argmat-dvisat: A division-based algorithm framework for solving argumentation problems using SAT[⋆]

Fuan Pu, Hang Ya, and Guiming Luo

School of Software, Tsinghua University, Beijing, China
`Pu.Fuan@gmail.com`, `yah16@mails.tsinghua.edu.cn`,
`gluo@tsinghua.edu.cn`

Abstract. `argmat-dvisat` provides a division-based algorithm framework to compute the reasoning problems in abstract argumentation based on SAT solvers. It firstly divides a (big) argumentation framework into an ordered sequence of (small) sub-frameworks according to the *directionality* of argumentation. Then, each sub-framework is computed based on its previous sub-frameworks in the sequence using SAT solvers. The solutions of the whole argumentation framework are yielded by integrating the solutions of all sub-frameworks. `argmat-dvisat` currently supports all reasoning tasks of `ST`, `CO`, `PR`, `GR` and `ID` semantics, as well as the special task "Dung's Triathlon".

Description

It has been shown that most computational problems in abstract argumentation frameworks are at least as hard as the NP-complete problems (see [1, 2]). The time required to solve these problems using any currently known algorithm may grow exponentially as the size of the problems increases. When faced with a big argumentation framework, the only approach, we can possibly expect to solve it, is to divide it into a sequence of solvable sub-frameworks. Our `argmat-dvisat` is such a solver. It provides a division-based algorithm framework for solving the reasoning tasks in abstract argumentation:

- First, the entire argumentation framework is decomposed into a sequence of small sub-frameworks. The data structure of these sub-frameworks is shown in Figure 1, in which *compID* represents the ID of a sub-framework, *argIdxs* is a vector and records the indexes of the arguments in this sub-framework, and *subsols* is used to store the solutions of this sub-framework. Here, we do not record the attack relations of the sub-frameworks, since we can access these relations from the entire argumentation framework according to *argIdxs*. The division is based on its strongly connected components (SCCs). Each component can be seen as a sub-framework. According to the concept of *directionality* in argumentation [3], some dependencies may exist within these sub-frameworks. Therefore, we topologically organize these sub-frameworks into an ordered sequence, in which each sub-framework depends on all its previous sub-frameworks in the sequence. In other words, solving the current sub-framework must require that its previous sub-frameworks have been solved, since the status of some arguments in this sub-framework depend on the arguments in these previous sub-frameworks.

⋆ This work was supported by the Fund NSFC61572279.

```
typedef struct {
  int               compID;  /**< The ID of the sub-framework */
  vector<int>       argIdxs; /**< The list of argument ID in the subAF */
  queue<bitvector> subsols; /**< Used to store the solutions of the subAF */
} SubAF_t;
```

Fig. 1. The data structure of sub-frameworks

- Second, we propose an algorithm framework to solve these sub-frameworks according to various reasoning tasks and various semantics. Taking the task to enumerate all stable extensions for example, its division-based algorithm is shown in Algorithm 1[1], in which each sub-framework is firstly encoded into CNF formulas (for the encoding approaches, see the description of our other solver `argmat-sat` also submitted to ICCMA-2017), and then a SAT solver is exploited to find the solutions of the sub-framework. Of course, these sub-frameworks can also be solved using other solvers. Note that each solution of a sub-framework indicates a different branch. Hence, the encoding and solving processes is dynamic, since they rely on the solutions of the previous sub-frameworks. We ensure this requirement by maintaining a global Boolean vector extension $\mathbf{x}$. On the one hand, this approach can record the information of the current branch. On the other hand, it allows the two processes to be able to access the dependent variables directly from $\mathbf{x}$. The algorithm is initialized by solving the sub-framework at level zero (line 4), and then uses a big loop (line 6-25) to search solutions of all the remaining sub-frameworks starting from level one. When the search backs to the level zero (line 7), the entire traversal is finished, and the big loop terminates. If the search goes to the final level (line 9), it means that all sub-frameworks are solved, and then all extensions at this branch are added into the result set (line 11 and 12). When all extensions at this branch are enumerated, the search backs to the last level and prepares to start a new branch (line 14). Now let us see the search goes into the intermediate levels (line 17-23). If the search of the last level is not completed, then we start a new branch by moving and integrating one of the solutions at last level into $\mathbf{x}$ (line 18), and solve the sub-framework in current level according to the newly changed $\mathbf{x}$ (line 19). After that, the search goes to next level in order to continue expanding all branches of the current sub-framework (line 20). If the search of the last level is completed, the search continues to backup (line 22).

