances is marked by a red vertical/horizontal cross (scale on the left y-axis); the corresponding weighted average run time on solved instances is marked by a green diagonal cross (scale on the right y-axis). "all" considers all configurations with model construction dis- or enabled rather than only the winning configurations. For an enlarged plot see App. D.8.

7 Potential of a Portfolio Solver

In the previous section we saw that some configurations behave complementarily. This motivates constructing *portfolio solvers* that consist of a set of configurations with the goal that the resulting solver performs better than any of its constituent configurations (e.g., [HLH97]). Different modes of execution are considered for portfolio solvers in the literature (e.g., [HLH97,LB+03,GS01, Xu+08]).

Perfect Oracle We assume an oracle that for each instance predicts (using no time and memory) an optimal configuration in a portfolio and then executes that configuration on that instance (see, e.g., [LB+03]). I.e., the performance of a portfolio solver on an instance is determined by the performance of an optimal configuration in a portfolio on that instance. If configurations do not collaborate (e.g., by exchanging partial results) that is a bound on the performance of a practical solver using that portfolio. An alternative view of this mode of execution is that each member of the portfolio is run on a separate processor in parallel until one configuration finishes while taking into account only the cost of one processor and disregarding the cost of other processors.

We estimate the potential of such a portfolio solver by considering all portfolios consisting of subsets of winning configurations with model construction dis- or enabled from Tab. 2. Figure 4 shows the result.

While individual configurations solve at most a weighted share of 0.752, using a portfolio helps to solve up to 0.931. All portfolios that solve a weighted share of 0.866 or more contain at least one of `ALASKA`, `NuSMV-BDD`, `TRP++`, and `TSPASS` and

at least one of LWB, NuSMV-SBMC, and pltl. All that solve 0.9 or more contain at least one of LWB and NuSMV-SBMC and at least one of TRP++ and TSPASS. The 4 best portfolios with two configurations are (LWB, TRP++), (LWB, TSPASS), (NuSMV-SBMC, TRP++), and (NuSMV-SBMC, TSPASS). Adding ALASKA to a portfolio that contains NuSMV-BDD does not help in most cases.

Perfect Task Switcher We now assume that all configurations of a portfolio are executed on a single processor in a time-sharing fashion with equal and infinitely small time slices, no task switching overhead, and memory usage not an issue (e.g., [HLH97]). I.e., rather than assuming a perfect oracle, we only assume a perfect task switcher. Now the performance of a portfolio solver with k configurations on an instance is determined by the performance of an optimal configuration in a portfolio on that instance multiplied by k (that might induce *time-out* even if some configuration solves the instance). If configurations do not collaborate this can be considered a portfolio solver that any practical portfolio solver using that portfolio should aim to beat. An alternative view is that each portfolio member runs on a separate processor in parallel until one member finishes and taking into account the cost for all processors.

For a plot analogous to Figure 4 see App. D.8. Here the best portfolio considered is (LWB, NuSMV-BDD, NuSMV-SBMC, TRP++), which solves a weighted share of 0.922. Otherwise, similar remarks as for the case of a perfect oracle apply.

Fast Presolver We now show that even a simplistic portfolio solver (implementable as shell script) can yield considerable benefits. We take the 4 best 2-configuration portfolios from above and use one of the two solvers as *fast presolver* [Xu+08] by executing it until it either solves an instance or reaches its (short) time limit. If the instance is not yet solved, then we execute the other solver for the remaining time (60 seconds minus the time limit of the presolver).

Results are shown in Tab. 3. In each case the portfolios using a fast presolver significantly increase the weighted share of solved instances while decreasing the weighted average run time over the respective portfolio members in isolation.

8 Conclusion

Benchmarks and data from our evaluation, available at [www], identify reference solvers with their command line options at the level of benchmark instances. This helps to improve existing solvers, provides a point of reference in the evaluation of new techniques, and can serve as a basis for developing heuristics for portfolio solvers. Our evaluation shows that solvers have different, complementary strengths and weaknesses. We do not declare any solver to be the winner (those who disagree are referred to Tab. 2). Instead, for a solver aiming to be competitive on a broad range of benchmarks we advocate a portfolio approach.

Acknowledgements J.-F. Raskin and N. Maquet for help with ALASKA and hosting the 1st author for 1 week. R. Goré and F. Widmann for help with pltl. B. Konev and M. Ludwig for help with TRP++ and TSPASS. C. Dixon for the **forobots** family. B. Jobstmann and G. Hofferek for help with the **amba** family. K. Rozier for feedback. A. Artale for supervising the 2nd author's MSc thesis. The ES group at FBK,

	1st in isolation		2nd in isolation		perfect oracle		perf. task switcher		1st as fast presolver 1 second		1st as fast presolver 2 seconds		2nd as fast presolver 1 second		2nd as fast presolver 2 seconds	
	share	time	share	time	share	time	share	time	share	time	share	time	share	time	share	time
(LWB, TRP++)	0.740	2.59	0.752	3.03	0.896	0.89	0.894	1.12	0.880	1.09	0.885	1.30	0.841	1.26	0.850	1.45
(LWB, TSPASS)	0.740	2.59	0.667	1.91	0.889	1.16	0.881	1.27	0.868	0.88	0.874	1.10	0.850	1.20	0.858	1.48
(NuSMV-SBMC, TRP++)	0.723	1.47	0.752	3.03	0.880	1.11	0.874	1.37	0.823	1.03	0.841	1.18	0.860	0.97	0.862	1.31
(NuSMV-SBMC, TSPASS)	0.723	1.47	0.667	1.91	0.867	1.41	0.853	1.60	0.813	1.00	0.831	1.21	0.837	1.17	0.840	1.42

Table 3. Performance of the 4 best 2-configuration portfolios in various execution modes. After the portfolio members in the 1st column there are 8 groups of 2 columns. In each group the 1st column shows the weighted share of solved instances, the 2nd column shows the weighted average run time on solved instances in seconds. The 1st and 2nd column groups are for the 1st and 2nd member of each portfolio in isolation. The 3rd and 4th groups are for perfect oracle and perfect task switcher modes. The 5th and 6th groups are for fast presolver mode with 1 and 2 seconds time limit when the 1st member of the portfolio is used as a fast presolver; the 7th and 8th groups are analogous for the 2nd member as a fast presolver. The time limits of 1 and 2 seconds were chosen among some that we tried as they represent a sweet spot that exhibits both an increase in weighted share of solved instances and a decrease in weighted average run time on solved instances.

esp. A. Cimatti, A. Mariotti, and M. Roveri, for discussion and support. The Provincia Autonoma di Trento (project EMTELOS) for financial support of the 1st author. The European Master's Program in Computational Logic for financial support of the 2nd author.

References

[acacia] http://www.antichains.be/acacia/. See p. 6.

[AG07] P. Abate and R. Goré. "The Tableau Workbench". In: *M4M*. 2007, pp. 55–67. Links: ee, Google Scholar. See p. 3.

[alaska] http://www.antichains.be/alaska/. See pp. 1, 3, 4, 6.

[anzu] http://www.iaik.tugraz.at/content/research/design_verification/anzu/. See p. 6.

[BC10] A. Biere and K. Claessen. "Hardware Model Checking Competition (presentation, slides only)". In: *Hardware Verification Workshop 2010, Edinburgh, UK, July 15, 2010*. 2010. Available from http://fmv.jku.at/biere/talks/Biere-HWMCC10-talk.pdf. See p. 3.

[BDF09] A. Behdenna, C. Dixon, and M. Fisher. "Deductive Verification of Simple Foraging Robotic Behaviours". In: *International Journal of Intelligent Computing and Cybernetics* 2.4 (2009), pp. 604–643. Links: ee, Google Scholar. See p. 6.

[Bee+01] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in Temporal Model Checking". In: *Formal Methods in System Design* 18.2 (2001), pp. 141–163. Links: Google Scholar. See p. 1.

[Ber+09] D. Le Berre, O. Roussel, L. Simon, A. Goerdt, I. Lynce, and A. Stump. "The SAT 2009 competition results: does theory meet practice? (presentation, slides only)". In: *SAT*. Ed. by O. Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009. ISBN: 978-3-642-02776-5. Available from http://www.satcompetition.org/2009/sat09comp-slides.pdf. See p. 3.

[Bie+09] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009. ISBN: 978-1-58603-929-5. Links: Google Scholar. See p. 6.

[BK08] C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 2008. Links: Google Scholar. See p. 1.

[Blo+07a] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. "Automatic hardware synthesis from specifications: a case study". In: *DATE*. Ed. by R. Lauwereins and J. Madsen. ACM, 2007, pp. 1188–1193. ISBN: 978-3-9810801-2-4. Links: ee, Google Scholar. See p. 6.

[Blo+07b] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. "Specify, Compile, Run: Hardware from PSL". In: *COCV*. Ed. by S. Glesner, J. Knoop, and R. Drechsler. Vol. 190(4). Electr. Notes Theor. Comput. Sci. Elsevier, 2007, pp. 3–16. Links: ee, Google Scholar. See p. 6.

[BS03] D. Le Berre and L. Simon. "The Essentials of the SAT 2003 Competition". In: *SAT*. Ed. by E. Giunchiglia and A. Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003, pp. 452–467. ISBN: 3-540-20851-8. Links: ee, Google Scholar. See p. 5.

[CGH97] E. Clarke, O. Grumberg, and K. Hamaguchi. "Another Look at LTL Model Checking". In: *Formal Methods in System Design* 10.1 (1997), pp. 47–71. Links: Google Scholar. See p. 4.

[Cim+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking". In: *CAV*. Ed. by E. Brinksma and K. Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002, pp. 359–364. ISBN: 3-540-43997-8. Links: ee, Google Scholar. See pp. 1, 3, 4.

[Cim+07] A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. "Boolean Abstraction for Temporal Logic Satisfiability". In: *CAV*. Ed. by W. Damm and H. Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 532–546. ISBN: 3-540-22342-8. Links: ee, Google Scholar. See p. 1.

[csmv] `http://www.kenmcmil.com/smv.html`. See p. 3.

[DGV99] M. Daniele, F. Giunchiglia, and M. Vardi. "Improved Automata Generation for Linear Temporal Logic". In: *CAV*. Ed. by N. Halbwachs and D. Peled. Vol. 1633. Lecture Notes in Computer Science. Springer, 1999, pp. 249–260. ISBN: 3-540-66202-2. Links: ee, Google Scholar. See p. 6.

[EL86] E. Emerson and C. Lei. "Efficient Model Checking in Fragments of the Propositional Mu-Calculus (Extended Abstract)". In: *LICS*. IEEE Computer Society, 1986, pp. 267–278. Links: Google Scholar. See pp. 4, 10, 11.

[Eme90] E. Emerson. "Temporal and Modal Logic". In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. 1990, pp. 995–1072. Links: Google Scholar. See pp. 1, 3.

[FDP01] M. Fisher, C. Dixon, and M. Peim. "Clausal temporal resolution". In: *ACM Trans. Comput. Log.* 2.1 (2001), pp. 12–56. Links: ee, Google Scholar. See pp. 1, 4.

[Fis+08] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Vardi. "A Framework for Inherent Vacuity". In: *HVC*. Ed. by H. Chockler and A. Hu. Vol. 5394. Lecture Notes in Computer Science. Springer, 2008, pp. 7–22. ISBN: 978-3-642-01701-8. Links: ee, Google Scholar. See p. 1.

[FJR09] E. Filiot, N. Jin, and J. Raskin. "An Antichain Algorithm for LTL Realizability". In: *CAV*. Ed. by A. Bouajjani and O. Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 263–277. ISBN: 978-3-642-02657-7. Links: ee, Google Scholar. See p. 6.

[GH06] J. Geldenhuys and H. Hansen. "Larger Automata and Less Work for LTL Model Checking". In: *SPIN*. Ed. by A. Valmari. Vol. 3925. Lecture Notes in Computer Science. Springer, 2006, pp. 53–70. ISBN: 3-540-33102-6. Links: ee, Google Scholar. See p. 6.

[GKS09] V. Goranko, A. Kyrilov, and D. Shkatov. "Tableau Tool for Testing Satisfiability in LTL: Implementation and Experimental Analysis". In: *M4M*. 2009, pp. 113–125. Links: ee, Google Scholar. See pp. 2, 3.

[Gor10] R. Goré. *Personal Communication*. 2010. See p. 3.

[GS01] C. Gomes and B. Selman. "Algorithm portfolios". In: *Artif. Intell.* 126.1-2 (2001), pp. 43–62. Links: ee, Google Scholar. See pp. 2, 12.

[GW09] R. Goré and F. Widmann. "An Optimal On-the-Fly Tableau-Based Decision Procedure for PDL-Satisfiability". In: *CADE*. Ed. by R. A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, 2009, pp. 437–452. ISBN: 978-3-642-02958-5. Links: ee, Google Scholar. See p. 4.

[GW10] R. Goré and F. Widmann. "An Experimental Comparison of Theorem Provers for CTL". In: *CLoDeM*. 2010. Links: Google Scholar. See p. 2.

[Har05] A. Harding. "Symbolic Strategy Synthesis For Games With LTL Winning Conditions". PhD thesis. University of Birmingham, 2005. Links: Google Scholar. See p. 6.

[Heu+95] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. "Propositional Logics on the Computer". In: *TABLEAUX*. Ed. by P. Baumgartner, R. Hähnle, and J. Posegga. Vol. 918. Lecture Notes in Computer Science. Springer, 1995, pp. 310–323. ISBN: 3-540-59338-1. Links: Google Scholar. See pp. 1, 3, 4.

[HH01] B. Hirsch and U. Hustadt. "Translating PLTL into WS1S: Application Description". In: *M4M*. 2001. Links: ee, Google Scholar. See pp. 1, 2.

[HJL05] K. Heljanko, T. Junttila, and T. Latvala. "Incremental and Complete Bounded Model Checking for Full PLTL". In: *CAV*. Ed. by K. Etessami and S. Rajamani. Vol. 3576. Lecture Notes in Computer Science. Springer, 2005, pp. 98–111. ISBN: 3-540-27231-3. Links: ee, Google Scholar. See p. 4.

[HK04] U. Hustadt and B. Konev. "TRP++: A temporal resolution prover". In: *Collegium Logicum*. Ed. by M. Baaz, J. Makowsky, and A. Voronkov. Vol. 8. Kurt Gödel Society, 2004, pp. 65–79. Links: Google Scholar. See pp. 2–4, 10.

[HKQ03] T. Henzinger, O. Kupferman, and S. Qadeer. "From Pre-Historic to Post-Modern Symbolic Model Checking". In: *Formal Methods in System Design* 23.3 (2003), pp. 303–327. Links: ee, Google Scholar. See pp. 4, 11.

[HLH97] B. Huberman, R. Lukose, and T. Hogg. “An Economics Approach to Hard Computational Problems”. In: *Science* 275.5296 (1997), pp. 51–54. Links: ee, Google Scholar. See pp. 2, 12, 13.

[HS01] U. Hustadt and R. A. Schmidt. “Formulae which Highlight Differences between Temporal Logic and Dynamic Logic Provers”. In: *Issues in the Design and Experimental Evaluation of Systems for Modal and Temporal Logics*. Ed. by E. Giunchiglia and F. Massacci. Dipartimento di Ingegneria dell’Informazione, Unversitá degli Studi di Siena, 2001, pp. 68–76. Links: Google Scholar. See pp. 1, 2.

[HS02] U. Hustadt and R. A. Schmidt. “Scientific Benchmarking with Temporal Logic Decision Procedures”. In: *KR*. Ed. by D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams. Morgan Kaufmann, 2002, pp. 533–546. ISBN: 1-55860-554-1. Links: Google Scholar. See pp. 1, 2, 5, 6.

[Hus+04] U. Hustadt, B. Konev, A. Riazanov, and A. Voronkov. “TeMP: A Temporal Monodic Prover”. In: *IJCAR*. Ed. by D. Basin and M. Rusinowitch. Vol. 3097. Lecture Notes in Computer Science. Springer, 2004, pp. 326–330. ISBN: 3-540-22345-2. Links: ee, Google Scholar. See p. 3.

[Jan99] G. Janssen. “Logics for Digital Circuit Verification: Theory, Algorithms, and Applications”. PhD thesis. Technische Universiteit Eindhoven, 1999. Links: Google Scholar. See pp. 1, 4.

[JB06] B. Jobstmann and R. Bloem. “Optimizations for LTL Synthesis”. In: *FMCAD*. IEEE Computer Society, 2006, pp. 117–124. ISBN: 0-7695-2707-8. Links: ee, Google Scholar. See p. 6.

[LB+03] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham. “A Portfolio Approach to Algorithm Selection”. In: *IJCAI*. Ed. by G. Gottlob and T. Walsh. Morgan Kaufmann, 2003, pp. 1542–1542. Links: Google Scholar. See pp. 2, 12.

[LH09a] M. Ludwig and U. Hustadt. “Fair Derivations in Monodic Temporal Reasoning”. In: *CADE*. Ed. by R. A. Schmidt. Vol. 5663. Lecture Notes in Computer Science. Springer, 2009, pp. 261–276. ISBN: 978-3-642-02958-5. Links: ee, Google Scholar. See p. 3.

[LH09b] M. Ludwig and U. Hustadt. “Resolution-Based Model Construction for PLTL”. In: *TIME*. Ed. by C. Lutz and J. Raskin. IEEE Computer Society, 2009, pp. 73–80. ISBN: 978-0-7695-3727-6. Links: ee, Google Scholar. See p. 4.

[LH10] M. Ludwig and U. Hustadt. “Implementing a fair monodic temporal logic prover”. In: *AI Commun.* 23.2-3 (2010), pp. 69–96. Links: ee, Google Scholar. See pp. 2–4.

[lwb] `http://www.lwb.unibe.ch/index.html`. See pp. 1, 3, 4.

[minisat] `http://minisat.se/`. See p. 4.

[Mou04] L. de Moura. *SAL: Tutorial*. 2004. Available from `http://sal.csl.sri.com/doc/salenv_tutorial.pdf`. See p. 3.

[Mou+04] L. de Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. “SAL 2”. In: *CAV*. Ed. by R. Alur and D. Peled. Vol. 3114. Lecture Notes in Computer Science. Springer, 2004, pp. 496–500. ISBN: 3-540-22342-8. Links: ee, Google Scholar. See p. 3.

[Nik10] M. Nikolic. "Statistical Methodology for Comparison of SAT Solvers". In: *SAT*. Ed. by O. Strichman and S. Szeider. Vol. 6175. Lecture Notes in Computer Science. Springer, 2010, pp. 209–222. ISBN: 978-3-642-14185-0. Links: ee, Google Scholar. See p. 6.

[nusmv] http://nusmv.fbk.eu/. See pp. 1, 3, 4.

[pbc10] http://www.cril.univ-artois.fr/PB10/. See p. 7.

[Pil+06] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. "Formal analysis of hardware requirements". In: *DAC*. Ed. by E. Sentovich. ACM, 2006, pp. 821–826. ISBN: 1-59593-381-6. Links: ee, Google Scholar. See p. 1.

[pltl] http://users.cecs.anu.edu.au/~rpg/PLTLProvers/. See pp. 1, 3, 4.

[Pnu77] A. Pnueli. "The Temporal Logic of Programs". In: *FOCS*. IEEE, 1977, pp. 46–57. Links: Google Scholar. See p. 1.

[prosyd] *Prosyd*. http://www.prosyd.org/. See p. 1.

[PT09] L. Pulina and A. Tacchella. "A self-adaptive multi-engine solver for quantified Boolean formulas". In: *Constraints* 14.1 (2009), pp. 80–116. Links: ee, Google Scholar. See p. 2.

[rozier] http://shemesh.larc.nasa.gov/people/kyr/benchmarking_scripts/benchmarking_scripts.html. See p. 6.

[run] A. Biere and T. Jussila. *Benchmark Tool Run*. http://fmv.jku.at/run/. See p. 5.

