UNIVERSITY “ALEXANDRU IOAN CUZA” IAȘI
FACULTY OF COMPUTER SCIENCE

DISSERTATION PAPER

Argumentation Framework with constraints

Proposed by

Drăgoi Ștefan

Session: July 2023

Scientific coordinator

Lect. Dr. Frăsinaru Cristian Andrei

Table of Contents

Introduction

In the last ten years, Argumentation has become an important approach in handling Artificial Intelligence problems. The main subject for research from this field is based on the *abstract argumentation* theory defined by Dung (1995). The central concept of this paper is that an argumentation framework is viewed as a directed graph in which arguments are represented as graph nodes and the attacks between arguments are represented as graph arcs. Given a graph described as above, the main tasks are to determine properties of framework's arguments (e.g. an argument is acceptable) and several properties of the framework's subsets (e.g. conflict – free, admissible, stable). Following Dung (1995) definition of abstract Argumentation framework there have been defined a couple of derived notions and extensions. In this paper, intially, we will focus on the theoretically aspects of this topic, providing a relevant and complex of an directed graph representing an Argumentation framework. Afterwards, we will define the most important properties of such a system and then, based on the visual representation, we will explain how and why the graph meets the properties defined. This will be done in order to have a clear and precise understanding of the subject, to be able later to focus on the programatically aspects of the problem. Having the topic defined and well understood, we will focus in the following chapters to address the subject from a more practical point of view. The core of the work is represented by an web application which has as the main backend component an algorithm based on constraints implemented in Java using the library „Choco – solver". The mentioned algorithm is designed to be able to receive as input a set of arguments (nodes) and a set of pairs representing attacks between system arguments and based on the defined constraints, it will be able to provide the properties that the user defined argumentation framework satisfies. The web application allows the user to draw a directed graph and its afferent arcs representing attacks between arguments and obtain the properties from the algorithm implemented in the backend.

1 Example of Argumentation

In this century argumentation has become a more relevant subject, most of the companies or schools organizing sessions where its participants are to express their opinion on a certain subject, in this way arguing why they think their perspective is the right one. In this way we can understand that argumentation is a major component of human intelligence. In order to illustrate better the principle of argumentation, we will present an example, a fictional situation where there is an argument between two people A and B, whose countries are at war, about "who is responsible for blocking negotiation in their region" (Dung, 1995). The example has been extracted from Dung article in 1995 referenced at (Dung, 1995).

"My government cannot negotiate with your government because your government doesn't even recognize my government.

A: Your government doesn't recognize my government either. The explicit content of B's utterance is that the failure of B's government to recognize B's government blocks the negotiation. This establishes the responsibility of B's government for blocking the negotiation by an implicit appeal to the following commonsense interpretation rule:

Responsibility attribution: If an actor performs an action which causes some state of affairs, then the actor is responsible for that state of affairs unless its action was justified.

A uses the same kind of reasoning to counterargue that **B**'s government is also responsible for blocking the negotiation as **B**'s government doesn't recognize **A**'s government either.

At this point, neither arguer can claim "victory" without hurting his own position. Consider the following continuation of the above arguments:

B: But your government is a terrorist government.

This utterance justifies the failure of **B**'s government to recognize **A**'s government. Thus, the responsibility attribution rule cannot be applied to make **B**'s government responsible for blocking the negotiation. So, this represents an attack on **A**'s argument. If the exchange stops here, then **B** clearly has the "last word", which means that he has successfully argued that **A**'s government is responsible for blocking the negotiation."

2 Related work

In recent years, as Dung proposal for Abstract argumentation framework (Dung, 1995), *boolean satisfiability* or SAT based applications for reasoning with abstract argumentation framework have blown blown up and have been published and proposed at different competitions, especially at International Competition on Computational Models of Argumentation (ICCMA). There are a couple of solutions implemented and presented, as said, at different competitions, but in this paper we will mention only the most relevant ones, the ones which received some prizes or had an exceptional performance.

2.1 CoQuiAAS

CoQuiAAS has been awarded as the best SAT argumentation solver at ICCMA in 2015. The application is named CoQuiAAS and because it is written in C++, one of the main advantage of it is that is a quick solver, the title suggests (Constrained-based Quick Abstract Argumentation Solver). The authors stated that the efficiency of the application is due to object oriented programming paradigm. They have used this paradigm also to give the application an "elegant conception" which is really important in software development. As known in the area of programming, C++ ensures having high computing performances which is not the case for other OOP languages. In addition, it was easier for the development team to integrate the used C++ tool named *coMSSExtractor*. This is a library which extracts MSS/coMSS pairs from a Partial Max-SAT instance. It integrates the Minisat SAT solver which made the implementation easier for CoQuiAAS as it did not need another solver to compute the set of requests. The core of the application is the interface *Solver* which defines a set of high-level methods. The initialization is made via *SolverFactory* class which returns an instance of interface. The abstract class SATBasetSolver is responsible for gathering features and initialization common to each solver that implements a SAT solution. Among these class, there is a DefaultSATBasedSolver and its subclasses, that have the role of calling coMSSExtractor APIs to solve the problems. The architects have designed the tool such that it can be used with other SAT solvers rather than the one used here. The steps of extending the tool using another library are simple and detailed explained in (Jean-Marie Lagniez, 2015): create a new class *MySolver* which extends SATBasedsolver and implement the required abstract methods (*initProblem*, *hasAModel*, *getModel* and *addBlockingClause*). As the authors specified, there is also an option to extend directly the interface Solver and implement its methods.

The benchmarks for this tool is also an important advantage and one of the main reasons why it received the first place at the competition in 2015. There were many tests with many instances. Each instance had between 5000 and 100 000 arguments an the approximate mean for determine the argumentation framework properties is 7.5 seconds.

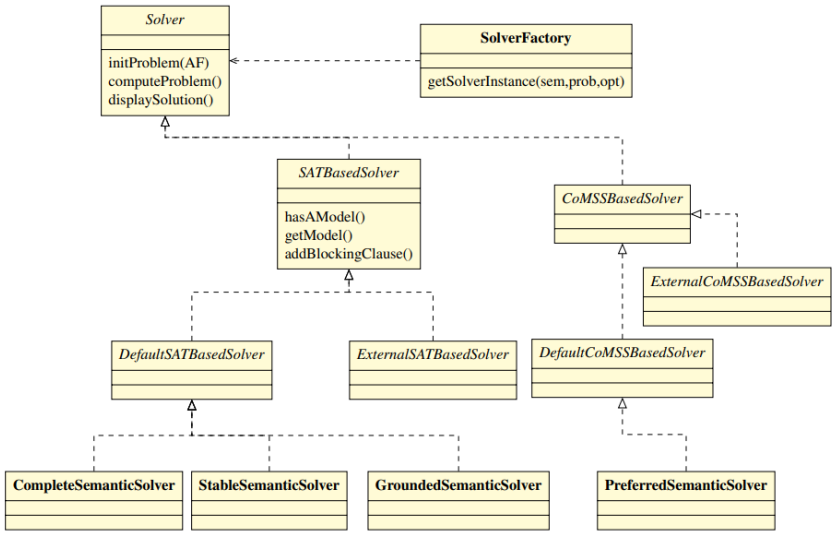

Figure 1 - Simplified class diagram of the solver part of CoQuiAAS (Jean-Marie Lagniez, 2015)

2.2 ArgSemSAT

The application is named "ArgSemSAT" and in (Federico Cerutti, 2019) they present the design chosen that led to the development of it. An important mention is that this tool is the winner of the preferred semantics track at the 2017 International Competition on Computational Models of Arguments (ICCMA 2017), "a biennial contest on problems associated to the Dung's model of abstract argumentation frameworks, widely recognized as a fundamental reference in computational argumentation". ArgSemSAT is set of search algorithms which are designed to solve "enumeration and skeptical/credulous acceptance problems for grounded, complete, preferred and stable semantics", as well as decision problems associated with these semantics. The application encodes the constraints in order to complete labellings of an argumentation framework as a SAT problem. As the tool presented in the previous section, this algorithm is implemented in C++, but it uses the Glucose SAT solver.

