Imperial College London

Department Of Computing

Learning Efficient Argumentation Strategies

Author:
Krzysztof W. Huszcza

Supervisors:
Dr. Francesca Toni
Dr. Robert Craven

Second marker:
Graham Deane

June 18, 2013

Submitted in part fulfilment of the requirements for the degree of Master of Engineering in Computing of Imperial College London

Abstract

Argumentation is currently a flourishing field in Artificial Intelligence. Assumption-Based Argumentation (ABA) is a general-purpose argumentation framework suitable for supporting decision making in areas such as medicine. ABA represents knowledge as a set of logic statements and can be used to derive whether some other statement is supported by that underlying knowledge. However, it suffers a serious problem - all current implementations of the derivation procedure are slow, making argumentation unusable for face-paced environments such as medicine where decisions need to be made quickly.

The aim of this work was to speed up the derivation process. The procedure is non-deterministic - there are various paths we can take to reach the same conclusions. Hence, it is possible to define a strategy which guides us through the derivation process. By a smart encoding of derivation strategies and applying genetic algorithm on encoded representation, we managed to learn key characteristics of an efficient strategy. We then applied that knowledge to create a set of heuristics which were twice as fast as the current default derivation strategies when measures on a synthetically generated input.

Acknowledgements

I would like to thank my family and friends for their constant support during my studies at Imperial College. Alone I would not have been able to make it this far!

I am also grateful to my supervisors Dr. Francesca Toni and Dr. Robert Craven for their guidance throughout this project and constant enthusiasm which kept me going.

Above all else, I would like to thank my Grandmother who raised me and supported me throughout my whole life. I dedicate this work to you. Dziekuje Babciu.

Contents

Chapter 1

Introduction

1.1 Motivation

Throughout our lifetime, each of us solves many complex problems. They arise from our everyday activities, from our work or from our studies. Sometimes we consult with other people to find the solution, but usually, at the end of the day, it is us making the final decision. In order to make that decision, we use all the knowledge we have gathered about the problem. We obtain that data by answering questions like: when did the problem occur, what possibly could have caused it and how does it manifest itself. Unfortunately, the issue with some of the hardest problems is that the amount of data we have on them is unimaginably large.

Imagine you are a doctor trying to diagnose a patient who has just arrived to the hospital. Usually it is a routine for you, but his particular symptoms seem very strange. You run many tests on the patient and you gain more and more knowledge about his current state. But you are still unsure what exactly the problem is. All your past medical studies and experience together with the data you gained by examining the patient form your knowledge base about the problem. However, since you cannot find a solution, you must have some data missing. Maybe you forgot about something you have learnt or experienced? Or maybe your knowledge is simply insufficient? In any case, you need to consult other sources – your colleges, available medical journals or international medical community. Now the data possessed by all of these sources form your knowledge base. However, you run into more problems. Most importantly, your patient is dying and you do not have time to consult all of these available sources. It could take a lifetime to go through all medical journals or to consult all experts in the area. Even if you had the time to do so or you had the information organised more efficiently already, you now have to analyse all of these opinions and points of views. They are subjective – they may differ or even contradict each other. But your patient is dying - you need to make a decision and you need to do it fast.

Medicine is not unique in this type of problems. They occur in other areas as well. From our short story we can identify two things all of them have in common. First of all, they relay on huge amounts of data which may be scattered all over the world and represented in different ways. Secondly, they require analysing these big volumes of (possibly inconsistent) data. Does it sound familiar? For anyone who has taken computer science degree it should. We have built computers to solve exactly this type of issues. And computers proved many times in the past that they are very good at solving them. All we have to do is build a decision-supporting computer system, which would take all of the information as an input, analyse it and produce a decision (a solution to the problem) as its output. Such systems have been built in the past and are being used every day.

Unfortunately, as our short story indicates, they are further constraints. First of all, our system needs to deal with inconsistencies in the data, e.g. differing opinions. Secondly, in areas such as medicine, the decision-making process needs to be fast. Finally, a human being has to be confident with the decision, so, ideally, the system should be able to justify its own output. Assumption-Based Argumentation (ABA) has been suggested as a possible approach to tackle all of the described issues.

1.2 Argumentation

Argumentation is a study of how conclusions could be reached using reasoning. Assumption-Based Argumentation is a logic framework built on top of that notion and used to support decision-making. [DKT09] The knowledge base is represented as a set of logic statements which describe: known opinions, known disagreements between opinions (i.e. counter-opinions) and, finally, inference rules which specify how further conclusions can be derived from what we already know. The framework is used to prove or disprove statements basing on its knowledge base. If it manages to prove a certain statement, we gain further confidence in the truth of that statement which may be very useful for our decision-making process.

For instance, in our medicine example, the doctor could input all the symptoms and a statement such as 'the patient has brain cancer'. The system would then try to validate that statement and if it manages to prove it, the doctor could gain further confidence that the brain cancer diagnosis is correct. The process of conducting that proof is very similar to a real argument two people could have in everyday life. Imagine a student and a lecturer arguing over an assignment mark. Lecturer's initial statement is that the student should get a C for the assignment. He supports his opinion by pointing out a crucial piece of material which student did not cover in his answer. The student tries to defend himself and attacks back by pointing out that the material was not covered anywhere during the lectures. It is the lecturer's turn to defend. He does so by arguing that the material was, in fact, covered during tutorial sessions. The dispute ends if one side manages to defeat all arguments of the other side.

Assumption-Based Argumentation works using the same principle. For a given input statement, the system tries to find all opinions recorded in the ABA knowledge base (also called ABA framework) which support that statement. Then it tries to attack all of these opinions by finding any disagreeing counter-opinions and so forth. Thus, for an input statement 'the patient has brain cancer' it constructs a full argument of whether the patient really has a brain cancer or not. The doctor could later view the whole argument – all supporting opinions, counter-opinions etc. to gain further confidence in the result of derivation. We will present more formal definition of the argumentation framework in chapter 2.

1.3 The problem

One crucial problem which is still not solved in the argumentation theory is the derivation time. The knowledge bases, such as the medical knowledge base, consist of millions of statements. Many areas (e.g. medicine) may require the answer to be found within minutes, if not seconds. Our work aims to tackle the problem of efficiently computing argumentation dispute. As we will see, at each stage of the derivation process we may have multiple choice points. For example, when we are defending our argument, we could have many possible ways in which we could attack our opponent's counter-argument. There may be one way which would literally take ages to compute as it leads to further long dispute. However, we may have a second, faster way of disproving opponent's arguments. Since we only need to prove him wrong once in order to invalidate his argument, we may considerable speed up the derivation process if we choose faster way to do so. So the problem narrows down to having some sort of efficient heuristics which would tell us which option to choose. The aim of this work is to find a way of obtaining these heuristics for ABA knowledge bases.

1.4 Contributions

The key fact which we will emphasise several times in the report is that our goal was to **learn** the efficient derivation strategies. Thus, we proposed a mathematical model which allows us to encode an arbitrary derivation strategy in terms of various measures. The primary type of measures we have investigated are graph measures. We investigated both the well-established centrality measures as well as measures which we proposed ourselves. We can pre-compute them by translating our knowledge base from an 'argumentation representation' to a connected graph. We applied a genetic algorithm to learn how each graph measure influences derivation speed.

We identified measures which have high positive correlation with performance and we performed additional experiments to discover and exemplify why exactly those measures are important to the derivation speed. We also identified measures and derivation strategies which did not show any correlation with performance. We then benchmarked the output of our learning on 400 synthetically generated and 30 real-world medical knowledge bases ('frameworks') against current default strategy.

For synthetically generated frameworks, the heuristics which we used on average improved the derivation speed - they were 2 times faster than the default strategy for small networks and about 1.5 times faster for medium and large frameworks. We also compared their performance with a set of strategies which were used to generate the synthetic frameworks in the first place. These strategies are considered to be 'good' or 'correct' strategies for solving a given synthetic framework as they are guaranteed to find a solution within some preset time-bound. In this case, for small networks our heuristics proved to be 3 times faster and for medium and large frameworks both groups had a very similar performance. Thus, we demonstrated that our solution is efficient and generic.

1.5 Summary of the report

In order to explain the derivation procedure and our approach to improving it, we first need to cover the basis of argumentation theory. We also need to refresh the reader's knowledge about Graph Theory and genetic algorithms. Thus, we begin by introducing necessary background information in chapter 2. We then move on to discuss the derivation problem in a bit more detail and then explore our approach to solving it (chapter 3). Before diving into the experiments, we will have a short detour where we talk about our experimental infrastructure (chapter 4). We then move on to the main part of the report where we discuss learning experiments we performed in order to find correlations between various measures and derivation speed (chapter 5). The correlations found are further validated by benchmarking them against current default strategies (chapter 6). We conclude with discussion about further work which can be done in this area (chapter 7).

Chapter 2

Background

2.1 Assumption-Based Argumentation

Broadly speaking, argumentation is a process of using logical reasoning to reach conclusions. It is currently a flourishing field in Artificial Intelligence and Logic. In his original paper, Dung [Dun95] summarised argumentation by an old saying:

The one who has the last word laughs best.

Assumption-Based Argumentation (ABA) is a general-purpose argumentation framework. Its various applications include decision-making, default reasoning and legal reasoning. [Ton12] It has also been applied to solving real world problems, such as medical decision-making. [CTC+12] Intuitively, ABA is a system which represents certain knowledge as a set of logic statements and can be used to infer whether some other statement (an input query) is supported by that underlying knowledge or not. In order to define ABA, we need to define two main underlying concepts – the representation of knowledge in ABA and the derivation process of a random query. To better explain the concept and provide some common intuition, we will first try to define ABA informally and then we will give a formal definition.

2.1.1 Definition of ABA framework

ABA framework (i.e. underlying knowledge base) consists of a set of logic statements which can be divided into:

- assumptions
- non-assumptions
- inference rules
- contraries

Assumptions are sentences in language which are open to challenge. [DKT09] They are uncertain, hence attacks will be always directed at them. A statement "It will rain" is a typical example of an assumption.

Non-assumptions are either certain statements (i.e. facts) or conclusions of a set of assumptions or non-assumptions (as specified by a inference rule). [DKT09] An example of non-assumption could be statement "John is a lecturer in computer science".

Inference rules specify how certain conclusions (non-assumptions) may be inferred from a set of assumptions and non-assumptions. [DKT09] In other words, they denote that a non-assumption is a consequence of some assumptions and non-assumptions. For instance, a statement "If John is a lecturer in computer science, then he is an expert in computer science" is a good example of an inference rule. We will usually denote them in the following way:

$$expert(john, computer\_science) \leftarrow lecturer(john, computer\_science)$$

We will usually refer to the left hand side of the inference rule as the head or the conclusion of the rule. Note that assumptions will never occur as the head of rules. Furthermore, we will usually refer to the right hand side of the inference rule as the body or the premises of the rule. Both, assumptions and non-assumptions can be among premises. An empty inference rule, such as this one:

$$lecturer(John, computer\_science) \leftarrow$$

indicates a fact. Facts always hold and will never be attacked by our opponent.

Finally, we have a set of **contraries**. Contrary is a sentence which represents a challenge against an assumption.[DKT09] Intuitively, it is a non-assumption or other assumption which disagree with a given assumptions. For instance, if our assumption is "The UK economy is strong" a non-assumption contrary to that particular assumption may be: "UK's unemployment level is big". We denote the contrary in the following way:

$$\overline{strongEconomy(UK)} = bigUnemployment(UK)$$

Definition 1 *Formally, an ABA framework is a tuple* $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^- \rangle$*where*

- *a pair* $(\mathcal{L}, \mathcal{R})$ *is a deductive system with language* $\mathcal{L}$ *and a set of inference rules* $\mathcal{R}$
- *a set of assumption* $\mathcal{A}$, $\mathcal{A} \subseteq \mathcal{L}$
- *a total mapping* $^-$ *from* $\mathcal{A}$ *to* $\mathcal{L}$*, where* $\bar{a}$ *is referred to as the contrary of* a

Having defined all the basic buildings blocks of ABA framework, we can now go ahead and define what we mean by an argument and by an attack.

2.1.2 Arguments and attacks

Arguments are deductions of claim (conclusion) supported by a set of assumptions. [DKT09] Intuitively, an argument is a possible derivation of a certain conclusion (non-assumption). The conclusion we want to derive may be a statement such as "John is good at playing Starcraft". We may represent it as a non-assumption $goodAt(john, starcraft)$. In order to deduce that non-assumption, we need to find an inference rule where this non-assumption is headed. Suppose we find a rule:

$$goodAt(john, starcraft) \leftarrow plays(john, starcraft), noLife(john).$$

In order to deduce the conclusion of that rule, we need to deduce all of the premises as well. We start with another non-assumption $plays(john, starcraft)$. We find the following rules:

$$\begin{aligned} plays(john, starcraft) &\leftarrow isComputerGame(starcraft), isPlayer(john) \\ isComputerGame(starcraft) &\leftarrow \\ isPlayer(john) &\leftarrow \end{aligned}$$

$plays(john, starcraft)$ can be further expanded into non-assumptions which are facts and always hold. Thus we look at the second premise - $noLife(john)$. It turns out to be an assumption. Since assumptions are uncertain by definition, we cannot prove them by further expanding them. We can think of it as if there were no inference rules with assumptions being the head in ABA Frameworks. In practice, we always have one rule for each assumption which is of the form $asm \leftarrow asm$ and comes handy when we query the ABA framework for assumptions.

Therefore, we are done. We have now constructed an argument for a conclusion $goodAt(john, starcraft)$ support by a set of assumptions $noLife(john)$. We used a top-down approach to construct that argument – i.e. we started from the conclusion and expanded it through inference rules until there was nothing more to prove. The process of constructing an argument is called a derivation. Note that we explicitly list only the assumptions we have used to construct the argument, omitting any facts. The assumptions are important because they are uncertain. Our argument relies on an uncertain believe that John has no life (outside of computer gaming). Thus, the argument is defeasible [DKT09] – there is a possibility of defeating it. In order to do so our dispute opponent may try to attack it by disproving the claim that John has no life. We will come back to attacks shortly. We will always denote arguments in the following way:

$$Arg = \{\ [Set\ of\ SupportingAssumptions]\ \} \vdash\ Conclusion$$

The easiest way to understand arguments is to visualise them as trees. An argument may be represented as a tree, with the claim being at the root of the tree. The nodes in the tree are either assumptions or non-assumptions. The edges are formed by the inference rules, with parent being the head of the rule and children being the premises. For instance, consider our John and Starcraft example above:

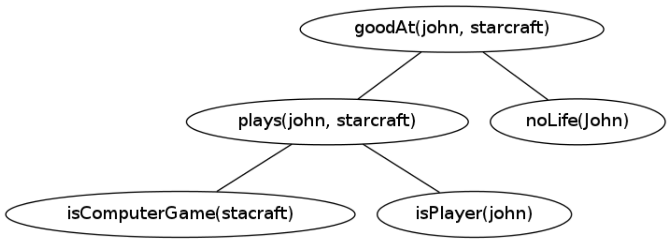


Figure 2.1: Starcraft argument tree

The notion of an attack is defined in terms of the contraries. We have previously derived argument (lets call it $A1$)

$$A1 = \{noLife(john)\} \vdash goodAt(john, starcraft)$$

We have also said that, because of the assumption $noLife(john)$ there is a potential for constructing an attack to defeat that argument. Our opponent will be able to construct an attack only if he manages to find a contrary with assumption $noLife(john)$ on the left-hand side. Suppose such a contrary exists:

$$\overline{noLife(john)} = hasGirlfriend(john)$$

i.e. if John has a girlfriend than clearly he must have some life outside of computer gaming. Now, all our opponent has to do is constructing an argument for the claim $hasGirlfriend(john)$. He will be using exactly the same proof method we have used to construct $A1$. Suppose he comes up with:

$$A2 = \{handsome(john), hasFreeTime(john)\} \vdash hasGirlfriend(john)$$

We now can say that our opponent has constructed an attack, i.e. his argument $A2$ attacks our argument $A1$.

Definition 2 *More formally, an argument $S2 \vdash c2$ attacks argument $S1 \vdash c1$ if and only if $c2$ is the contrary of an assumption in $S1$. [DKT09]*

This is clearly the case here, since the conclusion of $A2$ ($hasGirlfriend(john)$) is a contrary of an assumption in $A1$ ($noLife(john)$). However, we have not been defeated yet. Opponent's attack relies on an argument which is defeasible. It contains two uncertain assumptions - $handsome(john)$ and $hasFreeTime(john)$, which we, in turn, may try to attack. And so the dispute continues.

2.1.3 Acceptability of arguments and assumptions

We have previously stated many times that the Assumption Based Argumentation is used to prove or validate certain claims (inputs) by analysing its underlying knowledge base. However, we have not yet defined what we mean by the claim to be proved or valid. Actually, the literature refers to the process of 'accepting' the claim. Thus, an ABA framework is used to determine whether the input claim is to be accepted or not. [DKT09]

In order for the claim to be accepted, each argument generated by the dispute must be 'acceptable'. There are different notions regarding the acceptability of arguments. Each of them leads to a slightly different dispute. We will focus here on only two notions of acceptability of arguments: the set of generated arguments is acceptable either if it is **admissible** or if it is **grounded**. We will define both notions in terms of an attack. So, first of all:

Definition 3 *A set of arguments ArgSet1 attacks a set of arguments ArgSet2 if any argument in ArgSet1 attacks any argument in ArgSet2. [DKT09]*

Now we are ready to define the notions of acceptability of a set of arguments.

Definition 4 *A set of arguments is:*

- *admissible if it does not attack itself and it attacks every set of arguments that attacks it;*
- *complete if it is admissible and contains all arguments that it defends;*
- *grounded if its a minimal complete set (and so it is unique); [DKT09]*

Equivalently, we can also use the same notions for assumptions. We can say that the initial claim is acceptable if the set of assumptions supporting and defending the claim is acceptable (so is either grounded or admissible). [DKT09]

The definitions may seem hard to grasp but we will try to make them much more clearer by supplying an example. Consider Figure 2.2 (left). Suppose each node in the graph is an argument and an arrow indicates attacks between arguments. We will start with admissible sets. We are looking for sets of arguments which counterattack each attack against them and do not attack themselves. An empty set is a trivial example. Other examples are: [a], [a,c], [a,d], [d].

Now we can look for complete sets. Since we now that they are admissible, we only need to consider admissible sets. The first important bit in the definition of a complete set is the notion of defending an argument. An argument a is defended by set S if S 'neutralises' (attacks) all arguments attacking a. In our running example [d] defends itself, [a] does not defend [c] as it does not 'protect' it against d (it does not attack d) and so on. Also we know that nothing attacks a. Hence the definition holds trivially - every subset of arguments which is not attacked by a defends a.

The second important bit in the definition of a complete set is that it contains **all** arguments it defends. Thus [d] is not complete since it does not contain a and we said that it does defend a. In fact, since a is not attacked by anything, all complete sets must contain a. Hence the complete sets here are [a] and [a,c]. Grounded is minimal complete so, in this case, its [a].

One of the key differences between grounded and admissible is illustrated in Figure 2.2 (right). Both [d] and [c] are admissible sets. They are also complete. However, they are not grounded (an empty set is). Note that Figure 2.2 (right) is an example of an infinite dispute in which proponent and opponent shout at each other forever. Such a dispute could be admissible (i.e. proponent's arguments would be admissible and opponent's arguments would be admissible) but would not be grounded. This is the critical difference between admissible and grounded. Since the dispute never ends, admissible derivation algorithm has to apply special mechanism called filtering (or rather a special type of filtering) to make the dispute finite. Grounded derivation does not implement this type of filtering and would loop here forever.

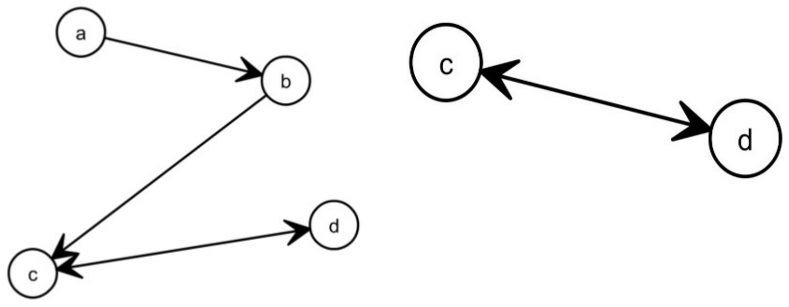


Figure 2.2: Argument graphs

Another way to look at the acceptability is by considering acceptable assumptions not arguments. Actually, the definitions are very similar:

Definition 5 *A set of assumptions A attacks another set of assumptions B if there is an argument supported by any subset of A which attacks an argument which is supported by any subset of B.*

So, if one argument attacks another, we also say that its assumptions 'attack' the other argument's assumptions. [Ton12] The definitions for admissible and grounded set of assumptions follow from the notion of attack (i.e. are identical to the definitions for arguments).

2.1.4 Dispute

We have now defined the ABA framework together with the most important notions of arguments, attacks and acceptability. We are now ready to move to the second side of the coin which is the dispute process itself. We will firstly show how the dispute may be represented as a tree, the so called "dispute tree". [DKT09] Then we will demonstrate the way in which we can generate an approximated dispute tree. The process is called 'dispute derivation' and interleaves constructing the arguments and determining their acceptability. [DKT09] The latter indicates that we will actually have two slightly different dispute derivations – one for admissible and one for grounded arguments.

The easiest way to understand argumentation dispute is to think about it as a game between two players. The game is similar to a real-life argument people (too) often have. The input query is the subject over which the players are arguing. Proponent is the player who proposes the input query and his goal is to defend it against the other player's – the opponent's attacks. Thus the game progresses in turns: first the proponent proposes a subject of the dispute, the opponent tries to attack by forming counter-arguments contrary to the input subject, the proponent defends himself by forming counter-arguments to opponent's counter-arguments and so forth. Unlike the real-life dispute, this one involves two computer agents so we are guaranteed that all proposed arguments are valid and, if there is a solution, the participants will both agree on it (both usually do not hold for human participants, they never hold for politicians). [Ton12]

Dispute trees

We can represent proponent's winning strategy as a dispute tree. [DKT09] The main characteristics of a dispute tree are:

- there are two types of nodes – a proponent node and an opponent node corresponding to the moves each of the players does
- each node corresponds to an argument formed by a given player
- the root is a proponent node with proponent's initial argument supporting his input claims
- for each proponent node N with argument X and for each opponent argument Y such that Y attacks X, there exists a child opponent node (child of N) with argument Y
- for each opponent node X there exists exactly one proponent node with argument Y, such that Y attacks X

The last two statements tell us that the opponent will be attacking proponent's arguments in all possible ways. The proponent, on the other hand, needs only one way to defend against opponent's attack. Both players use assumptions when constructing their arguments. Following our earlier intuition for admissible and grounded sets of arguments, we can say that a dispute tree is admissible if and only if there is no argument labelling both an opponent and a proponent node. It is grounded if and only if it is finite. [DKT09] Note that a grounded dispute tree is also admissible.

Lets construct a dispute tree using our ongoing Starcraft example:

$$\begin{aligned}
goodAt(john, starcraft) &\leftarrow plays(john, starcraft), noLife(john), hasReflex(john)\\
plays(john, starcraft) &\leftarrow isComputerGame(starcraft), isPlayer(john)\\
hasGirlfriend(john) &\leftarrow relationshipOnFacebook(john, marry), girl(marry),\\
&\quad\; gaveCardOnValentines(john, marry)\\
clumsy(john) &\leftarrow splitCoffeeOnComputer(john)\\
clumsy(john) &\leftarrow brokeLegOnBannanaSkin(john)\\
isComputerGame(starcraft) &\leftarrow\\
isPlayer(john) &\leftarrow\\
relationshipOnFacebook(john, marry) &\leftarrow\\
girl(marry) &\leftarrow\\
tookPity(john, marry) &\leftarrow\\
doesNotDrinkCoffee(john) &\leftarrow\\
neverHadBrokenLeg(john) &\leftarrow\\
\overline{noLife(john)} &= hasGirlfriend(john)\\
\overline{gaveCardOnValentines(john, marry)} &= tookPity(john, marry)\\
\overline{hasReflex(john)} &= clumsy(john)\\
\overline{splitCoffeeOnComputer(john)} &= doesNotDrinkCoffee(john)\\
\overline{brokeLegOnBannanaSkin(john)} &= neverHadBrokenLeg(john)
\end{aligned}$$

We are trying to prove whether John is good at Starcraft ($goodAt(john, starcraft)$. The proponent construct an argument which concludes the input query ($\{noLife(john), hasReflex(john)\} \vdash goodAt(john, starcraft)$). The opponent now tries to attack that argument by looking for contraries of assumptions used in proponent's argument and trying to build arguments with these contraries as heads. The contrary of $hasReflex(john)$ is $clumsy(john)$ and the contrary of $noLife(john)$ is $hasGirlfriend(john)$.

Note that the we can build two different arguments which support $clumsy(john)$ which the opponent does. Then the proponent counter-attacks in the same way and so the game continues. In this case, the dispute ends quickly. Usually, dispute trees contain thousands of nodes. The resulting tree is presented in Figure 2.3.

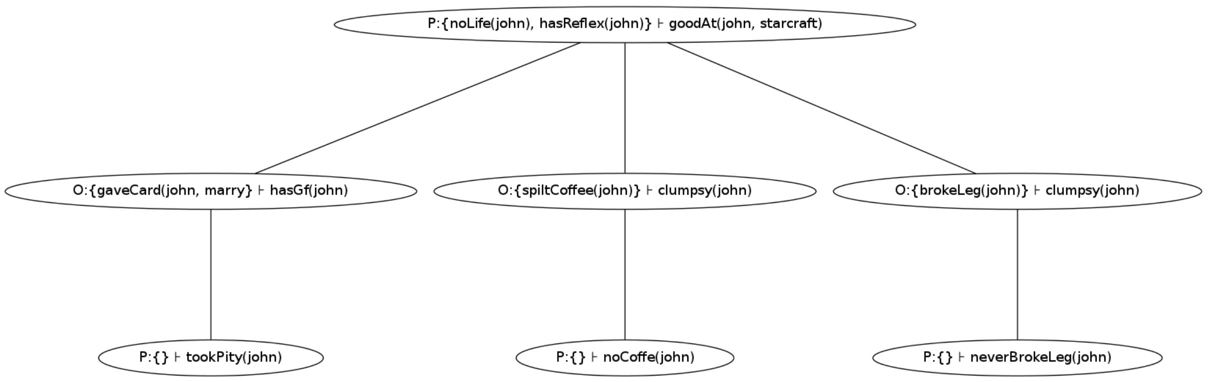

P:{noLife(john), hasReflex(john)} ⊢ goodAt(john, starcraft)
O:{gaveCard(john, marry} ⊢ hasGf(john)
O:{spiltCoffee(john)} ⊢ clumpsy(john)
O:{brokeLeg(john)} ⊢ clumpsy(john)
P:{} ⊢ tookPity(john)
P:{} ⊢ noCoffe(john)
P:{} ⊢ neverBrokeLeg(john)

Figure 2.3: Starcraft dispute tree

Dispute Derivations

As we have already mentioned, a dispute derivation may be understood as a game between two players – a proponent and an opponent. We present **structured X-dispute derivation** - the variant which we will be trying to speed up. [Ton12] We will focus on two different dispute derivations derived from the notion of acceptability – admissible derivation and grounded derivation. On success, admissible derivations return an admissible set of assumptions supporting the input claim. They generate admissible sets of arguments for both - proponent and opponent. They also approximate an admissible dispute tree. Grounded derivations, on the other hand, will generate a grounded set of assumptions supporting the input query, grounded set of arguments for each of the players, and will approximate grounded dispute trees.

Both dispute derivations will make use of several filtering mechanisms. We will define four filtering techniques in total. Generally speaking, the purpose of filtering is to avoid redundant computations. It may happen that a certain assumption was already used by either the proponent or the opponent. To avoid unnecessary re-computations, we will store all proponent's assumptions which were attacked by the opponent, we will call them **defences**, and all opponent's assumptions which were attacked by the proponent, which we will call **culprits**. [DKT09]

Filtering methods used will slightly differ for each type of derivation. Grounded derivations will only make use of two filtering methods, whereas admissible derivations will use all four. The following filtering methods are used in both admissible and grounded dispute derivations:

- of culprits by defences;
- of defences by culprits;

Remember that a dispute tree is admissible when we have no arguments labelling both the proponent and the opponent node (i.e. arguments attacking themselves). The condition is also a necessary (but not sufficient) condition for a grounded tree. The above filtering rules were created to enforce the admissible condition. When the proponent is choosing an assumption to attack, he must check whether he has not used that assumption before in his argument. Otherwise, he would be attacking his own argument.

Similarly, the opponent must check whether he is not attacking the assumptions he has used before in his argument.

Additional two filtering methods are only used in admissible dispute derivations:

- of culprits by culprits;
- of defences by defences;

They ensure that we do not use the same defence or culprit twice in the derivation. If they are allowed to repeat then an infinite loop would be created and the dispute would continue ad infinitum. Remember that infinite disputes are actually acceptable for admissible semantics. These filtering rules are used in admissible derivation process to ensure that the computation terminates. They are not used in grounded derivations, because grounded derivations on success are finite by definition which means that neither defences nor culprits would actually repeat. Of course, in practice, when we are running the algorithm and proposing some input query, we do not know whether the answer is grounded or not (i.e. whether the derivation will succeed or fail). This is exactly what we want to compute. So if we set the flag to grounded in the algorithm and run the computation, the computation may loop forever.

A single step in a dispute derivation corresponds to a move by either an opponent or a proponent. Four data structures are maintained at each step: [DKT09]

- a set P of proponent's 'arguments under construction' (also called potential arguments);
- a set O of opponent's 'arguments under construction';
- a set C of culprits;
- a set D of defences;

Potential arguments are partially constructed arguments which did not fully expand all non-assumptions yet. Intuitively, if you go back to section 2.2 where we defined arguments as trees, a potential argument would be equivalent to a subtree of an argument tree such that it contains the root of that argument tree. During the derivation arguments are represented as three-element data structures:

$$arg: \quad \{[marked\_set],\ [unmarked\_set],\ conclusion]\}$$

The conclusion is the sentence which the argument is proving (i.e. the root of the argument tree representation). The unmarked set consists of assumptions and non-assumptions which are yet to be expanded. Yet again recall argument tree representation. When we are building the argument we are growing the tree from the root. At each step we have a subtree with root as the conclusion. The unmarked set members are all leafs of that subtree. I.e. there are yet unexpanded assumptions/non-assumptions which we can expand at the next stage. When the unmarked set is empty then we have finished building the argument. We expand the argument by choosing assumptions or non-assumptions from the unmarked set and, in case of non-assumptions, substituting them with a body of a rule in which they occur as head. Note that unmarked set may grow and shrink during the derivation. The marked set size consists of only assumptions. It contains assumptions which were already selected from the unmarked set. At the end of the argument creation, it will contain all assumptions supporting the argument.

To illustrate, consider our Starcraft argument (Figure 2.1). When we were building the argument our marked and unmarked sets could have evolved as follows:

1. [ ], $[goodAt(john, starcraft)]$
2. [ ], $[plays(john, starcraft),\ noLife(john)]$
3. [ ], $[isComuterGame(starcrat),\ isPlayer(john),\ noLife(john)]$
4. [ ], $[isPlayer(john),\ noLife(john)]$
5. [ ], $[noLife(john)]$
6. $[noLife(john)]$, [ ]

The admissible dispute derivation may be represented graphically as in Figure 2.4 [Ton12]. The dispute ends when we have nothing else to prove: P and O are both empty. In this case we have succeeded and so we return an admissible set of assumptions - D. Otherwise, we proceed in steps.

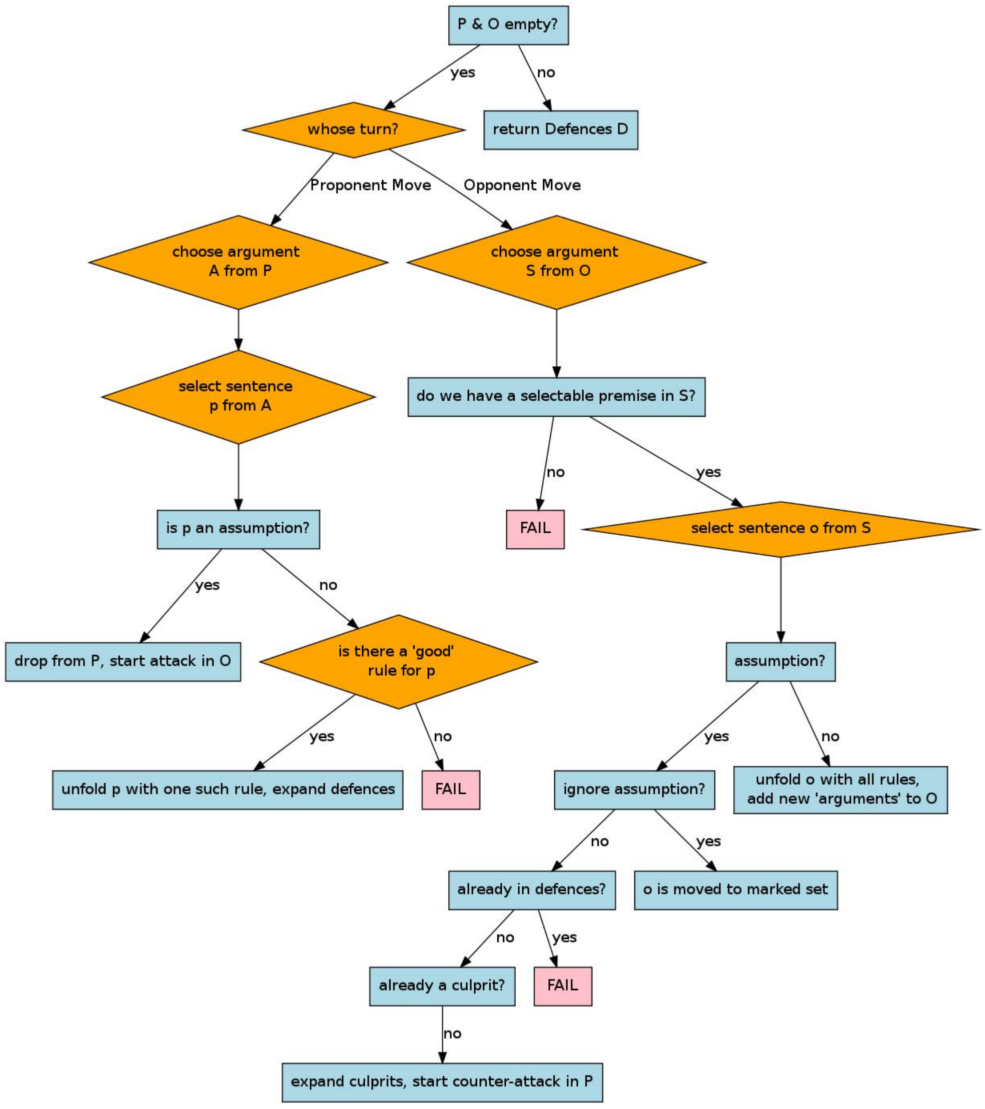


Figure 2.4: Simplified dispute derivation algorithm. It is slightly different version than presented in [Ton12] - we added an explicit argument choice node to bring it closer to the actual implementation

We first determine who moves in the current step. If it is proponent's turn, we select one argument (A) from his argument set P and we select one sentence (p) from A. If the sentence is an assumption, we drop it from P and start attack in O. Note that in this case the sentence would already be in the defences set before we select it, as we expand defences set when we expand rule bodies. If p is not an assumption, we look for any rules with p being a head. If there is at least one such rule and the rule does not contain any culprits in its body (i.e. it is a 'good' rule), we unfold p by replacing it in P with rule's body and we add any new assumptions in rule's body to defences.

If it is opponent's turn, he begins by selecting an argument S from the set of his available arguments O. If the argument's unmarked set is empty, i.e. there are no sentences available to pick, we fail. Otherwise, the opponent selects one sentence o from S. If it is a non-assumption, he unfolds the non-assumption in all possible ways (i.e. he attacks in all possible ways). Each different unfolding of a non-assumption forms a new opponent argument, which we add to O.

