Parallel SAT Solver Selection and Scheduling

Yuri Malitsky[1], Ashish Sabharwal[2], Horst Samulowitz[2], and Meinolf Sellmann[2]

[1] Cork Constraint Computation Centre, University College Cork, Ireland
y.malitsky@4c.ucc.ie
[2] IBM Watson Research Center, Yorktown Heights, NY 10598, USA
{ashish.sabharwal,samulowitz,meinolf}@us.ibm.com

Abstract. Combining differing solution approaches by means of solver portfolios has proven as a highly effective technique for boosting solver performance. We consider the problem of generating parallel SAT solver portfolios. Our approach is based on a recently introduced sequential SAT solver portfolio that excelled at the last SAT competition. We show how the approach can be generalized for the parallel case, and how obstacles like parallel SAT solvers and symmetries induced by identical processors can be overcome. We compare different ways of computing parallel solver portfolios with the best performing parallel SAT approaches to date. Extensive experimental results show that the developed methodology very significantly improves our current parallel SAT solving capabilities.

1 Introduction and Related Work

In the past decade, solver portfolios have boosted our capability to solve hard combinatorial problems. Portfolios of existing solution algorithms have excelled in competitions in satisfiability (SAT), constraint programming (CP), and quantified Boolean formulae (QBF) [8, 12, 14].

In the past years, a new trend has emerged, namely the development of parallel solver portfolios. The gold-winning ManySAT solver [6] is, when we ignore features like clause-sharing, a static parallel portfolio of the MiniSAT solver [5] with different parameterizations. At the SAT Competition 2011, an extremely simple static parallel portfolio, ppfolio [10], dominated the wall-clock categories on random and crafted SAT instances and came very close to winning the applications category as well.

The obvious next step is to consider dynamic parallel portfolios, i.e., portfolios that are composed based on the features of the given problem instance. Traditionally, sequential portfolios simply *select* one of the constituent solvers which appears best suited for the given problem instance. At least since the invention of CP-Hydra [8] and SatPlan [13], sequential portfolios also *schedule* solvers. That is, they may select more than just one constituent solver and assign each one a portion of the time available for solving the given instance. Yun and Epstein [15] introduced a heuristic method to build dynamic parallel portfolios. The method relies heavily on the observation that running a deterministic solver on more than one processor is a waste of time and is thus limited to the use of sequential SAT solvers only. A similar restriction applies to the work by Petrik and Zilberstein [9]. They introduced a method to compute static parallel schedules that are optimal with respect to the training instances, based on formulating the problem as a non-linear optimization problem and considering only sequential constituent solvers.

The best-performing *sequential* dynamic portfolio at the SAT Competition 2011 was 3S [7] where it won gold medals in the CPU-time category on random and crafted instances. 3S combines a fixed-time static solver schedule with the dynamic selection of *one* long-running SAT solver. To compute the static schedule offline and to select the long-running solver online, 3S combines low-bias nearest neighbor regression with integer programming optimization. In this paper, we augment this methodology to devise dynamic parallel SAT portfolios which include parallel SAT solvers.

2 SAT Solver Selector (3S)

Before considering the challenges of parallel portfolios, let us first review sequential 3S in more detail. 3S works in two phases, an offline learning phase, and an online execution phase.

- At Runtime: In the execution phase, as all dynamic solver portfolios, 3S first computes *features* of the given problem instance. In particular, 3S uses the same 48 core features as SATzilla [14]. Then, 3S selects $k \in \mathbb{N}$ instances that are most "similar" to the given instance in a training set of SAT instances for which 3S knows all runtimes of all solvers. Similarity in 3S is determined by the Euclidean distance of the (normalized) feature vectors of the given instance and the training instances. 3S selects the solver that can solve most of these k instances within the given time limit (ties are broken by shorter runtime). Finally, 3S first runs a fixed schedule of solvers for 10% of the time limit and then runs the selected solver for the remaining 90% of the available time.
- Offline: In the learning phase, which takes place during the development of the portfolio solver, 3S considers three tasks:
  1. Computation of features and simulation of solvers on all instances to determine their runtime on all training instances.
  2. To compute a desirable size k of the neighborhood, 3S employs a cross validation by random subsampling. That is, 3S repeatedly splits the training set into a base and a validation set and determines which size of k results in the best average validation set performance when using only the base set training instances to determine the long running solver.
  3. Lastly, 3S computes the fixed schedule of solvers that are run for 10% of the competition runtime. The objective when producing this schedule is to maximize the number of instances that can be solved within this reduced time limit. Among schedules that can solve the same number of instances, 3S selects one that minimizes the runtime of the schedule and

then scales this shorter schedule back to the 10% time limit by increasing the runtime of each solver in the schedule proportionally.