In current version of `argmat-dvisat`, this algorithm framework has also been applied for solving `CO`, `PR`, `GR` and `ID` semantics. For these semantics, another global Boolean vector, corresponding to the auxiliary variable vector in the CNF encoding of the `CO` semantics, is introduced to maintain the branch information. The way for searching the maximal semantics is referred from the assumption-based approach, which is described in `argmat-sat`. Our `argmat-dvisat` also supports the special track “Dung’s Triathlon”.

[1] In some literatures, the stable semantics is shown to be incompatible with the concept of directionality. In `argmat-dvisat`, however, we show that the stable semantics can also be computed based on this concept.

Algorithm 1 **The algorithm framework to enumerate all** `ST` **extensions**

```
Require: $\Delta = \langle \mathcal{X}, \mathcal{R} \rangle$ — input an AF with $|\mathcal{X}| = n$;
Ensure: $\mathcal{E}_{\texttt{ST}}(\Delta)$ — return all ST extensions of $\Delta$;
 1: Divide $\Delta$ into a sequence of sub-frameworks, denoted by subAFs$[0, 1, \cdots, m-1]$;
 2: $\mathcal{E}_{\texttt{ST}}(\Delta) \leftarrow \emptyset$;
 3: Initialize the Boolean vector extension $\mathbf{x} \leftarrow \mathbf{0}^n$;
 4: Solve subAFs$[0]$ using SAT solvers, and store the solutions into subAFs$[0].subsols$;
 5: $i \leftarrow 1$;                                          ▷ Specify the start point
 6: while true do
 7:     if i==0 then
 8:         Traversal ends, when $i$ is changed to zero;
 9:     else if $i == m$ then                 ▷ $i == m$ means all sub-frameworks are solved
10:         if subAFs$[i-1].subsols$ is not empty then
11:             Move and integrate a sub-solution of subAFs$[i-1].subsols$ into $\mathbf{x}$;
12:             Now, $\mathbf{x}$ is a stable extension, and thus is added into $\mathcal{E}_{\texttt{ST}}(\Delta)$;
13:         else
14:             $i \leftarrow i - 1$, i.e., back to the last sub-framework;
15:         end if
16:     else
17:         if subAFs$[i-1].subsols$ is not empty then
18:             Move and integrate a sub-solution of subAFs$[i-1].subsols$ into $\mathbf{x}$;
19:             Solve subAFs$[i]$ according to $\mathbf{x}$, and store its solutions into subAFs$[i].subsols$;
20:             $i \leftarrow i + 1$, i.e., move to the next sub-framework;
21:         else
22:             $i \leftarrow i - 1$, i.e., back to the last sub-framework;
23:         end if
24:     end if
25: end while
26: return $\mathcal{E}_{\texttt{ST}}(\Delta)$;
```

`argmat-dvisat` is implemented by C++, and meets the standard command interface of the requirements of ICCMA-2017. Its SAT engine is selected as CryptoMiniSat5[2]. `argmat-dvisat` is multi-threading and cross-platform (Windows and Unix OS). The source codes can be found on the website of our project `argumatrix`[3].

References

1. Dunne, P.E., Wooldridge, M.: Complexity of abstract argumentation. In: Argumentation in artificial intelligence. Springer (2009) 85–104
2. Baroni, P., Caminada, M., Giacomin, M.: An introduction to argumentation semantics. The Knowledge Engineering Review **26** (12 2011) 365–410
3. Baroni, P., Giacomin, M.: On principle-based evaluation of extension-based argumentation semantics. Artificial Intelligence **171**(10-15) (2007) 675–700

[2] `https://github.com/msoos/cryptominisat`

[3] `https://sites.google.com/site/argumatrix/`