If o is an assumption we have an option to ignore it. Once ignored, o is moved to a marked set of opponent's argument and will never be considered again. If we do not ignore o, we filter it by defences and by culprits. If it is not a member of any of the two, we add it to culprits and start an attack in P (we add the contrary of o to P).

Choice points

The dispute derivation algorithm we have presented in the previous section has number of points where we can explore different execution paths. These points are the key elements in our effort to speed up the execution of the derivation as exploring certain expensive paths may be avoided by using useful heuristics. In Figure 2.4 the choice points are marked by orange diamonds. There are 6 all together:

- player turn choice
- proponent argument choice
- opponent argument choice
- proponent sentence choice
- opponent sentence choice
- proponent rule choice (opponent expands non-assumptions using all possible rules)

Backtracking

The algorithm presented in Figure 2.4 implicitly employs a mechanism called backtracking. Each time we fail we have a possibility of 'moving back' in the derivation procedure and trying out other derivation paths. Although we have 6 choice points, in current implementation there are only 2 places to which we can backtrack - proponent's rule choice and opponent's ignore choice (which was not included as a choice point). For the former, we may backtrack and try out a different rule to prove a sentence.

Ignoring needs to be explained a bit more carefully as it is more of an 'implementation detail'. Suppose the opponent expands argument S and chooses certain assumption 'a' from S which is added to culprits and its contrary is added to proponent set. Suppose, however, that the proponent is unable to build a valid argument which attacks 'a'. He fails but the whole derivation is not finished yet. There may be other assumptions beside 'a' in S which proponent may try to attack. This is where ignoring is useful. We backtrack to 'a' ignore point and this time we ignore it (we push it to S marked set) in order to move to other assumptions in S.

2.1.5 ABA Graph

In order to speed up the derivation process we will be representing an ABA framework as a graph and computing various measures for that graph. The analysis of graph measures will guide us towards developing useful heuristics for the dispute derivation process. We will call such a representation an ABA Graph.

Definition 6 *Given an ABA framework represented as a tuple $\langle \mathcal{L}, \mathcal{R}, \mathcal{A}, ^{-} \rangle$, let: [Str12]*

- $V_1 = \mathcal{R}$ *- the set of ABA Framework rule nodes*
- $V_2 = \mathcal{A}$ *- the set of ABA Framework assumption nodes*
- $V_3 = \mathcal{L} - \mathcal{A}$ *- the set of ABA Framework non-assumption nodes*

and:

- E_1 *is the set of support edges, where support edge is an edge from a rule node to each single premise in the rule body (either an assumption node or a non-assumption node);*

- *E_2 is the set of proof edges, where proof edge is an edge from a non-assumption node to a rule node where the given non-assumption is headed;*
- *E_3 is the set of attack edges, where attack edge is an edge from an assumption node (call it A) to an assumption/non-assumption node (call it C), such that C is the contrary of A;*

ABA Graph $G = \langle V, E \rangle$is defined by

- *$V = V_1 \cup V_2 \cup V_3$ - a set of vertices of three types*
- *$E = E_1 \cup E_2 \cup E_3$ - a set of edges of three types*

ABA Graph corresponds directly to ABA framework which we have described in section 2.1. There are three types of vertices (nodes) which correspond directly to the building blocks of ABA frameworks. There are also three types of edges corresponding to the notions of arguments and attacks.

The graph is directed by definition. Also, the direction of an edge corresponds to the dispute derivation direction. Support edges and proof edges correspond to expanding potential arguments whereas attack edges correspond to formulating attacks against arguments. The graph does not need to be connected. It might contain a couple of disconnected components with no nodes attacking or supporting a node from other component. It may also contain cycles.

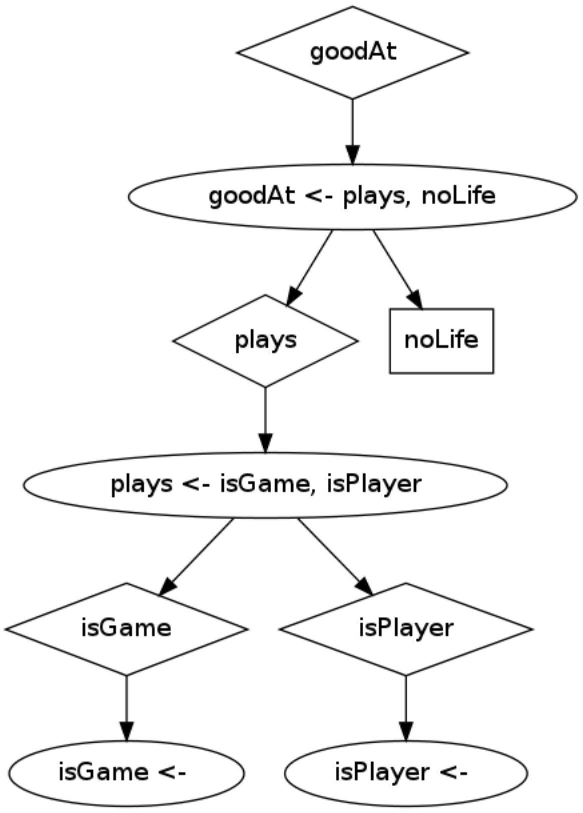


Figure 2.5: Starcraft example ABA Graph representation

Let us consider a simplified example of our previous Starcraft example (assumptions and non-assumptions were shorten to fit into the graph):

$$\begin{aligned} goodAt &\leftarrow plays, noLife \\ plays &\leftarrow isGame, isPlayer \\ isPlayer &\leftarrow \\ isGame &\leftarrow \end{aligned}$$

Non-assumptions are: *goodAt*, *plays*, *isGame*, *isPlayer*. Assumptions are: $noLife$. This get directly translated to the graph:

- the diamond nodes represent non-assumptions
- the box nodes represent assumptions
- the ovals represent rules

The corresponding ABA Graph is presented in Figure 2.5.

2.2 Graph Theory - centrality measures

Centrality measures determine the importance of a node in a graph. [New10] The bigger the centrality measure, the more 'central' (important) a given node is. In order to compute most of the centrality measures we have to use adjacency matrix representation of a graph as we will be calculating eigenvalues and eigenvectors of that matrix. The ABA Graph analysis will be performed once to find measures for all nodes. The produced measures will be then fed to the dispute derivation algorithm to guide the algorithm's execution at each point of choice. Hence, the measures which we present will only have to be computed once, off-line (i.e. we do not have to compute them for each input claim during the derivation). This is extremely important, since the matrix storing the graph will be very sparse. Consequently, computing eigenvalues and eigenvectors of a sparse matrix is expensive. However, each time we update an ABA framework, we also have to recompute all its measures.

2.2.1 Degree centrality

Degree centrality measures the number of edges connected to a node. We will be measuring both – degree-in centrality and degree-out centrality. Degree-in centrality counts the number of edges pointing to the given node, whereas degree-out centrality computes the number of edges coming from the given node. Degree centrality is the simplest of our measures and the most straight-forward to compute.

2.2.2 Eigenvector centrality

The problem with degree centrality is that it gives equal scores to each neighbour. However, intuitively a given node may be more important if a few highly important nodes point to it than if hundreds of unimportant nodes point to it. So we can slightly change the centrality measure to cater for that fact. When summing up the number of in-going or out-going edges, we assign weights to each of them. Each weight is equal to the centrality of a given neighbour. Eigenvector centrality is the weighted sum we obtain.

Eigenvector centrality can be computed as follows: [New10]

$$x_i = k^{-1} \sum_j A_{ij} x_j$$

where x_i, x_j are neighbouring nodes, A is the adjacency matrix and k is the largest eigenvalue of A

2.2.3 Katz centrality

The problem with eigenvector centrality is the computation. If we consider in-going edges only, a node with no other nodes pointing at it will have an eigenvalue centrality equal to 0. Hence, if that node itself points to other nodes, its weight to its neighbours' centrality will also be 0. The problem affects graphs which do not have any cycles. In such a graph, we will always have at least one node with eigenvector centrality equal to 0. Thus, the measure basically becomes useless for graphs without any cycles. It is an issue for ABA graph, because 0-cycle graphs can occur for some input frameworks.

Katz centrality addresses the problem. In order to get rid of 0-centrality nodes, we pre-assign a small centrality value to each node. The formula for computing Katz centrality becomes: [New10]

$$x_i = k^{-1} \sum_j A_{ij} x_j + \beta$$

where β is the initial centrality we give each node 'for free'.

Unfortunately, the software framework SNAP [Sta] which we will be using to compute all graph measures does not provide Katz centrality.

2.2.4 PageRank

Katz centrality has one disadvantage – if a node with high Katz centrality points to many other nodes, they will also get high Katz centrality. PageRank addresses that problem. The centrality we derive from our neighbours is proportional to their centrality, but we will divide that proportion by neighbour's out-degree.

The formula for computing PageRank is: [New10]

$$x_i = k^{-1} \sum_j A_{ij} \frac{x_j}{d_j^{out}} + \beta$$

where d_j^{out} denotes the out-degree of jth node.

2.2.5 Authority and Hub centrality

All of the centrality measures introduced so far give high centrality to a node pointed by nodes with high centrality. However, we may also be interested in nodes which point to the most interesting nodes. In other words, we may want to give node a high centrality if it points to other nodes with high centrality. There are networks where such measurements are desirable. The best known example is a citation network where we want to identify review articles. The review may contain a small amount of information itself, but it may point to useful sources of information.

Hence we can define two type of useful nodes in any networks - those which contain useful information themselves and those which point to useful information. The former group is called **authorities** whereas the latter group is called **hubs**. [New10] Note, authorities and hubs only exist in directed networks.

Authority centrality (x_i) of a vertex is proportional to the sum of hub centralities (y_j) of the vertices that point to it: [New10]

$$x_i = \alpha \sum_j A_{ij} y_j$$

Hub centrality of a vertex is proportional to the sum of authority centralities of the vertices that the given vertex point at: [New10]

$$y_i = \beta \sum_j A_{ij} x_j$$

where α and β are normalising constants.

2.2.6 Closeness centrality

Closeness centrality measures how close a given node is to all other nodes in the graph. In other words, it is an average distance from a given node to all other nodes. It measures how "influential" a given node might be. In order to understand that statement, we have to think about information spreading in a graph. The closer a given node is to all other nodes in the graph, the faster an information which passes through that node (or is produced by that node) will reach all other nodes in the graph. Since smaller average distance would actually mean that a node is more influential, we will take the reciprocal of the distance as the closeness measure.

Closeness measure is computed according to the following formula: [New10]

$$C_i = \frac{n}{\sum_j d_{ij}}$$

where d_{ij} is the length of path between nodes i and j and n is the number of nodes in the network

2.2.7 Betweenness centrality

A potentially more useful measure for our network than closeness is called betweenness. Betweenness is another measure which computes how "influential" a node is. Generally speaking, betweenness measures an extent to which a node lies on the path between other nodes. In the literature, it is usually defined in terms of message passing through a network (graph). If messages pass from source to destination using shortest available paths, betweenness would measure the number of shortest paths each node lies on. [New10] Betweenness is independent of other centrality measures – a high value of betweenness does not imply high values of other centrality measures.

As we can see, calculating betweenness for each node in the graph as defined above is extremely demanding computationally and will prove infeasible for larger frameworks. Betweenness could be approximated by a random walk betweenness. In the literature it is defined as performing a random walk from random sources to random destinations a couple of times and counting how many times a given node was visited for each random walks we performed. [New10] Unfortunately, the software package we are using to compute graph measures does not include the random walk betweenness. [Sta]

2.3 Evolutionary learning and genetic algorithms

2.3.1 Introduction to genetic algorithms

Computer Scientists very often look to nature for guiding metaphors. The trend started very early in Computer Science history in times of Alan Turing and John von Neumann. Computer Scientists back then looked for ways of simulating biological evolution. [Mit98] This promising area of research went silent for a while but fortunately resurged at 1980. Today, the area is know as Evolutionary computation and is believed to be applicable to many research problems. Evolutionary algorithms are widely used in areas such as optimisation, machine learning and economics, among many others.

Genetic algorithms form a subclass of evolutionary algorithms. They are effectively a search method which is loosely based on a genetic change of a population of individuals. [Jon88]. Usually, genetic algorithms consist of three fundamental elements:

- A Darwinian notion of 'fitness' of each population member. The 'fitness' governs how likely a certain individual is to influence future generations.
- Selection function which determines which members of the current population will contribute to production of the next population.
- Genetic operators which determine the genetic structure of offspring given the genetic material of their parents.

Genetic algorithms search a space of candidate solutions in order to obtain a desired one. Thus, each member of a population (also called a chromosome) can be thought of as a point in the search space of candidate solutions. [Mit98] The algorithms usually proceed in rounds (also called iterations or generations). In each round they successfully replace one population with another. Thus, a genetic algorithm can be defined as an iterative process [Mit98]:

1. Start with a randomly generated population sampled from the candidate solutions space.
2. Calculate fitness for each member of the population.
3. Repeat the following until the desired number of offspring has been created:
   (a) Select a pair of parent population members from the current population. The probability of selection should be an increasing function of fitness.
   (b) Apply genetic operators to produce offspring from a given parent pair.
4. Replace the current population with a population of offspring generated in the previous step.
5. Go to step 2 or terminate if termination conditions are applicable.

The exact algorithm structure may vary for different approaches. However, as we will see, it is compatible with our implementation used in this project. Genetic algorithms implementations always vary in:

- population member encoding
- how selection is performed
- genetic parameters (cross-over and mutation)

We will discuss the possible approaches to each of the above in the following sections.

2.3.2 Genetic algorithms applicability

Before we discuss possible genetic algorithms implementations, let us briefly consider in what cases genetic algorithms are applicable. As we already said, they are effectively a search methodology and they are often cases where other search methodologies outperform a genetic algorithm. Defining genetic algorithms applicability is very important in our cases as it was part of the project to asses whether this approach would be suitable for the problem and justify the decision.

M. Mitchell [Mit98] defines the following criteria:

- search space is large
- search space is known not to be perfectly smooth and unimodal (i.e. it is not a single smooth 'hill')
- search space is not well understood
- fitness function is noisy (e.g. when fitness function is based on some measurement of the environment, the measurement contains a lot of noise)

- the task does not require a global optimum to be found (a 'sufficiently' good solution is enough)

In each case, if the given criterion is not met, there is a high probability that another, more efficient search strategy exists which would outperform genetic algorithm. If the search space is not large enough, we can always search it exhaustively which guarantees that we will find a global optimum. Genetic algorithm, on the other hand, may end up in local optimum or any other point in the search space even when the space is small. If the search space is smooth and unimodal we can always apply steepest descent algorithms which would also converge to a global optimum in this case. If the search space is well understood, it is very likely that there exist a method using domain-specific heuristics which outperforms any general-purpose method such as genetic algorithm. If the task requires finding a global optimum we face the problem we have already described - a genetic algorithm may converge to some point other than global optimum. Finally, genetic algorithm is resilient to noise as it implicitly accumulates statistics over many generations (in the form of genetic material passed from generation to generation) so it may outperform other search methods when we have noisy data.

2.3.3 Population member encoding

Population member encoding is often the hardest and the most important part of the genetic algorithm. [Mit98] There can be many approaches to encoding a population and the encoding itself may be very problem dependent. However, there are a few standard approaches which are used most often. By far the most popular one is bit string encoding. Each population member is encoded as a string of bits. A single bit or a group of bits specify features of the population members. For example, we could encode ABA derivation strategies using binary encoding. Suppose each strategy consists of the following features:

- Feature 1 specifies which player should move first: proponent or opponent
- Feature 2 specifies which argument should each player choose: youngest or oldest
- Feature 3 specifies whether both players should expand assumptions first or non-assumptions first

We could encode the features in the following way:

- Feature 1: 0 encodes proponent moves first, 1 encodes opponent moves first
- Feature 2: 00 encodes both players choose youngest arguments, 01 - proponent chooses youngest argument, opponent chooses oldest. 10 - proponent chooses oldest, opponent youngest, 11 - both players choose oldest arguments.
- Feature 3: 0 encodes that both players should expand assumptions first, 1 encodes that both players should expand non-assumptions first.

And our population could be:

$$\begin{aligned} \textit{Population member } 1: &\quad 1001 \\ \textit{Population member } 2: &\quad 1101 \\ \textit{Population member } 3: &\quad 0100 \end{aligned}$$

Some authors claim that binary encoding is superior to other types of encoding. However, other authors argue that in many applications binary encoding is unnatural and other types of encoding should be used. [Mit98]. The generalisation of binary encoding is called 'value encoding'. Each population member is a string of any values - e.g. symbols, objects or real numbers.

2.3.4 Selection operators

Having defined the encoding scheme, a genetic algorithm designer then has to define a selection operator. Selection operator specifies how we choose individuals which will be used to create offspring and how many offspring should each pair create. Usually, the selection process is based on fitness - the 'fitter' the individual is, the higher the chances that he will be selected for mating and reproduction. There are several standard selection methods which has been proposed in the literature. Below we list and discuss the most important ones.

Fitness proportionate selection (roulette wheel selection)

In fitness proportionate selection, the probability that a certain member will be selected is computed by:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{2.1}$$

where f_i is the fitness value of ith population member and p_i is the probability of ith member selection.

It is called roulette wheel selection because the procedure can be explained using roulette wheel as an example. Suppose that we have a circular roulette wheel and its area sums to 1. We can now split the wheel such that each individual in the population is assigned a slice of wheel's area. The size of the slice is equal to the probability of selecting that particular member (p_i). The wheel is spun and, after it stops, the individual under the wheel's marker will be selected to become a parent. [Mit98] We repeat the procedure until we have the number of parents we require. Note, a certain individual may become a parent more than once.

Elitism and truncation

Most genetic algorithms are 'generational' - the new generation is constructed entirely from the offspring of the previous generation. There are, however, schemes which preserve some population members. In elitism some specified proportion of the fittest members is always preserved to the next generation. [Mit98] Another selection operator which is based on the same idea is called truncation. All population members are ordered by their fitness. We then select some fraction of the fittest individuals and use that fraction for reproduction.

Tournament selection

Another common selection operator is called tournament selection. There are several definitions of tournament selection in the literature. We will describe the simplest one. We choose two population members at random from the population. We then choose a random number r between 0 and 1. If r is greater than n (where n is some pre-specified parameter, for example 0.1) we choose the fitter of the two selected members and add it to the parents list. If r is lower than n we choose the less fit member. [Mit98]

2.3.5 Genetic operators

Having chosen the selection operator we can now move to specifying genetic operators of our algorithm. Genetic operators control how exactly the offspring population is created from the selected individuals. There are two genetic operator: crossover and mutation.

Crossover

Crossover specifies how population members are combined in order to produce offspring. The simplest crossover is called a single-point crossover. We choose a random position in a population member. We split each parent in the chosen position. We then exchange split parts and combine them to form offspring. For instance, consider two parents:

$$\begin{aligned} Parent\ 1: &\quad 0010 \\ Parent\ 2: &\quad 1101 \end{aligned}$$

Suppose we choose position 2 and split each parent to obtain:

$$\begin{aligned} Parent\ 1: &\quad 00 \ \mid \ 10 \\ Parent\ 2: &\quad 11 \ \mid \ 01 \end{aligned}$$

We can now combine split parts to produce new offspring:

$$\begin{aligned} Offspring\ 1: &\quad 00 \ \mid \ 01 \\ Offspring\ 2: &\quad 11 \ \mid \ 10 \end{aligned}$$

If we are not careful a single-point crossover may significantly decrease the performance. It happens when we choose a splitting point so that it splits functionality related bits. [Mit98] For example, we may have several bits encoding the same feature as we did in our encoding example where two bits where encoding player argument choice. We may prefer to avoid splitting these bits.

There are other variants of crossover operator. A special case of a single-point crossover where we split population members approximately in half is called a **'uniform crossover'**. A 'multi-point crossover' is a direct relative of a single-point crossover where we spit each population member in multiple points. Another type of crossover is a 'positional bias crossover' in which an exchange may happen at each position (each bit or each real number) guarded by some probability p. [Mit98]

2.3.6 Mutation

Mutation is usually used to prevent population from being stuck in the local optimum point. It is usually considered to be of secondary importance as most researchers claim that it is the crossover operator which drives the search forward. [Mit98]. Hence, mutation is usually a random process with a very small probability of occurring. The exact implementations of mutation operator heavily depends on which population encoding we choose. For example, for a binary encoding, a common mutation operator is to change one bit at a random position. For a real numbers encoding, with some probability p we could replace a single number by a random number drawn from some specified distribution. [MD89]

2.3.7 Learning real number parameters using genetic algorithms

Genetic algorithms has been applied surprisingly rarely to search problems where we have to find a combination of parameters which improves performance of some model. Real numbers are often the most intuitive representations of such parameters. Each population member is of the form

$$[parameter1, parameter2, parameter3...].$$

However there are very few examples where they were actually encoded using real numbers. It may be an influence of John Holland, who (in his most famous work) claims that binary encoding is usually superior to real-number encoding. [Hol75] [Jon88] [Mit98]

However, when there are records of applying real number encoding to parameters learning, these are usually successful stories. There are experimental and theoretical evidence stating that genetic algorithms can learn efficient model parameters encoded as real numbers very quickly. Typically, even for a very large search spaces (such as 10^{30}) acceptable results are found after only 10 generations. [Jon88] One of the better known applications of genetic algorithms to real number parameters learning was the work of David J. Montana and Lawrence Davis who tried to learn efficient neural network parameters. [MD89]

Since mutation is heavily encoding-dependent, it was worth investigate the best practises applied to mutate a population of real numbers. Researchers have investigated several approaches: [MD89] [Mit98] [SK93]

1. With probability p replace parameter value in the population member with a random values chosen from the initialisation probability distribution (i.e. the one we use to initialise our first population).
2. With probability p add a random value chosen from the initialisation probability distribution to population member's parameter value.
3. With probability p replace parameter value with one of the 10 most frequent values for that parameter observed in all population members.

Chapter 3

Finding optimal strategies - description of the approach

3.1 Problem statement

As we already mentioned the main topic of our thesis is analysing Assumption-Based Argument derivation problem. The issue is simple – the derivation is slow. Querying a random framework with around 50 sentences (assumptions/non-assumptions) using a naive strategy usually will not terminate. Synthetically generating frameworks with 50 sentences and queries which do terminate is a very challenging task in itself and a whole master thesis could be written on that issue.

The problem is that frameworks constructed from real data are much larger than 50 sentences. Also, the requirement is to perform ad-hoc queries on those frameworks which do terminate within seconds. Consider again a medical application of ABA. A doctor querying for patient's health will not want to wait hours to get the results back. In fact, 20 seconds derivation time is a reasonable upper-bound.

3.1.1 Possible solutions

There are two solutions to the derivation time problem. The first one uses parallelization. The idea is to run the same query on several machines with each parallel run using different derivation strategy. We could then measure each run progress and kill the one which we think are not progressing very well. This approach forms a separate master thesis and will not be discussed further here. As a side note, it is much harder to parallelize the derivation process itself, because we would have to somehow synchronise the defences and culprits set which are constantly changing during the derivation.

The second idea is to find an efficient derivation strategy. This is not entirely independent of the first idea, as we can use any efficient strategies found in parallel runs. Before discussing this idea any further, we have to define what we mean by derivation 'strategy'.

3.1.2 Derivation strategy

ABA dispute derivation progresses in discrete steps. At each step, one of the players expands one of its arguments by a single rule / contrary application. Each player may have a different strategy deciding which argument to expand and how to expand it. Each of them may have several options to consider. Player's preferences together with deciding which player should play at a given round form what we will call a 'derivation strategy' of the game.

More formally, recall that there are 6 choice points in the algorithm:

- turn choice: decides which of the two players makes a move in this turn
- proponent argument choice: decides which argument the proponent will expand at this step
- proponent sentence choice: decides which sentence the proponent will expand at this step
- proponent rule choice: decides which rule proponent will use to expand a sentence (non-assumption)
- opponent argument choice: decides which argument the opponent will expand at this steps
- opponent sentence choice: decides which sentence the opponent will expand at this step

There is no opponent rule choice as the opponent will use all rules to expand his non-assumptions. Derivation strategy determines which choices we make at each of these 6 points. To give an example, a valid derivation strategy for all 6 points could be:

- choose proponent to move first
- proponent chooses his oldest argument to expand (i.e. the one which hasn't been expand for the longest time)
- proponent chooses random sentence
- proponent chooses random rule to expand non-assumptions
- opponent chooses his oldest argument
- opponent chooses the first sentence alphabetically (e.g. from [a,b,c] he would choose a)

Different strategies will result in different derivations but will lead to the same solutions. The key difference between derivation strategies is their termination speed. We will show that it is possible to construct the derivation strategy which significantly improves the performance over the naive strategies like the one presented above.

Unless we state otherwise, we also make two key assumptions in all our discussions regarding derivation:

1. **We are always looking for all solutions**. This is crucial especially when we consider backtracking. If we were to look for just a single solution, backtracking would only occur when we fail in order to try another possibilities. However, since we are looking for **all** solutions, we will also backtrack when we succeed in order to find other derivation branches which lead to a solution.
2. **We only consider admissible derivation**. Before running our experiments we always explicitly set the derivation type to admissible. Thus we never run experiments with a grounded derivation semantics.

3.1.3 Branching and backtracking

In order to derive an efficient strategy, it is crucial to understand what really influences the structured X-dispute derivation speed. As we will learn in the experiments, there are two main factors affecting derivation time – branching and backtracking. Branching measures how many different branches of derivation we have to consider. For instance, when the proponent expands a non-assumption which is a head of three rules, he will get three possible branches of the derivation. The derivation can take any of the three branches, but regardless of whether that branch succeeds (leads to a valid solution) or fails (leads to no solution), the derivation will backtrack to the choice point and explore another branch by picking a different rule. Backtracking on success happens as well because we are searching for all solutions.

Branching is crucial to derivation time. For instance, consider the following ABA framework:

```
myRule(p0, [x0, y0, z0, t0, w0]).
myRule(y0, [a, b, c, d, e, f]).
myRule(z0, [g, h, i, j, k, l]).
myRule(t0, [m, n, o, p, q, r]).
myRule(w0, [s, t, u, v, w]).

....

myRule(x0, [x]).
myRule(x0, [y, z]).

....

contrary(w, c0).
myRule(c0, []).
```

where p0, x0, y0, z0, t0, w0 are all non-assumptions and the rest of sentences are all assumptions. Suppose that the player choice strategy is set to 'proponent moves as long as he has any valid moves'. Suppose the proponent is proving p0 (the first rule). He could expand y0, z0, t0 and w0 first - before expanding x0. Lets suppose he does that and then chooses to expand x0 using the first rule. If we are lucky, the opponent then could very quickly 'defeat' the proponent's argument by using contra-argument [ ] :- c0 (an argument is represented by 'unmarked set : - conclusion'). The derivation would fail and the proponent would have to choose the second rule for x0. The derivation would fail immediately as well, as the opponent would use the same argument [ ] :- c0.

Now consider another strategy in which the proponent chooses x0 first and then y0, z0, t0 and w0. Opponent then moves and does the same thing as before - disproves proponent's claim by using [ ] :- c0. However, now the proponent backtracks to x0 rule choice and he chooses the second rule. The problem is now we have to repeat the derivation of y0, z0, t0 and w0 leading to twice as much work as before.

Now suppose that we have yet another derivation strategy in which the proponent first chooses w0 and then chooses w. The opponent immediately seizes the opportunity and disproves p0 by using [ ] :- c0. This is the ideal, 'least amount of work strategy' where we avoid any branching and backtracking at all.

We can draw three important conclusions from the above example. First of all, we should avoid branching if possible. Secondly, we should do all non-branching work before branching as otherwise we will end up doing the same work twice. Finally, if we have a chance to kill derivation (e.g. [ ] :- c0) we should use it immediately.

The key question here is how do we measure the amount of branching a given choice leads to and the amount of work we will have to do when we choose it. Investigating various measures available at each choice point and their relationship with derivation time forms the better part of this report. Our initial idea was to use ABA graph measures to guide the derivation. This approach proved to be partially successful. We will present all measures, experiments and results in the next chapter.

3.2 Representing derivation strategies

Before we can discuss our approach to learning optimal (with respect to time) derivation strategies, we have to explain how we represent them. The representation is crucial as it needs to be generic enough to allow for optimality learning.

3.2.1 Measures

We start with a general discussion of 'measures'. Each derivation strategy is, in fact, a set of six different strategies – each associated with one of the six choice points in the algorithm. At each choice point we have a number of different options to choose from. For example, at proponent sentence choice point the proponent is choosing between one or more assumptions and non-assumptions. At this derivation step he will further expand only the premise he chooses. His choice may influence the derivation time as we have shown in the previous section.

We must differentiate his options and pick the one which we think is the most optimal. In order to do that we define a set of measures associated with each option. We will always use the same set of measures for all choices in a single choice point in one experiment. For example, if we are choosing between assumptions, we will always use the same set of measures for each assumption. But we will use (or better 'try out') different sets / combinations of measures in different experiments. Measures are a way to quantify derivation strategies. They come from various sources – they may be precomputed before the derivation starts (e.g. ABA Graph measures) or they may be measured during the derivation (e.g. the number of assumptions which are in defences or culprits). **Each measure we use has only non-negative values**.

To illustrate, we will show an example. Suppose we have chosen opponent to make his move. Suppose that the opponent has the following three arguments to choose from:

```

arg1: [t0, w0, x0, y0, c] :- p0

arg2: [a, b] :- p0

arg3: [x0] :- q0

non-assumption: [t0, w0, x0, y0, p0, q0]
assumptions: [a, b, c]
```

where each argument is denoted by: unmarked set :- root.

For each of the four premises we could define the following measures:

- the unmarked set size
- number of assumptions in unmarked set
- sum of PageRank centrality of all premises in the unmarked set measured from the ABA graph
- sum of degree-out of all premises in the unmarked set measured from the ABA graph

The first two measures are computed during the derivation. The last two are precomputed before the derivation begins. Suppose that the following information was precomputed from the ABA graph:

Node	PageRank	Degree-out
t0	0.002	1
w0	0.021	4
x0	0.008	2
y0	0.054	6
p0	0.021	4
q0	0.700	14
a	0.011	1
b	0.049	1
c	0.000	0

Table 3.1: Precomputed ABA measures

We could now compute the measures for each of the arguments by summing measures of its unmarked set members:

	Arg1	Arg2	Arg3
Unmarked Set Size:	5	2	1
Number of Assumptions:	1	2	0
ABA Graph PageRank:	0.806	0.060	0.008
ABA Graph Degree-out:	21	2	2

Table 3.2: Measures computed for each of the arguments

We could now use each measure separately to define a derivation strategy. For instance, we could say that the opponent should always choose an argument with smallest unmarked set size. In this case, the opponent will choose argument 3. Alternatively, we could say that the opponent should choose an argument with largest PageRank centrality which would lead to choosing argument 1.

We can also define a derivation strategy which combines two or more measures. For instance, consider the following: 'choose argument with smallest degree-out and in case of a tie choose argument with smallest unmarked set size'. Such strategy would result in choosing argument 3.

Our goal, however, was to learn these or similar strategies, not to guess them. We also wanted to learn which measures are the most important relative to others and what is the correlation between a given measure and derivation time. For example, we were interested in knowing whether we should make choices which have certain measures as small as possible, or whether it is a good idea to make choices with measures having largest value.

Thus, we needed something general which would be suitable for applying a machine learning technique and which would also allow us to spot trends in correlation between measures and performance. To fulfil these requirements we developed the most generic derivation strategy. For each choice point in the program we define a **decision function** which computes a weighted sum of all available measures (**decision value**).

3.2.2 Model

Suppose that $M = [M_1, M_2, M_3 ... M_n]$ denotes a set of n available measures for a given option. We also define a set of n weights: $W = [W_1, W_2, W_3 ... W_n]$ each weight associated with one measure. The decision function takes the form:

$$decision\_naive(M) = M_1 * W_1 + M_2 * W_2 + M_3 * W_3 + ... + M_n * W_n \tag{3.1}$$

The decision rule is to make choice with the largest $decision\_naive(M)$ value. We are now able to encode different strategies by choosing different weight vectors W. To see this lets go through an example. Consider our previous example of opponent argument choice and measures from 3.2:

Unmarked Set Size	w1
Number of Assumptions	w2
ABA Graph PageRank	w3
ABA Graph Degree-out	w4

Suppose we set the weight vector to the following values:

Unmarked Set Size (W1)	Number of Assumptions (W2)	PageRank (W3)	Degree-out (W4)
0	0	0	1

Using equation 3.1 we can now compute each argument's decision value:

$$decision\_naive(M_{arg_n}) = M_1^n * W_1 + M_2^n * W_2 + M_3^n * W_3 + M_4^n * W_4$$

Where M_i^n denotes the ith measure of nth argument. For our three arguments we have:

$$\begin{aligned}
Argument_1 : decision\_naive(M_{arg_1}) &= 5 * 0 + 1 * 0 + 0.806 * 0 + 21 * 1 = 21 \\
Argument_2 : decision\_naive(M_{arg_2}) &= 2 * 0 + 2 * 0 + 0.060 * 0 + 2 * 1 = 2 \\
Argument_3 : decision\_naive(M_{arg_3}) &= 1 * 0 + 0 * 0 + 0.008 * 0 + 2 * 1 = 2
\end{aligned}$$

The opponent would choose argument 1. The weight vector $[0, 0, 0, 1]$ encodes a strategy which always chooses an argument with the largest degree-out measure. Suppose we pick a different weight vector:

Unmarked Set Size (W1)	Number of Assumptions (W2)	PageRank (W3)	Degree-out (W4)
-1	0	0	0

And compute each argument's decision value again:

$$\begin{aligned}
Argument_1 : decision\_naive(M_{arg_1}) &= 5 * -1 + 1 * 0 + 0.806 * 0 + 21 * 0 = -5 \\
Argument_2 : decision\_naive(M_{arg_2}) &= 2 * -1 + 2 * 0 + 0.060 * 0 + 2 * 0 = -2 \\
Argument_3 : decision\_naive(M_{arg_3}) &= 1 * -1 + 0 * 0 + 0.008 * 0 + 2 * 0 = -1
\end{aligned}$$

The opponent now chooses argument 3. The weight vector $[-1, 0, 0, 0]$ encodes a strategy which chooses an argument with the smallest unmarked set size measure.

As we see, we have a way of setting a weight vector so that we get a strategy which prefers measures to be as small as possible or as big as possible. In the former case, we have to set the corresponding weight to -1. In the later case, we have to set the weight to $+1$. In order to make the maths sound, **we always require that each measure has only non-negative values**.

Finally, suppose we set the weights in the following way:

Unmarked Set Size (W1)	Number of Assumptions (W2)	PageRank (W3)	Degree-out (W4)
0.5	0	0	-1

And compute each argument's decision value yet again:

$$\begin{aligned}
Argument_1 : decision\_naive(M_{arg_1}) &= 5 * 0.5 + 1 * 0 + 0.806 * 0 + 21 * -1 = -18.5 \\
Argument_2 : decision\_naive(M_{arg_2}) &= 2 * 0.5 + 2 * 0 + 0.060 * 0 + 2 * -1 = -1 \\
Argument_3 : decision\_naive(M_{arg_3}) &= 1 * 0.5 + 0 * 0 + 0.008 * 0 + 2 * -1 = -1.5
\end{aligned}$$

We again choose the argument with the largest decision value - argument 2. Note, our strategy now corresponds to choosing arguments with smallest degree-out and largest unmarked set size. The absolute magnitude of each weight denotes how we prioritise each measure. Since -1 is larger than 0.5 in absolute terms, we will give priority to the measure associated with -1. Hence, our strategy here really is: choose arguments with smallest degree-out and if more than one arguments have similar degree-out value, choose the one with largest unmarked set size.

3.2.3 Making model sound - normalisation and avoiding 0 measure

So far we have been calling our decision value 'naive'. This is because there are two important issues with the decision value as it is now. We will consider each in turn.

