University of British Columbia
Department of Computer Science
Technical Report TR-2015-??

Sparkle: A PbO-based Multi-agent Problem-solving Platform

Holger H. Hoos
University of British Columbia
Department of Computer Science

24 December 2015

Abstract

We outline Sparkle, a platform for solving computationally challenging problems such as SAT. Sparkle is based on the idea that, in order to solve such problems most effectively, solver developers should be incentivised to focus their energy on work that maximally improves the state of the art, fully taking advantage of meta-algorithmic approaches such as algorithm selectors, schedulers and portfolios, as well as of high-performance, general-purpose algorithm configurators. We argue that based on existing work from the literature and readily available tools, simple versions of Sparkle can be implemented quite easily and used in various settings, including solver competitions and commercial applications. These simple versions can be extended in various ways, which further enhance the usefulness of the platform.

1 Introduction

Over the past decade, our ability to solve well-studied computationally challenging problems has increased substantially, as has the importance of high-performance solvers for these problems in the context of real-world applications. This can be seen, for example, in the case of the propositional satisfiability problem (SAT), one of the most prominent $\mathcal{NP}$-complete combinatorial decision problems. Although in the following, we focus on SAT, we note that our observations and the multi-agend problem solving system motivated by these are in no way restricted to SAT.

The work presented here is motivated by five insights:

Insight 1: The state of the art for solving SAT (and almost all other prominent computational problems whose effective solution requires carefully designed heuristics) is not defined by a single solver, but by a set of several non-dominated solvers with complementary strengths. In this context, a solver A is dominated if for every problem instance i of interest, there is another solver A' that performs better than A on i.

Insight 2: The existence of several non-dominated solvers can be leveraged by algorithm selection, scheduling and parallel portfolio methods. In the case of SAT, such meta-algorithmic design patterns have been demonstrated to yield excellent performance, *e.g.*, by the well-known SATzilla algorithm selection system (Xu et al., 2008, 2012).

Insight 3: State-of-the-art SAT solvers contain many design choices that can be exploited to optimise performance for particular sets of SAT instances (see, *e.g.*, Hutter et al., 2007, 2015); this is particularly the case for solvers developed based on the Programming by Optimisation (PbO) paradigm (see, *e.g.*, KhudaBukhsh et al., 2015).

Insight 4: Using automated design optimisation methods – particularly, cutting-edge algorithm configuration procedures such as SMAC (Hutter et al., 2011) – state-of-the-art solvers for specific types of SAT instances can be obtained, and these can be leveraged using meta-algorithmic design patterns such as per-instance algorithm selection (see, *e.g.*, Xu et al., 2010).

Insight 5: To advance our ability to effectively solve SAT (and similarly challenging problems), solver designers should be incentivised to focus their energy on work that maximally improves the state of the art, fully taking advantage of meta-algorithmic design patterns (see Insight 2) and automated design optimisation (see Insight 4).

Insight 5 implicitly underlies all work on portfolio-based SAT solvers, and the core idea was perhaps first stated explicitly by Xu et al. (2012). Unfortunately, despite the well-documented success of portfolio-based SAT solvers, the SAT community has not yet embraced this idea; on the contrary, in recent years, there have been modifications to the SAT solver competitions that effectively reduce the incentives to work on portfolio-based SAT solvers and specialised solvers that may contribute most to them and hence to the state of the art in SAT solving, in order to highlight monolithic solvers that achieve improved performance over broad and diverse sets of benchmark instances.

Therefore, in this work, we address the question how to achieve the incentivisation called for in Insight 5. In a nutshell, our answer to this question is to make it easy and rewarding to improve the state of the art, fully taking advantage of cutting-edge meta-algorithmic approaches, such as automated algorithm selection, scheduling and parallel portfolios, as well as automated design optimisation. To make this concrete, we outline our vision for *Sparkle*, a PbO-based multi-agent platform for SAT solving. In essence, clients submit instances to be solved to Sparkle, and solver providers submit solvers; Sparkle generates a per-instance selector based on all solvers available and uses this to solve instances, giving credit to each provider whose solver has successfully solved an instance. Of course, Sparkle is not restricted to SAT, but can be applied analogously to the many computational problems for which our five insights hold.

