Fundamenta Informaticae XXI (2001) 1001–1023 1001
IOS Press

Partitioning Search Spaces of a Randomized Search*

Antti E. J. Hyvärinen
Aalto University
Department of Information and Computer Science
P.0.Box 15400, FI-00076 AALTO, Finland
Antti.Hyvarinen@tkk.fi

Tommi Junttila
Aalto University
Department of Information and Computer Science
P.0.Box 15400, FI-00076 AALTO, Finland
Tommi.Junttila@tkk.fi

Ilkka Niemelä
Aalto University
Department of Information and Computer Science
P.0.Box 15400, FI-00076 AALTO, Finland
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.

Keywords: SAT solving, Randomized search, Distributed constraint-based search

1. Introduction

In this work 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

Address for correspondence: Antti E. J. Hyvärinen, Aalto University, Department of Information and Computer Science, P.0.Box 15400, FI-00076 AALTO, Finland, email: Antti.Hyvarinen@tkk.fi

*This is an extended version of a paper that appeared in the 11th Conference of the Italian Association for Artificial Intelligence (AI*IA 2009) [19].

subroutines or with little modification, in this way taking 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 speedups even when used in a grid environment with substantial communication and other delays [17]. The approach could be extended by applying particular restart strategies [29, 28] or using an algorithm portfolio scheme [14, 9]. 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 [18]. 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 [2, 38, 24] and scattering [16].

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 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. For example, partitioning, randomization, and the number of solutions all affect the expected time required to determine the satisfiability of a problem instance. We try to develop the understanding by studying the reduction in computation time when we move from solving the original instance to solving the instances derived by the partitioning method. This effectiveness is determined by the difference between the run time distribution of the original instance and that of the derived instances.

In this paper we investigate the interplay of partitioning and randomized solution techniques for SAT. We study analytically the expected run time of different combinations of partitioning and randomized solving under various assumptions concerning the effectiveness of the partitioning method and the number of solutions. We do not need any SAT specific assumptions in this study and, hence, the obtained analytic results generalize to a setting covering a wide range of search problems and corresponding randomized search procedures. The setting can be described as follows.

- We are given a search problem and a randomized complete search procedure for it. We assume that the run time of the procedure on a problem instance can be seen as a random variable with some distribution.

- We are given a partition method which takes as input a problem instance and divides the instance in some way to a given number of derived instances such that the problem instance has a solution if and only if at least one of the derived instances has a solution.

- We assume that the derived instances can be solved using the randomized search procedure and its run time on a derived instance can also be treated as a random variable with some distribution.

In addition to the general analytic results, we perform an extensive experimental study on the expected run times using SAT as the search problem, a state-of-the-art randomized SAT solver as the

search procedure and a previously proposed partitioning methods. The set of benchmarks consists of all instances from the applications category of the SAT solver competition organized as a satellite event of the SAT 2009 conference (SATCOMP-2009) as well as some other instances from the previous competitions[1]. The results from these experiments suggest potential weaknesses in approaches relying solely on partitioning and that in practise one should probably also randomize the partition function in order to increase the probability of obtaining good partitionings.

1.1. Related Work

Speedup anomalies, such as super-linear speedups, have been observed in several different implementations of parallel tree-based search. For example, [36] reports super-linear speedups in solving randomly generated satisfiable SAT instances in a parallel computing environment. Similar studies have been performed on many other related areas, such as or-trees [27], constraint logic programming [37, 31] and others [11]. Also occasional slowdowns in computation time have been reported for parallel tree-based search [25]. Both types of anomalies have been attributed to the distribution of solutions in the search space [32]. This work takes a different view on the subject by considering the high run time variations observed in modern SAT solvers [10]. The run time variation is formalized as probabilistic distributions, which can be used to explain both average-case slowdown and speedup.

Considering the simple distribution approach, it is often the case that randomization or altering the search heuristic provides substantial speedup. In particular, super-linear speedups observed in average run times of several benchmark families motivate improvements in the sequential algorithms. One such improvement, integrated to most modern SAT solvers, is the use of restart strategies [29, 14]. In the context of distributed solving the same phenomenon is studied, for example, in [22, 23] by altering the order in which the search space is traversed. In [32, 33], a similar approach is used in analyzing run times of satisfiable instances. Our alternate formulation seems to describe more accurately the experimentally observed behavior of SAT solvers.

Several parallel and distributed SAT solvers have been developed both using ideas based on partitioning and on simple distribution. This work aims at explaining the experimental observations made with solvers using a more analytical setting. We extend the results in [35] by analyzing the worst-case behavior of partitioning, separate studies on satisfiable and unsatisfiable instances, and the strategies mixing both partitioning and simple distribution. A direction orthogonal to our research involves inter-process communication and clause learning [4, 21, 34, 5, 13, 12, 1]. Some such systems, such as [8] with no load balancing and [24] with no clause learning, correspond fairly well to our approach, whereas more research is needed to establish to what extent the observations made in this work can improve partitioning with learned clauses. Partitioning and knowledge sharing has recently also been applied to parallel and distributed quantified boolean formula solving [26]; an interesting research topic would be to further investigate the roles of partitioning and randomization in QBF solving. This work contributes also to the experimental study in [3] on comparing partitioning and simple distribution. The partitioning approach followed in this work allows separation of the search heuristic from the heuristic used for constructing partitions, and therefore resembles that of [6].

Finally there are two closely related works that extend and complement the ideas presented in this work. The *partition tree* approach, first described in [16], can be seen as another approach for obtaining

[1] see `http://www.satcompetition.org/`

speedup with randomized solvers using partitioning, and has proved surprisingly efficient in solving SAT instances [20]. Interestingly, a recent, independent study [30] experiments with an approach closely related to the repeated partitioning approach (see Sect. 4) studied in this paper: in addition to running several independent copies of partitioning, clause learning is also applied in [30]. The results, both in [30] and in this paper, show that the approach enables better scalability to large amounts of resources when compared to the basic partitioning approach.

1.2. Outline

The rest of the work is structured as follows. Section 2 briefly reviews 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, i.e., 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 in an extensive experimental study. Conclusions are given in Section 7.

2. Randomization and Simple Distribution

Almost all modern state-of-the-art SAT solvers are based on backtracking search enhanced with sophisticated heuristics and search space pruning techniques. It is often the case that one branching decision would result in significantly smaller run time than another decision. Taking the wrong decision results in the solver getting stuck at hard subproblems, and to lessen this negative effect, most of these solvers also apply *restarts* and some form of *randomization* [10]. For instance, MiniSat [7] version 1.14 restarts the search periodically and makes 2% of its branching decisions pseudo-randomly. Use of restarts and randomness results in robustness in solving instances from a given instance family, but 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 that $t_{\max}$ is finite). Thus, we can 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 $\mathbb{E}(T) = \int_{t_{\min}}^{t_{\max}} t q'_T(t) \mathrm{d}t$. 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_{sd}. The cumulative run time distribution is now improved from $q_T(t)$ of the sequential case to $q_{T^n_{\text{sd}}}(t) = 1 - (1 - q_T(t))^n$. 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 $\mathbb{E}(T^8_{\text{sd}}) \approx 31$s (that is, around 20 times faster). For a more detailed analysis of running SDSAT in a parallel, distributed environment involving communication and other delays, see [17].