First, consider the following weights:

Unmarked Set Size (W1)	Number of Assumptions (W2)	PageRank (W3)	Degree-out (W4)
0	1	1	0

The corresponding strategy would be: choose arguments with the biggest number of assumptions and the biggest PageRank. We now compute the decision values associated with each argument:

$$\begin{aligned} Argument_1 : decision\_naive(M_{arg_1}) &= 5*0+1*1+0.806*1+21*0 = 1.806 \\ Argument_2 : decision\_naive(M_{arg_2}) &= 2*0+2*1+0.060*1+2*0 = 2.060 \\ Argument_3 : decision\_naive(M_{arg_3}) &= 1*0+0*1+0.008*1+2*0 = 0.008 \end{aligned}$$

And so the opponent chooses argument 2 once more. However, this is probably not what we would expect. Although argument 2 has 2 times bigger degree-out value than argument 1, its PageRank value is 13 times smaller than argument's 1 PageRank. Since we prioritise two measures equally, clearly we should be choosing argument 1 instead of argument 2. The problem is that each measure has a slightly different range. In order to make them statistically equally significant, we can normalise them. We will be using the following normalisation strategy:

$$decision\_value(M) = \frac{M_1}{Sum(M_1)} * W_1 + \frac{M_2}{Sum(M_2)} * W_2 + \frac{M_3}{Sum(M_3)} * W_3 + ... + \frac{M_n}{Sum(M_n)} * W_n \quad (3.2)$$

where $Sum(M_i)$ is defined for k available arguments as follows:

$$Sum(M_i) = \sum_{j=1}^{k} M_i^{arg_j}$$

I.e. we sum ith measure values for all arguments.

If we normalise all measures we will get the following table:

	Arg1	Arg2	Arg3
Normalised Unmarked Set Size	0.625	0.25	0.125
Normalised Number of Assumptions	0.34	0.66	0
Normalised ABA Graph PageRank	0.922	0.069	0.009
Normalised ABA Graph Degree-out	0.84	0.08	0.08

Table 3.3: Normalised measures computed for each of the arguments

Using 3.3 we can now recompute the decision values:

$$\begin{aligned}
Argument_1 : decision\_vaule(M_{arg_1}) &= 0.625 * 0 + 0.34 * 1 + 0.922 * 1 + 0.84 * 0 = 1.262 \\
Argument_2 : decision\_value(M_{arg_2}) &= 0.25 * 0 + 0.66 * 1 + 0.069 * 1 + 0.08 * 0 = 0.729 \\
Argument_3 : decision\_value(M_{arg_3}) &= 0.125 * 0 + 0 * 1 + 0.009 * 1 + 0.08 * 0 = 0.009
\end{aligned}$$

And we now choose argument 1 as required.

Another issue occurs when a certain measure has value 0. Consider the following measures:

	Arg1	Arg2	Arg3
Normalised Number of Assumptions	0	0	0

Now consider applying $W = -1$ and then $W = +1$ to the Normalised Number of Assumptions (M) measure. We will get 0 for each of the arguments in both cases. Regardless of what we choose W to be $W * M = 0$. This is inconsistent with what we have said - weight -1 corresponds making choices with measure as small as possible, while weight +1 corresponds to making choices with measures as big as possible. This is not a problem if we know the set of weights in advance and use them in derivation. However, our aim is to learn the optimal strategy (the optimal set of weights, optimal with respect to derivation time). Hence we would prefer to avoid unpredictable weight values as it may affect the learning process.

To solve the issue we seed each measure with a very small number instead of 0. We picked 0.00001 for the job. We carefully inspected each of the measures we introduced in the experiments to double check whether adding 0.00001 does influence the decision value. Most measures have values greater than 1. None of the measures have values in ranges less than 0.00001.

3.3 Learning optimal strategies

The single most important thing about our approach is: **we did not want to guess derivation strategies which just might work**. Instead, we wanted to learn those strategies by querying a big number of synthetically generated frameworks and observing the derivation performance. In this section we will describe the machine learning approach used for the task.

3.3.1 What do we want to learn

So far we have shown how we can encode different strategies. For example, a strategy which uses 3 measures per each choice point can be encoded as a vector of weights W with 18 weights (6 choice points each using 3 measures gives us $3 * 6 = 18$). Note, it is possible to use the same weight for different choice points. Also, it is possible to use different measures for different choice points so that the number of weights per choice point varies.

However, in any case, what we end up with is strategy encoded in a form of vector $W = [w1, w2, ...wn]$. Our aim is to learn the best performing strategies. Why do so? Recall that weight +1 indicated that we want to keep the associated measure as big as possible, while weight -1 indicated that we want to keep the associated measure as small as possible. We can generalise the statement to the following:

Positive weight value indicates that, when choosing between two alternatives (between arguments, sentences etc), the strategy would be to favour the alternative with bigger associated measure.

Negative weight value indicates that, when choosing between two alternatives (between arguments, sentences etc), the strategy would be to favour the alternative with smaller associated measure.

What we really want to learn here is not a specific combination of weights in a single strategy which just happen to make that particular strategy superior in terms of performance. Looking at such strategy as standalone may be useful but may also be misleading. Suppose, for instance, that a certain measure is completely irrelevant to the derivation time. But in the single best performing strategy that measure's weight is positive. We may be tricked into thinking that there is a sound heuristic which prefers bigger values of the measure to smaller values.

Instead of looking at a one specific best performing strategy and drawing conclusions from it, we will be looking at a larger group of well performing strategies. We will be trying to analyse that group, contrast it with a group of bad strategies and search for any trends and patterns.

For instance, lets consider a group of 1000 strategies (1000 weight vectors $[w_1, w_2, w_3...w_n]$). Suppose that w_1 corresponds to a PageRank value in the proponent sentence choice. Lets pick 25% of best performing strategies from that group. Suppose that we find out that 90% of strategies in that group have w_1 positive. We then may suspect that picking sentences with big PageRank measure in proponent sentence choice is a common pattern for best performing strategies and we can propose it as a valid heuristic.

Hence, the three most important rules we will follow when analysing the output are:

Positive weight values in the majority of the best-performing strategies indicates that the heuristic to always prefer alternatives with the corresponding measure high may improve derivation performance.

Negative weight values in the majority of the best-performing strategies indicates that the heuristic to always prefer alternatives with the corresponding measure low may improve derivation performance.

If we cannot find a correlation between derivation time and a certain measure and the same result repeats in a number of experiments, we may conclude that the measure is not relevant and a good derivation heuristic based on that measure cannot be built.

We will apply the above rules when analysing the output of experiments in next chapter.

3.3.2 How do we learn - genetic algorithm overview

Since we want to analyse the best performing strategies, we need to find those strategies first. We can think about the problem in terms of search. Suppose we fix the choice points we are considering and we fix measures used for each of the choice point. Further suppose that we encode each strategy as a vector $[w_1, w_2, w_3...w_n]$. Our search space consists of all possible such weight vectors.

We can now use a number of search approaches in order to obtain the best performing strategies. For that we decided to use genetic algorithm. As discussed in the Background chapter, a problem has to satisfy specific criteria in order to benefit from using genetic algorithm as a search strategy. Our problem meets most of the criteria:

- The search space is large. If our most complex models will have around 30 parameters.
- The search space is not smooth or unimodal. A small change in one parameter value may rapidly change the performance of a population member.
- The search space is not well understood. This is actually one of our assumptions - we do not make any assumption about the search space. We use genetic algorithm specifically in order to understand the search space better.
- The task does not require a global optimum.
- Its quite straightforward to describe our problem in terms of genetic algorithm.

The idea behind applying a genetic algorithm to the problem is summarised by the following two concepts:

- each strategy encoded as a weight vector will form a single population member
- fitness will be computed by measuring derivation time of a given strategy

To emphasise, lets explicitly define the key genetic concepts in terms of our problem terminology. From now on we will be using those terms when we talk about the learning process.

Definition 7 ***Population member*** - *a single strategy represented by a weight vector. Each population member will be of the form* $[w_1, w_2, w_3...w_n]$.

Definition 8 ***Fitness*** - *average derivation time of a population member measured on a set of ABA derivation frameworks. Thus, lower values of fitness correspond to better performing population members.*

This is crucial to understand the rest of the report. We will be using terms 'population member' and 'strategy' interchangeably. We will also be using terms 'fitness' and 'average derivation time' interchangeably.

Another view of our usage of genetic algorithm here is in terms of sampling. We are more interested in getting a large group of well performing members than a single best performing member. Thus a genetic algorithm may serve as a more elaborate way of sampling the search space to obtain well performing strategies. We then want to analyse those well performing strategies to learn something about the search space itself (i.e. about the derivation algorithm):

- what are the characteristics of strategies which outperform other strategies
- how can we use these characteristics to create useful heuristics

3.3.3 Genetic parameters

Having the basics in place, we are now ready to define other parameters of the learning algorithm.

Initialisation and fitness measurement

We begin by picking the initial population. Before the learning takes place, we have to agree on choice points we consider and what measures we use for each choice point. Each population member is a vector of n real numbers: $[w_1, w_2, w_3...w_n]$. We choose the initial members randomly. Since each w_i is a weight, we initialise it to a random value between -1 and $+1$. To ensure a satisfactory exploration of the search space, we need to select initial population size to be 'big enough'. We usually pick it to be between 256 and 512 in order to make the learning process feasible computationally.

Having generated the initial population, we begin the learning experiment by measuring fitness of each population. To do that we need to supply a set of training (framework, query) pairs. In most of our experiments we used 100 (framework, query) pairs, each generated so that there exists at least one default strategy for which the query terminates within 20 seconds (i.e. the frameworks are solvable).

To compute the fitness for each population member we measure member's derivation time for each (framework, query) pair and compute the average.

Selection operator

We then proceed to the second stage of genetic learning which is selection. We want to select the best performing members of population for mating. We usually select 1/2 of all population members. We will be using a variant of Fitness Proportionate Selection method. The selection probability is inversely proportional to member's fitness - the lower the average derivation time of a certain member, the 'fitter' the member and the higher the probability of his selection.

To compute the probability of choosing a certain member we first inverse its fitness. We then normalise that inverse by dividing it by the sum of inverses of all population members. As a result, we will get a valid probability - inversely proportional to member's fitness and summing to 1 over all selected members. This is summarised by the following equations:

Let $fitness_i$ be a fitness value of ith member, n be the number of population members.

$$\begin{aligned} inv\_fitness_i &= 1/fitness_i \\ probability_i &= \frac{inv\_fitness_i}{\sum_j^n inv\_fitness_j} * 100(\%) \end{aligned}$$

We then select a fraction (usually 1/2) of population for mating. We make selections with repetitions so there is a probability of choosing the same member more than once.

Crossover

Crossover is performed by combining selected individuals into pairs (parents) and mixing each pair to produce two children. The pairs are chosen randomly from selected individuals such that each selected individual is paired once. We perform a single-point uniform crossover - we always define the split point to be the centre point of each population member. Thus we split each parent in half and combine the parts to produce two children:

$$\begin{aligned} parent1 :&[x_1,\ x_2\ ...\ x_{\frac{n}{2}},\ x_{(\frac{n}{2}+1)}\ ...\ x_n] \\ parent2 :&[y_1,\ y_2\ ...\ y_{\frac{n}{2}},\ y_{(\frac{n}{2}+1)}\ ...\ y_n] \end{aligned}$$

And the offspring produced is:

$$\begin{aligned} child1 :&[x_1,\ x_2\ ...\ x_{\frac{n}{2}},\ y_{(\frac{n}{2}+1)}\ ...\ y_n] \\ child2 :&[y_1,\ y_2\ ...\ y_{\frac{n}{2}},\ x_{(\frac{n}{2}+1)}\ ...\ x_n] \end{aligned}$$

It is safe in our case to use a single-point uniform crossover as we do not have any large groups of common-functionality parameters which we could destroy by doing so.

Mutation

Mutation is the final step of each iteration of learning. There are various approached to mutating real numbers some of which we covered in the Background section. We chose approach which was also proposed by Schulze and Kremer in their protein structure experiments. [SK93] If a certain weight is selected to be mutated, we will be incrementing or decrementing that weight by a very small value (lets call it 'mutation value'). Whether we increment or decrement is a random decision, each has a probability of 0.5 to occur. To compute mutation value we use a standard deviation of the corresponding parameter computed from all population members. Suppose that a weight we are mutating has index 0 (i.e. it is the very first weight in the vector encoding a population member). To compute mutation value for member's w0 we consider all other population members and their w0 value. We take all those w0 values (all w0 values recorded for the population) and we compute their standard deviation which will be our mutation value.

Concrete example

Let us consider a toy example and perform one genetic algorithm iteration. Consider the following learning parameters:

- Each strategy consists of four weights: $[w_1, w_2, w_3, w_4]$
- Number of training frameworks: 4
- Selection rate: 0.5
- Mutation rate: 0.1

Further suppose we have the following input population (it may have come from previous iteration or it may have been generated randomly if it is the first iteration):

Member Index	w1	w2	w3	w4
1	1	0.5	-0.9	0.0
2	-1	0.0	-0.5	-1
3	0.5	0.1	-0.1	1
4	0.0	1	1	0.5

Table 3.4: Population weights

The first step is to compute average derivation time (fitness) in seconds for each member:

Index	Framework 1	Framework 2	Framework 3	Framework 4	Avg Deriv Time (Fitness)
1	0.5	0.5	1.5	2.5	1.25
2	0.1	0.4	0.2	0.3	0.25
3	0.6	0.9	0.4	0.1	0.50
4	0.7	0.3	2.5	3.5	1.75

We now compute the probability of selecting each population member:

Index	Fitness	Inverse fitness	Probability of selection
1	1.25	0.89	11%
2	0.25	4	54%
3	0.50	2	27%
4	1.75	0.57	8%

Since our selection rate is 0.5, we will be selecting two members. Suppose we select population member 2 and population member 3. We pair the two members and produce children:

Member Index	w1	w2	w3	w4
parent1	-1	0.0	-0.5	-1
parent2	0.5	0.1	-0.1	1

Member Index	w1	w2	w3	w4
child1	-1	0.0	-0.1	1
child2	0.5	0.1	-0.5	-1

Now we perform mutation of each child. Suppose that the mutation rate is 5%. We consider each weight (in each child) in turn. For each weight, we have 5% chances of mutating that weight. Suppose that for child1 we did not choose any weight for mutation and for child2 we selected w2. If we extract all w2 values from Table 3.4, we obtain [-0.9, -0.5, -0.1, 1.0]. We can now compute the standard deviation of that vector, which is: 0.708. Now we choose whether we want to add or subtract the computed value. Suppose we choose to add. Hence for child2 we obtain: $w2 = -0.5 + 0.708 = 0.208$ and our final output population is:

Member Index	w1	w2	w3	w4
member1 (child1)	-1	0.0	-0.1	1
member2 (child2)	0.5	0.1	-0.208	-1

Chapter 4

Experimental Infrastructure

4.1 Overview

Although research and experimentation was the core of the project, it would have been impossible to carry out any useful experiments without proper supporting infrastructure in place. Hence, in this chapter we will try to explain all parts of the experimental infrastructure. We will cover every design choice we have made as well as listing technical details and parameters. Hopefully, after studying this chapter, the reader will have more confidence in our approach and its soundness and correctness. Although we did not conduct any formal proof of mentioned properties, we made every possible software engineering effort to ensure that they hold. We especially wanted to ensure that if any output of learning seems incorrect or inconsistent it is not due to faulty software.

The second reason why we include this chapter in the main report is to provide a short documentation of the set-up. This can come in handy in case someone else will be continuing the project and reusing the same software. However, this chapter will be a very high-level overview of the infrastructure, not a detailed documentation down to the code-level.

The code-base is organised into 5 major components:

- Frameworks - component responsible for frameworks generation and enrichment, also contains frameworks database.
- Derivation - code responsible for ABA derivation.
- Learning - code responsible for genetic learning of optimal strategies.
- Benchmarking - component responsible for comparing different strategies and models.
- Output analysing - code responsible for reading the output of the learning phase and transforming into a more human-friendly form (plots or csv files)

The codebase is a mix of Sicstus Prolog [ICS], C++, Scala and bash scripts.

4.2 Frameworks

4.2.1 Frameworks generation

The generation scripts were not created solely for the purpose of this project. They were provided when the project started and were not written by the author of this report. The author added a few gluing bash scripts to automate the generation process and to validate its output. The core generation generation logic was written in Prolog.

When we generate a framework we have to specify the number of sentences (the number of assumptions and non-assumptions) we would like the framework to have. This number determines to which group the framework will qualify. We use four values, each one corresponding to one of the four groups:

- tiny frameworks: 30 sentences
- small frameworks: 55 sentences
- medium frameworks: 80 sentences
- large frameworks: 150 sentences

We never use tiny frameworks so for the rest of the report we will focus on small, medium and large frameworks. Only small frameworks are used in learning. All three types of frameworks are used in benchmarking.

We require all frameworks we generate to be solvable. I.e. a framework has to have a query (a sentence) and a strategy which solves the query within some reasonable time bound. This is required for the learning process. If we have a framework and a query on that framework which currently takes 15 seconds to solve with default strategy, we know that it is feasible to solve the framework and we have a room for improvement for our learning process to obtain better strategies. We can also benchmark our strategy against the default strategy. Thus we have to specify time upper-bound (in seconds) in which we want the default strategy to solve the query. If for a certain framework we cannot find a query and a default strategy which terminates before the time upper-bound is reached, we discard the framework. Depending on the size of the framework we use different default derivation time upper-bounds:

- small frameworks: 20 seconds
- medium frameworks: 60 seconds
- large frameworks: 120 seconds

Hence, when frameworks are generated they also have two parameters already:

- a query which terminated within the time upper-bound, also called **goal**
- a strategy which was set to solve that query, also called **default strategy**

One way to think about framework generations is in terms of (framework, goal) pairs. In fact, when we specify the number of frameworks to be generated by the script, in reality we specify the number of unique (framework, goal) pairs, as a certain framework may have a couple of interesting queries. The same is true for learning or benchmarking - when we specify the number of training or testing frameworks, in reality we specify the number of unique (framework, goal) pairs.

We also specify time lower-bound to the generation script. This is to ensure that the framework is not trivial - the default strategy has to take more than time lower-bound seconds to solve the goal. We usually set the lower-bound to 2 seconds.

The output of framework generation is a Prolog file with the framework itself (assumptions, non-assumptions and rules) and meta-data which consists of goal and strategy (expressed as a 'goal(G)' and 'strategy(S)' predicates).

4.2.2 Framework pre-processing

As all of our models use parameters based on ABA Graph measures, we have to generate that graph and compute the relevant measures before we can use the framework in learning or benchmarking. Some of the models also use another type of graph - the Assumption Graph which we will cover in the next chapter. Also, some of our models use what we call a 'Lookahead Subgraph' defined for each ABA Graph node. It is a subgraph of ABA Graph which consists of nodes located within a certain distance from the node of interest. It will be carefully explained in the next chapter (with pictures). Right now the important bit is that computing various measures of Lookahead Subgraph is also part of framework pre-processing and is performed here. Generally, any framework pre-processing and framework enrichment is performed in this dedicated component. The code is a mix of Prolog and C++ with bash scripts for automation and gluing.

The only input required is an ABA framework itself. The framework is represented as a Prolog file. All meta-data created by the pre-processing component is also represented as a Prolog predicate and written back to the ABA framework file. We used the following Prolog predicates to describe the meta-data:

```
% ABA Graph predicates:
aba_assumption_node(NId,Assumption).
aba_rule_node(NId,Head,Body).
aba_non_assumption_node(NId,Nonassumption).
aba_attack_edge(EdgeId,From,To).
aba_proof_edge(EdgeId,From,To).
aba_support_edge(EdgeId,From,To).

% Assumption Graph predicates:
ass_assumption_node(NId,Assumption).
ass_attack_edge(EdgeId,From,To).

% Graph measures
degree_centrality(NId,V).        % V = Value
closeness_centrality(NId,V).
betweenness_centrality(NId,V).
eigen_centrality(NId,V).
pagerank_centrality(NId,V).
hub_centrality(NId,V).
authority_centrality(NId,V).

% Lookahead measures
na_ass_set(AssId, Set).         % Assumptions in the lookahead subgraph
na_tree_size(AssId, Size).      % Size of the lookahead subgraph
na_attacks(AssId, Attacks).     % Number of attacks directed at subgraph
rule_ass_set(AssId, Set).       % Assumptions in the lookahead subgraph
rule_tree_size(AssId, Size).    % Size of the lookahead subgraph
rule_attacks(AssId, Attacks).   % Number of attacks directed at subgraph
```

The Prolog program is responsible for reading input ABA framework, generating ABA Graph and (if required) Assumption Graph and computing all Lookahead measures. C++ component is responsible for reading in the graph and computing all graph measures (degree, closeness, betweenness, pagerank, eigen, hub and authority centralities). The computation was outsourced to C++ network library called SNAP (Stanford Network Analysis Platform) [Sta] as there are no good network frameworks for Sicstus Prolog currently available.

4.3 Derivation

Once the framework has been pre-processed, it can be supplied to the derivation algorithm and queried. The derivation algorithm used for this work is called **proxdd** and is written in Prolog. The core of the algorithm was already supplied to the author of the report. For the purpose of this work, the algorithm was extended with additional derivation models which we describe in the next chapter. However, the core part of the algorithm was unchanged.

Each time a new model was developed it had to be tested for correctness. For that part we had a set of 18 test frameworks. The frameworks were carefully selected to cover the main parts of the algorithm (argument selection, rule selection, sentence selection, filtering and backtracking) as well as possible edge-cases: no solution for the input query, many (10+) solutions to the input query. We then compared the output of the new model with the output produced by one of the default strategies. We compared the solutions produced by both implementations. The test passed if the solution matched.

4.4 Learning

The learning code is the core of the project. As we described in the previous chapter, we used genetic algorithm for the learning purpose. It was developed in Prolog, with a few bash scripts used for automation and control (as usual). In order to run the learning process, the user has to set up two configuration files which specify:

- Directories of all required source files.
- Directory of output/log files.
- Frameworks used for training: framework directory, the number of frameworks we want to use from that directory and derivation timeout.
- The name of the model we are testing.
- All genetic algorithm parameters: mutation rate, selection rate and initial population size.

We initially used 50 small (framework,query) pairs for training. Then we decided to expand that number to 100. The usual genetic algorithm parameters were:

- initial population size: 256 (for models with less than 20 parameters), 512 (for models with 20 or more parameters)
- mutation rate: 0.1 (10% chances of mutating a feature of each population members after the crossover)
- selection rate: 0.5 (at each iteration we halved the population size)

It turned out that the main software engineering challenge in this part was the computation time of the whole learning process. Suppose that we use the typical configuration: 50 frameworks, 20 seconds timeout for each framework, 256 initial population members. Recall that each 'population member' is a different parametrisation of the model, i.e. it encodes a different derivation strategy. To compute each member's fitness, we have to run it on 50 frameworks, measure derivation time on each (at most 20 seconds) and compute the average. So, the first iteration of the algorithm would run $256 * 50 = 12800$ different derivations. The worst case is that each such derivation would take 20 seconds. However, the average case we observed from the experiments was around 15 seconds. Thus, the whole first iteration would take:

$$15 * 12800 \;=\; 192000 \; (seconds) \;= 53 \; (hours)$$

And remember that this is just the first iteration! The whole learning process would take a few days to complete. And for more complicated models, where we expanded training set size to 100 and initial population to 512, it would probably take close to a month to complete an experiment.

This is a serious issue. We first looked into decreasing the parameters which have a direct influence on computation time: initial population size, derivation timeout and number of training frameworks. However, when we considered each of them in turn, we realised that they are already minimal and cannot be further decreased. For models with more than 10 parameters, 256 population members barely cover the search space sufficiently enough. Decreasing the training set was out of question too. In order to have any significant decrease in derivation time we would have to cut the number by half. Unfortunately, if we wanted to learn on only 25 (framework, query) pairs, we would have to select them very carefully to cover the framework space sufficiently well. But we did not even know how to measure that coverage (i.e. what framework parameters to use). 50 frameworks was a bare minimum.

The only option was to cut the timeout to 10 seconds. However, we would have to generate new frameworks from scratch in order to get frameworks which are solvable within 10 second. Furthermore, it would not give us sufficient computation time boost, as it would decrease the time by half only. Finally, each decrease in maximal derivation time makes the problem less interesting as there would be less field for improvement.

For a brief moment we considered changing the approach. However, then we realise that there is a solution to our problem - parallelization. A genetic algorithm is an ideal candidate for parallelization as each population member is independent of all others. We decided to split the population into groups with up to 25 members in each group. Then we could measure fitness for each group separately and combine the results. Suppose we again have the usual configuration parameters: 50 frameworks, 20 seconds timeout for each framework, 256 initial population members, 15 seconds average derivation time. Suppose that this time we split networks into 26 groups and run the first phase of generic algorithm (fitness measuring) in parallel for each group. We will have 25 groups with 10 members and 1 group with 6 members. If we assume the ideal scenario, i.e. we have an access to 26 CPUs so that each group can run truly in parallel, our computation time would be:

$$10 * 50 * 15(seconds) = 7500(seconds) = 2(hours)$$

A significant improvement! We could now finish the whole learning process in less than a day. Fortunately, in reality we did have access to 26+ CPUs through Imperial College Condor system - a high-throughput batch-processing system. Thus, each group's fitness measurement was encapsulated in a single Condor task. For measuring fitness of the first population with 256 members, we had to submit 26 Condor tasks.

The final flow structure of our parallel genetic algorithm with the default parameters was:

1. Generate 256 population members randomly - this forms the initial population.
2. Split the population into smaller groups with no more than 20 members per group. For example, for 256 population size and 10 members per group, we would get 26 groups, 25 of which would have 10 member and 1 would have 6 members.
3. Measure fitness for each group in parallel. For example, if current population was 256, we would submit 26 condor tasks, each one responsible for measuring average derivation time of a different member group. All tasks outputted fitness to the same file. Thus, the output of this phase was a large fitness file listing (member, fitness) tuples.
4. Once all condor tasks finish, perform crossover and mutation by supplying the fitness file produced in the previous stage as an input. This stage proceeds as described in the previous chapter - 1/2 of population is selected for mating. Each mating pair produces 2 children. We apply mutation to each produced child. As a result, we will get a new population of strategies which will have half the number of elements of the previous population. For our running example, the new population would have 128 members.
5. Either go to step 2 with the new population obtained in the previous step or terminate.

4.5 Output analysis

The algorithm described in the previous section also logs all populations in a separate 'history' text file. For each population we log each member of the given population together with member's average fitness. As we can imagine, the 'history' file is big and very hard to manually inspect, especially if the model has more than 20 parameters. For example, consider the following:

```
[-0.1,0.2,-0.9,-0.8,-0.9,0.4,0.8,-0.5,0.8,-0.4,0.2,-0.6,-0.2,-0.2,0.0,0.7,0.1,0.9,0.6,-0.4,0.0,-0.9,-0.5,0.2,0.7,-0.3,-0.4] 15647.1
[-0.1,-0.2,-0.5,-0.5,-0.4,0.7,0.9,0.5,-0.9,-0.4,0.9,-0.7,0.6,0.8,-0.4,-0.9,0.5,0.8,0.5,-0.6,0.6,0.3,-0.4,-0.1,0.2,0.0,0.0] 14298.1
[0.4,0.6,0.6,0.8,0.5,-0.5,0.2,0.7,-0.7,0.5,-0.3,-0.6,-0.9,-0.3,0.6,0.5,0.6,0.9,0.8,-0.6,0.0,0.7,-1.0,-0.7,0.4,0.8,0.7] 13429.7
[0.1,0.4,0.5,0.0,-0.2,0.9,0.9,0.0,0.3,0.3,-0.2,0.3,0.2,0.9,0.7,0.2,0.0,0.8,-0.8,0.4,0.7,0.1,0.0,-0.8,0.1,-0.2,-0.8] 13810.1
[0.8,0.2,-0.2,0.1,-0.7,-0.5,-0.1,0.6,0.6,0.6,-0.4,-0.1,0.8,0.9,0.3,0.8,0.4,-0.4,-0.5,0.9,-0.9,0.1,0.6,0.1,0.7,0.5,0.3] 14183.1
[0.3,-0.6,-0.7,-0.8,0.1,0.8,-1.0,-0.2,0.3,-1.0,-0.4,0.1,0.6,-1.0,-0.8,-1.0,-0.1,-0.3,-0.2,0.9,-0.9,0.9,-0.1,0.3,0.4,0.1,0.0] 12939.1
```

The above is a subset of the history file, showing a population member (a derivation strategy) and fitness in the form '[population member], fitness'. We would ideally want to manipulate that output. For example, we may be interested in:

- computing average fitness
- printing population members in a tabulated form (csv)
- extracting certain subset of parameters together with fitness (to determine parameters - fitness correlation)
- computing the ratio of negative values for a certain parameter (the parameters are separated by colons) across all recorded strategies
- extracting only x% of best performing strategies (according to their fitness)

We could probably achieve some of these tasks with awk or sed. But we decided that it would be more convenient to define a separate utility to analyse the history file. The module was written in Scala. It allows for data extraction and manipulation. It also uses JFreeChart library [Obj] for producing plots from the extracted data. All plots presented in the next chapter were created using our Scala utility and JFreeChart library.

4.6 Benchmarking

Having learnt useful trends or strategies, we will be interested in validating what we learnt. We will be also interested in benchmarking the performance of our models. Hence, we have created a separate utility dedicating to benchmarking. It is very similar to the fitness measurement part of learning utility. It also runs into computational time problem. Hence, we had to use the parallelization trick and condor once again.

The benchmarking is performed on different types of frameworks: small, medium, large and medical. The medical frameworks were not synthetically generated. They are real-life problems represented as ABA Frameworks. They were provided by Imperial College Computational Logic and Argumentation group. We use substantially bigger number of frameworks in benchmarking than we use for learning - 300 small frameworks, 100 medium frameworks and 30 medical frameworks. Unfortunately, we only use 10 large frameworks as they are hard to generate within our specified solution time bounds (we require minimum 2 seconds and maximum 120 seconds to solve them).

The benchmarking script requires a configuration file as well, where we specify:

- input / output directories
- input frameworks directory and the number of frameworks we will be using
- location of a file with all strategies we are benchmarking

The output contains all strategies tested, together with their average derivation time. We also log another output file - one per strategy. That output lists all goals a given strategy queried, together with ABA Graph measures recorded for each goal node and the time needed to find the goal. The second output is used to analyse what graph parameters of the query may influence its derivation time. E.g. if a given query is a non-assumption, we will measure ABA Graph parameters of that non-assumption's node (closeness, betweenness, pagerank and so forth). We will present these results in Validation chapter.

One final thing to note here is that we always have to set a timeout on derivation in case it does not finish in seconds. The timeout we used depends on the size of the networks:

- 20 seconds for small frameworks
- 60 seconds for medium frameworks
- 120 seconds for large frameworks and medical frameworks

Chapter 5

Experiments and results

5.1 Recap of the approach

As we already emphasised, our aim was to **learn** the strategies (heuristics) which would improve derivation time. Hence, we wanted to avoid approach in which we first come up with an idea for some good strategy and then we fit that idea to our experiment. Such an approach could be called 'benchmarking' or 'validation', not 'learning'.

However, it is usually impossible to learn everything from data without using any prior knowledge. For example, although we used all available ABA Graph measures in our very first experiment, for more complicated models we had to preselect a set of measures we wanted to use. We learnt that making a model with all possible measures we can think of is not a feasible solution. In order for such a model to work, we would need to supply it with accordingly large number of training frameworks. Also, our initial population size would have to be big. Thus, we faced a trade-off between model complexity and computational feasibility.

For example, consider a model with 10 measures. Suppose that the model has a different decision function for sentence / rule / argument choice point for each player (one for proponent sentence, one for opponent sentence etc.). We have $2 * 2 + 1 = 5$ different decision functions as rule choice is only for proponent and we ignore player choice for now. If we suppose that our model is very generic, we probably want to apply all 10 measures in each possible decision function. If we do so, our model has $5 * 10 = 50$ parameters (weights) all together.

This creates several problems. As already mentioned, we would need more than 512 initial population members to cover a search space with 50 parameters well enough. Increasing initial population size makes it computationally harder and harder to learn anything. Also, it is very hard to analyse a model with 50 parameters, not to mention finding correlation between parameters. We quickly learnt that by using 50 parameters we over-parametrise the model and the output of learning is not sensible at all.

Hence, our subjective heuristic is to use at most 4 measures per decision function. The question was how do we select these measures. Our approach was to start with the most generic model which considers all ABA Graph measures (7 of them). In order to make it feasible, we had a single decision function for each choice point. Thus we had 7 parameters in the model all-together. The output of the experiment showed which measures show promise and which do not. This dictated which measures will be used in the following experiments. It also suggested which new measures may be added to the next model. Choosing appropriate measures was our subjective guess.

5.2 Output visualisation

Since the aim of the experiment was to see a correlation between model weights' signs and performance we needed to analyse the output of experiments to see that correlation. Especially, we wanted to see whether there is any correlation between a positive measure weight and good performance or between a negative measure weight and good performance. As already explained, the former would suggest using heuristics which keep the measure as big as possible, while the latter would indicate on a heuristics which keep the measure as small as possible.

The hard bit was to make sense of the output. Most of our experiments had 20+ parameters (weights) so it was very hard to spot any correlation just by looking at the numbers. To illustrate, we show a typical output of our experiment below:

w0	w1	w2	w3	...	w19	w20	fitness
-0.1	-0.2	-0.4	-0.9	...	0.5	-1	4232.5
-0.7	0.2	0	-0.1	...	0.7	-0.5	4356.1
-0.4	-0.4	-0.1	-0.4	...	0.7	-0.8	4364.8
-0.9	-0.1	0	-1	...	-0.2	0.9	4471.1
...	...	...	...	...	...	...	...
0.6	0.1	0.7	0.5	...	0.8	0	17847.9
0.2	0.2	0.3	-0.9	...	0	0.3	17884.5
0.2	0.5	0.4	0.3	...	0	0.5	18008.7
0.7	0.4	0.7	-0.8	...	0.7	0	18072.7

Table 5.1: 4 best and worst performing population members in tabular form

As we can see, if we have around 500 - 1000 rows of 20+ real number, it is very challenging to see any correlation. Our first approach was to look at only 10-20% best performing members and try to see whether most of them are positive or negative. However, this approach is tedious and error prone. Also, while it is generally possible to see large groups of positive / negative numbers, it gives us no clue on how important a certain measure is.

Hence, instead of looking at tables of numbers we decided to plot the numbers on graphs and look for correlations there. We started with a simple scatter plot:

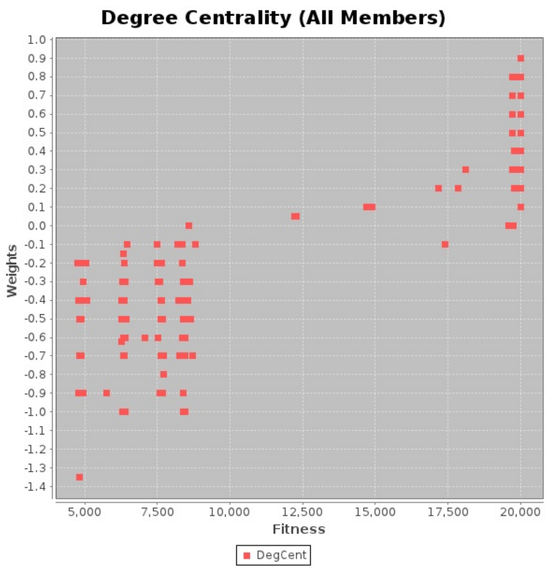


Figure 5.1: Fitness (Average Derivation Time) vs Degree Centrality plotted for 100% of results

Figure 5.1 illustrates the relationship between average derivation time (x axis) and ABA Graph degree centrality weight value (y axis). There is an evident correlation here - all best performing weights have negative value. All worst performing weights have positive value. This may lead to a heuristic which says that at measured choice point we want to make choices which have degree centrality as small as possible.

