Partitioning Search Spaces of a Randomized Search

Antti E. J. Hyvärinen, Tommi Junttila, and Ilkka Niemelä

Helsinki University of Technology TKK
Department of Information and Computer Science
{Antti.Hyvarinen,Tommi.Junttila,Ilkka.Niemela}@tkk.fi

Abstract. This paper studies the following question: given an instance of the propositional satisfiability problem, a randomized satisfiability solver, and a cluster of n computers, what is the best way to use the computers to solve the instance? Two approaches, simple distribution and search space partitioning as well as their combinations are investigated both analytically and empirically. It is shown that the results depend heavily on the type of the problem (unsatisfiable, satisfiable with few solutions, and satisfiable with many solutions) as well as on how good the search space partitioning function is. In addition, the behavior of a real search space partitioning function is evaluated in the same framework. The results suggest that in practice one should combine the simple distribution and search space partitioning approaches.

1 Introduction

In this paper we develop distributed techniques for solving challenging instances of the propositional satisfiability problem (SAT). We are interested in using the best available SAT solvers as black-box subroutines or with little modification and in this way take advantage of the rapid development of SAT solver technology.

One of the interesting features in current state-of-the-art SAT solvers is that they use randomization and that their run times can vary significantly for a given instance. This opens up new opportunities for developing distributed solving techniques. The most straightforward idea is to employ a *simple distribution* approach where one just performs a number of independent runs using a randomized solver. This leads to surprising good speed-ups even when used in a grid environment with substantial communication and other delays [1]. The approach could be extended by applying particular restart strategies [2,3] or using an algorithm portfolio scheme [4,5]. Another key feature in modern SAT solvers is the use of conflict driven clause learning techniques. This feature can be exploited in the simple distribution approach and it has been shown that combining parallel learning schemes with a simple restart strategy leads to a powerful distributed SAT solving technique [6].

Another approach to developing parallel SAT solving techniques is based on *partitioning* the search space to multiple parts which can be handled in parallel. This can be achieved by constraint-based partitioning where the search space for a SAT instance $\mathcal{F}$ is split to n *derived* instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ by including additional constraints to $\mathcal{F}$. Typical implementation techniques include guiding paths [7,8,9] and scattering [10].

Both simple distribution and partitioning have their strengths. The former has led to surprisingly good performance but for really challenging SAT instances it provides no

R. Serra and R. Cucchiara (Eds.): AI*IA 2009, LNAI 5883, pp. 243–252, 2009.

mechanism for splitting the search to more manageable portions to be treated in parallel. Search space partitioning techniques offer an approach to achieving this. However, the interaction between partitioning and randomized SAT solvers is poorly understood and the paper aims to shed new light on this problem. It studies in detail combination of constraint-based partitioning and randomized SAT solvers, and provides an analysis on how an efficient and robust implementation can be achieved.

The rest of the paper is structured as follows. Section 2 reviews briefly relevant key characteristics of modern randomized SAT solvers and the simple distribution approach. Section 3 studies analytically the expected run time of a plain partitioning approach where a SAT instance is partitioned and then a randomized SAT solver is used to solve the resulting instances. The section provides fundamental results for two limiting cases, for ideal and void partitioning functions. Section 4 extends the study to a setting where simple distribution and partitioning are mixed. Section 5 provides an implementation of a randomized partitioning function and Section 6 verifies the results briefly using experiments, and conclusions are given in Section 7. An extended version of this work with more experiments and proofs for the propositions is available through the first author's home pages (`http://www.tcs.hut.fi/~aehyvari/`).

2 Randomization and Simple Distribution

Most modern SAT solvers apply search *restarts* and some form of *randomization* to avoid getting stuck at hard subproblems [11]. For instance, MiniSat [12] version 1.14 restarts the search periodically and makes two percent of its branching decisions pseudo-randomly. Despite restarts and randomness, the run times of a SAT solver on an instance $\mathcal{F}$ can vary significantly between some minimum $t_{\min}$ and maximum $t_{\max}$ (we assume that $t_{\min} > 0$ and $t_{\max}$ is finite). Thus, we treat the run time of the solver on the instance as a random variable T and study the associated cumulative run-time distribution $q_T(t) = \Pr(T \leq t)$ (i.e. $q_T(t)$ is the probability that the instance is solved within t seconds) and its expected value $\mathbb{E}(T) = \int_{t_{\min}}^{t_{\max}} t q'(t) \mathrm{d}t$. As an example, observe the run-time distribution $q(t)$ (approximated by one hundred sample runs) of an instance given in the left hand side plot of Fig. 1. Depending on the seed given to the pseudo-random number generator of MiniSat v1.14, the run time varies from less than a second to thousands of seconds.

