On the Combination of Argumentation Solvers into Parallel Portfolios

Mauro Vallati[1], Federico Cerutti[2], and Massimiliano Giacomin[3]

[1] University of Huddersfield, United Kingdom
[2] Cardiff University, United Kingdom
[3] Università degli Studi di Brescia, Italy

Abstract. In the light of the increasing interest in efficient algorithms for solving abstract argumentation problems and the pervasive availability of multicore machines, a natural research issue is to combine existing argumentation solvers into parallel portfolios. In this work, we introduce six methodologies for the automatic configuration of parallel portfolios of argumentation solvers for enumerating the preferred extensions of a given framework. In particular, four methodologies aim at combining solvers in static portfolios, while two methodologies are designed for the dynamic configuration of parallel portfolios. Our empirical results demonstrate that the configuration of parallel portfolios is a fruitful way for exploiting multicore machines, and that the presented approaches outperform the state of the art of parallel argumentation solvers.

Keywords: Argumentation Reasoning, Parallel Computing, Algorithm Selection

1 Introduction

Dung's theory of abstract argumentation [11] is a unifying framework able to encompass a large variety of specific formalisms in the areas of nonmonotonic reasoning, logic programming and computational argumentation. It is based on the notion of argumentation framework (AF), consisting of a set of *arguments* and a binary *attack* relation between them. Arguments can thus be represented by nodes of a directed graph, and attacks by arcs. The nature of arguments is left unspecified: it can be anything from logical statements to informal natural language text. For instance, [21] shows how argumentation can be efficiently used for supporting critical thinking and intelligence analysis in military-sensitive contexts.

Different *argumentation semantics* declare the criteria to determine which arguments emerge as "justified" among conflicting ones, by identifying a number of *extensions*, i.e. sets of arguments that can "survive the conflict together". In [11] four "traditional" semantics were introduced, namely *complete*, *grounded*, *stable*, and *preferred* semantics. For a complete overview of subsequently proposed alternative semantics, the interested reader is referred to [3].

The main computational problems in abstract argumentation include *decision*—e.g. determine if an argument is in all the extensions prescribed by a semantics—and *construction* problems, and turn out to be computationally intractable for most argumentation semantics [12]. In this paper we focus on the *extension enumeration* problem, i.e. constructing *all* extensions for a given AF: its solution provides complete information about the justification status of arguments and allows for solving the other problems as well.

Nowadays, increases in computational power are mostly achieved through hardware parallelisation. Almost every machine on the market is equipped with a multicore CPU, therefore, parallel solvers are gaining importance in many areas of Artificial Intelligence. However, the manual constructions of parallel solvers is a very challenging task, as it often requires to design specific algorithms, rather than adapting existing sequential ones. One promising approach for exploiting the computational power provided by multicore machines is then to combine solvers into parallel portfolios, which have been recently introduced in areas such as SAT and ASP [1, 17]. Notably, while work has been done in the area of sequential portfolios for argumentation [10], there is a lack of approaches aiming at combining solvers into parallel portfolios.

In this work, we consider the automatic construction of static and dynamic portfolios of argumentation solvers for enumerating the preferred extensions of a given AF. In particular, we introduce four methodologies for configuring static portfolios, and two for the dynamic selection of solvers to be executed in parallel. The designed techniques are general, in the sense that they are able to configure portfolios for any given number of available cores—here we focus on the 4-cores case, which correspond to most widely available machines. Our extensive experimental analysis demonstrates that: (i) combining argumentation solvers in parallel portfolios is an effective way for exploiting multiple cores; (ii) static portfolios that execute more solvers on each core are extremely robust; and (iii) the configured parallel portfolios outperform state-of-the-art native parallel argumentation solvers.

2 Dung's Argumentation Framework

An argumentation framework [11] consists of a set of arguments and a binary attack relation between them.

Definition 1. *An* argumentation framework *(AF) is a pair* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ *where* $\mathcal{A}$ *is a set of arguments and* $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$*. We say that* $\boldsymbol{b}$ attacks $\boldsymbol{a}$ *iff* $\langle \boldsymbol{b}, \boldsymbol{a} \rangle \in \mathcal{R}$*, also denoted as* $\boldsymbol{b} \rightarrow \boldsymbol{a}$*.*

The basic properties of conflict–freeness, acceptability, and admissibility of a set of arguments are fundamental for the definition of argumentation semantics.