2.1 Schedule Computation

This last step deserves our special attention as it is at the core of what we will need to do to generalize the 3S methodology for the case of parallel solver execution. The problem again is to select a schedule of solvers – that is, a sequence of solvers with associated runtimes – that maximizes the number of instances solved within the reduced time limit. This problem is obviously an optimization problem, and it actually resembles a bit the set covering problem.

3S considers the following integer program (IP) to compute a solver schedule. Let $V_{S,t}$ denote the set of instances i that can be solved by solver S within time limit t.

Solver Scheduling IP

$$\begin{aligned}
&\min\ (C+1)\sum_i y_i + \sum_{S,t} t x_{S,t} \\
&s.t.\ y_i + \sum_{(S,t)\ \mid\ i \in V_{S,t}} x_{S,t} \geq 1 \qquad \forall i \\
&\quad \sum_{S,t} t x_{S,t} \leq C \\
&\quad y_i, x_{S,t} \in \{0,1\} \qquad \forall i, S, t
\end{aligned}$$

For all pairs of solvers S and time limits t, there is one variable $x_{S,t}$. Note that there are a number of solvers times the number of training instances of such variables as for each solver S 3S only considers time limits t where the solver just solves an instance in the training set. $x_{S,t}$ will be equal one if and only if, in our schedule, we will run solver S for t seconds.

The second set of variables are the y_i, one for each training instance i. Variable y_i will be one if and only if the solver schedule cannot solve instance i.

The first set of constraints ensure that each instance is covered – either because one of the selected solver/time pairs means that the respective solver can solve the instance in the allocated time, or because the instance is counted as not covered by y_i. The final knapsack constraint simply ensures that the total schedule time does not exceed the reduced time limit C.

The objective is first to minimize the number of uncovered instances. The second criterion is to minimize the time of the schedule. Both is achieved simultaneously by minimizing the term $(C+1)\sum_i y_i + \sum_{S,t} t x_{S,t}$. The latter summand obviously minimizes the total time scheduled. The first summand minimizes the number of uncovered instances. Note that the factor $C + 1$ ensures that the objective will always be less for schedules that solve at least one more instance, even when the scheduled time would increase from 0 seconds to the maximum of C seconds.

For further details on 3S the reader is referred to [7] which also contains a in-depth comparison to CP-Hydra [8].

2.2 Reducing the Number of Integer Variables

3S has 26 constituent SAT solvers, 11 of which are considered with two parameterizations each. Consequently, the scheduler considers 37 solvers. Moreover, unlike prior SAT portfolios, the 3S portfolio is identical for all categories in the SAT competition (application, crafted, and random). Consequently, it is based on a vast set of training instances, almost 5,500. Therefore, the IP above has more than 200,000 variables and more than 5,000 constraints. Although solved offline, to solve this IP more quickly, 3S uses a heuristic in [7]. 3S first solves the continuous relaxation of the solver scheduling IP. That is, it considers the linear program (LP) where constraints $y_i, x_{S,t} \in \{0,1\}$ are replaced by $0 \leq y_i, x_{S,t} \leq 1$.

When solving this relaxed LP the simplex algorithm [4] will only consider variables where, at some point during the optimization, it is beneficial to introduce these variables. In practice, during the optimization the vast majority of the 200,000 variables will never be set to a value greater than zero. Without going into the theory of linear programming, the important aspect is that the simplex algorithm has a *precise necessary condition* to determine whether a variable can improve the objective or not. Namely, in each step of the optimization, the simplex algorithm prices each constraint with a so-called "dual value."

For each variable, it then computes a "reduced cost." The latter is defined as the actual cost factor in the objective of the variable, minus the sum of the variable's coefficients in each constraint times the dual price of that constraint. Formally, when c_j is the cost coefficient, A_{ij} is the matrix coefficient for variable z_j on constraint i, and when π_i is the dual price for constraint i, then the reduced costs $\bar{c}_j$ for variable z_j are defined as:

$$\bar{c}_j = c_j - \sum_i A_{ij}\pi_i.$$

Now, the simplex algorithm will only consider setting variable z_j to a value different from 0 when $\bar{c}_j < 0$. We then say, that the variable (or the respective column in the matrix) has *negative reduced costs.*

So 3S solves the relaxation LP by introducing one (potentially new) variable in each iteration. Then, to solve the actual integer problem, 3S removes all variables from the solver scheduling IP which have never been introduced during the optimization. This reduction in the number of integer variables speeds up the solution to the integer program. However, Kadioglu et al. [7] showed that the solutions found in this manner are near-optimal in practice and, on average, work just as well on the test set as the optimal schedule would.