This non-constant run time phenomenon can be exploited in a parallel environment by simply running n SAT solvers on the same instance $\mathcal{F}$ in parallel and terminating the search when one of the solvers reports the solution. We call this approach *Simple Distributed SAT solving* (SDSAT) and denote its run time by the random variable T^n_{sdsat}. The cumulative run time distribution is now improved from $q_T(t)$ of the sequential case to $q_{T^n_{\text{sdsat}}}(t) = 1 - (1 - q_T(t))^n$. This approach can be surprisingly efficient. As an example, for the instance in the left hand side plot of Fig. 1 the expected run-time in the sequential case is $\mathbb{E}(T) \approx 623$s while for eight solvers $\mathbb{E}(T^8_{\text{sdsat}}) \approx 31$s (that is, around 20 times less). For a more detailed analysis of running SDSAT in a parallel, distributed environment involving communication and other delays, see [1].

Although the SDSAT approach can reduce the expected time to solve an instance, it cannot reduce it below the minimum run time $t_{\min}$. For an example, observe the sequential run time distribution $q(t)$ of another instance given in the right hand side plot

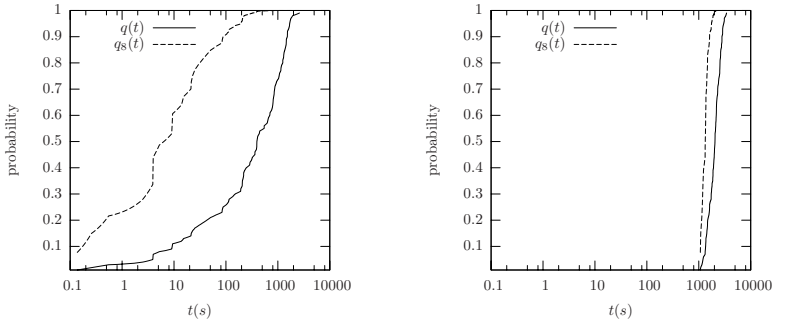

Fig. 1. The run time distributions of two instances for single (*the* $q(t)$ *plots*) and eight (*the* $q_8(t)$ *plots*) randomized SAT solvers

of Fig. 1; the variation of the run time is significantly smaller and the instance seems to have no short run times. Consequently, running eight SAT solvers in parallel does not reduce the expected run time significantly; in numbers, $\mathbb{E}(T) \approx 2,065$s while for eight solvers $\mathbb{E}(T^8_{\text{sdsat}}) \approx 1,334$s (i.e., only less than two times faster). Even more importantly, the *minimum run time stays the same irrespective of how many parallel solvers are employed*. As a summary, we can establish the following properties for the expected run time of the SDSAT approach:

Proposition 1. $t_{\min} \leq \mathbb{E}(T^n_{\text{sdsat}}) \leq \mathbb{E}(T)$ *for each* $n \geq 1$. *Furthermore,* $\mathbb{E}(T^n_{\text{sdsat}}) \to t_{\min}$ *when* $n \to \infty$.

As we have seen, SDSAT can allow super-linear speedup (meaning $\mathbb{E}(T^n_{\text{sdsat}}) < \mathbb{E}(T)/n$) for instances with strongly varying run time. However, as the maximum speedup obtainable with SDSAT is $\mathbb{E}(T)/t_{\min}$, this can only happen for "smallish" values of n and for more than $\mathbb{E}(T)/t_{\min}$ solvers the speedup is guaranteed to be sub-linear.