Definition 2. *Given an AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*:*

- *a set* $S \subseteq \mathcal{A}$ *is a* conflict–free *set of* Γ *if* $\nexists\ \boldsymbol{a}, \boldsymbol{b} \in S$ *s.t.* $\boldsymbol{a} \rightarrow \boldsymbol{b}$*;*

- *an argument* $\boldsymbol{a} \in \mathcal{A}$ *is* acceptable *with respect to a set* $S \subseteq \mathcal{A}$ *of* Γ *if* $\forall \boldsymbol{b} \in \mathcal{A}$ *s.t.* $\boldsymbol{b} \rightarrow \boldsymbol{a}$, $\exists\ \boldsymbol{c} \in S$ *s.t.* $\boldsymbol{c} \rightarrow \boldsymbol{b}$;
- *a set* $S \subseteq \mathcal{A}$ *is an* admissible set *of* Γ *if* S *is a conflict–free set of* Γ *and every element of* S *is acceptable with respect to* S *of* Γ.

An argumentation semantics σ prescribes for any AF Γ a set of *extensions*, denoted as $\mathcal{E}_\sigma(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by σ. Here we need to recall the definition of preferred (denoted as PR) semantics only.

Definition 3. *Given an* AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*: a set* $S \subseteq \mathcal{A}$ *is a* preferred extension *of* Γ*, i.e.* $S \in \mathcal{E}_{\mathsf{PR}}(\Gamma)$*, iff* S *is a maximal (w.r.t. set inclusion) admissible set of* Γ.

3 Configuring Parallel Portfolios of Argumentation Solvers

In this section we describe the techniques we designed for combining argumentation solvers into parallel portfolios. Each approach requires as input: (i) the number of cores and the runtime available for the configured portfolio, (ii) a set of basic solvers that given an AF return the corresponding set of preferred extensions, (iii) a set of training AFs, and (iv) measures of performance of solvers on the training set. Solvers' performance is measured in terms of Penalised Average Runtime (PAR) score. This metric trades off coverage (i.e. the percentage of AFs successfully processed by the cutoff time) and runtime for successfully analysed AFs: runs that do not solve the given problem get ten times the cutoff time (PAR10), other runs get the actual runtime. The PAR10 score of a solver on a set of AFs is the average of the relevant scores.

3.1 Static Parallel Portfolios

The first approach, called *S-Naive*, orders solver according to PAR10 performance achieved on the training instances. Given k available cores, top k solvers are allocated to one core each, and run for all the available runtime.

The second approach for generating static portfolios is called *S-Overall*, and also assigns one solver per core. It starts from an empty portfolio, and iteratively adds the solver—not already included—that maximises the improvement of the PAR10 score of the portfolio. It continues until no more cores are available, or it is not possible to further improve the PAR10 score of the portfolio on the training instances. In the latter case, remaining x cores are allocated to the x solvers with the best PAR10 that are not yet member of the portfolio.

Two other approaches, called *Iterative-Single* and *Iterative-All*, are inspired by the hill-climbing method introduced in [15]. The mentioned approach had to be extended in order to be able to handle multiple parallel cores, and to generate portfolios for minimising the (expected) runtime.

S-Iter-Single configures a different sequential portfolio for each available core. Given a core, this method starts by an empty portfolio, and iteratively either add a new basic solver to the portfolio, or extend the allocated CPU-time of a solver already added to the portfolio, depending on what maximises the decrement of the PAR10 score for the considered sequential portfolio. Once the sequential portfolio for the given core has been configured, the selected solvers are removed from the pool of available solvers, and the process moves to the next core. As it is apparent, the portfolio configured for a given core has no information about the other portfolios, running on the other processing units, or the number of available cores. This approach aims at reducing the complexity of generating a parallel portfolio, and at the same time maximising the diversity of included solvers.

S-Iter-All configures a single general parallel portfolio by considering at the same time all the available cores, and distributing solvers among them. In each step of the configuration process, either the CPU-time allocated to one included solver is increased, or a new solver is added to the portfolio. In the latter case, the solver is scheduled on the core that allows it the earliest start.

3.2 Dynamic Parallel Portfolios

Dynamic portfolios rely on instance features for configuring an instance-specific portfolio. For each AF a vector of features is computed; each feature is a real number that summarises a potentially important aspect of the considered AF. Similar instances should have similar feature vectors, and, on this basis, portfolios are configured using empirical performance models [16].