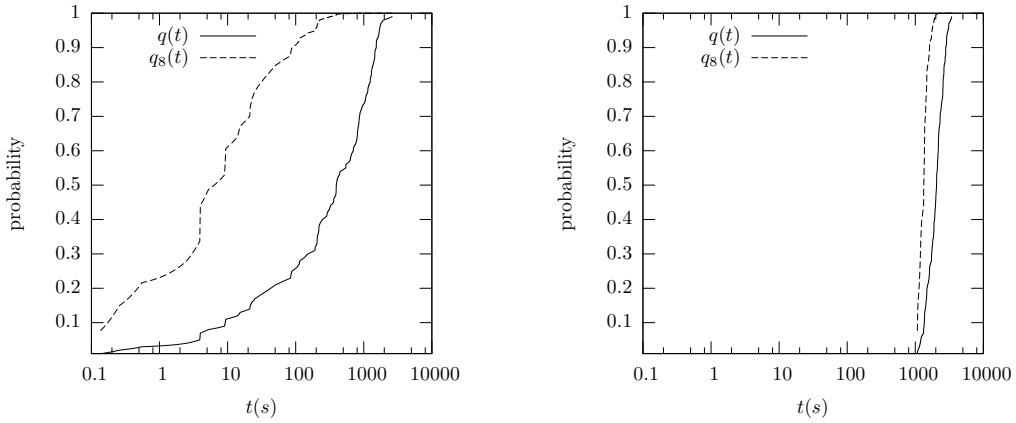

Figure 1. The run time distributions of two instances (a satisfiable one with many solutions on left and an unsatisfiable one on right). The $q(t)$ graph gives the run time distribution of a sequential, randomized SAT solver, while the $q_8(t)$ graph gives the distribution when eight such solvers are run in parallel.

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 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 $\mathbb{E}(T^8_{\text{sd}}) \approx 1,334$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 2.1. $t_{\text{min}} \leq \mathbb{E}(T^n_{\text{sd}}) \leq \mathbb{E}(T)$ for each $n \geq 1$. Furthermore, $\mathbb{E}(T^n_{\text{sd}}) \to t_{\text{min}}$ when $n \to \infty$.

Proof:
Recall that by definition $q_T(t) = 0$ for all $t < t_{\text{min}}$ and $q_T(t) = 1$ for all $t \geq t_{\text{max}}$. By applying integration by parts, i.e. $\int_a^b f(t)g'(t)\mathrm{d}t = f(b)g(b) - f(a)g(a) - \int_a^b f'(t)g(t)\mathrm{d}t$, the expected value given by $\mathbb{E}(T) = \int_0^{t_{\text{max}}} tq'_T(t)\mathrm{d}t$ can be rewritten as

$$\mathbb{E}(T) = t_{\text{max}} - \int_0^{t_{\text{max}}} q_T(t)\mathrm{d}t. \tag{1}$$

Recall that $q_{T^n_{\text{sd}}}(t) = 1 - (1 - q_T(t))^n$ and thus $q_{T^n_{\text{sd}}}(t) = 0$ when $t < t_{\text{min}}$, $0 < q_{T^n_{\text{sd}}}(t) \leq 1$ when $t_{\text{min}} \leq t < t_{\text{max}}$, and $q_{T^n_{\text{sd}}}(t) = 1$ when $t \geq t_{\text{max}}$. Now $\int_0^{t_{\text{max}}} q_{T^n_{\text{sd}}}(t)\mathrm{d}t \leq t_{\text{max}} - t_{\text{min}}$ and therefore $\mathbb{E}(T^n_{\text{sd}}) = t_{\text{max}} - \int_0^{t_{\text{max}}} q_{T^n_{\text{sd}}}(t)\mathrm{d}t \geq t_{\text{min}}$.

As $0 \leq q_T(t) \leq 1$ and $(1 - q_T(t))^n \leq (1 - q_T(t))$ for all $n \geq 1$, we have that $q_{T^n_{\text{sd}}}(t) = 1 - (1 - q_T(t))^n \geq q_T(t)$. This implies that $\int_0^{t_{\text{max}}} q_{T^n_{\text{sd}}}(t)\mathrm{d}t \geq \int_0^{t_{\text{max}}} q_T(t)\mathrm{d}t$ and thus $\mathbb{E}(T^n_{\text{sd}}) \leq \mathbb{E}(T)$.

When $n \to \infty$, we have that $q_{T^n_{\text{sd}}}(t) = 1 - (1 - q_T(t))^n \to 1$ for all $t_{\text{min}} \leq t \leq t_{\text{max}}$. Thus, $\int_0^{t_{\text{max}}} q_{T^n_{\text{sd}}}(t)\mathrm{d}t \to t_{\text{max}} - t_{\text{min}}$ and $\mathbb{E}(T^n_{\text{sd}}) = t_{\text{max}} - \int_0^{t_{\text{max}}} q_{T^n_{\text{sd}}}(t)\mathrm{d}t \to t_{\text{min}}$. □

As illustrated earlier in this section, SDSAT can allow super-linear speedup (meaning $\mathbb{E}(T^n_{\text{sd}}) < \mathbb{E}(T)/n$) for instances that have a strongly varying run time. However, as the maximum speedup obtainable with SDSAT is $\mathbb{E}(T)/t_{\min}$, super-linear speedup can only happen for small 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

$$\mathcal{F} \equiv \mathcal{F}_1 \vee \cdots \vee \mathcal{F}_n. \tag{2}$$

Now, in order to find whether $\mathcal{F}$ is satisfiable or not, we solve, in parallel, all the derived instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ and deduce that $\mathcal{F}$ is satisfiable if and only 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 Section 4. One way to implement partitioning functions is described in [16] (also see Section 5), where each $\mathcal{F}_i$ is obtained from $\mathcal{F}$ by conjoining it with a set of additional partitioning constraints.[2] In addition to the requirement (2), partitioning functions often ensure that the models of $\mathcal{F}_1, \ldots, \mathcal{F}_n$ are mutually disjoint.

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 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 present 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: an 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 Section 4 after some composite approaches mixing simple distribution and plain partitioning have been described, and again in Section 6.2 in conjunction with a detailed experimental analysis. A real partitioning function is explained in Section 5 and experimented with in Section 6.

3.1. Unsatisfiable Instances

Assume that an unsatisfiable instance $\mathcal{F}$ is partitioned into n new derived instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ fulfilling Eq. (2). *All these new instances* need to be shown unsatisfiable to deduce that $\mathcal{F}$ is unsatisfiable. When

[2] As explained in [15], guiding paths [2, 38] can also be interpreted as partitioning constraints.

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 the run time distribution $q_{T_i}(t) = q_T(nt)$. 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 $q_{T^n_{\text{part(ideal)}}}(t) = q_T(nt)^n$. Based on this, we have the following interesting results. First, ideal partitioning functions can provide *at most linear expected speedup* on unsatisfiable instances:

Proposition 3.1. $\mathbb{E}(T^n_{\text{part(ideal)}}) \geq \mathbb{E}(T)/n$ for each $n \geq 1$.

Proof:
By definition, the probability that an ideally partitioned instance is solved in time t or less is $q_T(nt)$. Since the instance is unsatisfiable, all n instances need to be solved by the partitioning approach. Therefore, for unsatisfiable instances, the probability that the plain partitioning approach has solved the instance is $q_{T^n_{\text{part(ideal)}}}(t) = q_T(nt)^n$. By (1), the expected run time is

$$\mathbb{E}(T^n_{\text{part(ideal)}}) = \frac{t_{\max}}{n} - \int_0^{t_{\max}/n} q_T(nt)^n \mathrm{d}t = \frac{t_{\max}}{n} - \int_0^{t_{\max}} \frac{1}{n} q_T(\tau)^n \mathrm{d}\tau,$$

where the last equality is obtained by substituting $t = \tau/n$ and $\mathrm{d}t = \frac{\mathrm{d}\tau}{n}$ in the middle equation.

Since $q_T(t) \leq 1$ for all $t \geq 0$, we have $q_T(t)^n \leq q_T(t)$ when $n \geq 1$. Thus,

$$\mathbb{E}(T) = t_{\max} - \int_0^{t_{\max}} q_T(t)\mathrm{d}t \leq t_{\max} - \int_0^{t_{\max}} q_T(t)^n \mathrm{d}t = n\mathbb{E}(T^n_{\text{part(ideal)}}).$$

Note that the probability that one of the parallel solvers ends up running an execution corresponding to the highest run time $t_{\max}/n$ approaches 1 as $n \to \infty$. Therefore, $\mathbb{E}(T^n_{\text{part(ideal)}}) \to \frac{t_{\max}}{n}$ when $n \to \infty$, that is, the expected run time of the plain partitioning approach under the ideal partitioning function tends to $t_{\max}/n$. Hence, linear speedup of the expected run time is only obtained when $t_{\max} = t_{\min}$. □

On the other hand, the expected run time is never worse than that of solving the original instance with one solver:

Proposition 3.2. $\mathbb{E}(T^n_{\text{part(ideal)}}) \leq \mathbb{E}(T)$ for each $n \geq 1$.