However, the correlation is not always evident. Consider the following scatter plot:

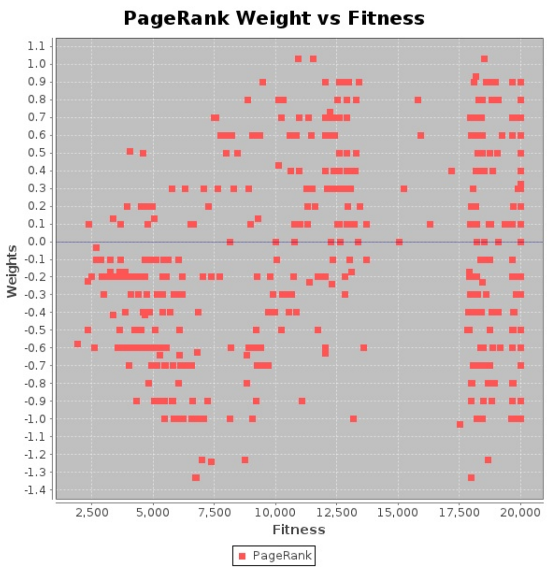


Figure 5.2: Fitness (Average Derivation Time) vs PageRank plotted for 100% of results

Figure 5.2 illustrates the relationship between average derivation time and ABA Graph PageRank centrality weight value. It is very hard to see any correlation. The only thing we notice (if we exclude the rightmost, worst performing dots) is that the diagram is slightly skewed to the right. This actually indicates a similar correlation as in figure 5.1 but it is much harder to see.

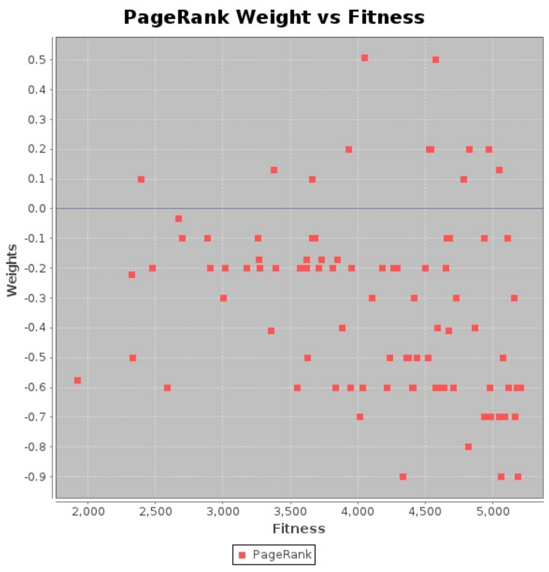


Figure 5.3: Fitness (Average Derivation Time) vs PageRank plotted for 20% of best performing results

One possible solution is to consider only x% of best performing members. Consider the same figure again, but this time we focus only on 20% best performing population members. We make it explicit in plot 5.3.

However, all we want to observe is whether negative weights dominate in the best performing strategies. To see that we do not have to plot all points. Instead, we can compute a **negative weight ratio**: for a certain weight we can sum up all population members where the weight is negative and divide it by the total number of members in a population. We can similarly compute a **positive weight ratio**. This led us to two more ideas on how to graphically represent successful outcomes.

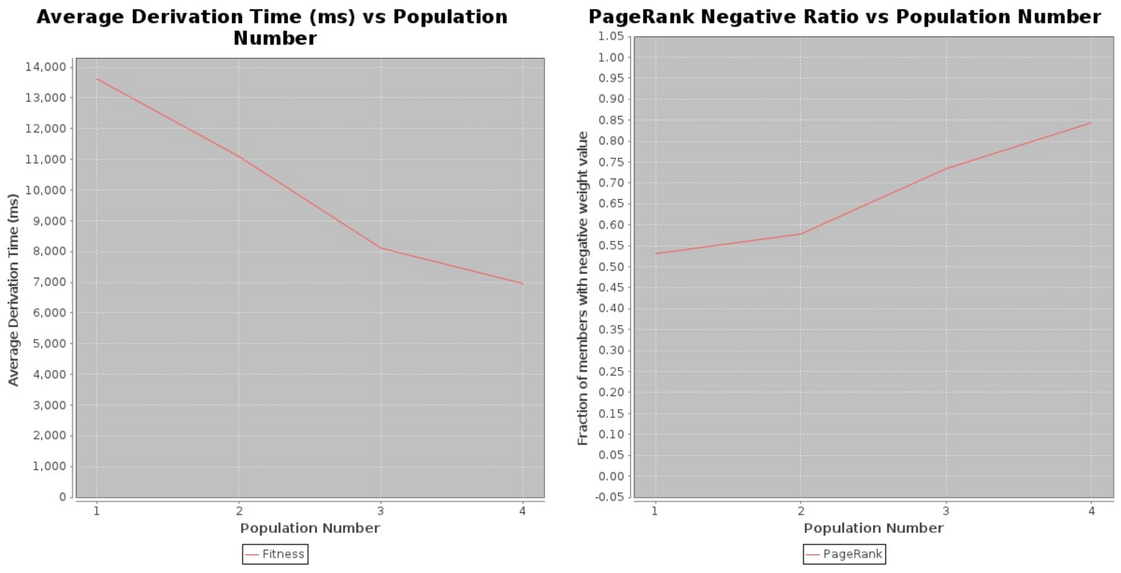


Figure 5.4: Average fitness vs population number (left), % of negative occurrences of PageRank vs population number (right)

Figure 5.4 illustrates the first idea. On the left we can see an average derivation time plotted for four consecutive populations. We can see that the derivation time decreases as populations are improving through genetic algorithm. On the right we see a similar plot, but this time instead of average derivation time we plot the ratio of negative PageRank weights for the same populations.

We can see that the ratio is increasing. Thus, there are many more negative PageRank weights in population 4 than there are in population 1. Since the average derivation time is decreasing and negative PageRank weight ratio is increasing in consecutive populations we may suspect that there is a relationship between the two. We will be using that approach when showing experiment results.

We can also show the relationship between negative (positive) ratio and derivation time more explicitly. Each genetic experiment outputs 5-7 populations of strategies (we choose manually when to stop). Each population consists of different strategies encoded with weights together with their fitness value. Suppose that we treat all populations as a one, singe population. We list every strategy we ever measured together with its average derivation times. We can now assign the strategies to groups.

The first group contains all of them, the second contains 75% best performing strategies (0.75 percentile), the third one contains 50% best performing strategies, the fourth one 25% and then 10%, 5% and 2.5%. We thus obtain what we call best performing percentiles. For a certain weight, we can now compute the negative weight ratio for each percentile (group). As a result, we will obtain a plot which shows how many negative weights we have among 2.5% best performing members, 5% members and so on.

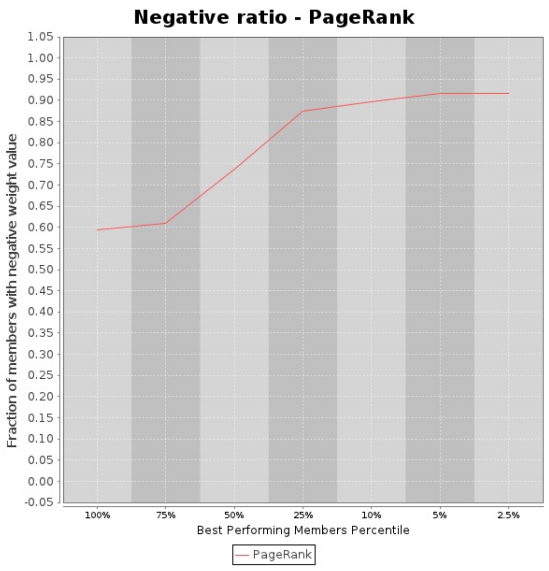


Figure 5.5: PageRank negative weight ratio in experiment best performing percentile

Figure 5.5 is an example of the percentile plot for PageRank (the same measure we used in all previous figures). We can see that that negative PageRank weight ratio is increasing with performance. This show a direct relationship between PageRank measure and derivation time. This is the most important type of plot - we will be showing most of our experimental results in this way. So it is worth studying a bit more carefully.

Remember that we are plotting negative weight ratio. So, if the there is an increasing trend plotted, we may be saying that negative weights dominate in best performing population members. If the trend was down-slopping, we could be saying that positive weights dominate in best performing population members. But, to really see a correlation it is not enough to just look at upward or downward slopping trends. We also have to consider the ratio itself. If the ratio is 0.6 for 10% best group and then goes down to 0.5 for 5% and 0.4 for 2.5% we will not consider it as a trend.

5.3 First model - ABA Generic

5.3.1 Justification of the model

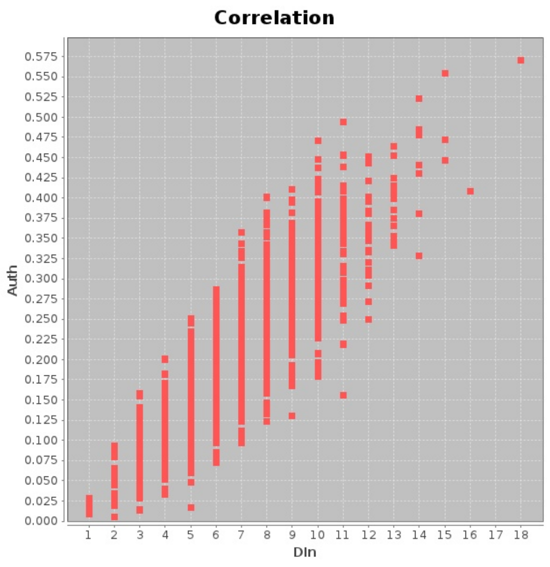
ABA Generic is the very first model we used. The very first idea we had on how to improve derivation performance was to use graph measures and guide the derivation process by computing those measures for each choice point and then using them to make a decision. The main aim of this experiment was to validate that approach by:

- checking whether we can improve derivation speed by using graph measures
- checking whether we have any correlations between derivation speed and graph measures used

5.3.2 ABA Graph analysis

As a part of the experiment, we performed an analysis of ABA Graph measures. We wanted to answer two questions here. First of all, we wanted to see whether the measures are correlated and how they are correlated. Secondly, we wanted to know whether there are any other special characteristics of the graph. Recall that ABA Graph has three types of nodes: assumption nodes, non-assumption nodes and rule nodes. Each type of node may have a slightly different characteristics. Hence, we performed the analysis separately for each type of node. To perform the analysis, we used 400 different ABA Graphs constructed from three types of ABA frameworks: small, medium and large.

Assumption nodes



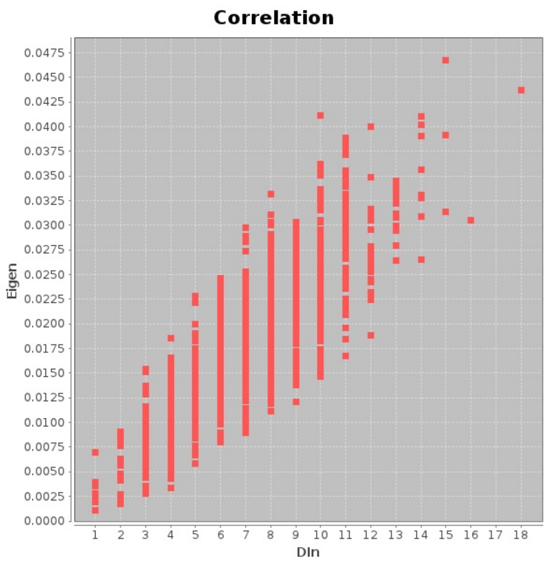
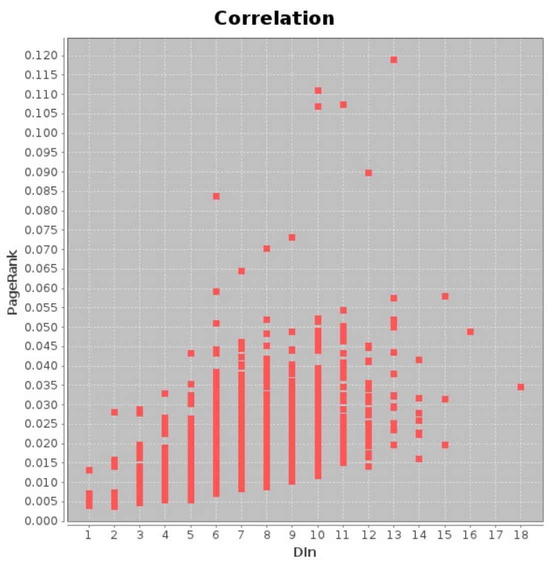


Figure 5.6: Correlation between degree-in and authority (left), and degree-in and eigen centrality (right)



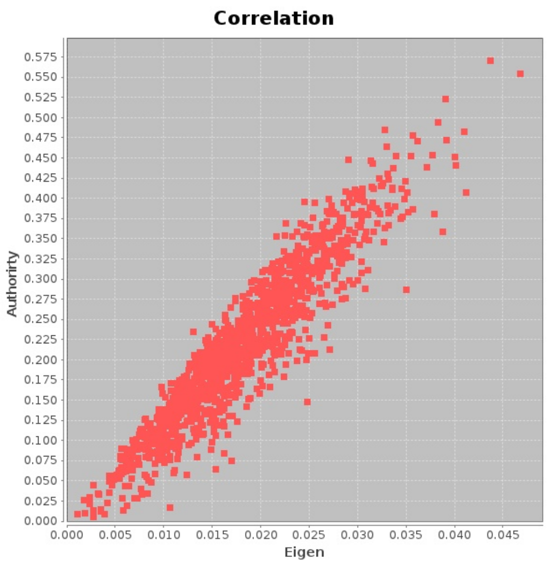
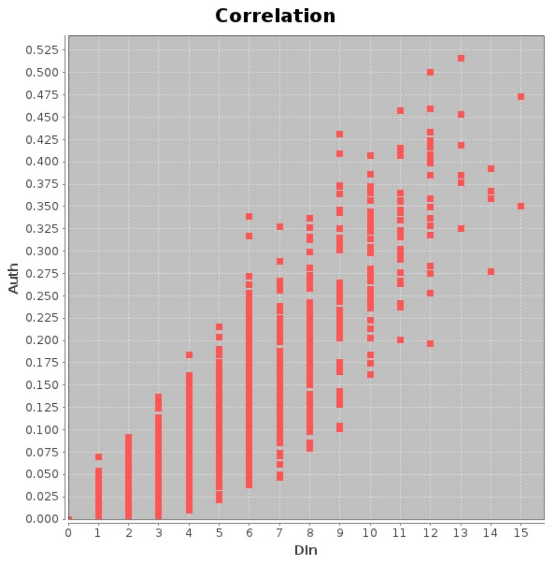


Figure 5.7: Correlation between degree-in and PageRank (left), and eigen centrality and authority (right)

As we can see, degree-in, authority and eigen centrality are strongly correlated (especially the last two). This may indicate that using all of them in a single model is redundant. On the other hand, PageRank is not correlated with any of the measures. We include PageRank correlation with degree here because we will be using these two measures in most of our models. No further correlations were found.

Non-assumption nodes



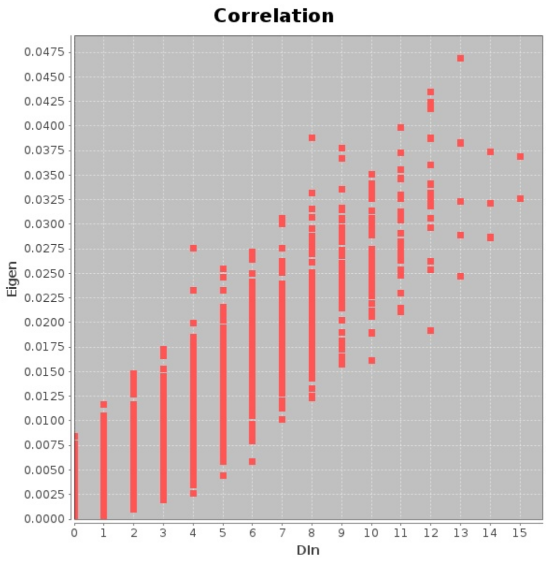
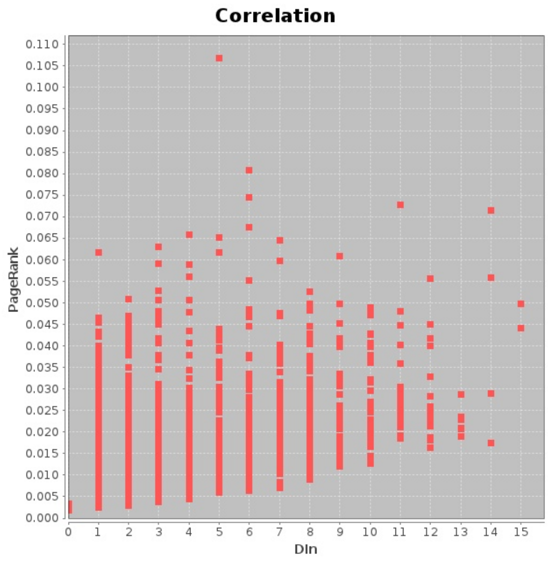


Figure 5.8: Correlation between degree-in and authority (left), and degree-in and eigen centrality (right)



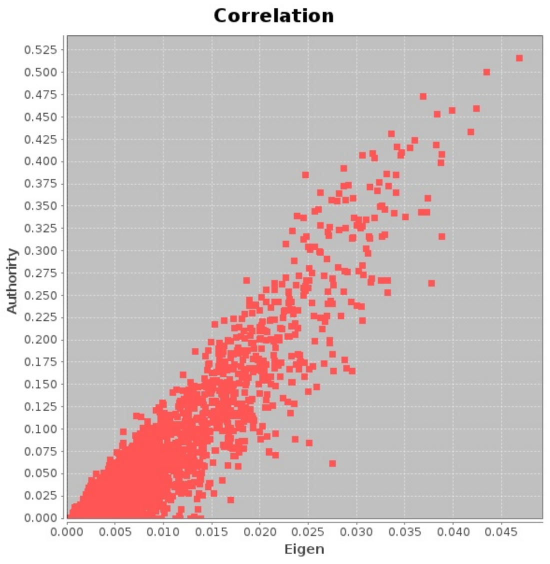


Figure 5.9: Correlation between degree-in and pagerank (left), and eigen centrality and authority (right)

The output is very similar to assumption nodes. Thus, if we consider degree-in, authority and eigen centrality, the two sets of nodes are very similar.

Rule nodes

No correlations found.

5.3.3 Model description

Measures

Measures considered:

- degree centrality
- closeness centrality
- betweenness centrality
- eigen centrality
- pagerank centrality
- hub centrality
- authority centrality

All measures were obtained from ABA graph. I.e. we translated input ABA framework into an ABA Graph and computed all 7 measures for each node of the graph.

Choice points considered:

- proponent sentence choice
- opponent sentence choice
- proponent rule choice
- proponent argument choice
- opponent argument choice

As a side note, proponent vs opponent turn choice was not considered here. Proponent was given priority to move every time. This is a simplification of the problem. The aim of this experiment was to deduce the importance of each measure and decide which measures will be reused in future models, not to find efficient strategies for opponent and proponent or contrast the respective strategies of different players. In some future experiments, we will be running a single experiment twice, once giving a proponent priority and once giving priority to opponent. We will then be contrasting strategies for each player. However, to manage the complexity, we never implemented a model in which player turn choice was included.

For all other choice points we applied the same set of measures. Getting measures for sentence and rule choice was straightforward. All we had to do was to read those measures straight from the input ABA Graph as each sentence and rule node in the graph has all measures precomputed. For example, suppose that some data equivalent to tables 5.2 and 5.3 is attached to the input framework.

Sentence Node	Degree	Closeness	Betweenness	Eigen	PageRank	Hub	Authority
a	3	0.345	92.4	0.07	0.0048	0.0621	0.058
b	1	0.221	0	0.25	0.021	0.0121	0.123
x0	2	0.289	295.64	0.0067	0.012	0	0.1421

Table 5.2: Sentence input measures

Rule Node	Degree	Closeness	Betweenness	Eigen	PageRank	Hub	Authority
x0 :- [c,d]	3	0.0	0	0.091	0.042	0.053	0.0
x0 :- []	1	0.123	225.1	0.012	0.01	0.0236	0.0

Table 5.3: Rule input measures

Suppose we are a proponent and we have to choose a sentence to expand. We have three options: a, b, x0 (a,b are assumptions, x0 is a non-assumption). All we have to do to get measures associated with these options is to read them from input file (which includes information equivalent to 5.2). One way to think about it is that our derivation procedure has full access to 5.2 so it can read all relevant measures from the table.

Argument choice is slightly more complicated. We do not have argument nodes in ABA Graph. Generating all possible arguments from ABA framework is computationally infeasible. Thus, we do not have precomputed measures per each argument which we can easily read from input framework. To define any sensible measures for an argument we look at its unmarked set. In fact, this will be the case for all experiments, so we highlight it here:

To obtain argument's measures we look at measures associated with its unmarked set.

This is the most intuitive formulation of argument's measure. Sentences in unmarked set intuitively denote 'what is left to do' in order to finish deriving the full argument.

We will show how we compute argument measure by running through a short example. Suppose we have two arguments:

```
  arg1: [a,b] :- p0
  arg2: [x0] :- q0
```

where each argument is written in the form: unmarked set :- head. To compute measures for each argument we add measures precomputed for sentences in its unmarked set. For 'arg1', we would add measures precomputed for 'a' and for 'b'. For arg2, we take measures precomputed for 'x0'. We obtain:

Argument	Degree	Closeness	Betweenness	Eigen	PageRank	Hub	Authority
arg1	4	0.566	92.4	0.32	0.0258	0.0743	0.181
arg2	2	0.289	295.64	0.0067	0.012	0	0.1421

Table 5.4: Argument measures

Parameters (weights)

We applied the same set of weights for each choice point. In other words, the model had just 7 parameters (weights):

- degree centrality weight
- closeness centrality weight
- betweenness centrality weight
- eigen centrality weight
- pagerank centrality weight
- hub centrality weight
- authority centrality weight

To clarify, let us go through a short example. We will explicitly show how different choice points are evaluated.

A first step is to fix a strategy. Suppose we have:

w1	w2	w3	w4	w5	w6	w7
-1.0	0.5	0.0	1.0	0.25	0.25	0.0

Table 5.5: Derivation strategy

Suppose the proponent is currently moving. First, he will be selecting an argument. Suppose he is making a choice between arguments defined in table 5.4. He has to compute decision value associated with each argument. For that he will be using measures defined in 5.4 (call them M_{arg1} and M_{arg2}) and weights defined in 5.5 (vector W).

Before computing decision value, we have to normalise the measures:

Argument	Degree	Closeness	Betweenness	Eigen	PageRank	Hub	Authority
arg1	0.67	0.67	0.24	0.98	0.68	1	0.56
arg2	0.33	0.33	0.76	0.02	0.32	0	0.44

Table 5.6: Argument measures - normalised

We can now apply them to compute decision value for each argument:

$$decision\_value(arg1) = W * normalised(M_{arg1})$$
$$= 0.67 * -1.0 + 0.67 * 0.5 + 0.24 * 0 + 0.98 * 1.0 + 0.68 * 0.25 + 1 * 0.25 + 0.56 * 0.0 = 1.065$$

$$decision\_value(arg2) = W * normalised(M_{arg2})$$
$$= 0.33 * -1.0 + 0.33 * 0.5 + 0.76 * 0 + 0.02 * 1.0 + 0.32 * 0.25 + 0 * 0.25 + 0.44 * 0.0 = -0.065$$

The proponent chooses argument 1. Now, the proponent has to choose the sentences from arg1's unmarked set size which he wants to expand. Again, we computed the normalised measures for each sentence:

Sentence Node	Degree	Closeness	Betweenness	Eigen	PageRank	Hub	Authority
a	0.75	0.61	1	0.22	0.19	0.84	0.68
b	0.25	0.39	0	0.78	0.81	0.16	0.32

Table 5.7: Sentence input measures - normalised

And now we use exactly the **same** set of weights as for argument choice to compute the decision value for each sentence:

$$decision\_value(a) = W * normalised(M_a)$$
$$= 0.75 * -1.0 + 0.61 * 0.5 + 1 * 0 + 0.22 * 1.0 + 0.19 * 0.25 + 0.84 * 0.25 + 0.68 * 0.0 = 0.0325$$

$$decision\_value(b) = W * normalised(M_b)$$
$$= 0.25 * -1.0 + 0.39 * 0.5 + 0 * 0 + 0.78 * 1.0 + 0.81 * 0.25 + 0.16 * 0.25 + 0.32 * 0.0 = 0.9675$$

And we choose b. In this model the same vector of weights - e.g. [-1.0, 0.5, 0.0, 1.0, 0.25, 0.25, 0.0] is applied to each choice point (proponent, opponent, rule, sentence, argument etc). In later models we will introduce separate weight for each of the choice points (e.g. proponent sentence degree weight, opponent argument pagerank weight etc.).

5.3.4 Learning outcomes

Representing outcome graphically

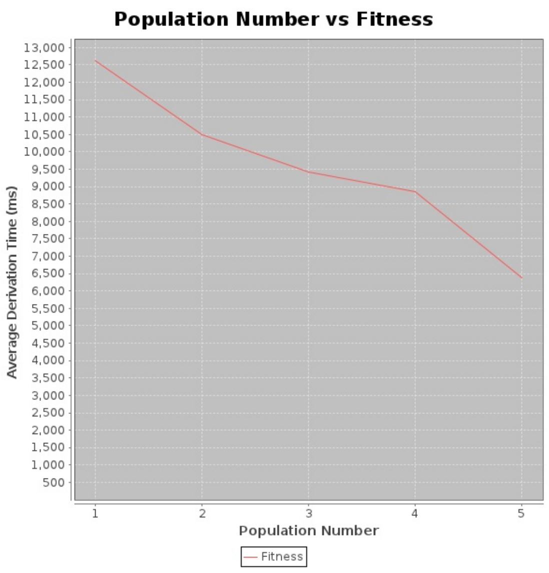


Figure 5.10: Average Fitness for consecutive populations

Figure 5.10 shows that average fitness was improving with consecutive populations. The output for each measure is expressed as percentile graphs below:

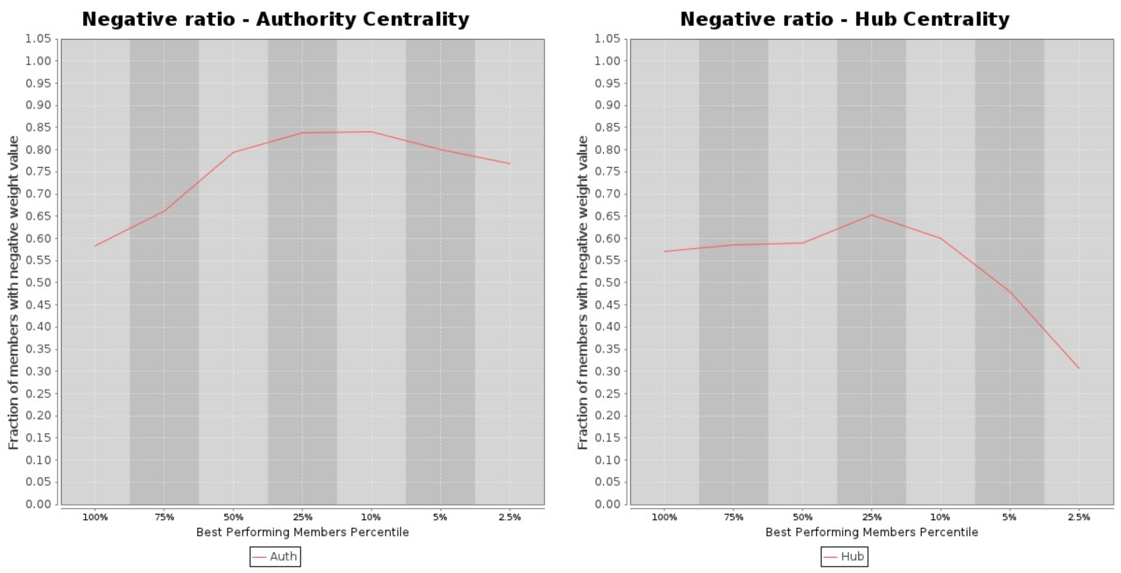


Figure 5.11: Fitness (Average Derivation Time) vs Authority (left) and Hub Centrality (right) for 100% of results

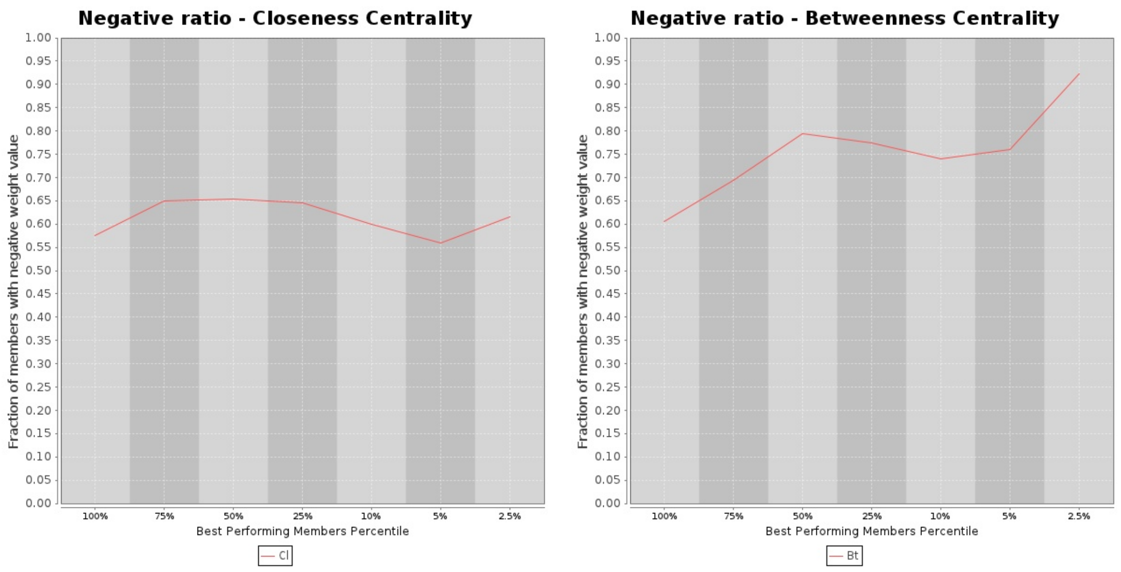


Figure 5.12: Fitness (Average Derivation Time) vs Closeness (left) and Betweenness (right) for 100% of results

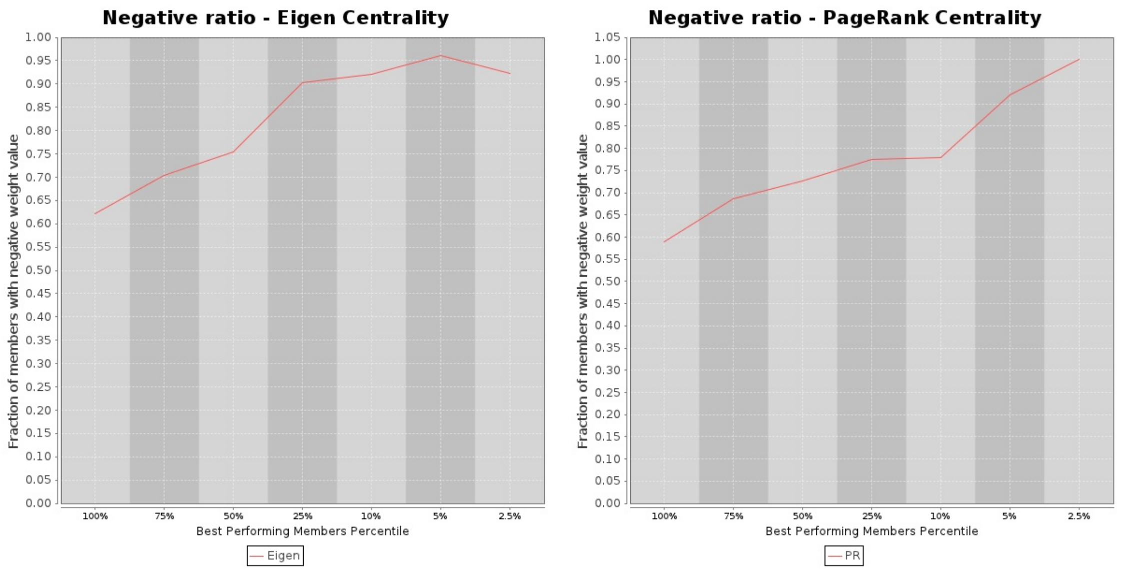


Figure 5.13: Fitness (Average Derivation Time) vs Eigen Centrality (left) and PageRank centrality (right) for 100% of results

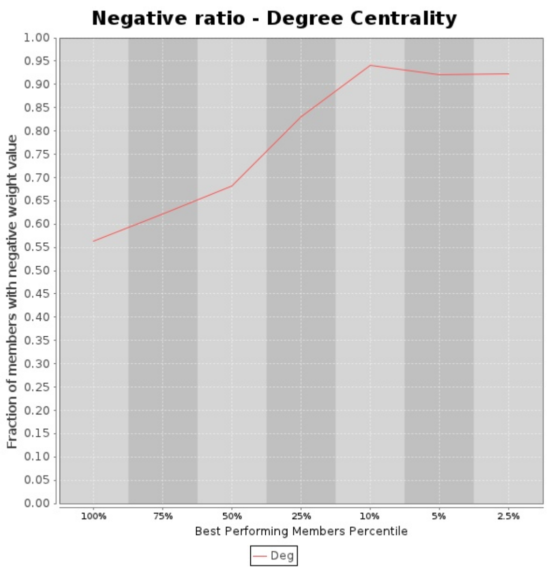


Figure 5.14: Fitness (Average Derivation Time) vs Degree Centrality for 100% of results

5.3.5 Output analysis

We can observe correlations for most of the measures. The only flat percentile graph we observed belongs to closeness centrality (Figure 5.12 left). This may lead to a belief that closeness is not the best candidate for derivation heuristics. The tendency for Hub Centrality is also uncertain (Figure 5.11 right). Although the diagram is downward-sloping it does so only for 5% and 2.5% best performing strategies. In these groups, we noted a larger number of positive weights than negative weights but the tendency is very uncertain. For all other measures the diagrams are mostly upward slopping and the ratio of negative to positive weights is usually higher than 0.6 even for only 50% of best strategies. Hence all other measures seem to be converging toward negative weight values. In some cases the tendency is much more evident than in others.

The strongest correlation was observed for degree centrality (Figure 5.14), eigen centrality (Figure 5.13 left) and PageRank centrality (Figure 5.13 right). They all converge toward negative values which suggests that best performing strategies prefer choices with small degree centrality, eigen centrality and PageRank centrality measures. However, the experiment does not give us an explicit justification as to why these measures may be important. As of now, we can only guess.

Eigen centrality is correlated with degree-in centrality (Figure 5.9 right). Minimising eigen centrality may be equivalent to minimising degree-in centrality. Authority is strongly correlated with eigen centrality and also with degree-in (Figure 5.8). It is not surprising that authority also converges toward negative weight values, just like eigen centrality. PageRank centrality is not correlated with any other measure.

A slightly less evident tendency was observed for betweenness centrality which seems to be converging toward negative weight values.

5.3.6 Results validation

One of the problems with above graphs is that we are not exactly certain which measure is the most important. Several measures are correlated and we would like to choose one of them for further experiments as choosing all would be redundant. Also, we identified measures with the strongest correlation with performance and we would like to validate that.