In this investigation we consider the largest set of features available for AFs [6]. Such set includes 50 features, extracted by exploiting the representation of AFs both as directed (loss-less) or undirected (lossy) graphs. Features are extracted by considering aspects such as the size of graphs, the presence of connected components, the presence of auto-loops, etc. The features extraction process has been parallelised in order to minimise its impact on portfolio performance. Remarkably, as features are extracted by considering two different representations of AFs, their extraction is suitable to be parallelised on two different cores. Following this approach, feature extraction usually requires less than 1 wallclock time second on average.

In this work we propose two techniques, inspired by [10], for generating per-instance parallel portfolios, both based on regression models. Regression models predict the runtime needed by a solver for analysing the considered AF on the basis of the extracted features and on the performance observed on training instances.

The *R-Overall* approach orders the solvers according to the predicted runtime on the given AF, and then allocates one solver per core. While simplistic, this approach aims at reducing the detrimental impact of underestimation mistakes of the predictive model. Instead, the proposed *R-Iterative* technique initially allocates the solvers predicted to be fastest on available processing units. However, each solver is run only for its predicted CPU-time (increased by 10%

for accounting minor underestimation mistakes). If a solver does not successfully analyse the considered AF in the allocated CPU-time, it is stopped and no longer available to be selected, and the process iterates by selecting a different solver.

4 Experimental Analysis

Our experimental analysis aims to evaluate the fruitfulness of parallel portfolios for solving hard argumentation problems, by focusing on the problem of enumerating preferred extensions.

We randomly generated 2,000 AFs based on four different graph models: Barabasi-Albert [2], Erdös-Rényi [13], Watts-Strogatz [23] and graphs featuring a large number of stable extensions (hereinafter StableM).

Erdös-Rényi graphs [13] are generated by randomly selecting attacks between arguments according to a uniform distribution. While Erdös-Rényi was the predominant model used for randomly generated experiments, [5] investigated also other graph structures such as *scale-free* and *small-world* networks. As discussed by Barabasi and Albert [2], a common property of many large networks is that the node connectivities follow a *scale-free* power-law distribution. This is generally the case when: (i) networks expand continuously by the addition of new nodes, and (ii) new nodes attach preferentially to sites that are already well connected. Moreover, Watts and Strogatz [23] show that many biological, technological and social networks are neither completely regular nor completely random, but something in the between. They thus explored simple models of networks that can be tuned through this middle ground: regular networks *rewired* to introduce increasing amounts of disorder. These systems can be highly clustered, like regular lattices, yet have small characteristic path lengths, like random graphs, and they are named *small-world* networks by analogy with the small-world phenomenon. The AFs have been generated by using `AFBenchGen2` [7]. It is worthy to emphasise that Watts-Strogatz and Barabasi-Albert produce undirected graphs: in this work, differently from [5], each edge of the undirected graph is then associated with a direction following a probability distribution, that can be provided as input to `AFBenchGen`. Finally, the fourth set has been generated using the code provided in `Probo` [8] by the organisers of ICCMA-15 [20].

In order to identify challenging frameworks—i.e., neither trivial nor too complex to be successfully analysed in the given CPU-time—AFs for each set have been selected using the protocol introduced in the 2014 edition of the International Planning Competition [22]. This protocol lead to the selection of AFs with a number of arguments between 250 and 650, and number of attacks between (approximately) 400 and 180,000.

The set of AFs has been divided into training and testing sets. For each graph model, we randomly selected 200 AFs for training, and the remaining 300 for testing. Therefore, out of the 2,000 AFs generated, 800 have been used

for training purposes, while the remaining 1,200 have been used for testing and comparing the performance of trained approaches.

We considered all the 15 solvers that took part in the EE-PR track of ICCMA-15 [20]. For the sake of clarity and conciseness, we removed from the analysis single solvers that did not successfully analyse at least one AF or which were always outperformed by another solver. The interested reader is referred to [19] for detailed descriptions of the solvers. Hereinafter, we will refer to such systems as *basic solvers*, regardless of the approach they exploit for solving argumentation-related problems.

Experiments have been run on a cluster with computing nodes equipped with 2.5 Ghz Intel Core 2 Quad Processors, 4 GB of RAM and Linux operating system. A cutoff of 600 wallclock seconds was imposed to compute preferred extensions for each AF. For each solver we recorded the overall result: success (if it solved the considered problem), crashed, timed-out or ran out of memory.