The performance of this set of algorithms is achieved due to the fact that there are applied a series of SAT-reductions. More specifically, developers did not apply decomposition techniques on the argumentation framework as there were a bad experience in previous approaches that led to a reduced performance. In addition to that, it has been "exploited classical SAT solvers with hard clauses, leaving apart the use of soft clauses which may play a role in the identification of preferred extensions" (Federico Cerutti, 2019). An important mention is that not the usage of a more advanced SAT-based techniques was the main reason for the performance and efficiency improvement, but the mentioned features of reducing to SAT.

Related to the implemented algorithms, these perform a search "in the space of the complete labellings" (Federico Cerutti, 2019) and for each step in the search process there is a call to another non-deterministic function which returns a complete labelling satisfying a number of specific constraints. The next step is to encode the accepted solutions as a propositional formula in order to obtain a complete labelling with its corresponding satisfying assignment. This approach made the application extensible and provides support for developers to extend it using a mean of SAT solvers.

2.3 µ–toksia

µ–toksia is a system which covers a couple reasoning tasks over standard and dynamic abstract argumentation frameworks, solving all tracks and reasoning tasks considered in the 2019 edition of International Competition on Computational Models of Argumentation (ICCMA 2019). It has been ranked the first in all reasoning tasks in the main track of ICCMA 2019 and "has been shown to scale noticeably better on the dynamic track tasks than its current competitors" (Andreas Niskanen, 2020). The application is built to make use of SAT solvers in an incremental way, this solver being instantiated once during an execution, keeping the state of the SAT solver through its API by doing iterative computations.

The µ–toksia system is implemented in C++, making use of STL data structures and interfaces to the Glucose and CryptoMiniSAT SAT solvers. The application has been implemented without using specialized algorithm that would have made difficult for users to extend the SAT solver APIs. The system in composed of four main components as follows:

- *AF.h*: class responsible with mapping arguments to integers and back, storing the attacks;
- *SATSolver.h*: generic interface to a SAT solver which allows the users to integrate any SAT salver respecting the contract of the interface;
- *Encodings.h*: the implementation of SAT encodings and also functions that calculate the properties of an argumentation framework (conflict-free, stable, admissible, etc.);
- *CredAcceptance.h, SkeptAcceptance.h, SingleExtension.h, CountExtensions.h*: Implementations of the SAT-based algorithms for the corresponding reasoning tasks.

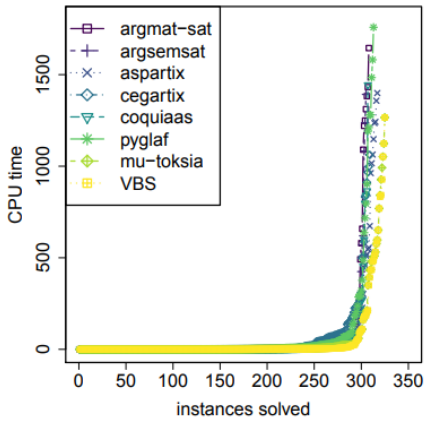

Figure 2 - Performance comparation between existing tools (Andreas Niskanen, 2020)

2.4 Pyglaf

Besides the tools and system presented in this section, Pyglaf has a different approach in solving computational problems if abstract argumentation framework as described in (Dung, 1995) by using circumscription. Circumscription is a "nonmonotonic logic formalizing common sense reasoning by means of a second order semantics, which essentially enforces to minimize the extension of some predicates" (Alviano, 2021).

The application is implemented in Python, as the name suggests, and uses a library called “Circumscriptino”, a circumscription solver that extends Glucose, this library being also created by the author of the mentioned reasoner, Mario Alviano. The communication between the application and used library is minimal, a single call to the solver per execution and is made via a stream processing. In order to obtain the properties (which are described in detail in section 3) the application make linear reductions.

To calculate the set of admissible extensions A, Pyglaf makes request to Circumscriptino to expand A as much as possible by computing an admissible extension that maximize the accepted arguments which are not present in the set A (Alviano, 2021).

3 Abstract Argumentation framework

In 1995, Phan Minh Dung enunciated the theory of *Abstract Argumentation framework* as "a pair of a set of arguments, and a binary relation representing the attack relationship between arguments. An argument is an abstract entity whose role is solely determined by its relations to other arguments" (Dung, 1995). The framework has been defined as abstract because in the article there has not been paid attention to the internal structure of the arguments, but on the relationships and the properties determined by these.

An argumentation framework is defined as a pair:

AF = <S, attacks>,

where S is a set of arguments and *attacks* is a binary relation on S (attacks ⊆ S x S).

In this paper, we will represent an attack between two arguments A and B as (A, B), which means that A attacks B.

In the following subchapters we are going to define and describe each property an argumentation framework can satisfy. In order to have a clear and correct perspective over the concept of these, I have decided to create a complex and relevant example with multiple arguments and several relationships between them (attacks).

The example of this framework we propose consists of a set S of thirteen arguments:

- S = {A, B, C, D, E, F, G, H, I, J, K, L, M}

and a set R of eighteen attacks between arguments (in Figure 3 attacks are represented as arcs between nodes)

- R = {(A, B), (B, D), (D, C), (C, A), (D, E), (E, F), (G, F), (J, G), (D, L), (L, J), (H, G), (I, G), (H, I), (I, H), (K, H), (K, I), (M, K), (K, M)}.

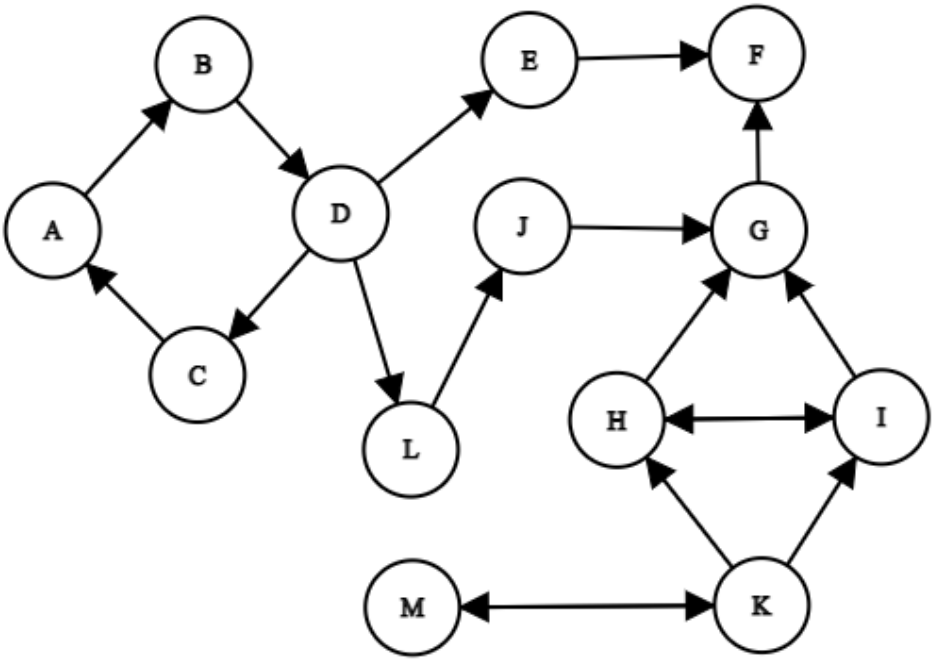

Figure 3 – Example 1 of argumentation framework graph

3.1 Attacks

As defined above, in abstract argumentation framework is a binary relation on S (the set of arguments) which in the example provided in Figure 3 is represented as an arc between two nodes. In our example, there exists an attack from A to B written in this paper as **(A, B)**. The attacks can be mutual, and in the example, there is an attack from node M to K and another attack from K to M. It is worth to be mentioned that, even if it is not illustrated in the graph, in an argumentation framework an argument can attack itself.

