Introducing EqArgSolver: an argumentation solver using equational semantics

Odinaldo RODRIGUES
King's College London, Department of Informatics, email: odinaldo.rodrigues@kcl.ac.uk

Abstract. In this paper we introduce EqArgSolver, an argumentation solver using equational semantics. EqArgSolver is based on the prototype GRIS submitted to the first International Competition on Computational Models of Argumentation, but provides two major advancements over GRIS. Firstly, it uses the much more efficient discrete version of the Gabbay-Rodrigues Iteration Schema [7], and secondly, it requires at most 1/8 of the memory footprint per node used by GRIS. Experimental results show that EqArgSolver runs up to 12× faster than GRIS and is able to handle a much larger class of argumentation graphs.

Keywords. Numerical argumentation, argumentation solvers, numerical methods

1. Introduction

One of the most important tasks in argumentation theory is the reasoning about the status of arguments in a given argumentation framework $\langle S, R \rangle$. This can be done according to several semantics mainly using the concepts of an *extension* or a labelling function λ that assigns a label from $\{\mathbf{in}, \mathbf{out}, \mathbf{und}\}$ to each argument in S. The two concepts are interchangeable, as an extension can be obtained from a labelling function and vice-versa. A third way of giving semantics is via Gabbay's *equational approach* [5] that sees an argumentation framework as a graph generating equations. Again, for some classes of equations, there is a direct correspondence between the solutions to the equations and the concepts of extensions and labelling functions via an appropriate mapping of the numerical values.

In [6], Gabbay and Rodrigues proposed the Gabbay-Rodrigues Iteration Schema, a numerical iterative method that can be used in the computation of extensions of argumentation frameworks in the traditional Dung sense. The schema takes an initial assignment of values $V_0 : S \longmapsto [0, 1]$ and produces a new assignment V_{i+1} in terms of the values at iteration i. The values in the schema eventually converge and in the limit of the sequence $V_0, V_1, \ldots$ we can construct a complete extension simply by taking the nodes with value 1. The schema was used in the prototype GRIS [10] submitted to the first International Competition on Computational Models of Argumentation [3,4].

One disadvantage of the use of the schema in the computation of argumentation semantics is the need to determine the values in the limit of the sequence. In

a computer, this can only be approximated by iterating sufficiently many times until one can be satisfied that the approximated values correspond to the values in the limit. With this in mind, GRIS performed relatively well as a proof of concept, but left open a number of theoretical and technical questions about the use of the schema in more intensive argument computation tasks.

In [7], Gabbay and Rodrigues proposed a simplified version of the schema called the *Discrete Gabbay-Rodrigues Iteration Schema* that under special conditions is guaranteed to converge in time linear to the number of nodes in the argumentation network. Besides requiring many fewer iterations than the full-fledged schema, the discrete version requires less computation effort at each step and its implementation can be done on a much lower memory footprint.[1]

This paper describes the basic theoretical underpinnings of the discrete version of the schema and the technical details involved in its implementation in EqArgSolver. In experimental results, EqArgSolver performed up to 12× faster than GRIS and was able to handle much larger argumentation networks.

As for GRIS, EqArgSolver can be used in the solution to the following classes of problems. Given an argumentation network $\langle S, R\rangle$: to produce one or all of the extensions of the network under the grounded and preferred semantics; and given an argument $X \in S$ to decide whether X is accepted credulously or sceptically according to one of these two semantics. Problems to EqArgSolver must be submitted according to `probo`'s syntax (see [3]). EqArgSolver is not currently able to handle problems in the *complete* and *stable* semantics, although these enhancements are in the development pipeline (see Section 6).

The rest of the paper is organised as follows. Section 2 provides a basic background, including the main theoretical underpinnings on which EqArgSolver is based. Section 3 provides an overview of the implementation of EqArgSolver. This is followed by a comprehensive example illustrating the computation steps in Section 4. The performances of EqArgSolver and GRIS are compared in Section 5. Finally, Section 6 concludes the paper giving some directions for future work.

2. Background

In [6], Gabbay and Rodrigues proposed the so-called *Gabbay-Rodrigues Iteration Schema* defined as follows. Let $\langle S, R\rangle$ be an argumentation network, $Att(X) = \{Y \in S \,|\, (Y, X) \in R\}$, and V_0 be an initial assignment of values from $[0, 1]$ to the nodes in S. Let for each $X \in S$, $MA_i(X) = \max_{Y\in Att(X)}\{V_i(Y)\}$ and let the equation below define the value of the node X at all subsequent iterations (i.e., $V_1, V_2, \ldots$):

$$V_{i+1}(X) = (1 - V_i(X)) \cdot \min\{1/2, 1 - MA_i(X)\} + V_i(X) \cdot \max\{1/2, 1 - MA_i(X)\}$$

To facilitate the explanations in the rest of this paper, if $V : S \longmapsto [0, 1]$ is any assignment of values from $[0, 1]$ to a set of arguments S, then we define the sets $in(V) = \{X \in S \mid V(X) = 1\}$ and $out(V) = \{X \in S \mid V(X) = 0\}$.

