Stochastic Local Search Algorithms for Abstract Argumentation under Stable Semantics

Matthias THIMM
University of Koblenz-Landau, Germany

Abstract. We present a family of stochastic local search algorithms for finding a single stable extension in an abstract argumentation framework. These incomplete algorithms work on random labellings for arguments and iteratively select a random mislabeled argument and flip its label. We present a general version of this approach and an optimisation that allows for greedy selections of arguments. We conduct an empirical evaluation with benchmark graphs from the previous two ICCMA competitions and further random instances. Our results show that our approach is competitive in general and significantly outperforms previous direct approaches and reduction-based approaches for the Barabási-Albert graph model.

Keywords. abstract argumentation, algorithms, stochastic local search

1. Introduction

Abstract argumentation frameworks [7] provide a simple formal representation for discussing general aspects pertaining to computational models of argumentation. These frameworks abstract from the inner structure of arguments and focus solely on the interaction between arguments by means of a conflict relation. Formally, an abstract argumentation framework is a directed graph where vertices are identified with arguments and a directed edge between two arguments models an *attack* of the first argument onto the second argument. Despite the fact that abstract argumentation frameworks are a simple formalism for a computational model of argumentation, several semantical issues can already be discussed in this framework, giving rise to a plethora of different semantics [1]. The core notion of the semantics of an abstract argumentation framework is the *extension*, i. e., a set of arguments that are justifiable as a whole and provide a specific standpoint for reasoning. Determining these extensions can be computationally demanding due to the fact that already the underlying decision problems lie up to the second level of the polynomial hierarchy [8].

Algorithmic approaches to abstract argumentation [4] have recently gained attention in the community, mainly due to the International Competition of Computational Models of Argumentation (ICCMA), where the third instance is currently being organised.[1] We follow in this line of work and contribute to the field by developing novel algorithms for the problem of determining *some* stable extension and evaluating them with a compara-

[1]`http://argumentationcompetition.org/2019`

tive empirical evaluation with state-of-the-art solvers from the recent two competitions. Our algorithms follow the paradigm of *stochastic local search*, i. e., incomplete optimisation algorithms that aim at reaching an optimal value of a target function by small random changes of the parameters, see e. g. [2, Chapter 6] for a deeper discussion in the context of solving the satisfiability problem (SAT). The core idea of our algorithms is as follows. Considering the labelling approach to the semantics of abstract argumentation frameworks, we start from a labelling that randomly assigns the acceptability status in and out to all arguments of the input argumentation framework. As long as this labelling is not stable—i. e. as long as the arguments labelled in do not form a stable extension—we select one mislabelled argument and flip its acceptability status. Albeit being a simple idea it can outperform traditional algorithms, in particular on *random* instances with little structure.

In summary, the contributions of this paper are as follows.

1. We develop a family of stochastic local search algorithms for computing a single stable extension of an abstract argumentation framework (Section 4)
2. We compare the empirical performance of our approach with state-of-the-art solvers from the previous two competitions (Section 5)

Background on abstract argumentation is given in Section 2, related works are discussed in Section 3, and we conclude with a summary in Section 6.

2. Abstract Argumentation

An *abstract argumentation framework* AF is a tuple $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ where Arg is a set of arguments and $\rightarrow$ is a relation $\rightarrow \subseteq \mathsf{Arg} \times \mathsf{Arg}$. For two arguments $\mathscr{A}, \mathscr{B} \in \mathsf{Arg}$ the relation $\mathscr{A} \rightarrow \mathscr{B}$ means that argument $\mathscr{A}$ attacks argument $\mathscr{B}$. For $\mathscr{A} \in \mathsf{Arg}$ define $\mathscr{A}^- = \{\mathscr{B} \mid \mathscr{B} \rightarrow \mathscr{A}\}$. Semantics are given to abstract argumentation frameworks by means of extensions [7] or labellings [3]. In this work, we use the latter. A labelling L is a function $L : \mathsf{Arg} \rightarrow \{\mathtt{in}, \mathtt{out}, \mathtt{undec}\}$ that assigns to each argument $\mathscr{A} \in \mathsf{Arg}$ either the value in, meaning that the argument is accepted, out, meaning that the argument is not accepted, or undec, meaning that the status of the argument is undecided. Let $\mathtt{in}(L) = \{\mathscr{A} \mid L(\mathscr{A}) = \mathtt{in}\}$ and $\mathtt{out}(L)$ resp. $\mathtt{undec}(L)$ be defined analogously. A labelling L is called *conflict-free* if for no $\mathscr{A}, \mathscr{B} \in \mathtt{in}(L)$, $\mathscr{A} \rightarrow \mathscr{B}$.