Proof:
The proposition is proven by induction on n. When $n = 1$, $\mathbb{E}(T^n_{\text{part(ideal)}}) = \mathbb{E}(T)$. Assume that the claim holds for some $n \geq 1$. To prove that the claim holds for $n + 1$, we express the expected run time using the derivative of the cumulative run time distribution. As before, $q_T(t)$ denotes the probability that the original instance is solved at or before time t. Then $q_T(nt)$ denotes the probability that a partition produced by the ideal partitioning function from the original instance is solved at or before time t. By definition, the expected run time of the plain partitioning approach for unsatisfiable instances with ideal partitioning function is

$$\mathbb{E}(T^n_{\text{part(ideal)}}) = \int_0^{t_{\max}} t q'_{T^n_{\text{part(ideal)}}}(t)\mathrm{d}t,$$

where $q'_{T^n_{\text{part(ideal)}}}(t) = \frac{\mathrm{d}}{\mathrm{d}t} q_T(nt)^n = n^2 q_T(nt)^{n-1} q'_T(nt)$. Using this and noting that $q'_T(nt) = 0$ when $t \geq t_{\max}/n$, we get

$$\mathbb{E}(T^n_{\text{part(ideal)}}) = \int_0^{t_{\max}/n} tn^2 q_T(nt)^{n-1} q'_T(nt) \mathrm{d}t.$$

By substituting $nt = \tau$ above, we get

$$\begin{aligned}\mathbb{E}(T^n_{\text{part(ideal)}}) &= \int_0^{t_{\max}} \frac{\tau}{n} n^2 q_T(\tau)^{n-1} q'_T(\tau) \frac{1}{n} \mathrm{d}\tau = \int_0^{t_{\max}} \tau q_T(\tau)^{n-1} q'_T(\tau) \mathrm{d}\tau \\ &\geq \int_0^{t_{\max}} \tau q_T(\tau)^{n} q'_T(\tau) \mathrm{d}\tau = \mathbb{E}(T^{n+1}_{\text{part(ideal)}}),\end{aligned}$$

where the inequality follows from $q_T(\tau) \leq 1$ for all $0 \leq \tau \leq t_{\max}$. □

When the number n of SAT solvers running in parallel is increased, the expected run time $\mathbb{E}(T^n_{\text{part(ideal)}})$ approaches $t_{\max}/n$, i.e., linear speedup 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 $\mathbb{E}(T^n_{\text{sd}}) < \mathbb{E}(T^n_{\text{part(ideal)}})$ and others for which $\mathbb{E}(T^n_{\text{part(ideal)}}) < \mathbb{E}(T^n_{\text{sd}})$ when n is small. For the case $\mathbb{E}(T^n_{\text{part(ideal)}}) < \mathbb{E}(T^n_{\text{sd}})$, observe the graphs *part(ideal)* and *sdsat* in the right hand side plot of Fig. 6 (the underlying distribution is the one in the right hand side plot of Fig. 1; the instance is unsatisfiable). On the other hand, Fig. 2 demonstrates an artificial unsatisfiable instance with a distribution where the SDSAT approach (the *sdsat* graph) performs better than the partitioning approach (the *part(ideal)* graph) up to large values of n. The distribution is artificially generated, having

$$q_T(t) = \begin{cases} 0 & \text{for } 0 \leq t < 1, \\ 0.8 & \text{for } 1 \leq t < 1000000, \text{ and} \\ 1 & \text{for } t \geq 1000000. \end{cases} \tag{3}$$

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 new instances $\mathcal{F}_i$ 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 partitioning functions*:

Proposition 3.3. $\mathbb{E}(T^n_{\text{part(void)}}) \geq \mathbb{E}(T)$ for each $n \geq 1$.

Proof:
Because $q_T(t) \leq 1$ for all $t \geq 0$, $q_{T^n_{\text{part(void)}}}(t) = q_T(t)^n \leq q_T(t)$ and thus, by Eq. (1), $\mathbb{E}(T^n_{\text{part(void)}}) = t_{\max} - \int_0^{t_{\max}} q_T(t)^n \mathrm{d}t \geq t_{\max} - \int_0^{t_{\max}} q_T(t) \mathrm{d}t = \mathbb{E}(T)$. □

In fact, *the more resources one uses, the closer to the maximum run time one gets*: $\mathbb{E}(T^n_{\text{part(void)}}) = t_{\max} - \int_0^{t_{\max}} q_T(t)^n \mathrm{d}t \to t_{\max}$ when $n \to \infty$.

The results presented in this section for the unsatisfiable instances are summarized in Fig. 3. The inequalities presented in the figure hold for all distributions $q_T(t)$ and all $n \geq 2$. The ordering in the figure is not total as the order between two elements may depend on both the distribution and on n. For example, speedup in SDSAT can be super-linear or sub-linear and thus the order of $\mathbb{E}(T^n_{\text{sd}})$ and $\mathbb{E}(T)/n$ depends on the distribution. Similarly, the order of $t_{\min}$ and $\mathbb{E}(T)/n$ depends on the value of n as well.

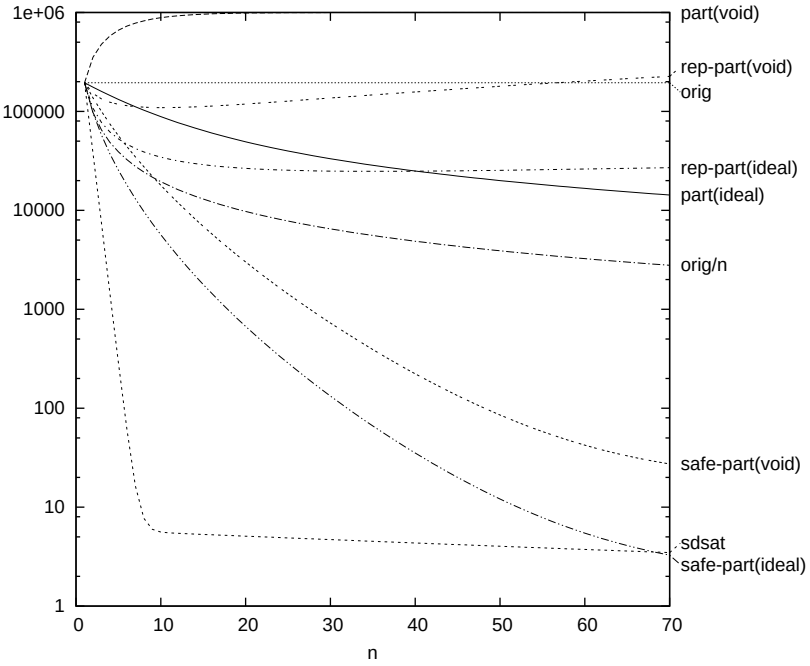

Figure 2. The expected run times of different approaches on an (artificial) unsatisfiable instance having the distribution described in Eq. (3).

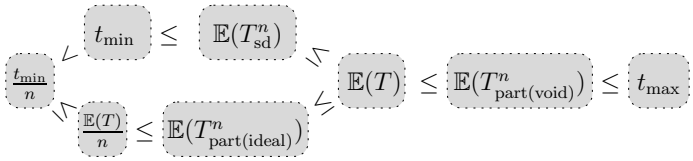

Figure 3. Partial order in terms of expected run time for the plain partitioning and the simple distribution approaches when solving unsatisfiable instances.

3.2. Satisfiable Instances with Many Solutions

We next consider the case when a satisfiable instance $\mathcal{F}$ is partitioned into n new instances $\mathcal{F}_1, \ldots, \mathcal{F}_n$ fulfilling Eq. (2). 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 all new instances $\mathcal{F}_i$ are satisfiable, postponing the case where only one is satisfiable to the next section.