Take the initial configuration of values V_0 and for each $i \geq 0$ and $X \in S$, let the value $V_{i+1}(X)$ be calculated from $V_i(X)$ according to the schema above. In [6], Gabbay and Rodrigues showed that, for all nodes X, the values in the

[1] In a 64-bit machine, $\frac{1}{8}$-th of the space required by GRIS.

sequence $V_0(X), V_1(X), V_2(X), \ldots$, converge to the so-called equilibrium value of X, denoted $V_e(X) =^{def} \lim_{i\to\infty} V_i(X)$. Gabbay and Rodrigues further showed that the nodes with $V_e(X) = 1$ correspond to a complete extension. As for [2], this extension corresponds to the minimal complete extension containing the maximal admissible subset of $in(V_0)$. If $V_0(X) = 1/2$ for all $X \in S$, then the set $in(V_0) = \varnothing$ is the minimally (trivially) admissible subset of S and, consequently, the set $in(V_e)$ will correspond to the minimal complete extension of $\langle S, R\rangle$, i.e., its grounded extension. If the argumentation network is decomposed into layers, the schema can also be used to propagate the values of a solution of a layer to layers that depend on them. These are its main computational tasks in the prototype GRIS [10].

Obviously, in order to use the schema, one needs to find a way to determine the equilibrium values in a finite number of computations. In GRIS, each iteration i computes the values $V_i(X)$ of all nodes X from the values $V_{i-1}(X)$. The more one iterates, the more this value approximates the real value $V_e(X)$. As a result, the number of nodes and edges in the network and the precision required in the approximation of the limit values all have a direct effect on GRIS' overall execution time. Optimising the efficiency of the approximation of the value $V_e(X)$ was hence an important objective of any future implementation using the equational semantics.

In [7], Gabbay and Rodrigues proposed a discrete version of the schema. Unlike its full-fledged counterpart, the discrete version cannot be used with any set of initial values, but under certain initial values it converges to the exact same values of its full-fledged version without the need for approximation and using much simpler computation steps. It is therefore an ideal replacement. This schema is presented below.

Definition 2.1 *Let $\mathcal{N} = \langle S, R\rangle$ be an argumentation framework and V_0 be an assignment of values from $\{0, 1/2, 1\}$ to the nodes in S. The* Discrete Gabbay-Rodrigues Iteration Schema *is defined by the following system of equations* ($\mathrm{T}_\mathbf{d}$)*, where the value $V_{i+1}(X)$ of each node in iteration $i+1$ is defined in terms of the values of the nodes in iteration i as follows:*

$$V_{i+1}(X) = 1 - \max_{Y \in Att(X)} \{V_i(Y)\} \tag{$\mathrm{T}_\mathbf{d}$}$$

The following correspondence between numerical values and argument labels will prove useful.

Definition 2.2 (Caminada/Gabbay-Rodrigues Translation) *A labelling function λ and a valuation function V can be inter-defined by identifying the value 1 with the label* **in***, the value 0 with the label* **out***, and the value $1/2$ with the label* **und***.*

Caminada and Modgil's algorithm [9] does something similar, but with labels instead. Their algorithm relies on searches through sets of nodes. Equation ($\mathrm{T}_\mathbf{d}$), on the other hand, can be implemented in terms of native data types and operations without the need for searches. This makes it very efficient. Because of the correspondence between numerical values and labels, it is, of course, also possible to write ($\mathrm{T}_\mathbf{d}$) in terms of labels by replacing subtraction and max with appropriate conditional statements and assignments.

In what follows, we will refer to a valuation V defined from a labelling function λ by V_λ and to a labelling function λ obtained from a valuation V by λ_V.

The discrete version of the schema can be used in several computational tasks. For instance, it allows to quickly check whether a given assignment yields a complete extension, as seen below.

Theorem 2.1 *[7, Theorem 4] Let $\langle S, R \rangle$ be an argumentation framework and V_i an assignment of values from $\{0, 1/2, 1\}$ to the nodes in S. λ_{V_i} is a complete labelling function if and only if $V_{i+1}(X) = V_i(X)$, for all $X \in S$.*

Remember that there is a direct correspondence between the nodes assigned the label **in** by a complete labelling function λ and a *complete* extension. It is easy to see that the complexity of the checks in Theorem 2.1 is not higher than that of the checks performed using labelling functions and that the checks can be implemented in a single loop through all relevant nodes. However, the checks in Theorem 2.1 can be implemented very efficiently because of the arithmetic nature of the operations.