We employed a simple scheme to do a quick validation. We set the weights of the model so that we make the notion we want to validate explicit. For instance, if we want to check whether what we learnt about degree centrality is correct - i.e. efficient heuristic would prefer to keep it small, we could check the performance of the following strategies:

Degree Weight	w2	w3	w4	w5	w6	w7
-1.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0

Table 5.8: Degree competing strategies

i.e. we explicitly set Degree Weight to -1 and to +1 and we let the two strategies compete. We can do the same for the rest of strategies. Here is what we obtained:

Num	Degree	Closeness	Betweenness	Eigen	PageRank	Hub	Authority	Avg T (ms)
1	-1	0	0	0	0	0	0	7031.2
2	0	-1	0	0	0	0	0	16173.7
3	0	0	-1	0	0	0	0	13959.5
4	0	0	0	-1	0	0	0	9168.9
5	0	0	0	0	-1	0	0	10341.6
6	0	0	0	0	0	-1	0	19487.1
7	0	0	0	0	0	0	-1	13208.7
8	1	0	0	0	0	0	0	20000
9	0	1	0	0	0	0	0	20000
10	0	0	1	0	0	0	0	20000
11	0	0	0	1	0	0	0	20000
12	0	0	0	0	1	0	0	20000
13	0	0	0	0	0	1	0	19175.4
14	0	0	0	0	0	0	1	20000

Table 5.9: Results for small frameworks with timeout 20sec

Evidently, degree centrality proved to be the most significant measure for derivation time (strategy 1), followed by eigen centrality (strategy 4) and PageRank (strategy 5). Also, we can see that learnt tendency toward negative values holds - the strategies which prefer choices with larger ABA graph measures timeout (i.e. the strategies with positive weights - from strategy number 8 to 14), while the strategies which minimise the measures perform much better (strategies from 1 to 7 excluding 6). The interesting case is for hub centrality. Although figure 5.11 (right) may indicate that there is a slight tendency toward positive weights, we said that the trend is very uncertain. However, consider strategies 13 and 6. In the former, we set the weight explicitly to +1 and in the latter we set the weight to -1. We see that their performance is very similar. Finally, closeness centrality, although did not show any correlation in Figure 5.12, here we can see that the 'negative-weight' closeness (strategy 2) slightly outperforms the 'positive weight' closeness (strategy 9).

5.3.7 Conclusions and next course of action

Below we present several ideas why it may be beneficial to minimise degree centrality (note, degree centrality is computed by summing degree-in and degree-out of nodes). We will come back to these ideas and explain them carefully in later experiments. They will help us justify measures we use in other models.

- Choosing non-assumptions/arguments with small degree-out may end derivation and force back-tracking. For instance, consider opponent argument choice where he has two arguments - arg1 has empty unmarked set and was not attacked by proponent, arg2 has many premises in its set. Arg1 would have a small degree-out measure, arg2 would have a bigger degree-out measure. If we choose arg1 we effectively end derivation (or the current path in derivation) because the proponent has no way of attacking arg1. The algorithm would backtrack to the previous proponent rule choice to choose another derivation path. If we choose arg2, we may prolong the derivation and do some unnecessary work (expanding arg2) since, regardless of what arg2 is, the opponent will, at some time in future, finally choose arg1 anyway, thus ending derivation and leading to the same conclusion.

- Proponent choosing assumptions with 0 degree-out or arguments with assumptions which has 0 degree-out reduces the number of ways he can be attacked. Recall that an assumption node in ABA Graph has degree out either 0 (if it does not have any contrary) or 1 (if it does).

As we can see, we need to further experiment to discover the exact reasons why degree centrality is significant. In next experiments we try to split degree weight into many independent weights. We will introduce degree-in and degree-out measures and separate measures for assumptions and non-assumptions. We will also differentiate between proponent and opponent and sentence, argument, rule choice points.

We can list similar conclusions for authority centrality and eigen centrality as they are correlated with degree centrality. A very weak correlation was observed for closeness and hub centralities so we will not be experimenting further with these measures. We also observed correlation between performance and negative PageRank centrality which we will further investigate in future experiments.

Moreover, we observed a similar correlation for betweenness. Recall that betweenness measures the number of shortest paths in the graph a given node lies on. In ABA terms, this could approximate the number of derivation paths a given node lies on, i.e. the number of times a given nodes occurs in any derivation. So the experiment would suggest that the better performing strategies prefer to choose nodes which occur less often in derivations. However, the definition of betweenness excludes backtracking so it is a very rough approximation. It is also infeasible to compute for larger frameworks. Hence, it is a very interesting measure but we will not consider it in further experiment. In order to be useful, it would probably need to be redefined.

All of the above discussion relates to assumption and non-assumption nodes and measures. For rule nodes, we have not spotted any correlations in graph parameters. This creates a problem - which measures should we use for rule nodes. Since we do not have any data our best guess is to use the same measures as for assumption and non-assumption nodes.

5.4 Second model - ABA Simple

5.4.1 Model justification

As the previous experiment indicated, not all ABA Graph measures may be a good candidate to be used in efficient derivation heuristics. Also, some ABA graph measures are correlated and so, following Occam's Razor, it is enough to focus on just one of the correlated measures, reducing the number of parameters in the model. We thus had to choose measures on which we will be focusing. We immediately got rid of closeness centrality and hub centrality as they showed the least correlation with performance in the previous experiment. We also excluded betweenness as it was only slightly correlated with performance and, more importantly, it is extremely hard to compute (infeasible for real-life networks).

Degree centrality, eigen centrality and authority are correlated, as shown on figures 5.6 and 5.8. As we already mentioned, we decided to choose only one of them and the choice fell on degree centrality, as it showed the strongest correlation with derivation time in the previous experiment. Since PageRank was not correlated with any other measure and we saw in ABA Generic Experiment that it is quite possible that PageRank influences derivation time, we included it here as well.

Thus, there were two main aims of the experiment. First of all, we wanted to further explore the correlation between degree centrality and derivation speed but this time without a dozen of other parameters interfering with learning process. Secondly, we wanted to see whether PageRank is worth further experimentation as a measure. Finally, we included a new parameter - unmarked set size for arguments choice point. It is our initial attempt at discovering whether our intuition with choosing arguments with empty unmarked set to speed up derivation (previous section) would manifest here as a strong correlation between unmarked set size and performance.

5.4.2 Model description

The model is a simplified version of ABA Generic model considered in the previous experiment. Hence, the two are very similar. ABA Simple uses only three measures:

- M1: Degree Centrality
- M2: PageRank Centrality
- M3: Unmarked Set Size

The first two weights are used in each choice point we consider: proponent/opponent sentence choice, proponent rule choice and proponent/opponent argument choice. The last one is used only for argument choice.

Computing measures for sentences and rules is straightforward - we simply read Degree Centrality and PageRank Centrality precomputed for sentence node and rule node straight from input ABA Framework (same as in ABA Generic). Computing measures for arguments is slightly more complicated. Argument Degree Centrality and Argument PageRank Centrality are computed in exactly the same way as in ABA Generic - we sum degree and PageRank measures of each sentence in argument's unmarked set. To compute Unmarked Set Size, we count the elements in argument's unmarked set.

We use the same set of weights for each choice point (again, same as ABA Generic). Thus, we have only three parameters:

- W1: Degree Centrality Weight
- W2: PageRank Centrality Weight
- W3: Unmarked Set Size Weight

For proponent/opponent sentence choice and for proponent rule choice, we compute the decision value of each possible derivation choice in the following way:

$$decision\_value(choice) = W1 * M1(choice) + W2 * M2(choice)$$

For argument choice, we also include unmarked set size (M3):

$$decision\_value(argument) = W1 * M1(argument) + W2 * M2(argument) + W3 * M3(argument)$$

5.4.3 Learning outcomes

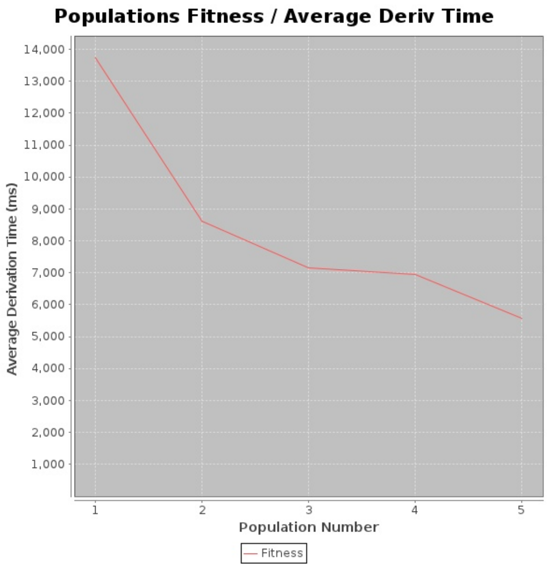


Figure 5.15: Average Fitness for consecutive populations

Figure 5.15 shows that average fitness was improving with consecutive populations. The output for each measure is illustrated below:

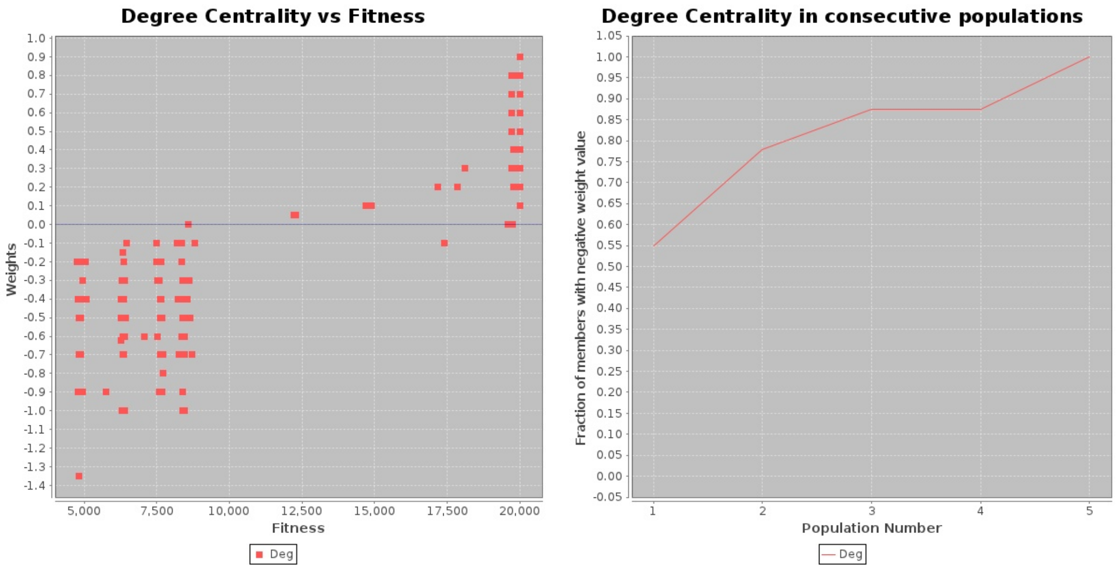


Figure 5.16: Degree Centrality weight vs Derivation Time - scatter plot (left), Degree Centrality changes in consecutive populations (right)

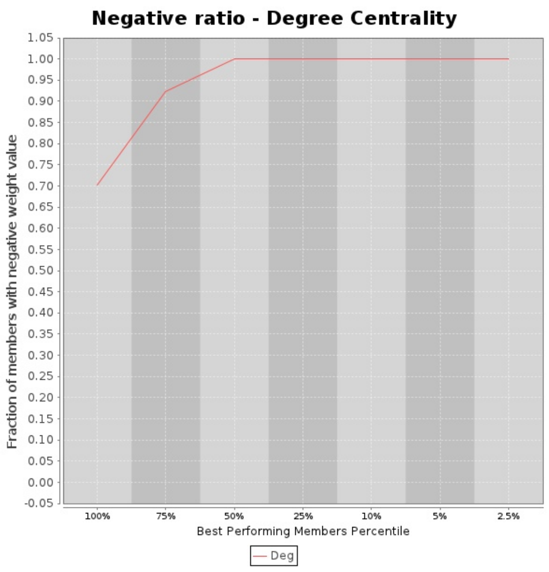
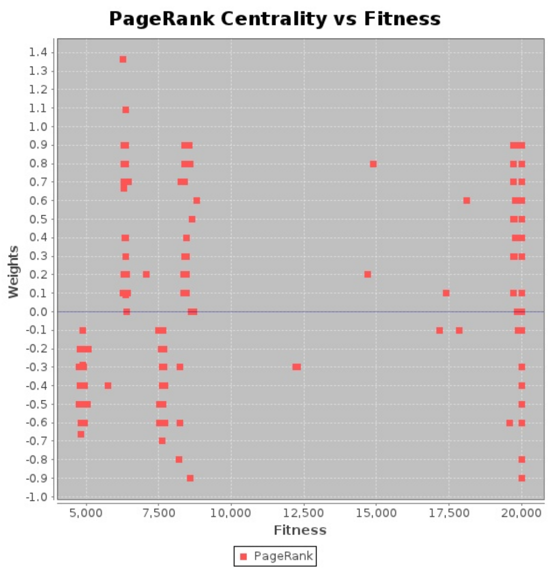


Figure 5.17: Degree Centrality weight in best performing percentiles



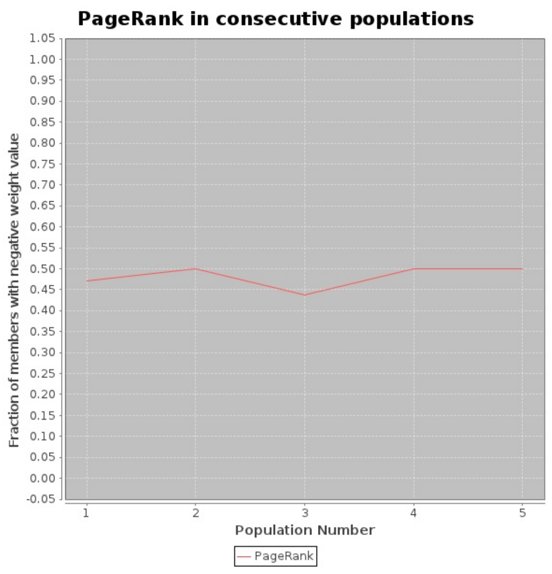


Figure 5.18: PageRank weight vs Derivation Time - scatter plot (left), PageRank changes in consecutive populations (right)

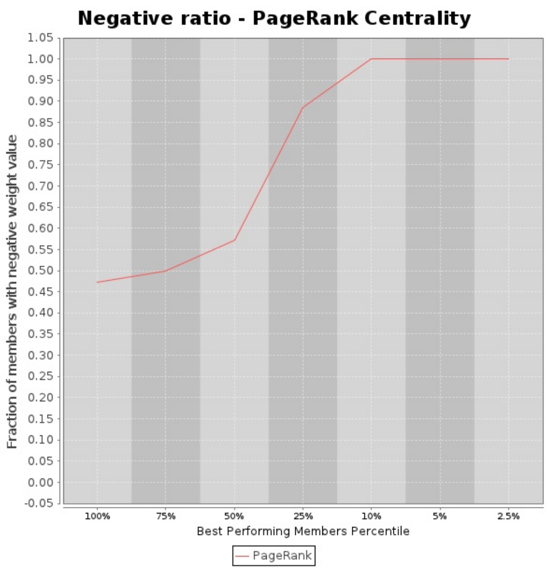
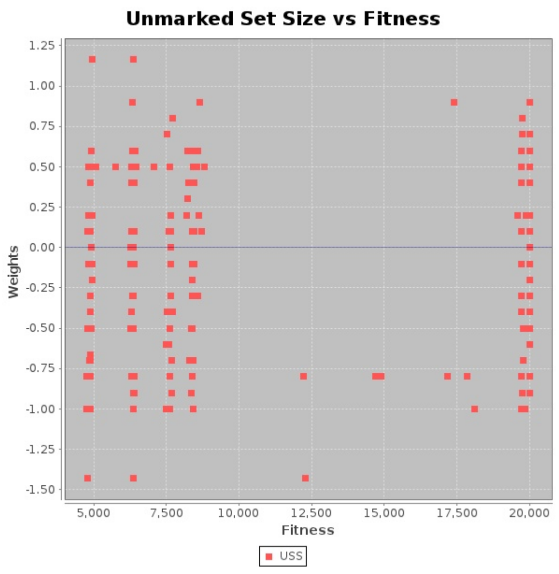


Figure 5.19: Degree Centrality weight in best performing percentiles



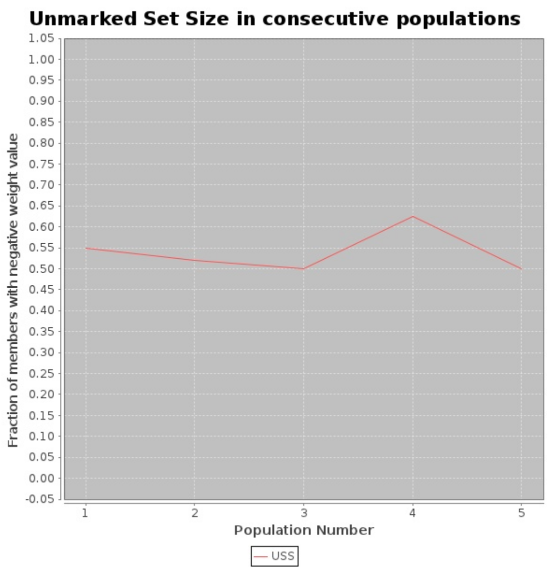


Figure 5.20: Unmarked Set Size weight vs Derivation Time - scatter plot (left), Unmarked Set Size changes in consecutive populations (right)

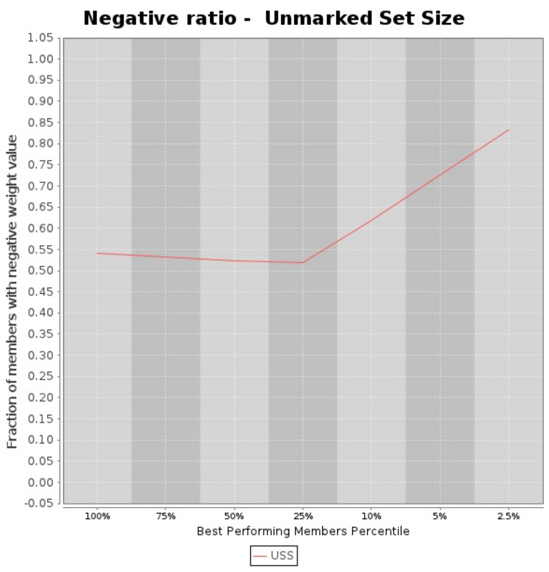


Figure 5.21: Unmarked Set Size weight in best performing percentiles

5.4.4 Results validation

Similarly as in ABA Genetic case, in order to validate the results we prepared strategies where we set parameters manually in order to match the output of the learning. We then benchmarked those strategies on small and medium synthetic frameworks.

	Deg	PR	USS	Description	Small (ms)	Medium (ms)	Large (ms)
1	-1	0	0	Low Degree	8394.52	37100	46684.44
2	0	-1	0	Low PageRank	11585.5	44699.33	86692.22
3	0	0	-1	Low Unmarked Set Size	13717.44	49982.67	92184.44
4	1	0	0	High Degree	19973.66	60000	120000
5	0	1	0	High Pagerank	19956.875	60000	120000
6	0	0	1	High Unmarked Set Size	20000	60000	120000
7	-1	-0.5	0	Low Degree, low PageRank	15317.025	43329.22	66252.22
8	1	0.5	0	High Degree, high PageRank	19960.9	60000	120000

Table 5.10: Result of benchmarking strategies on Small, Medium and Large frameworks

5.4.5 Output analysis and discussion

Evidently, degree centrality has the biggest impact on derivation time. As we can see in figure 5.16 (left), the strategies were split into two groups - all best performing members have negative degree centrality weight while all worst performing members have positive degree centrality weight. It seems to be a decisive factor when it comes to derivation performance. Other figures further validate that notion. In figure 5.17 we again see a that negative degree centrality dominates - all best 50% performing members (0.5 best percentile) have degree centrality -1. Also, derivation time is decreasing in consecutive populations (figure 5.15) while the number of negative degree centrality weights is increasing (figure 5.16).

The trend seems much weaker when we consider PageRank measure. Figure 5.18 (left) shows that it is hard to find any correlation between PageRank and performance. While the very best performing members have negative PageRank value (the group with average derivation time around 5000 ms), the next best performing group (6000ms) has mostly positive PageRank. Although the number of negative

PageRank weights is increasing for best performing percentiles (figure 5.19), the same cannot be said about consecutive populations where negative PageRank weights do not seem to dominate (Figure 5.18 right).

Finally, it is very hard to argue that there is any trend for Unmarked Set Size parameter. Although it does seem to converge toward negative values when we consider the best performing members (figure 5.21), the trend is not very significant. For instance, if we consider a group of 25% best performing strategies, only slightly more than 50% of them have negative unmarked set size (figure 5.21 again). Contrast it with Degree Centrality, where in the same group 100% members have Degree Centrality = -1 (figure 5.17).

The output is compatible with validation we have performed. Low Degree strategy (number 1) is the best performing strategy for all types of networks considered. Low PageRank strategy (2) falls behind, especially in the largest set of frameworks. Note that any strategy which maximises any of the measures (strategies 4, 5, 6, 8) mostly time-out and have much worse performance than strategies which minimise instead. Furthermore, if we add mix Degree Centrality and PageRank (strategy 7) we actually decrease the performance, especially for small frameworks.

This leads us to the following conclusions:

- Degree Centrality is important for derivation performance and may form a basis for a valid heuristic speeding ABA derivations.
- In order to propose useful heuristics, we need to learn why Degree Centrality impacts performance, which we will try to do in next experiments.
- PageRank had much smaller impact on performance. When we mixed it with Degree Centrality (Table 5.10), the performance decreased. Thus, we decided to exclude it from future models.
- The experiment did not show any correlation between performance and Unmarked Set Size. This parameter was introduced to check whether a strategy which prefers choosing smaller arguments (i.e. arguments with less 'work to do') over larger arguments (or vice-versa) may form a valid heuristic. Neither validation (where we set unmarked set size parameter to -1) nor learning showed any evidence which would support this claim.

5.5 Third model - ABA Simple Extended

5.5.1 Model justification

In the previous experiments we saw that Degree Centrality has a huge impact on derivation performance. However, we have not yet discovered why. When discussing ABA Generic experiment we listed two possible reasons:

- choosing smaller degree-out arguments/non-assumptions to reduce the amount of work we have to do and to favour empty/nearly empty arguments which may end derivation (especially in opponent's case)
- choosing 0 degree-out assumptions or smaller degree-out arguments/non-assumptions leading to fewer assumptions reduces the number of attacks we are exposed to (may be especially significant for proponent)

We had an initial go at validating the first one by introducing Unmarked Set Size to the previous experiment, but it was unsuccessful.

In this experiment, we will focus on testing Degree Centrality. Since we suspect that degree-out and degree-in may have different impact on derivation, we will separate Degree Centrality into Degree-In Centrality and Degree-Out Centrality. We also suspect that:

- Different choice points may have different significance in derivation performance and may require different priorities/different measures. For example, we could have a strategy which picks arguments with smallest degree-out but picks rules with largest degree-out.
- Different players may also have different priorities. For example, the opponent may prefer to choose arguments with largest unmarked set, while the proponent may prefer to choose arguments with smallest unmarked set.
- Assumptions and non-assumptions may also require different treatment, e.g. a different set of measures. Hence, we will be treating them separately as well.

5.5.2 Model description

The result of separating Degree Centrality is that we will have many more parameters. But we will still follow a heuristic not to choose more than 4 parameters for each choice point (for each decision value equation). In fact, we had three iterations of ABA Simple Extended model, each of them gradually increasing the granularity of Degree Centrality. The first model was still using single Degree Centrality, but used different weights for proponent, opponent and for each choice point. The second model introduced different weights for assumptions and non-assumption. The third model finally split Degree Centrality into Degree-In and Degree-Out centralities. We initially used PageRank centrality as well, but then we decided that it is not showing any promise and the model is over-parametrised anyway and so we removed it.

We arrived at model which uses 21 measures/parameters (i.e. weights, a single weight corresponds to each measure). We will discuss each parameter in the context of its corresponding choice-point.

Proponent sentence choice

We have five parameters here:

- sentence proponent assumption in degree
- sentence proponent assumption out degree
- sentence proponent non assumption in degree
- sentence proponent non assumption out degree
- sentence type

The last parameter is unique and is shared between proponent and opponent. Although it can take any value from -1 to +1 (initially), it should be treated as a binary weight. Sentence Type decides which type of sentences we are expanding. The rule is:

If Sentence Type is $>= 0$ we are always choosing assumptions over non-assumptions (provided that we have any to choose from). If Sentence Type is < 0 we are always choosing non-assumptions over assumptions (provided that we have any to choose from).

The first strategy is often referred to as **eager selection**, while the second as **patient selection**. So when Sentence Type $>= 0$ and we have any assumptions, we do not need to consider non-assumptions at all and vice-versa. This allows us to introduce different measures for assumptions and non-assumptions.

Once we pick which type of sentences we are considering, we apply the corresponding measures as usual. Here, we have two - Degree-In Centrality and Degree-Out Centrality. We read them directly from input ABA Graph, normalise them and multiply them by their corresponding weights to get decision value. We then pick sentence with highest decision value.

Opponent sentence choice

We have five parameters here as well:

- sentence opponent assumption in degree
- sentence opponent assumption out degree
- sentence opponent non assumption in degree
- sentence opponent non assumption out degree
- sentence type

We use different set of weights than in proponent choice. Thus each player is given his unique set of weights (we are done with sharing). Other than that, the computation is exactly the same as in proponent sentence choice described in previous subsection.

Rule choice

There are two parameters here:

- rule proponent in degree
- rule proponent out degree

We read Degree-In and Degree-Out Centralities straight from the input ABA framework and we compute decision value as usual. There is no opponent rule choice as non exist in the algorithm (opponent always chooses all rules to expand his non-assumptions).

Proponent Argument choice

We have five parameters:

- arg proponent unmarked size
- arg proponent assumption in degree
- arg proponent assumption out degree
- arg proponent non assumption in degree
- arg proponent non assumption out degree

Although the previous experiment showed that Unmarked Set Size was irrelevant to the derivation, we decided to stick with it and give it a second chance. We thought that maybe Unmarked Set Size will matter only for proponent or only for opponent, or one of the players will want to maximise it while the other will want to minimise it.

We do not have any sentence type parameter here. We compute decision value by plugging in 5 parameters, non-assumption and assumption parameters together. The process is very similar to ABA Generic model and ABA Simple model. The only difference is that instead of computing a single Degree Centrality value, we now compute 4 measures - in/out degree for assumptions/non-assumptions. We proceed as in previous models - we read in/out degree from input framework for both assumptions and non-assumption and then we do the sum - we add assumptions' in/out degrees and non-assumptions' in/out degrees. For example, suppose our argument's unmarked set consists of 4 sentences - 2 non-assumptions (x0 and y0) and two assumptions (a and b). We have the following:

Sentence Node	Degree-In	Degree-Out
a	2	1
b	3	0
x0	4	3
y0	5	7

Table 5.11: Sentence input measures

From which we can compute the following argument measures:

Unmarked Set Size	Asm Deg-In	Asm Deg-Out	Non-asm Deg-In	Non-asm Deg-Out
4	5	1	9	10

Table 5.12: Argument measures

We repeat the process for all arguments we are considering. Then we apply the usual technique - we normalise the measures and multiply them by corresponding weights to get the decision value.

Opponent Argument choice

We proceed in exactly the same way as in proponent's case.

Turn choice

We did not add turn choice to the model. Instead, we run the experiment twice - in first run we always give priority to proponent to move first (provided he has any valid moves). In the second run, we always give priority to opponent. Both runs showed a very similar results with major trends being the same. We will be following exactly the same approach in all later experiments.

5.5.3 Output analysis and discussion

Since we have 21 parameters, we will not be including all graphs showing each parameter's correlation with performance. Instead, we will be focusing on parameters for which we noted correlations. In this section, we will list and describe the major patterns observed.

Negative Degree-Out Centrality in opponent argument choice

As we suspected, there is a strong trend for negative Degree-Out weight for non-assumptions. It is especially visible in opponent's argument choice (figure 5.22). The same trend was observed regardless of whether proponent or opponent was given the priority to move. This may be a basis of a useful heuristic - **opponent should choose arguments with smallest ABA graph degree-out of its unmarked sentences**. Remember that argument's non-assumption degree-out measure is computed by summing degree-outs of argument's unmarked non-assumptions. Hence, the heuristic would favour arguments with smallest unmarked set size (as there are fewer non-assumptions to sum). This is indeed the case here, as illustrated in Figure 5.23 where we can see Unmarked Set Size weight also converging toward negative values. Also, the heuristic would generally prefer arguments which are 'smaller'. I.e. if we consider an argument as a tree with its head being a root, the heuristic would generally prefer trees with less nodes. Consider two trees derived from ABA Graph from Figure 5.24.

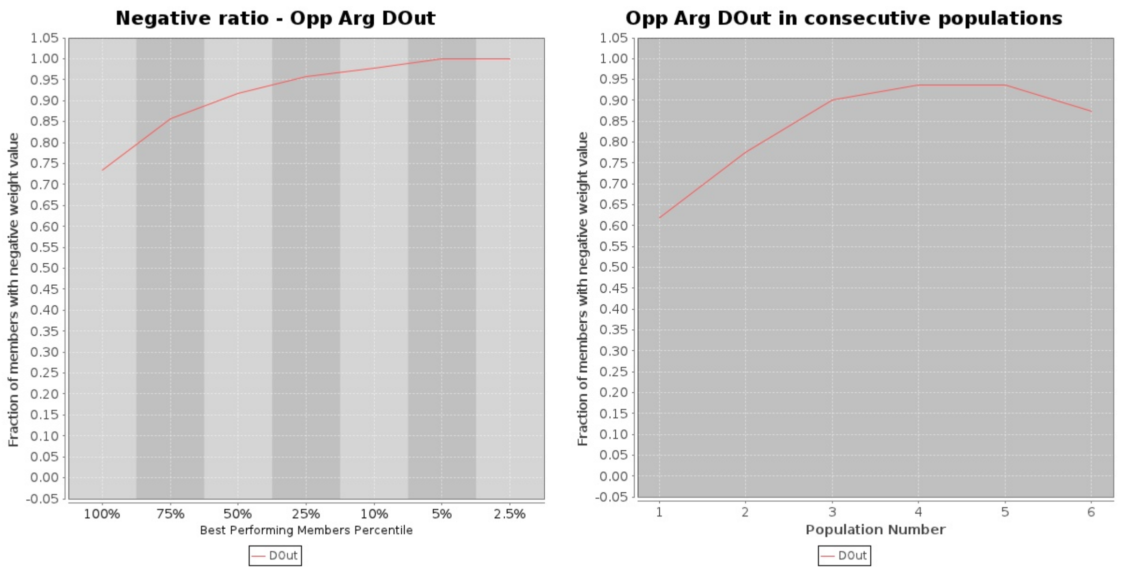


Figure 5.22: Fraction of negative Non-Assumption Degree-Out Centrality weights for opponent argument choice increases in best performing percentiles (left), it also increases in consecutive populations (right)

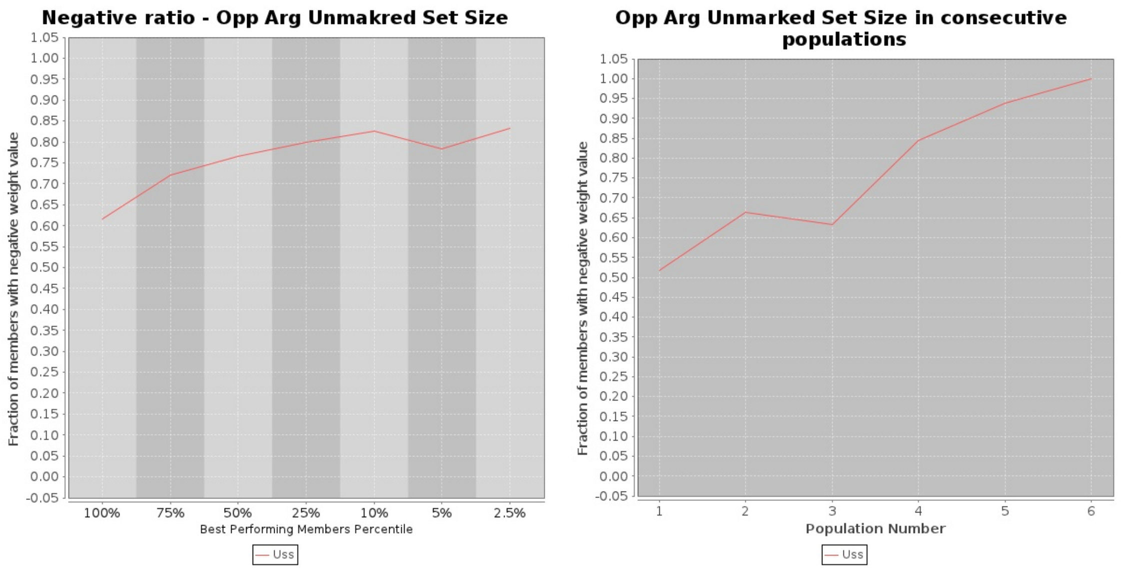


Figure 5.23: Fraction of negative Unmarked Set Size weights for opponent argument choice increases in best performing percentiles (left), it also increases in consecutive populations (right)

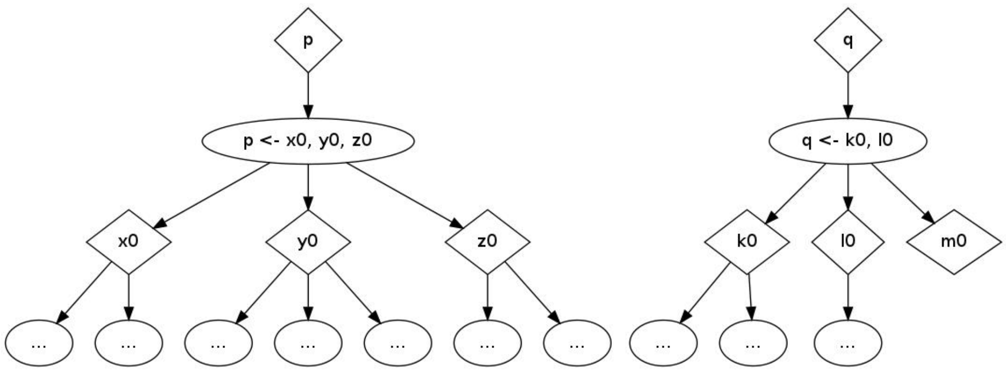


Figure 5.24: Arg1 tree representation (left) vs Arg2 tree representation (right)

Arg1 non-assumption degree-out is 7, arg2 non-assumption degree-out is 3. Although the trees are not fully expanded yet, there is a high probability that the first tree will be bigger than the second one. Hence, degree-out is an estimator of how big the argument may be. But why should the opponent choose to expand 'smaller' arguments first? Suppose the opponent is choosing from the following arguments:

```
arg1:  p0 :- [a,z0]
arg2:  q0 :- [x0, y0]
arg3:  w0 :- []
```

Arg3 has the smallest non-assumption degree-out. Intuitively, by choosing it, the opponent wins as the proponent has no way to counter-attack. In practice the algorithm backtracks to last branching point (for example, proponent rule choice). Suppose we choose arg2 instead and spend 10000 derivation steps to expand it. Then we finally choose arg3 and the derivation backtracks to the exactly same branching point. We have just wasted 10000 derivation steps for doing useless work. Hence, by choosing smaller arguments for opponent, we enforce backtracking to happen sooner than later and reduce useless work. Choosing arguments with Small Unmarked Set Size has a similar effect. The key difference is that Degree-Out Centrality approximate the size of argument a bit more accurately. Consider Figure 5.24 again - both arguments has unmarked set size equal 3.

No patter found in proponent argument choice

No such pattern was found in proponent argument choice. To see why we have to consider ABA derivation algorithm again. Unlike the opponent, if the proponent chooses an argument with empty unmarked set, he does not 'win'. In order to win, he has to reduce **all** of his arguments to empty unmarked set and defend against all opponent's attacks. The algorithm does not backtrack either. After reducing one of his arguments, the proponent will have to choose another one anyway. Thus, he gains nothing by reaching arguments with empty unmarked set as soon as possible.