[RV10] K. Rozier and M. Vardi. "LTL Satisfiability Checking". In: *STTT* 12.2 (2010), pp. 123–137. Links: ee, Google Scholar. See pp. 1–3, 5, 6.

[RV11] K. Rozier and M. Vardi. "A Multi-encoding Approach for LTL Symbolic Satisfiability Checking". In: *FM*. Ed. by M. Butler and W. Schulte. Vol. 6664. Lecture Notes in Computer Science. Springer, 2011, pp. 417–431. ISBN: 978-3-642-21436-3. Links: ee, Google Scholar. See pp. 2, 5.

[SB05] L. Simon and D. Le Berre. "Some Results and Lessons from the SAT Competitions (invited talk, slides only)". In: *Second International Workshop on Constraint Propagation and Implementation, Sitges, Spain, October 1, 2005*. 2005. Available from http://www.lri.fr/~simon/recherche/papiers/InvitedTalk-CPAI2005-Simon.zip. See pp. 3, 5.

[Sch10] V. Schuppan. "Towards a notion of unsatisfiable and unrealizable cores for LTL". In: *Sci. Comput. Program.* In Press (2010). DOI: 10.1016/j.scico.2010.11.004. Links: ee, Google Scholar. See pp. 5, 6.

[Sch98] S. Schwendimann. "A New One-Pass Tableau Calculus for PLTL". In: *TABLEAUX*. Ed. by H. de Swart. Vol. 1397. Lecture Notes in Computer Science. Springer, 1998, pp. 277–292. ISBN: 3-540-64406-7. Links: ee, Google Scholar. See pp. 1, 4.

[SM07] H. Samulowitz and R. Memisevic. "Learning to Solve QBF". In: *AAAI*. AAAI Press, 2007, pp. 255–260. ISBN: 978-1-57735-323-2. Links: Google Scholar. See p. 2.

[SS01] G. Sutcliffe and C. Suttner. "Evaluating general purpose automated theorem proving systems". In: *Artif. Intell.* 131.1-2 (2001), pp. 39–54. Links: ee, Google Scholar. See p. 3.

[Sta] StatSoft, Inc. *Electronic Statistics Textbook*. StatSoft, Tulsa, OK, USA. Available from http://www.statsoft.com/textbook/. See p. 7.

[STV05] R. Sebastiani, S. Tonetta, and M. Vardi. "Symbolic Systems, Explicit Properties: On Hybrid Approaches for LTL Symbolic Model Checking". In: *CAV*. Ed. by K. Etessami and S. Rajamani. Vol. 3576. Lecture Notes in Computer Science. Springer, 2005, pp. 350–363. ISBN: 3-540-27231-3. Links: ee, Google Scholar. See p. 6.

[The96] The VIS Group. "VIS: A System for Verification and Synthesis". In: *CAV*. Ed. by R. Alur and T. Henzinger. Vol. 1102. Lecture Notes in Computer Science. Springer, 1996, pp. 428–432. ISBN: 3-540-61474-5. Links: ee, Google Scholar. See p. 3.

[trp++] http://www.csc.liv.ac.uk/~konev/software/trp++/. See pp. 2–4.

[trp] http://www.csc.liv.ac.uk/~ullrich/TRP/. See p. 6.

[tspass] http://www.csc.liv.ac.uk/~michel/software/tspass/. See pp. 2–4.

[Wol85] P. Wolper. "The Tableau Method for Temporal Logic: An Overview". In: *Logique et Analyse* 28.110–111 (1985), pp. 119–136. Links: Google Scholar. See p. 1.

[Wul+08] M. De Wulf, L. Doyen, N. Maquet, and J. Raskin. "Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking". In: *TACAS*. Ed. by C Ramakrishnan and J. Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 63–77. ISBN: 978-3-540-78799-0. Links: ee, Google Scholar. See pp. 1–6, 11.

[www] http://www.schuppan.de/viktor/atva11/. See pp. 2, 5, 7, 13.

[Xu+08] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. "SATzilla: Portfolio-based Algorithm Selection for SAT". In: *J. Artif. Intell. Res. (JAIR)* 32 (2008), pp. 565–606. Links: ee, Google Scholar. See pp. 2, 12, 13.

A Concepts and Terminology

Some terminology below is inspired by that in SAT competitions [SB05,Ber+09]. For a general reference on experimental comparison of algorithms see, e.g., [Joh02].

A.1 Benchmarks

A single formula considered for solving is an *instance*. An instance is either satisfiable (*sat*) or unsatisfiable (*unsat*), which, due to non-termination of solvers, may not be known (*unknown*). Each *sat* instance has a set of *models* satisfying the instance. Besides *sat* and *unsat* running a solver on an instance may lead to the solver using more than the maximum allocated time (*time-out*) or memory (*mem-out*), or to an error due to a wrong result or a crash (*error*). A wrong result can be either a *sat*/*unsat* answer when the opposite is true or a model that does not satisfy the given instance. A *sat* (resp. *unsat*) instance is *solved* by some solver if the solver answers *sat* (resp. *unsat*) on that instance. If model construction is desired, then a *sat* answer must be accompanied by a model satisfying the instance.

The *size* of a formula is given by the sum of the number of occurrences of temporal and propositional operators and atomic propositions. The size of a model is the number of states if the model is a finite prefix and the sum of the length of the stem and the loop (without counting the first repeating state twice) if the model is lasso-shaped.

An instance may be used in *non-negated* or (less often) *negated versions*. Several related instances are grouped into *families*. Families may also be grouped into families, leading to the notion of *subfamilies*. A set of instances that differ (only) by varying some difficulty-related parameter is sometimes called a *series*. It is customary to group families according to their nature into the *categories* **application**, **crafted**, and **random**. Sometimes the *specialties* of only *sat*, only *unsat*, and both *sat* and *unsat* instances are distinguished. We use **bold font** for sets of instances.

Sometimes (e.g., [BS04,SS01]) instances are altered syntactically by, e.g., reordering clauses in a formula, renaming variables, etc. This is done to prevent solvers from recognizing instances (e.g., [BS03]) and/or because the performance of a solver may change (e.g., [Nik10,BS03,BS04].) While a solver whose performance is independent of the specific syntax of an instance is certainly nice [SS01], in an application-oriented setting names and structure can give clues to improve performance [BS04].

A.2 Solvers

A *tool* is a program that takes an instance as input and answers *sat*, *unsat*, or *unknown*. For *sat* instances it may construct a model. Often the behavior of a tool can be tuned via program options. We call a tool with specific option values

a *configuration*. We denote configurations in `teletype`. We sometimes use *solver* as a generic term that may either mean tool or configuration.

A configuration is *complete* if it solves every instance when provided with sufficient time and memory; it is *incomplete* otherwise.

A configuration is *buggy* if it results in *error* on at least one instance. A tool is buggy if one of its configurations is buggy.

A tool is a *state-of-the-art contributor* (sota) if an instance is solved only by configurations of that tool (see also [SS01]). Given a set of configurations C the *virtual best solver* (vbs) is the hypothetical solver using the best configuration in C on any given instance (e.g., [Ber+09]).

A.3 Competitions and Evaluations

The term *competition* is typically used for events where participants can submit both solvers and benchmarks, where participating solvers are run on participating benchmarks, and where one or more winners are announced. *Evaluations* typically do not formally declare winners. Evaluations are often initial or early stage events before (possibly) "upgrading" to a competition. Some examples are the CADE ATP System Competition (CASC) [casc,Sut10], the Hardware Model Checking Competition (HWMCC) [hwmcc10,BC10], the QBF competitive evaluation (QBFEVAL) [qbveval,Pes+10], the Pseudo-Boolean Competition [pbc10,MR10,MR06], the SAT Competition [satcomp,Ber+09,BS06], the SATRace [satrace10,Sin+10], and the Satisfiability Modulo Theories Competition (SMT-COMP) [smtcomp,Bar+08].

Some competitions and evaluations have different *tracks* with competition/evaluation for each track, e.g., parallel [Ber+09], proof generating [BRS07], or special input format [BRS07] solvers.

Competitions and evaluations often proceed in a number of *stages* (e.g., [BS04]). In early stages all solvers participate. Then buggy or badly performing solvers are ruled out and remaining solvers are run again, often with increased resources.

Solvers that do not meet some participation criteria or are discovered to be buggy, are often declared *hors-concours*, i.e., they are run along with other participating solvers (though not always in all stages), but only for informational purposes. In particular, they are not eligible for awards. See, e.g., [SB05].

A.4 Scoring and Ranking

Ranking is the process of ordering the solvers in a comparison based on their (and possibly other solvers') performance. *Scoring* is the process of assigning numerical value(s) to a solver in a comparison based on their (and possibly other solvers') performance. In competitions this is an important and sometimes contentious issue (e.g., [Ber+09]).

A frequent scoring and ranking scheme (e.g., [Sut10,Ber+09]) ranks by higher number of solved instances and breaks ties using lower average (or, equivalently,

sum) of time taken on solved instances. Several more elaborate schemes have been proposed; see, e.g., [Gel11,Nik10,NPT07,Pul06,Ber+09,SS01].

A.5 Cactus Plots

Cactus plots are frequently used in comparisons to display performance of solvers (e.g., [SS01,SB05]). For a given set of solvers and benchmark instances a cactus plot provides a good impression of the share of solved instances and the distribution of resources required for a solver on that set of instances. However, it loses the correlation between individual instances in the set of instances, i.e., it does not allow to compare the performance of solvers on specific instances or subsets of instances.

A cactus plot shows one curve for each solver. The x- and y-coordinates of the points of a curve are obtained as follows. Instances are on the x-axis and are sorted in increasing order of run time.[4] Note that the order of instances on the x-axis may differ between solvers. Therefore, the x-axis is typically labeled by number of instances. Instances not solved by a particular solver are not plotted for that solver; in that case the curve for that solver does not reach the right border of the plot.

The y-axis shows the run time required. The y-axis either shows the run times required for individual instances (*non-accumulated*, e.g., [Ber+09]) or, less frequently, the sum of the run times up to that instance (*accumulated*, e.g., [Bar+08]). Cactus plots with accumulated run times make it easy to identify winners when scoring and ranking by higher number of solved instances and ties broken by lower sum of time taken on solved instances. The further to the right a certain solver's curve reaches, the more instances it solves (right-most is best). The lower the right end point of a solver's curve lies, the fewer time it takes (lowest is best). However, accumulated run times make it harder to see the time required on individual benchmarks than non-accumulated run times. Obtaining the sums of run times in a non-accumulated cactus plot requires measuring the areas below the curves (integration); correspondingly, determining run times on individual benchmark instances in an accumulated cactus plot can be done by taking the gradients of the curves (differentiation).

A.6 Peter Principle Point

Frequently solvers show the following characteristic [SS01]. While a large part of the benchmarks is solved even within a fairly small time limit, increasing the time limit will lead to more solved benchmarks only up to a certain point. Beyond that even a substantially larger time limit will not help (much). Sometimes this point is called *Peter Principle Point* [SS01] (after [PH69]). It is significant for comparisons as it is one indication of whether the chosen time limit is appropriate for the participating solvers and benchmarks. Cactus plots help to identify that point as, when a solver reaches that point on some benchmark set, then its curve in the corresponding cactus plot will start to rise steeply.

[4] Or other measure of interest such as memory. W.l.o.g. we use run time as an example.

A.7 Contour/Discrete Raw Data Plots: Historical References

For early appearances of similar plots (under different names) see, e.g., [Bri39] and [Lou73] (also via [FD,WF09,Fal08]). Spectra in physics and staining of chromosomes to obtain karyograms in biology come to mind as being somewhat related in other fields.

B Benchmarks — More Details

B.1 Detailed Overview of Benchmark Families

In Tab. 4 and 5 we provide a more detailed description of the benchmark families than in Tab. 1. Benchmark families are described at a lower level of subfamilies. In addition to what is shown in Tab. 1 scaling parameter and values are given in col. 2 and 3, size of the smallest instance in col. 4, and overall number of instances in col. 5.

B.2 Tree Structure of Benchmark Families

In Fig. 5 and 6 we show the tree structure of the benchmark families used for computing the weighted share of solved instances and the weighted average run time on solved instances (see Sect. 5). The leafs in the tree contain the instances.

In category **random** each leaf represents several leaf subfamilies that differ in a size-related parameter, shown in set notation. For example, family **trp_N5y** has 14 subfamilies $1, 5, 10, \ldots, 40$. Each of the leaf subfamilies in category **random** contains 10 instances.

Each instance of a leaf subfamily is assigned equal weight. Similarly, each subfamily of a family is assigned equal weight. Note that not in all computations all instances participate. Hence, the overall weight assigned to an instance may differ. As an example, for the computation of the average run time on *solved* instances of some configuration only the instances actually solved by that configuration participate in the computation. All unsolved instances and, consequentially, all subfamilies without solved instances are removed from the tree for the computation of the weights.

C Discussion

C.1 Isn't this just an exercise in data collection? Shouldn't you put your data to some actual use before publishing a paper?

A broad solver comparison such as this yields valuable results in itself, the most important of which we list below. For some references on the motivation of solver competitions and evaluations see, e.g., [SS01,LRD10,SBH05].

family	scales in number of	scaling values	min. size	max. size	num.	num. *sat*	num. *unsat*	num. *unkn.*	source	description
application										
acacia_demo-v22	–	–	27	76	10	10	–	–	[acacia, FJR09, JB06]	window screens
acacia_demo-v3_demo-v3	requests / grants	1–12	37	378	12	12	–	–	[acacia, FJR09, JB06]	arbiters (scaled up)
acacia_demo-v3_demo-v3_c			37	378	12	12	–	–		variant of **demo-v3** of the form $(\wedge_i a_i) \wedge (\wedge_i g_i)$
acacia_demo-v3_demo-v3_cl			41	426	12	12	–	–		variant of **demo-v3_c** requiring that each request is true infinitely often
acacia_example	–	–	7	144	25	25	–	–	[acacia, FJR09, JB06]	mostly arbiters and traffic light controllers
alaska_lift_lift	floors	2–18	174	1942	17	17	–	–	[alaska, Wul+08, Har05]	lifts
alaska_lift_lift_f			176	1976	17	17	–	–		variant of **lift** that fixes a bug [Sch10] preventing the lift from leaving floor 0
alaska_lift_lift_l			182	2014	17	–	17	–		variant of **lift** requiring to serve each floor infinitely often
alaska_lift_lift_f_l			184	2048	17	17	–	–		combines **lift_f** and **lift_l**
alaska_lift_lift_b			177	4344	17	17	–	–		lifts with binary rather than one-hot encoding of floors
alaska_lift_lift_b_f			179	4378	17	17	–	–		combines **lift_b** and **lift_f**
alaska_lift_lift_b_l			185	4416	17	–	17	–		combines **lift_b** and **lift_l**
alaska_lift_lift_b_f_l			187	4450	17	17	–	–		combines **lift_b**, **lift_f**, and **lift_l**
alaska_szymanski	–	–	149	183	4	4	–	–	[alaska, Wul+08, STV05]	mutual exclusion protocols
anzu_amba_amba	masters	2–18	1045	6065	17	17	–	–	[anzu, Blo+07a]	advanced microcontroller bus architectures (scaled up)
anzu_amba_amba_c			1045	6065	17	17	–	–		variant of **amba** of the form $(\wedge_i a_i) \wedge (\wedge_i g_i)$
anzu_amba_amba_cl			1057	6173	17	9	–	8		variant of **amba_c** letting each master own the bus infinitely often
anzu_genbuf_genbuf	senders	1–20	685	5725	20	20	–	–	[anzu, Blo+07b]	generalized buffers (scaled up)
anzu_genbuf_genbuf_c			685	5725	20	20	–	–		variant of **genbuf** of the form $(\wedge_i a_i) \wedge (\wedge_i g_i)$
anzu_genbuf_genbuf_cl			689	5805	20	8	–	12		variant of **genbuf_c** requiring each sender to request to send infinit. often
forobots	–	–	631	636	39	14	25	–	[BDF09]	foraging robots
crafted										
roz_cnt_counter	bits	2–20	103	751	19	19	–	–	[rozier, RV10]	serial counters (models of length exponential in number of bits)
roz_cnt_counterCarry			103	715	19	19	–	–		variant of **counter** using carry bits
roz_cnt_counterCarryLinear			93	345	19	19	–	–		combines **counterCarry** and **counterLinear**
roz_cnt_counterLinear			99	387	19	19	–	–		variant of **counter** with linear rather than quadratic formula size
rozier_pattern_C1formula	n	1 or 2–13 step 1, 20–100 step 10, 100–1000 step 100	4	4000	31	31	–	–	[rozier, RV10, GH06]	$\vee_{i=1}^{n} \mathbf{GF} p_i$
rozier_pattern_C2formula			4	4000	31	31	–	–		$\wedge_{i=1}^{n} \mathbf{GF} p_i$
rozier_pattern_Eformula			3	3000	31	31	–	–		$\wedge_{i=1}^{n} \mathbf{F} p_i$
rozier_pattern_Qformula			6	5994	30	30	–	–		$\wedge_{i=1}^{n-1} ((\mathbf{F} p_i) \vee (\mathbf{G} p_{i+1}))$
rozier_pattern_Rformula			8	7992	30	30	–	–		$\wedge_{i=1}^{n-1} ((\mathbf{GF} p_i) \vee (\mathbf{FG} p_{i+1}))$
rozier_pattern_Sformula			3	3000	31	31	–	–		$\wedge_{i=1}^{n} \mathbf{G} p_i$
rozier_pattern_Uformula			4	2000	30	30	–	–		$(\ldots (p_1 \mathbf{U} p_2) \mathbf{U} \ldots) \mathbf{U} p_n$
rozier_pattern_U2formula			4	2000	30	30	–	–		$p_1 \mathbf{U} (\ldots \mathbf{U} (p_{n-1} \mathbf{U} p_n) \ldots)$
schuppan_O1formula	n	2–10 step 1, 10–100 step 10, 100–1000 step 100	15	4007	27	27	–	–	(new)	$(\wedge_{i=1}^{n} (a_i \vee b_i)) \wedge ((\mathbf{G} c) \wedge (\mathbf{X} \neg c))$
schuppan_O2formula			13	6001	27	–	15	12		$(\wedge_{i=1}^{n-1} (\mathbf{FG}(a_i \leftrightarrow a_{i+1}))) \wedge \mathbf{FG}(a_n \leftrightarrow \neg a_1)$
schuppan_phltl	pigeons / time slots	2–10, 20, ..., 100	27	40501	18	–	10	8	(new)	Temporal formulation of the pigeonhole principle [Bie+09]. There is just one hole that must be occupied by each pigeon infinitely often; if a pigeon occupies the hole at time t, then it must also do so at time $t+n-1$ where n is the number of pigeons.