Let us consider the case of ideal partitioning functions 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*: $\mathbb{E}(T^n_{\text{part(ideal)}}) = \mathbb{E}(T^n_{\text{sd}})/n$. Therefore, when compared to solving the original instance, we notice that *on satisfiable instances with many solutions we may expect at least linear speedup:*

Proposition 3.4. $\mathbb{E}(T^n_{\text{part(ideal)}}) \leq \mathbb{E}(T)/n$ for each $n \geq 1$.

Proof:
A direct consequence of the fact $\mathbb{E}(T^n_{\text{part(ideal)}}) = \mathbb{E}(T^n_{\text{sd}})/n$ and of Prop. 2.1 stating that $\mathbb{E}(T^n_{\text{sd}}) \leq \mathbb{E}(T)$. □

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 speedup 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 3.5. $\mathbb{E}(T^n_{\text{part(void)}}) = \mathbb{E}(T^n_{\text{sd}})$.

The results on the expected run times of satisfiable instances with many solutions are summarized in Fig. 4 in a form similar to that used at the end of Sect. 3.1.

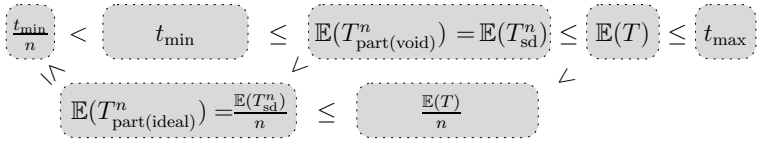

Figure 4. Partial order in terms of expected run time for the plain partitioning and the simple distribution approaches when solving satisfiable instances with many solutions.

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 $\mathcal{F}_1, \ldots, \mathcal{F}_n$ having mutually disjoint models, exactly 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 a priori known which the new instance is satisfiable.

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 speedup is obtained with an ideal partitioning function*:

Proposition 3.6. $\mathbb{E}(T^n_{\text{part(ideal)}}) = \mathbb{E}(T)/n$.

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 speedup nor in loss of efficiency*:

Proposition 3.7. $\mathbb{E}(T^n_{\text{part(void)}}) = \mathbb{E}(T)$.

The presented results are again summarized in Fig. 5.

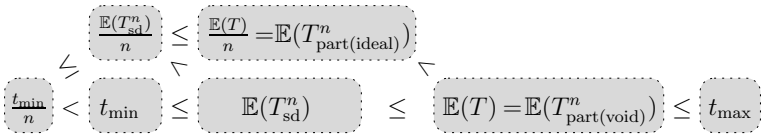

Figure 5. Partial order in terms of expected run time for the plain partitioning and the simple distribution approaches when solving satisfiable instances with exactly one solution.

4. Composite Approaches

The analysis of the previous section shows that the plain partitioning approach can potentially obtain even super-linear speedups, whereas an improper, void partition function can, by Prop. 3.3, result in expected run time worse than that of one solver.

This section studies two *composite approaches* which aim at avoiding the problem of increasing the expected run time. In particular, we are interested in analytically showing whether one of the approaches is superior to the other, and whether the approaches can guarantee that their expected run time is never higher than the expected run time of one solver. The answer to the first question turns out to be positive whereas the answer to the latter question is unfortunately negative.

Let n be the maximum number of search procedures that can be run 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 i splitting the instance $\mathcal{F}$ into k derived instances $\mathcal{F}_{i,1}, \ldots, \mathcal{F}_{i,k}$ and solve 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 simple distribution and partitioning compared to repeated partitioning: the instance $\mathcal{F}$ is partitioned into $k = \lfloor \sqrt{n} \rfloor$ derived instances $\mathcal{F}_1, \ldots, \mathcal{F}_k$ and each instance $\mathcal{F}_i$ is solved with k search procedures in parallel. The run time of this is denoted by the random variable $T^n_{\text{safe-part}}$.

We will prove properties of these two approaches, and to clarify the proofs, we introduce two operators, D^k_{sd} and D^k_{part}, which describe respectively the effects of the simple distribution and plain partitioning approaches to a run time distribution $q(t)$. Note that the definition covering the partitioning approach is more general than what is strictly required for the proofs, since it includes a function $\epsilon(k)$ capturing the effect of the partitioning to the run time: $\epsilon(k) = k$ for ideal partitioning function, and $\epsilon(k) = 1$ for void partitioning function. The operators D^k_{sd} and D^k_{part} map a distribution function to another distribution function in the following manner. Firstly,

$$D^k_{\text{sd}}(q(t)) = 1 - (1 - q(t))^k$$

is the run time distribution resulting from applying the simple distribution approach to a solving process with run time distribution $q(t)$. Secondly, the result of the operator D^k_{part} depends on whether the instance to be partitioned is satisfiable or not. In the satisfiable case we use r to denote the number of partitions

containing solutions. Then

$$D^k_{\text{part}}(q(t)) = \begin{cases} q(\epsilon(k)t)^k & \text{if all partitions are unsatisfiable, and} \\ 1-(1-q(\epsilon(k)t))^r & \text{if } r \text{ out of } k \text{ partitions are satisfiable} \end{cases}$$

is the run time distribution resulting from applying the ideal partitioning approach to a search process with run time distribution $q(t)$.

By combining these operators, we can obtain an expression for the distributions of safe partitioning and repeated partitioning approaches. In safe partitioning, an instance is partitioned once and all resulting partitions are solved with the simple distribution approach. Therefore the distribution for the unsatisfiable case is

$$D^k_{\text{part}}(D^k_{\text{sd}}(q(t)))) = (1-(1-q(\epsilon(k)t))^k)^k,$$

and for the case where r of the k partitions are satisfiable

$$D^k_{\text{part}}(D^k_{\text{sd}}(q(t))) = 1-(1-(1-(1-q(\epsilon(k)t))^k))^r.$$

When the satisfiable case is simplified, we have

$$D^k_{\text{part}}(D^k_{\text{sd}}(q(t))) = \begin{cases} (1-(1-q(\epsilon(k)t))^k)^k & \text{if unsatisfiable, and} \\ 1-(1-q(\epsilon(k)t))^{kr} & \text{if satisfiable,} \end{cases} \tag{4}$$

where r is the number of satisfiable partitions resulting from the partitioning ($1 \leq r \leq k$). In repeated partitioning an instance is split k times and each instance is solved with a single solver. Therefore the distribution for unsatisfiable instances is

$$D^k_{\text{sd}}(D^k_{\text{part}}(q(t))) = 1-(1-q(\epsilon(k)t)^k)^k$$

and for satisfiable instances with r satisfiable partitions

$$D^k_{\text{sd}}(D^k_{\text{part}}(q(t))) = 1-(1-(1-(1-q(\epsilon(k)t))^r))^k.$$

When the satisfiable case is simplified, we have as a summary

$$D^k_{\text{sd}}(D^k_{\text{part}}(q(t))) = \begin{cases} 1-(1-q(\epsilon(k)t)^k)^k & \text{if unsatisfiable, and} \\ 1-(1-q(\epsilon(k)t))^{rk} & \text{if satisfiable,} \end{cases} \tag{5}$$

where again $1 \leq r \leq k$.

We see directly from Eqs. (4) and (5) that if the instance is satisfiable, then $D^k_{\text{sd}}(D^k_{\text{part}}(q(t))) = D^k_{\text{part}}(D^k_{\text{sd}}(q(t)))$, implying $\mathbb{E}(T^n_{\text{safe-part}}) = \mathbb{E}(T^n_{\text{rep-part}})$. Therefore we have immediately the following result:

Proposition 4.1. $\mathbb{E}(T^n_{\text{safe-part}}) = \mathbb{E}(T^n_{\text{rep-part}})$ for satisfiable instances.

The question is more demanding if the instance is unsatisfiable:

Proposition 4.2. $\mathbb{E}(T^n_{\text{safe-part}}) \leq \mathbb{E}(T^n_{\text{rep-part}})$ for unsatisfiable instances.