Negative non-assumption Degree-Out Centrality in proponent sentence choice

Unlike in proponent argument choice, negative Degree-Out Centrality was observed in proponent sentence choice (Figure 5.25). Again, we observed the same trend regardless of which player had priority to move. Again this forms a basis for a heuristic - **when choosing non-assumptions for proponent, choose the one with smallest degree-out measure**. Again, why should we do that? Consider another two small ABA graphs (Figure 5.26). y0 has degree-out 3, while x0's degree-out is 1. Suppose the proponent chooses y0 first. Since y0 is a head of three rules, the proponent will be forced to backtrack three times to expand y0 in three possible ways. This may be unavoidable. However, the problem is when the proponent chooses y0, expands it in one possible way (say by applying rule1) and then expands x0. Although x0 does not immediately causes branching, it may be expensive to expand (thousands of derivation steps). Suppose we then fail at some point (or succeed and find a solution) and backtrack to y0 choice again. We choose rule2 this time. And then we choose x0 again and repeat our thousands of derivation steps to expand it.

The key issue here is with the amount of work we do before we reach branching point. As the example illustrates, it is always better to do work before we branch. We could then avoid doing expensive derivations in each derivation branch. Another thing to note here is that the more work we do before branching the higher the probability that we could fail and not even reach the branching point. Suppose we choose x0 first and fail before picking y0. We will backtrack somewhere higher up in the derivation and may never comeback to x0/y0 choice again.

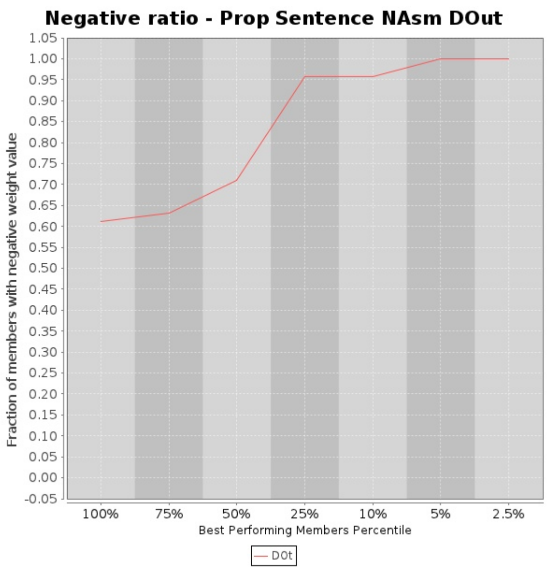


Figure 5.25: Fraction of negative Non-Assumption Degree-Out Centrality weights for proponent sentence choice increases in best performing percentiles)

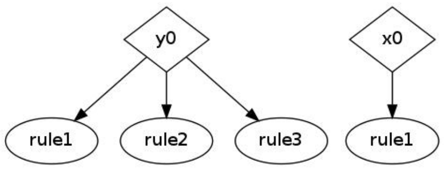


Figure 5.26: Two non-assumptions which proponent can choose represented as ABA graph

Negative degree-out validation

As we have just seen, we learnt two useful degree-out heuristics - both the opponent and the proponent want to minimise their degree-out, the opponent mainly for argument choice and the proponent mainly for non-assumption choice. Let us explicitly set the parameters of the model to obtain the heuristics and validate them by querying 300 small and 100 medium networks. We already followed the same approach in the previous experiments. As a side note, we initially considered moving all validation to the next chapter. However, we will often use validation to draw conclusions supporting (or not) what we learnt. Since we want to have all models and parameters related discussion in one place, we will be interleaving discussion and validation for all future models.

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Reduces branching by minimising degree-out	Negative non-assumption out-degree (-1) and negative unmarked set size (-0.25) for all choice points	7729	37828	4260	26289
Increases branching by maximising degree-out	Positive non-assumption out-degree for all choice points	19918	60000	17366	54000
Increases branching for proponent and reduces branching for opponent	Positive non-assumption out-degree for all proponent choices, negative for all opponent choices	10520	49020	7074	34880

Table 5.13: Different degree-out strategies benchmarked on small and medium frameworks. The results reported are **average derivation times in milliseconds**.

Consider Table 5.13. Two sets of results denote cases when we give each player priority to move first. The strategy parametrised with negative degree-out weights significantly outperforms the strategy with positive degree-out weights regardless of which player is given priority to move. This is consistent with what we learnt. Also, it outperforms the strategy which assigns positive proponent-choice degree-out weights and negative opponent-choice degree-out weights. Finally, we notice that all strategies where the opponent was given priority to move outperformed the corresponding 'proponent moves first' strategies.

No pattern found for rule choice

None of the experiments we performed with ABA Simple Extended model indicated on any relationship between derivation and Degree Centrality or PageRank (previous model iterations) for rule choice. A possible explanation comes again if we consider ABA derivation algorithm. Throughout derivation once a certain non-assumption is chosen by the proponent, it will be expanded in all possible ways. It is different than opponent's expansion where the non-assumption gets expanded in all possible ways immediately. For proponent, we expand it in one way and then backtrack to try other ways. Nevertheless, we will expand it in all possible ways before derivation ends. It seems that the trick is to choose right non-assumptions in the first place as already discussed. Once a non-assumption is chosen, choosing rule to expand it may be less important.

Sentence type choice - preferring assumptions over non-assumptions

All experiments also agreed in another matter - when faced with a sentence choice between assumptions and non-assumptions, we should favour choosing assumptions. Recall that we introduced an explicit binary weight to control that choice. Figure 5.27 (left) illustrates how many negative sentence-type weight values we had in best performing percentiles of strategies. The right figure shows how the negative ratio was changing in consecutive (improving) populations. In both cases the negative ratio decreases. We have many more positive instances of sentence-type weight value in best performing strategies. Positive weight corresponded to choosing assumptions over non-assumptions (eager selection). Thus, the result seem to indicate that **when faced with sentence choice, choose assumptions first**.

We regret not splitting the weight into proponent sentence-type choice and opponent sentence-type choice which would be more insightful. For proponent choosing assumptions over non-assumptions maximises work we do before branching since non-assumptions lead to branching as already demonstrated. However, for opponent the opposite is true - the opponent branches and backtracks to his assumption choice to decide whether to mark the assumption or not. On the other hand, by choosing assumptions the opponent expands culprits set, increasing the probability of backtracking through filtering. Hence there is no obvious explanation why choosing assumptions is a better choice for the opponent. We will investigate that in future experiments.

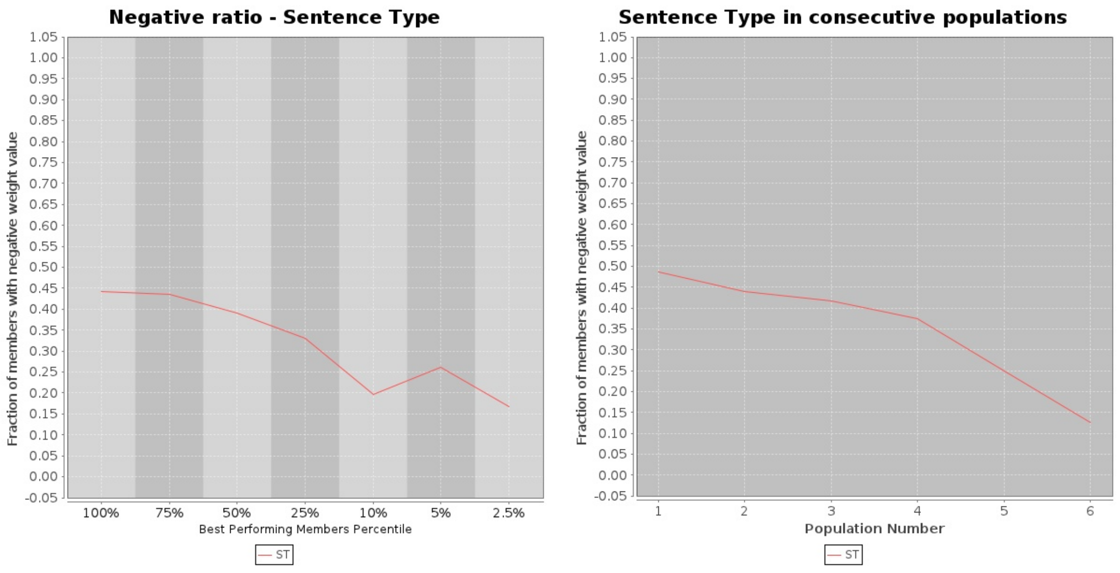
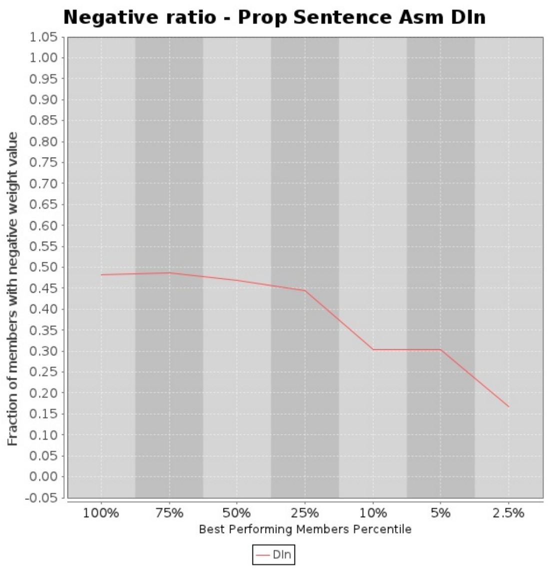


Figure 5.27: Fraction of negative Sentence Type weights decreases in best performing percentiles (left), it also decreases in consecutive populations (right)

Selecting high-degree assumptions for proponent sentence choice

While so far we have only discussed non-assumptions, the experiments also showed some trending between assumptions' measures and performance. It was far less significant than what we spotted for non-assumptions, but we present the results nevertheless (Figure 5.28). The figures indicate that the best performing strategies preferred choosing assumptions with high degree-in and degree-out for proponent, i.e. 'popular assumptions'. High degree-in indicates that the assumption occurs in bodies of many rules. We may prefer to choose such an assumption because it increases chances that we will use filtering to reduce the amount of work we have to do (filtering forces backtracking in some cases). We will explain that clearly and investigate assumptions and filtering in future experiments.



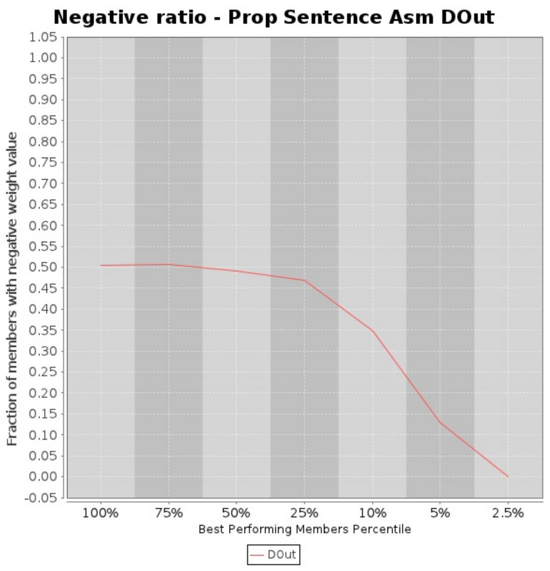


Figure 5.28: Fraction of negative weights increases in best performing population for proponent sentence choice. The weight presented are assumption degree-in (left) and assumption degree-out (right).

We remain uncertain about any trends here so we decided to ask 300 small frameworks and 100 medium frameworks about their point of view on the issue.

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Picks popular assumptions for proponent and unpopular for opponent	Positive assumption in-degree and out-degree for proponent choices, negative for opponent	14003	53260	10200	41222
Picks unpopular assumptions for proponent and popular for opponent	Negative assumption in-degree and out-degree for proponent, positive for opponent	11801	46523	8888	41099
Picks popular assumptions for both players	Positive assumption in-degree and out-degree for all choice points	13916	53403	10190	41189

Table 5.14: Different assumption strategies benchmarked on small and medium frameworks. The results reported are **average derivation times in milliseconds**.

Consider Table 5.14. 'Popular' assumptions refer to assumptions with high degree-in/degree-out measures. The last strategy listed is the one we learnt. However, it is not superior to any other strategies. The only thing suggested by Table 5.14 is the superiority of 'opponent moves first' strategies when contrasted with 'proponent moves first' strategies (again).

For now the most only true conclusion we can draw is that we do not have any major assumption trends. This may indicate that the current graph (ABA Graph) is not the best tool to reason about assumptions and their impact on derivation. We will check that explicitly in the next experiment. Moreover, in future experiments we will try out a different graph to see how assumption choices may influence derivation.

5.5.4 Next course of action

As demonstrated by ABA Generic, ABA Simple and ABA Simple Extended models, graph measures can be used to form derivation strategies with good performance. In next experiments we will be introducing measures which are not derived directly from ABA Graph. Nevertheless, the idea to introduce those measures came directly from analysing ABA graph measures and their correlation with performance. We will also try to experiment more with assumptions and filtering. We will look for measures defined on assumptions which may show some correlation with performance.

5.6 Experiments chronology

Before we move to the next experiment, we would like to give some background on what was the chronology of experiments' runs. Knowing the chronology will help to understand some (otherwise strange) design choices. In next two models, we validate our rule which says that we should not use more than 5 parameters per decision value equation. We also used PageRank measure although previous experiments indicated that PageRank is not useful for ABA derivation. This is because we do not present the experiments here chronologically, exactly as we performed them. The real chronological order was: we run ABA Simple Extended with PageRank first but we did not get satisfactory results (both in terms of derivation time speed-up and in terms of clear trends which could explain the speed-up). We moved on to ABA Lookahead and Assumption Lookahead models but we did not get satisfactory results either. We then decided that one of the reasons why the results were not good was that we were over-complicating the models. Thus, we decided to come back to ABA Simple Extended, remove PageRank measure and re-run the experiment. The results were much more clear and are presented in the previous section. However, due to time constraint, we did not simplify and rerun ABA Lookahead and Assumption Lookahead models. Instead, we moved on to the last model which we call Assumption Dynamic.

Thus the models presented below could probably be improved to reveal more interesting correlations. Nevertheless, we present them below to illustrate the ideas we had and some interesting output we obtained. They can also be considered as a negative result - we will, to some extend, show that certain ideas are not applicable to ABA Derivation speed-up. As Thomas A. Edison once said (some Internet sources also contributed the quote to Albert Einstein):

$$I\ have\ not\ failed,\ I\ have\ just\ found\ 10,000\ ways\ that\ don't\ work.$$

5.7 Fourth model - ABA Lookahead

5.7.1 Model justification

In previous experiments we mostly focused on non-assumptions and their degree-out. We argued that we should be choosing non-assumptions and arguments with small degree-out in order to fail and backtrack more quickly, reduce derivation branching and maximise the amount of work we do before reaching branching point (a point to which we will later have to backtrack). However, we did not employ filtering in any form. Since filtering occurs when a certain player uses an assumption which has already been used in derivation, we thought that in order to investigate filtering we have to focus our model on assumptions.

In our first 'assumption model' we still use ABA Graph. Almost all measures we use in the model are based on ABA Degree-In and Degree-Out centrality measured only for assumption nodes. We do not measure anything for non-assumption and rule nodes. Measures applied when choosing between non-assumptions and rules are derived from assumptions. To do that we look at assumptions to which choosing a given non-assumption or rule leads (i.e. assumptions further down in the derivation tree). We will explain that clearly in the next section.

There are two important factors related to assumptions which could potentially speed up the derivation process. The first one is filtering. In certain situations reusing assumptions which has already been used in the derivation leads to immediate derivation failure and backtracking. For instance, each time we perform filtering of defences by culprits and we find out that a given defence was already used by the opponent as a culprit, we will fail and backtrack. Consider a culprit set C = [a] and proponent choosing between two arguments:

```
arg1:  p0 :- [a,b,x0,y0]
arg2:  q0 :- [x0]

D: []
C: [a]
```

Although the second argument may have smaller Degree-Out Centrality (as computed in ABA Simple Extended model), suppose we choose the first argument and then select 'a' as a sentence. We then would apply filtering of defences by culprits, ending current derivation branch (failing) and causing backtracking. In terms of speeding up derivation process, choosing arg1 is much more optimal. We do not have an explicit parameter which says whether a given assumption is in defences or culprits set (we will add it later). Using ABA Graph, however, we can say something about the 'popularity' of certain assumptions. If an assumption has high Degree-In Centrality, then it occurs in many rules. Intuitively, assumptions with higher Degree-In have higher probability of being reused in the derivation. Thus, it may be a good idea to choose arguments with highest assumption Degree-In.

The second factor is related to assumption Degree-Out. In ABA graph, assumptions Degree-Out is either 0 or 1. 1 indicates that an assumption has a contrary and thus can be attacked. We may be interested in choosing assumptions which are not attacked by anything, as deriving an attack adds work for each player. Building on top of that, we may be interested in expanding arguments or non-assumptions which lead to assumptions which are not attacked. I.e. we would prefer arguments and non-assumptions which minimise the number of attacks targeted at them.

5.7.2 Model description

As we already mentioned, the model uses mostly assumption measures. This creates a problem - how do we define measures for non-assumptions, rules and arguments. We had a similar problem previously with arguments were we derived argument measures from its unmarked set members. We will do a similar thing here but we will focus only on assumptions. Lets illustrate the idea. Consider Figure 5.29 (left).

Suppose we want to define measures for a non-assumption x0 in terms of assumptions further down in the tree. The first step is to decide how far down the tree we will be looking. We can define a parameter to specify that. It will measure the maximum distance from the non-assumption we are considering to assumptions down in the graph (with distance here being the number of edges). Lets call that parameter **non-assumption lookahead**.

Suppose we set it to 5 here. We then look for all assumptions which are located at most 5 hops away from x0. We find 5 assumptions - a (distance = 2), b, c, d (distance = 4) and e (distance = 5). However, we will exclude e from the result as we had to cross an attack edge ($d -> n0$) to get to e. This means that e will belong to a different argument than x0. And so we do not want to include it. The general rule is **when using any lookahead mechanism to look further into the graph, never cross attack edges**.

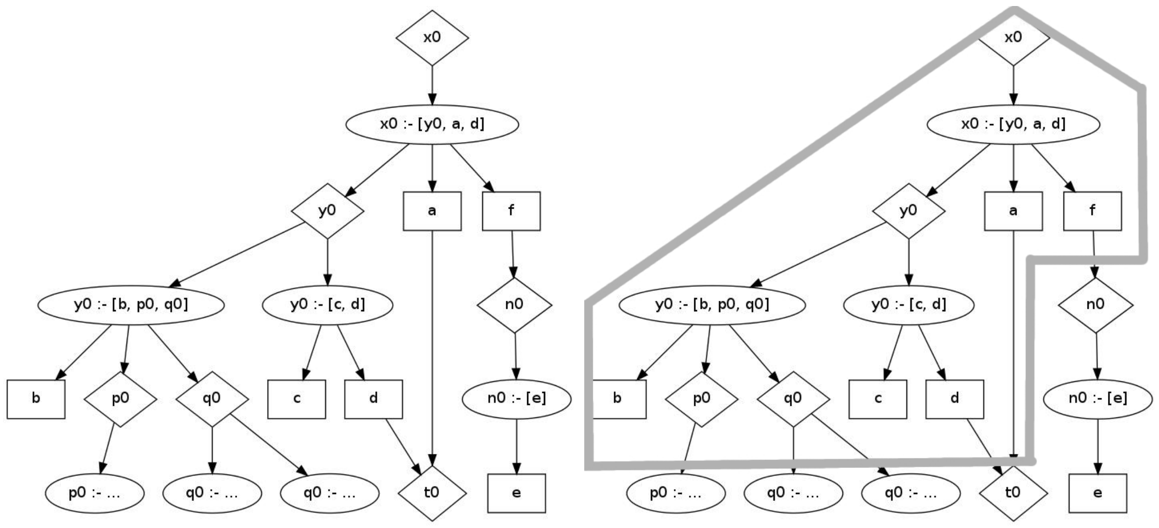


Figure 5.29: ABA Graph (left), x0 Lookahead-4 Subgraph (right)

A very important fact is that lookahead assumptions can always be precomputed. All we need is an ABA framework. We found that lookahead 10 is feasible to compute even for large synthetic frameworks. However, for real-life frameworks we have to stick with lookahead 5 at most as it becomes computationally infeasible beyond that.

So for x0 we got the following set of 'lookahead 5 assumptions': [a,b,c,d]. For each of them we have an easy access to all ABA Graph measures: degree-in, degree-out, PageRank. Thus we can define non-assumption's measures by summing its lookahead assumptions measures. For example, we would compute: $degree\_in(x0) = degree\_in(a) + degree\_in(b) + degree\_in(c) + degree\_in(d)$. We can proceed in exactly the same way to produce all rule graph measures in terms of rule's lookahead assumptions. We start at the rule node and look for assumptions at most **rule lookahead** from that rule node. We pre-compute that set and include it with the input framework. During the derivation we proceed in exactly the same way as for non-assumptions to compute rule measures.

Apart using ABA Graph assumption measures we also decided not to abandon the most important conclusions from previous experiment. However, instead of using non-assumption degree-out, we made the measure more explicit. We said that if we visualise an argument as a tree, degree-out approximates the size of the tree. We also learnt that in a lot of cases we want to choose arguments or non-assumptions which minimise degree-out. Since we are already analysing a part of ABA Graph in order to find lookahead assumptions, we could also compute the size of that subgraph and use it instead of degree-out.

For example, consider again Figure 5.29 (left). Count the number of nodes which are reachable from x0 in 4 or less hops. Remember not to cross attack edges. There are 13 nodes (including x0 non-assumption node) which are within distance 4 from x0. We marked them on Figure 5.29 (right). We will call that parameter **Subgraph Size**.

Deriving measures for assumptions is much more straightforward as we can read them directly from the graph. In order to derive measures for arguments we again consider their unmarked set. We then compute measures for assumptions and non-assumptions as we would for a sentence choice. Finally, we add them up to get measures per argument.

All in all, we use 29 parameters in the model. Table 5.15 summarises what measures where applied to each choice point. Sentence Type is the 29th parameter.

Choice point	Measures used
Proponent Sentence Assumption	Degree-In, Degree-Out, PageRank
Proponent Sentence Non-assumption	Degree-In, Degree-Out, PageRank, Subgraph Size
Opponent Sentence Assumption	Degree-In, Degree-Out, PageRank
Opponent Sentence Non-assumption	Degree-In, Degree-Out, PageRank, Subgraph Size
Proponent Rule	Degree-In, Degree-Out, PageRank, Subgraph Size
Proponent Argument	Unmarked Set Size, Degree-In, Degree-Out, PageRank, Subgraph Size
Opponent Argument	Unmarked Set Size, Degree-In, Degree-Out, PageRank, Subgraph Size

Table 5.15: Model measures

5.7.3 Output analysis and discussion

Maximising Assumption Degree-In

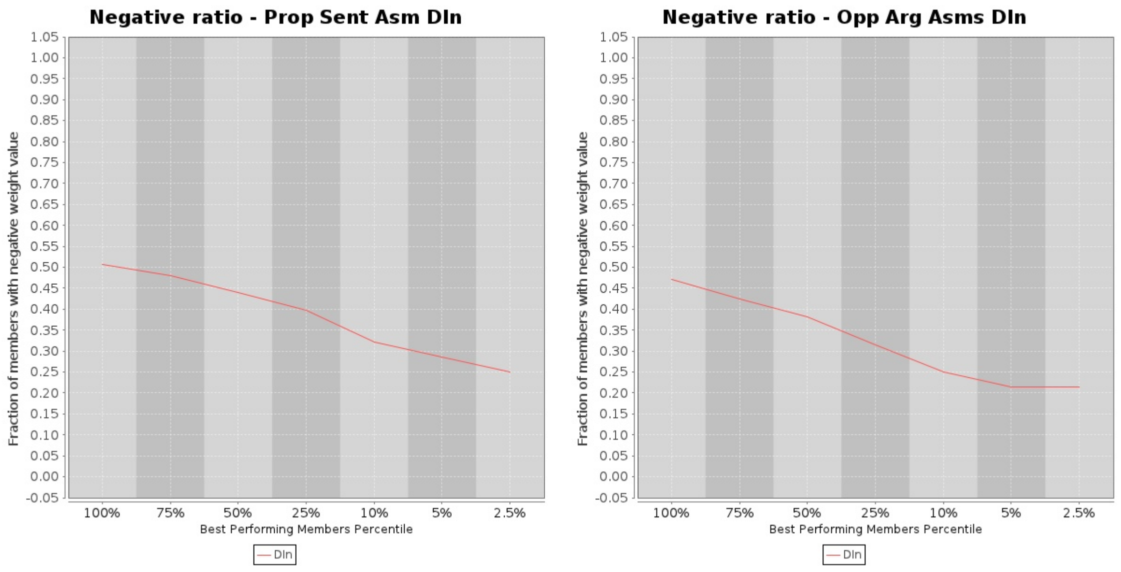


Figure 5.30: Fraction of negative Assumption Degree-In Centrality weights decreases in best performing percentiles for proponent sentence choice (left) and for opponent argument choice (right)

As Figure 5.30 illustrates, the ratio of negative assumption degree-in weights is decreases in best performing percentiles, which indicates that the best strategies prefer to choose assumptions with big degree-in. The trend was spotted for proponent sentence choice and for opponent argument choice. As we already explained, by choosing popular assumptions (assumptions with high degree-in) we increase the probability of exploiting filtering and forcing derivation to fail. However, the trend is very weak - the curve is slopping down very gently. Also, in both cases the negative weights ratio never goes below 0.2. As we have seen in the previous model, if we have a real correlation, we can usually spot it already for 0.25 or even 0.5 best performing percentile.

Minimising Assumption Degree-Out

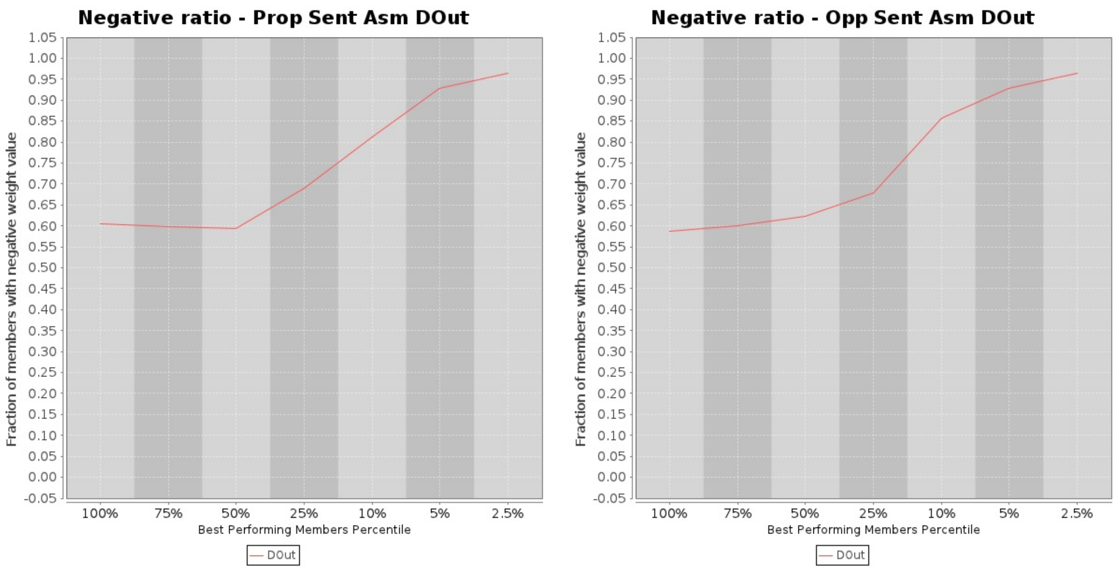


Figure 5.31: Fraction of negative Assumption Degree-Out Centrality weights increases in best performing percentiles for proponent sentence choice (left) and for opponent sentence choice (right)

We spotted a much stronger tendency for assumptions degree-out parameter (Figure 5.31). The tendency manifested for assumptions in proponent sentence choice and opponent sentence choice. The negative weights dominate in best performing percentiles which suggests a heuristic to minimise assumption degree-out when making choices in derivation. Since in ABA Graph assumptions may either have degree-out 1 or 0, depending on whether they are attacked or not, the output indicates that both players prefer to choose assumptions which are not attacked by anything.

Validating assumption trends

As usual, we have decided to validate what we have just learnt about choosing assumptions with big ANA Graph degree-in and small ABA Graph degree-out. Thus we set the parameters manually and run validation on a set of 300 small and 100 medium frameworks.

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Chooses most central nodes	Positive degree-in and degree-out in all choice points	19849	60000	16243	54109
Maximises degree-in, minimises degree-out	Positive degree-in and negative degree-out in all choice points	19849	60000	14252	51732
Minimises degree-in, maximises degree-out	Negative degree-in and positive degree-out in all choice points	19833	60000	15252	50433

Table 5.16: Different assumption heuristics benchmarked on small and medium frameworks. The results are report in milliseconds.

In short, Table 5.16 says that the heuristics we have learnt do not work. The strategy which we have learnt is the second strategy presented in Table 5.16. If we contrast it with its 'polar opposite' strategy (the third one) we see that they have a very similar performance. This is the only time that the output of validation directly contradicts the output of learning. We did not investigate this issue further because:

1. Due to the general lack of results, we decided to abandon ABA Lookahead model and introduce a radically new approach in order to find any useful assumption heuristics.
2. As we will see in the next subsection, the performance of ABA Lookahead model is determined by the subgraph size measure and so we focused on analysing it.

Subgraph parameter

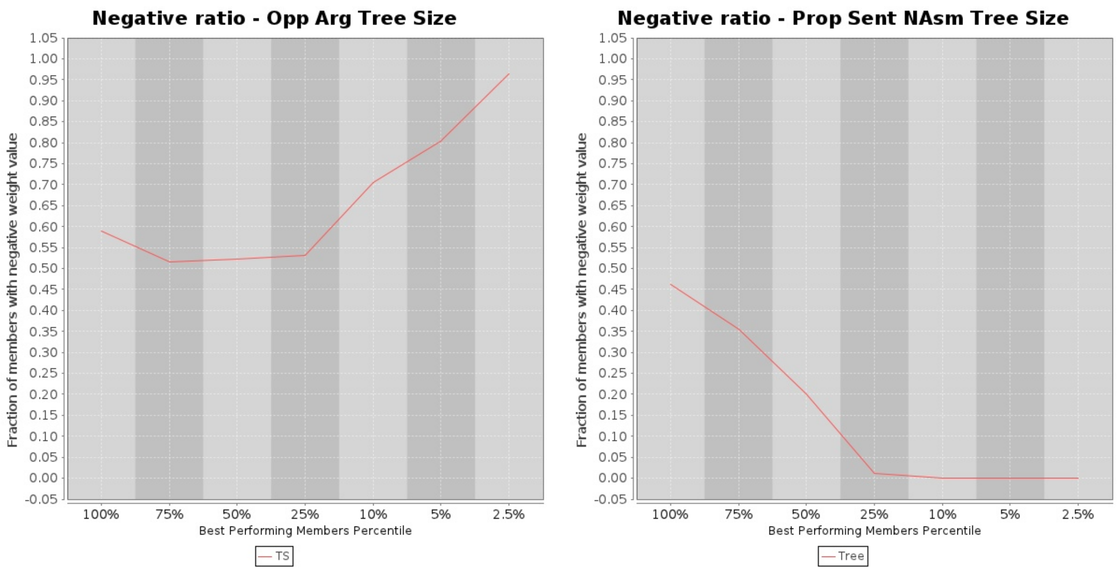


Figure 5.32: Fraction of negative Subgraph Size weights in best performing percentiles for opponent argument choice (left) and for proponent sentence choice (right)

Previous experiments strongly indicated that both players should prefer non-assumptions with negative ABA degree-out. We obtained that result regardless of which player was given a priority to move. One could expect that subgraph size parameter, which is a direct extension of ABA non-assumption degree-out, would also converge to negative values in ABA Lookahead model. It did so for opponent argument choice (Figure 5.32 left). However, surprisingly, an opposite trend was noted for proponent sentence non-assumption choice (Figure 5.32 right) which shows subgraph size negative weight ratio rapidly converging to 0. This indicates that a proponent should be choosing non-assumptions with largest subgraph size parameter which contradicts our discussion in ABA Simple Extended experiment. However, the correlation presented in Figure 5.32 (right) was observed **only in experiment run which gives move priority to proponent**. No such correlation was observed when the opponent was given a priority to move. To validate that, we manually set the weights and asked our test frameworks library. We present the results in Table 5.17. The first strategy minimises subgraph size for both - proponent and opponent. The second strategy maximises subgraph size for proponent non-assumption choice and minimise for opponent argument choice, i.e. the strategy plotted in Figure 5.32.

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Reduces branching by minimising subgraph size	Negative subgraph size in all choice points	11141	49777	5676	38943
Increases branching for proponent reduces for opponent	Positive subgraph size for all proponent's choice points, negative for opponent's	4772	34882	8628	44674

Table 5.17: Opposite subgraph size heuristics benchmarked on small and medium frameworks. The results are report in milliseconds.

As we can see, when proponent is given the priority to move, the the second strategy significantly outperforms the first one which is consistent with the output of learning from Figure 5.32. The same does not hold when we give the priority to opponent. We will further discuss this outcome in the last two experiments which show exactly the same result.

5.7.4 Conclusions and next course of action

We think it is useful to wrap up the discussion here and list the key outcomes of the experiment:

- Very few trends were found for ABA Lookahead model. Even when we did find trends, they were not as strong as in the previous model. For example, consider the percentile graph. When we discussed negative degree-out for ABA Simple Extended model (Figure 5.22) we saw that the fraction of negative weights was equal to 1 for 0.5 percentile already. On the other hand, consider Figure 5.31(left). For 0.5 percentile only around 60% of weights are negative. This is a much weaker trend.
- More importantly, we failed to validate assumptions degree-in and degree-out strategies which we learnt in the experiment.
- From observations above we can draw the following conclusion - **If there is any relationship between derivation performance and assumptions, using ABA Graph measures alone will not allow us to model that relationship**. Thus, we need to consider using other measures. This will be the focus of our last two models.

5.8 Fifth model - Ass Lookahead

5.8.1 Assumption Graph

A key conclusion from the previous experiment was that ABA Graph alone will not allow us to learn assumption-centred heuristics. One possible approach which we will investigate in this experiment involves defining a new graph which explicitly represents attacks between assumptions. We will call the new graph **Assumption Graph**. We will then define derivation strategies based on Assumption Graph measures just like we did with ABA Graph in previous experiments.

Assumption Graph has just one type of node - assumption node and one type of edge - attack edge. Usually, when we talk about assumptions attacking each other, we refer to sets of assumptions. Repeating the definition from Chapter 2:

Definition 9 *A set of assumptions A attacks another set of assumptions B if there is an argument supported by a subset of A which attacks an argument supported by a subset of B.*

We could also define direct attacks between assumptions as:

Definition 10 *Assumption a1 attacks assumption a2 iff a1 supports arg1, a2 supports arg2, arg1 attacks arg2.*

Generally, it is infeasible to build a graph which models all attacks between assumptions. It would require constructing all arguments, which is infeasible even for small frameworks. Hence we use an approximation. We will derive Assumption Graph straight from ABA Graph. The idea is similar to lookaheads from the previous model. We will be looking for assumptions which are lookahead distance away from each other in ABA Graph and which are separated by exactly one attack edge. Hence, we define an attack between assumptions to be:

Definition 11 *Assumption 'a' attacks assumption 'b' iff there exists a path from 'b' to 'a' in ABA Graph such that the path uses an attack edge exactly once and is shorter than Assumption Lookahead distance.*

By using **Assumption Lookahead** parameter we restrict the length of the path. Otherwise, it would be infeasible to generate Assumption Graph from large frameworks. Note that in ABA Graph an attack edge goes from an assumption being attacked to a contrary. In Assumption Graph, an attack edge will go from attacking assumption to assumption being attacked. Lets illustrate Assumption Graph derivation through example. Consider ABA Graph presented in Figure 5.33 (left). Suppose we choose Assumption Lookahead to be 5. The result Assumption Graph is presented in 5.33 (right).