3 Partitioning

The basic idea in the form of partitioning we use in this paper is quite simple: given a SAT instance $\mathcal{F}$ and a positive integer n, use a *partitioning function* to compute a set $\mathcal{F}_1, \ldots, \mathcal{F}_n$ of *derived* SAT instances such that

$$\mathcal{F} \equiv \mathcal{F}_1 \vee \cdots \vee \mathcal{F}_n. \tag{1}$$

Now, in order to find whether $\mathcal{F}$ is satisfiable, we solve, in parallel, all $\mathcal{F}_1, \ldots, \mathcal{F}_n$ and deduce that $\mathcal{F}$ is satisfiable if at least one of $\mathcal{F}_1, \ldots, \mathcal{F}_n$ is. This method is called the *plain partitioning approach* in order to distinguish it from the composite approaches in Sect. 4. One way to implement partitioning functions is described in [10] (also see Sect. 5), where each $\mathcal{F}_i$ is obtained from $\mathcal{F}$ by conjoining it with a set of additional partitioning constraints.[1] In addition to the requirement (1), partitioning functions often ensure that the models of $\mathcal{F}_1, \ldots, \mathcal{F}_n$ are mutually disjoint.

[1] As explained in [13], guiding paths [7,8] can also be interpreted as partitioning constraints.

Intuitively, the ideal case is that the partitioning function can partition the instance $\mathcal{F}$ into n new instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ so that each new instance $\mathcal{F}_i$ is n times easier to solve than the original. That is, if the original instance $\mathcal{F}$ has the cumulative run time distribution $q_T(t)$, then the distribution of each $\mathcal{F}_i$ is $q_{T_i}(t) = q_T(nt)$. In this case we say that the partition function is *ideal* for the instance. As obtaining ideal partitioning functions can be difficult, we also consider the case of a *void* partitioning function where the partitioning fails totally, resulting in new instances which are as hard to solve as the original, i.e. have the same distribution $q_{T_i}(t) = q_T(t)$. This is a realistic scenario because modern, clause-learning SAT solvers, such as MiniSat, use sophisticated heuristics in the search: it is possible that values of certain variables are practically never considered. If the partition function constrains only these irrelevant variables, the difficulty of the instance does not decrease, and thus such a function is void.

In this section, we give an analytic study of the efficiency of the plain partitioning approach, under both ideal and void functions, when the fact that the SAT solver is randomized is taken into account. As the efficiency depends heavily on the satisfiability of the instance, we consider three cases: unsatisfiable instance, a satisfiable instance with many solutions, and a satisfiable instance with a unique solution. We have also simulated the plain partitioning approach on run time distributions of some real SAT instances; some results are given later in Sect. 4 after some composite approaches mixing simple distribution and plain partitioning have been described. A real partitioning function is considered in Sect. 5.

3.1 Unsatisfiable Instances

Assume that an unsatisfiable instance $\mathcal{F}$ is partitioned into n new instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ fulfilling Eq. (1). All new instances need to be shown unsatisfiable to deduce that $\mathcal{F}$ is unsatisfiable. When performed in parallel, this corresponds to waiting for the termination of the "unluckiest" run.

In the case of ideal partitioning function, each new instance $\mathcal{F}_i$ is n times easier to solve than the original $\mathcal{F}$, having run time distributions $q_{T_i}(t) = q_T(nt)$. We denote the random variable capturing the run time of the resulting plain partitioning approach under an ideal partitioning function by $T^n_{\text{part(ideal)}}$. As all the new instances have to be solved (in parallel), the corresponding run time distribution is $q_{T^n_{\text{part(ideal)}}}(t) = q(nt)^n$. Based on this, we have the following interesting results. First, ideal partitioning functions can provide *at most linear expected speed-up* on unsatisfiable instances:

Proposition 2. $\mathbb{E}(T^n_{\text{part(ideal)}}) \geq \mathbb{E}(T)/n$ *for each* $n \geq 1$.

In fact, it can be shown that linear speed-up can only be obtained on instances that have a constant run time distribution, i.e. when $t_{\min} = t_{\max}$. However, the expected run time is never worse than that of solving the original instance with one solver:

Proposition 3. $\mathbb{E}(T^n_{\text{part(ideal)}}) \leq \mathbb{E}(T)$ *for each* $n \geq 1$.