Table 4. Detailed overview of benchmark families in categories **application** and **crafted**. **roz_cnt** abbreviates **rozier_counter**.

family	scales in number of	scaling values	min. size	max. size	num.	num. *sat*	num. *unsat*	num. *unkn.*	source	description
random										
rozier_formulas_n1_P0.3			11	134		87	13	–		
rozier_formulas_n1_P0.5			11	155		94	6	–		
rozier_formulas_n1_P0.7			14	164		94	6	–		
rozier_formulas_n1_P0.95			14	176		96	4	–		
rozier_formulas_n2_P0.3			11	146		94	6	–		
rozier_formulas_n2_P0.5			11	158		96	4	–		
rozier_formulas_n2_P0.7			11	164		99	1	–		
rozier_formulas_n2_P0.95			14	182		100	0	–		Random formulas in the style of [DGV99] (random syntax trees). Parameters other than size of syntax tree: $n = 1 \ldots 5$ different atomic propositions and $p \in \{\frac{1}{3}, 0.5, 0.7, 0.95\}$ probability of choosing **U** or **R** rather than $\neg$, $\vee$, $\wedge$, or **X**. For each combination of parameters we use 10 instances. In the original family there are 500 instances per combination of parameters as well as further syntax tree sizes.
rozier_formulas_n3_P0.3			11	140		98	2	–		
rozier_formulas_n3_P0.5	size of syntax tree	10, 20, . . . , 100	11	155	100	97	3	–	[rozier, RV10]	
rozier_formulas_n3_P0.7			11	167		100	0	–		
rozier_formulas_n3_P0.95			11	182		100	0	–		
rozier_formulas_n4_P0.3			11	149		96	4	–		
rozier_formulas_n4_P0.5			11	146		98	2	–		
rozier_formulas_n4_P0.7			11	158		97	3	–		
rozier_formulas_n4_P0.95			14	176		100	0	–		
rozier_formulas_n5_P0.3			11	134		98	2	–		
rozier_formulas_n5_P0.5			11	152		100	0	–		
rozier_formulas_n5_P0.7			11	164		99	1	–		
rozier_formulas_n5_P0.95			14	185		100	0	–		
trp_N5x		1, 2, 3, 4, 5, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 25, 27, 30, 32, 35, 40	45	505	240	131	109	–		
trp_N5y		1, 5, 10, 15, 18, 20, 23, 25, 27, 28, 29, 30, 35, 40	164	544	140	94	46	–		
trp_N12x	clauses	1, 2, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 16, 19, 22, 24, 25, 26, 30, 33, 37, 38, 39, 40, 42, 44, 45, 47, 49, 53, 57, 58, 59, 60, 66, 72, 78, 84, 90, 96	94	1202	400	225	175	–	[trp, HS02]	Random formulas generated by filling in random literals in a fixed formula template. **x** and **y** mark different templates. 5 and 12 indicate a size related parameter of the templates. The scaling parameter fixes the number of clauses that contain the random literals. For each combination of parameters we use 10 instances. In the original family there are 100 instances per combination of parameters.
trp_N12y		1, 12, 24, 36, 42, 45, 46, 47, 48, 54, 56, 58, 59, 60, 61, 66, 72, 84, 96	507	1422	190	123	67	–		

Table 5. Detailed overview of benchmark families in category **random**.

Fig. 5. Tree structure of benchmark families in categories **application** and **crafted**.

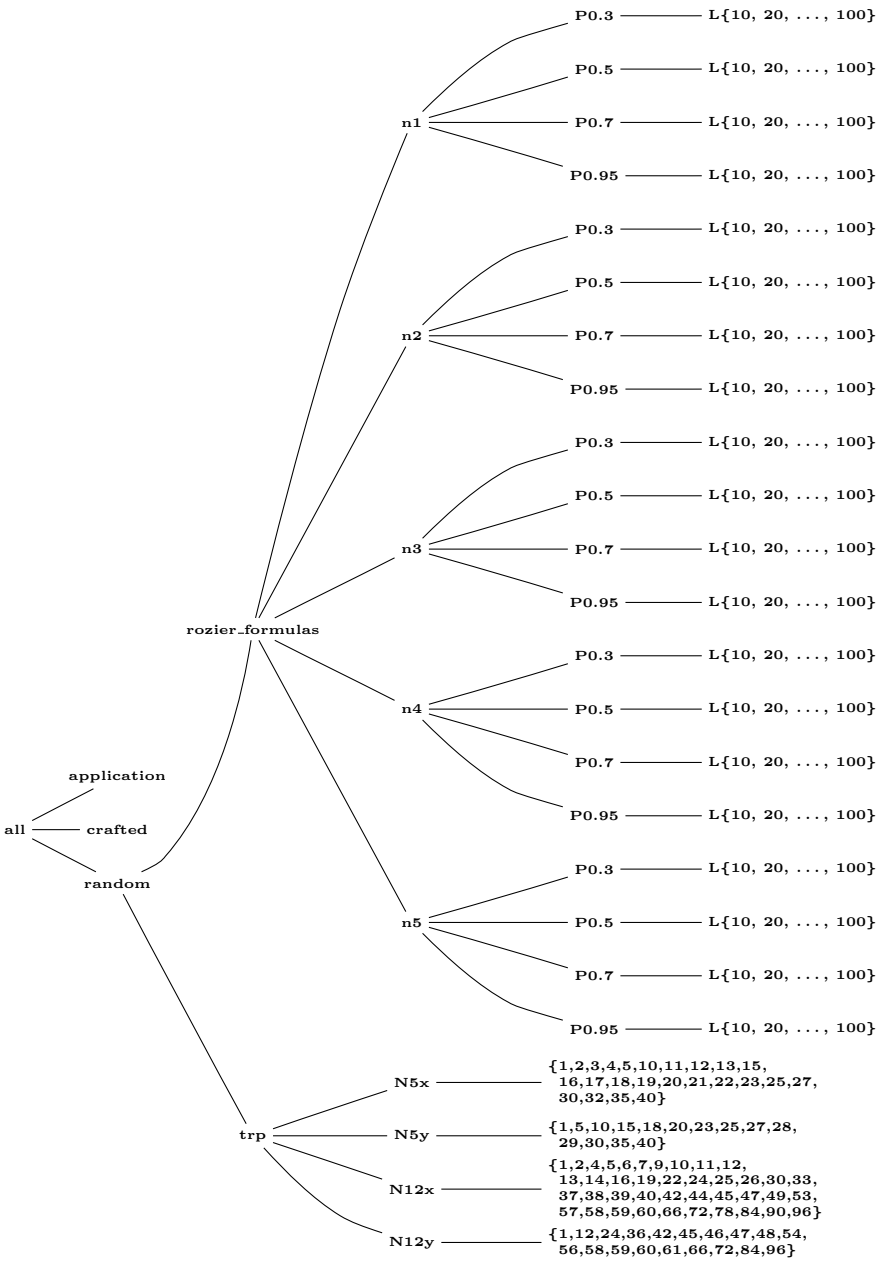

Fig. 6. Tree structure of benchmark families in category **random**.

Comparison of Solvers and Identification of Reference Solvers The most important result of a solver comparison is an indication of the performance of the solvers, both in absolute terms and relative to each other. Specifically, this includes the identification of reference solvers that can, e.g., be used to compare new ideas with. For example, [RV07,RV10] is used in [Mon+08] and [Wul+08] to justify using `NuSMV` [Cim+02] for comparison, in [TV10] and [EF10] to justify using `SPOT` [DLP04] as reference, and in this paper to rule out some model checkers. An important point in identifying reference solvers is to include the corresponding command line options or commands; we have seen more than one instance in which `NuSMV` was used as a reference solver for comparison, and superiority of a new technique was claimed, but it turned out that `NuSMV` was in fact used in a suboptimal fashion (see [Wul+08] as discussed in Sect. 6 as an example). Our data help to identify reference solvers at both aggregate and instance levels, including relevant options or commands.

Improvement of Solvers Often, solver comparisons trigger bugs that can then be fixed by solver authors, leading to more robust solvers (e.g., [SBH05,MR06, BJ07,SB05] and this paper). If the number of configurations that can be entered into a competition or evaluation per participant is limited, then solver authors may be encouraged to develop heuristics to choose a good configuration rather than leaving this task to the user [LRD10]. Similarly, solver competitions and evaluations may help to develop common file formats for benchmarks and get developers to incorporate them in their solvers (e.g., [BMS05,MR06,BJ07]). Note that the latter two points are not the case for this paper.

Collection of Benchmarks Another widely acknowledged purpose of a solver competition or evaluation is to collect a set of benchmarks and make them available in a uniform format as well as identify the easy and difficult ones (e.g., [BMS05,MR06,BJ07]). We make our benchmarks available at [www] in the formats of all solvers we used.

Dissemination A solver comparison may also help to make solver developers aware of trends they may have overlooked. Moreover, it can raise visibility (e.g., [BJ07]) and serve as an entry point to understand the state of the art for people outside the field.

Resource for Deeper Analyses The data obtained in solver comparisons provide a valuable resource for deeper analyses. Some examples are

- *Development of algorithm portfolios:* [Bou+09,NT09] use data from SatEx [SC01] to investigate algorithm portfolios.
- *Development of scoring and ranking methods:* Van Gelder [Gel11] evaluates a proposed ranking method on data from the SAT 2009 competition [Ber+09]. Pulina et al. [NPT07,Pul06] use the results of QBFEVAL'05 [NPT06] to compare and develop scoring and ranking methods. Kullmann [Kul06] discusses

the method used in the SAT 2005 competition [BS06] on the data for the random instances. Zarpas [Zar05] contrasts his own experience of picking a "best" SAT solver with results obtained in the SAT 2003 competition [BS03].
- *Development of data visualization methods:* Purdom et al. [PBS05] apply a parsimony algorithm to the results of the SAT 2002 competition [SBH05] in order to obtain a classification of solvers similar to phylogenetic trees used by biologists. The method has been used to illustrate the outcome of subsequent SAT competitions [BS04,BS05,BRS07].
- *Investigation of specific aspects:* Reports from solver comparisons are constrained in terms of their authors' effort as well as page limits. Hence, more specific analyses (e.g., on subsets of the benchmarks) are often deferred to separate publications. Examples are [Zar06] and [Kul06].

Limitations Clearly, solver comparisons have their limitations, too (see, e.g., [SS01]). Most notably they only provide a snapshot 1. of the solvers involved 2. on the set of benchmarks used 3. subject to the rules of the comparison 4. at a certain point in time. The solvers involved will likely differ in maturity and sophistication of implementation as well as be written by authors with different programming skills and in different programming languages. Finally, the data itself does not provide a direct explanation of why the solvers behave the way they do.

C.2 Why didn't you perform a public competition or evaluation?

Many solver comparisons are performed as public competitions or evaluations where submissions of both solvers and benchmarks are solicited. This clearly has the advantage of giving all interested parties a chance to submit their work, possibly improving selection of solvers and benchmarks. Our reason for not doing so is simply that this work developed over time in a gradual fashion. I.e., we did not start with the idea of performing a comprehensive comparison of solvers for propositional LTL. Rather, originally, we were just intrigued by the results in [Wul+08] that reported superiority of `ALASKA` over `NuSMV` on some benchmarks and performed a small comparison between `ALASKA`, `NuSMV-BDD`, and `NuSMV-SBMC`. On a separate thread we are interested in unsatisfiable cores for propositional LTL [Sch10]. Hence, we added `TRP++` and `TSPASS` to the comparison to gauge their potential for extracting unsatisfiable cores (extracting unsatisfiable cores from resolution proofs is well established in propositional SAT [GN03, ZM03b,ZM03a]). Finally, when we decided to perform a comprehensive comparison, we included `LWB` and `pltl` as representatives of tableau-based approaches. See Sect. 3 why we think that our selection of solvers is still representative of the field.

This paper shows that capable solvers for propositional LTL are available. What is somewhat lacking are benchmarks that are strongly inspired by realistic applications and drive current solvers to or beyond the limit of their capabilities even when using significantly larger time out values. Once such benchmarks are

available we think that the idea of holding a public competition or evaluation should be given serious consideration.

C.3 Relevance of Available CPU Time

In a comparison such as this the available overall CPU time is a major limiting factor. Our main stage took more than 90 CPU days. Hence, when making choices (such as time out values and numbers of benchmark instances), their consequences in terms of required CPU time need to be kept in mind.

C.4 Are 60 seconds time out enough for the main stage?

Factors that influence the time out value include the difficulty and number of benchmarks to be considered, the number of configurations to run, and the available overall CPU time (e.g., [SB05]). Many comparisons use time out values larger than 60 seconds (see, e.g., CASC-J5 [Sut10] with a couple of minutes to SAT'09 [Ber+09] with almost 3 hours for some tracks). Moreover, the cactus plots in Fig. 10–13 in App. D.2 suggest that the Peter Principle Point [SS01] (see also Sect. A.6) may not have been reached for all solvers.

However, we think that the available, relevant benchmarks should have a strong influence on the choice of time out value, too. Note that we collected most available benchmarks for satisfiability of propositional LTL known to us. Before scaling up some of these benchmark families all instances were solved by some configuration in less than 1.5 seconds (though no configuration solved all instances). We then scaled up some benchmarks families and added variants and new ones. We focused most of our effort on the **application** category, which we regard as the most important one. In this category we scaled up by a factor of 2.[5] It currently contains 361 instances; out of those, 20 remained unsolved overall and each individual configuration left at least 66 unsolved. Hence, it seems that 60 seconds is a reasonable time out value for the currently available solvers and benchmarks in category **application**. We hope to see more application inspired benchmarks in the future that will require larger time out values.

C.5 Are 10 seconds time out enough for the preliminary stage?

While a larger time out value might be nice, the share of runs resulting in *time-out* is larger here than in the main stage. We compensated for that by including rather excluding configurations in the main stage in case of doubt, and by confirming our decisions with solver authors. Remember also that only `TRP++` and `TSPASS` required a preliminary stage.

[5] While higher factors are clearly possible, we think that scaling up a benchmark coming from an application too far incurs the risk of leaving the space of realistic application benchmarks. For example, we scaled up **alaska_lift** from 9 to 18 floors and **anzu_amba** from 9 to 18 masters.

C.6 Why is only a subset of the instances in the random category used?

In category **random** we only used 10 instances per parameter combination out of the 500 (**rozier_formulas**) or 100 (**trp**) available in the original benchmark families. In addition, for **rozier_formulas** we only used a subset of families. With our current set of configurations we saw 38444 *time-out* results for **rozier_formulas** and 49056 *time-out* results for **trp**. Simple extrapolation assuming a constant share of *time-out* results suggests a lower bound of more than 9 CPU years of computation time for considering the complete benchmark families. We do not think that in a solver comparison focused on breadth in terms of solvers and benchmarks such as ours the additional knowledge gain would justify that cost. A specific investigation — such as locating phase transitions (e.g., [HS02]) — might necessitate using that many instances; however, in that case the number of solvers (configurations) involved would likely be much smaller.

C.7 Why didn't you perform analysis x?

We did not perform deeper analyses of specific aspects such as phase transition phenomena [HS02] as our comparison is focused on breadth rather than depth and, hence, choosing any particular aspect will be largely arbitrary.

Several alternative scoring and ranking methods have been proposed (e.g., [Gel11,Nik10,NPT07,Pul06,Ber+09,SS01]) and one might think that applying one or more of these could provide valuable additional insight. However, most of these schemes naturally aim at capturing the behavior of a solver in one or a few numbers and tend to imply a corresponding linear order on the solvers. We think that a major result of this investigation is that there are complementary behaviors between groups of solvers. In our opinion the contour/discrete raw data plot suggested by us in Fig. 1 is better suited to capture this. In the future we might want to refine the idea of contour/discrete raw data plots by, e.g., combining them with the solver classification via parsimony algorithms from [SBH05].

Finally, remember that in either case our data are available at [www] for further analyses by third parties.

References

[acacia] http://www.antichains.be/acacia/. See p. 24.

[alaska] http://www.antichains.be/alaska/. See p. 24.

[anzu] http : / / www . iaik . tugraz . at / content / research / design _ verification/anzu/. See p. 24.

[Bar+08] C. Barrett, M. Deters, A. Oliveras, and A. Stump. "Design and Results of the 3rd Annual Satisfiability Modulo Theories Competition (SMT-COMP 2007)". In: *International Journal on Artificial Intelligence Tools (IJAIT)* 17.4 (2008), pp. 569–606. Links: ee, Google Scholar. See pp. 21, 22.

[BC10] A. Biere and K. Claessen. “Hardware Model Checking Competition (presentation, slides only)”. In: *Hardware Verification Workshop 2010, Edinburgh, UK, July 15, 2010*. 2010. Available from http://fmv.jku.at/biere/talks/Biere-HWMCC10-talk.pdf. See p. 21.

[BDF09] A. Behdenna, C. Dixon, and M. Fisher. “Deductive Verification of Simple Foraging Robotic Behaviours”. In: *International Journal of Intelligent Computing and Cybernetics* 2.4 (2009), pp. 604–643. Links: ee, Google Scholar. See p. 24.

[Ber+09] D. Le Berre, O. Roussel, L. Simon, A. Goerdt, I. Lynce, and A. Stump. “The SAT 2009 competition results: does theory meet practice? (presentation, slides only)”. In: *SAT*. Ed. by O. Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009. ISBN: 978-3-642-02776-5. Available from http://www.satcompetition.org/2009/sat09comp-slides.pdf. See pp. 20–22, 28, 30, 31.

[Bie+09] A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009. ISBN: 978-1-58603-929-5. Links: Google Scholar. See p. 24.

[BJ07] A. Biere and T. Jussila. “HWMCC’07: Hardware Model Checking Competition 2007 (presentation, slides only)”. In: *CAV*. Ed. by W. Damm and H. Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007. ISBN: 3-540-22342-8. Available from http://fmv.jku.at/biere/talks/Biere-HWMCC07-talk.pdf. See p. 28.

[Blo+07a] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. “Automatic hardware synthesis from specifications: a case study”. In: *DATE*. Ed. by R. Lauwereins and J. Madsen. ACM, 2007, pp. 1188–1193. ISBN: 978-3-9810801-2-4. Links: ee, Google Scholar. See p. 24.

[Blo+07b] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. “Specify, Compile, Run: Hardware from PSL”. In: *COCV*. Ed. by S. Glesner, J. Knoop, and R. Drechsler. Vol. 190(4). Electr. Notes Theor. Comput. Sci. Elsevier, 2007, pp. 3–16. Links: ee, Google Scholar. See p. 24.

[BMS05] C. Barrett, L. de Moura, and A. Stump. “Design and Results of the First Satisfiability Modulo Theories Competition (SMT-COMP 2005)”. In: *J. Autom. Reasoning* 35.4 (2005), pp. 373–390. Links: ee, Google Scholar. See p. 28.

[Bou+09] M. Bougeret, P. Dutot, A. Goldman, Y. Ngoko, and D. Trystram. “Combining multiple heuristics on discrete resources”. In: *IPDPS*. IEEE, 2009, pp. 1–8. Links: ee, Google Scholar. See p. 28.

[Bri39] W. Brinton. *Graphic Presentation*. New York City: Brinton Associates, 1939. Available from http://www.archive.org/details/graphicpresentat00brinrich. See p. 23.

[BRS07] D. Le Berre, O. Roussel, and L. Simon. “The SAT’07 Contest: the Final Results! (presentation, slides only)”. In: *SAT*. Ed. by J. Marques-Silva and K. Sakallah. Vol. 4501. Lecture Notes in Computer Science. Springer, 2007. ISBN: 978-3-540-72787-3. Available from http://sat07.ecs.soton.ac.uk/slides/sat07-sat-competition.pdf. See pp. 21, 29.

[BS03] D. Le Berre and L. Simon. "The Essentials of the SAT 2003 Competition". In: *SAT*. Ed. by E. Giunchiglia and A. Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003, pp. 452–467. ISBN: 3-540-20851-8. Links: ee, Google Scholar. See pp. 20, 29.

[BS04] D. Le Berre and L. Simon. "Fifty-Five Solvers in Vancouver: The SAT 2004 Competition". In: *SAT (Selected Papers)*. Ed. by H. Hoos and D. Mitchell. Vol. 3542. Lecture Notes in Computer Science. Springer, 2004, pp. 321–344. ISBN: 3-540-27829-X. Links: ee, Google Scholar. See pp. 20, 21, 29.

[BS05] D. Le Berre and L. Simon. "The SAT'05 Competition (presentation, slides only)". In: *SAT*. Ed. by F. Bacchus and T. Walsh. Vol. 3569. Lecture Notes in Computer Science. Springer, 2005. ISBN: 3-540-26276-8. Available from `http://www.satcompetition.org/2005/presentation-last.pdf`. See p. 29.

[BS06] D. Le Berre and L. Simon. "Special Volume on the SAT 2005 competitions and evaluations". In: *JSAT* 2.1-4 (2006). Links: ee, Google Scholar. See pp. 21, 29.

[casc] `http://www.cs.miami.edu/~tptp/CASC/`. See p. 21.

[Cim+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking". In: *CAV*. Ed. by E. Brinksma and K. Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002, pp. 359–364. ISBN: 3-540-43997-8. Links: ee, Google Scholar. See p. 28.