2 Sparkle for Parameter-less Solvers

We first consider a simplified case, in which the performance parameters for all solvers are fixed at a predetermined setting. (Performance parameters are those whose value affect only the performance of a given solver.) At the conceptual level, the Sparkle platform then comprises the following components:

- a collection S of SAT solvers
- a collection I of SAT instances and reference results for those instances

- a collection E of feature extractors
- feature data F computed using the feature extractors in E for the instances in I
- performance data (here: running times) P for the solvers in S on the instances in I
- a procedure C for constructing a portfolio-based selector R based on a subset of S, optimised for performance on a subset of I, using feature data from F and performance data from P
- a procedure N for computing the contributions of any solver $s \in S$ to a given portfolio-based selector R when solving a given instance i

We assume that the portfolio-based selector follows the SATzilla design and therefore uses pre- and backup solvers in addition to an algorithm selector based on a supervised machine learning procedure (see, *e.g.*, Mohri et al., 2012). (Evidently, other portfolio-based selectors and selector construction methods, as well as scheduling and parallel portfolio construction methods can be used instead of SATzilla.) Performance can be assessed and optimised using various metrics, such as PAR-10, the penalised average running time with a penalty factor 10, which averages running time over a given set of instances, counting each timed-out run as 10 times the given cutoff time.

In addition, we need to provide:

- a mechanism for performing runs of solvers from S, of feature extractors from E, of the portfolio-based selector R, of the construction procedure C and of the contribution analysis procedure N
- a user interface (UI) that makes it easy to add solvers to S, to remove solvers from S and to update solvers in S; to submit an instance i to be solved; and to access performance and contribution data for solvers in S

The simplest version of Sparkle then works as follows:

- To initialise the platform, a platform operator seeds I with a collection of SAT instances suitable for training selectors (*e.g.*, taken from past SAT competitions), installs a collection E of feature extractors and triggers the computation of the feature data F. The operator also adds one or more reference solvers to S and adds performance data for these, obtained by running them on all instances I with a fixed cutoff time t_{max}, to P. Finally, the operator runs the construction procedure C to obtain an initial portfolio-based selector R.
- When a new instance i is to be solved, its features are computed using the feature extractors in E; the resulting feature vector is then passed to the portfolio-based selector R, which runs one or more solvers from S. During and/or after this process, procedure N is used to keep track of the contributions made by each solver.
- When a solver provider submits a new solver s, s is added to S and run on all instances in I with cutoff time t_{max}; the resulting performance data are added to P and made available to the submitter (*e.g.*, in the form of a CSV file sent via e-mail). If the result for any $i \in I$ disagrees with the reference result for this instances, s is rejected as incorrect, and the instance i along with the output of S on i is provided to the submitter (*e.g.*, via e-mail). If s is not rejected, C is run to obtain a new portfolio-based selector R that may utilise s.

- When a solver provider withdraws a solver s, s is removed from S and the performance data for s is removed from P. Then C is run to obtain a new portfolio-based selector R that no longer uses s.
- When a solver provider updates a solver s, the old version from s is removed from S, and its performance data is removed from P. Then, the new version of S is added, and the same steps are carried out as for a newly submitted solver.
- The contributions of solvers can be checked at any time by their submitters and platform operators, including detailed contributions on all instances solved by the system as well as aggregate contribution data (in particular: total contribution per solver).