When the number n of SAT solvers run in parallel is increased, the expected run time $\mathbb{E}(T^n_{\text{part(ideal)}})$ approaches $t_{\max}/n$, i.e., linear speed-up w.r.t. the *maximum* run time. Plain partitioning with ideal partitioning functions and simple distribution cannot be totally ordered; there are distributions for which $\mathbb{E}(T^n_{\text{sdsat}}) < \mathbb{E}(T^n_{\text{part(ideal)}})$ and others for which

$\mathbb{E}(T^n_{\text{part(ideal)}}) < \mathbb{E}(T^n_{\text{sdsat}})$ when n is smallish. However, as $\mathbb{E}(T^n_{\text{part(ideal)}}) \leq t_{\max}/n$ and $\mathbb{E}(T^n_{\text{sdsat}}) \geq t_{\min}$, we have that $\mathbb{E}(T^n_{\text{part(ideal)}}) < \mathbb{E}(T^n_{\text{sdsat}})$ for sufficiently large n.

Let us next consider the case of a void partitioning function, i.e. the case when the partitioning fails so that the run time distribution $q_{T_i}(t)$ of each new instance $\mathcal{F}_i$ is equal to $q_T(t)$ of the original instance $\mathcal{F}$. We denote by $T^n_{\text{part(void)}}$ the run time of the resulting plain partitioning approach. As all $\mathcal{F}_i$ have to be solved, the run time distribution of $T^n_{\text{part(void)}}$ is $q_{T^n_{\text{part(void)}}}(t) = q_{T_i}(t)^n = q_T(t)^n$. From this it follows that for unsatisfiable instances *it is not possible to obtain any speedup with void functions*:

Proposition 4. $\mathbb{E}(T^n_{\text{part(void)}}) \geq \mathbb{E}(T)$ *for each* $n \geq 1$.

In fact, *the more resources one uses, the closer to the maximum run time one gets*: $\mathbb{E}(T^n_{\text{part(void)}}) \to t_{\max}$ when $n \to \infty$.

3.2 Satisfiable Instances with Many Solutions

We next consider the case when a satisfiable SAT instance $\mathcal{F}$ is partitioned into n new instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ fulfilling Eq. (1). In order to deduce that $\mathcal{F}$ is satisfiable, it is enough to show that *any* of the new instances is satisfiable. In this section we assume that each new instance $\mathcal{F}_i$ is satisfiable, postponing the case where only one is satisfiable to the next section.

Let us consider the case of ideal partitioning function first. Again, we denote the random variable describing the run time of the resulting plain partitioning approach by $T^n_{\text{part(ideal)}}$. As the probability that none of the n solvers has solved the associated new instance within time t is $(1-q_T(nt))^n$, run time distribution of $T^n_{\text{part(ideal)}}$ is $q_{T^n_{\text{part(ideal)}}}(t) = 1-(1-q_T(nt))^n$. Several interesting properties follow from this. First, with n parallel solvers, *the expected run time is* n *times smaller than that of the Simple Distributed SAT*: $\mathbb{E}(T^n_{\text{part(ideal)}}) = \mathbb{E}(T^n_{\text{sdsat}})/n$. Therefore, when compared to solving the original instance, we notice that *on satisfiable instances with many solutions we may expect at least linear speed-up:*

Proposition 5. $\mathbb{E}(T^n_{\text{part(ideal)}}) \leq \mathbb{E}(T)/n$ *for each* $n \geq 1$.

When the number n of parallel SAT solvers is increased, $\mathbb{E}(T^n_{\text{part(ideal)}})$ approaches $t_{\min}/n$. Thus, one can in principle obtain almost linear speed-up w.r.t. *the minimum run time*.

In the case of a void partitioning function the run time of each new instance is the same as that of the original. As each new instance is assumed to be satisfiable, solving any of them is enough to deduce the satisfiability of the original instance. Therefore, *for satisfiable instances with many solutions, the plain partitioning approach with a void partitioning function effectively reduces to Simple Distributed SAT:*

Proposition 6. $\mathbb{E}(T^n_{\text{part(void)}}) = \mathbb{E}(T^n_{\text{sdsat}})$.