Gabbay and Rodrigues have shown in [7] that under the particular initial assignment all-**und**, the discrete and full-fledged versions of the schema will *always* converge to the same values. As a result, the discrete Gabbay-Rodrigues Iteration Schema if started with the all-**und** initial assignment will compute the grounded extension. The discrete version is much more efficient, because it converges without approximation and the computations at each iteration are much simpler. In Section 3 we discuss how they can be implemented in terms on positive integers only.

More details of the utilisation of the discrete schema are discussed in the following section.

2.1. Using the Discrete Gabbay-Rodrigues Iteration Schema in the Computation of Argumentation Semantics

It is well known that several semantics of an argumentation framework $\langle S, R \rangle$ can be computed by decomposing the network into *layers* [1,8]. In simple terms a layer can be thought of as the maximum collection of nodes whose acceptability status can be collectively calculated in the same step. This can be better understood in terms of the strongly connected components (SCCs) of the graph. Loosely speaking, a trivial SCC is an argument not involved in any cycle; and a non-trivial SCC C of $\langle S, R \rangle$ is a maximal subset $C \subseteq S$, such that for all nodes $X, Y \in C$, there is a path from X to Y and a path from Y to X via the attack relation. Note that self-attacking nodes always belong to some non-trivial SCC. Layer 0, by definition, contains all non-trivial SCCs whose attackers only belong to the SCC itself (called "initial" in [1]) plus all of the trivial SCCs whose nodes have no attackers, plus all other trivial SCCs whose attackers are trivial SCCs also in Layer 0. Layer 1 contains all SCCs having all attackers either in layer 0 or layer 1 and at least one attacker in layer 0, and so forth.

For example, layer 0 of the network in Figure 1 contains the two non-trivial SCCs $SCC_1 = \{A, B\}$ and $SCC_2 = \{C, D\}$ and the collection of trivial SCCs $SCC_3 = \{G\}$ and $SCC_4 = \{H\}$. Layer 1 contains the non-trivial SCC $SCC_5 =$

$\{E, F\}$. The intuition here is that the computation of the statuses of the nodes in a layer of higher order depends on the computation of the statuses of the nodes in the layers of lower orders. In our example, one can see that the statuses of E and F can only be computed given the statuses of B and C. The statuses of the nodes in the trivial SCCs G and H can be computed in sequence within the same layer. An excellent explanation of this technique can be found in [8].

Figure 1. A complex argumentation framework.

In the computation of the *grounded* semantics, all that is needed in the calculation of the result of a non-trivial SCC $\mathcal{C}$ is the propagation of the results of the nodes in the previous layers on which $\mathcal{C}$ depends (if there are any). For example, since there are no prior dependencies in the SCCs in layer 0, the first SCCs involved in the computation of the semantics of the network of Figure 1 are SCC_1, SCC_2, SCC_3 and SCC_4. The only undisputed nodes in any network are the ones with no attackers. These get value **in**. The nodes in cycles at layer 0 all get value **und**. These values are then propagated down the layer until all nodes in the layer get a value.

With this in mind, the nodes in the non-trivial SCCs of layer 0 get the (unique) solution: $A = \textbf{und}, B = \textbf{und}, C = \textbf{und}, D = \textbf{und}$. Since G has no attackers, it gets value **in**. When this value is propagated to H, it gets value **out**, yielding the solution $G = \textbf{in}, H = \textbf{out}$ for the trivial SCCs in layer 0.

In [7], Gabbay and Rodrigues showed that we can do the same task *numerically*, if *i)* We use the correspondence in Definition 2.2; *ii)* We give the initial value $1/2$ to all nodes; and *iii)* We calculate final values for the nodes using the *Discrete Gabbay-Rodrigues Iteration Schema* (Definition 2.1).[2]

The schema guarantees a correspondence between the nodes in the grounded extension and those with final value 1.

Note that since the non-trivial SCCs in a layer are independent from each other, we can use Equation ($\mathrm{T}_\mathbf{d}$) in each one of them independently[3] to obtain all the partial solution(s) to a layer. The computation of layers of higher order pose no problems either. All that needs to be done is to propagate the final values of the nodes of lower order in a given solution to the nodes that they attack.

The example below illustrates these ideas.