We note that this simple version of Sparkle can be implemented easily based on an existing implementation of SATzilla (Xu et al., 2012). Feature extraction, solver runs and portfolio-based selector construction could happen on a single, powerful machine, on a cluster or using cloud computing infrastructure, primarily depending on the difficulty and number of instances to be solved. If solvers are submitted, modified or retracted frequently enough that constructing a new portfolio-based selector R every time the solver set changes becomes computationally too expensive, construction of a new R can be restricted, *e.g.*, to at most once per hour. Of course, construction of a new R (and the prerequisite collection of performance data for new solvers) can be carried out concurrently with the use of the current portfolio-based selector for solving incoming problem instances, using different processors or processor cores. In case a newly submitted solver s is found to be dominated by the solvers already in S, *i.e.*, on every instance in I to be slower than some $s' \in S$, s can be rejected, saving the construction of a new portfolio-based selector.

Solver contributions can be calculated in various ways, ranging from (1) simple tallying of the running times of solvers that succeeded in solving a given instance i to (2) marginal contribution (*i.e.*, cost of omission) (Xu et al., 2012) and (3) Shapley value (Fréchette et al., 2016). Method 1 is trivial to implement and requires only negligible additional computation. Additionally, it is very intuitive and easy to understand for solver providers. However, as argued by Xu et al. (2012), this approach may give substantial credit to a solver A that performs well, but whose capability is almost matched by one or more other solvers that, were A not available, could be run instead. Contrasting this with another solver, B, that has a more unique contribution to overall portfolio performance, in the sense that if it were not available, there would be no other solvers to compensate, it becomes clear that Method 1 has a serious weakness. To see this, notice that B may be called much less frequently than A, and therefore get substantially less credit than A. At the same time, omitting B from solver set S leads to a larger loss in performance than omitting A, rendering B more valuable to the portfolio represented by S.

Method 2 addresses this problem by defining the contribution as the drop in performance caused by removing a solver from a given portfolio (Xu et al., 2012). To compute this accurately for a given solver A, we have to construct a portfolio that does not use A and assess its performance. Fortunately, under the assumption that portfolio performance is generally close to the performance of a perfect portfolio-based selector, *i.e.*, one that always selects for a given SAT instance the solver from S that performs best on it, we can estimate the marginal contribution based on perfect selectors with and without A, and the performance of such perfect selectors can be calculated very efficiently from the performance data P. Furthermore, the assumption is known to be reasonable for state-of-the-art selector construction methods (see, *e.g.*, Kotthoff, 2015; Xu et al., 2012). However, Method 2 still has a weakness, as pointed out by Fréchette et al. (2016): If we include two copies of A (or, more realistically, two solvers with almost identical performance) in S, their marginal contribution is (close to) zero.

Method 3 addresses this problem by defining solver contribution using a fundamental concept from coalitional game theory: the Shapley value (Shapley, 1953). In a nutshell, the Shapley value is an aggregate

measure over the marginal contribution of a solver from portfolios based on arbitrary subsets of S. Consequently, it does not assign contributions close to zero to solvers with very similar performance on the given set of instances. While the Shapley value has desirable theoretical properties, it is generally costly to compute. As shown by Fréchette et al. (2016), by compactly representing the marginal contribution for the given solvers, their Shapley values can be computed efficiently, although the computational cost can still be considerable for large sets of solvers.

Using any of the three methods for calculating solver contributions, there is a clear incentive for solver developers to focus their efforts improving the state of the art in SAT solving, as represented by a cutting-edge portfolio-based algorithm selector such as SATzilla. The Sparkle platform operationalises this incentive by constructing the portfolio-based selector, tracking solver contributions and making detailed information on the performance and contribution of their solvers available to solver developers; it also provides a fair and, if deployed in the right way, trusted way to assess solver contributions. The use of Sparkle can be made even easier by making available to solver developers a local testing harness that permits them to check their solver for compliance with Sparkle prior to submission, in terms of input and output format as well as in terms of the way in which it is called from the command line.