Arguably, the most important property of a semantics is its admissibility. A labelling L is called *admissible* if and only if for all arguments $\mathscr{A} \in \mathsf{Arg}$

1. if $L(\mathscr{A}) = \mathtt{out}$ then there is $\mathscr{B} \in \mathsf{Arg}$ with $L(\mathscr{B}) = \mathtt{in}$ and $\mathscr{B} \rightarrow \mathscr{A}$, and
2. if $L(\mathscr{A}) = \mathtt{in}$ then $L(\mathscr{B}) = \mathtt{out}$ for all $\mathscr{B} \in \mathsf{Arg}$ with $\mathscr{B} \rightarrow \mathscr{A}$,

and it is called *complete* if, additionally, it satisfies

3. if $L(\mathscr{A}) = \mathtt{undec}$ then there is no $\mathscr{B} \in \mathsf{Arg}$ with $\mathscr{B} \rightarrow \mathscr{A}$ and $L(\mathscr{B}) = \mathtt{in}$ and there is a $\mathscr{B}' \in \mathsf{Arg}$ with $\mathscr{B}' \rightarrow \mathscr{A}$ and $L(\mathscr{B}') \neq \mathtt{out}$.

The intuition behind admissibility is that an argument can only be accepted if there are no attackers that are accepted and if an argument is not accepted then there has to be some reasonable grounds. The idea behind the completeness property is that the status of an argument is only undec if it cannot be classified as in or out. Different types of classical semantics can be phrased by imposing further constraints. In particular, a complete labelling L

- is *grounded* if and only if $\texttt{in}(L)$ is minimal,
- is *preferred* if and only if $\texttt{in}(L)$ is maximal, and
- is *stable* if and only if $\texttt{undec}(L) = \emptyset$.

All statements on minimality/maximality are meant to be with respect to set inclusion. If L is a complete/grounded/preferred/stable labelling then $\texttt{in}(L)$ is also called the corresponding complete/grounded/preferred/stable extension.

In this paper we focus on the stable semantics and investigate algorithms that are able to find a single stable labelling of a given abstract argumentation framework AF. Recall that the problem of deciding whether a stable labelling exists is NP-complete [8].

3. Related Work

According to [6], algorithms for solving reasoning problems in abstract argumentation can generally be categorised into two classes: *reduction-based* approaches and *direct* approaches.

Reduction-based approaches such as ASPARTIX-D [9,11] and ArgSemSAT [5] translate the given problem for abstract argumentation—such as determining a single stable extension—into another formalism and use dedicated (and mature) systems for that formalism to solver the original problem. For example, ASPARTIX encodes the problem of finding a stable extension in abstract argumentation into the question of finding an answer set of an answer set program [12]. Due to the direct relationship of answer sets and stable models the answer set program only needs to model the semantics of the abstract argumentation framework in a faithful manner and represent the actual framework. ASPARTIX-D then makes use of the Potassco ASP solvers[2] to solve the reduced problem and translate their output back to the original question. Similarly, ArgSemSAT decodes the problem as a SAT instance and uses the Glucose[3] SAT solver to solve the latter. Internally, solvers such as the Potassco ASP solvers and SAT solvers make use of sophisticated search strategies such as *conflict-driven nogood learning* or *conflict-driven clause learning*, see [12,2] for details.

Direct approaches to solve reasoning problems in abstract argumentation are inspired by similar search strategies but directly realise these algorithms for abstract argumentation. For example, solvers such as ArgTools [15] and heureka [13] are based on the DPLL (Davis-Putnam-Logemann-Loveland) backtracking algorithm from SAT solving [2, Chapter 3]. Basically, they exhaustively explore the search space of all possible sets of arguments to determine, e. g., a stable extension but include various optimisations and specific search strategies to prune the search space as much as possible to keep runtime low. Another direct solver, EqArgSolver [16], uses a different approach though, and is inspired by an iteration scheme originally designed to solve problems for probabilistic argumentation [10]. For a more detailed discussion of the different approaches to solving problems in abstract argumentation see [4].

The above approaches to solve reasoning problems in abstract argumentation are complete, i. e., when they terminate they always produce the correct answer. In this pa-

[2]`http://potassco.sourceforge.net`
[3]`http://www.labri.fr/perso/lsimon/glucose/`

per, we will use *stochastic local search*, a non-deterministic and not necessarily complete search procedure. To the best of our knowledge, only the recent work [14] applied stochastic local search to abstract argumentation before. However, Niu et al. address preferred semantics and they approach differs conceptually from ours. They operate on a CNF representation of the extension finding problem and applied the stochastic local search algorithm *Swcca* on that representation instead of the actual graph representation. Therefore, the approach of [14] is reduction-based, while we pursue a direct approach.

The International Competition of Computational Models of Argumentation (ICCMA) is a bi-annual event that assesses the performance of solvers for various tasks related to abstract argumentation in a competitive setting. In order to compare our approach to state-of-the-art solvers, we used the best three approaches from the last competition[4] (ICCMA'17) from the stable semantics track as reference solvers. These solvers were pyglaf, goDIAMOND, and argmat-sat. In addition, we also included ASPARTIX-D which won the corresponding track in ICCMA'15 but did not participate in ICCMA'17 due to a conflict of interest. As these four solvers are all reduction-based approaches (pyglaf and argmat-sat are based on SAT-reductions while goDIAMOND and ASPARTIX-D are based on ASP-reductions) we also included all three direct solvers participating in the stable semantics track from ICCMA'17: ArgTools, heureka, and EqArgSolver. These seven solvers therefore constitute the state-of-the-art in determining a single stable extension, taking different implementation paradigms into account.