3.2 Attackers and defends

In an abstract argumentation framework, two properties of an argument are defined: to be an attacker and to defend.

- A node A ∈ S is said to be an **attacker** if ∃ B ∈ S such that (A, B).

In Figure 3 all arguments are attackers. For our example, argument B is an attacker because it attacks D: (B, D) and also argument D is an attacker because (D, C), (D, E) and (D, L). The notion of *attack* can be applied for an argument and a set of arguments. An argument A attacks a set S' if and only if ∃ B ∈ S' such that (A, B). Similarly, a set S' attacks an argument A if and only ∃ B ∈ S' such that (B, A).

- A node A ∈ S is said to be that **defends** a node C ∈ S, if ∀ B ∈ S such that (B, C), then (A, B) (i.e. A defends C if A attacks all arguments that attack C).

In the provided example, argument A defends D because A attacks all the arguments that attack D (i.e. B). In this case, the "defending" is mutual because D defends A due to the attack (D, C), C being an attacker for A. It is important to mention that we can say that a set of arguments *defend* an argument or another set of arguments. In the example, {L, K} defends argument G because: (K, H), (K, I) and (L, J), H, I and J being attackers for G. At the same time, argument B defends the set of arguments {C, E, L} because (D, C), (D, E), (D, L) and B attacks D (B,D), so it is defends these three arguments.

3.3 Conflict – free set

- A set S' of arguments is said to be **conflict – free** if for ∀ A, B ∈ S', then there are no attacks between A, B.

A relevant mention in this subchapter is that S' can contain only one node. As mentioned before, an argument can attack itself and then the set made up of that argument is not conflict – free. In the example presented, there are couple of this kind of sets: {A, D, J, F, H, M}, {B, C, E, L, G, K}, {K, J, F, D}.

3.4 Acceptable arguments

In abstract argumentation framework:

- An argument A is defined as **acceptable** with respect to a set of arguments S', if ∀ B such that (B, A), then B is attacked by S'.

This means that the argument is acceptable if there exists a set of arguments which defends it from all its attackers. For our example from Figure 3 – Example 1 of argumentation framework graphFigure 3, all arguments are acceptable. In that case it is simple to obtain this property since there is no argument in the framework which is not attacked by another argument. Let us give

some concrete examples: F is an acceptable argument because it is attacked by E and G. Both two are attacked by another two arguments as follows: (D, E) and (J, G). In the same way, the argument M is acceptable because it is attacked by K and M is defended by itself by attacking node K (we called it a mutual attack)

3.5 Admissible set

The property of admissibility is one of the most relevant when talking about abstract argumentation framework. This one defines a strong relationship between members of a set as follows:

- A set S' of arguments is said to be **admissible** if and only if:
  - S' is conflict – free;
  - For ∀ A ∈ S', if B attacks A (B, A), then S' attacks B;

The definition can be stated more informally: a set is admissible if there are no attacks between arguments of that set (conflict – free) and for any argument from the set is attacked by an outsider, then another argument from the set attacks the attacker. Another explanation for the second condition of the property is that the arguments of the admissible set defend themselves. As mentioned for previous properties, also admissibility can be applied to a set with only one argument. To have a clear and better understanding, we will explain the property using the example provided in Figure 3. The shortest and simple admissible sets are {M} and {K} because M does not attack itself and is attacked by K being able be defended by attacking back node K, thus being defended. Due to the fact that M is defended by an argument of set {M}, this makes the set admissible. The same logic applies to set {K}. Another small admissible set is {A, D}. There are no attacks between them, so satisfy the first rule. The second rule is also satisfied with the following explanation: A is attacked by argument C, which is attacked in turn by an argument of the set {A, D} (i.e. D with (D, C)). D is attacked by B, which is attacked by an argument of the set (i.e. A with (A, B)). Other admissible sets are: {B, C, E, L}, {M, H, F, D}, {D, A, J, F}, {A, D, J, M, H}, {A, D, J, F, M, H}, {B, C, E, L, G, K}.

3.6 Preferred extension

As Dung (1995):

- A **preferred** extension is "a maximal (with respect to set inclusion) admissible set of AF", where AF is Argumentation Framework.

Defining above what an admissible set means, we can select the maximal sets in order to obtain the preferred ones: {A, D, J, F, M, H} and {B, C, E, L, G, K}. Having these two definitions, we can extract an important theorem of an argumentation framework related to admissibility and preference: for ∀ admissible sets, there ∃ at least one preferred set.

3.7 Characteristic function

In abstract argumentation framework is defined a function named the **characteristic function** which has the following definition:

- $F : 2^S \rightarrow 2^S$
- F(S') = {A | A is acceptable with respect to S'}

The definition on characteristic function can be expressed with other words as follows: F(S') = the set of arguments which are defended by the arguments of S'. Referring the graph from the Figure 3 we can calculate the characteristic function of some arguments: F({A}) = {D} because A defends only one argument in the graph which is D; F({D}) = {A, F, J}, by attacking K, E and L, D is defending A, F and J. Function F can be calculated for a set with more than one argument: F({L, K}) = {G, K, I, H}, F({A, L, I}) = {D, G, I, F}, F({B, C, D, H}) = {A, B, C, D, E, L, J, F, H, G}.

3.8 Complete extension

As the property of a set to be admissible, to be a complete extension is also one of the most important one in an abstract argumentation framework. It defines also a strong relationship between members of a set:

- A **complete** extension is an admissible set of arguments if and only if for ∀ A ∈ S', A acceptable with respect to S', A belongs to S'.

With other words, a set S' is complete if all the arguments that set S' defends are included in S'. The fact that we have defined the characteristic function before, allows us to give another definition of complete extension: a set S' is complete if F(S') = S'.

In the argumentation framework provided as example in the chapter 1, there is a complete extension: S' = {A, D, F, J}. We can proof this by calculating the characteristic function for each argument of the set S' and using the definition of completion F(S') = S'. To avoid any confusion related to the letter F whether is an argument or the characteristic function, for this proof, we will note **argument** F as F", so the complete function S' now becomes {A, D, F", J}.

- F(A) = {D};
- F(D) = {A, F", J};
- F(F") = ∅;
- F(J) = {F"}.

Having he characteristic function calculated for each argument of set S', we can calculate now F(A, D, F", J) = F(A) ∪ F(D) ∪ F(F") ∪ F(J) = {A, D, F", J} = S'. So, the relation for a complete extension is satisfied: F(S') = S'.

3.9 Grounded extension

In subchapter 3.6 we defined the property of a set to be preferred as being the maximal admissible set in an argumentation framework. Also defining complete extension and the characteristic function, we can define the notion of grounded extension which is the opposite of a preferred extension:

- A **grounded** extension is the least fixed point of F (characteristic function).

In the example presented in Figure 3 a grounded extension is the empty set {}.

3.10 Stable extension

Another important property of a set in an argumentation framework is to be stable. We will provide de definition of this one followed then by some examples to have a good understanding of it:

- A set of arguments S' is defined as **stable** extension if

- S' is conflict – free
- $\nexists$ A $\in$ S \ S' such that S' does not attack A, where S is the set of all arguments in an argumentation framework and S' is a subset of it.