3 Parallel Solver Portfolios

The objective of this work is to generalize the 3S technology for the development of parallel SAT solver portfolios. At the core of 3S lie two optimization problems. The first is the selection of the long running solver primarily based on the maximum number of instances solved. The second is the solver scheduling problem.

Consider the first problem when there are $p > 1$ processors available. The objective is to select p solvers that, as a set, will solve the most number of instances. Note that this problem can no longer be solved by simply choosing the one solver that solves most instances in time. Moreover, we will now need to decide how to integrate the newly chosen solvers with the ones from the static schedule. The second problem is the solver scheduling problem discussed before, with the additional problem that solvers need to be assigned to processors so that the total makespan is within the allowed time limit.

A major obstacle in solving these problems efficiently is the symmetry induced by the identical processors to which we can assign each solver. Symmetries can hinder optimization very dramatically as equivalent (partial) schedules (which can be transformed into one another by permuting processor indices) will be considered again and again by a systematic solver. For example, when there are 8 processors, for each schedule there exist over 40,000 (8 factorial) equivalent versions. An optimization that used to take about half a second may now easily take 6 hours.

Another consideration is the fact that a parallel solver portfolio may obviously include parallel solvers as well. Assuming there are 8 processors and a parallel solver employs 4 of them, there are 70 different ways to allocate processors for this solver. The portfolio that we will develop later will have 37 sequential and 2 4-core parallel solvers. The solver scheduling IP that needs to be solved for this case has over 1.5 million variables. Eventually, we will apply our technology to a set of 72 of the latest SAT solvers from 2011, among them four parallel solvers which we will consider to run with 1, 2, 3, and 4 processors. Note that, in the parallel case, we will need to solve these IPs *at runtime*. Consequently, where 3S could afford to price out all variables at each step, we will need a more sophisticated method to speed up the optimization time – which directly competes with the remaining time to solve the actual SAT problem that was given.

3.1 Parallel Solver Scheduling

Recall again that we need to solve two different optimization problems. The first is to compute a static schedule for the first 10% of the allowed runtime. This problem is solved once, offline. The second optimization problem schedules solvers for the remaining 90% of the allowed time. This is done instance-specifically, taking into account the specific features of the SAT instance that is given.

We will address both optimization problems by considering the following IP. Let $t_S \geq 0$ denote the minimum time that solver S must run in the schedule, let $M = \{S ; | ; t_S > 0\}$ denote the set of solvers that have a minimal runtime, let p be the number of processors, and let $n_S \leq p$ denote the number of processors that solver S requires.

Parallel Solver Scheduling IP - CPU time

$$\begin{aligned}
\min\ & (pC+1)\sum_i y_i + \sum_{S,t,P} t n_S x_{S,t,P} \\
s.t.\ & y_i + \sum_{(S,t)\ \mid\ i\in V_{S,t}, P\subseteq\{1,\ldots,p\}, |P|=n_S} x_{S,t,P} \geq 1 \qquad \forall i \\
& \sum_{S,t,P\subseteq\{1,\ldots,p\}\cup\{q\}, |P|=n_S} t x_{S,t,P} \leq C \qquad \forall q\in\{1,\ldots,p\} \\
& \sum_{S,t,P\subseteq\{1,\ldots,p\}, |P|=n_S, t\geq t_S} x_{S,t,P} \geq 1 \qquad \forall S\in M \\
& \sum_{S,t,P\subseteq\{1,\ldots,p\}, |P|=n_S} x_{S,t,P} \leq N \\
& y_i, x_{S,t,P} \in \{0,1\} \qquad \forall i, S, t, P\subseteq\{1,\ldots,p\}, |P|=n_S
\end{aligned}$$

Variables y_i are exactly what they were before. There are now variables $x_{S,t,P}$ for all solvers S, time limits t, *and subsets of processors* $P \subseteq \{1,\ldots,p\}$ with $|P| = n_S$. $x_{S,t,P}$ is 1 if an only if solver S is run for time t on the processors in P in the schedule.

The first constraint is again to solve all instances with the schedule or count them as not covered. There is now a time limit constraint *for each processor*. The third set of constraints ensures that all solvers that have a minimal solver time are included in the schedule, with an appropriate time limit. The last constraint finally places a limit on the number of solvers that can be included in the schedule.

The objective is again to minimize the number of uncovered instances. The secondary criterion is to minimize the total CPU time of the schedule.

Remark 1. Note that the IP above needs to be solved both offline to determine the static solver schedule (for this problem $M = \emptyset$ and the solver limit is infinite) and *during the execution phase* (when M and the solver limit are determined by the static schedule computed offline). Therefore, we absolutely need to be able to solve this problem quickly, despite its huge size and its inherent symmetry caused by the multiple processors.