Proof:
It suffices to show that $D^n_{\text{sd}}(D^n_{\text{part}}(q(\epsilon(n)t))) \leq D^n_{\text{part}}(D^n_{\text{sd}}(q(\epsilon(n)t)))$. This is equivalent to $1-(1-x^n)^n \leq (1-(1-x)^n)^n$ for $0 \leq x \leq 1$. We will show this by showing $f(x) = (1-(1-x)^n)^n - 1 + (1-x^n)^n \geq 0$ for $0 \leq x \leq 1$. First note that $f(x) = f(1-x)$ is symmetric with respect to $x = 1/2$. Since $f(0) = 0$, it suffices to show that $f(x)$ is increasing when $0 \leq x \leq 1/2$, that is, $\mathrm{d}/\mathrm{d}x f(x) \geq 0$, whenever $0 \leq x \leq 1/2$. By computing the derivative, we have

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}x} f(x) &= n^2 \left(1-(1-x)^n\right)^{n-1} (1-x)^{n-1} - n^2 (1-x^n)^{n-1} x^{n-1} \\ &= n^2 \left(\left(1-(1-x)^n\right)^{n-1} (1-x)^{n-1} - (1-x^n)^{n-1} x^{n-1} \right).\end{aligned}$$

Since n^2 is positive, it suffices to confirm that the parenthesized expression is positive. By rearranging the terms, we get

$$\begin{aligned}&\left(\left(1-(1-x)^n\right)^{n-1} (1-x)^{n-1} - (1-x^n)^{n-1} x^{n-1} \right) = \\ &\qquad \left((1-x) - (1-x)^{n+1} \right)^{n-1} - \left(x - x^{n+1} \right)^{n-1}.\end{aligned}$$

The expression above is positive, since the distance between $(1-x)$ and $(1-x)^{n+1}$ is greater than or equal to the distance between x and x^{n+1} whenever $0 \leq x \leq 1/2$, as can be verified by confirming that the claim holds for $n = 1$ and noting that $\mathrm{d}/\mathrm{d}n((1-x) - (1-x)^{n+1}) = -(1-x)^{n+1} \log(1-x) \geq \mathrm{d}/\mathrm{d}n(x - x^{n+1}) = -x^{n+1} \log x$. The latter can be shown using induction by noting that $-(1-x)^n \log(1-x) \geq -x^n \log x$ for $n = 2$, assuming that the claim holds for $n = k$ and noting that in this case also $-(1-x)^{k+1} \log(1-x) \geq -x^{k+1} \log x$, since $(1-x) \geq x$ when $0 \leq x \leq 1/2$. □

Simulations suggest an even stronger result for the ordering of safe and repeated partitioning approaches: according to them, $D^m_{\text{sd}}(D^n_{\text{part}}(q(\epsilon(n)t))) \leq D^n_{\text{part}}(D^m_{\text{sd}}(q(\epsilon(n)t)))$ for all $m, n \geq 1$.

By Props. 4.1 and 4.2 the cumulative run time distribution of safe partitioning is greater than the cumulative run time distribution of repeated partitioning for all t. We now show that when the number of resources n is fixed, there are distributions of unsatisfiable instances for which the expected run time of the safe partitioning approach is greater than the expected run time of a single solver. From this it directly follows that whenever partitioning is performed, it is possible that the expected performance of the approach is worse than that of a single solver. An example is a two-step distribution similar to the one described in Eq. (3), where probability of solving the instance exactly at time t_1 is p and the probability of solving the instance exactly at time t_2 is $(1-p)$. The expected run time of a single solver for this type of instance is $\mathbb{E}(T) = pt_1 + (1-p)t_2$. The safe partitioning approach with void partitioning function has the expected run time $\mathbb{E}(T^n_{\text{safe-part(void)}}) = \left(1-(1-p)^n\right) t_1 + \left(1-(1-(1-p)^n)^n\right) t_2$. For example, if $p = 0.01$, $t_1 = 1$, $t_2 = 1,000,000$ and $n = 2$, the expected run time of safe partitioning approach is approximately 1% higher than the expected run time of a single solver.

In practice such distributions seem to be rare, and safe partitioning approach performs well in this analytical setting. To illustrate the approaches and the results presented in Sects. 2, 3, and 4, Fig. 6 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 behavior of the SDSAT as well as all the considered partitioning approaches applying a void partitioning function are depicted by the same graph (the *sdsat+others* graph). The instance at the right hand side is unsatisfiable.

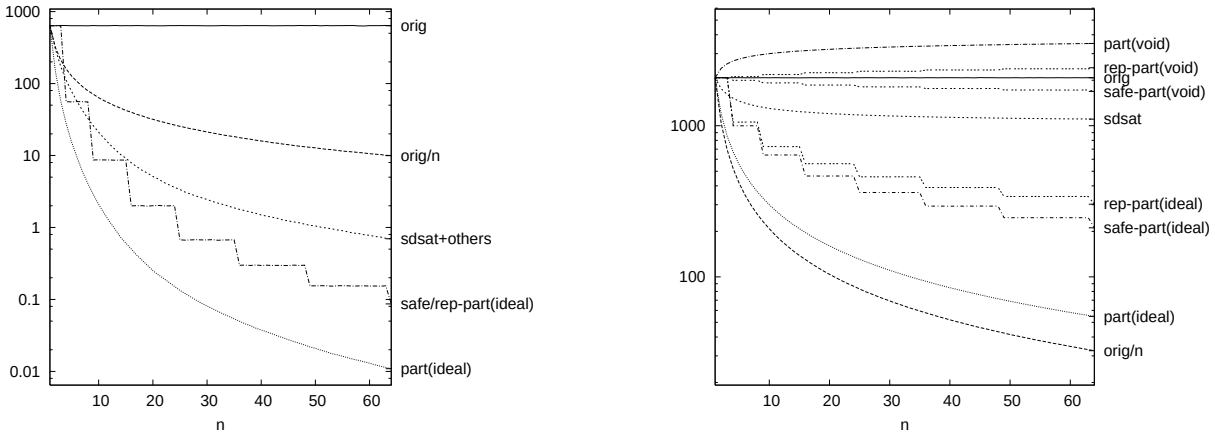

Figure 6. Expected run times (in seconds) of different approaches on the instances in Fig. 1 when the number n of SAT solvers run in parallel is varied.

Finally, we refer back to the results for the distribution described in (3) that are presented in Fig. 2. The distribution is useful in demonstrating some anomalies that can be observed with repeated partitioning. For example, repeated partitioning with void partitioning function (the *rep-part(void)* graph) may result in an expected run time which is higher than the sequential expected run time (the *orig* graph) for large n, and its performance may be decreasing for some range of n even with ideal partitioning function (the *rep-part(ideal)* graph). For this particular function, a "local minimum" for the expected run time of repeated partitioning with an ideal partitioning function is reached when $n \approx 35$, and is only reached again when $n \approx 1600$.

5. Implementing a Randomized Partitioning Function

We have implemented a partition function to observe the differences and similarities between the analytical model discussed in the previous sections and the practice. This section briefly describes how a partitioning function called *scattering* [16] is implemented. The implementation is based on MiniSat 1.14, it 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 (300 seconds in these experiments) 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 C_i 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 $\mathcal{F}_i$ 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.