With other words, a set S' is a stable extension if and only if its arguments attack all the arguments of the framework that do not belong to S'. In our example, some of the stable extensions are: {B, C, E, L, H, M}, {B, C, E, L, G, K}.

3.11 Threat and defender

In an abstract argumentation framework there are defined the notions of an argument which is a threat for a set and an argument which is a defender for a set:

- An argument A is said to be a threat for a set of arguments S' if: for $\forall$ B $\in$ S', then (A, B) and $\nexists$ (B, A).

More formally, an argument is a threat for a set if it attacks all the arguments of the set and none of the set arguments attacks A. In the example, argument D is considered a threat for the set {C, E, L} because (D, C), (D, E), (D, L) and none of these three arguments attack D. In the same manner K is a threat for {H, I}. On the other hand, K is not a threat for M because M attacks K back, so the property of stabilization does not stand.

- An argument A is defined as a defender for a set S' if it attacks the threat of S'.

In the example from the Figure 3, a defender for the set {C, E, L} is B because node B attacks the threat of the set, argument D. Another example of defender is J for the set {F} because there is an attack (J, G) and G is a threat for {F}.

3.12 Coherent argumentation frameworks

In the previous subsection we have defined essential properties of an abstract argumentation framework as Dung has stated them in (Dung, 1995). In this subsection we want to define the notion of coherent argumentation frameworks which is a condition for the coincidence between subsets which are stable and subsets which are preferred. Generally, if there exists an extension which is preferred, but at the same time is not stable, this indicates that there could be some "anomalies" in the corresponding argumentation framework (Dung, 1995).

Example 3: Given an abstract argumentation framework AF = <S, attacks>, with S = {A} and attacks = {(A, A)}. This example of AF has an empty preferred extension {} which is not stable, so we can conclude that AF is not coherent. In the following, we will try to define a set of properties in order to avoid such "anomalies" which make an argumentation framework not coherent.

3.12.1 Controversial argument

To have a better understanding of this property, we need to define what an *indirect attack* and *indirect defend* means. Let us start with a practical example:

- Let *C* be a proposition of a debate: "is better to work with PHP, than with .NET";
- Argument A_0 supports C: "PHP is easier to learn";
- Argument A_1 attacks A_0: ".NET is faster than PHP";
- Argument A_2 attacks A_1 and so, it supports statement C and argument A_0: "PHP is an open source project, so the community is bigger";

- Argument A_3 attacks A_2: “Yes, but .NET is supported by a high-tech company, Microsoft, so it makes it more stable”.

If we stop at this point, argument A_0 is defeated and the most decisive role in this defeat is played by argument A_3 which indirectly attacks argument A_0. At the same time, if we have had an argument A_4 which attacks A_3, we could have said that argument A_4 indirectly defends argument A_0 by attacking its indirect attacker A_3.

An argument A is said to be ***controversial*** with respect to other argument B if A indirectly attacks B and at the same time it indirectly defends B. It is clear from the definition that an argument is controversial with respect to an argument B.

An argumentation framework F is ***uncontroversial*** if none if its arguments is controversial.

An argumentation framework F is ***limited controversial*** if, for a non-infinite sequence of arguments A_0, A_1, … A_n, … there is no argument A_{i+1} such that is controversial with respect to A_i.

Having defined these properties, Dung has stated that “Every limited controversial or uncontroversial argumentation framework is coherent”. This property is sufficient to avoid that inconsistency presented in Example 3.

4 Practical aspects of Argumentation Framework

In this section we present some practical aspects of Argumentation Framework in several domains such as: games, healthcare and debates. In the first subsection we want to focus on how Dung proposal can be applied in solving the stable marriage problem, how to represent data, how to transform it in an argumentation framework. In the second one we will focus on an application of this subject in which decides whether a treatment is superior to another treatment based on a table with statistical information regarding chances of developing a certain disease based on a specific treatment. In the third subsection we will present an overview of how argumentation framework and its extensions can be used to decide a winner of a debate, how such a system can be integrated with a Social network application.

4.1 Argumentation framework and the Stable Marriage Problem (SMP)

In this subsection we want to provide a perspective based on argumentation framework on how to approach the Stable marriage problem by translating the preferences of each participant in the game seen as arguments in an attack.

4.1.1 Stable Marriage Problem background

The stable marriage problem is finding a stable match between a set of *n* men and n women. Each of these n men has a preference order for the woman he wants to marry, likewise, each woman has a preference order for the man. The final matching must be stable, which means that there are no man and woman who both prefer marriage with each other rather than stay with their current partner. This problem has firstly been introduced by Gale and Shapley in 1962 (D. Gale, 1962). In the same paper, Gale and Shapley proposed an algorithm named "GSS" which solves this problem. The main idea of this algorithm is to iterate over all free man while there is any. Every free man checks every woman in its preference list if whether is free or not. If positive, then they become engaged. If the woman is not free, then she chooses either engaging with him or remain with the current one. An engagement can be broken if a woman is asked for a better option according to her preference list. The time of Gale-Shapley algorithm is O(n^2).

In a more formal way, a solution for this problem is a one-one correspondence S:M->W such that: ∄ (m, w) s.t. m prefers *w* to *S(m)* and *w* prefers m to $S^{-1}(w)$, where *m* represents a man and *w* represents a woman (Dung, 1995).

4.1.2 Argumentation framework approach

The stable marriage problem can be expressed as finding a stable extension of an argumentation framework AF = (A, *attacks*) as follows: given a marriage (A, B), where A prefers D to B (D being another candidate of opposite sex of A), then D represents a threat to the presented marriage (see threats from section 3.11). So, informally, a potential scenario in which (A, D) is a marriage, then this represents an attack to marriage (A, B). At the same time, this attack can be removed if D is already married to someone whom D prefers to A, based on the definition of the problem from section 4.1.1.

Let us define the SMP as an abstract argumentation framework:

- M is the set of *n* men;
- W is the set of *n* women;
- AF = (A, *attacks);*

- A = M x W;
- attacks ⊆ A x A:
    - (C, D) attacks (A, B) iff:
        1. A = C and A prefers D to B,
           or
        2. D = B and B prefers C to A.

A set of marriages S ⊆ A is considered a solution for the problem iff S is a **stable extension** of the corresponding abstract argumentation framework.

To have a clear view of this approach in the stable marriage problem, let us consider the following example of a potential problem where we included an extension in which there are three participants with undefined orientation which can be married with any of other participants with this kind of orientation. Let P = {m, w, p1, p2, p3}, where m is a man, w is a woman and p1, p2, p3 have undefined orientation. In the preference list, both *m* and *w* have each other as first option. In this case, there is a marriage (m, w). On the other hand, there is a love triangle between p1, p2, p3 as follows: p1 loves p2, p2 loves p3 and p3 loves p1. It is clear that in this case we cannot arrange these participants in order to create a marriage. The only stable marriage of this example is between *m* and *w*. It is worth mentioning that in the corresponding argumentation framework we have exactly one stable and preferred extension which is {m, w}. At the same time, there is nothing wrong with the corresponding argumentation framework resulted for this problem. If there is something wrong, then it is the problem itself, the world we are trying to model is wrong.

4.2 Argumentation framework in healthcare

Evidence-based decision making is becoming increasingly important in healthcare. In (Anthony Hunter, 2012), authors has presented a framework based on Dung argumentation framework for representing and synthesizing knowledge from clinical trials being considered a couple of outcome indicators. Briefly, the system generates and evaluates arguments, which will be seen in the next subsections are represented by a treatment or a combination of treatments, whether a treatment is superior, equivalent or inferior to other based on the available evidence. An important role in "filtering" the evidences are meta-arguments which indicates if a specific argument is based on a weaker evidence.