3 Sparkle for Parameterised Solvers

The version of Sparkle considered in the previous section exploited Insight 2 – the fact that there are effective algorithm selection, scheduling and parallel portfolio methods that can be used to leverage complementary strengths of non-dominated solvers. In practice, many of these solvers are (or easily can be) parameterised, such that they can be configured for optimised performance on different types of SAT instances. The degree to which the performance of state-of-the-art SAT solvers can be optimised for specific types of instances has been demonstrated recently in the Configurable SAT Solver Competitions (CSSC) (Hutter et al., 2015). This motivated Insight 4 from the introductory section, and to exploit the configurability of SAT solvers, we now consider an extension of the basic version of Sparkle discussed earlier.

In addition to the components introduced in the previous section, this extended version of Sparkle also comprises the following:

- configuration spaces for some (or all) of the solvers in S, where a configuration space consists of a list of parameters, a domain of possible values for each such parameter, a (possibly empty) set of conditional parameter dependencies and a (possibly) empty set of constraints on combinations of parameter values
- an automated algorithm configuration procedure O for optimising the performance of a given solver $A \in S$ on a subset of I, using feature data from F and performance data from P

In addition, we need to provide:

- a mechanism for performing runs of the automatic configuration procedure O
- user interface (UI) affordances that make it easy to specify configuration spaces for solvers in S, as well as to modify and remove such specifications; to specify the subsets of instance set I to be used for a run of the configuration procedure O on a specific solver $A \in S$, to launch such a run and to assess the performance of the configurations of A thus obtained; and to add to, remove from or replace within S the solver configurations thus obtained.

In this version Sparkle, solver submitters or platform operators can specify configuration spaces for parameterised solvers and use the configuration procedure O provided by the platform to optimise solver performance for specific types of SAT instances characterised by sets of instances available on Sparkle (as part of instance set I). This makes it possible to configure solvers for overall performance across broad sets of instances, as well as, more importantly, for specific types of instances – the latter makes it even easier for solver developers to achieve contributions to the state-of-the-art in SAT solving by determining and then submitting configurations of their solvers that boost the performance of the portfolio-based selector constructed by Sparkle.

To make it even easier for solver developers to maximise the impact of their solvers, Sparkle could permit them to use the configuration procedure O to optimise the marginal contribution of a given solver A to the portfolio-based selector constructed by Sparkle. The mechanism for doing this is the same as used at the core of the Hydra portfolio-based selector construction procedure (Xu et al., 2010, 2011), and, as in Hydra, several complementary configurations of a given solver A could be determined by running multiple iterations of O, where one or more configurations of A are added to S at the end of each iteration. Within Sparkle, new configurations are determined in the context of a set of separate and possibly very different solvers, rather than from a single parameterised solver, as in the work of Xu et al. (2010) and (Xu et al., 2011); however, the mechanism used for configuring a solver to complement an existing portfolio remains the same.

The configurator O can be any state-of-the-art general-purpose algorithm configuration procedures, such as SMAC (Hutter et al., 2011), and to specify configuration spaces, the pcs-format supported by SMAC and ParamILS (Hutter and Ramage, 2015) can be used. To further facilitate the use of the configurator within Sparkle, a testing harness can be supplied to solver developers, which lets them check the compliance of their solver and configuration space specification with the requirements of Sparkle's configuration mechanism.

Evidently, configuring a solver can incur a substantial computational cost. Depending on the way Sparkle is used (*e.g.*, for a competition or in a commercial context), the amount of computation solver submitters can use for configuration within Sparkle can be limited or commoditised.

4 A SAT Solver Competition based on Sparkle

For more than a decade, the SAT solver competition series has been a showcase and a driver for progress in SAT solving. However, while portfolio-based methods have participated and achieved considerable success, leveraging the complementary strengths of a large and diverse set of solvers that contribute to the state of the art in SAT solving, the focus of much of the SAT community has remained on producing monolithic solvers that perform well on different kinds of instances within each of the three main categories of the competition series (random, hand-made or crafted, and application or industrial instances). It has been pointed out in the literature that there is much to be gained by recognising the contribution of SAT solvers to state-of-the-art portfolio-based methods (see, *e.g.*, Xu et al., 2012; Fréchette et al., 2016).