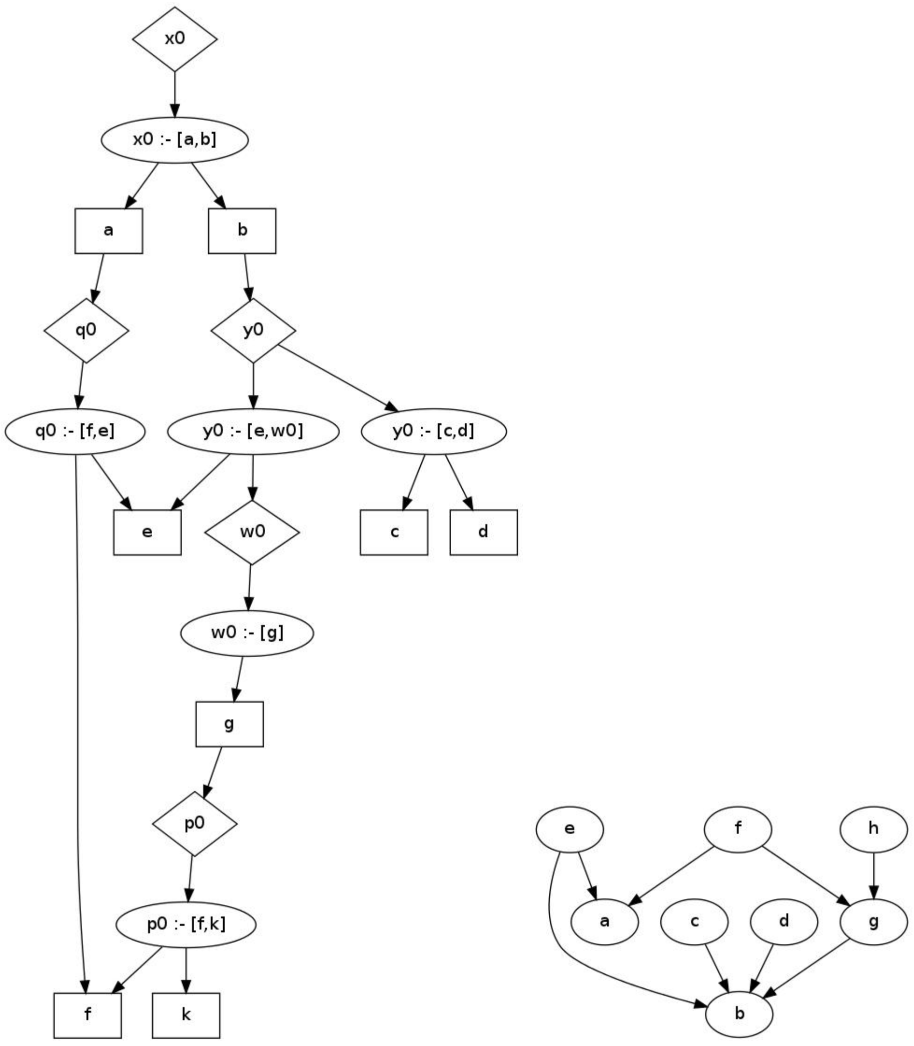


Figure 5.33: Input ABA Graph (left), derived Assumption Graph (right)

Before building the model we performed an analysis of Assumption Graph measures. As in the ABA Graph case, we wanted to see whether measures are correlated and whether we have any special characteristics of the graph which could affect learning process. A desirable property is that we no longer have three types of nodes and edges and we can treat everything uniformly. Also, recall that in ABA Graph assumption nodes had degree-out 0 or 1 and rule nodes had degree-in always equal to 1. We do not have these undesirable properties here.

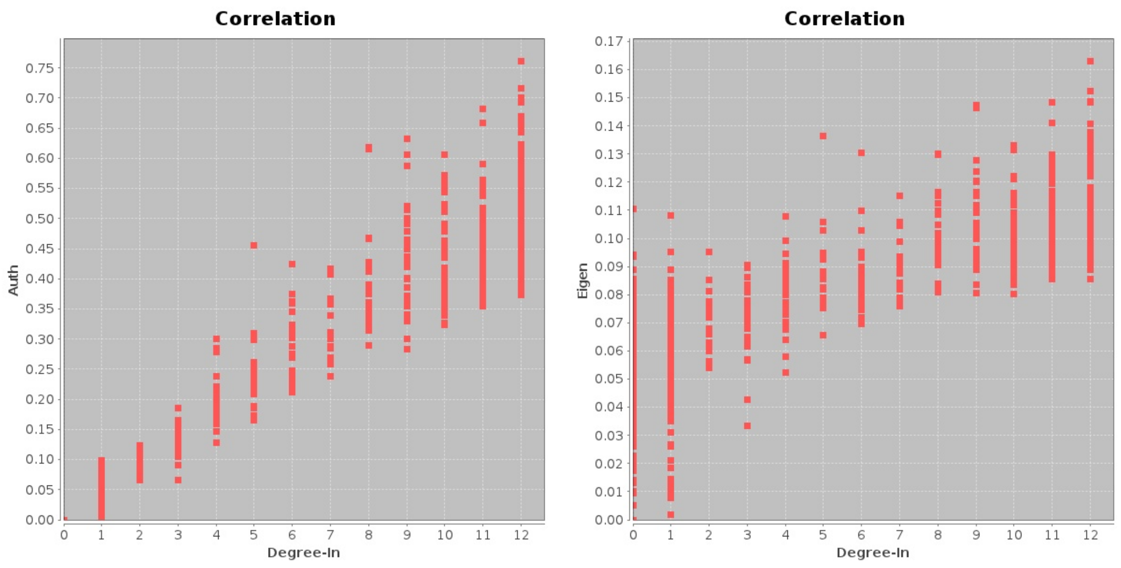


Figure 5.34: Correlation between degree-in and authority (left), and degree-in and eigen centrality (right)

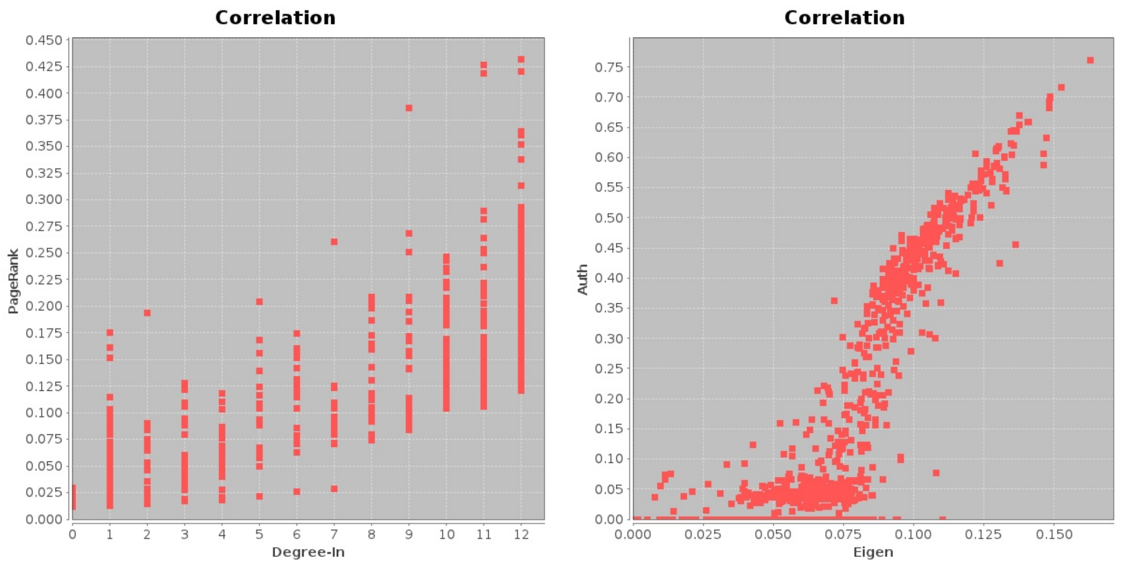


Figure 5.35: Correlation between degree-in and pagerank (left), and eigen centrality and authority (right)

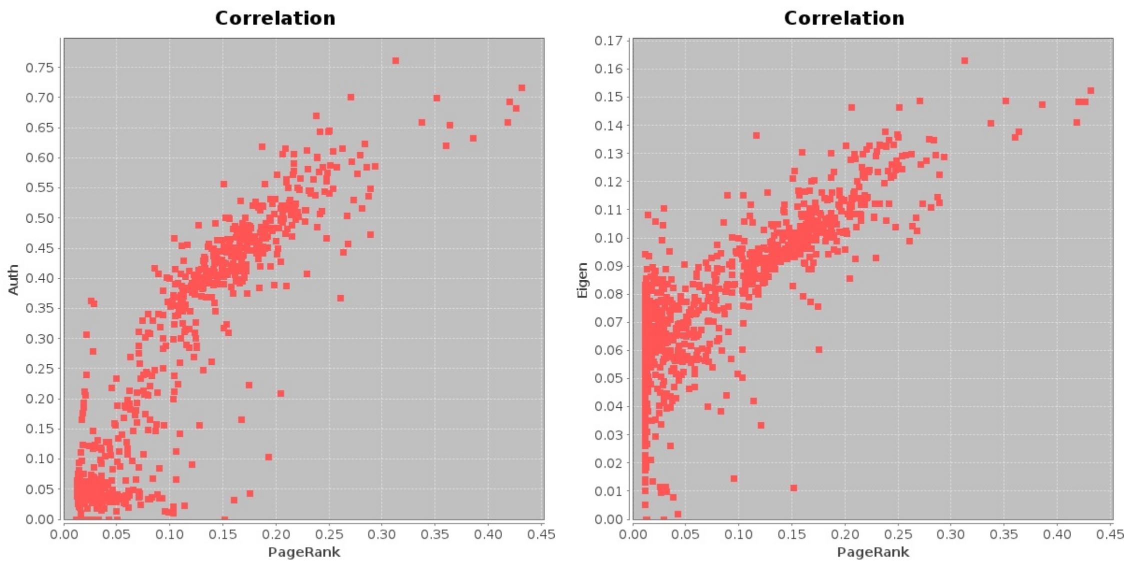


Figure 5.36: Correlation between pagerank and authority centrality (left) and eigen centrality (right)

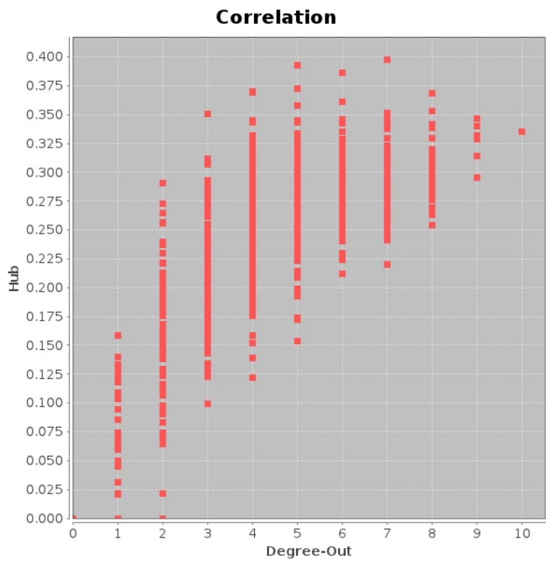


Figure 5.37: Correlation between degree-out and hub centrality

Correlations between measures are much weaker than in ABA Graph. We could say that there is a correlation between degree-in and authority centrality (Figure 5.34). But we could have a hard time justifying correlation between degree-in and eigen centrality (Figure 5.34 (right)) or degree-in and PageRank (Figure 5.35 (left)). Similarly, its hard to argue about correlation between degree-out and hub centralities (Figure 5.37). However, there is a correlation between eigen centrality, authority and PageRank centralities (Figures 5.35 (right) and 5.36).

In theory, authority, PageRank and eigen centralities are 'in-centrality' measures as we discussed in the Background Chapter. When we compute authority, PageRank or eigen centrality for a given node, we mainly consider incoming edges. This explains their relationship with degree-in centrality measure. Similarly, hub centrality is an 'out-centrality' measure. To compute it we mainly consider outgoing edges. Hence its relationship with degree-out.

Although the above is theory, the results show that the correlations are not so obvious. Hence, it could be worthwhile to investigate all possible pairs of in-centrality and out-centrality measures. However, due to the time constraint, in this experiment we only focused on the simplest pair - degree-in and degree-out centralities.

5.8.2 Model justification

The idea behind this model (as well as the previous one) is: **choosing the right assumptions may lead to faster derivation**. In the previous model we were not able to prove that statement. Thus, we radically changed the measures we use by introducing a new graph. We focus on two measures of Assumption Graph - degree-in and degree-out. Both have very intuitive explanation - for a given assumption 'a': degree-in measures how many other assumptions attack 'a', whereas degree-out measures how many assumptions 'a' attacks in turn. Our aim is to learn which type of assumptions should we prefer. We have four types:

- assumptions with high degree-out (aggressive assumptions) - they attack a lot of assumptions
- assumptions with low degree-out (peaceful assumptions) - they attack a very few or 0 assumptions
- assumptions with high degree-in (victimised assumptions) - they are attacked by a lot of other assumptions
- assumptions with low degree-in (untroubled assumptions) - they are not attacked by anyone or by a very few other assumptions.

Note that from now on each time we write about 'degree-in' or 'degree-out' we refer to measures from Assumption Graph, not ABA Graph. To avoid confusion, if we mean measures from ABA Graph, we will explicitly call them 'ABA Graph degree-in' etc.

Furthermore, we also did not give up on our favourite subgraph size measure. It will be interesting to see whether the current experiment agrees with the previous one where we had a hard time trying to justify what we learnt about subgraph size. We are also interested in discovering whether we can form strong enough heuristics based on assumption measures alone which could compete with degree-out / subgraph size heuristics.Finally, we introduced a new measure which counts the number of attacks we expose ourselves to when making a certain choice. We discussed it in the previous experiment and here we decided to make the idea explicit.

5.8.3 Model description

If we consider the implementation only, this model is very similar to the previous one. It also uses the notion of lookahead (hence its name) to precompute the set of assumptions for a given non-assumption or rule. It then computes non-assumption and rule measures by summing Assumption Graph measures computed for their corresponding lookahead assumptions. As we said, there are two measures of interest - degree-out and degree-in and both are computed for non-assumptions and rules. So the procedure is exactly the same as in the previous model. The only difference is that we use different graph. Similarly, we compute subgraph size measure for each non-assumption and rule in exactly the same way as we did in the previous model.

Number of attacks is the new measure so we will spend a bit more time explaining it here. As the name suggests, it measures how many attacks we may be exposed to if we choose certain non-assumption, rule or argument. For non-assumptions and rules we compute it by (again) looking at ABA framework subgraph which consists of ABA Graph nodes located within lookahead distance from the non-assumption/rule node. This time we count outgoing attack edges which are adjacent to the subgraph. It is worth explaining it by example. Consider Figure 5.38. The grey line marks the subgraph of ABA framework which is within 4 hops from x0. There are 3 outgoing attack edges adjacent to the subgraph which we mark by thick red line. Thus, our estimated number of attacks against x0 is 3. Note that this is only an estimation, as not all of the attacks may lead to a valid counter-argument.

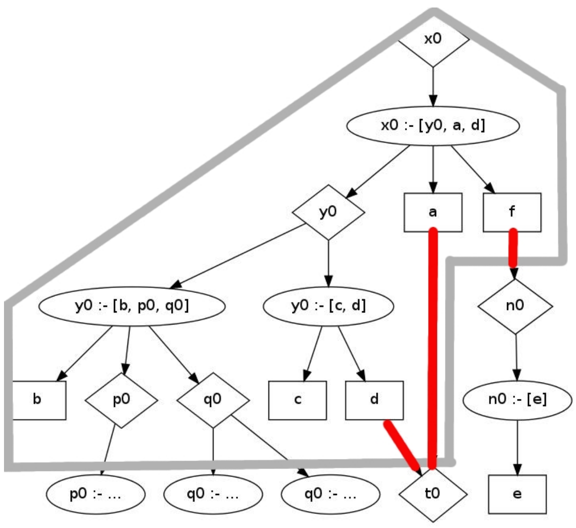


Figure 5.38: x0 lookahead-4 subgraph

Having define measures for non-assumptions and assumptions, we can define measures for arguments. Here we follow exactly the same logic as always - we consider argument's unmarked set, we compute measures for each member of the unmarked set (either assumptions or non-assumptions) and we sum them to get argument measures.

To summarise, we use the following measures for each of the choice point:

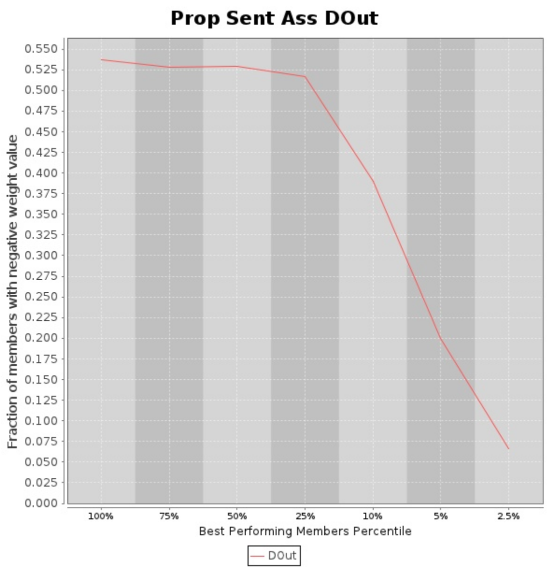
Choice point	**Measures used**
Proponent Sentence Assumption	Degree-In, Degree-Out
Proponent Sentence Non-assumption	Degree-In, Degree-Out, Number of Attacks, Subgraph Size
Opponent Sentence Assumption	Degree-In, Degree-Out
Opponent Sentence Non-assumption	Degree-In, Degree-Out, Number of Attacks, Subgraph Size
Proponent Rule	Degree-In, Degree-Out, Number of Attacks, Subgraph Size
Proponent Argument	Unmarked Set Size, Degree-In, Degree-Out, Number of Attacks, Subgraph Size
Opponent Argument	Unmarked Set Size, Degree-In, Degree-Out, Number of Attacks, Subgraph Size

Table 5.18: Model measures

5.8.4 Output analysis and discussion

Maximising assumption degree-out for proponent

Although the trend here is not crystal-clear, we do see a tendency for decreasing number of negative degree-out weights for proponent sentence assumption choice (Figure 5.39 left) and proponent argument choice (Figure 5.39 right). I.e. the number of positive degree-out weights dominates in the best performing percentiles of strategies. The majority of best performing strategies prefers choosing assumptions and arguments with bigger degree-out measures. Recall that assumption's degree-out measures the number of assumptions which the assumption is attacking. Hence, here we prefer choosing an 'aggressive' assumption - a one which attacks many other assumptions.



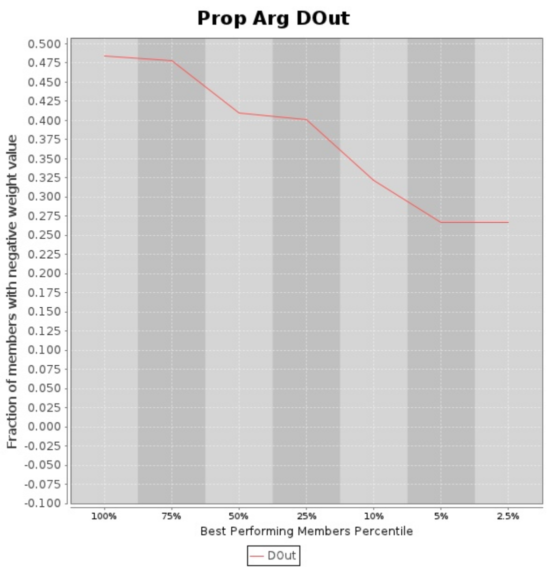


Figure 5.39: Fraction of degree-out negative weights decreases in best performing strategies for proponent sentence choice (left) and proponent argument choice (right).

Minimising assumption degree-out for opponent sentence choice

Figure 5.40 illustrates that in the majority of best performing strategies the opponent chooses non-assumptions which have small degree-out. To be more precise, he chooses non-assumptions which lead to assumptions with small degree-out (from the definition of non-assumption measures). Thus, the opponent seems to prefer 'peaceful' assumptions - the one which do not attack many other assumptions. This is the opposite strategy to what the proponent does.

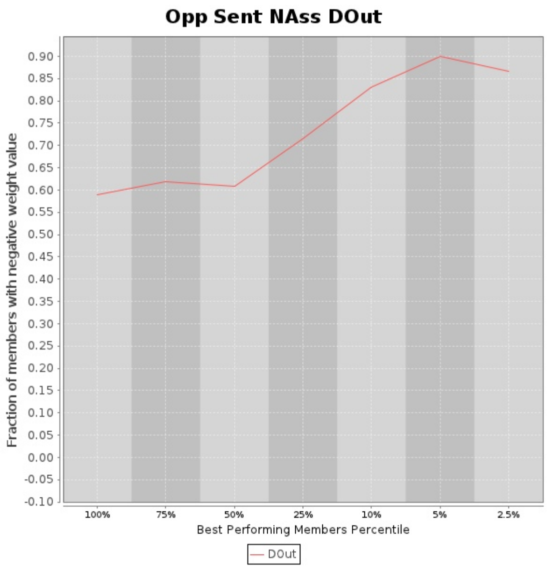


Figure 5.40: Fraction of degree-out negative weights increases in best performing strategies for opponent non-assumption sentence choice.

Validating learnt degree-out heuristics

As we have seen before, although Figures 5.39 and 5.40 show correlation between degree-out measures and performance, the correlation is weak and hence may be irrelevant to the actual derivation performance. We have to check that claim with our test framework set. Consider Table 5.19 where we show two derivation strategies. The first one corresponds to what we learnt (maximise degree-out for proponent choices, minimise degree-out for opponent choices), the second one is the polar opposite. As we can see, our learnt strategy significantly outperforms the second strategy. Unfortunately, it is far from being satisfactory in terms of performance. We were also unable to find any intuitive explanation of learnt heuristics.

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Chooses aggressive assumptions for proponent and peaceful for opponent	Positive degree-out in all proponent's choice points, negative degree-out in all opponent's choice points	8119	31929	9236	41477
Chooses peaceful assumptions for proponent and aggressive for opponent	Negative degree-out in all proponent's choice points, positive in all opponent's choice points	19915	60000	16504	52001

Table 5.19: Different degree-out strategies benchmarked on small and medium frameworks. The results are report in milliseconds.

Sentence choice - choosing assumptions over non-assumptions

Again, we obtained a strong tendency toward preferring assumptions over non-assumptions when choosing sentences for both players (Figure 5.41).

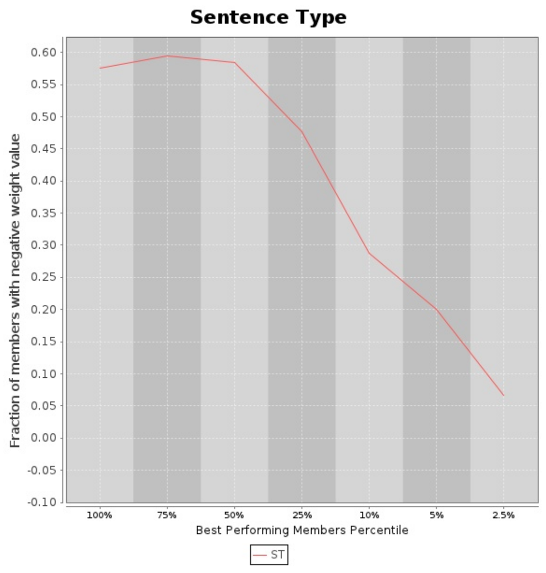


Figure 5.41: Fraction of sentence type negative weights decreases in best performing strategies.

Opponent argument choice - minimising assumption degree-in and minimising number of attacks

Figure 5.42 (left) shows the fraction of degree-in negative weights in opponent argument choice. As we see, the fraction is increasing. Thus, here the opponent seems to prefer assumptions with lower degree-in - assumptions which are not attacked by many other assumptions. He also seems to prefer expanding arguments which have lower number of potential counter-arguments (Figure 5.42). The correlations observed here are much stronger than in the previous case. We can observe that negative weight ratio reaches 0.8-0.9 in 0.25 percentile. This is even more explicit in degree-in case, where negative degree-in weight significantly dominates as early as in 0.5 percentile.

Although the trend is apparent, we will still need to validate it on our test frameworks. Consider Table 5.20. The first incorporates our learnt opponent argument degree-in heuristics - it tries to minimise it. Its opposite strategy is presented in the second row. The difference in performance in significant, especially when proponent is given priority to move.

Similarly, the third strategy minimises the number of attacks each player exposes himself to when choosing an argument, whereas the fourth strategy maximises that number. We note a similar difference in performance in favour of the heuristic which we learnt (third row).

It is also hard to intuitively explain heuristics proposed in this section. One thing to note here is that there are somehow very similar - when the opponent chooses arguments with assumptions which are attacked by fewer other assumptions, there exists a high chance that he implicitly chooses arguments which are attacked by few other arguments. Thus, the two measures he is minimising here approximate the same thing. One possible explanation could be that when expanding argument which has smaller chances of being counter-attacked, the opponent is getting closer to winning. Recall that when the opponent formulates a valid argument, the derivation immediately fails and backtracks. Thus, heuristics which 'support' the opponent in his struggle against the proponent may also lead to faster derivations.

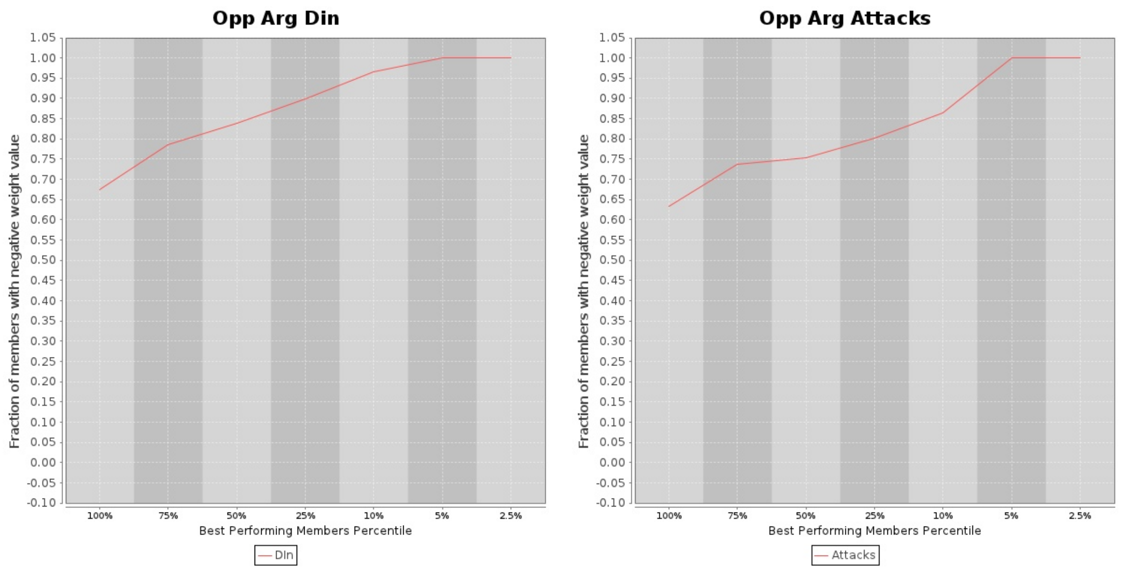


Figure 5.42: Fraction of degree-in negative weights decreases in best performing strategies for opponent argument choice. (left) Fraction of attack number negative weights decreases in best performing strategies for opponent argument choice. (right)

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Chooses victimised assumptions for proponent and untroubled assumptions for opponent	Positive proponent degree-in, negative opponent degree-in in all choice points	6282	31779	10123	47219
Chooses untroubled assumptions for proponent and victimised assumptions for opponent	Negative proponent degree-in, positive opponent degree-in in all choice points	19833	60000	13514	41561
Minimises number of attacks for both: proponent and opponent	Negative number of attacks for both players in all choice points	11140	40830	7678	42182
Maximises number of attacks for both players	Positive number of attacks for both players in all choice points	19823	60000	17158	52757

Table 5.20: Different degree-out strategies benchmarked on small and medium frameworks. The results are report in milliseconds.

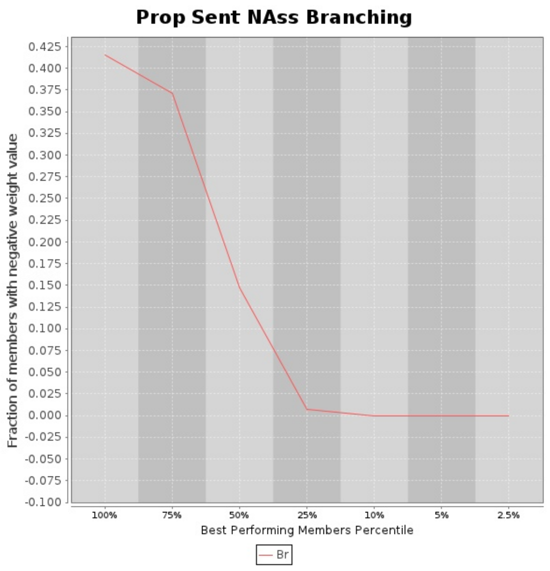
Minimising Subgraph Size for opponent but maximising for proponent

Consider Figure 5.43. The right figure shows the ratio of Subgraph Size negative weights for opponent non-assumption choice. It is increasing - the opponent prefers choosing non-assumptions which expand into smaller derivation trees. This is compatible with our extensive discussion in ABA Simple Extended model where we were always minimising ABA Graph degree-out centrality - a direct relative of subgraph size measure. In ABA Simple Extended model that fact was a decisive factor when it came to derivation performance.

Consider Figure 5.44 where we plot subgraph size negative weights fraction as measured for opponent argument choice. We have always argued that choosing 'small' arguments potentially leads to faster derivation failure and, as a result, speeds up the derivation. In Figure 5.44 we also see that negative weights dominate in best performing strategies. However, it is far from sharply rising figures in ABA Simple Extended model where half of the strategies had negative degree-out weight ratio equal to 1. Nevertheless, there is an evident trend toward negative subgraph size weights (a trend to minimise subgraph size) for opponent choices.

However, consider Figure 5.43 (left) where we show fraction of subgraph size negative weights for proponent non-assumption choice. It is decreasing which is a complete contrary to the results of ABA Simple Extended experiment. Proponent here prefers non-assumptions with bigger subgraph size which in ABA Simple Extended terms would roughly correspond to choosing non-assumption with higher ABA Graph degree-out. This trend was only observed for experiments where we gave proponent priority to move. It is consistent with the previous experiment.

We can manually set the subgraph size weight and use test frameworks to see whether we have just said does not fall apart. The results are presented in Table 5.21. They are identical to what we obtain in the previous experiment (Table 5.17). It should not be surprising, considering that we are using exactly the same measure in both experiments. We will discuss this result in the final experiment.



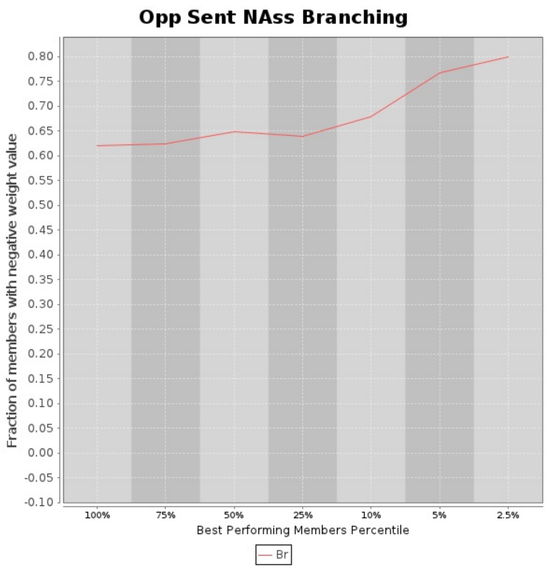


Figure 5.43: Fraction of Subgraph Size negative weights decreases in best performing strategies for proponent non-assumption choice (left) but decreases for opponent non-assumption choice (right).

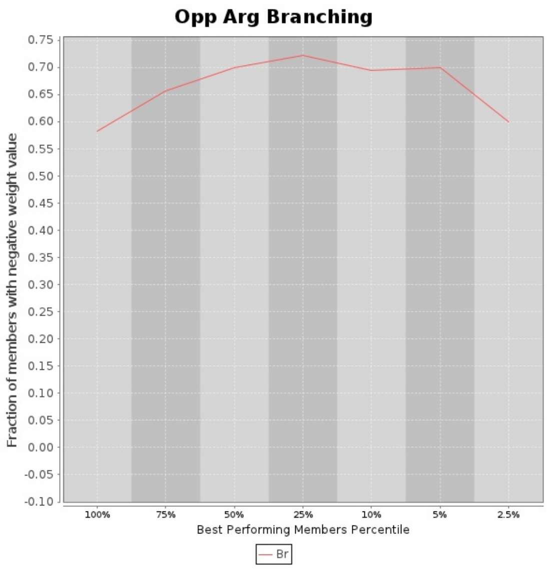


Figure 5.44: Fraction of Subgraph Size negative weights for opponent argument choice.

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Reduces branching by minimising subgraph size	Negative subgraph size in all choice points	11490	45999	5972	39454
Increases branching for proponent reduces for opponent	Positive subgraph size for all proponent's choice points, negative for opponent's	4935	35244	8908	45774

Table 5.21: Different subgraph size strategies benchmarked on small and medium frameworks. The results are report in milliseconds.

5.8.5 Conclusions and next course of action

Again, we think that it is useful to finish the discussion with a few key points learnt from the experiment:

- Although we found valid assumptions heuristics they still did not offer a derivation boost similar to degree-out/subgraph size heuristics. Compare best performing strategies from Table 5.21 with strategies from Tables 5.20 or 5.19. The former outperform the latter by a significant margin.
- We had problems with intuitively understanding some of the learnt trends. It was easy to find intuition and explain learnt trends on toy derivation examples for the first three models. However, here we struggled and could not relate learnt output to any simple intuitive explanation.

However, we still do not give up on the notion that assumptions play a key role in the derivation performance. What we realised here is that we approached the problem too indirectly. We defined a new graph and argued that certain properties of that graph correspond to certain properties of the derivation process. Most importantly, we argued that by choosing 'central' or 'important' assumptions (assumptions which have higher probability of repeating themselves in derivation process) we facilitate the filtering and speed derivation up. But if we really want to facilitate the filtering, why not model that directly as one of the parameters. Thus, in the last experiment we consider what we call 'defences/culprits membership' - we keep track of defences and culprits sets throughout derivation and compute intersections between defences/culprits and the set of assumptions we are currently considering. This allows us to encode strategies which can either choose assumptions which are in defences or culprits or stay away from selecting such assumptions.

5.9 Sixth model - Assumption Dynamic

5.9.1 Model justification

This is the last model we developed. We had two primary goals when developing the model:

- We wanted to have one last try on learning correlation between assumption selection and derivation. In the last two models we were unsuccessful. We decided here to make our approach much more direct. Since our main focus was on learning how assumption choice may help in filtering, we made the relationship between assumptions and filtering explicit by introducing new parameters to the model.
- If possible, we wanted to contrast two approaches we have investigated so far. We have just discussed the first one - choosing the right assumptions to drive derivation and facilitate the filtering. The second one was presented in ABA Simple Extended experiment and concentrates on reducing

derivation branching, killing branches as soon as possible (by choosing 'smaller' opponent arguments to hasten failures) and maximizing work before we branch. We also saw the evidence of the approach in the last two models were the efficient strategy for the opponent was often to minimise the Subgraph Size parameter. Hence, the model parameters here will be a mix of parameters related to the two approaches.

5.9.2 Model description

As we already mentioned, we have radically changed parameters used in the model. We have added a number of parameters which are explicitly related to assumptions.