4. Stochastic Local Search Algorithms

The term *stochastic local search* denotes search algorithms that aim at iteratively improving a target function by small random changes in its arguments. A well-known application for stochastic local search is SAT, in particular due to the GSAT [17] and WalkSAT algorithms [18], see also [2, Chapter 6]. The central idea of the GSAT/WalkSAT algorithm is as follows. Initially, some random interpretation is selected and, as long as this interpretation does not satisfy the input formula, some variable of some unsatisfied clause is selected at random and its truth value is flipped in the interpretation. Eventually, i. e., with probability strictly greater than zero, if the formula is satisfiable then some satisfying interpretation is found in this way. However, note that GSAT/WalkSAT, and stochastic local search in general, is an *incomplete algorithm*. That means in the case of an unsatisfiable formula the GSAT/WalkSAT algorithm will not terminate. In order to obtain practical systems, either a timeout is used and unsatisfiability is proclaimed with some confidence value or such an algorithm is combined with a more efficient algorithm that shows unsatisfiability.

In this section, we will present stochastic local search algorithms for determining a stable extension of an abstract argumentation framework. More precisely, we address the problem SE-ST from the ICCMA competition:

SE-ST	**Input**:	An argumentation framework $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$
	Output:	a stable extension E of AF or NO if there are no stable extensions

Our algorithms will be incomplete as well, meaning that they will never output NO in case of non-existence of extensions but (conceptually) loop forever. We will, however,

[4]`http://argumentationcompetition.org/2017`

constrain the running time of our implementations by providing the algorithm with a maximal number of tries before terminating with a notification of failure.

We proceed in this section as follows. In Section 4.1 we will first present a base algorithm that directly adapts the GSAT/WalkSAT algorithm for the above problem. In Section 4.2 we will present an optimisation by adding deterministic greedy moves to the random moves of the algorithm.

4.1. The base algorithm WalkAAF

Our base algorithm WalkAAF is a direct implementation of the GSAT/WalkSAT idea outlined above. Instead of working with a propositional interpretation, WalkAAF works with a labelling and each iteration a label of some argument is modified. As our aim is to obtain a stable labelling, we wish to avoid mislabeled arguments defined as follows.

Definition 1. Let L be a labelling for $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$. An argument $\mathcal{A} \in \mathsf{Arg}$ is *mislabeled* in L if

- $L(\mathcal{A}) = \texttt{undec}$, or
- $L(\mathcal{A}) = \texttt{out}$ and there is no $\mathcal{B} \rightarrow \mathcal{A}$ with $L(\mathcal{B}) = \texttt{in}$, or
- $L(\mathcal{A}) = \texttt{in}$ and
    - $*$ there is $\mathcal{B} \rightarrow \mathcal{A}$ with $L(\mathcal{B}) \neq \texttt{out}$ or
    - $*$ there is $\mathcal{A} \rightarrow \mathcal{B}$ with $L(\mathcal{B}) \neq \texttt{out}$

The following result follows straightforwardly from the definition of a stable labelling and is given without proof.

Proposition 2. *A labelling L is stable iff there is no mislabeled argument in L.*

Algorithm 1 depicts the $\mathsf{WalkAAF}_{N,M}$ base algorithm which implements the GSAT/WalkSAT idea without any optimisations but with restarts [2, Chapter 6]. The algorithm works with two externally given parameters N and M with $N, M \in \mathbb{N}$. The parameter N gives the maximal number of runs (=restarts) of the algorithms before the algorithm terminates with a notification of failure (FAIL). The parameter M gives the number of iterations in each run. Each run starts with determining some random labelling L that labels all arguments with either `in` or `out` (line 2). Note that as we wish to obtain a stable labelling, we completely neglect the label `undec`. If the labelling L is already stable then the algorithm terminates (line 4/5) . Otherwise, due to Proposition 2 there is at least one mislabeled argument in L. The algorithm selects one of those arguments at random (line 7) and "flips" its acceptance status (lines 8–11), i. e., if it is currently labeled `in` it is changed to `out` and vice versa. The algorithm repeats this process for M steps. If we did not find a stable labelling, we do a "restart", i. e., start from a new random labelling. After N unsuccessful restarts, the algorithm terminates with FAIL (line 12).

The algorithm $\mathsf{WalkAAF}_{N,M}$ is *sound* in the following sense.

Proposition 3. *If L is a labelling returned from a call to* $\mathsf{WalkAAF}_{N,M}$ *on input* AF *then L is a stable labelling of* AF.