Using the proposed Sparkle platform, which was in no small part motivated by these considerations, it would be rather straightforward to run a future SAT competition, or a track of such a competition, in a way that makes it easy and rewarding to improve the state of the art in SAT solving, fully taking advantage of cutting-edge meta-algorithmic approaches. The competition (or track) organisers would set up and operate the platform, which also contains two sets of benchmark instances: A public set (assembled from instances

used in previous competitions) available during solver development and testing, and a private set, used for assessing solver performance in the actual competition. Solver contributors obtain access to the platform, along with a fixed configuration and evaluation budget per team, which they can use to configure their solver on arbitrary parts of the public instance set, using one or more automated algorithm configurators integrated into the Sparkle competition platform. They also obtain detailed data on the performance of their solver from evaluation runs on individual instances that they can use for further development and for manual or automatic configuration of their solver.

To focus the total computational budget available for the competition on the most promising submissions, solvers could be evaluated in a first stage on a validation set (which could be a subset of the instances subsequently used as the public instance set), and only solvers that reach a set performance threshold in this stage would proceed into the main competition and be rewarded a fixed budget for configuration and evaluation. Solvers that produce incorrect results would also be disqualified at this stage.

The evaluation on the private competition set is then conducted as follows: Using the set of all (configured) solvers (where the number of solvers and solver configurations per team may be restricted) and the public instance set as training data, a portfolio-based SAT solver, P, is produced, using a state-of-the-art construction procedure, such as SATzilla (Xu et al., 2008, 2012) or AutoFolio (Lindauer et al., 2015). The contribution of each component solver to the performance of P on the private set of competition instances is then determined, and awards are given according to a robust ranking (Fawcett et al., 2015). Component solver contribution could be assessed using marginal contribution (Xu et al., 2012) or according to Shapley value (Fréchette et al., 2016); we see advantages in both approaches but note that the former captures the contribution to the actual set available for portfolio construction and might therefore be preferable.

Notice that using this competition design, developers of solvers that show substantially improved performance over the current state of the art over a broad subset of the competition instances and may have therefore won in the traditional competition setting will still be recognised. Therefore, the incentive for trying to achieve such fundamental improvements will not be lost. At the same time, however, the Sparkle competition setting formally recognises narrower contributions to the state of the art, characterised by substantial performance advantages over all other solvers on specific types or families of SAT instances, and thus provides an incentive for working on such solvers.

Furthermore, the proposed competition design encourages solver developers to leverage state-of-the-art automated configuration procedures, by making it easy to use such configurators and to critically assess the effects thus obtained. At the same time, automated configuration within the Sparkle competition platform is made available in a uniform and fair way to all competitors, while leaving more room for human judgement than in the Configurable SAT Solver Competitions (Hutter et al., 2015); in particular, our design allows each team to choose their own subset of training instances, and to divide their computation time budget between multiple rounds of configuration and evaluation.

5 Commercial SAT Solving using Sparkle

The propositional satisfiability problem (like many other $\mathcal{NP}$-hard problems) is of substantial commercial interest; in particular, SAT solvers are used routinely for hardware- and software-verification (see, *e.g.*, Prasad et al., 2005) as well as for diagnosis (see, *e.g.*, Grastien et al., 2007), planning and scheduling (see, *e.g.*, Kautz and Selman, 1999; Béjar and Manya, 2000). SAT solvers also play a key role in the burgeoning area of software synthesis (see, *e.g.*, Srivastava et al., 2013). In many cases, solver performance is of

considerable importance, as it defines the range of problems that can be tackled as well as the efficiency of the processes in which a SAT solver is used and the quality of their outcome. We expect the trend to use SAT solvers for tackling suitably encoded problems from other areas not only to continue, but to accelerate, as AI technology becomes more pervasive and our ability of solving larger and harder SAT instances further improves. Therefore, we see much promise in using Sparkle in the context of commercial SAT solving.