3.3 Satisfiable Instances with One Solution

When a satisfiable instance $\mathcal{F}$ with only one satisfying truth assignment is partitioned into n new instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$, it is likely that only one of the new instances is satisfiable while the others are unsatisfiable. Therefore, the satisfiable new instance has to

be solved to deduce that $\mathcal{F}$ is satisfiable. The problem is that it is not known which of the new instances this is.

In the case of ideal partitioning function, the run time of the satisfiable new instance is n times smaller than that of the original instance. Therefore, if all the n new instances are solved in parallel and the solving is terminated as soon as the satisfiable new instance is solved, *linear speed-up is obtained with an ideal partitioning function*:

Proposition 7. $\mathbb{E}(T^n_{\text{part(ideal)}}) = \mathbb{E}(T)/n$.

In the case of a void partitioning function, the run time of the satisfiable new instance is the same as that of the original instance. Thus, *using a void partitioning function results neither in speed-up nor in loss of efficiency*:

Proposition 8. $\mathbb{E}(T^n_{\text{part(void)}}) = \mathbb{E}(T)$.

4 Composite Approaches

The analysis of the previous section shows that the plain partitioning approach can potentially obtain even super-linear speed-ups, whereas an improper, void implementation can by Prop. 4 result in worse expected run time than that of one solver. The two approaches presented here aim at being at least as efficient as solving the instance with one solver. Assume that we have resources to run n SAT solvers in parallel and consider the following approaches that mix simple distribution and plain partitioning.

- *Repeated Partitioning*. In this approach, we run in parallel $k = \lfloor \sqrt{n} \rfloor$ copies of the plain partitioning approach, each copy splitting the instance $\mathcal{F}$ into k new instances $\mathcal{F}_{i,1}, \ldots, \mathcal{F}_{i,k}$ and solving each $\mathcal{F}_{i,j}$ once. We denote the random variable describing the run time of this approach by $T^n_{\text{rep-part}}$.
- *Safe Partitioning*. This approach reverses the order of SDSAT and partitioning compared to repeated partitioning: the instance $\mathcal{F}$ is partitioned into $k = \lfloor \sqrt{n} \rfloor$ new instances $\mathcal{F}_1, \ldots, \mathcal{F}_k$ and each new instances $\mathcal{F}_i$ is solved with k SAT solvers in parallel. The run time of this is denoted by the random variable $T^n_{\text{safe-part}}$.

Unfortunately, when using repeated partitioning on an unsatisfiable instance and the partitioning function is void, the experiments show that $\mathbb{E}(T) \leq \mathbb{E}(T^n_{\text{rep-part}})$ and that $\mathbb{E}(T^n_{\text{rep-part}}) \to t_{\max}$ when $n \to \infty$. However, safe partitioning (i) is as good as repeated partitioning (i.e. $\mathbb{E}(T^n_{\text{safe-part}}) = \mathbb{E}(T^n_{\text{rep-part}})$) on satisfiable instances and we conjecture that it is at least as good (i.e. $\mathbb{E}(T^n_{\text{safe-part}}) \leq \mathbb{E}(T^n_{\text{rep-part}})$) on unsatisfiable ones when the same partition function is applied; (ii) is equal to SDSAT on satisfiable instances with many solutions when the partition function is void; and (iii) seems experimentally at least as fast as solving the original instance with one solver, i.e., $\mathbb{E}(T^n_{\text{safe-part}}) \leq \mathbb{E}(T)$ even when the instance is unsatisfiable and the partitioning function is void.

To illustrate the approaches and results presented in Sects. 2, 3, and 4, Fig. 2 shows the expected run times of different approaches when applied to the same instances as in Fig. 1. As the left hand side instance is satisfiable with many solutions, the "sdsat+others" line depicts the behavior of the SDSAT approach as well as all the considered partitioning approaches when a void partitioning function is applied. The instance at the right hand side is unsatisfiable.