Proof. The only case $\mathsf{WalkAAF}_{N,M}$ returns a labelling L is in line 5 where stability of L is ensured in line 4. □

Algorithm 1 $\mathsf{WalkAAF}_{N,M}$ algorithm ($N, M \in \mathbb{N}$)

Input: $\mathsf{AF} = (\mathsf{Arg}, \rightarrow)$ AAF
Output: L a stable labelling (or FAIL if the search failed)

```
1:  for $i = 1, \ldots, N$ do
2:      $L \leftarrow$ randomize in and out
3:      for $j = 1, \ldots M$ do
4:          if L is stable then
5:              return L
6:          else
7:              Pick random mislabeled argument $\mathcal{A}$
8:              if $L(\mathcal{A}) = \texttt{in}$ then
9:                  $L(\mathcal{A}) \leftarrow \texttt{out}$
10:             else
11:                 $L(\mathcal{A}) \leftarrow \texttt{in}$
12: return FAIL
```

Proposition 4. *If* AF *has no stable labellings then any call* $\mathsf{WalkAAF}_{N,M}$ *on input* AF *with finite* N, M *terminates with* FAIL.

Proof. As AF has no stable labellings line 5 will never be reached in $\mathsf{WalkAAF}_{N,M}$. As both N and M are finite, eventually line 12 is executed and FAIL is returned. □

Unfortunately, $\mathsf{WalkAAF}_{N,M}$ is not complete as it may return FAIL even if stable labellings exist.

Example 5. Consider the argumentation framework in Figure 1. Let L_1 be the labelling defined as

$$L_1(\mathcal{A}_1) = \texttt{in} \quad L_1(\mathcal{A}_2) = \texttt{out} \quad L_1(\mathcal{A}_3) = \texttt{in} \quad L_1(\mathcal{A}_4) = \texttt{out} \quad L_1(\mathcal{A}_5) = \texttt{in}$$

Assume L_1 is randomly selected in line 2 of $\mathsf{WalkAAF}_{N,M}$. L_1 is not a stable labelling and argument $\mathcal{A}_3$ and $\mathcal{A}_5$ are mislabeled according to Definition 1. Assume $\mathcal{A}_3$ is selected in line 7 to be relabelled. Then we obtain a new labelling L_2 defined via

$$L_2(\mathcal{A}_1) = \texttt{in} \quad L_1(\mathcal{A}_2) = \texttt{out} \quad L_2(\mathcal{A}_3) = \texttt{out} \quad L_2(\mathcal{A}_4) = \texttt{out} \quad L_2(\mathcal{A}_5) = \texttt{in}$$

Still, L_2 is not stable and argument $\mathcal{A}_4$ is mislabeled. Now $\mathcal{A}_4$ is now selected in line 7 to be relabelled. Then we obtain a new labelling L_3 defined via

$$L_3(\mathcal{A}_1) = \texttt{in} \quad L_3(\mathcal{A}_2) = \texttt{out} \quad L_3(\mathcal{A}_3) = \texttt{out} \quad L_3(\mathcal{A}_4) = \texttt{in} \quad L_3(\mathcal{A}_5) = \texttt{in}$$

Still, L_3 is not stable and arguments $\mathcal{A}_4$ and $\mathcal{A}_5$ are mislabeled. Note that this process can be repeated indefinitely (or until the maximum number M of iterations is reached) without ever obtaining a stable labelling. However, note that the framework does indeed possess a stable labelling, namely L_{st} defined via

$$L_{st}(\mathcal{A}_1) = \texttt{out} \quad L_{st}(\mathcal{A}_2) = \texttt{in} \quad L_{st}(\mathcal{A}_3) = \texttt{out} \quad L_{st}(\mathcal{A}_4) = \texttt{in} \quad L_{st}(\mathcal{A}_5) = \texttt{out}$$

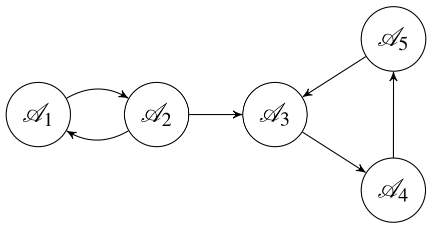


Figure 1. The argumentation framework from Example 5.

We conclude the presentation of the base algorithm $\mathsf{WalkAAF}_{N,M}$ with a theoretical analysis of its runtime and space complexity.

Proposition 6. *For input* $\mathsf{AF} = (\mathsf{Arg}, \to)$ *and* $N, M \in \mathbb{N}$, *$\mathsf{WalkAAF}_{N,M}$ runs in worst-case time of* $O(NM|\mathsf{Arg}|^2)$ *and needs space* $O(|\mathsf{Arg}|)$.