Given the large availability of quad-core processing units, in this work we focus on portfolios configured to run on four cores. Notably, the proposed configuration techniques are general, and can be straightforwardly exploited for configuring portfolios for a different number of cores.

After a set of experiments on a subset of the training instances, the M5-rule technique [14] has been selected for generating the regression models for predicting the expected runtime of solvers.

4.1 Results

Table 1 compares the results of solvers and the proposed parallel portfolio approaches on the testing set of 1,200 AFs. In ICCMA, solvers have been evaluated by considering only coverage (in case of ties the overall runtime on solved instances). Here results are shown in terms of PAR10, coverage and number of time an approach has been the fastest. These results clearly indicate that combining solvers in parallel portfolios is a very fruitful way for reducing the wall-clock time required for successfully analyse an AF. Table 1 also shows the performance of the Virtual Best Solver (VBS). The VBS shows the performance of a (virtual) oracle which always selects the best (fastest) solver for the given framework. The performance gap between the best basic solver and the VBS gives a good indication about the level of complementarity of the considered reasoners.

Remarkably, the performance of the S-Overall and R-Overall approaches are very close to those of the VBS: this is a clear indication that the proposed techniques are able to effectively select and combine the basic solvers. In particular, the per-instance portfolio R-Overall is able to identify the fastest solver, and run it first, in the 95.7% of the cases. It is also interesting to note that iterative approaches, i.e. those that are allowed to run sequentially more than one solver per core, are not able to fruitfully exploit this additional degree of freedom. Our intuition is that the underestimation of the CPU-time needed by basic solvers, due to the fact that training instances are smaller than and/or different from testing ones, is strongly affecting the performance of iterative approaches. On

Table 1. Performance, in terms of PAR10 and coverage (cov.)—percentage of AFs successfully analysed—of the considered *basic solvers* and generated parallel portfolios, for solving the preferred enumeration problem on the complete testing set (All) of 1,200 AFs. F.t column indicates the number of times a system has been the fastest among considered. Systems are listed in the order of increasing PAR10.

System	**PAR10**	**Cov.**	**F.t**
VBS	562.9	91.4	1118
S-Overall	569.6	91.3	968
R-Overall	573.6	91.2	1070
S-Iter-All	665.9	89.8	629
R-Iterative	907.3	85.4	911
S-Naive	1013.3	84.3	511
S-Iter-Single	1032.4	84.1	214
Cegartix	1350.4	79.1	229
ArgSemSAT	1916.2	69.1	35
LabSATSolver	2050.3	66.8	9
prefMaxSAT	2057.2	66.8	273
DIAMOND	2417.0	61.0	1
ASPARTIX-D	2728.6	56.1	4
ASPARTIX-V	2772.2	55.2	21
CoQuiAas	3026.4	50.5	78
ASGL	3477.3	43.2	1
Conarg	3696.3	39.3	158
ArgTools	3906.2	35.2	322
Gris	4543.7	24.4	174

the contrary, overall approaches do not need to consider this aspect, at the cost of a reduced number of solvers that can be run.

In order to shed some light on the configuration of static portfolios, Table 2 shows the CPU-time allocated to each solver by the proposed techniques. Interestingly, the S-Iter-Single approach shows a behaviour that is very different from what can be observed for S-Iter-All. The former opted for including a huge number of solvers, in fact all but one; the latter instead is allocating most of the cores to a single solver each. Unsurprisingly, given the fact that it is by far the best basic solver, all the static approaches are including Cegartix. It is the only solver that is identified by all the techniques as important and worthy to be run. Similarly, we noticed that per-instance portfolios are usually including it. As a side note, given the good performance observed on training instances, the R-Iterative approach tend to allocate to Cegartix a short amount of CPU-time.

When considering the results shown in Table 1 and Table 2, the question about the actual importance of Cegartix—or the other considered basic solvers—for a portfolio naturally arises. In other words, what is the added value provided by a solver to a portfolio? For answering this question, thus getting some insights into the state of the art of argumentation solvers for enumerating preferred extensions, we rely on the notion of *state-of-the-art contributors* (SOTA) [18,

Table 2. Solvers included in the portfolios configured by the proposed techniques. ■ indicates that the solver has been selected for running on a core for the maximum available time, otherwise allocated CPU-time seconds are shown.