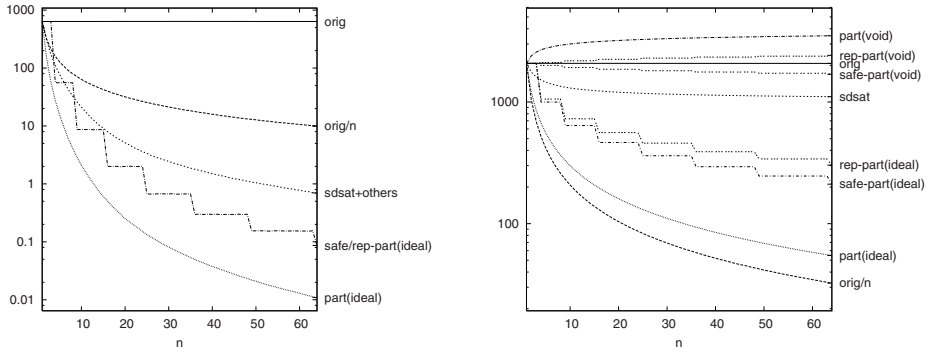

Fig. 2. Expected run times (in seconds) of different approaches on the instances in Fig. 1 when the number n of SAT solvers run in parallel is varied

5 Implementing a Randomized Partitioning Function

This section briefly describes how we implemented a partitioning function called *scattering* [10]. The implementation is based on MiniSat 1.14, constructs partitions having pairwise disjoint models, and is randomized so that the partitions differ depending on the random seed given as input to the function.

The function works in two phases. First, the function simply runs MiniSat as is to obtain heuristic values for the Boolean variables in the instance. The first phase ends when the instance is solved or a fixed time limit (currently 300 seconds) is reached. If the instance was not solved, the function enters the second phase, where the derived instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ are constructed from $\mathcal{F}$ by adding constraints. For each $1 \leq i \leq n-1$, the scattering function uses the obtained heuristic values to select a conjunction of d_i literals, $C_i = l_{i,1} \wedge \ldots \wedge l_{i,d_i}$, and conjoins it to the derived instance $\mathcal{F}_i$. The function ensures that no derived instances share models by inserting the negation $(\neg l_{i,1} \vee \ldots \vee \neg l_{i,d_i})$ of the conjunction C_i in F_i to each derived instance $\mathcal{F}_j$ with $i < j \leq n$. Scattering is a partition function, since the last instance contains only the negations of the conjunctions corresponding to the "remaining" search space. Finally, the randomization of the scattering function follows naturally from that of MiniSat: the derived instances depend on the random seed passed to MiniSat.

6 Experimental Results on Partitioning

The following experiments study the behavior of the presented approaches both under ideal and real (scattering) partitioning functions on some real-world SAT instances. A summary of the results is presented in Table 1. The instances represent hard SAT formulas in the sense that their randomized run times often exceed one hour. Furthermore, `cube-11-h14-sat` is a satisfiable instance where the scattering function always resulted in a unique satisfiable derived instance, `dated-10-13-s` is a satisfiable instance where the scattering function always resulted in several satisfiable derived instances (unless solved by the function), and `AProVE07-09` is an unsatisfiable instance.

The first column, labeled T, reports the sequential run-time distribution of the instances. The next two columns, labeled "simple distribution" in the table, report the results for Simple Distributed SAT solving for eight and 64 resources. The next four columns, labeled "ideal partitioning", report run-time distributions when an ideal partitioning function is used: first for the plain partitioning approach with eight and 64 resources, and then for safe and repeated partitioning approaches. The last four columns, labeled "scattering", report the run time distributions obtained when scattering (recall Sect. 5) is used as the partitioning function; first for plain partitioning with eight and 64 resources, and then for safe and repeated partitioning approaches with 64 resources. The rows of the table report the expected, minimum, median and maximum run times together with the first and third quartile (the values of t such that $q(t) \leq 0.25$ and $q(t) \leq 0.75$).

The run-time distributions for SDSAT and "ideal partitioning" approaches were obtained by solving the instance one hundred times with MiniSat 1.14. The resulting distributions were then used to compute the results analytically. None of the results include delays associated with parallel environments.