[DGV99] M. Daniele, F. Giunchiglia, and M. Vardi. "Improved Automata Generation for Linear Temporal Logic". In: *CAV*. Ed. by N. Halbwachs and D. Peled. Vol. 1633. Lecture Notes in Computer Science. Springer, 1999, pp. 249–260. ISBN: 3-540-66202-2. Links: ee, Google Scholar. See p. 25.

[DLP04] A. Duret-Lutz and D. Poitrenaud. "SPOT: An Extensible Model Checking Library Using Transition-Based Generalized Büchi Automata". In: *MASCOTS*. Ed. by D. DeGroot, P. Harrison, H. Wijshoff, and Z. Segall. IEEE Computer Society, 2004, pp. 76–83. ISBN: 0-7695-2251-3. Links: ee, Google Scholar. See p. 28.

[EF10] R. Ehlers and B. Finkbeiner. "On the Virtue of Patience: Minimizing Büchi Automata". In: *SPIN*. Ed. by J. van de Pol and M. Weber. Vol. 6349. Lecture Notes in Computer Science. Springer, 2010, pp. 129–145. ISBN: 978-3-642-16163-6. Links: ee, Google Scholar. See p. 28.

[Fal08] A. de Falguerolles. "L'analyse des données; before and around". In: *Journ@l Electronique d'Histoire des Probabilités et de la Statistique* 4.2 (2008). Links: ee, Google Scholar. See p. 23.

[FD] M. Friendly and D. Denis. *Milestones in the history of thematic cartography, statistical graphics, and data visualization*. Available from `http://www.datavis.ca/milestones/`. See p. 23.

[FJR09] E. Filiot, N. Jin, and J. Raskin. "An Antichain Algorithm for LTL Realizability". In: *CAV*. Ed. by A. Bouajjani and O. Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 263–277. ISBN: 978-3-642-02657-7. Links: ee, Google Scholar. See p. 24.

[Gel11] A. Van Gelder. "Careful Ranking of Multiple Solvers with Timeouts and Ties". In: *SAT*. Ed. by K. Sakallah and L. Simon. Vol. 6695. Lecture Notes in Computer Science. Springer, 2011, pp. 317–328. ISBN: 978-3-642-21580-3. Links: ee, Google Scholar. See pp. 22, 28, 31.

[GH06] J. Geldenhuys and H. Hansen. "Larger Automata and Less Work for LTL Model Checking". In: *SPIN*. Ed. by A. Valmari. Vol. 3925. Lecture Notes in Computer Science. Springer, 2006, pp. 53–70. ISBN: 3-540-33102-6. Links: ee, Google Scholar. See p. 24.

[GN03] E. Goldberg and Y. Novikov. "Verification of Proofs of Unsatisfiability for CNF Formulas". In: *DATE*. IEEE Computer Society, 2003, pp. 10886–10891. ISBN: 0-7695-1870-2. Links: ee, Google Scholar. See p. 29.

[Har05] A. Harding. "Symbolic Strategy Synthesis For Games With LTL Winning Conditions". PhD thesis. University of Birmingham, 2005. Links: Google Scholar. See p. 24.

[HS02] U. Hustadt and R. A. Schmidt. "Scientific Benchmarking with Temporal Logic Decision Procedures". In: *KR*. Ed. by D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams. Morgan Kaufmann, 2002, pp. 533–546. ISBN: 1-55860-554-1. Links: Google Scholar. See pp. 25, 31.

[hwmcc10] http://fmv.jku.at/hwmcc10/. See p. 21.

[JB06] B. Jobstmann and R. Bloem. "Optimizations for LTL Synthesis". In: *FMCAD*. IEEE Computer Society, 2006, pp. 117–124. ISBN: 0-7695-2707-8. Links: ee, Google Scholar. See p. 24.

[Joh02] D. Johnson. "A Theoretician's Guide to the Experimental Analysis of Algorithms". In: *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*. Ed. by M. Goldwasser, D. Johnson, and C. McGeoch. Vol. 59. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 2002, pp. 215–250. Links: Google Scholar. See p. 20.

[Kul06] O. Kullmann. "The SAT 2005 Solver Competition on Random Instances". In: *JSAT* 2.1-4 (2006), pp. 61–102. Links: ee, Google Scholar. See pp. 28, 29.

[Lou73] T. Loua. *Atlas statistique de la population de Paris*. J. Dejey & cie, 1873. Available from http://gallica.bnf.fr/ark:/12148/bpt6k81402n. See p. 23.

[LRD10] C. Lecoutre, O. Roussel, and M. van Dongen. "Promoting robust black-box solvers through competitions". In: *Constraints* 15.3 (2010), pp. 317–326. Links: ee, Google Scholar. See pp. 23, 28.

[Mon+08] M. Montali, P. Torroni, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, and P. Mello. "Verification from Declarative Specifications Using Logic Programming". In: *ICLP*. Ed. by M. de la Banda and E. Pontelli. Vol. 5366. Lecture Notes in Computer Science. Springer, 2008, pp. 440–454. ISBN: 978-3-540-89981-5. Links: ee, Google Scholar. See p. 28.

[MR06] V. Manquinho and O. Roussel. "The First Evaluation of Pseudo-Boolean Solvers (PB'05)". In: *JSAT* 2.1-4 (2006), pp. 103–143. Links: ee, Google Scholar. See pp. 21, 28.

[MR10] V. Manquinho and D. Roussel. "Fifth Pseudo-Boolean Competition PB10 (presentation, slides only)". In: *SAT*. Ed. by O. Strichman and S. Szeider.

Vol. 6175. Lecture Notes in Computer Science. Springer, 2010. ISBN: 978-3-642-14185-0. Available from http://www.cril.univ-artois.fr/PB10/slides-PB10.pdf. See p. 21.

[Nik10] M. Nikolic. "Statistical Methodology for Comparison of SAT Solvers". In: *SAT*. Ed. by O. Strichman and S. Szeider. Vol. 6175. Lecture Notes in Computer Science. Springer, 2010, pp. 209–222. ISBN: 978-3-642-14185-0. Links: ee, Google Scholar. See pp. 20, 22, 31.

[NPT06] M. Narizzano, L. Pulina, and A. Tacchella. "Report of the Third QBF Solvers Evaluation". In: *JSAT* 2.1-4 (2006), pp. 145–164. Links: ee, Google Scholar. See p. 28.

[NPT07] M. Narizzano, L. Pulina, and A. Tacchella. "Ranking and Reputation Systems in the QBF Competition". In: *AI*IA*. Ed. by R. Basili and M. Pazienza. Vol. 4733. Lecture Notes in Computer Science. Springer, 2007, pp. 97–108. ISBN: 978-3-540-74781-9. Links: ee, Google Scholar. See pp. 22, 28, 31.

[NT09] Y. Ngoko and D. Trystram. "Combining SAT solvers on discrete resources". In: *HPCS*. IEEE, 2009, pp. 153 –160. Links: ee, Google Scholar. See p. 28.

[pbc10] http://www.cril.univ-artois.fr/PB10/. See p. 21.

[PBS05] P. Purdom, D. Le Berre, and L. Simon. "A parsimony tree for the SAT2002 competition". In: *Ann. Math. Artif. Intell.* 43.1 (2005), pp. 343–365. Links: ee, Google Scholar. See p. 29.

[Pes+10] C. Peschiera, L. Pulina, A. Tacchella, U. Bubeck, O. Kullmann, and I. Lynce. "The Seventh QBF Solvers Evaluation (QBFEVAL'10)". In: *SAT*. Ed. by O. Strichman and S. Szeider. Vol. 6175. Lecture Notes in Computer Science. Springer, 2010, pp. 237–250. ISBN: 978-3-642-14185-0. Links: ee, Google Scholar. See p. 21.

[PH69] L. Peter and R. Hull. *The Peter Principle: Why Things Always Go Wrong*. New York: William Morrow and Company, 1969. Links: Google Scholar. See p. 22.

[Pul06] L. Pulina. "Empirical Evaluation of Scoring Methods". In: *STAIRS*. Ed. by L. Penserini, P. Peppas, and A. Perini. Vol. 142. Frontiers in Artificial Intelligence and Applications. IOS Press, 2006, pp. 108–119. ISBN: 978-1-58603-645-4. Links: Google Scholar. See pp. 22, 28, 31.

[qbveval] http://www.qbflib.org/. See p. 21.

[rozier] http://shemesh.larc.nasa.gov/people/kyr/benchmarking_scripts/benchmarking_scripts.html. See pp. 24, 25.

[RV07] K. Rozier and M. Vardi. "LTL Satisfiability Checking". In: *SPIN*. Ed. by D. Bosnacki and S. Edelkamp. Vol. 4595. Lecture Notes in Computer Science. Springer, 2007, pp. 149–167. ISBN: 978-3-540-73369-0. Links: ee, Google Scholar. See p. 28.

[RV10] K. Rozier and M. Vardi. "LTL Satisfiability Checking". In: *STTT* 12.2 (2010), pp. 123–137. Links: ee, Google Scholar. See pp. 24, 25, 28.

[satcomp] http://www.satcompetition.org/. See p. 21.

[satrace10] http://baldur.iti.uka.de/sat-race-2010/. See p. 21.

[SB05] L. Simon and D. Le Berre. “Some Results and Lessons from the SAT Competitions (invited talk, slides only)”. In: *Second International Workshop on Constraint Propagation and Implementation, Sitges, Spain, October 1, 2005*. 2005. Available from http://www.lri.fr/~simon/recherche/papiers/InvitedTalk-CPAI2005-Simon.zip. See pp. 20–22, 28, 30.

[SBH05] L. Simon, D. Le Berre, and E. Hirsch. “The SAT2002 competition”. In: *Ann. Math. Artif. Intell.* 43.1 (2005), pp. 307–342. Links: ee, Google Scholar. See pp. 23, 28, 29, 31.

[SC01] L. Simon and P. Chatalic. “SatEx: A Web-based Framework for SAT Experimentation”. In: *Electronic Notes in Discrete Mathematics* 9 (2001), pp. 129–149. Links: ee, Google Scholar. See p. 28.

[Sch10] V. Schuppan. “Towards a notion of unsatisfiable and unrealizable cores for LTL”. In: *Sci. Comput. Program.* In Press (2010). DOI: 10.1016/j.scico.2010.11.004. Links: ee, Google Scholar. See pp. 24, 29.

[Sin+10] C. Sinz, A. Gupta, Y. Hamadi, H. Jain, D. Le Berre, P. Manolios, Y. Novikov, and F. Merz. “SAT-Race 2010 (presentation, slides only)”. In: *SAT*. Ed. by O. Strichman and S. Szeider. Vol. 6175. Lecture Notes in Computer Science. Springer, 2010. ISBN: 978-3-642-14185-0. Available from http://baldur.iti.uka.de/sat-race-2010/downloads/SAT-Race-2010-Presentation.pdf. See p. 21.

[smtcomp] http://www.smtcomp.org/. See p. 21.

[SS01] G. Sutcliffe and C. Suttner. “Evaluating general purpose automated theorem proving systems”. In: *Artif. Intell.* 131.1-2 (2001), pp. 39–54. Links: ee, Google Scholar. See pp. 20–23, 29–31.

[STV05] R. Sebastiani, S. Tonetta, and M. Vardi. “Symbolic Systems, Explicit Properties: On Hybrid Approaches for LTL Symbolic Model Checking”. In: *CAV*. Ed. by K. Etessami and S. Rajamani. Vol. 3576. Lecture Notes in Computer Science. Springer, 2005, pp. 350–363. ISBN: 3-540-27231-3. Links: ee, Google Scholar. See p. 24.

[Sut10] G. Sutcliffe. “The CADE-22 automated theorem proving system competition - CASC-22”. In: *AI Commun.* 23.1 (2010), pp. 47–59. Links: ee, Google Scholar. See pp. 21, 30.

[trp] http://www.csc.liv.ac.uk/~ullrich/TRP/. See p. 25.

[TV10] D. Tabakov and M. Vardi. “Optimized Temporal Monitors for SystemC”. In: *RV*. Ed. by H. Barringer, Y. Falcone, B. Finkbeiner, K. Havelund, I. Lee, G. Pace, G. Rosu, O. Sokolsky, and N. Tillmann. Vol. 6418. Lecture Notes in Computer Science. Springer, 2010, pp. 436–451. ISBN: 978-3-642-16611-2. Links: ee, Google Scholar. See p. 28.

[WF09] L. Wilkinson and M. Friendly. “The History of the Cluster Heat Map”. In: *The American Statistician* 63.2 (2009), pp. 179–184. Links: ee, Google Scholar. See p. 23.

[Wul+08] M. De Wulf, L. Doyen, N. Maquet, and J. Raskin. “Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking”. In: *TACAS*. Ed. by C Ramakrishnan and J. Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 63–77. ISBN: 978-3-540-78799-0. Links: ee, Google Scholar. See pp. 24, 28, 29.

[www] http://www.schuppan.de/viktor/atva11/. See pp. 28, 31.

[Zar05] E. Zarpas. “Benchmarking SAT Solvers for Bounded Model Checking”. In: *SAT*. Ed. by F. Bacchus and T. Walsh. Vol. 3569. Lecture Notes in Computer Science. Springer, 2005, pp. 340–354. ISBN: 3-540-26276-8. Links: ee, Google Scholar. See p. 29.

[Zar06] E. Zarpas. “Back to the SAT05 Competition: an a Posteriori Analysis of Solver Performance on Industrial Benchmarks”. In: *JSAT* 2.1-4 (2006), pp. 229–237. Links: ee, Google Scholar. See p. 29.

[ZM03a] L. Zhang and S. Malik. “Extracting Small Unsatisfiable Cores from Unsatisfiable Boolean Formula (presentation, slides only)”. In: *SAT*. Ed. by E. Giunchiglia and A. Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003. ISBN: 3-540-20851-8. Available from `http://research.microsoft.com/users/lintaoz/papers/SAT_2003_core.pdf`. See p. 29.

[ZM03b] L. Zhang and S. Malik. “Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications”. In: *DATE*. IEEE Computer Society, 2003, pp. 10880–10885. ISBN: 0-7695-1870-2. Links: ee, Google Scholar. See p. 29.

D Additional Graphs

D.1 Contour/Discrete Raw Data Plots

Memory

Figure 7 is analogous to Fig. 1 but for memory usage.

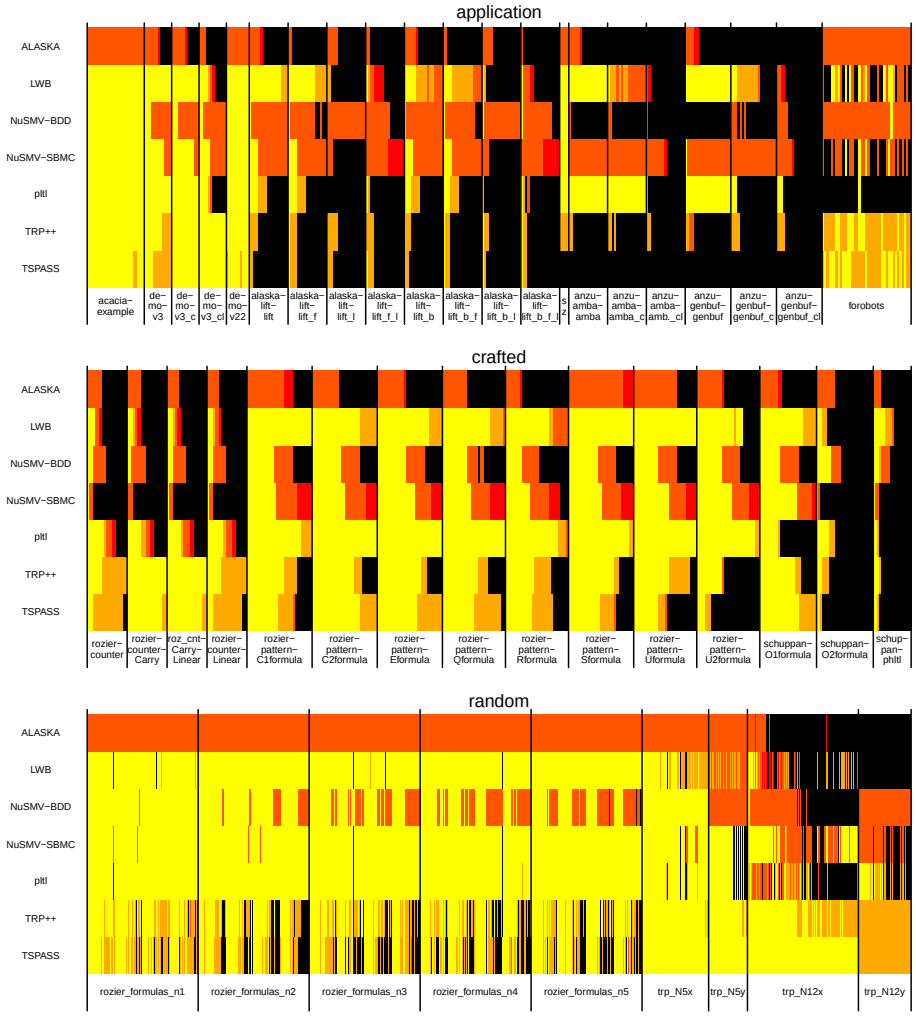

Fig. 7. Contour/discrete raw data plots of memory usage for the winning configurations with model construction dis- or enabled (all instances). Instances are on the x-axes (only identified by their families), configurations on the y-axes. Each rectangle represents the memory usage of one configuration on one instance. **sz** abbreviates **alaska_szymanski**, **roz_cnt** abbreviates **rozier_counter**, and **demo** stands for **acacia_demo**. Memory usage is encoded using the following colors:
$\leq$ 1 MB; $>$ 1 MB, $\leq$ 10 MB; $>$ 10 MB, $\leq$ 100 MB; $>$ 100 MB, $\leq$ 2048 MB; unsolved.

Run Time VBS

Figure 8 is analogous to Fig. 1 but using the vbs of each tool.

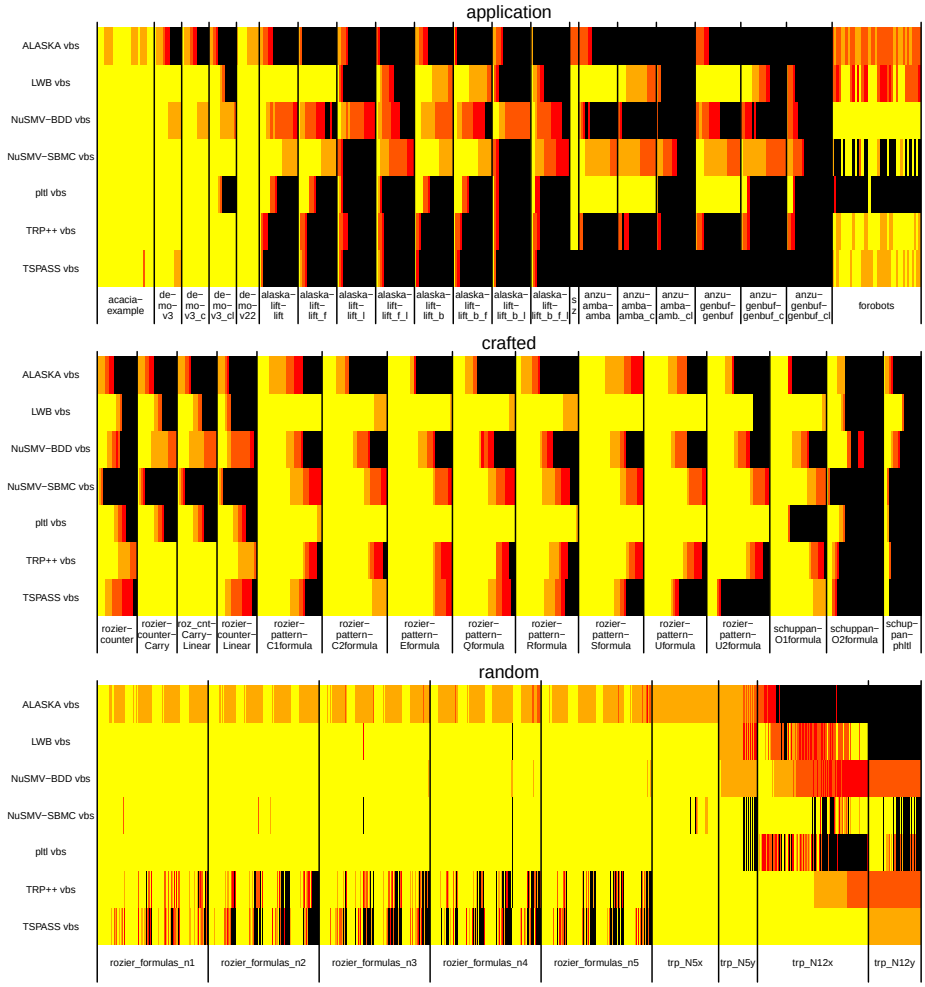

Fig. 8. Contour/discrete raw data plots of run time for the vbs of each tool (all instances). Instances are on the x-axes (only identified by their families), configurations on the y-axes. Each rectangle represents the run time of one configuration on one instance. **sz** abbreviates **alaska_szymanski**, **roz_cnt** abbreviates **rozier_counter**, and **demo** stands for **acacia_demo**. Run times are encoded using the following colors: ≤ 0.1 sec; > 0.1 sec, ≤ 1 sec; > 1 sec, ≤ 10 sec; > 10 sec, ≤ 60 sec; unsolved.