Solver	**S-Naive**	**S-Overall**	**S-Iter-All**	**S-Iter-Single**
Cegartix	■	■	570	210
ArgSemSAT	■		■	420
LabSATSolver	■			150
prefMaxSAT	■	■		210
DIAMOND				300
ASPARTIX-D				300
ASPARTIX-V				300
CoQuiAas				150
ASGL				
Conarg			30	30
ArgTools		■	■	150
Gris		■	■	180

24]. SOTA assess the contribution of a basic solver by the performance decrease of the VBS when the considered solver is omitted. This method reflects the added value due to a given solver much more effectively than comparing average performance. This is because, for instance, the SOTA method can identify—and correctly recognise—solvers that may have poor performance on average, but are able to analyse some extremely challenging AFs that would not be analysed by any other basic solver.

Figure 1 presents the top five basic solvers according to their marginal PAR10 contribution—evaluated following the SOTA method previously recalled—to the VBS. For the sake of readability, solvers with low marginal score—i.e, less than 1 PAR10 point—are not shown.

Surprisingly, the largest marginal PAR10 increment is provided by Gris, followed by Cegartix and prefMaxSAT. These results are in apparent contradiction with results shown in Table 1: however, they are explained by the great performance of Gris on Barabasi AFs. It is the only considered basic solver that is able to analyse the vast majority of such frameworks. Similarly, prefMaxSAT does not show outstanding overall performance, but it tends to be fast on some challenging frameworks. It is also interesting to notice the very limited contribution of ArgTools to the VBS. ArgTools is the solver that would have been ranked first according to the number of time it has been the fastest. Yet its contribution is limited because such AFs are quickly addressed also by other solvers.

To assess the generalisation ability of the proposed approaches for static and per-instance portfolios, we exploited the *leave-one-out* methodology. Starting from the original training set composed by 800 AFs, we removed all the frameworks corresponding to one set at a time, and randomly oversampled frameworks from the remaining three sets—in order to have again approximately 800 frameworks for training. The generated portfolios were then tested on the complete testing set of 1,200 frameworks. The results of this analysis are presented in Table 3.

Static portfolios usually show better generalisation performance; per-instance approaches tend to be more sensitive to the lack of representativeness of the

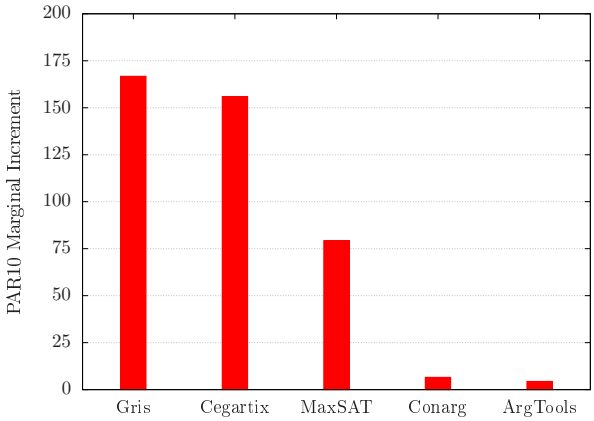

Fig. 1. PAR10 Marginal increments, with regard to the VBS, given by the top five solvers that took part in ICCMA 2015. PAR10 of the VBS, including all the available solver, is 562.9.

Table 3. Performance, in terms of PAR10 and coverage (cov.)—percentage of AFs successfully analysed—-of the systems considered in this study on the complete testing set, when trained on a training set not containing AFs of that structure (leave-one-set-out scenario). Best results in bold.

	Barabasi		Erdos		Stable		Watts	
Solver	**PAR10**	**Cov.**	**PAR10**	**Cov.**	**PAR10**	**Cov.**	**PAR10**	**Cov.**
R-Iterative	1310.8	78.7	1223.8	80.2	989.4	84.1	835.9	86.8
R-Overall	**823.9**	**87.1**	**574.8**	**91.3**	744.0	88.4	799.2	87.6
S-Iter-All	958.7	85.2	664.1	89.9	957.1	84.8	699.8	89.2
S-Iter-Single	1598.0	74.2	1068.7	83.5	1218.3	80.8	1056.0	83.8
S-Overall	958.6	85.2	664.1	89.9	**569.6**	**91.3**	**645.0**	**90.1**
S-Naive	1916.2	69.1	1350.4	79.1	1916.2	69.1	1916.2	69.1