The distributions in columns T^8_{scatter} and T^{64}_{scatter} are obtained by running the plain partitioning approach with the scattering function fifty times using different random seeds. The resulting distribution was directly used to compute the values for the repeated partitioning approach ($T^{64}_{\text{rep-part}}$) under "scattering". To compute the results for the column $T^{64}_{\text{safe-part}}$ under "scattering", each derived instance was solved seven more times, thereby directly simulating an implementation of safe partitioning with scattering. The run times do not include the time required to run the scattering function. If the instance was solved while scattering, the run time is reported as zero.

The results in "simple distribution" columns show good scalability for `dated-10--13-s` and moderate scalability for other instances, as predicted by analytical results when t_{min} is close to $\mathbb{E}(T)$. The columns under "ideal partitioning" show that partitioning can in theory result in even better speed-up for these instances. Surprisingly, in the actual implementation ($T^8_{\text{scatter}}, T^{64}_{\text{scatter}}$) we see that plain scattering results in higher expected run-times than simple distribution for these instances. This reflects the difficulty of obtaining ideal partitioning functions.

Comparison of the "ideal partitioning" approaches confirms the discussion in Sect. 4. In particular, safe partitioning results in lower expected run time than repeated partitioning for unsatisfiable instances. However, the results under "scattering" show the opposite; repeated partitioning has consistently lower expected run time than safe partitioning. For example, observe the expected run times for safe and repeated partitioning approaches for the instance `cube-11-h14-sat` with unique satisfiable derived instance: in "ideal partitioning" they are equal, whereas the scattering-based safe partitioning is significantly worse than the scattering-based repeated partitioning approach. To study this, we computed run-time distributions for some of the satisfiable derived instances (not shown in the table), and it turns out that their expected run times varied between 109.1 and 4,773 seconds. Thus the hardness (expected run time) of a derived instance produced by scattering is also a random variable with possibly a very large range, and running the scattering function independently several times increases the probability of finding derived instances with low expected run times. This explains the good speed-up obtained by repeated partitioning when compared to safe partitioning.

Table 1. Comparing approaches for parallel search

		simple distrib.		ideal partitioning				scattering			
	T	T^{8}_{sdsat}	T^{64}_{sdsat}	$T^{8}_{\text{part(ideal)}}$	$T^{64}_{\text{part(ideal)}}$	$T^{64}_{\text{safe-part}}$	$T^{64}_{\text{rep-part}}$	T^{8}_{scatter}	T^{64}_{scatter}	$T^{64}_{\text{safe-part}}$	$T^{64}_{\text{rep-part}}$
					`cube-11-h14-sat`						
Exp	4,832	3,110	2,685	604,0	75.50	388.7	388.7	3,537	3,378	2,265	839.6
Min	2,629	2,629	2,629	328.6	41.07	328.6	328.6	117.5	69.49	13.93	117.5
Q_1	3,641	2,748	2,629	455.1	56.89	343.5	343.5	2,291	1,193	1,665	141.7
Med	4,661	3,009	2,640	572.7	72.83	376.1	376.1	3,459	3,473	2,381	338.2
Q_3	5,730	3,362	2,741	716.3	89.53	420.2	420.2	4,662	5,288	3,182	1,719
Max	10,050	10,050	10,050	1,256	157.0	1,256	1,256	11,500	7,199	4,405	11,500
					`dated-10-13-s`						
Exp	2,266	128.2	16.45	16.02	0.2570	2.056	2.056	261.2	317.3	21.22	0.2566
Min	10.09	10.09	10.09	1.262	0.1577	1.262	1.262	0	0	0	0
Q_1	283.2	28.00	10.09	3.499	0.1577	1.262	1.262	0	0	0	0
Med	784.7	109.4	10.58	13.67	0.1653	1.322	1.322	19.86	9.858	7.037	0
Q_3	2,093	181.4	17.63	22.68	0.2755	2.204	2.204	227.7	84.70	17.77	0
Max	37,930	37,930	37,930	4,741	529.6	4,741	4,741	6,449	10,095	172.0	6,449
					`AProVE07-09`						
Exp	4,016	2,361	1,685	759.7	117.7	392.5	586.7	2,598	1,719	1,632	1,559
Min	1,552	1,552	1,552	194.1	24.26	194.1	194.1	1,261	486.3	723.9	1,261
Q_1	3,086	2,033	1,552	668.0	106.2	352.3	554.2	1,757	1,122	1,271	1,466
Med	3,905	2,389	1,563	770.6	110.3	396.4	581.4	2,267	1,437	1,637	1,567
Q_3	4,732	2,666	1,736	843.7	129.8	414.3	606.4	3,550	1,894	1,912	1,694
Max	9,302	9,302	9,302	1,163	145.3	1,163	1,163	4,539	6,617	2,968	4,539