Note also that the parallel solver scheduling IP does not directly result in an executable solver schedule. Namely, the IP does not specify the actual start times of solvers. In the sequential case this does not matter as solvers can be sequenced in any way without affecting the total schedule time or the number of instances solved. In the parallel case, however, we need to ensure that the parallel processes are in fact run in parallel. We omit this aspect from the IP above to avoid further complicating the optimization. Instead, after solving the parallel solver IP, we heuristically schedule the solvers in a best effort approach, whereby we may preempt solvers and eventually even lower the runtime of the solvers to obtain a legal schedule. In our experiments presented later it turned out that in practice the latter was never necessary. Hence, the quality of the schedule was never diminished by the necessity to schedule processes that belong to the same parallel solver at the same time.

3.2 Solving the Parallel Solver Scheduling IP

We cannot afford to solve the parallel solver scheduling IP exactly during the execution phase. Each second spent on solving this problem is one second less for solving the actual SAT instance. Hence, we revert to solving the problem heuristically by employing *column generation*, whereby the generation of IP variables is limited to the root node.

While 3S prices all columns in the IP during each iteration, fortunately we actually do not need to do this here. Consider the reduced costs of a variable. Denote with $\mu_i \leq 0$ the dual prices for the instance-cover constraints, $\pi_q \leq 0$ the dual prices for the processor time limits, $\nu_S \geq 0$ the dual prices for the minimum time solver constraints, and $\sigma \leq 0$ the dual price for the limit on the number of solvers. Finally, let $\bar{\nu}_S = \nu_S$ when $S \in M$ and 0 otherwise. Then:

$$\bar{c}_{S,t,P} = n_S t - \sum_{i \in V_{S,t}} \mu_i - \sum_{q \in P} t\pi_q - \bar{\nu}_S - \sigma.$$

The are two important things to note here: First, the fact that we only consider variables introduced during the column generation process means that we *reduce the processor symmetry* in the final IP. While it is not impossible, it is unlikely that the variables that would form a symmetric solution to a schedule that can already be formed from the variables already introduced would have negative reduced costs.

Second, to find a new variable that has the most negative reduced costs, we do not need to iterate through all $P \subseteq \{1, \ldots, p\}$ for all solver/time pairs (S, t). Instead, we order the processors by their decreasing dual prices. The next variable introduced will use the first n_S processors in this order as all other selections of processors would result in higher reduced costs.

3.3 Minimizing Makespan and Post Processing the Schedule

We now have everything in place to develop our parallel SAT solver portfolio. In the offline training phase we compute a static solver schedule based on all training instances for 10% of the available time. We use this schedule to determine a set M of solvers that must be run for at least the static scheduler time at runtime. During the execution phase, given a new SAT instance we compute its features, determine the k closest training instances, and compute a parallel schedule that will solve as many of these k instances in the shortest amount of CPU time possible.

In our experiments we consider a second variant of the parallel solver scheduling IP where the secondary criterion is not to minimize CPU time but the *makespan* of the schedule. The corresponding IP is given below, where variable

m measure the minimum idle time for all processors. The reduced cost computation changes accordingly.

Parallel Solver Scheduling IP - Makespan

$$
\begin{aligned}
&\min\ (C+1)\sum_i y_i - m \\
&s.t.\ y_i + \sum_{(S,t)\ \mid\ i \in V_{S,t}, P \subseteq \{1,\ldots,p\}, |P| = n_S} x_{S,t,P} \geq 1 \qquad \forall i \\
&\quad m + \sum_{S,t,P \subseteq \{1,\ldots,p\} \cup \{q\}, |P| = n_S} t x_{S,t,P} \leq C \qquad \forall q \in \{1,\ldots,p\} \\
&\quad \sum_{S,t,P \subseteq \{1,\ldots,p\}, |P| = n_S, t \geq t_S} x_{S,t,P} \geq 1 \qquad \forall S \in M \\
&\quad \sum_{S,t,P \subseteq \{1,\ldots,p\}, |P| = n_S} x_{S,t,P} \leq N \\
&\quad y_i, x_{S,t,P} \in \{0,1\} \qquad \forall i, S, t, P \subseteq \{1,\ldots,p\}, |P| = n_S
\end{aligned}
$$

Whether we minimize CPU time or makespan, as remarked earlier, we post-process the result by assigning actual start times to solvers heuristically. We also scale the resulting solver times to use as much of the available time as possible. For low values of k, we often compute schedules that solve all k instances in a short amount of time without utilizing all available processors. In this case, we assign new solvers to the unused processors in the order of their ability to solve the highest number of the k neighboring instances.