4.2.1 Problems

In general, the formulation of guidelines involves collecting a massive amounts of evidence. In medicine, for example, "guidelines are based on a rapidly growing body of biomedical evidence, such as clinical trials and other scientific studies" (Anthony Hunter, 2012). As the evidence in this domain needs to be systematically reviewed and aggregated, there is an important human expenditure and effort to produce evidence-based guidelines. At the same time, these guidelines can become out-of-date quickly, not surprisingly knowing that PubMed, the online repository of biomedical abstracts run by the US National Institute of Health, is growing at the rate of 2 articles per minute. Considering these aspects, there is a need of a framework which is able to aggregate and make evidence-based recommendations based on large repositories.

4.2.2 Approach

The developed framework allows for arguments and counterarguments to be constructed and compared based on diverse criteria. At the same time, the framework provides a solution for reasoning with the evidence according to the patient class to which it applies.

The input of this framework is a table of evidence comparing pairs of treatments. Each row in the table has the combination of treatments, comparation type, the outcome, the statistical significance etc. For treatments presented in the input table, the developed system based on argumentation would determine for any treatments t_1 and t_2 whether t_1 is superior to t_2, or t_1 is equivalent to t_2 or t_1 is inferior to t_2. These assessments are determined by the resulting arguments and counterarguments.

The output of the application framework is a graph named "*superiority graph*" which is similar with the Dung style of graph to represent abstract argumentation framework. It is a directed graph where each node is representing a treatment (those exposed in the input table) where arcs have the following significance:

- An arc from t_1 to t_2 means that treatment t_1 is superior to t_2;
- A bidirectional arc between t_1 and t_2 means that t_1 is equivalent t_2;

4.2.3 Analogy between treatment and argumentation framework

The following example has been retrieved from (Anthony Hunter, 2012): "*Consider arguments A1 = 'Patient has hypertension so prescribe diuretics', A2 = 'Patient has hypertension so prescribe beta-blockers', and A3 = 'Patient has emphysema which is a contraindication for beta-blockers'. Here, we assume that A1 and A2 attack each other because we should only give one treatment and so giving one precludes the other, and we assume that A3 attacks A2 because it provides a counterargument to A2.*". The resulted graph for this example is the following:

Figure 4 - Analogy between medicine and AF

Mentioning in section 3 all properties that we are interested in about an abstract argumentation framework, now we are able to calculate them having the following relevant result: extension $\{A_1, A_3\}$ is the only **complete**, **grounded** and **preferred** extension. In this case, the conclusion is clearly that the patient should take the A1-A3 combination of treatments in order to treat the disease.

4.2.4 Algorithm

To have a better understanding of how this framework works, we will start with an example presented in (Anthony Hunter, 2012):

	Left	Right	Outcome indicator	Value	Net	Sig	Type
e_{81}	CP	NC	breast cancer	1.04	<	yes	RCT
e_{82}	CP	NC	ovarian cancer	0.99	>	yes	MA
e_{83}	CP	NC	pregnancy	0.05	>	yes	RCT
e_{84}	CP	NC	thrombosis	1.02	<	yes	MA

Figure 5 - Comparing use of contraceptive pill (CP) and no contraception (NC)

In this input table we have the following notations and explanations:

- Left is the left arm of people, those who took the contraceptive pill;
- Right is the right arm of people, those who did not take the contraceptive pill;
- Outcome indicator is what is being measured;
- Value is the value of that measure, which will be explained below how it is calculated and what it means;
- Net could have 3 values: > (SUPERIOR), < (*INFERIOR*) and ~ (*EQUIVALENT*). These signs are closely related to the field *Value*. If value is greater than 1, then the net outcome is *INFERIOR*. If the value is less than 1, then the net outcome is *SUPERIOR*. Else is *EQUIVALENT*.
- Sig means whether the value is statistically significant or not.
- Type is the evidence type which in this case is whether RCT (Randomized clinical trial) or MA (Meta-analysis).

One of the most important input in the table is field *Value* which represents the risk ratio for the outcome indicator. It is the proportion of between people with indicator in left arm and people with indicator in right arm. Thus, the result could be:

- Value > 1, iff people in the left arm tend to have the indicator more than people in the right arm;
- Value < 1, iff people in the left arm tend to have the indicator less than people in the right arm.

From the table there could be extracted four arguments with their respective relationship of "is superior", "is inferior":

A_1 = <{{e_{82}, e_{83}}, CP > NP}>
A_2 = <{{e_{82},} CP > NP}>
A_3 = <{{e_{83}}, CP > NP}>
A_4 = <{{e_{81}, e_{84}}, CP < NP}>
A_5 = <{{e_{81}}, CP < NP}>
A_6 = <{{e_{84}}, CP < NP}>

The next steps in the presented application are to normalize the values from the input and to transform them in benefits function. After normalizing and expressing the benefits function, the results from example from Figure 5 are:

- Benefits(A_1) = {(ovarian cancer, 0.99),(pregnancy, 0.05)}
- Benefits(A_2) = {(ovarian cancer, 0.99)}
- Benefits(A_3) = {(pregnancy, 0.05)})
- Benefits(A_4) = {(breast cancer, 0.96),(thrombosis, 0.98)}

- Benefits(A_5) = {(breast cancer, 0.96)}
- Benefits(A_6) = {(thrombosis, 0.98)}

Based on other parameters that are taken into account in this framework, it is defined the relationship between benefits on whether one treatment is preferred to other treatment or not. Given the way that an argument graph is represented with the specified arguments (an one directional arc means that treatment t_1 is preferred over treatment t_2 and a bidirectional arc means that treatments t_1 and t_2 are equivalent), then the resulted graph from the exposed benefits and association between arguments and benefits is the following (Anthony Hunter, 2012):

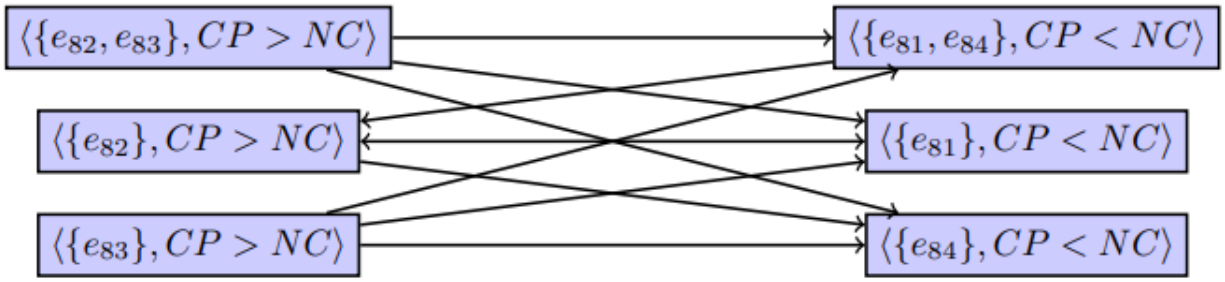

Figure 6 - contraceptive pill (CP) and no contraception (NC) argument graph

The next important step is to generate and apply the meta-arguments of the input data. As stated in (Anthony Hunter, 2012), in general, an argument graph allows any kind of argument to appear in the graph, without considering its importance or relevance from a statistically point of view. Here the meta-arguments come in application. The arguments included in the argument graph need to be somehow "filtered". There are multiple types of meta-arguments, but in this paper, we will include only the most important ones which are:

- Not statistically significant
- Non-randomized and non-blind trials
- Meta-analysis for a narrow patient group

In the presented example, there are two not significantly arguments: $\{e_{82}, e_{83}\}$ and $\{e_{83}\}$, so they are removed from the argument graph. The final argument graph looks as follows:

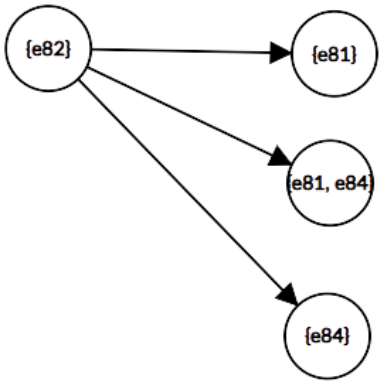

Figure 7 - contraceptive pill (CP) and no contraception (NC) final argument graph

The results are then calculated in order to obtain the **grounded** and **preferred** extensions of the system. If there is a non-empty grounded extension, then the winner is that argument. It

there is an empty grounded extension, then there are multiple preferred extensions and so the winner should be calculated as an aggregation of the extensions (not all).