Here, we consider a business model where a service provider operates a Sparkle platform, to which customers submit SAT instances to be solved, along with a price $p(F)$ they are willing to pay for the solution of any given instance F (*i.e.*, a model satisfying F or a proof that F is unsatisfiable). It would be possible for the service provider to provide a quote $q(F)$ for the price of solving F, based on empirical performance models (Hutter et al., 2014b); this quote could be used as guidance by the customer when determining the price $p(F)$. Customers may also pay a (small) base fee for submitting an instance to be solved to help cover the service provider's expenses caused by unsolved instances. This fee could be charged on a per-instance basis or folded into a subscription model. Solver providers submit their solver and pay a monthly fee f for the use of the platform; they also receive credit $c(F) < p(F)$ whenever their solver is selected to solve an instance F and solves F successfully. The difference $p(F) - c(F) > 0$ is used to cover the operating expenses of and to generate profit for the service provider. Solver providers are also given access to information collected and compiled by the Sparkle platform that helps them understand and improve their solver's performance, and thereby earnings. Such information can include data on solver runs (successful and unsuccesful), processed to ensure confidentiality of the instances provided by clients using suitable summarisation and scrambling techniques; detailed performance data on benchmark sets provided by the service provider; empirical performance models and performance predictions produced by these; and contribution to the state-of-the-art portfolio-based solver generated and run by Sparkle to solve customer instances. Some of these data might be included in the platform access fee, others may be offered for an additional fee.

Credit for solving instances could potentially be distributed to multiple solvers, using marginal contribution, Shapley value or similar concepts to quantify contributions. Such schemes would be less intuitive for solver providers but could help mitigate risk by the service provider. (To see this, notice that having multiple solvers with similar performance represents an advantage for the service provider, since it reduces the drop in overall performance – and hence, revenue – caused by withdrawal of one solver.) Unfortunately, they incur considerable overhead by running multiple solvers on customer instances. This drawback could be addressed by using performance predictions obtained from empirical performance models in lieu of actual observed performance, or by means of a credit distribution scheme partially based on performance contributions determined on the training set of instances used for selector construction.

If the platform offers configuration services, solver providers pay a fee for their use; this fee depends on the desired configuration budget and is calculated to cover the cost incurred for the service provider and, potentially, to generate some profit. The solver provider selects a set of training instances used for configuration (some of these could be provided via Sparkle, others uploaded by the solver provider) along with all other information required by the configuration procedure, and receives one or more optimised configurations, along with detailed validation performance results and, perhaps, functional ANOVA results on parameter importance (Hutter et al., 2014a). The service operator can carry out configuration using a state-of-the-art automated algorithm configuration procedure, such as SMAC (Hutter et al., 2011).

The service provider can run Sparkle on a dedicated cluster or using commodity cloud computing services; in both cases, it is advantageous to run all system components, including solvers, on virtualised hardware, in order to make it easier to provide a carefully designed execution environment and software stack, and to facilitate scaling of the service.

6 Extensions

The Sparkle platform outlined so far can be extended in many directions. Multiple construction procedures for portfolio-based algorithm selectors could be used in combination with cross-validation to determine the best resulting selector. The selector construction procedure could be configured, using a system like AutoFolio (Lindauer et al., 2015). Rather than constructing a selector for a single solver, we could build a parallel portfolio (see, *e.g.*, Lindauer et al., 2016; Gomes and Selman, 2001; Huberman et al., 1997) or a per-instance selector for a parallel portfolio (Xu et al., 2010, 2011; Kadioglu et al., 2010), thus exploiting parallel computing resources to solve SAT instances submitted to Sparkle faster. More than one configurator could be integrated into Sparkle and made available to solver developers. Selector construction and configuration procedures could be contributed and, depending on the contributions achieved by their use to overall problem solving performance, attributed credit analogously to the solvers in S.