Example 2.1 *Consider* $SCC_1 = \{A, B\}$ *and give its nodes initial value* $1/2$ *at iteration* 0*. The sequence of values in the discrete scheme immediately converges since the values of all nodes in the SCC remain the same at iteration* 1*. Therefore* $1/2$ *is the final values of* A *and* B*. The reasoning for the* $SCC_2 = \{C, D\}$ *is the same with both nodes also getting final value* $1/2$*.*

Now consider the collection of trivial SCCs SCC_3 *and* SCC_4*, whose nodes are also given initial value* $1/2$ *at iteration* 0*. The value of* G *at iteration* 1 *becomes* 1*, whereas the value of* H *remains* $1/2$*. At iteration* 2*, the value of* G *remains* 1 *and the value of* H *becomes* 0*. These values then remain the same at every*

[2]The sequence will converge in time linear to the number of nodes in the SCC being computed.

[3]In fact, we could do this in parallel, although this is not currently used in EqArgSolver.

successive iteration, giving final solution $G = 1$ and $H = 0$. This completes the computation of this layer.

Proceeding to layer 1, it is easy to see that E and F also get final value $1/2$ and therefore the unique solution for the whole network is:

$$A = \textbf{und}, B = \textbf{und}, C = \textbf{und}, D = \textbf{und}, G = \textbf{in}, H = \textbf{out}, E = \textbf{und}, F = \textbf{und}$$

This corresponds exactly to the grounded extension of the network.

Given a solution to a layer, the discrete schema correctly propagates the solution's values to the appropriate nodes in a layer of higher order. For instance, in the calculation of the solution to layer 1, the relevant nodes in layer 0 are B and C. Therefore, given partial solution $A = \textbf{out} \equiv 0$, $B = \textbf{in} \equiv 1$, $C = \textbf{out} \equiv 0$ and $D = \textbf{in} \equiv 1$, we start the schema with initial values $B = 1$, $C = 0$, and $E = F = 1/2$. In iteration 1, we get that $E = 0$, $F = 1/2$. In iteration 2, we get $E = 0$ and $F = 1$. The values then converge and we get the final solution $A = \textbf{out} \equiv 0$, $B = \textbf{in} \equiv 1$, $C = \textbf{out} \equiv 0$, $D = \textbf{in} \equiv 1$, $E = \textbf{out} \equiv 0$ and $F = \textbf{in} \equiv 1$, which yields an extension.

The above ideas constitute the basis of the GR Grounder module (see Fig. 2). A second module is responsible for generating preferred solutions. This works basically by applying the grounder to a SCC and then checking whether nodes with undecided values remain. If this is the case, then we may attempt to break cycles down by making some nodes in the cycle **in** and then propagating the results to layers of higher order. This component of the solver follows a similar algorithm to that of Sanjay and Caminada [9]. This will become clearer in the comprehensive example given in Section 4.

3. System Overview

EqArgSolver is implemented in C++ using basic C++11 data structures. No special libraries for solving equations are needed, since the arithmetic operations in the discrete version of the Gabbay-Rodrigues Iteration Schema are trivial.

As we mentioned in the Introduction, EqArgSolver is currently limited to the grounded and preferred semantics. The basic workflow for the computations involving problems in these semantics is depicted in Fig. 2, with the exception that in the grounded semantics, intermediate preferred solutions do not need to be generated and so the *preferred solutions generator* is not used and the *partial preferred solutions* data store only contains one solution. All solutions are kept in memory.

The solver receives a number of parameters as arguments in the command line following `probo`'s syntax [3]. A validator module checks that the problem definition is sound, in which case the solver proceeds to computing the strongly connected components (SCCs) of the network using a specially modified version of Tarjan's algorithm [11] and arranging them into layers that can be used in successive computation steps as described in [8]. Once the network is decomposed into layers, for decision problems involving the acceptance of an argument, the solver stops at the last layer containing the input argument. This allows for instance to give a "No" answer when an argument is found not to be accepted in a partial

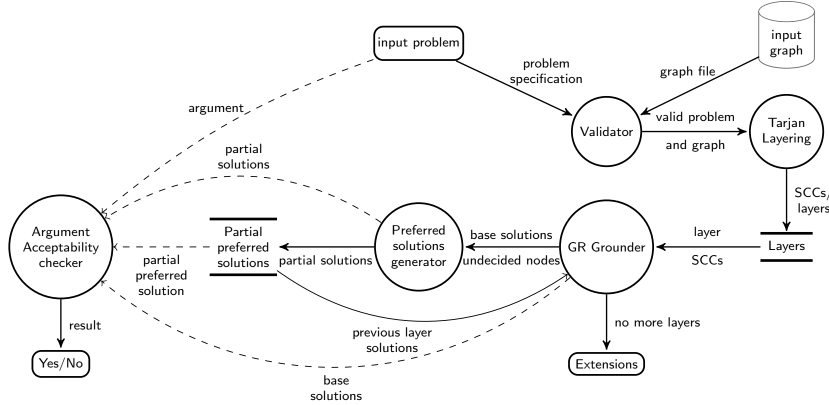

Figure 2. Basic workflow of the preferred semantics calculator.

solution generated in a skeptical decision problem (or "Yes" if the argument is accepted in a partial solution in a credulous decision problem).