The winner of the example presented in this paper is e_{82} which means that the most preferred outcome is to have reduced considerable chances to be pregnant than reduce not so significantly chances to have breast cancer or thrombosis.

4.3 Real-time debate analysis using abstract argumentation framework

In the present, there is no applications or tools which supports real-time debates by allowing users to support a point of view and to visualize participants arguments. Therefore, the authors of article (Benjamin Delhomme, 2022) have introduced a new tool called *AIPA* which allows the users who participate at an online debate to: formalize and visualize, in real-time, the arguments of the participants, in order to be able to determine acceptable arguments and conflicting ones. In can be also used to summarize a debate in a way that is easy for the user to follow and understand. However, the main strength of this application is that it represents the arguments as a traceable and transparent chain even for those users who have no expertise in argumentation.

4.3.1 Problems

As highlighted above, in a debate, the discussion could become hard to follow, difficult to find and not structured, this cause could lead to redundant arguments and inability of the debate participants and spectators to conclude the debate, to determine which arguments are supported, who are the supporters and the opponents, which argument is a counterargument for another argument.

4.3.2 Approach

In (Benjamin Delhomme, 2022), authors defines the notion of Argued Discussion Model (ADM) which is a pair <A, T>, where T are the arguments of a debate divided in two categories: *Statements* and *Conclusions*, and T is a binary relationship between arguments. Statements are also divided into two types: *StatementFor* and *StatementAgainst*.

In order to have a clear view of the terms defined above, we will include an example from (Benjamin Delhomme, 2022) representing a public consultation about increasing fuel tax of a group of citizens. There are two arguments: A_1 = "Increasing taxed reduces fuel consumption and therefore pollution" and A_2 = "Increasing taxes penalizes the poorest citizens". Of course, the conclusions in a such debate are clear: C_1 = "Fuel tax should be increased" and C_2 = "Fuel tax should be decreased". An important definition in this approach is that conclusions are mutually exclusive, thus these should be disjointed. In general, a conclusion should be the negation of the other one. The way this tool works is continued in the presented example. Users can provide StatementsFor an argument, for example for argument A_1. Instead, another participant can provide a StatementAgainst A_1, let us say Sa_1 as follows: "This can only be true if credible and accessible alternatives to petrol cars are proposed, which is not the case today".

4.3.3 Algorithm

As mentioned above, the application is designed to use a model called *ADM* which converts the debate discussion in an *ADM graph* which is in turn transformed into an Abstract Argumentation Framework graph, as Dung has written in (Dung, 1995). Each time a new argument is added in the system, a new transformation is triggered in order to obtain the AAF graph.

An important definition within AIPA is the way a debate is concluded and the discussion is solved:

"The rules determining the graph status are defined as follows:

- ***Accepted**: if only one Conclusion is included in the grounded extension;*
- ***Conflictual**: if there is no conclusion in the grounded extension and if more than one Conclusion are included in the preferred extensions;*
- ***Rejected**: if there are no Conclusions in grounded extension and no Conclusions in the preferred extensions.*

Translation is an essential part of the AIPA algorithm. The ADM graph which consists of debate arguments and arcs which can represent a *StatementFor* or a *StatementAgainst* is translated in an Abstract Argumentation Framework graph. In (Benjamin Delhomme, 2022) there are defined rules on how to translate an *StatementFor* arc and a *StatementAgainst* arc in a simple directed graph. In the following figure, we will expose a transformation as it would happen in a real-time debate analyzed using AIPA tool:

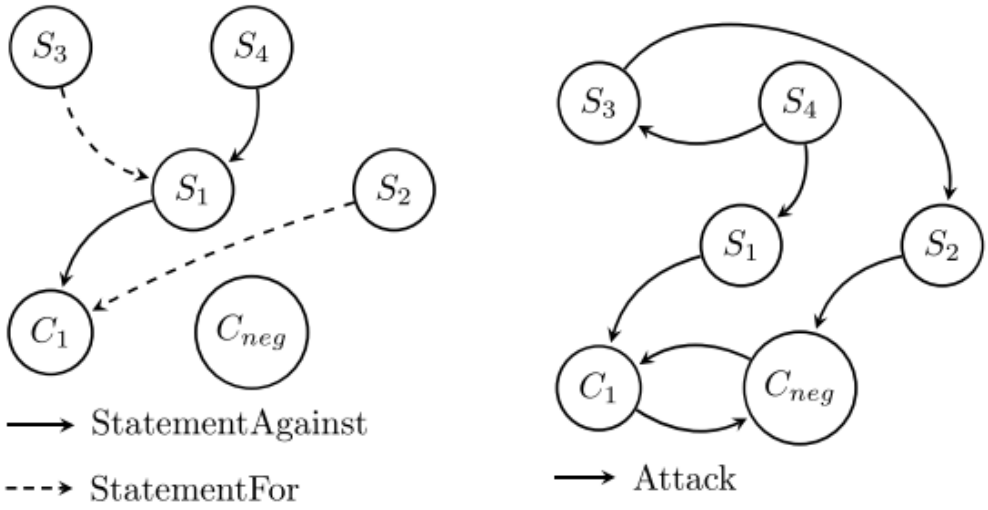

Figure 8 – in the left ADM graph with StatementFor and StatementAgainst arcs, in right the corresponding Dung style graph

So, the connection between Abstract Argumentation Framework and ADM model is made via translating the model argument to an AAF graph, then the "interface" AIPA.

The results in the Figure 8 after calculating de desired extensions are:

- Grounded extensions = $\{S_4, C_1, S_2\}$;
- Preferred extensions = $\{S_4, C_1, S_2\}$.

Respecting the definition stated above, the debate could end as a debate Conclusion is accepted which is C_1.

4.4 Argument web

Argument web has been created following the need of an infrastructure which allows interconnected arguments to be posted in the Web through a comprehensive underlying ontology of argument. It is a URI-addressable structure of linked argument with the scope of allowing the users to follow a "red line" of arguments about a particular topic or posted by a particular person,

across different forums, blogs, editorials, social networks and other media platforms. Mainly, in Web, users can post, comment, share or argue on a couple of topics, this can be done in almost every social platform, for example Reddit, a discussion forum where people are able to comment and discuss about any subject, or Twitter and Facebook, platforms which promote online interaction, but not online critical discussions. That is why there is a need for better online argument and debate platforms which supports critical thinking and structured discussions. Some websites offer databases with structured argument (Debatepedia), providing a numerous high-quality debated about different topics.

Argument Web has the main purpose to provide a Web infrastructure which allows users to store, retrieve and analyze linked argument data. It is based on a common ontology for argument called the Argument Interchange Format (AIF) that combines two technologies: natural linguistic models of argument and abstract mathematical models of argument. Argument Web it is also a feasible solution to investigate mathematical aspects of arguments from diverse online platforms being expressed in natural language. This infrastructure could also provide support for capturing dialogue protocols. These allows users to determine which type of answers can be given to which type of questions. For example, a statement like "*Why is invading Syria militarily feasible?*" can force other parties to give reasons for their claim as "*We should invade Syria because it is militarily feasible*". (Floris Bex, 2013).