Memory VBS

Figure 9 is analogous to Fig. 7 but using the vbs of each tool.

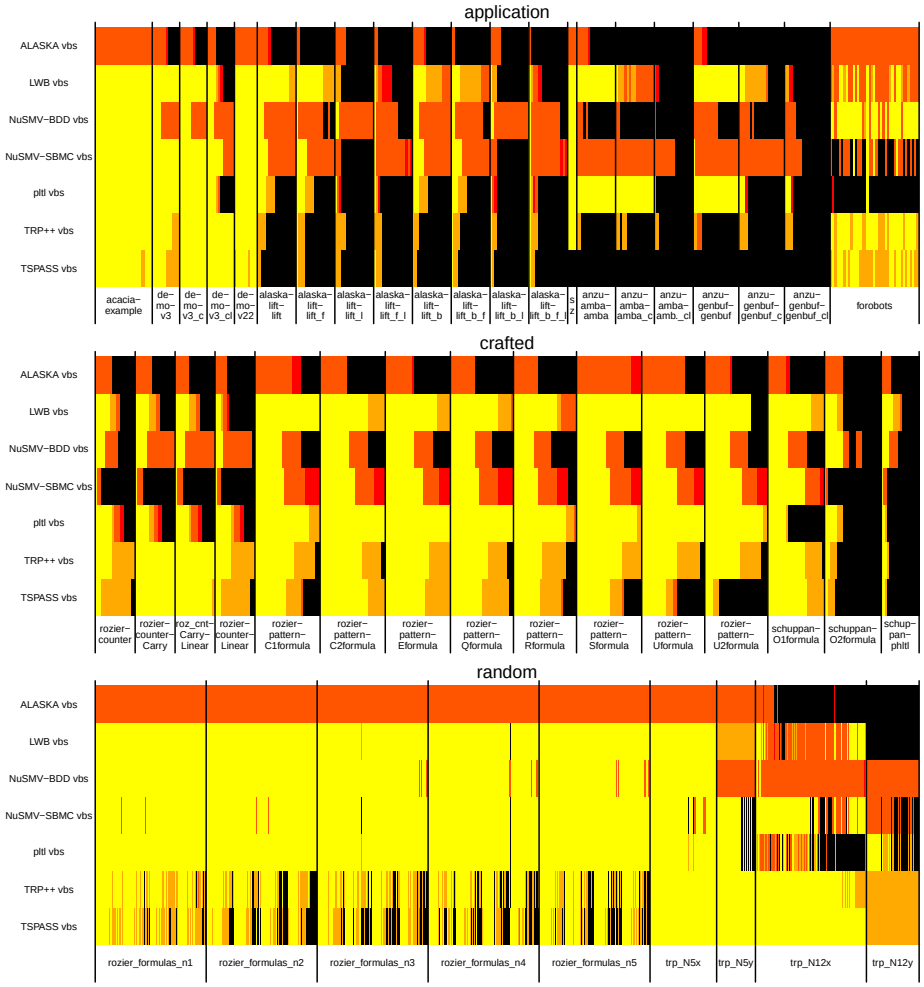

Fig. 9. Contour/discrete raw data plots of memory usage for the vbs of each tool (all instances). Instances are on the x-axes (only identified by their families), configurations on the y-axes. Each rectangle represents the memory usage of one configuration on one instance. **sz** abbreviates **alaska_szymanski**, **roz_cnt** abbreviates **rozier_counter**, and **demo** stands for **acacia_demo**. Memory usage is encoded using the following colors:
≤ 1 MB; > 1 MB, ≤ 10 MB; > 10 MB, ≤ 100 MB; > 100 MB, ≤ 2048 MB; unsolved.

D.2 Cactus Plots

Run Time

In Fig. 10 we show cactus plots of the run time for the winning configurations with model construction dis- or enabled for all instances.

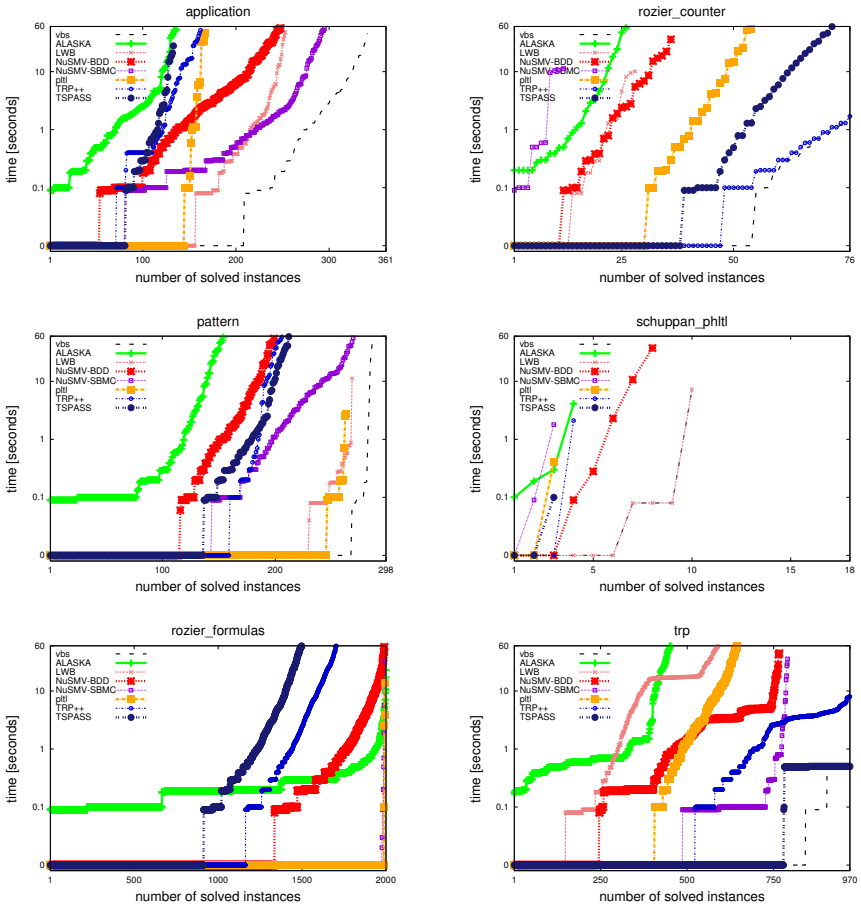

Fig. 10. Cactus plots of run time for the winning configurations with model construction dis- or enabled (all instances). The x-axes show the number of solved instances (sorted by increasing run time for each configuration). The y-axes show the run time in seconds. The upper left plot is category **application**; the upper right plot and both plots in the middle row are category **crafted** with families **rozier_counter**, **pattern**, and **schuppan_phltl**; the last row is category **random** with families **rozier_formulas** and **trp**.

Memory

Figure 11 is analogous to Fig. 10 but for memory usage.

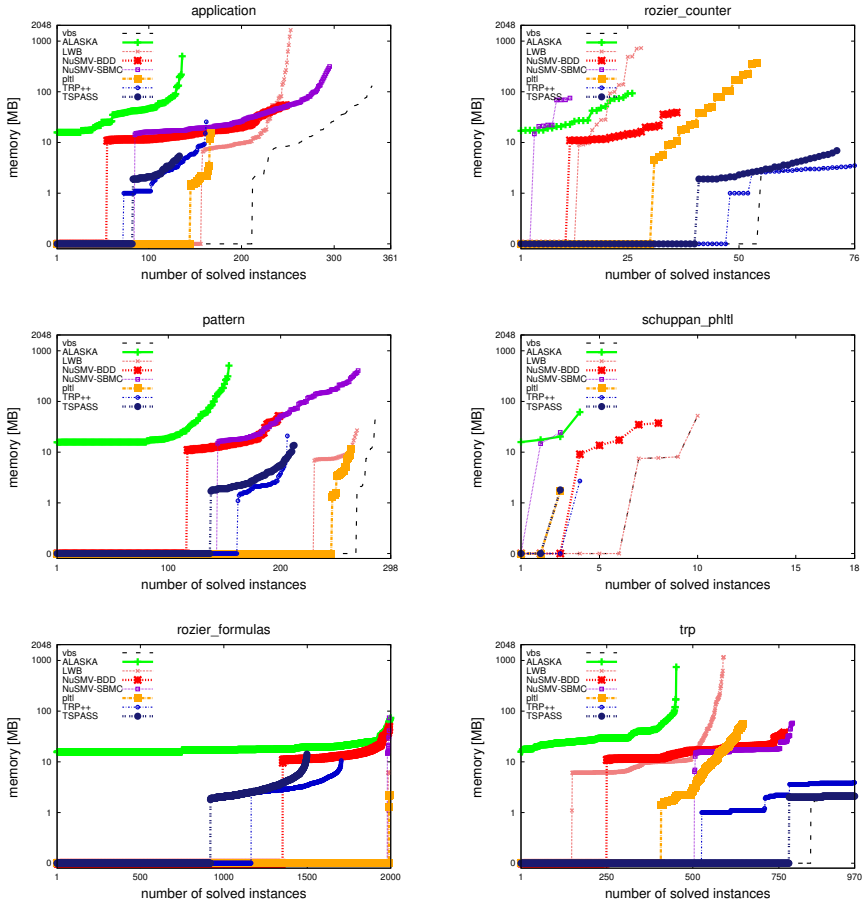

Fig. 11. Cactus plots of memory usage for the winning configurations with model construction dis- or enabled (all instances). The x-axes show the number of solved instances (sorted by increasing memory usage for each configuration). The y-axes show the memory usage in MB. The upper left plot is category **application**; the upper right plot and both plots in the middle row are category **crafted** with families **rozier_counter**, **pattern**, and **schuppan_phltl**; the last row is category **random** with families **rozier_formulas** and **trp**.

Run Time VBS

Figure 12 is analogous to Fig. 10 but using the vbs of each tool.

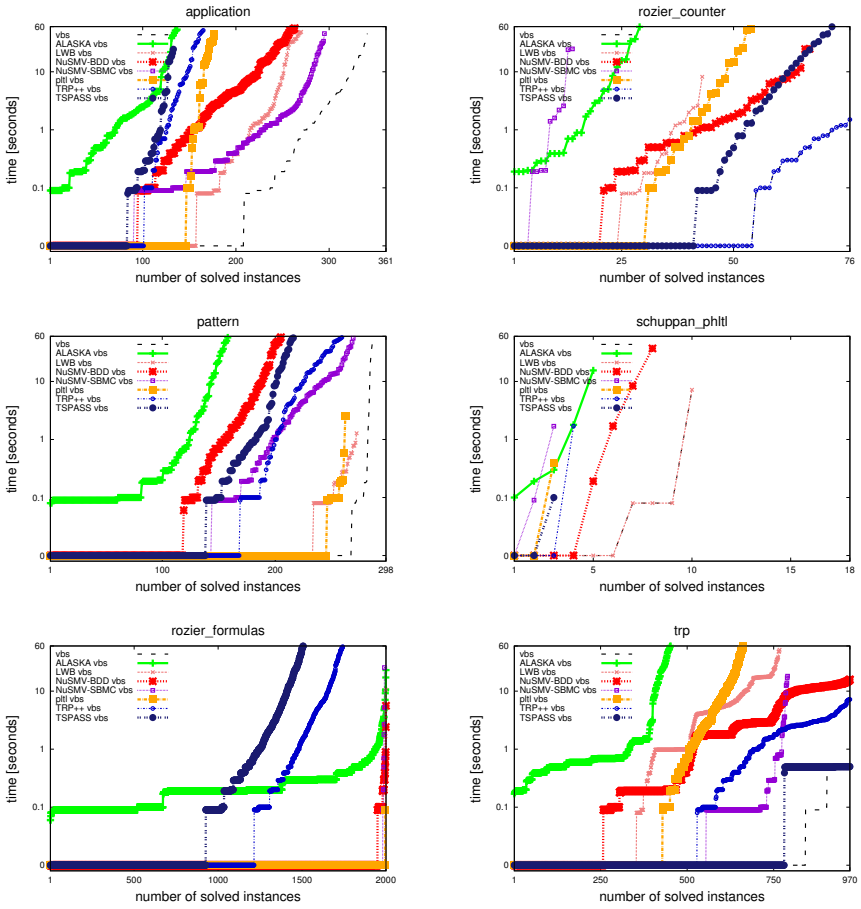

Fig. 12. Cactus plots of run time for the vbs of each tool (all instances). The x-axes show the number of solved instances (sorted by increasing run time for each configuration). The y-axes show the run time in seconds. The upper left plot is category **application**; the upper right plot and both plots in the middle row are category **crafted** with families **rozier_counter**, **pattern**, and **schuppan_phltl**; the last row is category **random** with families **rozier_formulas** and **trp**.

Memory VBS

Figure 13 is analogous to Fig. 11 but using the vbs of each tool.

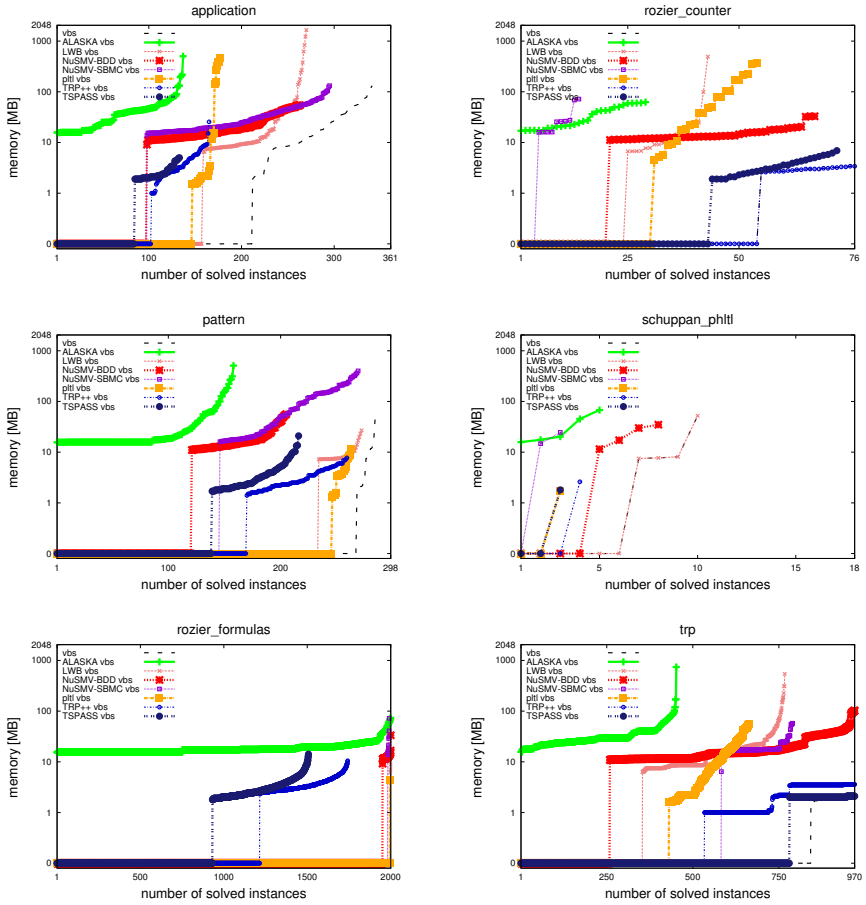

Fig. 13. Cactus plots of memory usage for the vbs of each tool (all instances). The x-axes show the number of solved instances (sorted by increasing memory usage for each configuration). The y-axes show the memory usage in MB. The upper left plot is category **application**; the upper right plot and both plots in the middle row are category **crafted** with families **rozier_counter**, **pattern**, and **schuppan_phltl**; the last row is category **random** with families **rozier_formulas** and **trp**.

D.3 Cactus Plots *Sat* versus *Unsat* Instances 1

Run Time

In Fig. 14 we show cactus plots of the run time for the winning configurations with model construction dis- or enabled split into *sat* and *unsat* instances. Here we show the more familiar cactus plots. See Fig. 18–21 in App. D.4 for alternative plots that use an equal distribution between *sat* and *unsat* instances and show the run time for *sat* and *unsat* instances for one configuration in one plot.

Memory

Figure 15 is analogous to Fig. 14 but for memory usage.

Run Time VBS

Figure 16 is analogous to Fig. 14 but using the vbs of each tool.

Memory VBS

Figure 17 is analogous to Fig. 15 but using the vbs of each tool.

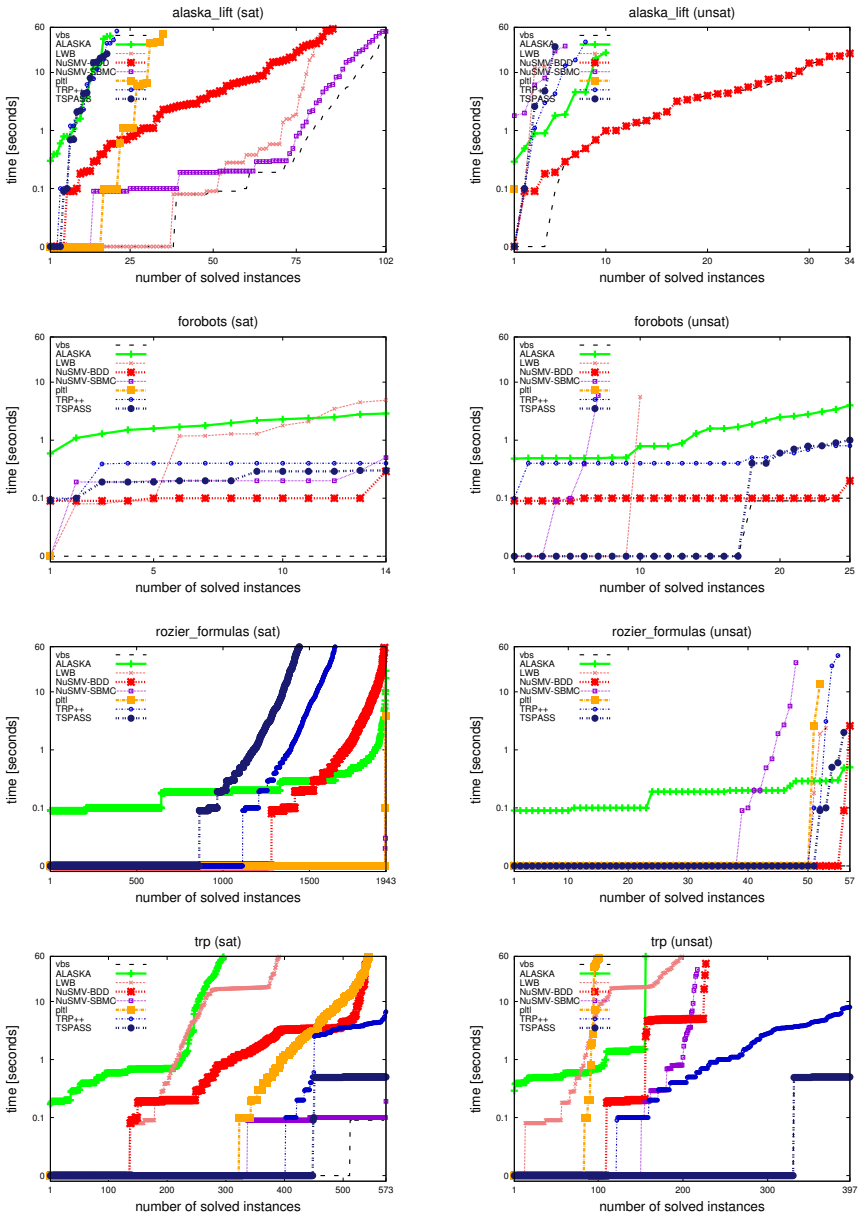

Fig. 14. Cactus plots of run time for the winning configurations with model construction dis- or enabled split into *sat* and *unsat* instances. The x-axes show the number of solved instances (sorted by increasing run time for each configuration). The y-axes show the run time in seconds. The left column shows *sat*, the right column *unsat* instances. Only families that contain both *sat* and *unsat* instances are shown (see Tab. 1 for numbers of *unsat* instances). We do not show **crafted** as there seems to be too little commonality between the *sat* and the *unsat* subfamilies to be useful.