We now detail some of the particularities in the computation of the grounded and preferred semantics.

Grounded Semantics. For the computation of the solutions to the problems involving the grounded semantics, the decomposition of the network into layers is not strictly necessary. The Grounder module based on Equation ($\mathrm{T}_{\mathbf{d}}$) can be applied to the entire network at once and the nodes with equilibrium value 1 will correspond to its grounded extension. However, since the decomposition of the network into layers can be performed very efficiently in our implementation, this extra cost is offset by gains obtained through the computation by layers in all but a few special cases. Our strategy for the grounded semantics is then to feed the result of each layer into the next layer's computation until either all layers are computed (e.g., in an enumeration problem) or we reach the minimum necessary depth for an answer (e.g., in an argument acceptance check).

Preferred Semantics. As we have seen, the solution of problems involving the preferred semantics involves the computation of partial network solutions for each layer. Each of these solutions needs to be propagated and adequately combined. The key point in EqArgSolver's implementation is that after the grounder module is invoked for a particular SCC (using all required values from previous layers), some nodes in the SCC may still be left with value **und**. These nodes could potentially be assigned the value 1 in a preferred extension. So our (naive) implementation at this stage simply assigns the value 1 to all such nodes and then corrects illegal values in a manner which is the numerical counterpart to the Caminada and Modgil's labelling-based algorithm [9]. The results of every candidate solution are then propagated to the next layer using the grounder again and the whole process repeats. Obviously, these results need to be kept in memory until the computation is finished (alternative implementations may save on memory requirements but involve more computation). In GRIS, the values in the sequence were real

numbers taken from the unit interval $[0, 1]$, which in the implementation were represented as a `double` native data type. In a 64-bit machine, this required at least 8 bytes per node plus the memory requirements of the data structure used to associate values to nodes (the exact details of the representation are not relevant here). An extra advantage of the use of the discrete version is that we only have values in $\{0, \frac{1}{2}, 1\}$. In order to avoid multiplication involving the rational number $\frac{1}{2}$, all equations are multiplied by 2, shifting the values to the set $\{0, 1, 2\}$, where $0 \equiv$ **out**, $1 \equiv$ **und** and $2 \equiv$ **in**. This is not only more computationally efficient (since it only involves integer values), but also can be represented with as few as 4 bits per node. In order to avoid the use of custom data types, EqArgSolver uses the smallest available native integer data type, `char`, which requires 8 bits.

There is still ample scope for optimisation. For instance, in decision problems, a careful analysis of the argument involved may identify partial solutions of particular interest without the need to generate all partial solutions (the blind generation of partial solutions can quickly exhaust resources in highly complex networks).

Solutions to the problems in the complete and stable semantics are not currently supported may also be computed with appropriate modifications. A quick and easy way to do so for the complete semantics is to generate all intermediate solutions by invoking the grounder module at key points (each invocation resulting in an intermediate complete extension). Similarly, a simple check at the end of the preferred solutions generation can identify if an extension is stable. Similarly, it can quickly abandon a partial solution when nodes are left with value **und**. These considerations will be taken into account in the development of the full version of EqArgSolver for the next international competition.

4. A Comprehensive Example

This section provides a comprehensive example illustrating the ideas discussed so far. For this, the sample network shown in Figure 1 will be used again.

Initially, the graph is decomposed into SCCs and each SCC is assigned a layer. In this example, the decomposition generates two layers as explained in Section 2.1. These two layers are then processed as follows.

Phase 1: Apply the discrete version of the schema to the SCCs SCC_1, SCC_2 and SCC_3 and SCC_4 in layer 0 giving initial value $1/2$ to all nodes.

The tables below show how the values of the nodes evolve through the iterations of the schema. The subscript i in V_i indicates the iteration number. A value in bold has converged and will no longer change. The computation can stop when all values in a SCC have converged.