Proof. As lines 2–11 are repeated N times we get the factor N and as lines 4–11 are repeated M times in each outer iteration we get the factor M. Note that line 2 needs linear time in Arg (as each argument gets label), assuming that a random choice can be made in constant time. Within lines 4–11 all operations need constant time except line 4 and line 7. Line 4 can be implemented by iterating through all arguments and checking whether they are mislabeled according to Definition 1. This amounts to at most $|\mathsf{Arg}|^2$ and we also get the list of mislabeled arguments, so line 7 becomes constant.

Space complexity of $O(|\mathsf{Arg}|)$ is obvious as only a labelling L is used as data structure. □

The theoretical time complexity of $O(NM|\mathsf{Arg}|^2)$ can be improved by using intelligent data structures and update operations. For example, if a labelling L' is obtained from a labelling L by only flipping the acceptance status of one argument, only arguments directly connected to it may become mislabeled or correctly labelled. Arguments not connected to it either stay mislabeled or correctly labeled. Therefore, the stability check in the next iteration needs only to take new local information into account and will therefore run significantly faster than $O(|\mathsf{Arg}|^2)$. In our implementation HAYWOOD we incorporated several such optimisations and we report on its empirical performance in Section 5.

4.2. Greedy moves

The base algorithm $\mathsf{WalkAAF}_{N,M}$ makes no distinction between the mislabeled arguments and selects, at each iteration, a mislabeled argument uniformly at random. But already the WalkSAT algorithm [18] takes some more information into account and does, with some certain probability, an occasional *greedy step* instead of a purely random one. In WalkSAT this greedy move consists of flipping a variable that amounts to a maximal number of clauses to be satisfied. For our algorithm $\mathsf{WalkAAF}_{N,M}$ we can implement a similar idea by using the notion of *flipping count* defined as follows.

Definition 7. Let $\mathsf{AF} = (\mathsf{Arg}, \to)$ be an abstract argumentation framework, $\mathscr{A} \in \mathsf{Arg}$, and L a labelling. Let $L_{\overline{\mathscr{A}}}$ denote the labelling that is the same as L except that $L_{\overline{\mathscr{A}}}(\mathscr{A}) = \mathtt{in}$ if $L(\mathscr{A}) = \mathtt{out}$ and $L_{\overline{\mathscr{A}}}(\mathscr{A}) = \mathtt{out}$ if $L(\mathscr{A}) = \mathtt{in}$. Then the *flipping count* of $\mathscr{A}$ wrt. L, abbreviated by $f(L, \mathscr{A})$, is the number of mislabeled arguments in L minus the number of mislabeled arguments in $L_{\overline{\mathscr{A}}}$.

In other words, the larger the flipping count of an argument $\mathscr{A}$ wrt. L the more impact flipping $\mathscr{A}$ has for L to become a stable labelling.

Example 8. Consider again the argumentation framework in Figure 1 and the labelling L given via $L(\mathscr{A}_1) = \texttt{out}$, $L(\mathscr{A}_2) = \texttt{out}$, $L(\mathscr{A}_3) = \texttt{in}$, $L(\mathscr{A}_4) = \texttt{in}$, and $L(\mathscr{A}_5) = \texttt{out}$.

Note that there are four mislabeled arguments in L: $\mathscr{A}_1$, $\mathscr{A}_2$, $\mathscr{A}_3$, $\mathscr{A}_4$. If we would flip the acceptance status of $\mathscr{A}_1$, thus obtaining a labelling L' with $L'(\mathscr{A}_1) = \texttt{in}$, $L'(\mathscr{A}_2) = \texttt{out}$, $L'(\mathscr{A}_3) = \texttt{in}$, $L'(\mathscr{A}_4) = \texttt{in}$, and $L'(\mathscr{A}_5) = \texttt{out}$, there would be two mislabeled arguments left: $\mathscr{A}_3$ and $\mathscr{A}_4$. Therefore, $f(L,\mathscr{A}_3) = 2$. Similarly we get

$$f(L,\mathscr{A}_2) = 0 \qquad f(L,\mathscr{A}_3) = 1 \qquad f(L,\mathscr{A}_4) = 1 \qquad f(L,\mathscr{A}_5) = -1$$

Our first extension of $\mathsf{WalkAAF}_{N,M}$ now takes another external parameter $G \in [0,1]$ and every time some argument has to be selected in line 7 of Algorithm 1, with probability G some argument with maximal flipping count is selected, instead of a random mislabeled one. We denote this new algorithm by $\mathsf{WalkAAF}^{G}_{N,M}$ and, due to space limitations, do not provide the full listing.

Note that adding the occasional greedy move to $\mathsf{WalkAAF}_{N,M}$ does not change the soundness and incompleteness of the algorithm, nor the runtime and space complexity analysis. In fact, keeping track of the flipping counts of all arguments and selecting one with maximal value can be efficiently realised using Fibonacci heaps, which support update and extraction operations in (amortised) constant and logarithmic runtime, respectively, and need only linear space.