The number d_i of literals chosen for each conjunction C_i is determined so that each partition should have an equally large search space. Let $t(\mathcal{F})$ denote the estimate of the time required to solve a formula $\mathcal{F}$

sequentially. Then the time required to solve each $\mathcal{F}_i$ should be comparable to $t(\mathcal{F})/n$. Since the solution spaces of the instances are distinct, we may assume that the solving times $t(\mathcal{F}_j) = t(\mathcal{F})/n, 1 \leq j < i$ of the previously constructed problems can be subtracted from the total solving time $t(\mathcal{F})$ to get the time required to solve the remaining problem. Since the time for the problem $\mathcal{F}_i$ should also be equal to $t(\mathcal{F})/n$, a proportion r_i should be constructed from the remaining problem of which the run time is approximately $t(\mathcal{F}) - (i-1)t(\mathcal{F})/n$, yielding the equation

$$\frac{t(\mathcal{F})}{n} = \left(t(\mathcal{F}) - (i-1)\frac{t(\mathcal{F})}{n}\right) r_i,$$

which, when solved for r_i, becomes

$$r_i = \frac{1}{n-i+1}.$$

As an approximation of the computational cost we assume the worst case behavior, that is, $t(\mathcal{F}) = O(2^m)$, where m is the number of variables in $\mathcal{F}$. Hence, the problem is reduced to that of finding d_i which minimizes the difference $|2^{-d_i} - r_i|$. For example in the experiments with $n = 8$ we used the values $d_i = 3$ for $1 \leq i \leq 3$, $d_i = 2$ for $4 \leq i \leq 6$, $d_7 = 1$ and $d_8 = 0$. Similarly, in the experiments with $n = 64$ we used the values $d_i = 6$ for $1 \leq i \leq 22$, $d_i = 5$ for $23 \leq i \leq 43$, $d_i = 4$ for $44 \leq i \leq 54$, $d_i = 3$ for $55 \leq i \leq 59$, $d_i = 2$ for $60 \leq i \leq 62$, $d_{63} = 1$ and $d_{64} = 0$.

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

This section studies how the approaches presented in this work perform in solving real-world SAT instances from SAT competitions. We first present results on all the benchmark instances from the the *applications* category of the International SAT Competition 2009 (SATCOMP-2009, `http://www.satcompetition.org/`), and then study more closely the distributions of some hard instances from previous competitions.

6.1. SAT Competition 2009 Instances

The experiments in this section provide an overview on how the presented approaches perform in solving structured SAT problem instances. The benchmark set for this experiment consists of the 292 instances from the *applications* category of the SATCOMP-2009 solver competition. The results are summarized in Figs. 7 (satisfiable instances) and 8 (unsatisfiable instances). Solving of all the instances was attempted once using each of the following approaches: the sequential solver (the *sd1* graph), simple distribution with 64 parallel cores (the *sd64* graph), partitioning with the scattering function to 8 (the *sc8* graph) and 64 (the *sc64* graph) partitions, and the safe (the *safe-part* graph) and repeated (the *rep-part* graph) partitioning approaches using 64 parallel cores. In all approaches making use of partitioning (the safe, repeated, and plain partitioning approaches) the scattering function was allowed to run for 300 seconds, after which it had to produce the partitions. Two choices were made to simplify the interpretation of the results. Firstly, the learned clauses obtained during the 300 seconds were not included to the resulting partitions and therefore the speedup cannot be caused by this type of preprocessing. Secondly, the results

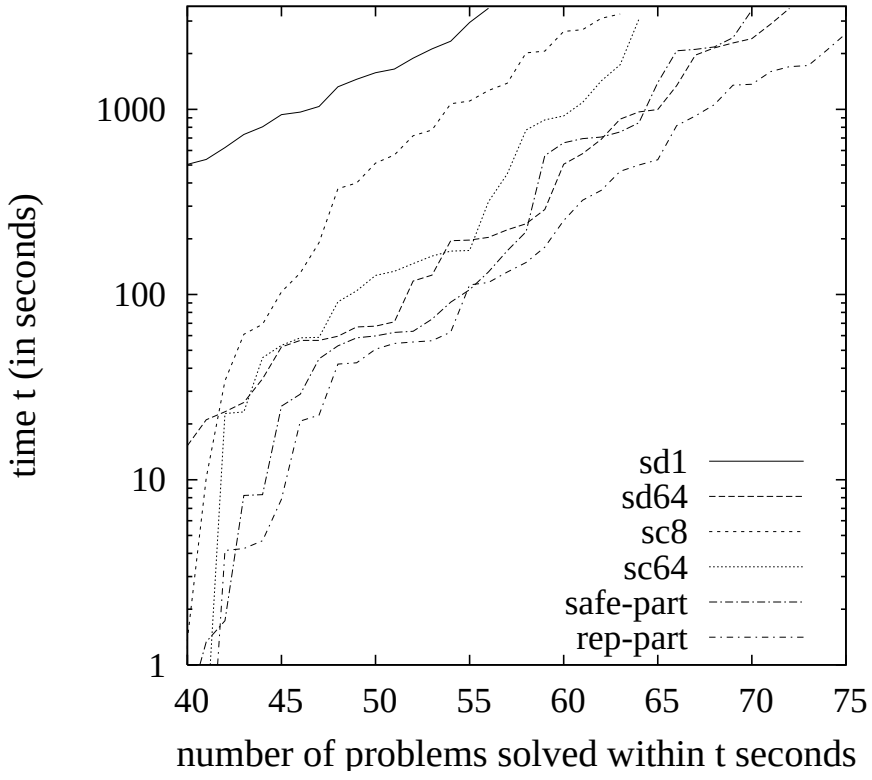

Figure 7. Results for satisfiable instances from the *applications* category of SATCOMP-2009

do not include the scattering function run time. If the scattering function solved the instance during the 300 seconds, the run time is reported as 0. The results do not include communication delays and were obtained using Intel Xeon 5130 and E5345 CPUs running at 2 GHz and 2.33 GHz respectively, and AMD Opteron CPUs running at 2.6 GHz, 2.2 GHz and 1.8 GHz. Although the computing environment was heterogeneous, it was ensured that each graph data was obtained using a similar proportion of the available CPUs, hence making the graphs comparable[3]. Each parallel execution was limited to using one hour of wall clock time, and one gigabyte of memory, both values determined based on earlier experience from running similar experiments. The results are presented by showing, for each approach, the number of instances solved within a given wall-clock time.

Among the approaches presented in this work, the best way of using 64 computers to solve an instance is the repeated partitioning approach. This results from the randomness in the partition function and therefore seemingly contradicts with our analytical study stating that safe partitioning should perform better than repeated partitioning. Experiments in the following sub-section confirm this phenomenon using a homogeneous environment.

Considering satisfiable instances and the plain partitioning approach with the scattering partitioning

[3]The proportions were approximately 33% Xeon 5130, 14% E5345, 14% Opteron 2.6 GHz, 14% Opteron 2.2 GHz, and 25% Opteron 1.8 GHz consistently on all 6 approaches.

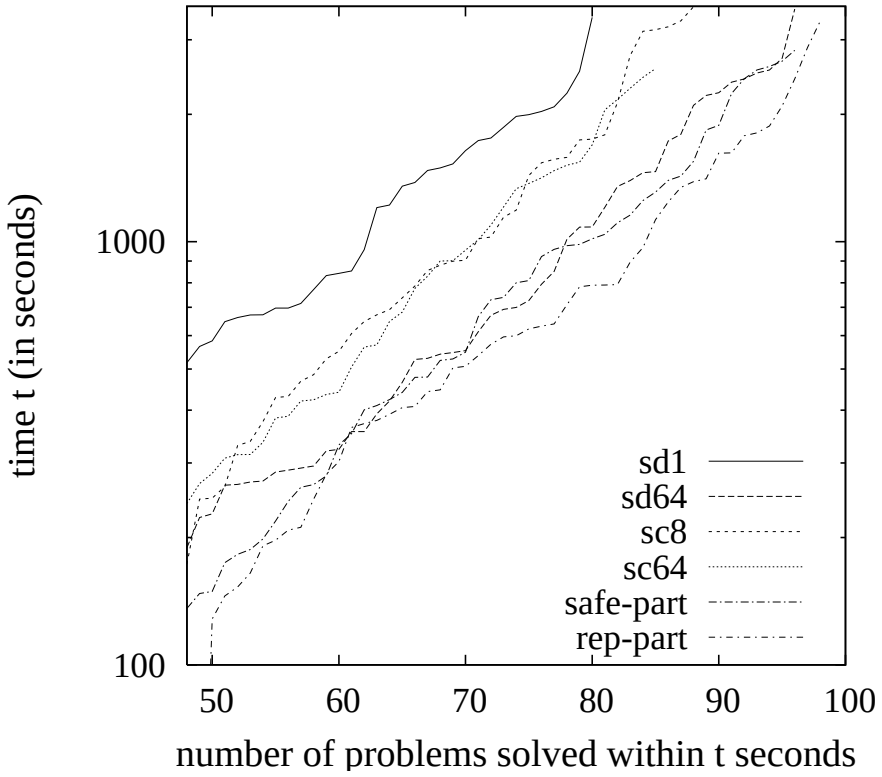

Figure 8. Results for unsatisfiable instances from the *applications* category of SATCOMP-2009