For the commercial version of Sparkle, solver providers could charge different fees per time unit for using their solver. The solver selection process would then take this differential pricing into account, aiming to minimise the cost for the client. Solver providers may be given the option to submit open- or closed-source (binary) versions of a solver; open-source solvers would then obtain more revenue, but also be available to other solver developers to modify, provided the resulting solvers stay open-source. The revenue generated by such derived solvers would be shared between all developers whose work forms part of their lineage, as tracked by Sparkle (possibly aided by source code analysis).

Customers of the commercial version of Sparkle could be offered bundle pricing or tiered prices, depending on the time required for solving a given SAT instance. Similar arrangements could be offered to solver developers for the use of Sparkle's automatic algorithm configuration facilities. Use fees for solver providers could be annually, life-time or per solver update (the last of these options reflects the cost incurred for constructing a new portfolio-based selector). Discounts could be offered for solving instances that can be integrated into the instance set used within Sparkle for constructing selectors and configuring solvers. Solver providers may allow service operators to configure their solvers to obtain better portfolio-based selectors; the increase in revenue thus produced would be shared between the solver provider and service provider.

Finally, the service provider might act as match maker between solver providers and clients who want to run private, local copies of a solver; using solver performance data from the Sparkle platform, the service provider could determine which solvers are of interest to a given customers and advertise those solvers to them. The service provider could also provide an execution environment and software harness (VM snapshot) to the customer that makes it easy to use the private copy of the solver just as they would have used the public Sparkle platform. In exchange, the service provider would receive a share of the proceeds from each licensing deal facilitated in this way. Of course, such deals could be extended to private copies of a Sparkle platform using more than one solver. In this case, fractions of the licensing revenue would be passed on to the solver providers, *e.g.*, based on the marginal contribution (*i.e.*, cost of omission) of that solver on a suitably selected set of test instances.

7 Conclusions

We have described a problem solving platform called Sparkle that is strongly based on the concept of programming by optimisation (PbO). Specifically, Sparkle leverages state-of-the-art algorithm selection, scheduling and portfolio techniques to exploit complementary strengths found within sets of solvers for a

given problem. Sparkle also facilitates the use of cutting-edge automatic algorithm configuration techniques for finding within large design spaces of highly parameterised solvers configurations that contribute most to a given set of solvers and solver configurations. Sparkle is a multi-agent platform, in that each solver and solver configuration can be seen as an agent that, once submitted to Sparkle and integrated into its portfolio of solvers, can potentially create value by helping Sparkle solve incoming problem instances more effectively. The key idea behind Sparkle is to create incentives for solver developers to focus on improvements on the state-of-the-art in solving a given problem, as represented by effective automatic selection, scheduling or parallel portfolio mechanisms in conjunction with a potentially large number of solvers and solver configurations with complementary strengths.

As we have argued, using the example of the prominent propositional satisfiability (SAT) problem, Sparkle can provide an attractive basis for solver competitions as well as for a commercial service. Of course, Sparkle is not limited in any way to SAT solving, but can be applied to any problem for which there is at least one parametric solver. It will be most effective for problems for which the state of the art comprises several rather different solvers or solver configurations, as is the case for SAT, mixed integer programming, TSP, AI planning, machine learning and many other important problems. Sparkle has not yet been implemented, but the vision outlined here can be realised in a straightforward way, leveraging recent progress in the meta-algorithmic design techniques that also enable programming by optimisation (PbO).

Sparkle is conceptually related to crowd-sourced problem solving platforms such as Foldit (Cooper et al., 2010), in that it leverages the strength of multiple, complementary approaches for solving a given problem. Unlike Foldit, Sparkle strongly exploits meta-algorithmic techniques. On the other hand, Sparkle, as outlined here, provides only limited incentives for collaboration between solvers and solver developers beyond that facilitated by the meta-algorithmic procedure, such as SATzilla, that controls the use of solvers for tackling specific problem instances. Extending Sparkle to facilitate other forms of cooperation between solvers or solver developers is an interesting avenue for future work.