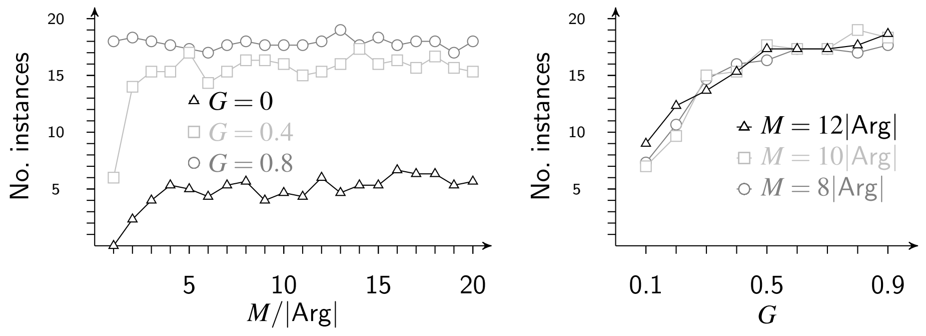
5. Experiments

In the following, we experimentally evaluate the performance of the different variants of our algorithm. First, we discuss how different parameter settings affect the overall performance. Second, we compare the performance of our optimised algorithm with state-of-the-art argumentation solvers. As benchmark graphs for all our experiments, we used (subsets of) graphs used in the First and Second International Competitions on Computational Models of Argumentation (ICCMA15[5] and ICCMA17[6]), see also [19], as well as random graphs based on the Barabási-Albert model generated using AFBenchGen2[7]. As our algorithms are incomplete they are not able to produce correct results for graphs that do not have a stable extension. Therefore, graphs where no solver (neither our solver nor any of the state-of-the-art solvers) was able to produce a stable extension, were omitted in all our experiments. We provide some details on the benchmark graphs below.

5.1. Implementation details

The algorithms from Section 4 have been implemented in the solver HAYWOOD which is written in C, licensed under LGPLv3, and available online[8]. As a preprocessing step, the solver first computes the grounded labelling of an input argumentation framework and removes both the arguments labelled `in` and `out` in the grounded labelling from the ar-

[5]`http://argumentationcompetition.org/2015/`
[6]`http://argumentationcompetition.org/2017/`
[7]`https://sourceforge.net/projects/afbenchgen/`
[8]`http://taas.tweetyproject.org`



(a) Influence of the parameter M (b) Influence of the parameter G

Figure 2. Influence of the parameters of WalkAAF on performance; for all experiments we set $N = \infty$ but set a timeout of 10 minutes; all data points are averaged over 3 runs

gumentation framework. By doing so, we avoid risking to label arguments differently by the stochastic local search later that are already predetermined by the grounded labelling (recall that all arguments labelling `in` by the grounded labelling are also labelled `in` by every stable labelling). After a stable labelling has been determined on the remaining framework, all arguments previously removed are re-introduced with their correct label from the grounded labelling.

In order to implement greedy moves, HAYWOOD uses a Fibonacci heap as priority queue. Whenever an argument is re-labeled, the flipping counts of this argument and all neighbouring arguments are recomputed and updated in the Fibonacci heap. If a greedy move is selected, the top argument of the Fibonacci heap is taken.

5.2. Parameter optimisation

A first series of experiments aims at optimising the two parameters of our algorithm, the frequency of restarts M and the probability of making a greedy move G. For all experiments we set (conceptually) $N = \infty$ but introduced a time out after 10 minutes. For the parameter optimisation we used the 5th data set of ICCMA'15, which contains supposedly the hardest instances for stable semantics. Out of the 24 benchmark graphs in this set 20 graphs possessed at least one stable extension and only these were considered in the following. For the parameter M we quickly discovered a (linear) dependency of the number of arguments of a framework and suitable choices for M. In other words, if a framework has more arguments, M must be proportionally larger. This is quite obvious as stochastic local search needs, in the best case, linearly more time on linearly larger frameworks. We therefore decided to define M always as a factor in terms of the number of arguments of the framework at hand. For example, setting $M = 8|\mathsf{Arg}|$ means that we set M to the value 8 times the number of arguments in the individual framework.

We ran HAYWOOD on the 20 benchmark graphs with parameters $M \in \{1|\mathsf{Arg}|, \ldots, 20|\mathsf{Arg}|\}$ and $G \in \{0.1, \ldots, 0.9\}$ and counted the number of graphs that could be solved within the time limit of 10 minutes. We ran this experiment three times and took the average number of solved graphs for each parameter combination.