4 Experimental Results

Using the methodology above, we built two parallel portfolios. The first based on the 37 constituent solvers of 3S [7]. We refer to this portfolio as p3S-37. The second portfolio that we built includes two additional solvers, 'Cryptominisat (2.9.0)' [11] and 'Plingeling (276)' [2], both executed on four cores. We refer to this portfolio as p3S-39. It is important to emphasize that *all solvers that are part of our portfolio were available before the SAT Competition 2011.* We would have liked to compare our portfolio builder with other parallel portfolios. However, existing works on parallel portfolios do not accommodate parallel solvers. Consequenlty, in our experiments we will compare p3S-37 and p3S-39 with the state of the art in parallel SAT solving. The winners in the parallel tracks at the 2011 SAT Competition were the parallel solver portfolio 'ppfolio' [10] and 'Plingeling (587f)' [3], both executed on eight cores. Note that these competing solvers are new solvers that were introduced for the SAT Competition 2011.

As our benchmark set of SAT instances, to the $5,464$ instances from all SAT Competitions and Races between 2002 and 2010 [1], we added the $1,200$ (300

Table 1. Average performance comparison parallel portfolio when optimizing CPU time and varying neighborhood size k based on 10-fold cross validation.

CPU Time	10	25	50	100	200
Average (σ)	320 (45)	322 (43.7)	329 (42.2)	338 (43.9)	344 (49.9)
Par 10 (σ)	776 (241)	680 (212)	694 (150)	697 (156)	711 (221)
# Solved (σ)	634 (2.62)	636 (2.22)	636 (1.35)	637 (1.84)	636 (2.37)
% Solved (σ)	99.0 (0.47)	99.2 (0.39)	99.2 (0.27)	99.2 (0.28)	99.2 (0.41)

Table 2. Average performance comparison parallel portfolio when optimizing Makespan and varying neighborhood size k based on 10-fold cross validation.

Makespan	10	25	50	100	200
Average (σ)	376.1 (40.8)	369.2 (42.9)	374 (40.7)	371 (40.8)	366 (36.9)
Par 10 (σ)	917 (200)	777 (192)	782 (221)	750 (153)	661 (164)
# Solved (σ)	633 (2.16)	635 (2.28)	634.9 (2.92)	635 (1.89)	637 (2.01)
% Solved (σ)	98.8 (0.39)	99.1 (0.39)	99.1 (0.46)	99.2 (0.32)	99.3 (0.34)

application, 300 crafted, 600 random) instances from last years SAT Competition 2011. Based on this large set of SAT instances, we created a number of benchmarks. Based on all SAT instances that can be solved by at least one of the solvers considered in p3S-39 within 5,000 seconds, we created an equal 10 partition. We use this partition to conduct a ten-fold cross validation, whereby in each fold we use nine partitions as our training set (for building the respective p3S-37 and p3S-39 portfolios), and evaluate the performance on the partition that was left out before. For this benchmark we report average performance over all ten splits. On top of this cross-validation benchmark, we also consider the split induced by the SAT Competition 2011. Here we use all instances prior to the competition as training set, and the SAT Competition instances as test set. Lastly, we also created a competition split based on application instances only.

As performance measures we consider the number of instances solved, average runtime, and PAR10 score. The PAR10 is a penalized average runtime where instances that time out are penalized with 10 times the timeout. Experiments were run on dual Intel Xeon 5540 (2.53 GHz) quad-core Nehalem processors with 24 GB of DDR-3 memory.

Impact of the IP Formulation and Neighborhood Size In Tables 1 and 2 we show the average cross-validation performance of p3S-39 when using different neighborhood sizes k and the two different IP formulations (tie breaking by minimum CPU time or minimizing schedule makespan). As we can see, the size of the neighborhood k affects the most important performance measure, the number of instances solved, only very little. There is a slight trend towards larger k's working a little bit better. Moreover, there is also not a great difference between the two IP formulations, but on average we find that the version that breaks ties by minimizing the makespan solves about 1 instance more per split. Based on

Table 3. Performance of 10-fold cross validation on all data. Results are averages over the 10 folds.

Cross	p3S-37		p3S-39	
Validation	4 core	8 core	4 core	8 core
Average (*sigma*)	420 (22.1)	**355 (31.3)**	435 (48.5)	366 (36.9)
Par 10 (*sigma*)	991 (306)	679 (176)	1116 (256)	**661 (164)**
Solved (*sigma*)	630 (4.12)	633 (2.38)	631 (2.75)	**637 (2.01)**
% Solved (*sigma*)	98.3 (0.63)	98.8 (0.35)	98.5 (0.49)	**99.3 (0.34)**

these results, p3S in the future refers to the portfolio learned on the respective training benchmark using $k = 200$ and the IP formulation that minimizes the makespan.