Node	V_0	V_1
A	$1/2$	**$1/2$**
B	$1/2$	**$1/2$**

Node	V_0	V_1
C	$1/2$	**$1/2$**
D	$1/2$	**$1/2$**

Node	V_0	V_1	V_2	V_3
G	$1/2$	1	**1**	**1**
H	$1/2$	$1/2$	0	**0**

If the problem at hand only involves the grounded semantics, all we have to do next is to run the discrete schema again giving value $1/2$ to all nodes in layer 1 and keeping the values computed above for the nodes in layer 0. It is easy to check that the values of E and F will remain $1/2$. Since layer 1 is the last layer, this is the only solution and the grounded extension contains the only

node with value 1, i.e., G. The upshot of all this is that, strictly speaking, there is no need to decompose the graph into layers in grounded semantics problems. The grounded extension can be computed for the whole graph by calculating the final values of all nodes in one fell swoop. In earlier experiments, the difference in execution time for the computation of the grounded extension with or without decomposition into layers was negligible. However, decomposition offers a definite advantage in decision problems involving large graphs and arguments appearing in shallow layers. Of course one can only identify the layer of an argument by performing the decomposition itself and since the decomposition algorithm is very efficient, this is the strategy used in EqArgSolver.

We now proceed to explain the steps taken for the *preferred* semantics.

Phase 2*:* Generate partial preferred solutions to layer 0.

Some SCCs may still be left with undecided nodes after applying the discrete schema to a layer. In this example, both SCC_1 and SCC_2 are left with undecided nodes. For each such SCC, a version of Caminada and Modgil's algorithm for finding preferred extensions is then applied (see [9]). This will generate 2 complimentary solutions for each SCC with exactly one node labelled **in** in each. Combining these solutions "horizontally" gives us the 4 partial preferred solutions below to layer 0 of the argumentation framework (the superscript indicates the layer):

$\boldsymbol{S_1^0}$: $A=1$, $B=0$, $C=1$, $D=0$, $G=1$, $H=0$; $\boldsymbol{S_2^0}$: $A=1$, $B=0$, $C=0$, $D=1$, $G=1$, $H=0$
$\boldsymbol{S_3^0}$: $A=0$, $B=1$, $C=1$, $D=0$, $G=1$, $H=0$; $\boldsymbol{S_4^0}$: $A=0$, $B=1$, $C=0$, $D=1$, $G=1$, $H=0$

Note that for decision problems involving the acceptance of an argument belonging to layer 0 the correct answer can already be given without the need to proceed to layer 1.

Phase 3*:* Propagate the partial solutions found for layer 0 into layer 1. We say that we "ground" layer 1 with a base solution found for layer 0. The equations for the nodes in layer 1 are: $V_{i+1}(E) = 1 - \max\{V_i(B), V_i(F)\}$; and $V_{i+1}(F) = 1 - \max\{V_i(C), V_i(E)\}$.

As before, the values of all nodes in the layer itself are initialised to $1/2$, giving $V_0(E) = V_0(F) = 1/2$. Since the equations involve nodes belonging to layer 0, they are computed using as initial values the final values of these nodes found in each partial solution to layer 0. These partial solutions are called the *base solutions* of the solutions to this layer. This yields $\boldsymbol{S_1^1}$–$\boldsymbol{S_4^1}$ below:

$\boldsymbol{S_1^1}$	V_0	V_1	V_2	V_3
B	**0**	**0**	**0**	**0**
C	**1**	**1**	**1**	**1**
E	$1/2$	$1/2$	1	**1**
F	$1/2$	0	**0**	**0**

$\boldsymbol{S_2^1}$	V_0	V_1	V_2
B	**0**	**0**	**0**
C	**0**	**0**	**0**
E	$1/2$	$\boldsymbol{1/2}$	$\boldsymbol{1/2}$
F	$1/2$	$\boldsymbol{1/2}$	$\boldsymbol{1/2}$

$\boldsymbol{S_3^1}$	V_0	V_1	V_2
B	**1**	**1**	**1**
C	**1**	**1**	**1**
E	$1/2$	0	**0**
F	$1/2$	0	**0**

$\boldsymbol{S_4^1}$	V_0	V_1	V_2	V_3
B	**1**	**1**	**1**	**1**
C	**0**	**0**	**0**	**0**
E	$1/2$	0	**0**	**0**
F	$1/2$	$1/2$	1	**1**

Phase 4*:* As before, for each solution to a SCC in layer 1 where undecided nodes remain, search for possible partial preferred solutions. Any solution found in this way needs to be "vertically" combined with the base solution that originated it.

In our example, only $\boldsymbol{S_2^1}$ leaves some nodes with undecided values in SCC $SCC_5 = \{E, F\}$. There are 2 partial preferred solutions to this SCC with values $E = 1$ and $F = 0$, and $E = 0$ and $F = 1$, respectively. Combining them vertically with the solution $\boldsymbol{S_2^0}$ yields the two preferred solutions $\boldsymbol{S_{2.1}^1}$ and $\boldsymbol{S_{2.2}^1}$. The set of all preferred solutions with their corresponding preferred extensions is then given below.