training set, with regards to testing instances. On the one hand, results presented in Table 3 confirm that the performance of static portfolios remain very stable regardless of the used training set, and the S-Overall portfolio is consistently among the best options. On the other hand, per-instance regression-based portfolio R-Overall guarantees very good generalisation. In particular, the predictive model is able to provide accurate runtime predictions even when Barabasi or Erdos frameworks are removed from the training set. This suggests that the exploited features are sufficiently informative for the task of identifying solvers to run in parallel. However, as indicated by the generally poor performance of R-Iterative, predicting the actual runtime is significantly harder.

Comparison with SoA of Parallel Argumentation Reasoning In order to assess the fruitfulness of exploiting parallel computational units by combining

Table 4. Performance, in terms of PAR10 and coverage (cov.)—percentage of AFs successfully analysed—of the generated parallel portfolios and of the parallel solver P-SCC-REC, for solving the preferred enumeration problem on the testing set of 120 AFs. Systems are listed in the order of increasing PAR10.

System	PAR10	Cov.
S-Iter-Single	15.8	100.0
R-Overall	21.0	100.0
S-Iter-All	28.6	100.0
S-Naive	29.8	100.0
S-Overall	39.7	100.0
R-Iterative	71.0	99.2
P-SCC-REC	816.8	89.2

sequential solvers into parallel portfolios, and to better contextualise the performance of the configured portfolios, in this section we compare the generated static and per-instance portfolios with the state of the art of parallel solvers for enumerating preferred extensions: P-SCC-REC [9].

P-SCC-REC is based on the SCC-recursive schema [4]. It recursively decomposes a framework so as to compute semantics labellings (that are in a one-to-one relationship with extensions) on restricted sub-frameworks, in this case strongly-connected components, in order to reduce the computational effort. The labellings computation is then parallelised: each core is dedicated to a single SCC, and results are merged at each recursion layer. Here we consider the P4 version, which exploits four parallel cores.

It has been shown that P-SCC-REC takes advantage of available parallel cores when the number of SCCs is higher than 40 [9]. In the benchmarks considered in our analysis, this is not usually the case. Therefore, for this comparison we generated 120 AFs following the distribution used in [9]. These are extremely large AFs, with a number of SCCs between 90 and 150, the number of arguments between 2,700 and 6,000, and considering different uniformly distributed probabilities of attacks, either between arguments or between different SCCs, leading to AFs with a number of attacks between approximately 100 thousands and more than 1 million.

Given the extreme size of the AFs, we observed that the sequential CPU-time required for extracting features can be significant, between 2 and 250 seconds. However, the extraction process can be parallelised on the available cores. In this way, we were able to reduce the wall-clock time to one third of the sequential time. In the followings, extraction time is included in the portfolio results.

Table 4 shows the PAR10 and coverage performance of P-SCC-REC, and of the parallel portfolio approaches. Portfolios have been configured according to the complete training set exploited in the previous analysis. Despite this, results clearly indicate that portfolio-based approaches, either static or per-instance, are able to outperform the state of the art parallel solver P-SCC-REC. The portfolio generated following the R-Iterative approach is the only portfolio-based system

that is not able to solve the 100% of the testing frameworks. According to our analysis, this is due to a huge overestimation of the runtime of solvers. The testing AFs are significantly larger than those used for training purposes.

Summary Results of this extensive experimental analysis support the hypothesis that combining solvers in parallel portfolios is, at the state of the art, the most fruitful way for exploiting multicore machines for abstract argumentation problems. Sequential solvers are able to provide very good performance, due to the higher level of optimisation, and their complementarity allows to combine effectively them in portfolios.

5 Conclusion

In the light of the current trend of increasing computational power through hardware parallelisation, we presented six approaches for configuring parallel portfolios of argumentation solvers for enumerating preferred extensions. We introduced four methodologies for configuring static portfolios, and two techniques for configuring and executing dynamic portfolios.

Our extensive experimental analysis: (i) assessed the marginal increments given by state-of-the-art solvers; (ii) demonstrated that static portfolios tend to generalise better on previously unseen testing AFs; (iii) confirmed that parallel portfolios outperform the state-of-the-art parallel argumentation solver, thus are a fruitful way for exploiting multicore machines.