7 Conclusions

The paper investigates distributed techniques for solving challenging SAT instances and focuses on combining constraint-based search space partitioning with randomized SAT solving techniques. The paper studies first analytically the expected run time of a plain partitioning approach where a SAT instance is partitioned and then a randomized SAT solver is used to solve the resulting instances. Analytical results are derived for two limiting cases, for ideal and void partitioning functions. The investigation is then extended to a setting where simple distribution and partitioning are mixed. Finally the paper proposes a randomized partitioning function and compares the function against the ideal case.

The analytical results show that partitioning can potentially lead to catastrophic failures where an increase in computing resources leads to a decrease in solving efficiency for unsatisfiable instances. The empirical results show in part that a good implementation is usually able to avoid this failure, but plain partitioning can nevertheless be worse than an approach based on simple distributed SAT solving (SDSAT). Both problems are avoided in practice with safe and repeated partitioning. The experimental and analytical comparisons show an interesting relationship between the safe and repeated partitioning

approaches, suggesting that an ideal partitioning function would profit from safe partitioning whereas randomness in the partitioning function can be better exploited with repeated partitioning.

Acknowledgments. The authors are grateful for the financial support of the Academy of Finland (projects 122399 and 112016), Helsinki Graduate School in Computer Science and Engineering, Jenny and Antti Wihuri Foundation, Emil Aaltosen Säätiö, Finnish Foundation for Technology Promotion, and Technology Industries of Finland Centennial Foundation.

References

1. Hyvärinen, A.E.J., Junttila, T.A., Niemelä, I.: Strategies for solving SAT in grids by randomized search. In: Autexier, S., Campbell, J., Rubio, J., Sorge, V., Suzuki, M., Wiedijk, F. (eds.) AISC 2008, Calculemus 2008, and MKM 2008. LNCS (LNAI), vol. 5144, pp. 125–140. Springer, Heidelberg (2008)
2. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Information Processing Letters 47(4), 173–180 (1993)
3. Luby, M., Ertel, W.: Optimal parallelization of Las Vegas algorithms. In: Enjalbert, P., Mayr, E.W., Wagner, K.W. (eds.) STACS 1994. LNCS, vol. 775, pp. 463–474. Springer, Heidelberg (1994)
4. Huberman, B.A., Lukose, R.M., Hogg, T.: An economics approach to hard computational problems. Science 275(5296), 51–54 (1997)
5. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence 126(1–2), 43–62 (2001)
6. Hyvärinen, A.E.J., Junttila, T., Niemelä, I.: Incorporating clause learning in grid-based randomized SAT solving. Journal on Satisfiability, Boolean Modeling and Computation 6, 223–244 (2009)
7. Böhm, M., Speckenmeyer, E.: A fast parallel SAT-solver: Efficient workload balancing. Annals of Mathematics and Artificial Intelligence 17(4–3), 381–400 (1996)
8. Zhang, H., Bonacina, M., Hsiang, J.: PSATO: A distributed propositional prover and its application to quasigroup problems. Journal of Symbolic Computation 21(4), 543–560 (1996)
9. Jurkowiak, B., Li, C., Utard, G.: A parallelization scheme based on work stealing for a class of SAT solvers. Journal of Automated Reasoning 34(1), 73–101 (2005)
10. Hyvärinen, A.E.J., Junttila, T., Niemelä, I.: A distribution method for solving SAT in grids. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 430–435. Springer, Heidelberg (2006)
11. Gomes, C.P., Selman, B., Crato, N., Kautz, H.A.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. Journal of Automated Reasoning 24(1/2), 67–100 (2000)
12. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
13. Hyvärinen, A.E.J.: Approaches to grid-based SAT solving. Research Report TKK-ICS-R16, Helsinki University of Technology (June 2009)