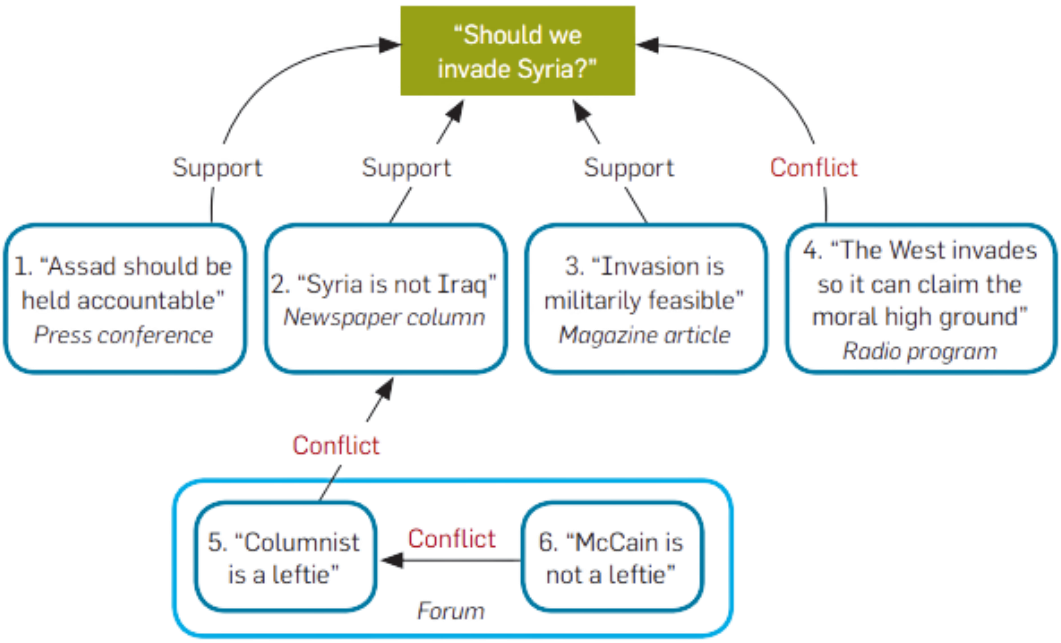

Figure 9 - Example of linked argument data (Floris Bex, 2013)

5 Algorithm

As stated in the previous chapters, the main purpose of this dissertation work is to provide a web application in which the users are able to add arguments as graph nodes in a canvas, add attacks between them (directed edges / arcs) and get the argumentation framework properties. Besides the user interface, there is an algorithm based on constraints implemented in Java. In order to determine the properties I used a java library called Choco-solver which provides an important amount of methods through which the constrains are defined and a solver which retrieve the problem solution based on the defined constrains.

5.1 Constraint Programming

"Constraint programming is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently applied with success to many domains, such as scheduling, planning, vehicle routing, configuration, networks, and bioinformatics" (Francesca Rossi, 2006). A Constraint Satisfaction Problem (CSP) is defined by a set of variables, each one with a domain of values, and a set of constraints defining the feasible combinations of values. Solving a CSP consists of assigning each variable a value in its domain such that all constraints are satisfied. It is then possible to add one or many optimization criteria. The term Constraint Programming (CP) comes from the fact that the problem is described in a declarative way through its constraints. It is a very nice programming paradigm where, instead of giving a procedure about how to solve the problem, the user simply specifies the list of properties that must hold in a solution: "Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it." (Freuder, 1997). This clear separation between the technical aspects and the functional aspects goes beyond programming. It is a different way of thinking that is highly valuable when talking to business representatives for industrial projects.

Once the CP model has been established, it can be solved using a generic CP solver, such as Choco Solver.

5.2 Choco-solver

Choco Solver is an Open Source Java library for Constraint Programming that has been used for more than 20 years by many companies and universities. Choco is among the most efficient solvers available and has a high code quality. Being a Java library, it can be integrated very easily to develop innovative web-services for decision making. Choco-solver comes with:

- various type of variables (*integer, boolean, set, graph* and *real*);
- various state-of-the-art constraints (*alldifferent, count, nvalues,* etc.);
- various search strategies, from basic ones (*first_fail, smallest,* etc.) to most complex (impact-based and activity-based search);
- explanation-based engine, that enables conflict-based back jumping, dynamic backtracking and path repair.

5.3 Argumentation framework algorithm

As stated in the chapter 3, an Argumentation Framework could have a set of properties. In many articles, especially (Dung, 1995), these properties are enounced as *Acceptability semantics* as follows: Let F = (A, R) be an AF and B ⊆ A.

- B is an admissible set iff it is conflict-free and defends its elements.
- B is a preferred extension iff it is a maximal (for set ⊆) admissible set.
- B is a stable extension iff it is a preferred extension that attacks any argument in A\B.
- B is a complete extension iff it is conflict-free and it contains all the arguments it defends.

Example 2. Let us consider an Argumentation Framework: $F_1 = (A_1, R_1)$, where A_1 = {A, B, C, D} and R_1 = {(A, B), (B, A), (A, C), (C, D)}. In order to have a better understanding, we will present the directed graph associated with F_1:

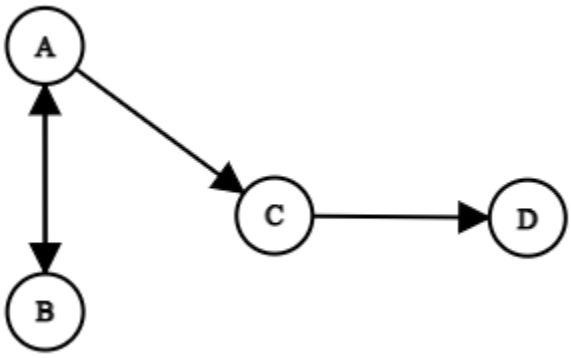

Figure 10 - Example 2 of Argumentation Framework graph

In the following sections we will present five mapping from Dung's Abstract Argumentation properties into CSP. The approach is the following: having as example the argumentation framework designed in Figure 10 - Example 2 of Argumentation Framework graph Figure 10, we will define a CSP instance which will be the representation of a property as a Choco-solver constraint. The solutions of that constraint are the extensions (sets of arguments) satisfying the desired properties. For each property expressed as a constrained, the arguments are declared as variables. Each variable can take two values: 0 or 1 having the following meaning: the arguments is or is not included in the extension, so the type of these variables is bool. The associated type in Choco-solver is *BoolVar*.

5.3.1 Conflict – free as CSP

The first and most relaxed constrained in an Abstract Argumentation Framework is conflict – free. As mentioned and described in the chapter Conflict – free set3.3, a conflict – free extension is a set of arguments that do not attack each other. In this case, having Figure 10 as example, there is an attack between arguments A and B (expressed as (A, B) ∈ R), this will be transformed in a constraint that states that the variables A and B cannot take the value 1 (or true) at the same time in the solution. Respecting the structure of a constraint, this one has the following form: ((A, B, C, D), (1, 0, 0, 1), (0, 1, 1, 0), (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1), (0, 0, 0, 0)). It is important to mention that the equivalent formula for the above constraint is A ⇒ ¬B, for (A, B) ∈ R.

Therefore, we can define the propositional formula for a conflict – free extension in an Argumentation Framework:

Definition 1: Let F(A, R) be an Argumentation Framework. A free – conflict constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall x_i \in S$, $D_i = \{0, 1\}$, C = { a ⇒ ¬b | (a, b) ∈ R }.

Applying the definition written above on example from Figure 10, the set C = { A ⇒ ¬B, B ⇒ ¬A, A ⇒ ¬C, C ⇒ ¬D }.

In Choco-solver there are defined methods to express propositional logic, so in the algorithm, we iterate over the attacks in the graph and *post* the constraint to the model.