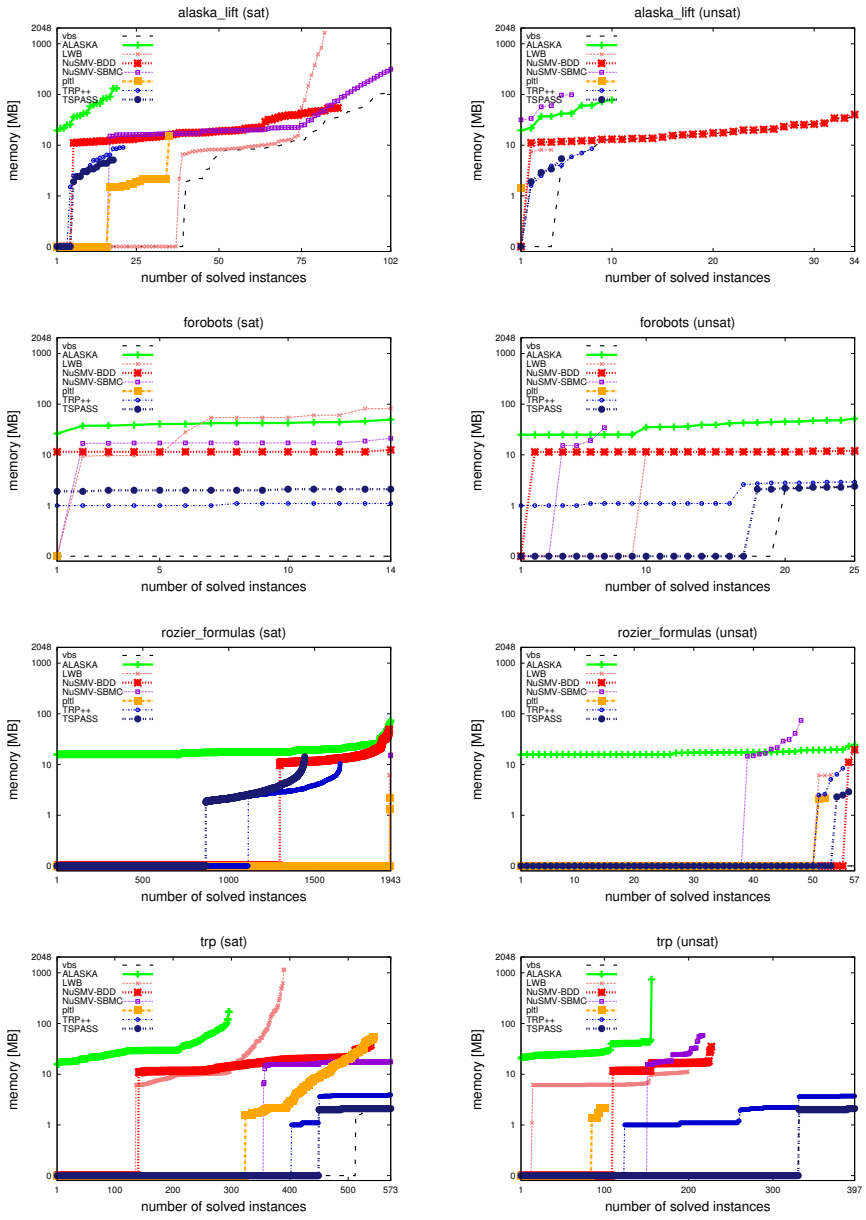

Fig. 15. Cactus plots of memory usage for the winning configurations with model construction dis- or enabled split into *sat* and *unsat* instances. The x-axes show the number of solved instances (sorted by increasing memory usage for each configuration). The y-axes show the memory usage in MB. The left column shows *sat*, the right column *unsat* instances. Only families that contain both *sat* and *unsat* instances are shown (see Tab. 1 for numbers of *unsat* instances). We do not show **crafted** as there seems to be too little commonality between the *sat* and the *unsat* subfamilies to be useful.

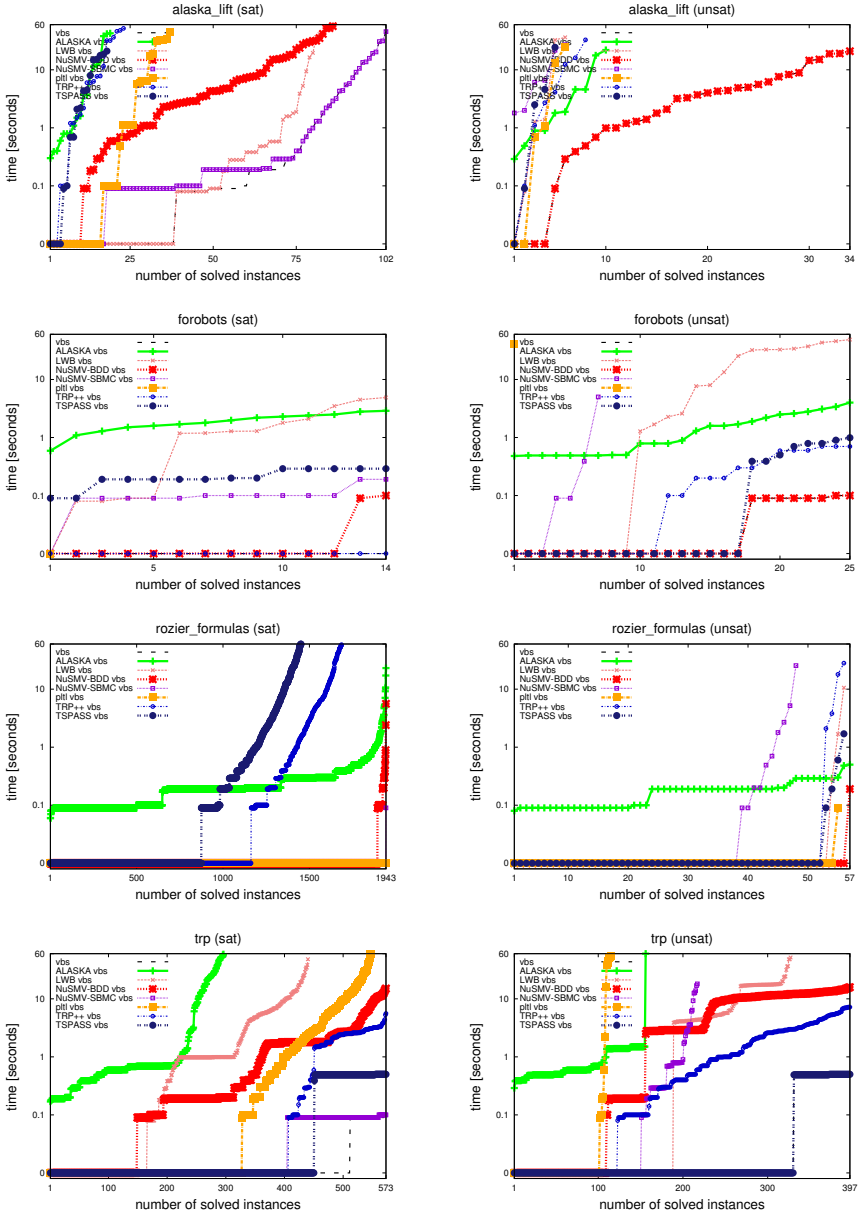

Fig. 16. Cactus plots of run time for the vbs of each tool split into *sat* and *unsat* instances. The x-axes show the number of solved instances (sorted by increasing run time for each configuration). The y-axes show the run time in seconds. The left column shows *sat*, the right column *unsat* instances. Only families that contain both *sat* and *unsat* instances are shown (see Tab. 1 for numbers of *unsat* instances). We do not show **crafted** as there seems to be too little commonality between the *sat* and the *unsat* subfamilies to be useful.

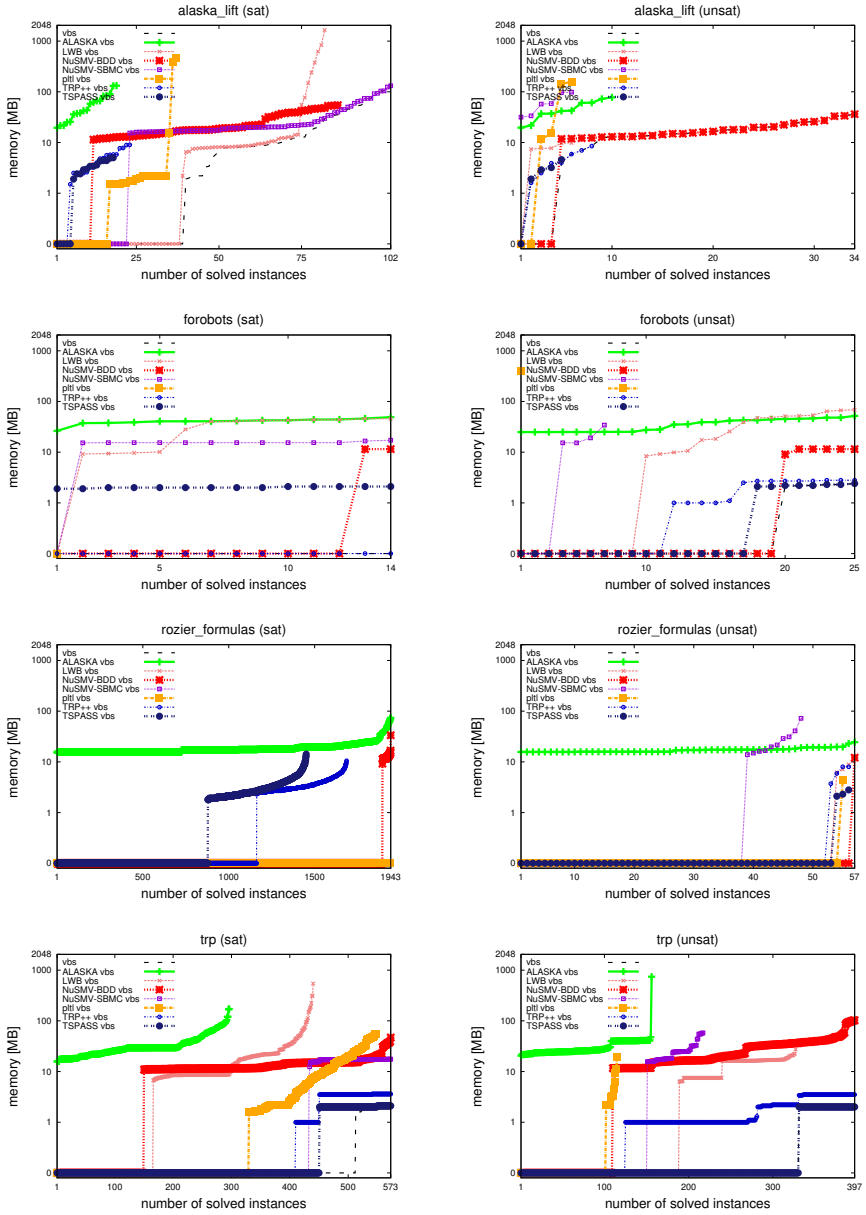

Fig. 17. Cactus plots of memory usage for the vbs of each tool split into *sat* and *unsat* instances. The x-axes show the number of solved instances (sorted by increasing memory usage for each configuration). The y-axes show the memory usage in MB. The left column shows *sat*, the right column *unsat* instances. Only families that contain both *sat* and *unsat* instances are shown (see Tab. 1 for numbers of *unsat* instances). We do not show **crafted** as there seems to be too little commonality between the *sat* and the *unsat* subfamilies to be useful.

D.4 Cactus Plots *Sat* versus *Unsat* Instances 2

Run Time

Figure 18 shows cactus plots comparing the run time for the winning configurations with model construction dis- or enabled for *sat* and *unsat* instances. In **alaska_lift** the *unsat* instances are found in **lift_l** and **lift_b_l**. For corresponding *sat* instances we took the families that are most similar, namely, **lift_f_l** and **lift_b_f_l**. The families **rozier_formulas** and **trp** are not homogenuous but consist of subfamilies with different parameters such as size or number of distinct atomic propositions. The distribution between *sat* and *unsat* instances is not uniform across parameters. To avoid bias in Fig. 18 only a subset of instances is used there such that the number of *sat* and *unsat* instances per combination of parameters is the same. This is achieved by dropping "superfluous" instances. This inherently assumes that within one combination of parameters instances exhibit similar characteristics; we visually checked two such samples (one of which is shown) and found little difference in the relevant general trends.

Memory

Figure 19 is analogous to Fig. 18 but for memory usage.

Run Time VBS

Figure 20 is analogous to Fig. 18 but using the vbs of each tool.

Memory VBS

Figure 21 is analogous to Fig. 19 but using the vbs of each tool.

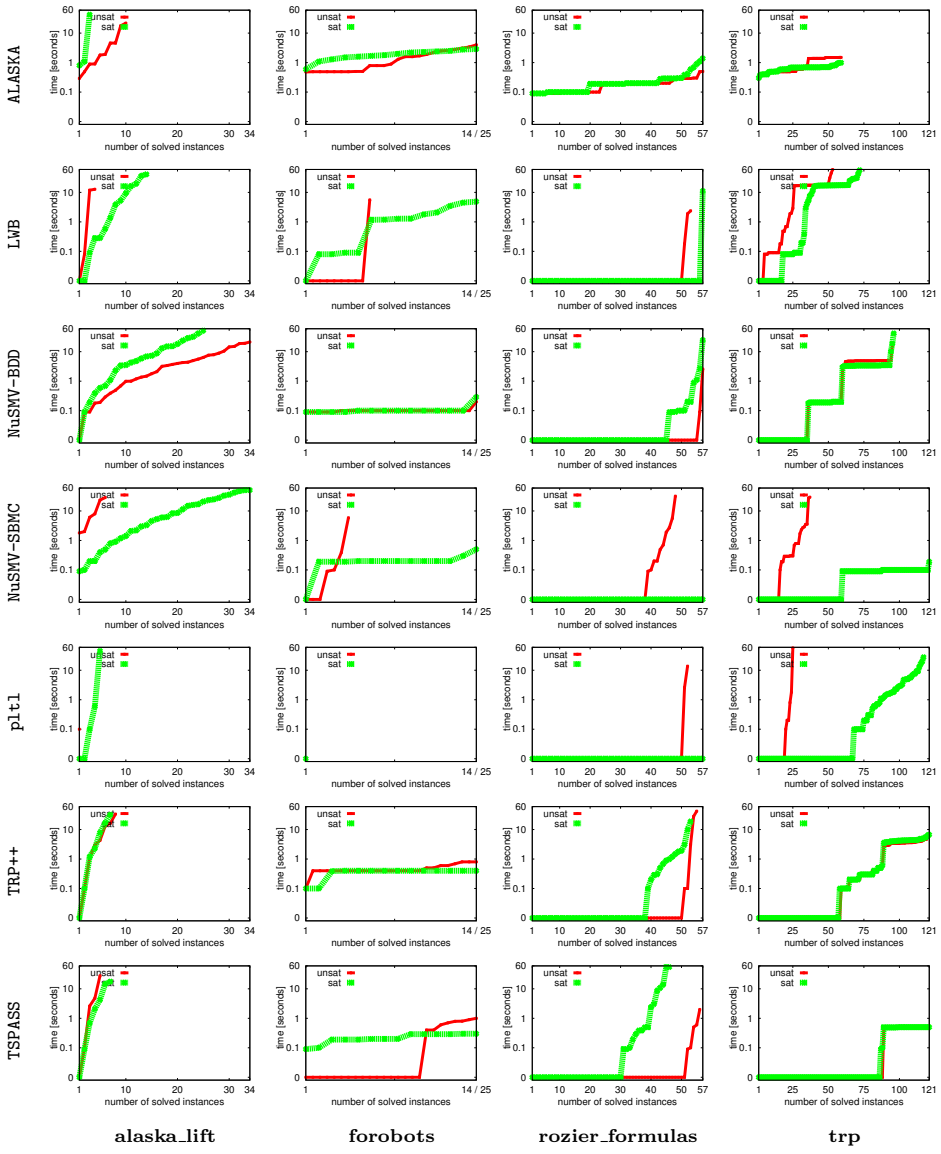

Fig. 18. Cactus plots of run time for the winning configurations with model construction dis- or enabled comparing *sat* (green, thick line) and *unsat* (red, thin line) instances. The x-axes show the number of solved instances (sorted by increasing run time for each configuration). The y-axes show the run time in seconds. Each line represents a configuration, each column a family of instances. For **rozier_random** and **trp** not all instances are considered (see text). Note that there are only 14 *sat* but 25 *unsat* instances in **forobots** so that different scales on the x-axis are used.