$\boldsymbol{S_1^1}$: $A = 1\ B = 0\ C = 1\ D = 0\ E = 1\ F = 0\ G = 1\ H = 0 \Rightarrow \{A, C, E, G\}$
$\boldsymbol{S_{2.1}^1}$: $A = 1\ B = 0\ C = 0\ D = 1\ E = 1\ F = 0\ G = 1\ H = 0 \Rightarrow \{A, D, E, G\}$
$\boldsymbol{S_{2.2}^1}$: $A = 1\ B = 0\ C = 0\ D = 1\ E = 0\ F = 1\ G = 1\ H = 0 \Rightarrow \{A, D, F, G\}$
$\boldsymbol{S_3^1}$: $A = 0\ B = 1\ C = 1\ D = 0\ E = 0\ F = 0\ G = 1\ H = 0 \Rightarrow \{B, C, G\}$
$\boldsymbol{S_4^1}$: $A = 0\ B = 1\ C = 0\ D = 1\ E = 0\ F = 1\ G = 1\ H = 0 \Rightarrow \{B, D, F, G\}$

In the next section, we discuss the results of some experiments we have done in order to compare the performance gain obtained via the use of the discrete schema over its full-fledged version.

5. Empirical Evaluation

The performance of EgArgSolver was compared with that of GRIS' using `probo`'s benchmark suite [3]. A number of graphs were randomly generated using `probo`'s *Grounded*, *SCC*, and *Stable* generators. The Grounded generator generates graphs that are likely to have a large grounded extension; the SCC generator generates graphs that contain many SCCs; and the Stable generator generates graphs that are likely to contain many stable, preferred and complete extensions. In our experiments, graphs generated by the SCC and Stable generators were much harder to produce (via `probo`) and solve (via EqArgSolver/GRIS). This is because of the intrinsic nature of these graphs with respect to the number and type of SCCs they contain. For this reason, the benchmarks were limited to a small set of 10 randomly generated graphs using the SCC and Stable generators. However, we randomly generated 50 graphs of larger size using the Grounded generator – see Table 1 for details. At generation time, the generators also eliminate graphs that are deemed 'too simple' (using `probo`'s terminology), according to minimum requirements the graphs must satisfy. In decision problems, we checked 10 randomly selected arguments against each of the graphs in each category (i.e., 10 arguments were checked against each of the 50 graphs generated by the Grounded generator, and 10 arguments were checked against each of the 10 graphs generated by each of the SCC and Stable generators). The average execution times of GRIS and EqArgSolver in the tests performed are given in Figures 3 and 4. Following `probo`'s terminology, each problem in the tables is identified by a *prefix*, where GR stands for the grounded semantics and PR stands for the preferred semantics; and a *suffix*, where EE stands for "enumerate all extensions"; SE stands for "give one extension"; DC stands for "decide if an argument is accepted in any extension"; and DS for "decide if an argument is accepted in all extensions".

Class	graphs	avg. nodes	avg. edges	decision problems
Grounded	50	945	9020	10 × 50
SCC	10	317	4800	10 × 10
Stable	10	277	2494	10 × 10

Table 1. Details of the graphs used in the benchmarking of EqArgSolver.

All test results show a large performance gain of EqArgSolver over GRIS. The largest gains, as expected, are in the problems that depend more directly on the grounding module. In Figure 3, we can see that EqArgSolver is up to 12× faster than GRIS both in enumeration and decision problems in the graphs

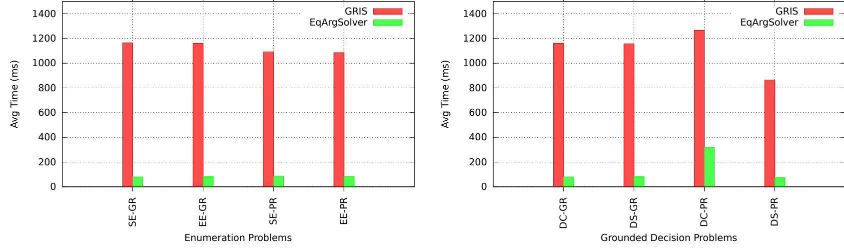

Figure 3. Average execution times for enumeration (L) and decision problems (R) in graphs using `probo`'s Grounded generator.

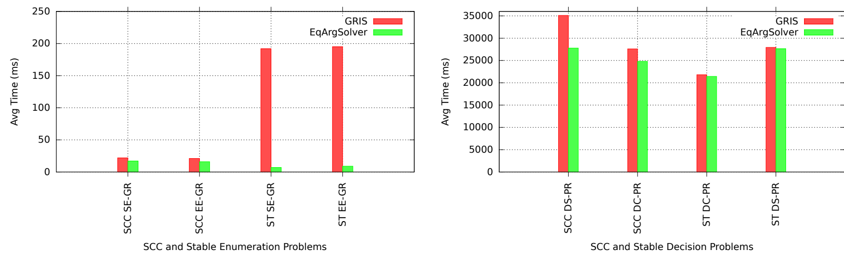

Figure 4. Average execution times using `probo`'s SCC and Stable (ST) generators. (L) shows compartive times for the EE and SE enumeration problems in the grounded semantics and (R) shows the comparative times for the DC and DS decision problems in the preferred semantics.

generated by `probo`'s Grounded generator. The execution time is slightly higher in the decisions problems, although not very significantly so.