5.3.2 Stable as CSP

As mentioned in in the subchapter Conflict – free set 3.10, a stable extension is a set S' of arguments with the property that the arguments of the set attack all the other arguments in the argumentation framework that do not belong to S'. For simplicity reasons, we will continue with the same approach as described in the previous sub subchapter to define the constraint, the domain for the variables and other specifications to express the transformation from abstract argumentation framework property to propositional logic.

Definition 2: Let F(A, R) be an Argumentation Framework. A stable constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall x_i \in S$, $D_i = \{0, 1\}$, C = { a ⇔ $\bigwedge_{b:(b,a)\in R} \neg b$ | a ∈ A }.

It is worth to mention that the Definition 2 has been inspired from (Creignou, 1995).

Illustrating the definition stated above about stable constraint on the Argumentation Framework example from Figure 10 we obtain the following set of constraints *C* from CSP definition:

C = { A ⇔ ¬B, B ⇔ ¬A, C ⇔ ¬A, D ⇔ ¬C }.

Solving the problem with the respective constraint from set C, the solution containing stable extensions is S = ((A, B, C, D), (1, 0, 0, 1), (0, 1, 1, 0)). Thus, the two stable extensions are {A, D} and {B, C}.

5.3.3 Admissible as CSP

In the subchapter 3.5 we described what an admissible extension means: a set is said to be an admissible extension if and only if there are no attacks between arguments of the set (is a conflict – free set) and for any argument from the set is attacked by an outsider, then another argument from the set or the attacked argument itself attacks the attacker. To be able to apply these, we need to transform it to propositional logic as stated in the following definition:

Definition 3: Let F(A, R) be an Argumentation Framework. An admissible constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall x_i \in S$, $D_i = \{0, 1\}$, C = { (a ⇒ $\bigwedge_{b:(b,a)\in R} \neg b$) ∧ (a ⇒ $\bigwedge_{b:(b,a)\in R}(\bigvee_{c:(c,b)\in R} c)$) | a ∈ A }.

It is important to mention that the Definition 3 has been inspired from (Leila Amgoud, 2011).

Applying this definition to the example designed in Figure 3Figure 10, we will obtain the following set of constraints C:

C = { B ⇒ ¬A , A ⇒ ¬B, (C⇒ ¬A) ∧ (C ⇒ B), (D ⇒ ¬C) ∧ (D ⇒ A) }.

Given the fact that arguments can take only values from set (0, 1), the solution of admissible constraint over the example presented in Figure 10 is: ((A, B, C, D), (0, 0, 0, 0), (1, 0, 0, 0), (0, 1, 0, 0), (1, 0, 0, 1), (0, 1, 1, 0)). Transforming the binary solution into argumentation framework arguments the resulted admissible sets are: {{}, {A}, {B}, {A, D}, {B, C}}.

5.3.4 Complete as CSP

The notion of a complete extension has been defined in the subchapter 3.8 as a set that does contain arguments with the property that it is conflict – free, it is admissible and in addition to these, all the arguments that are defended by any argument of this set belongs to respective set. The last restriction can be translated in other words as: an argument of a complete extension does not defend another argument that does not belong to the complete set. The propositional logic for the complete extension is written in the below definition:

Definition 4: Let F(A, R) be an Argumentation Framework. An complete constraint associated with defined F is a tuple (S, D, C) where S = A, $\forall\ x_i \in S$, D_i = {0, 1}, C = { $(a \Rightarrow \bigwedge_{b:(b,a)\in R} \neg b) \wedge (a \Leftrightarrow \bigwedge_{b:(b,a)\in R}(\bigvee_{c:(c,b)\in R} c))$ | a ∈ A }.

It is worth to mention that the Definition 4 has been inspired from (Leila Amgoud, 2011).

As we can observe, the constraints for admissible extension and complete extension are similar, the admissible constraint being more relaxed than the complete one.

C = { B ⇒ ¬A , A ⇒ ¬B, (C⇒ ¬A) ∧ (C ⇒ B), (D ⇒ ¬C) ∧ (D ⇒ A) }.

This expression could have been applied also as a propositional logic in the implemented algorithm, but due to the fact that one of the main target of the tool that we want to develop in within this dissertation work is performance, we do not solved this Argumentation Framework property as a CSP because it would have cost us extra memory and time, so the complexity of the algorithm would have increased a lot, besides that, we made use of the other definition of a complete extension also described and exemplified in detail in the subchapter 3.8. To be able to present the other definition, we need to remind what is the characteristic function in an Abstract Argumentation Framework. This concept has been detailed in the subchapter 3.7. Summarily, the characteristic function of a set of arguments is equal with the set of arguments that are defended by the first set. The definition of a complete extension that we used in the implementation is the following: a complete set S' is an admissible set with F(S') = S'. Therefore, to have a faster algorithm, we used this definition in our implementation. Having the admissible from the previous method in the algorithm, the next step is to calculate the characteristic function for each admissible resulted set and to determine those which are equal with the set for whose characteristic function is calculated.

Determining the characteristic function of the obtained admissible sets for the example provided in Figure 10, the results are the following:

F({B}) = {B, C}, admissible extension {B} is not complete;

F({A}) = {A, D}, admissible extension {A} is not complete;

F({A, D}) = {A, D}, admissible extension {A, D} is complete;

F({B, C}) = {B, C}, admissible extension {B, C} is complete.

5.3.5 Preferred extension

The definition of a preferred extension has been provided in the subchapter 3.6 as "a maximal (with respect to set inclusion) admissible set of AF" (Dung, 1995). For the same reason

as the previous approach with the complete extension, we did not use a constraint to determine the preferred extensions in the provided Argumentation Framework. Instead, we implemented this functionality using two maps:

- One having as key an argument and as value a list of arguments that the key argument;
- One having as key an argument and as value a list of arguments which attacks the key argument.

It is worth to mention that we used the same maps in the previous step when we calculate the characteristic function for a given set of arguments.

Bibliography

Dung, P. M. (1995). *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.* Bangkok 10501, Thailand: Division of Computer Science, Asian Institute of Technology, GPO Box 2754.

Alviano, M. (2021). The pyglaf argumentation reasoner. *ICCMA2021*.

Andreas Niskanen, M. J. (2020). µ-toksia: An Efficient Abstract Argumentation Reasoner. *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, 800-804.

Anthony Hunter, M. W. (2012). Aggregating evidence about the positive and negative effects of treatments. *Artificial Intelligence in Medicine*, 173-190.

Benjamin Delhomme, F. T.-Z. (2022). An interface between natural language and abstract argumentation frameworks for real-time debate analysis. *Decision Support Systems*.

Choco-solver. (n.d.). Retrieved from https://choco-solver.org/

Creignou, N. (1995). The class of problems that are linearly equivalent to Satisfiability or a uniform method for proving NP-completeness. *Theoretical Computer Science*.

D. Gale, L. S. (1962). College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 9-15.

Dung, P. M. (1995). *On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games.* Bangkok 10501, Thailand: Division of Computer Science, Asian Institute of Technology, GPO Box 2754.

Federico Cerutti, M. G. (2019). How we designed winning algorithms for abstract argumentation and which insight we attained. *Artificial Intelligence*, 1-40.

Floris Bex, M. S. (2013). Implementing the Argument Web. *Communications of the ACM*.

Francesca Rossi, P. v. (2006). *Handbook of Constraint Programming.*

Freuder, E. C. (1997). *Constraints.*

Jean-Marie Lagniez, E. L.-G. (2015). CoQuiAAS: A Constraint-based Quick Abstract. *IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*.

Leila Amgoud, C. D. (2011). Argumentation Frameworks as Constraint. *International Conference on Scalable Uncertainty Management*.