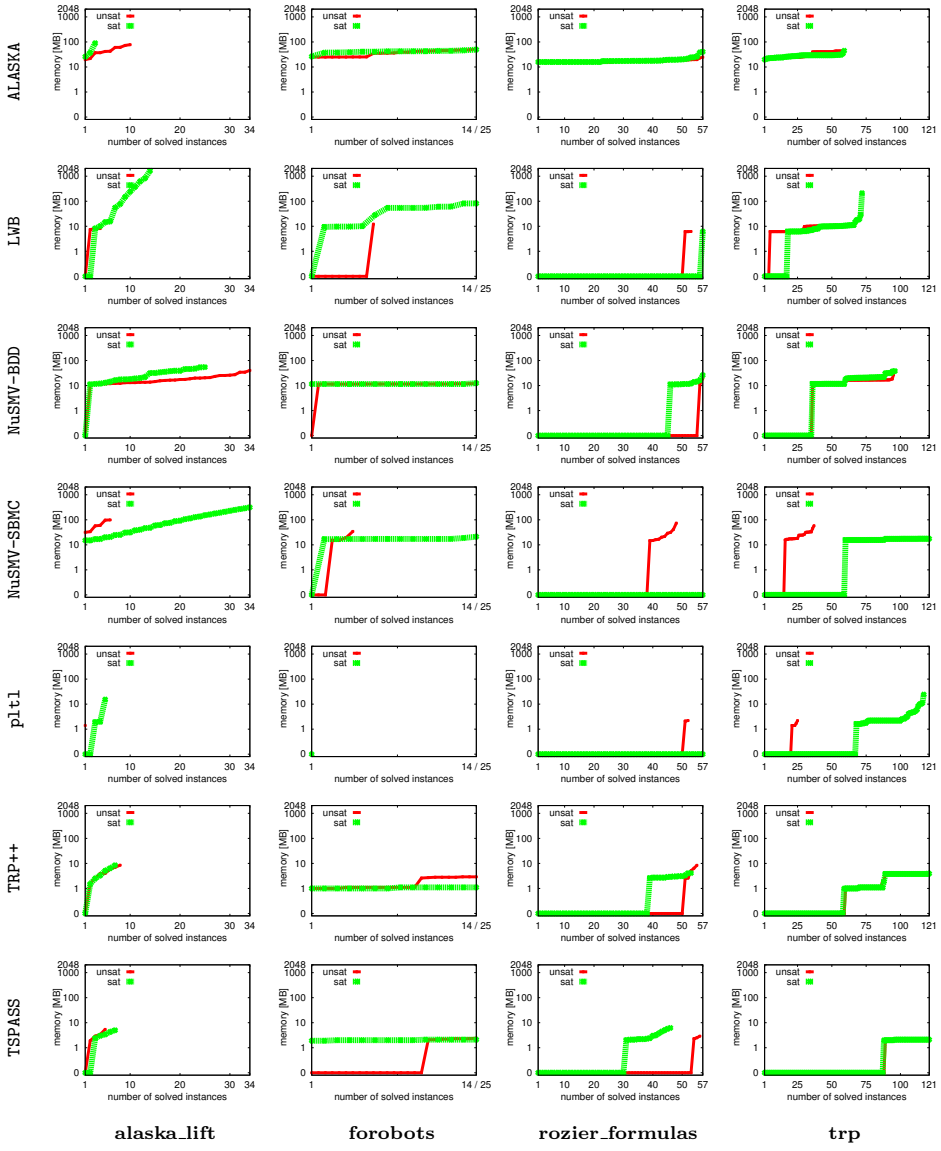

Fig. 19. Cactus plots of memory usage for the winning configurations with model construction dis- or enabled comparing *sat* (green, thick line) and *unsat* (red, thin line) instances. The x-axes show the number of solved instances (sorted by increasing memory usage for each configuration). The y-axes show the memory usage in MB. Each line represents a configuration, each column a family of instances. For **rozier_random** and **trp** not all instances are considered (see text). Note that there are only 14 *sat* but 25 *unsat* instances in **forobots** so that different scales on the x-axis are used.

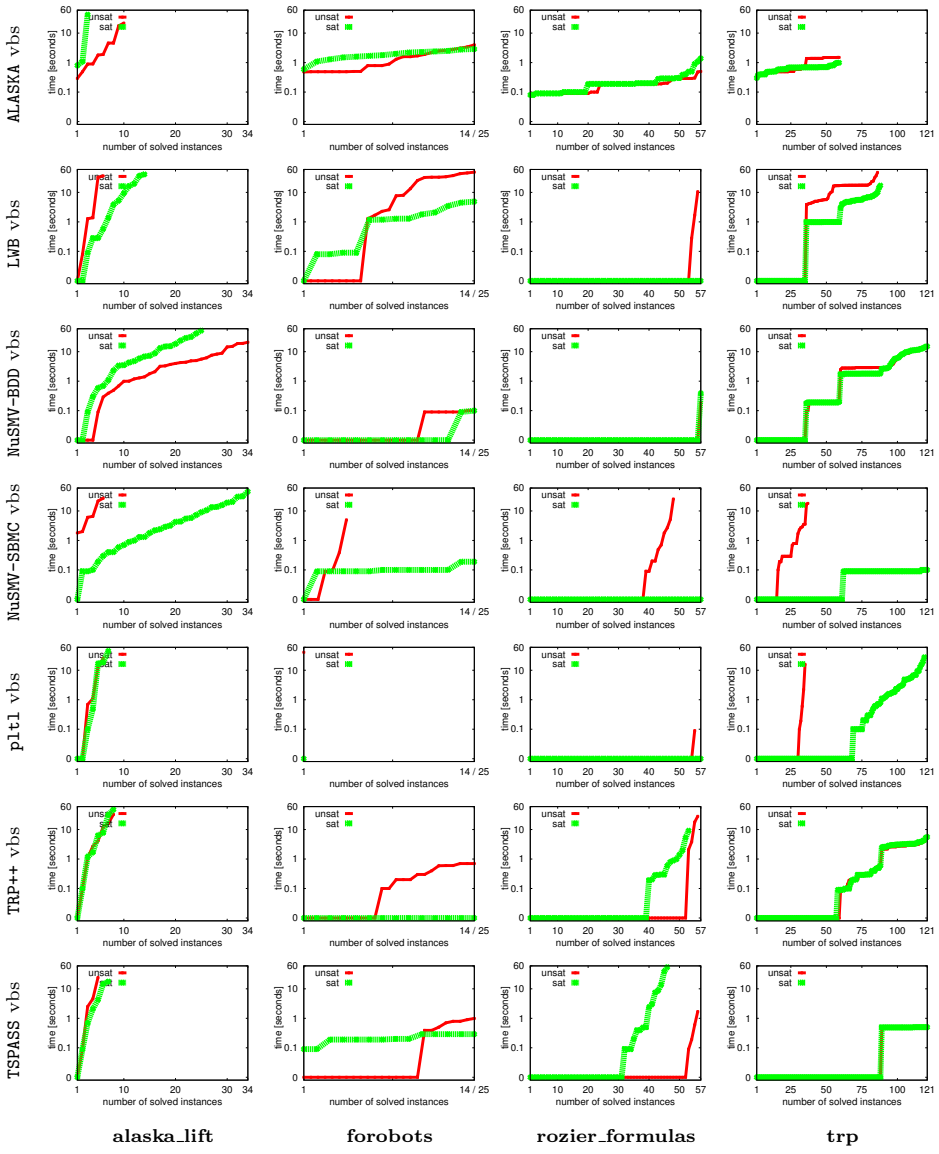

Fig. 20. Cactus plots of run time for the vbs of each tool comparing *sat* (green, thick line) and *unsat* (red, thin line) instances. The x-axes show the number of solved instances (sorted by increasing run time for each configuration). The y-axes show the run time in seconds. Each line represents a configuration, each column a family of instances. For **rozier_random** and **trp** not all instances are considered (see text). Note that there are only 14 *sat* but 25 *unsat* instances in **forobots** so that different scales on the x-axis are used.

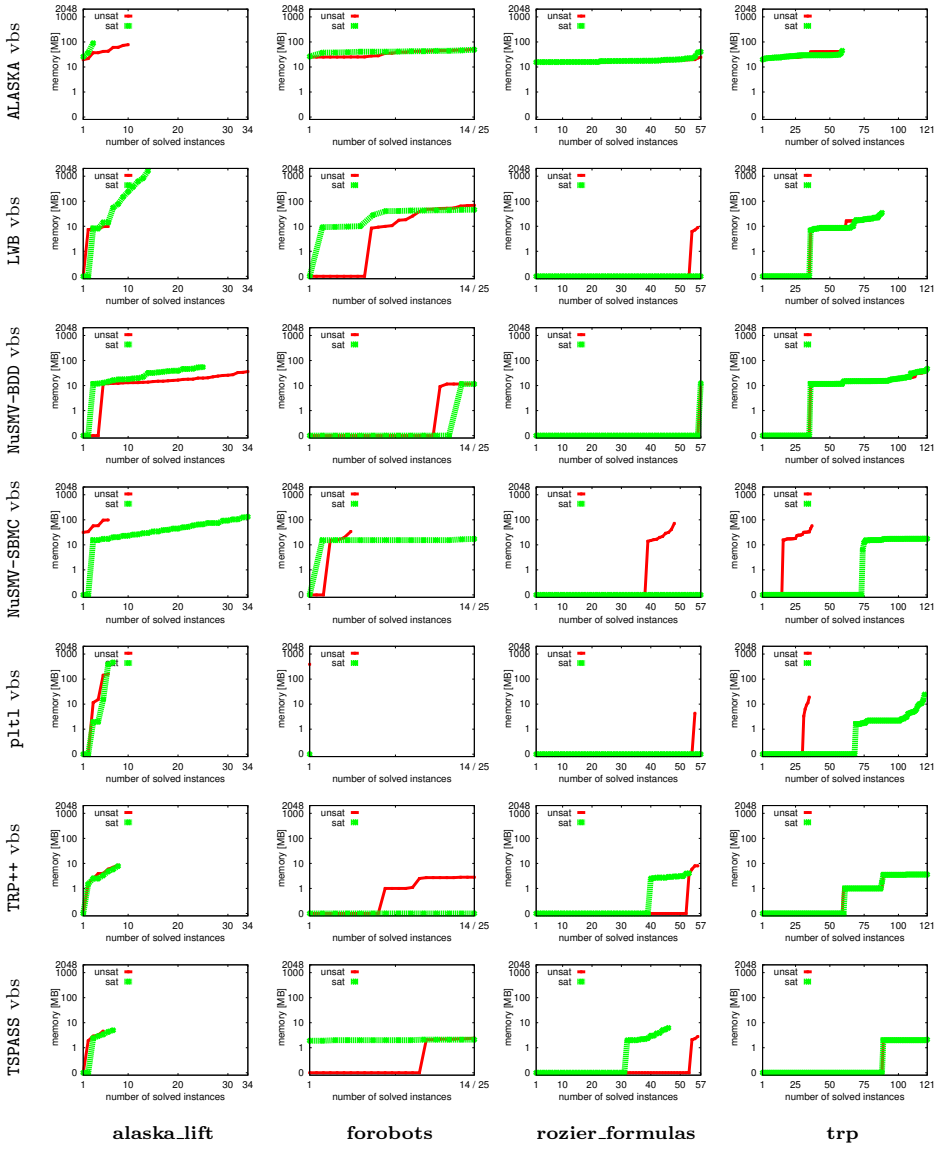

Fig. 21. Cactus plots of memory usage for the vbs of each tool comparing *sat* (green, thick line) and *unsat* (red, thin line) instances. The x-axes show the number of solved instances (sorted by increasing memory usage for each configuration). The y-axes show the memory usage in MB. Each line represents a configuration, each column a family of instances. For **rozier_random** and **trp** not all instances are considered (see text). Note that there are only 14 *sat* but 25 *unsat* instances in **forobots** so that different scales on the x-axis are used.

D.5 Plots Run Time and Memory versus Size

Run Time

Figure 22 shows plots of the run time versus instance size for the winning configurations with model construction dis- or enabled (all instances).

Memory

Figure 23 is analogous to Fig. 22 but for memory usage.

Run Time VBS

Figure 24 is analogous to Fig. 22 but using the vbs of each tool.

Memory VBS

Figure 25 is analogous to Fig. 23 but using the vbs of each tool.

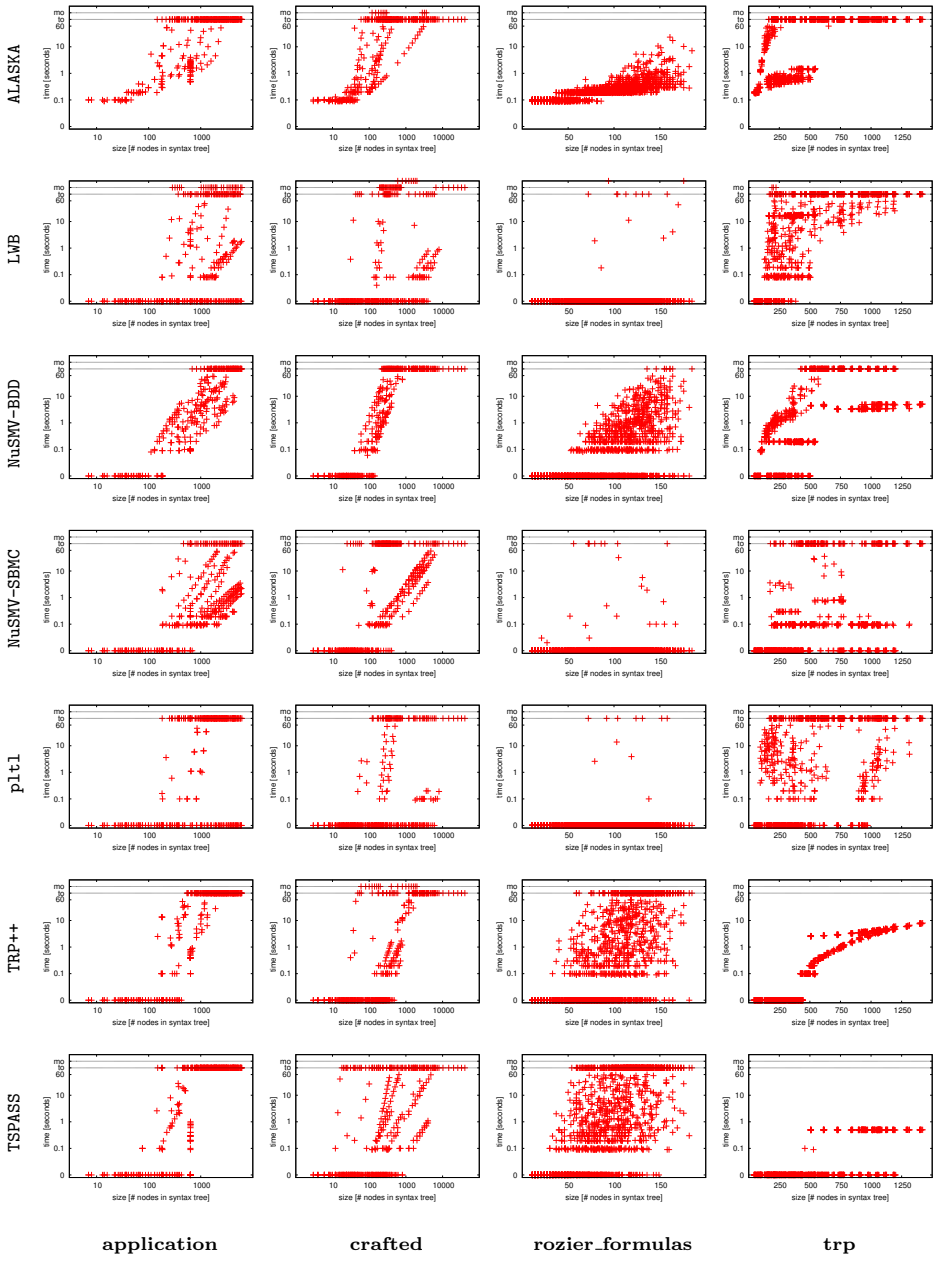

Fig. 22. Plots of run time for the winning configurations with model construction disor enabled versus instance size. The x-axes show the instance size given by the number of nodes in the syntax tree. The y-axes show the run time in seconds. Notice that the x-axes are logarithmic for **application** and **crafted** and linear for **rozier_formulas** and **trp**. Each line represents a configuration, each column a family.

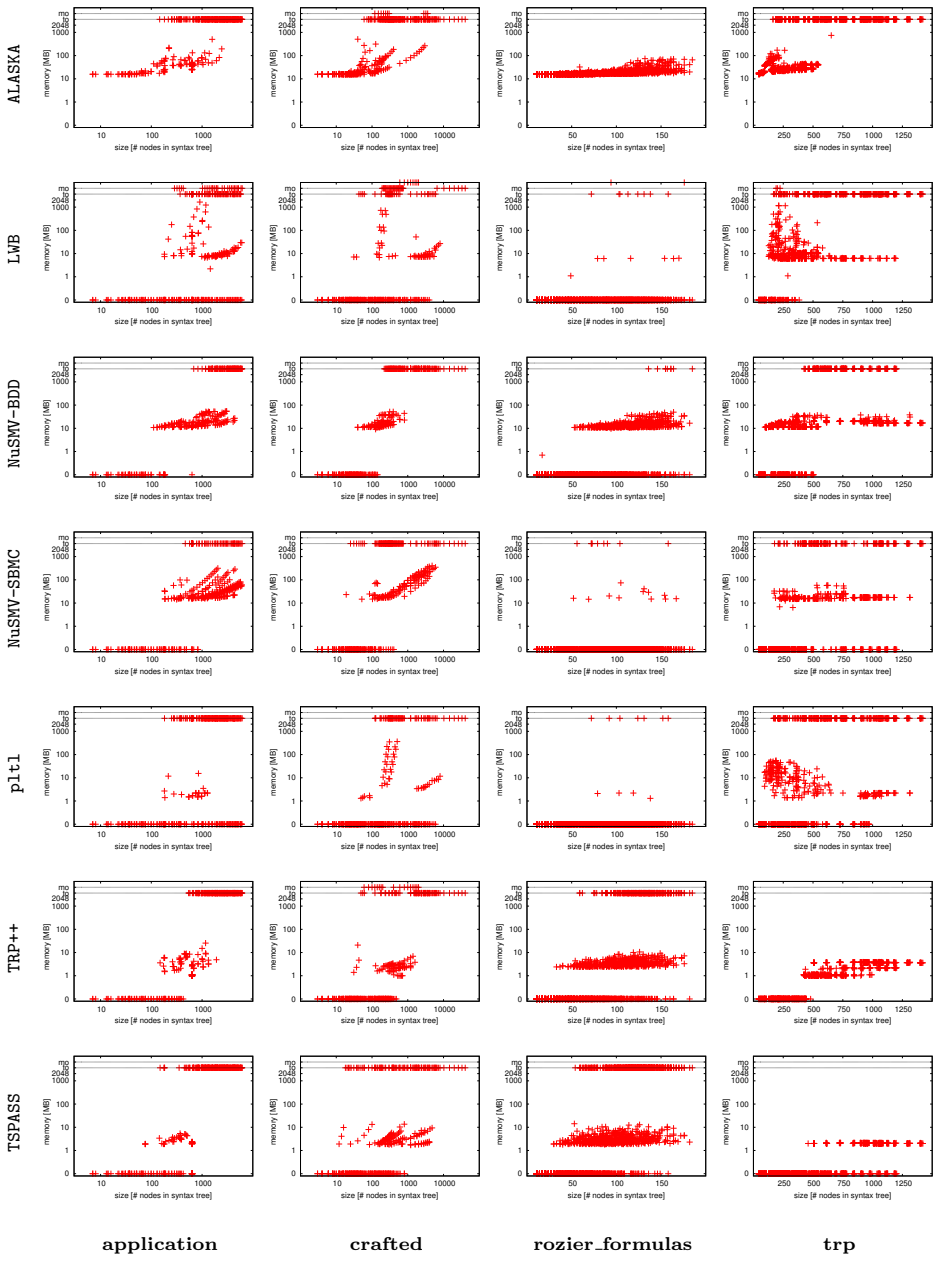

Fig. 23. Plots of memory usage for the winning configurations with model construction dis- or enabled versus instance size. The x-axes show the instance size given by the number of nodes in the syntax tree. The y-axes show the memory usage in MB. Notice that the x-axes are logarithmic for **application** and **crafted** and linear for **rozier_formulas** and **trp**. Each line represents a configuration, each column a family.