4.1 Impact of Parallel Solvers and the Number of Processors

Next we investigate the impact of employing parallel solvers in the portfolio. In Tables 3 and 4 we compare the performance of p3S-37 (without parallel solvers) and p3S-39 (which employs two 4-core parallel solvers) on the cross-validation and on the competition split. We observe a small difference in the number of solved instances in the cross-validation, and a significant gap in the competition split.

Two issues are noteworthy about that competition split. First, since this was the latest competition and these instances were also used for the parallel track, the instances in the test set of this split are significantly harder than the instances from earlier years. The relatively low percentage of instances solved even by the best solvers at the SAT Competition 2011 is an indication for this. Second, some instance families in this test set are completely missing in the training partition. That is, for a good number of instances in the test set there may be no training instance that is very similar. These features of any competition-induced split (which is the realistic split scenario!) explain why the average cross-validation performance is often significantly better than the competition performance. Moreover, they explain why p3S-39 has a significant advantage over p3S-37: When a lot of the instances are out of reach of the sequential solvers within the competition timeout *then the portfolio must necessarily include parallel solvers to perform well.*

As a side remark: the presence of parallel solvers is what makes the computation of parallel portfolios challenging in the first place. Not only do parallel solvers complicate the optimization problems that we considered earlier. In the extreme case, if all solvers were sequential, we could otherwise have as many processors as solvers, and then a trivial portfolio would achieve the performance of the virtual best solver. That is to say: The more processors we have, the easier solver selection becomes. We were curious to see what would happen when we made the selection harder than it actually is under the competition setting and reduced the number of available processors to 4. For both p3S-37 and p3S-39, the cross-validation performance decreases only moderately while, under the

Table 4. Performance of the solvers on all 2011 SAT Competition data.

Competition	3S		//3S		VBS
	4 cores	8 cores	4 cores	8 cores	
Average	1907	1791	1787	1640	1317
Par 10	12,782	12,666	11,124	10,977	10,580
Solved	843	865	853	892	953
% Solved	70.3	72.1	71.1	74.3	79.4

competition split, performance decays significantly. At the same time, the advantages of p3S-39 over p3S-37 shrink a lot. As one would expect, the advantage of employing parallel solvers decays with a shrinking number of processors.

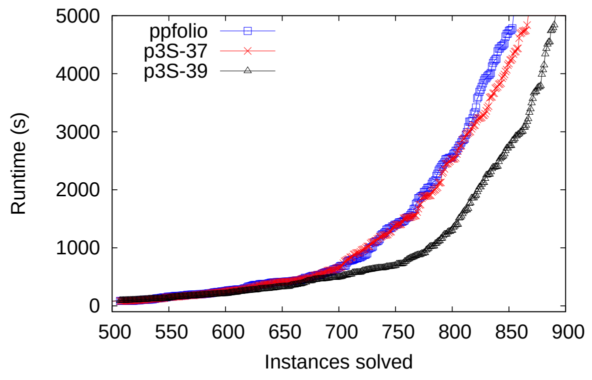
4.2 Parallel Solver Selection and Scheduling vs. State-of-the-Art

The dominating parallel portfolio to date is 'ppfolio' [10]. In the parallel track at the SAT Competition 2011, it won gold in the crafted and random categories and came in just shy to winning the application category as well where it was beat by just one instance. In the application category, the winning solver was 'Plingeling (587f)' run on 8 cores. We compare against both competing approaches in Figures 1 and 2.

Instances From All SAT Categories The top plot in Figure 1 shows the scaling behavior in the form of a "cactus plot" for 8-core runs of ppfolio, p3S-37, and p3S-39, for the competition split containing all 1,200 instances used in the 2011 SAT Competition. This plot shows that p3S-39 (whose curve stays the lowest as we move to the right) can solve significantly more instances than the other two approaches for any given time limit larger than around 800 sec. We also see that p3S-37, based solely on sequential constituent solvers, performs similar to ppfolio for time limits up to 3,000 sec, and begins to outperform it for larger time limits.

This comparative performance profile is by no means accidental. It is well known in SAT that an instance that is exceedingly difficult to solve for one solver poses almost no problem at all for another. This is the deeper reason why the 10% static schedules are well motivated, because there exists a realistic chance that one of the solvers scheduled for a short period of time will solve the instance.