Figure 2 shows the results of the parameter optimisation. In particular, Figure 2 (a) shows how many graphs could be solved with three different but fixed values for G and

	ICCMA'15							ICCMA'17 - B					Sum
	1	2	3	4	5	8	9	1	2	3	4	5	
pyglaf	24	24	24	18	20	2	6	33	33	28	49.6	27.3	289
argmat-sat	24	24	24	18	20	2	6	32	32	32	50	27.6	291.6
goDIAMOND	24	24	24	18	20	2	6	33	30	31	50	31	293
ASPARTIX-D	24	24	24	18	20	2	6	33	33	31	50	28.6	293.6
ArgTools	24	24	24	18	20	2	6	33	31	31	44.6	25	282.6
heureka	24	24	24	16	9	2	6	33	33	22	40	25	258
EqArgSolver	24	24	24	4	0	2	6	33	27	16	25	0	185
HAYWOOD-1	24	24	24	18	17.3	1.6	6	32	29	27.6	45.6	22.6	272
HAYWOOD-2	24	24	24	18	17.6	2	6	31.6	29.6	27.6	45.6	24.3	274.6
HAYWOOD-3	24	24	24	18	18	1.6	6	31.6	29	27.3	46	24.6	274.3
HAYWOOD-4	24	24	24	18	18.3	1.3	6	32.3	28	27.3	46	24.3	273.6

Table 1. Performance comparison A (ICCMA'15/ICCMA'17 benchmarks, number of solved instances) of our approach (HAYWOOD) with the best three reduction-based solvers from ICCMA'17 (pyglaf, argmat-sat, goDIAMOND), the best reduction-based solver from ICCMA'15 (ASPARTIX-D), and all direct solvers from ICCMA'17 (ArgTools, heureka, EqArgSolver); our approach is parametrised with four versions for $M = 2|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-1), $M = 6|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-2), $M = 10|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-3), and $M = 14|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-1); every cell gives the number of correctly solved instances within the time limit of 10 minutes (averaged over 3 runs)

increasing values for M. On the other hand, Figure 2 (b) shows how many graphs could be solved with three different but fixed values for M and increasing values for G.

We see that very low values for M are generally decreasing the performance of the solver significantly while increasing the parameter G has a significant positive impact with the optimal value being in between 0.8 and 0.9 for the test set (note that very large values for G such as 0.999 or even 1 result in almost zero solved instances as greedy moves alone are not sufficient; this is not visible from Figure 2 (b) alone). Moreover, for medium and high values of G such as $G = 0.8$ the value of M has almost no impact. In fact, many graphs from the test set are solved without a single restart.

From the result of the parameter optimisation we decided to take four variants of HAYWOOD into the performance comparison: for $M = 2|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-1), $M = 6|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-2), $M = 10|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-3), and $M = 14|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-4).

5.3. Performance comparison

The next experiment (performance comparison A) compares the performance of our four solver variants with the seven state-of-the-art solvers mentioned earlier on the actual benchmark graphs from both ICCMA'15 and ICCMA'17. More precisely, the solvers we included are pyglaf 0.2, argmat-sat 1.0.0, goDIAMOND 0.6.6, ASPARTIX-D (ICCMA'15 version), ArgTools 2.0.0, heureka 0.2, and EqArgSolver 2.76. We used all test sets 1-5, 7-9 from ICCMA'15[9] and group B from ICCMA'17 (the latter contained the graphs used for the stable semantics track) but removed all graphs without a stable extension (in particular we removed the complete test set 7 from ICCMA'15 because of this reason). We ended up with 322 benchmark graphs in total[10].

[9]Note that test set 6 was not used in ICCMA'15

[10]The used instances can be downloaded from `http://mthimm.de/misc/slsinst.zip`

	Barabási-Albert random graphs with varying number of arguments										Sum
	10k	20k	30k	40k	50k	60k	70k	80k	90k	100k	
pyglaf	17	23	28	36	41	51	67	70	77	82	492
goDIAMOND	31	56	76	111	140	168	213	259	300	335	1689
ASPARTIX-D	12	17	23	30	35	44	62	61	70	76	430
ArgTools	47	168	369	644	1004	1447	2017	2629	3359	4130	15814
HAYWOOD-1	7	7	9	12	16	19	21	22	29	33	175
HAYWOOD-2	6	5	9	11	11	13	17	21	23	27	143
HAYWOOD-3	6	6	9	8	11	13	17	19	24	27	140
HAYWOOD-4	5	7	7	9	10	13	20	20	24	25	140

Table 2. Performance comparison B (Barabási-Albert graphs, total runtime) of our approach (HAYWOOD) with the best three reduction-based solvers from ICCMA'17 (pyglaf, argmat-sat, goDIAMOND), the best reduction-based solver from ICCMA'15 (ASPARTIX-D), and all direct solvers from ICCMA'17 (ArgTools, heureka, EqArgSolver); all solvers which had at least one timeout on any instance have been removed (these were argmat-sat, heureka, EqArgSolver); our approach is parametrised with four versions for $M = 2|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-1), $M = 6|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-2), $M = 10|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-3), and $M = 14|\mathsf{Arg}|, G = 0.8$ (HAYWOOD-1); every cell gives the total runtime in seconds (rounded) on all correctly solved instances within the time limit of 10 minutes (averaged over 3 runs)

We asked every solver to determine a stable extension and set a timeout of 10 minutes. We repeated this experiment three times and took the average over these three runs. Table 1 shows the results in terms of number of solved instances within the time limit (the larger the better). We can see that the performance of our approach does not reach the performance of the best reduction-based approaches but beats two of the three direct approaches (note also that the used version of ArgTools is an updated version and not the one used at ICCMA'17).