Future work includes the investigation of techniques for combining solvers in *mixed* portfolios, i.e. partly dynamically and partly statically configured. Given the number of abstract argumentation computational problems, we are also interested in identifying methodologies for generating portfolios of solvers that can solve different problems in parallel, therefore minimising the overall time required to get a complete overview of a given AF.

Acknowledgement The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

References

1. Balyo, T., Sanders, P., Sinz, C.: Hordesat: A massively parallel portfolio SAT solver. In: Theory and Applications of Satisfiability Testing SAT. pp. 156–172 (2015)
2. Barabasi, A., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 11 (1999)
3. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. Knowledge Eng. Review 26(4), 365–410 (2011)
4. Baroni, P., Giacomin, M., Guida, G.: SCC-recursiveness: a general schema for argumentation semantics. Artificial Intelligence 168(1-2), 165–210 (2005)
5. Bistarelli, S., Rossi, F., Santini, F.: Benchmarking Hard Problems in Random Abstract AFs: The Stable Semantics. In: COMMA 2014. pp. 153–160 (2014)

6. Cerutti, F., Giacomin, M., Vallati, M.: Algorithm selection for preferred extensions enumeration. In: Proceedings of COMMA. pp. 221–232 (2014)
7. Cerutti, F., Giacomin, M., Vallati, M.: Generating structured argumentation frameworks: Afbenchgen2. In: Proceedings of COMMA. pp. 467–468 (2016)
8. Cerutti, F., Oren, N., Strass, H., Thimm, M., Vallati, M.: A benchmark framework for a computational argumentation competition. In: Proceedings of COMMA. pp. 459–460 (2014)
9. Cerutti, F., Tachmazidis, I., Vallati, M., Batsakis, S., Giacomin, M., Antoniou, G.: Exploiting parallelism for hard problems in abstract argumentation. In: Proceedings of AAAI. pp. 1475–1481 (2015)
10. Cerutti, F., Vallati, M., Giacomin, M.: Where are we now? state of the art and future trends of solvers for hard argumentation problems. In: Proceedings of COMMA. pp. 207–218 (2016)
11. Dung, P.M.: On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming, and n-Person Games. Artificial Intelligence 77(2), 321–357 (1995)
12. Dunne, P.E., Wooldridge, M.: Complexity of abstract argumentation. In: Argumentation in AI, chap. 5, pp. 85–104. Springer-Verlag (2009)
13. Erdös, P., Rényi, A.: On random graphs. I. Publ. Math. Debrecen 6, 290–297 (1959)
14. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)
15. Helmert, M., Röger, G., Karpas, E.: Fast Downward Stone Soup: A baseline for building planner portfolios. In: Proceedings of the ICAPS-11 Workshop of AI Planning and Learning (PAL) (2011)
16. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: Methods & evaluation. Artificial Intelligence 206, 79–111 (2014)
17. Lindauer, M.T., Hoos, H.H., Hutter, F.: From sequential algorithm selection to parallel portfolio selection. In: Learning and Intelligent Optimization - 9th International Conference, LION. pp. 1–16 (2015)
18. Sutcliffe, G., Suttner, C.: Evaluating general purpose automated theorem proving systems. Artificial Intelligence 131(1), 39–54 (2001)
19. Thimm, M., Villata, S.: System descriptions of the first international competition on computational models of argumentation (iccma'15). arXiv preprint arXiv:1510.05373 (2015)
20. Thimm, M., Villata, S., Cerutti, F., Oren, N., Strass, H., Vallati, M.: Summary report of the first international competition on computational models of argumentation. AI Magazine (2016)
21. Toniolo, A., Norman, T.J., Etuk, A., Cerutti, F., Ouyang, R.W., Srivastava, M., Oren, N., Dropps, T., Allen, J.A., Sullivan, P.: Agent Support to Reasoning with Different Types of Evidence in Intelligence Analysis. In: Proc. of AAMAS. pp. 781–789 (2015)
22. Vallati, M., Chrpa, L., Grzes, M., McCluskey, T., Roberts, M., Sanner, S.: The 2014 international planning competition: Progress and trends. AI Magazine (2015)
23. Watts, D.J., Strogatz, S.H.: Collective dynamics of 'small-world' networks. Nature 393(6684), 440–442 (1998)
24. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: Evaluating component solver contributions to portfolio-based algorithm selectors. In: Proceedings of SAT. pp. 228–241 (2012)