At a higher level, we are observing the same here. Sequential SAT solvers have, for a good number of instances, the chance to solve an instance within some time. Consequently, a portfolio of sequential solvers only can, up to a point, compete with a solver portfolio that incorporates parallel solvers as well. However, a realistic set of hard instances, as the one considered at the SAT Competition, also contains instances that are very hard to solve, even by the best solver for that instance. Some of the instances will not be solvable by any sequential algorithm within the available time. This is why it is so important to be able to include parallel solvers in a parallel portfolio.



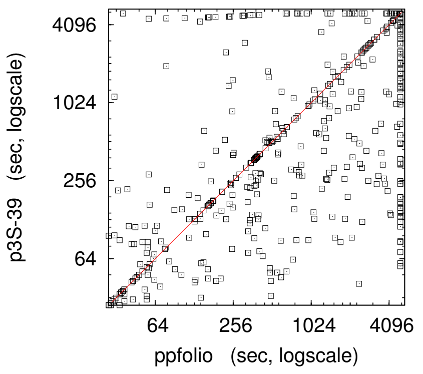
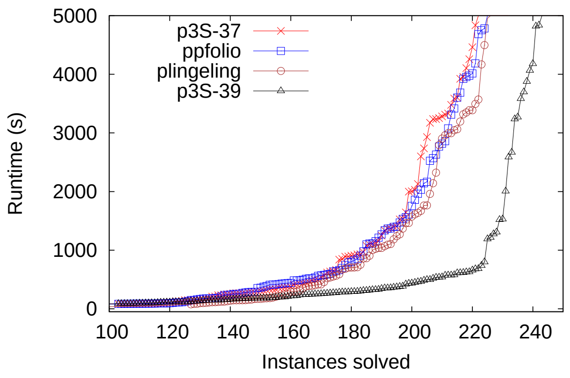


Fig. 1. Comparison on all 1200 instances used in the 2011 SAT Competition, across all categories. Left: cactus plot depicting the scaling behavior of solvers. Right: per-instance comparison between ppfolio and p3S-39.

The bottom plot in Figure 1 shows the per-instance performance of p3S-39 vs. ppfolio, with runtimes in log-scale on both axes. More points being below the diagonal red line signifies that p3S-39 is faster than ppfolio on a large majority of the instances. ppfolio also times out on many instances that p3S-39 can solve, as evidenced the large number of points on the right margin of the plot.

Overall, p3S-39 was able to solve 892 instances, 47 more than ppfolio. p3S-37 was somewhere in-between, solving 20 more than ppfolio. In fact, even with only 4 cores, p3S-37 and p3S-39 solved 846 and 850 instances, respectively, more than the 845 ppfolio solved on 8 cores.

Industrial Instances Traditionally, portfolios did not excel in the category for industrial SAT instances. In part, this is because there are much fewer representative training instances available than in the random or crafted categories. That is to say, at the competition there is a much better chance to encounter an application instance that is very much different from anything that was considered during training. Moreover, progress on solvers that work well on industrial instances is commonly much more pronounced. Since competition portfolios are



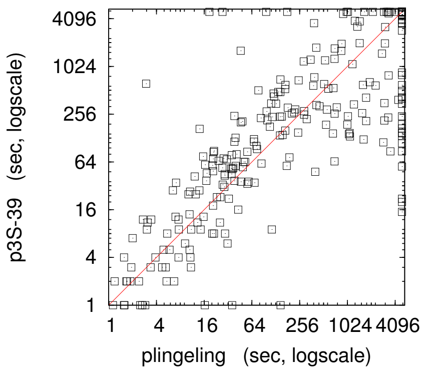
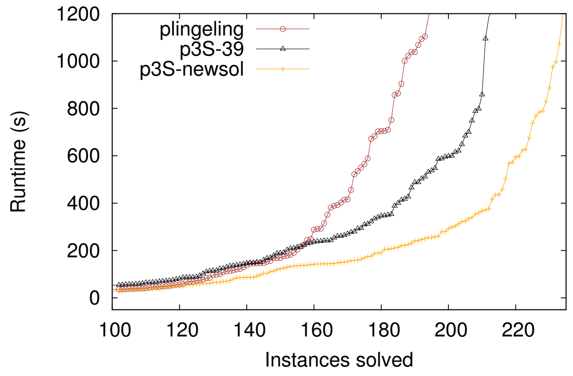


Fig. 2. Comparison on the 300 application category instances used in the 2011 SAT Competition. Left: cactus plot depicting the scaling behavior of solvers. Right: per-instance comparison between Plingeling and p3S-39.

based on older solvers, even their intelligent selection may not be enough to make up for the progress on the solvers themselves.