function, increasing the number of parallel cores from 8 to 64 (Fig. 7, the *sc8* and *sc64* graphs) helps to solve more instances. However, the difference is mainly in the time required to solve the instances and not in the number of instances solved within one hour. Increasing the number of resources in solving unsatisfiable instances (Fig. 8) shows another slightly surprising effect. The plain partitioning approach does not seem to scale when resource usage is increased from 8 (the *sc8* graph) to 64 (the *sc64* graph). Using 64 cores effectively resulted in fewer problems being solved within an hour. While outside the scope of this work, a closer study should be performed on different partition heuristics to reveal if this is an artifact of the scattering approach or if these results generalize to other partitioning functions as well. We also observe that using simple distribution (the *sd64* graph) is very competitive compared to the partition-based approaches, and that only combining the two approaches (the *safe-part* and the *rep-part* graphs) results in better performance.

Finally, the results agree with the analytical study in that speedup is, in general, more easily obtained for the satisfiable instances (Fig. 7) than for the unsatisfiable instances (Fig. 8).

6.2. Further Analysis with Some Hard Instances

This section deepens the previous analysis by concentrating on fewer instances in a much more controlled environment. All instances are solved using the same CPU (Intel Xeon 5130 running at 2 GHz). While the CPU is dual-core and each computer featured two such CPUs, the experiments assured that exactly one solver was running at any time on the computer. Therefore the results should be unaffected by any interference caused, for example, by memory bus congestion. All instances were run until a solution was found, which enables computing a fairly reliable estimate of the run time distributions for these instances. A summary of the results is presented in Tables 1 and 2. The instances represent hard SAT

Table 1. Comparing approaches for parallel search

		simple distrib.		ideal partitioning				scattering			
	T	T^8_{sd}	T^{64}_{sd}	$T^8_{\text{part(ideal)}}$	$T^{64}_{\text{part(ideal)}}$	$T^{64}_{\text{safe-part}}$	$T^{64}_{\text{rep-part}}$	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

formulas in the sense that their run times often exceed one hour. They represent the classes of instances discussed in the analytical study. In particular, `cube-11-h14-sat` is a satisfiable instance where the scattering function always resulted in exactly one satisfiable derived instance, and `dated-10-13-s` is a satisfiable instance where the scattering function always resulted in several satisfiable derived instances (unless solved by the function). The instance `AProVE07-09` is an unsatisfiable instance, as are the four instances described in Table 2. 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 (SDSAT) for 8 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

Table 2. Comparing approaches for parallel search

		simple distrib.		ideal partitioning				scattering			
	T	T^8_{sd}	T^{64}_{sd}	$T^8_{\text{part(ideal)}}$	$T^{64}_{\text{part(ideal)}}$	$T^{64}_{\text{safe-part}}$	$T^{64}_{\text{rep-part}}$	T^8_{scatter}	T^{64}_{scatter}	$T^{64}_{\text{safe-part}}$	$T^{64}_{\text{rep-part}}$
					`linvrinv5`						
Exp	2829	2440	2207	417.3	60.11	331.9	367.1	1354	1311	1107	986.4
Min	2168	2168	2168	271.1	33.88	271.1	271.1	895.9	815.5	738.5	895.9
Q_1	2644	2297	2168	377.7	56.33	323.4	359.2	1068	1088	873.8	904.6
Med	2799	2481	2172	414.2	62.11	331	366.9	1232	1229	1018	1000
Q_3	2943	2548	2225	443.8	62.93	339.2	372.3	1707	1604	1359	1033
Max	4046	4046	4046	505.7	63.22	505.7	505.7	2682	1896	2301	2682
					`contest03-SGI_30_50_30_20_3-dir`						
Exp	1433	1221	1116	210.6	29.24	165.4	191.3	1324	1107	1111	1060
Min	1076	1076	1076	134.6	16.82	134.6	134.6	925.8	650.9	889.3	925.8
Q_1	1309	1186	1076	200.8	28.36	161.4	185.4	1174	944.7	1027	955.1
Med	1421	1222	1083	206.6	29.23	164.5	192.2	1332	1115	1124	1071
Q_3	1550	1273	1169	220.4	30.32	169.1	196.2	1461	1272	1197	1142
Max	1993	1993	1993	249.2	31.14	249.2	249.2	1698	1588	1284	1698
					`hwb-n28-02-S818962541`						
Exp	4654	4031	3624	667.5	93.19	561.3	597.3	2247	2303	1881	1807
Min	3596	3596	3596	449.6	56.19	449.6	449.6	1285	1867	1279	1285
Q_1	4506	3685	3596	610.7	92.36	556.9	588.2	2065	2087	1881	1601
Med	4660	4062	3598	665	93.22	563.8	595.4	2227	2413	1921	1949
Q_3	4767	4299	3615	718.3	95.19	572.5	601.6	2495	2509	1952	1986
Max	6191	6191	6191	773.9	96.74	773.9	773.9	2634	2973	2107	2634
					`unsat-set-b-fclqcolor-10-07-09`						
Exp	2027	1446	1133	333.7	51.19	212.7	285.8	1656	1487	1149	1108
Min	1012	1012	1012	126.5	15.81	126.5	126.5	913.4	387	558.9	913.4
Q_1	1630	1346	1012	303.7	48.43	196.5	277.9	1362	1024	941.2	945.7
Med	2090	1481	1033	316.5	51.59	205.3	286.7	1536	1460	1259	1056
Q_3	2304	1553	1284	364.8	54.72	218.4	294.5	2050	1670	1385	1235
Max	3652	3652	3652	456.5	57.06	456.5	456.5	3347	4626	1631	3347

partitioning approach with 8 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 Section 5) is used as the partitioning function; first for plain partitioning with 8 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 an even better speedup for these instances. The results on all instances in Table 1 and two of the instances in Table 2 provide further evidence that in the actual implementations (columns T^{8}_{scatter} and T^{64}_{scatter}) plain partitioning often results in higher expected run-times than the simple distribution approach. This reflects the difficulty of obtaining ideal partitioning functions.

A comparison of the "ideal partitioning" approaches confirms the discussion in Section 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 speedup obtained by repeated partitioning when compared to safe partitioning.

7. Conclusions

The work investigates distributed techniques for solving challenging SAT instances and focuses on combining constraint-based search space partitioning with randomized SAT solving techniques. The work 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 analytical results show that the plain partitioning approach can potentially lead to catastrophic failures where an increase in computing resources leads to a decrease in solving efficiency for unsatisfiable instances.

The investigation is then extended to a setting where simple distribution and partitioning are mixed. The mixing results in two new approaches, the *safe* and the *repeated* partitioning. In the former approach, the instance is partitioned once and the solving of the resulting derived instances is attempted several times in parallel. In the latter approach, the instance is partitioned several times whereas the solving of the derived instances is attempted once. Analytical results show that in the model of partitioning employed, the safe partitioning approach has lower expected run time than the repeated partitioning approach. However, both approaches may suffer from the same catastrophic failures as the plain partitioning approach. Many of the analytical results are directly applicable to a wide range of search

procedures with properties similar to propositional satisfiability checking.

The significance of the analytical results is evaluated with two complementing experiments. To study the previously analyzed approaches, the work describes a randomized partitioning function called the scattering function. In the first experiment, the approaches are applied on all instances of the *applications* category of the 2009 SAT solver competition. The second experiment consists of a closer study on a smaller number of instances where the approaches are repeated several times to eliminate random effects from the results.