However, stochastic local search algorithms usually show their true advantage when applied to random instances [2, Chapter 6]. So we did another experiment (performance comparison B) on benchmark graphs generated using the Barabási-Albert model (this models showed the most significant impact compared to other random models such as Erdős-Rényi and Watts-Strogatz). In particular, we generated 10 graphs with 10,000 to 100,000 arguments (in steps of 10,000 arguments) using AFBenchGen2[11] with the `BA_WS_probCycles` parameter set to 0.9 (this parameter controls the probability of an argument being part of a cycle). Again we asked every solver to determine a stable extension, set a timeout of 10 minutes, and repeated this experiment 3 times, taking the average over number of solved instances and runtimes. Almost all solvers were able to solve all 100 instances in all three runs, the remaining ones (argmat-sat, heureka, EqArgSolver) were not considered further. Table 2 shows the results in terms of total runtime (in seconds) over all solved instances (the lower the better).

The results of the latter experiment show a significant performance increase of our approach. While the best direct approach needed a bit over 4 hours to compute all solutions and the best reduction-based approach a bit over 7 minutes, our approach only needed about 2-3 minutes, therefore quite drastically outperforming even the best direct approaches. Although the general performance of our approach is mid-range, the drastic performance jump on specific instances such as instances generated by the Barabási-Albert model, shows that the approach is indeed competitive.

[11]`https://sourceforge.net/projects/afbenchgen/`

6. Summary and Future Work

We developed stochastic local search algorithms for the problem of finding a single stable extension of an abstract argumentation framework. By means of an empirical evaluation we showed that our approach is competitive and outperforms state-of-the-art in certain random graph models.

In this paper we focused on the problem of finding a single stable extension but the general algorithm can be applied to other problems and semantics as well, in particular concerning problems of deciding credulous or skeptical acceptance (and less to the problem of enumerating extensions). Future work is about exploring the feasibility of the approach to these other problems.

References

[1] P. Baroni, M. Caminada, and M. Giacomin. An introduction to argumentation semantics. *The Knowledge Engineering Review*, 26(4):365–410, 2011.

[2] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[3] Martin W.A. Caminada and Dov M. Gabbay. A Logical Account of Formal Argumentation. *Studia Logica*, 93(2–3):109–145, 2009.

[4] F. Cerutti, S.A. Gaggl, M. Thimm, and J.P. Wallner. Foundations of implementations for formal argumentation. In P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, editors, *Handbook of Formal Argumentation*, chapter 15. College Publications, 2018.

[5] F. Cerutti, M. Giacomin, and M. Vallati. ArgSemSAT: Solving argumentation problems using SAT. In *Proceedings of COMMA'14*, 2014.

[6] G. Charwat, W. Dvořák, S.A. Gaggl, J.P. Wallner, and S. Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artificial Intelligence*, 220:28–63, 2015.

[7] P.M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

[8] W. Dvořák and P.E. Dunne. Computational problems in formal argumentation and their complexity. In P. Baroni, D. Gabbay, M. Giacomin, and L. van der Torre, editors, *Handbook of Formal Argumentation*, chapter 15. College Publications, 2018.

[9] U. Egly, S.A. Gaggl, and S. Woltran. Answer-set programming encodings for argumentation frameworks. Technical Report DBAI-TR-2008-62, Technische Universität Wien, 2008.

[10] D. Gabbay and O. Rodrigues. A self-correcting iteration schema for argumentation networks. In *Proceedings of COMMA'14*, 2014.

[11] S.A. Gaggl and N. Manthey. ASPARTIX-D: ASP argumentation reasoning tool - Dresden. In *System Descriptions of ICCMA'15*. ArXiv, 2015.

[12] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer Set Solving in Practice*. Morgan & Claypool Publishers, 2012.

[13] N. Geilen and M. Thimm. Heureka - a general heuristic backtracking solver for abstract argumentation. In *Proceedings of TAFA'17*, 2017.

[14] D. Niu, L. Liu, and S. Lü. New stochastic local search approaches for computing preferred extensions of abstract argumentation. *AI Communications*, 31(4):369–382, 2018.

[15] S. Nofal, K. Atkinson, and P.E. Dunne. Looking-ahead in backtracking algorithms for abstract argumentation. *International Journal of Approximate Reasoning*, 78:265–282, 2016.

[16] O. Rodrigues. A forward propagation algorithm for the computation of the semantics of argumentation frameworks. In *Proceedings of TAFA'17*, 2017.

[17] B. Selman, H.J. Levesque, and D.G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of AAAI'92*, 1992.

[18] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Second DIMACS Implementation Challenge*. AMS, 1996.

[19] M. Thimm and S. Villata. The first international competition on computational models of argumentation: Results and analysis. *Artificial Intelligence*, 252:267–294, 2017.