Figure 4 shows the statistics for the enumeration problems in the grounded semantics (L) and for the decision problems in the preferred semantics (R) using graphs generated by `probo`'s SCC and Stable generators. There are still performance gains in all problems especially in the enumeration ones, but there is now less variation between GRIS and EqArgSolver in the decision problems. This is likely due to the fact that the types of graphs in this class of tests benefit less from gains in performance of the grounding module.

These preliminary results, admittedly over a small sample, are very encouraging and serve to clearly demonstrate the advantages of using the discrete version of the schema over its full-fledged version.

6. Conclusions and Discussion

With GRIS, we showed that we could successfully use numerical methods in the solution of computational problems of traditional argumentation semantics. GRIS used the full version of the Gabbay-Rodrigues Iteration Schema, put forward in [6], for "grounding" tasks (e.g., computation of grounded extensions and the propagation of values of a solution through a network). The main disadvantage of the schema for this purpose was the need for approximation of the values in the limit of a sequence.

In [7], Gabbay and Rodrigues proposed a simplified version of the schema, called the *Discrete Gabbay-Rodrigues Iteration Schema*. Giving initial value $1/2$

to all nodes in a network, the simplified version will converge to exactly the same values as its full-fledged counterpart without the need for approximation. The iterations in the simplified schema also involve much simpler computations.

We used the discrete version of the schema in the implementation of EqArg-Solver and made a number of data representation improvements. For instance, given that the discrete version of the schema only uses the values $\{0, 1/2, 1\}$, we multiplied all equations by 2 in the implementation, shifting the values to the set $\{0, 1, 2\}$. This allowed us to use the smallest available native data type in C++, i.e., `char`, which is only 8 bits long (as opposed to GRIS' `double` which is 8 *bytes* long). This allowed for an immediate 8-fold reduction in the memory requirements. In general, these improvements resulted in a vast gain in performance as described in Section 5.

There is, of course, much more work to be done. We need to incorporate intermediate checks in the generation of preferred extensions to deal with problems in the complete and stable semantics.

We have been running benchmarks over a much larger sample of graphs and have identified areas where performance can be improved. In particular, the search for a solution to some decision problems can be made more efficient, by carefully employing strategies specifically tailored to the problem at hand.

Finally, the computation of solutions to the problems in the preferred semantics can be further improved by designing a more efficient algorithm for maximisation of accepted nodes. This is currently work in progress.

References

[1] P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168(1):162 – 210, 2005.

[2] M. Caminada and G. Pigozzi. On judgment aggregation in abstract argumentation. *Autonomous Agents and Multi-Agent Systems*, 22(1):64–102, 2011.

[3] F. Cerutti, N. Oren, H. Strass, M. Thimm, and M. Vallati. The first international competition on computational models of argumentation (ICCMA15). `http://argumentationcompetition.org/2015/index.html`, 2015.

[4] F. Cerutti, N. Oren, H. Strass, M. Thimm, and M. Vallati. Summary report of the first international competition on computational models of argumentation. *AI Magazine*, 37(1):102, 2016.

[5] D. M. Gabbay. Equational approach to argumentation networks. *Argument and Computation*, 3:87–142, 2012. DOI: 10.1080/19462166.2012.704398.

[6] D. M. Gabbay and O. Rodrigues. Equilibrium states in numerical argumentation networks. *Logica Universalis*, pages 1–63, 2015.

[7] D. M. Gabbay and O. Rodrigues. Further applications of the Gabbay-Rodrigues iteration schema in argumentation and revision theories. In C. Beierle, G. Brewka, and M. Thimm, editors, *Computational Models of Rationality*, volume 29, pages 392–407. College Publications, 2016.

[8] B. Liao. *Efficient Computation of Argumentation Semantics*. Elsevier, 2014.

[9] S. Modgil and M. Caminada. Proof theories and algorithms for abstract argumentation frameworks. In Guillermo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 105–129. Springer US, 2009.

[10] O. Rodrigues. GRIS system description. In M. Thimm and S. Villata, editors, *System Descriptions of the 1st International Competition on Computational Models of Argumentation*, pages 37–40. Cornell University Library, 2015. `http://arxiv.org/abs/1510.05373`.

[11] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.