Acknowledgements: Some of the ideas discussed in this document have their roots in joint work and discussions with Frank Hutter, Chris Fawcett and Kevin Leyton-Brown.

References

Béjar, R. and Manya, F. (2000). Solving the round robin problem using propositional logic. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00)*, pages 262–266.

Cooper, S., Khatib, F., Treuille, A., Barbero, J., Lee, J., Beenen, M., Leaver-Fay, A., Baker, D., Popović, Z., et al. (2010). Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760.

Fawcett, C., Hoos, H. H., Vallati, M., and Gerevini, A. E. (2015). What competition results really mean – ranking solvers using statistical resampling. *Manuscript in preparation*.

Fréchette, A., Kotthoff, L., Rahwan, T., Hoos, H. H., Leyton-Brown, K., and Michalak, T. (2016). Using the shapley value to analyze algorithm portfolios. In *30th AAAI Conference on Artificial Intelligence (AAAI-16)*. To appear (7 double-column pages).

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

Grastien, A., Anbulagan, A., Rintanen, J., and Kelareva, E. (2007). Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd National Conference on Artificial Intelligence – Volume 1*, AAAI'07, pages 305–310. AAAI Press.

Huberman, B., Lukose, R., and Hogg, T. (1997). An economics approach to hard computational problems. *Science*, 265:51–54.

Hutter, F., Babić, D., Hoos, H. H., and Hu, A. J. (2007). Boosting verification by automatic tuning of decision procedures. In *Proceedings of Formal Methods in Computer Aided Design (FMCAD'07)*, pages 27–34.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pages 507–523. Springer Berlin Heidelberg.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2014a). An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pages 754–762.

Hutter, F., Lindauer, M. T., Balint, A., Bayless, S., Hoos, H. H., and Leyton-Brown, K. (2015). The configurable SAT solver challenge (CSSC). *Artificial Intelligence*, accepted for publication.

Hutter, F. and Ramage, S. (2015). Manual for SMAC version v2.10.03-master. Available on-line at: `http://www.cs.ubc.ca/labs/beta/Projects/SMAC/v2.10.03/manual.pdf`.

Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014b). Algorithm runtime prediction: Methods & applications. *Artificial Intelligence*, 206:79–111.

Kadioglu, S., Malitsky, Y., Sellmann, M., and Tierney, K. (2010). ISAC – instance-specific algorithm configuration. In *Proc. 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 751–756.

Kautz, H. A. and Selman, B. (1999). Unifying SAT-based and graph-based planning. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 318–325.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H., and Leyton-Brown:, K. (2015). SATenstein: Automatically building local search sat solvers from components. *Artificial Intelligence*, accepted for publication.

Kotthoff, L. (2015). On algorithm selection, with an application to combinatorial search problems. *Constraints*, 20(4):481–482.

Lindauer, M., Hoos, H., Hutter, F., and Schaub, T. (2015). Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778.

Lindauer, M., Hoos, H. H., Leyton-Brown, K., and Schaub, T. (2016). Automatic construction of parallel portfolios via algorithm configuration. *Artificial Intelligence*, accepted for publication.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press.

Prasad, M. R., Biere, A., and Gupta, A. (2005). A survey of recent advances in SAT-based formal verification. *International Journal on Software Tools for Technology Transfer*, 7(2):156–173.

Shapley, L. S. (1953). A value for n-person games. In *In Contributions to the Theory of Games, volume II*, pages 307–317. Princeton University Press.

Srivastava, S., Gulwani, S., and Foster, J. S. (2013). Template-based program verification and program synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5-6):497–518.

Xu, L., Hoos, H., and Leyton-Brown, K. (2010). Hydra: Automatically configuring algorithms for portfolio-based selection. *AAAI*, 10:210–216.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pages 16–30.

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT 2012)*, LNCS 7317, pages 228–241.