Figure 2 shows similar comparisons, but on the competition split restricted to the application category, and with Plingeling as one of the competing solvers. The cactus plot on top still shows a significantly better scaling behavior of p3S-39 than both Plingeling and ppfolio. The scatter plot shows that Plingeling, not surprisingly, is able to solve several easy instances within just a few seconds (as evidenced by the points on the bottom part of the left edge of the plot), but begins to take more time than p3S-39 on challenging instances and also times out on many more instances (shown as points on the right edge of the plot).

Overall, with 8 cores, p3S-39 solved 248 application category instances, 23 more than ppfolio and 22 more than Plingeling. Moreover, p3S-37, based only on sequential constituent solvers, was only two instances shy of matching Plingeling's performance. This performance improvement is quiete significant. In the application category, the best performing algorithms usually lie just a couple of instances solved apart. In 2011, the top ten solvers solved between 200 and 215



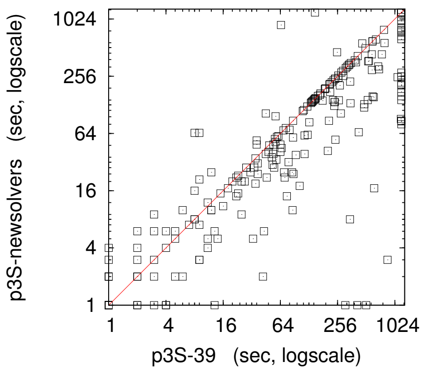


Fig. 3. Performance of p3S built using latest solvers, on all the 300 application category instances used in the 2011 SAT Competition. Left: cactus plot depicting the scaling behavior of solvers. Right: per-instance comparison between p3S-39 and p3S-newsolvers.

instances. An improvement of over 20 instances over the best-performing solver from nine months ago is much more than we had expected.

2012 Competition Portfolio Finally, to demonstrate the efficacy of the method presented here, we trained a parallel portfolio based on 40 of the latest available parallel and sequential SAT solvers. Two of them were run on 1, 2, 3, and 4 processors. For all solvers, we consider a secondary setting where the given instance is first simplified by the Satelite program. In total, we have thus 92 solvers, 6 of them run in parallel on 2, 3, or 4 processors.

In Figure 3 we compare the performance of this portfolio against Plingeling, the winning parallel solver in the 2011 SAT Competition nine months ago, and p3S-39, our portfolio of solvers from 2010 and before. We observe that our method of devising parallel portfolios continues to result in strong performance and generalizes well to this extended set of solvers and corresponding training data. The parallel portfolio based on the latest SAT solvers currently competes in the 2012 SAT Challenge.

5 Conclusion

We presented the first method for devising *drynamic parallel solver portfolios that accommodate parallel solvers*. Our approach is based on the recently introduced SAT Solver Selector and Scheduler (3S). We combine core methods from machine learning, such as nearest neighbor regression, with methods from optimization, in particular integer programming and column generation, to produce parallel solver schedules *at runtime*. We compared different formulations of the underlying optimization problems and found that minimizing makespan as a tie breaking rule works slightly better than minimizing CPU time.

We compared the resulting portfolio, p3S-39, with the current state-of-the-art parallel solvers on instances from all SAT categories and from the application category only. We found that p3S-39 marks a very significant improvement in our ability to solve SAT instances.

References

[1] SAT Competition: http://www.satcomptition.org.

[2] A. Biere. Lingeling, plingeling, picosat and precosat at sat race 2010. Technical report, Johannes Kepler University, Linz, Austria, 2010.

[3] A. Biere. Lingeling and friends at the sat competition 2011. Technical report, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, 2011.

[4] G. Dantzig. *Linear programming and extensions.* Princeton University Press, Princeton, N.J.,, 1963.

[5] N. Een and N. Sorensson. An extensible sat-solver [ver 1.2], 2003.

[6] Y. Hamadi, S. Jabbour, and S. Lakhdar. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2009.

[7] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann. Algorithm selection and scheduling. *CP*, 2011.

[8] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. *Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[9] M. Petrik and S. Zilberstein. Learning static parallel portfolios of algorithms. *Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

[10] O. Roussel. Description of ppfolio, 2011. http://www.cril.univ-artois.fr/ roussel/ppfolio/solver1.pdf.

[11] M. Soos. Cryptominisat 2.9.0, 2011.

[12] D. Stern, H. Samulowitz, R. Herbrich, T. Graepel, L. Pulina, and A. Tacchella. Collaborative expert portfolio management. *AAAI*, 2010.

[13] M. Streeter and S. Smith. Using decision procedures efficiently for optimization. *ICAPS*, pp. 312–319, 2007.

[14] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *JAIR*, 32(1):565–606, 2008.

[15] X. Yun and S. Epstein. Learning algorithm portfolios for parallel execution. *Workshop on Learning and Intelligent Optimization*, 2012.