The empirical results show that in the plain partitioning approach a good implementation is usually able to avoid the catastrophic failure, but plain partitioning can nevertheless be worse than an approach based on simple distributed SAT solving (SDSAT). The results further show that both safe and repeated partitioning perform better than plain partitioning or SDSAT. Based on the experiments, the repeated partitioning approach can exploit the randomness in the partitioning function, and, in an apparent contradiction to the analytical results, yields in practise lower expected run times than safe partitioning. This results from the straightforward model of the partition function used in the analysis, and suggests that the use of more complicated models can help improve understanding of distributed SAT solving approaches.

Acknowledgments. The authors are grateful for the useful comments from the anonymous reviewers. The work has received 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, the Nokia Foundation and Technology Industries of Finland Centennial Foundation.

References

[1] Blochinger, W., Sinz, C., Küchlin, W.: Parallel Propositional Satisfiability Checking with Distributed Dynamic Learning, *Parallel Computing*, **29**(7), 2003, 969–994.

[2] Böhm, M., Speckenmeyer, E.: A Fast Parallel SAT-solver: Efficient workload balancing, *Annals of Mathematics and Artificial Intelligence*, **17**(4–3), 1996, 381–400.

[3] Bordeaux, L., Hamadi, Y., Samulowitz, H.: Experiments with Massively Parallel Constraint Solving, *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009.

[4] Chrabakh, W., Wolski, R.: GridSAT: a System for Solving Satisfiability Problems using a Computational Grid, *Parallel Computing*, **32**(9), 660–687.

[5] Chu, G., Stuckey, P. J.: *PMiniSat — A parallelization of MiniSat 2.0*, SAT Race 2008 system description, 2008, `http://baldur.iti.uka.de/sat-race-2008/descriptions/solver_32.pdf`.

[6] Dequen, G., Vander-Swalmen, P.: Toward Easy Parallel SAT Solving, *21st IEEE International Conference on Tools with Artificial Intelligence*, IEEE Computer Society, 2009.

[7] Eén, N., Sörensson, N.: An Extensible SAT-solver, *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003, Selected Revised Papers*, 2919, Springer, 2004.

[8] Gil, L., Flores, P., Silveira, L. M.: PMSat: a parallel version of MiniSAT, *Journal on Satisfiability, Boolean Modeling and Computation*, **6**, 2008, 71–98.

[9] Gomes, C. P., Selman, B.: Algorithm portfolios, *Artificial Intelligence*, **126**(1–2), 2001, 43–62.

[10] 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), 2000, 67–100.

[11] Grama, A., Kumar, V.: State of the Art in Parallel Search Techniques for Discrete Optimization Problems, *IEEE Transactions on Knowledge and Data Engineering*, **11**(1), 1999, 28–34.

[12] Hamadi, Y., Jabbour, S., Sais, L.: Control-based Clause Sharing in Parallel SAT Solving, *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009.

[13] Hamadi, Y., Jabbour, S., Sais, L.: ManySAT: a Parallel SAT Solver, *Journal on Satisfiability, Boolean Modeling and Computation*, **6**, 2009, 245 – 262.

[14] Huberman, B. A., Lukose, R. M., Hogg, T.: An Economics Approach to Hard Computational Problems, *Science*, **275**(5296), 1997, 51–54.

[15] Hyvärinen, A. E. J.: *Approaches to Grid-Based SAT Solving*, Research Report TKK-ICS-R16, Helsinki University of Technology, June 2009.

[16] Hyvärinen, A. E. J., Junttila, T., Niemelä, I.: A Distribution Method for Solving SAT in Grids, *9th International Conference on Theory and Applications of Satisfiability Testing*, 4121, Springer, 2006.

[17] Hyvärinen, A. E. J., Junttila, T., Niemelä, I.: Strategies for Solving SAT in Grids by randomized search, *Proceedings of the 9th International Conference on Artificial Intelligence and Symbolic Computation*, 5144, Springer, 2008.

[18] 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**, 2009, 223–244.

[19] Hyvärinen, A. E. J., Junttila, T., Niemelä, I.: Partitioning Search Spaces of a Randomized Search, *AI*IA 2009 International conference of the Italian Association for Artificial Intelligence*, 5883, Springer, 2009.

[20] Hyvärinen, A. E. J., Junttila, T., Niemelä, I.: Partitioning SAT Instances for Distributed Solving, *17th International Conference on Logic Programming, Artificial Intelligence, and Reasoning*, 6397, Springer, 2010.

[21] Inoue, K., Soh, T., Ueda, S., Sasaura, Y., Banbara, M., Tamura, N.: A competitive and cooperative approach to propositional satisfiability, *Discrete Applied Mathematics*, **154**(16), 2006, 2291–2306.

[22] Janakiram, V. K., Agrawal, D. P., Mehrotra, R.: A Randomized Parallel Backtracking Algorithm, *IEEE Transactions on Computers*, **37**(12), December 1988, 1665–1676.

[23] Janakiram, V. K., Gehringer, E. F., Agrawal, D. P., Mehrotra, R.: A Randomized Parallel Branch-and-Bound Algorithm, *International Journal of Parallel Programming*, **17**(3), 1988, 277–301.

[24] 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), 2005, 73–101.

[25] Lai, T.-H., Sahni, S.: Anomalies in Parallel Branch-and-Bound Algorithms, *Communications of the ACM*, **27**(6), 1984, 594 – 602.

[26] Lewis, M., Marin, P., Schubert, T., Narizzano, M., Becker, B., Giunchiglia, E.: PaQuBE: Distributed QBF Solving with Advanced Knowledge Sharing, *12th International Conference on Theory and Applications of Satisfiability Testing*, 5584, Springer, 2009.

[27] Li, G.-J., Wah, B. W.: Computational Efficiency of Parallel Combinatorial OR-Tree Searches, *IEEE Transactions on Software Engineering*, **16**(1), 1990, 13–31.

[28] Luby, M., Ertel, W.: Optimal Parallelization of Las Vegas Algorithms, *11th Annual Symposium on Theoretical Aspects of Computer Science*, 775, Springer, 1994.

[29] Luby, M., Sinclair, A., Zuckerman, D.: Optimal Speedup of Las Vegas Algorithms, *Information Processing Letters*, **47**(4), 1993, 173–180.

[30] Ohmura, K., Ueda, K.: c-sat: A Parallel SAT Solver for Clusters, *12th International Conference on Theory and Applications of Satisfiability Testing*, 5584, Springer, 2009.

[31] Prestwich, S., Mudambi, S.: Improved Branch and Bound in Constraint Logic Programming, *First International Conference on Principles and Practice of Constraint Programming*, 976, Springer, 1995.

[32] Rao, V. N., Kumar, V.: Superlinear Speedup in Parallel State-Space Search, *8th ARCS Conference on Foundations of Software Technology and Theoretical Computer Science*, 338, Springer, 1988.

[33] Rao, V. N., Kumar, V.: On the Efficiency of Parallel Backtracking, *IEEE Transactions on Parallel and Distributed Systems*, **4**(4), 1993, 427 – 437.

[34] Schubert, T., Lewis, M., Becker, B.: PaMiraXT: Parallel SAT Solving with Threads and Message Passing, *Journal on Satisfiability, Boolean Modeling and Computation*, **6**, 2009, 203–222.

[35] Segre, A. M., Forman, S. L., Resta, G., Wildenberg, A.: Nagging: A scalable fault-tolerant paradigm for distributed search, *Artificial Intelligence*, **140**(1/2), 2002, 71–106.

[36] Speckenmeyer, E., Monien, B., Vornberger, O.: Superlinear Speedup for Parallel Backtracking, *1st International Conference on Supercomputing*, 297, Springer, 1998.

[37] Véron, A., Schuerman, K., Reeve, M., Li, L.-L.: Why and How in the ElipSys OR-parallel CLP System, *Proceedings of the 5th International PARLE Conference*, 694, Springer, 1993.

[38] Zhang, H., Bonacina, M., Hsiang, J.: PSATO: A Distributed Propositional Prover and its Application to Quasigroup Problems, *Journal of Symbolic Computation*, **21**(4), 1996, 543–560.