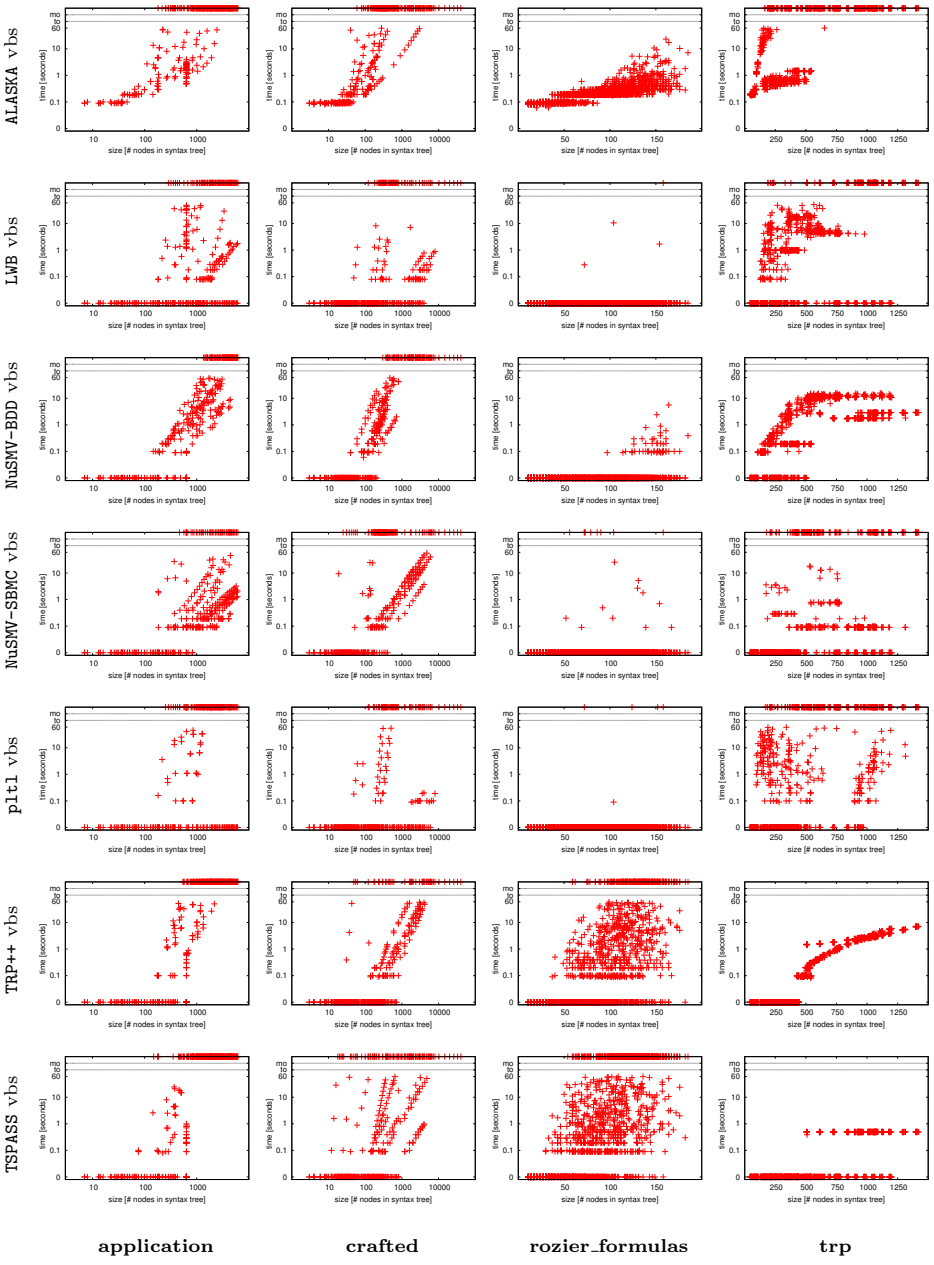

Fig. 24. Plots of run time for the vbs of each tool with model construction dis- or enabled versus instance size. The x-axes show the instance size given by the number of nodes in the syntax tree. The y-axes show the run time in seconds. Notice that the x-axes are logarithmic for **application** and **crafted** and linear for **rozier_formulas** and **trp**. Each line represents a configuration, each column a family.

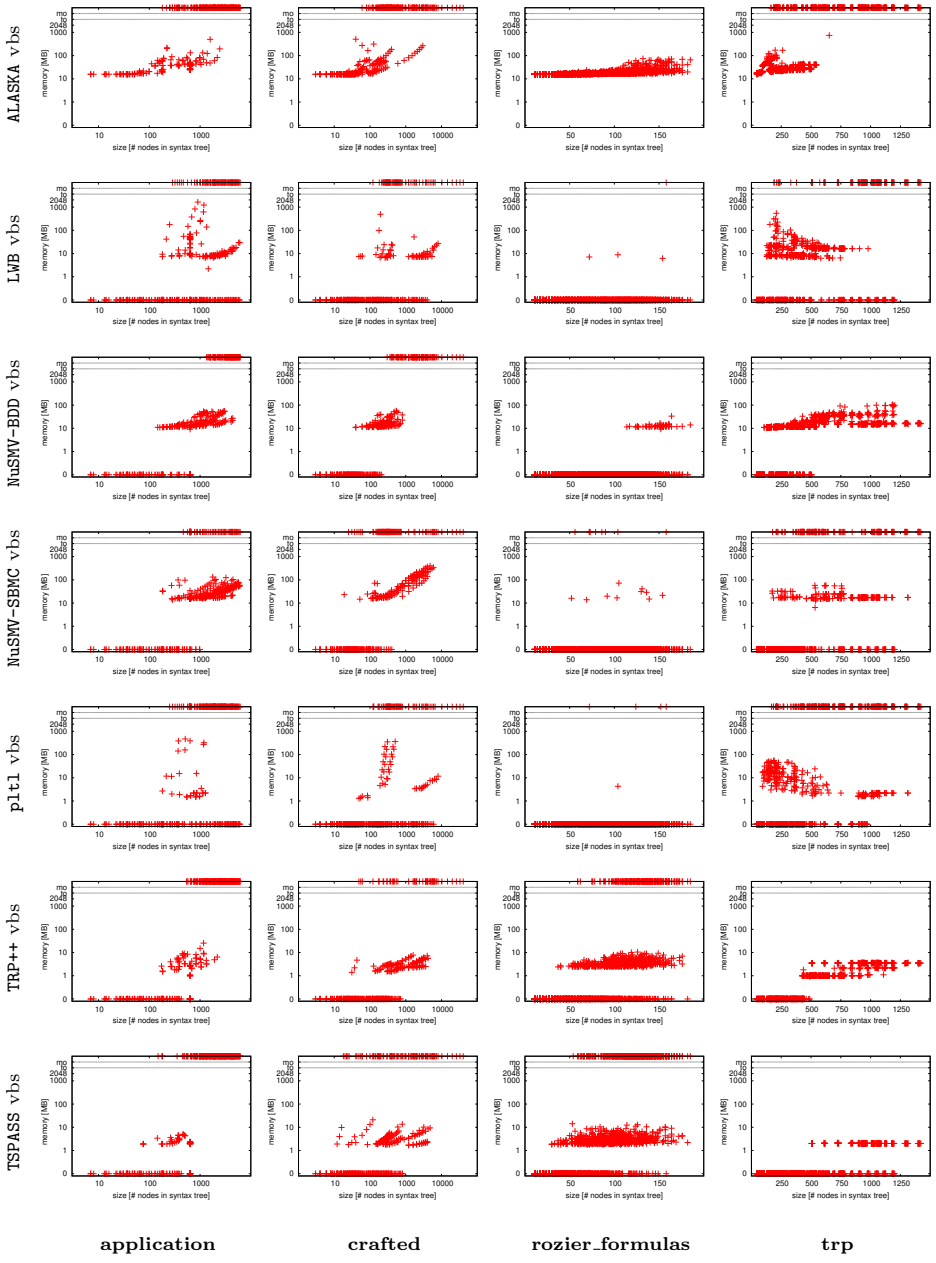

Fig. 25. Plots of memory usage for the vbs of each tool with model construction disor enabled versus instance size. The x-axes show the instance size given by the number of nodes in the syntax tree. The y-axes show the memory usage in MB. Notice that the x-axes are logarithmic for **application** and **crafted** and linear for **rozier_formulas** and **trp**. Each line represents a configuration, each column a family.

D.6 Scatter Plots Run Time Non-negated Versus Negated Instances of rozier_formulas

Figure 26 shows the variation in run time on non-negated versus negated versions of instances of **rozier_formulas** for the winning configurations with model construction disabled.

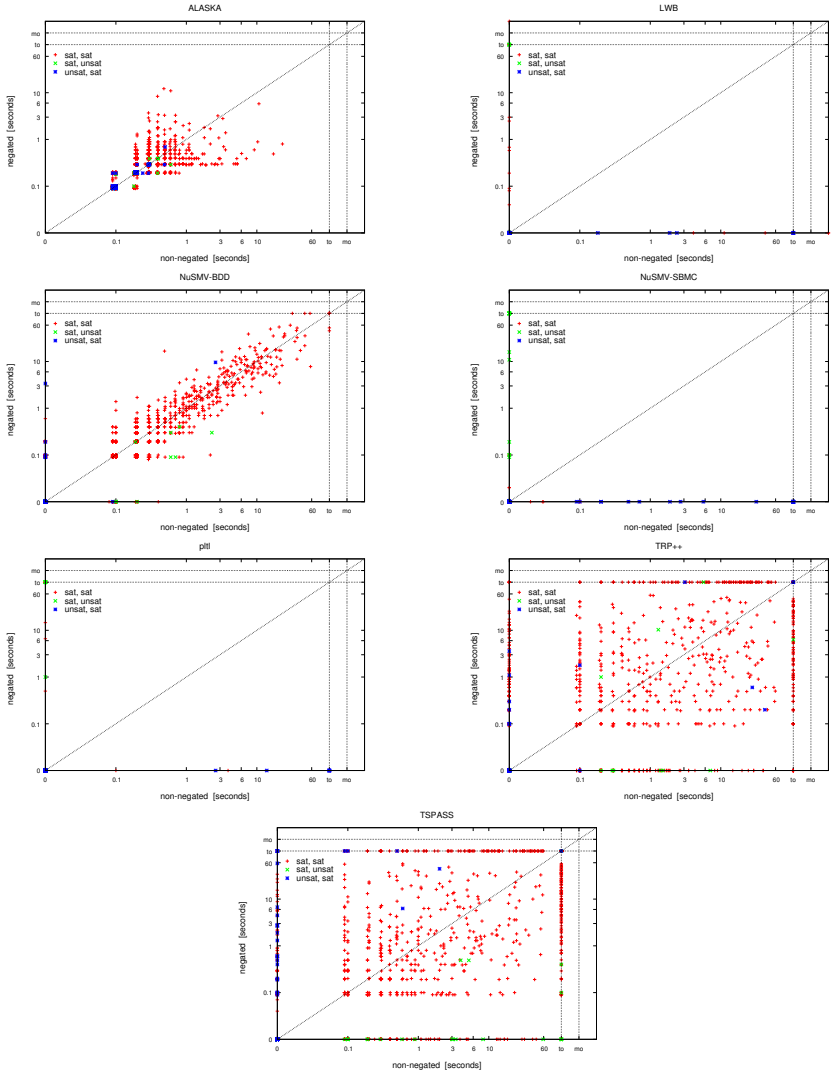

Fig. 26. Scatter plots of run time in seconds for the winning configurations on non-negated (x-axis) versus negated versions (y-axis) of instances of **rozier_formulas**.

D.7 Scatter Plots Run Time Model Construction Enabled Versus Disabled

Figure 27 shows the variation in run time between model construction enabled and disabled on all *sat* instances for the winning configurations with model construction enabled.

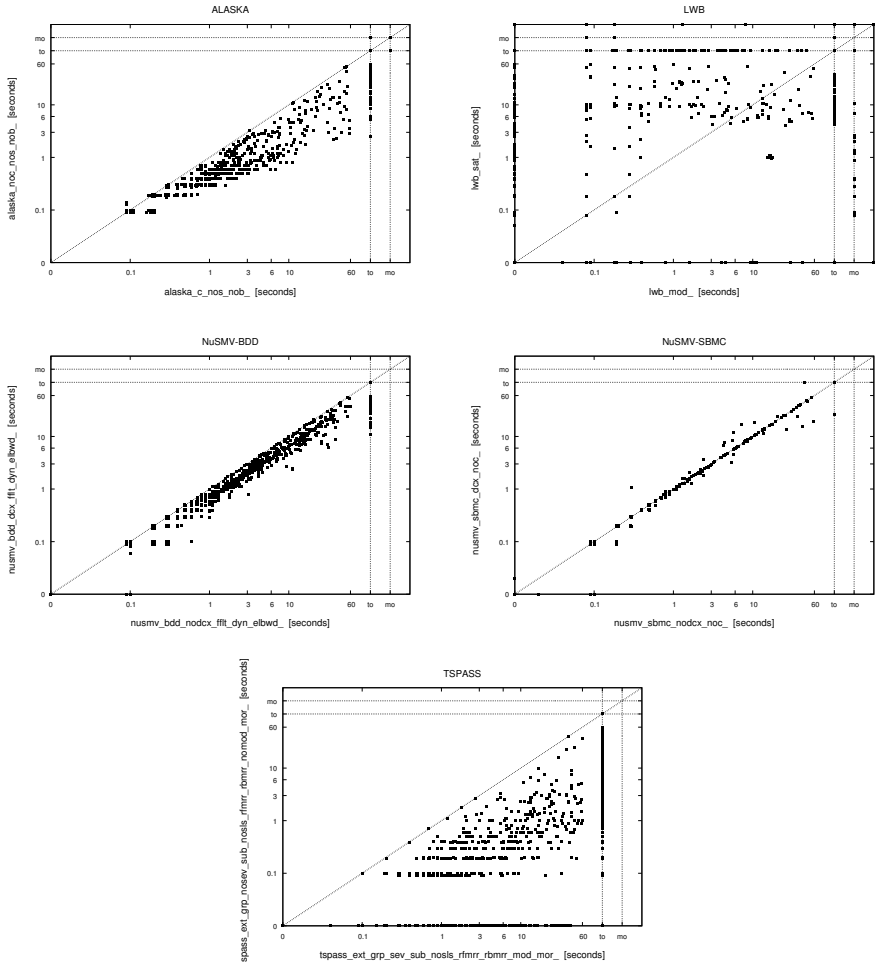

Fig. 27. Scatter plots of run time in seconds between model construction enabled (x-axis) and disabled (y-axis) on all *sat* instances for the winning configurations with model construction enabled.

D.8 Portfolio Plots

Perfect Oracle

Figure 28 is an enlarged version of Fig. 4.

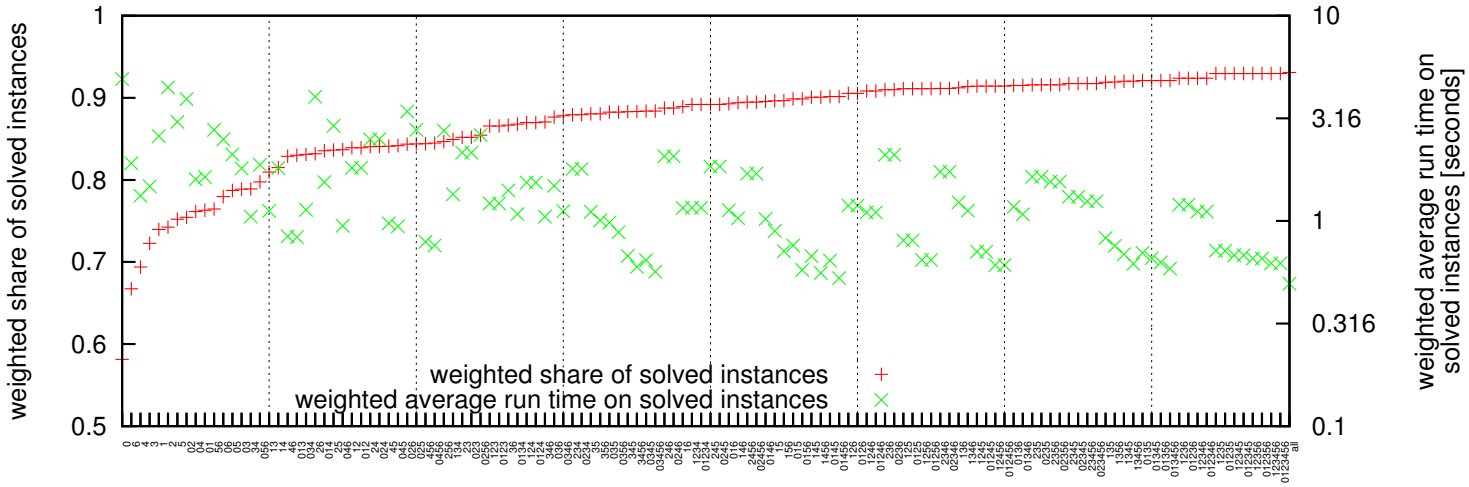

Fig. 28. Potential of a portfolio solver consisting of subsets of the winning configurations with model construction dis- or enabled using a perfect oracle. Portfolios are identified by their constituent configurations: 0: `ALASKA`; 1: `LWB`; 2: `NuSMV-BDD`; 3: `NuSMV-SBMC`; 4: `pltl`; 5: `TRP++`; 6: `TSPASS`. On the x-axis are the portfolios sorted in increasing order of weighted share of solved instances; ties are broken by decreasing order of weighted average run time on solved instances. For each portfolio the weighted share of solved instances is marked by a red vertical/horizontal cross (scale on the left y-axis); the corresponding weighted average run time on solved instances is marked by a green diagonal cross (scale on the right y-axis, in seconds). "all" considers all configurations with model construction dis- or enabled rather than only the winning configurations.

Perfect Task Switcher

Figure 29 is analogous to Fig. 28 but using perfect task switcher mode.

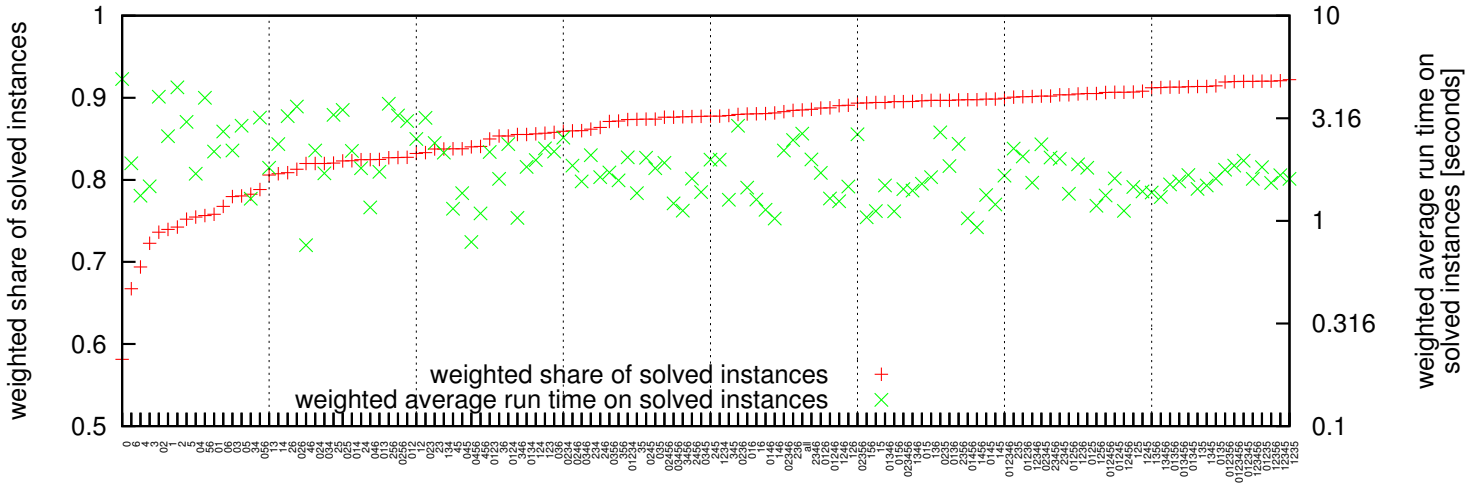

Fig. 29. Potential of a portfolio solver consisting of subsets of the winning configurations with model construction dis- or enabled using a perfect task switcher. Portfolios are identified by their constituent configurations: 0: `ALASKA`; 1: `LWB`; 2: `NuSMV-BDD`; 3: `NuSMV-SBMC`; 4: `pltl`; 5: `TRP++`; 6: `TSPASS`. On the x-axis are the portfolios sorted in increasing order of weighted share of solved instances; ties are broken by decreasing order of weighted average run time on solved instances. For each portfolio the weighted share of solved instances is marked by a red vertical/horizontal cross (scale on the left y-axis); the corresponding weighted average run time on solved instances is marked by a green diagonal cross (scale on the right y-axis, in seconds). "all" considers all configurations with model construction dis- or enabled rather than only the winning configurations.