For non-assumption and rule nodes we have added two parameters:

- the number of assumptions belonging to non-assumption's/rule's Lookahead Subgraph
- the size of intersection between the union of defences and culprit and the set of assumptions in non-assumption's/rule's Lookahead Subgraph

Recall the definition of Lookahead Subgraph. It is a subgraph of ABA Graph which consists of all nodes located within lookahead distance of a given non-assumption or rule node. To illustrate how we compute the new parameters, consider our running example (Figure 5.45). The subgraph is marked by grey contours. We have also marked all assumption nodes in red. To compute the first new measure, we count the assumptions belonging to the Lookahead Subgraph. In this case we have 5. To compute the second measure, we need to look at defences and culprits as well. Suppose we have the following:

- Defences: [a,h]
- Culprits: [c,g]

To compute the second measure we do the union of defences and culprits - [a,c,g,h] and the compute the intersection between [a,c,g,h] and our Lookahead Subgraph assumptions - [a,b,c,d,f]. We get [a,c] and hence the measure is 2.

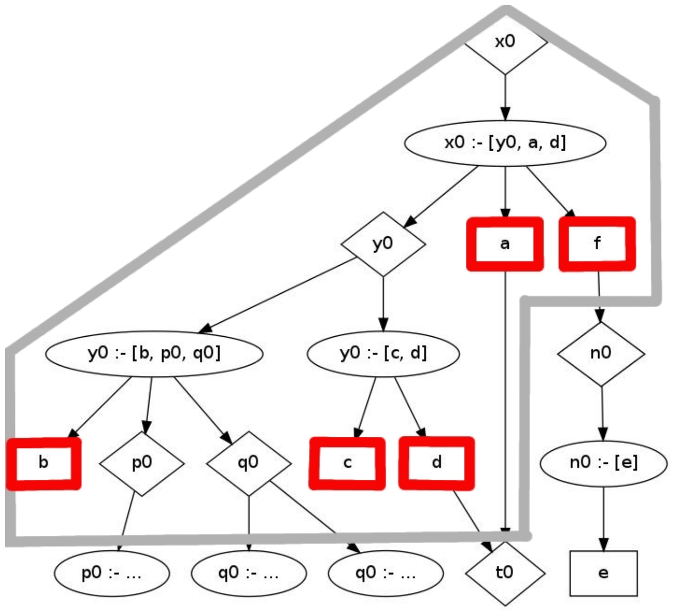


Figure 5.45: x0 lookahead-4 subgraph.

Hence, two new parameters measure:

- How many assumptions a certain choice leads to. If our goal is to expand defences or culprits, we would probably prefer choices with bigger number of assumptions in Lookahead Graph. If we want to avoid immediate attacks, we would probably prefer choices with the measure being smaller.
- How many of these assumptions are in defences/culprits. Again, if we want to force the filtering, we would prefer to make choices with this measure being bigger. If we want to avoid filtering, we would choose the opposite.

For non-assumption and rule choice we also have Subgraph Size measure which is computed in exactly the same way as in previous models.

Arguments have exactly the same measures as non-assumptions and rules. Argument measures are again derived from members of argument's unmarked set. Yet again, consider an example. Suppose we are computing measures for argument arg1 with unmarked set size: [m0, n0, a, b]. Suppose we know the following about non-assumptions:

NAsm	Size of Lookahead Subgraph	Assumptions in Subgraph
m0	14	[a,e,f,g]
n0	28	[c]

Suppose that the union of defences and culprits U = [a,b,d]. To compute argument's number of assumptions we compute the size of the union between assumptions in argument's unmarked set and assumptions in m0's and n0's subgraph. We get [a,b,c,e,f,g] and hence the measures is 6. To compute the number of assumptions in D or C we take the intersection of argument's assumptions and U. We get [a,b] and hence the measure is 2. To compute argument's subgraph size we add its non-assumptions subgraph sizes. We get 42. We also add the number of assumption in arguments unmarked set which is 2 (in case the argument does not have any non-assumptions in unmarked set). The final measure is 44.

We have also defined three new parameter for assumption choice:

- Binary parameter similar to Sentence Type parameter - decides whether we should give priority to assumptions which are in Defences or Culprits or not. We will again use the following rule: if the parameter is $>= 0$ we always give priority to assumptions which are in Defences or Culprits (if there are any). If the parameter is < 0 we will give priority to assumptions which are not in Defences or Culprits.
- Number of defence and culprits in contrary's subgraph. Since we do not have access to a full argument supporting the contrary, we will estimate that measure by considering assumptions from contrary's Lookahead Subgraph
- Graph Size of the contrary of assumption.

Lets consider an example. Suppose we are a proponent and we are choosing between three assumptions: [a,b,c]. Lets consider the following parameters:

w1	w2	w3	D	C
1	-1	0	[a]	[c]

Where w1, w2 and w3 are three assumption choice parameter weights (for parameters as defined above respectively). Since the first assumption parameter is set to +1, we will be considering only assumptions which are either in D or C. We will be choosing between [a, c]. Suppose the contrary of a is x0 and c does not have a contrary. Thus, the two other measures for c will be 0 (in fact, it will be 0.00001 as discussed in Chapter 3). We need to compute measures for a. Consider again figure 5.45. The number of assumptions in Defences or Culprits which are in the subgraph of x0 is 2 (a and c). The size of the subgraph for x0 is 12. Hence we have

- measures for a: [2, 12]
- measures for c: [0.00001, 0.00001]

Since the second weight is negative and the third one is 0, we choose c.

To summarise, the following measures are used for each choice point:

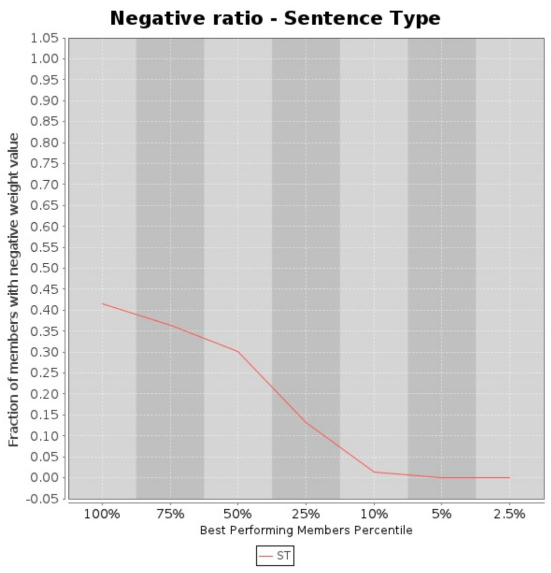
Choice point	**Measures used**
Proponent Sentence Assumption	D/C membership (binary), Number of D/C assumptions in contrary's lookahead subgraph, Size of contrary's lookahead graph
Proponent Sentence Non-assumption	Number of assumptions in subgraph, Number of D/C assumptions in subgraph, Size of subgraph
Opponent Sentence Assumption	D/C membership (binary), Number of D/C assumptions in contrary's lookahead subgraph, Size of contrary's lookahead graph
Opponent Sentence Non-assumption	Number of assumptions in subgraph, Number of D/C assumptions in subgraph, Size of subgraph
Proponent Rule	Number of assumptions in subgraph, Number of D/C assumptions in subgraph, Size of subgraph
Proponent Argument	Number of assumptions in subgraph, Number of D/C assumptions in subgraph, Size of subgraph
Opponent Argument	Number of assumptions in subgraph, Number of D/C assumptions in subgraph, Size of subgraph

Table 5.22: Model measures

5.9.3 Output analysis and discussion

No correlation between filtering measures and performance

This is the most crucial outcome of the experiment. The last three models were driven with a single idea in mind - assumptions do matter in derivation performance. However, the success was very minor (if any). One of the reason which lead us to believe in assumptions was that in each model the sentence type weight was always converging toward choosing assumptions over non-assumptions. We got the same results even here. (Figure 5.46 left)



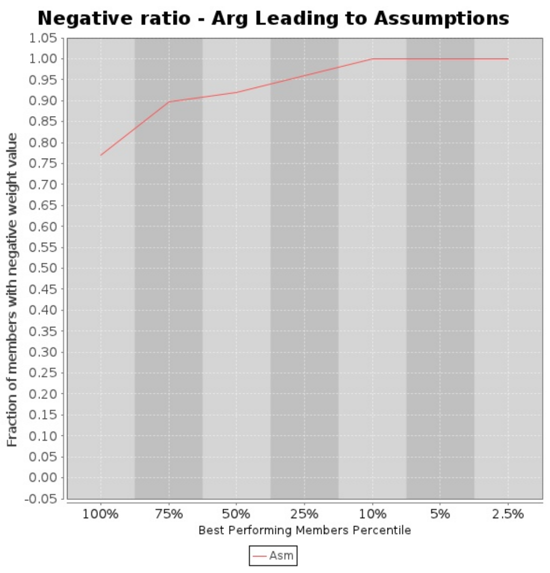


Figure 5.46: Positive weights dominate for sentence type (left), negative values dominate in 'number of assumptions' weight for opponent argument choice (right)

Another was the analysis of the derivation algorithm and the observation that by applying the filtering of defences by culprits, we may immediately end derivation if we happen to try to reuse a culprit as a defence. Hence, we should be facilitating that fact and driving our derivation so that if the failure happens anyway, it should happen as soon as possible.

The above is all true, but either it is not sufficient to create a heuristic that could significantly improve performance or we did not consider the right approach to do so. In any case, the current model did not show any correlation between any of assumption measures and performance. The only exception was opponent argument choice when we had a significant tendency toward choosing argument with smaller number of assumptions (Figure 5.46 (right)).

However, we noticed exactly the same trend for subgraph size measure as in two previous experiments (we shall discuss it in the next section). Thus, the main conclusion here is that parameters which explicitly measure derivation branching and speed-up backtracking by choosing empty arguments are by far more important to the derivation speed that any other measures we tried. Consider Table 5.23. The four strategies encode four different approaches to using defences and culprits during the derivation. Looking at the first two strategies we may be tempted to say that, generally speaking, its better to move away from defences and culprits (i.e. choose arguments which do not have them).

We may try to fit intuition behind that choice - e.g. 'both players prefer to expand their defences/culprits hence they choose argument in which those do not occur'. Even better, looking at Table 5.23 again, we may also say that the third strategy is outperforming the others, so the proponent should be choosing derivation paths leading to defences or culprits whereas the opponent should be doing the opposite. Intuitively, it makes sense as then the opponent would be expanding culprits set and the proponent would be actively looking to facilitate that expansion by choosing defences which belong to culprits and thus failing (and speeding derivation).

Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Moves toward D/C	Pos D/C membership and number of D/C in subgraph weights	19544	60000	17566	54923
Moves away from D/C	Neg D/C membership and number of D/C in subgraph weights	11526	56408	7703	36410
Proponent goes towards D/C, opponent goes away	Pos D/C weights for proponent, neg for opponent	8706	59336	10563	43965
Proponent goes away from D/C, opponent goes towards	Neg D/C weights for proponent, pos for opponent	19775	60000	13844	45231

Table 5.23: Different defences and culprits strategies benchmarked on small and medium frameworks. The results are report in milliseconds.

Although the above may be true, we propose another explanation of numbers in Table 5.23. Consider Table 5.24 and consider its similarity with Table 5.23. The key is that two types of measures are correlated themselves. If we choose arguments with smaller number of defences and culprits, we are likely choosing a smaller argument. And vice-versa, if we are choosing a smaller argument (i.e. smaller subgraph size) we are likely choosing an argument with smaller number of defences and culprits. Thus, the first strategy in Table 5.24 corresponds to the second strategy of Table 5.23. Similarly, if we the proponent is choosing argument with bigger number of defences and culprits, he is probably choosing arguments with bigger subgraph size. Hence, the thirds strategy in Table 5.23 corresponds to the second strategy of Table 5.24. Our claim is that the performance of strategies in Table 5.23 is mostly driven by their correlation with branching measures, not by the fact that they are following efficient 'assumption strategy' (i.e. they are choosing the right assumptions).

Thus we claim that measures in Table 5.24 are the key factors to derivation performance. Our final proof is the third strategy in Table 5.24 where we mixed branching heuristics with defences / culprits heuristics from Table 5.23 . Contrast it with the second row in 5.24, which is exactly same strategy but without d/c heuristics mixed it. Performance-wise, adding d/c heuristics did not improve anything.Quite the contrary, the first-row strategy has slightly better performance than the third-row strategy.

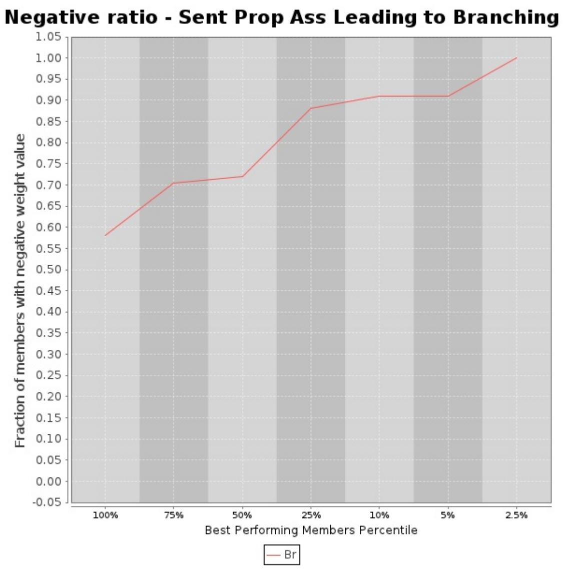
Description		Proponent Priority		Opponent Priority	
Strategy description	Weights Used	Small	Medium	Small	Medium
Reduces branching by minimising subgraph size	Neg subgraph size in all choice points	10589	58043	4492	28341
Increases branching for proponent reduces for opponent	Pos subgraph size for all prop's choice points, neg for opp's	4077	28564	9309	47564
Increases branching for proponent, reduces for opponent, proponent goes towards D/C, opponent goes away from D/C	Negative subgraph size (-1) for opponent, positive (+1) for proponent, positive D/C measures for proponent (+0.25), negative for opponent (-0.25)	5269	42208	×	×

Table 5.24: Different defences and culprits strategies benchmarked on small and medium frameworks. The results are report in milliseconds.

Subgraph size parameter analysis

Like the two previous model, this model also shows the following trend:

- If we give move priority to opponent, we will observe domination of negative weights in all kind of branching measures (Figures 5.47 and 5.48)
- If we give move priority to proponent, we will observer domination of negative weights in all branching measures (Figure 5.49) except proponent non-assumption choice (Figure 5.50). This is also validated by Table 5.24.



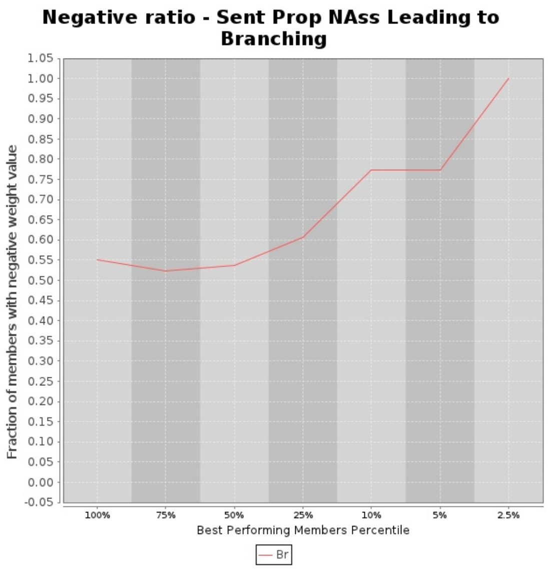
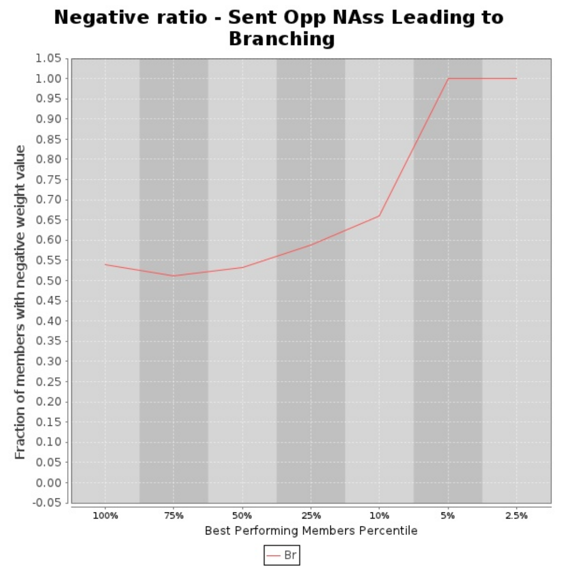


Figure 5.47: Proponent assumption choice - choosing assumptions which lead to smaller counter-arguments (left), proponent non-assumption choice - choosing non-assumptions with smaller subgraph size (right)



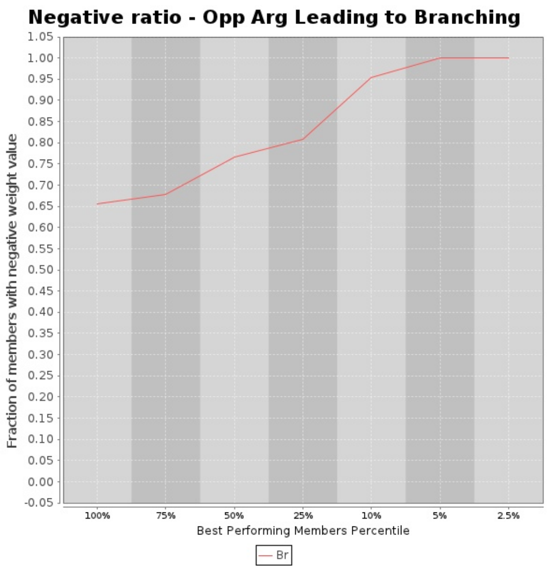
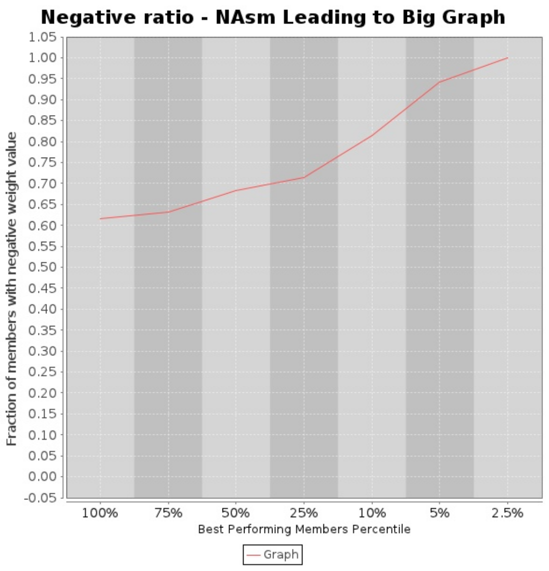


Figure 5.48: Opponent non-assumption choice - choosing non-assumptions with smallest subgraph size (left), opponent argument choice - choosing argument with smallest subgraph size (right)



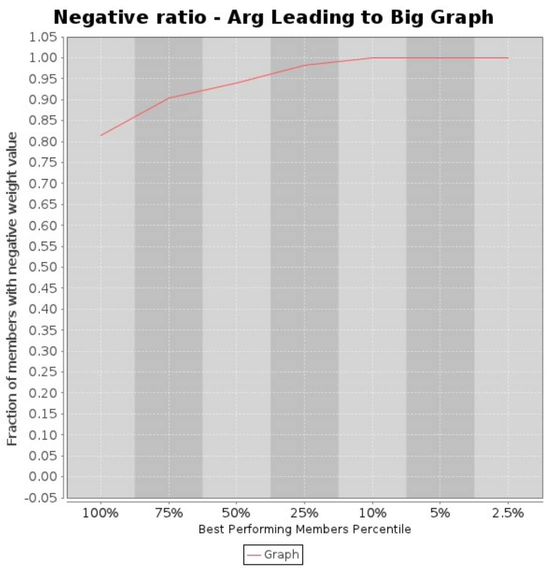


Figure 5.49: Opponent sentence choice - minimising non-assumption subgraph size (left), opponent argument choice - minimising argument subgraph size (right)

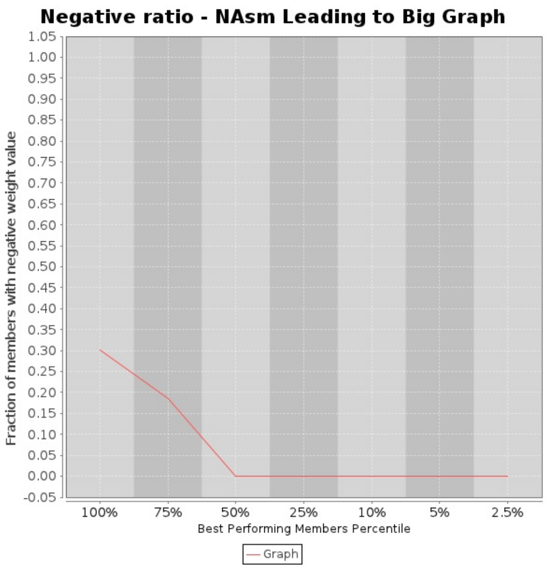


Figure 5.50: Proponent sentence choice - maximising non-assumption subgraph-size

If the proponent is given the priority to move, he will expand all of his non-assumptions anyway, before the opponent makes any move. Hence, non-assumption choice parameters determine in what order the proponent expand his non-assumptions. This may be crucial to derivation time as each non-assumption choice with non-assumption which is headed in more than one rule leads to backtracking.

Thus, through non-assumption choice, the proponent actually chooses a backtracking strategy. Suppose we can choose between two non-assumptions: x0 and y0. x0 heads three rules whereas y0 heads two. Its very likely that x0 will have a higher subgraph size than y0. Depending on the order in which we choose the two, the derivation may branch in two different ways (Figure 5.51). Since the proponent prefers large subgraph size alternatives, he will follow the first branching strategy. We are not certain what the intuition is behind this choice.

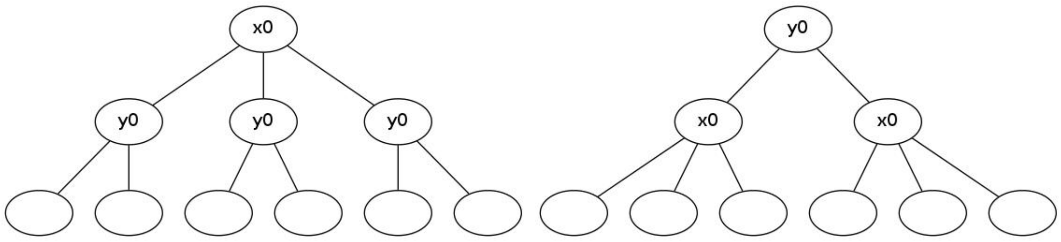


Figure 5.51: Two backtracking strategy: choosing x0 (high degree-out / high subgraph size node) first (left) vs choosing y0 (low degree-out / low subgraph size node) first (right)

5.10 Common trends

Finally let us close our discussion with a short summary of repeating trends, which we observed in most of the experiments:

1. **Sentence type choice: prefer assumptions over non-assumptions**. We obtained the same result in all experiments where we had an explicit measure controlling which type of sentences we should consider first.
2. **Opponent argument choice: choose empty/smaller arguments**. This was measured by ABA Graph degree-out and unmarked set size in the first three experiments and by subgraph size in later experiments. In all of them the corresponding weight values were mostly negative for all best performing strategies.
3. **Proponent non-assumption choice: maximise branching when proponent given priority to move**. If the proponent has priority to move, he will expand all his non-assumptions anyway. The experiments indicate that for performance gains he should expand non-assumptions which lead to highest dispute branching first.
4. **Proponent non-assumption choice: maximise branching when opponent given priority to move**. However, when opponent is given the priority to move, experiments with ABA Simple Extended and Assumption Dynamic models indicate that the proponent should be choosing non-assumptions which lead to smallest dispute branching. This is intuitive - the more we branch, the longer it takes derivation to finish.
5. **Assumption-based measures we considered do not influence derivation**. We have tried various possibilities to measure how smart assumption choice may influence the derivation performance. Our approaches included: ABA Graph based measures, Assumption (attack) Graph based measured, explicit filtering measures. None of the approaches worked.
6. **Rule choice: no heuristics found**. This should be expected. Since we are looking for all solutions, the rule choice does not matter as we will backtrack to that choice anyway and explore all possibilities. It probably would have been significant if we were instead looking just for a single solution and ending derivation after it has been found.

Chapter 6

Validation

6.1 Models evaluation

The results presented here were benchmarked on 300 small, 100 medium, 10 large and 30 medical (framework,query) pairs. The full output of each benchmarked strategy is available in Appendix A. For each model we selected a group of interesting strategies which highlight the trends we learnt in the experiments.

We use two types of 'default strategies' here:

- default strategy - a strategy which is a default implementation in proxdd (prolog implementation of the derivation algorithm)
- generation strategy - a strategy with which the (framework, query) pair was generated; recall that in order to ensure that (framework, query) pairs are none-trivial and solvable at the same time after we generate a framework we test it on the query using some default strategy - the generation strategy. We only accept it for learning and benchmarking if it finds all solution to the input query within a preselected time interval.

The default strategy applies the following heuristics:

- always give priority to proponent to move
- choose argument with smallest unmarked set size (for both, proponent and opponent)
- choose non-assumptions over assumptions
- choose rule with the smallest body

Therefore the generation strategy which we present here is really a set of derivation strategies, one for each benchmarked (framework, query) pair. Each of them is guaranteed to solve its corresponding framework within some time interval (1-20 seconds for small frameworks, 1-60 seconds for medium and 1-120 seconds for large). Thus, they represent a set of 'good' or 'correct' derivation strategies for a given (framework, query). Note that for medical frameworks there is no generation strategy.

6.1.1 Models comparison

We begin our analysis by comparing the performance of each model assessed based on their best performing derivation strategy. We were unable to collect measurement from large frameworks for ABA Generic model due to a memory overflow issue which we could not solve. The results are presented in Table 6.1.

Model Name	**Small**	**Medium**	**Large**
ABA Generic	7619	28473	×
ABA Simple	8394	37100	46684
ABA Simple Extended	4010	26504	36991
ABA Lookahead	4935	35244	58301
Assumption Lookahead	4851	27658	59277
Assumption Dynamic	3229	25963	47274
Default Strategy	8140	40640	69448
Generation Strategy	10632	24221	33937

Table 6.1: Best performing strategies from each model benchmarked on small, medium and large frameworks. The results are reported in **milliseconds**.

We see that the difference in performance between simple models (the first two) and the rest is significant for small networks. We also see that there are two models slightly outperforming the rest: ABA Simple Extended and Assumption Dynamic. This is not a coincidence - the majority of interesting strategies presented below was constructed using these two models. Furthermore, we see that four of of our models beat the default derivation strategy. The biggest gain is for small frameworks derivation times where we observe approximately 2 times better performance for our proposed models. The difference decreases for medium and large frameworks, but our models still manage to outperform the default strategy by approximately 1.5.

When we compare our heuristics with generation strategy, we observe significant gains in performance for small networks. The generation strategy slightly outperforms our models for medium and large networks. However, the best strategies of ABA Simple Extended and Assumption Dynamic models manage to keep up with it even for large frameworks. The generation strategy is really a set of 'good' strategies for solving a given (framework, query) pair. Taking that fact into account, the performance of our models is a very positive result.

6.1.2 Heuristics validation

We compare here the best performing and most interesting strategies from all models. For ABA Simple Extended and Assumption Dynamic models we also include single best performing strategy found by the genetic algorithm (i.e. the best performing population member in all populations every recorded), because these two models showed the best performance during training. We first show the performance of the heuristics on the synthetic frameworks and contrast it with the default heuristic. Then we repeat the process for medical frameworks.

Best performing strategies

Number	Model	Player choice	Short description	Parameters used
1	ABA Simple Extended	opponent	Reduces branching by minimising degree-out	Negative non-assumption out-degree (-1) and negative unmarked set size (-0.25) for all choice points
2	ABA Simple Extended	proponent	Best performing training strategy - minimises branching	Negative non-asm deg-out, positive asm in-deg and out-deg
3	ABA Simple Extended	opponent	Best performing training strategy - minimises branching	Negative non-asm deg-out, positive asm in-deg and out-deg
4	ABA Simple Extended	proponent	Minimises branching and picks most popular assumptions	Negative non-asm deg-out (-1), positive asm in-deg and out-deg (+0.5)
5	ABA Simple Extended	opponent	Minimises branching and picks most popular assumptions	Negative non-asm deg-out (-1), positive asm in-deg and out-deg (+0.5)
6	Assumption Lookahead	proponent	Considers all observed trends	Neg subgraph size for opp, pos for prop (+1), pos prop deg-in (0.25), neg opp deg-in (-0.75), pos prop deg-out, neg opp (0.5), neg number of attacks for both (-0.25)

Table 6.2: Best performing and most interesting strategies

Strategy	**Model**	**Player choice**	**Short description**	**Parameters used**
7	Assumption Lookahead	opponent	Considers all observed trends - opponent priority	Neg subgraph size (-1) and num of attacks (-0.5), pos deg-in and deg-out for prop (0.25), neg for opp (-0.25)
8	Assumption Dynamic	opponent	Proponent reduces branching, goes away from D/C, maximising assumptions, opponent reduces branching, goes toward D/C, minimising assumptions	Neg subgraph size for all choice points, negative D/C weights for proponent, positive D/C weights for opponent, positive number of assumptions for proponent, negative for opponents
9	Assumption Dynamic	proponent	Increases branching for proponent, reduces for opponent, proponent goes towards D/C, opponent goes away from D/C	Negative subgraph size (-1) for opponent, positive (+1) for proponent, positive D/C measures for proponent (+0.25), negative for opponent (-0.25)
10	Assumption Dynamic	opponent	Reduces branching for both players, both players move away from D/C and minimise assumptions	Negative subgraph size (-1) for both players in all choice points, negative D/C weights and ’number of assumptions’ weights (-0.25) for both players
11	Assumption Dynamic	proponent	Best training strategy learnt - proponent priority to move	Neg subgraph size for opp, pos for prop, neg D/C weights
12	Assumption Dynamic	opponent	Best performing strategy learnt - opponent priority to move	Neg subgraph size for prop and opp, pos D/C weights

Table 6.3: Best performing and most interesting strategies

One common thing for all strategies which is not mentioned in the table is that they all prioritise assumptions over non-assumptions.

Performance on synthetic frameworks

Strategy	Small	Medium	Large
1	4260	26289	66668
2	4550	28317	51317
3	4610	25458	80000
4	7314	38774	89366
5	4010	26504	36991
6	4851	27658	59277
7	7888	42611	120000
8	4748	25799	57168
9	5269	42208	106667
10	4750	28777	63492
11	3229	25963	47274
12	5629	32834	69191
Default	8140	40640	69448
Generation	10632	24221	33937

Table 6.4: Strategies performance on synthetic small, medium and large frameworks. The results are reported in milliseconds.

We illustrate the difference in performance for the 12 strategies we selected in Table 6.4. The majority of them outperform the default strategy. Strategies 5 and 10 (which we have already seen in Table 6.1) manage to keep up with the generation strategy.

Performance on medical frameworks

Strategy	Derivation time in milliseconds
1	116001
2	40006
3	116000
4	40005
5	116010
6	40014
7	104012
8	108012
9	40023
10	108004
11	40012
12	104008
Default	56002

Table 6.5: Strategies performance on real-life medical frameworks. The results are reported in milliseconds.

The performance of our strategies on real life frameworks is significantly different than on synthetic frameworks. Approximately half of the strategies took 40 seconds on average to solve the input query. But the other half took 100 seconds. Interestingly, the best performing strategies on synthetic frameworks - strategies 5 and 10 are members of the latter group. If we take a closer look at Tables 6.2 and 6.3 we notice that the best performing group (strategies 2,4,6,9 and 11) all have one feature in common - the proponent is given the priority to move. The other half of the strategies all give priority to the opponent.

However, if we only consider the better performing group, we can see that there is an improvement over the default strategy.

Another interesting fact to note is that they all have very similar performance. The only possible explanation (which does not involve analysing the input frameworks) is that all strategies were built around our most important heuristics which we learnt in the previous chapter. They all prioritise assumptions over non-assumptions and they all minimise subgraph size / degree-out, the only exception being proponent non-assumption choice where subgraph size is maximised. In order to understand the output here a bit better we would have to analyse the medical frameworks and compare their characteristics with what we learnt about synthetic frameworks. Also, a small amount of frameworks we used here (30) may be another factor leading to the extraordinary uniformity of results.

6.2 Validation summary

To conclude, we have demonstrated that the strategies we propose outperform the default strategy by a large margin. They also significantly outperform the generation strategy on small networks and tie with it on medium and large frameworks. Since the generation strategy is really a set of different strategies, each one considered as a 'good' or 'correct' strategy to solve a given (framework, query) pair, the results show that the strategies we learnt are efficient and generic - they perform well on different types of frameworks.

Chapter 7

Conclusions

The aim of our work was to analyse the main issue with ABA structured X-dispute derivation - its time efficiency. We wanted to get further insights on what influences the derivation performance and how to improve it. We made the analysis from the algorithmic point of we, we did not consider factors such as hardware or cache. We also wanted the discussion and the outcomes to be generic. Thus, we used Machine Learning approach applied to a set of artificially generated frameworks. To do that we first had to define a model which encodes an arbitrary measure-based derivation strategy. Hence, we were able to apply a genetic algorithm in order to search the space of all possible derivation strategies and identify the best performing ones.

The genetic algorithm served primary as a sampling tool. It allowed us to generate a large sample of efficient derivation strategies. We could then analyse that sample to spot any correlations between measures we used to define the derivation strategy and performance. Mathematical properties of our strategy encoding schema allowed us to perform such analysis.

We were successful in identifying a couple of trends which influence the performance of ABA derivations. First of all we analysed measures obtained from ABA framework graph representation - ABA Graph. We used measures which are well-established in Graph Theory such as PageRank or degree centrality. We identified a strong tendency between degree-out centrality and derivation time. We also learnt that the majority of ABA Graph centrality measures did not show any correlation with performance. Hence we did not include those measures in our further discussion.

We experimented a lot with degree-out centrality measure in order to explain its impact on derivation time. We discovered what is really intuitive - degree-out measure controls derivation branching and hence should be minimised. Also, by picking arguments with small degree-out for opponent we hasten inevitable derivation failures which also speeds up the process.

Having explored the standard centrality measures, we moved on to defining our own - some based on ABA Graph and some on our own ideas. We further validates our notion with degree-out by modelling it more explicitly (through a subgraph size measure). A lot of our ideas were driven by a belief that assumptions play a key role in speeding up the derivation process due to filtering. We thought that we could develop an efficient derivation strategy based on assumption measures. Unfortunately, this proved to be a dead-end.

Most of the time after learning about certain trends we also immediately tried to validate those trends by constructing a derivation heuristic which reflects them and benchmarking that heuristic on a set of synthetic frameworks. In the final part of the project, we identified a set of efficient heuristics which we learnt through experimentation and we validated them on synthetic frameworks and real-life medical data.

We compared their performance with current default derivation strategy and we noticed 1.5 - 2 times improvement in the synthetic framework case. Even more importantly, most of our strategies outperformed or had a very similar performance to what we called a 'generation strategy' - a set of strategies, one per framework, used to generate their corresponding frameworks. Since each of generation strategies is tailored for a single framework and is guaranteed to solve the framework within some small time-bound, by beating them or tying with them we demonstrated that the derivation strategies we learnt are efficient and generic - they work well for different types of frameworks.

7.1 Future work

The most natural extension to the project would be to implement an efficient derivation strategy basing on our analysis presented in this report. The heuristics which we proposed are only parametrisation of a generic model. Hence, they may be less efficient than a specific implementation. We could then benchmark the implemented strategy against medical frameworks and see whether we have any significant speed improvement.

Another possible extensions involve further work with the models we defined to understand some of the learnt trends a bit better. First of all, we never gave a clear and intuitive explanation to our sentence choice strategy which prioritises assumptions over non-assumptions. We did not have enough data to infer anything more here. As a next step we could define a new model where we have that parameter split into two - one for proponent sentence choice and one for opponent sentence choice. Such a strategy worked in degree-out centrality case where it allowed us to understand the learning output.

Secondly, we never explained why the proponent should be choosing non-assumptions which lead to bigger derivation branching when he is given priority to move. We also never defined any useful heuristics for turn choice. In most of our experiments we always gave priority to one of the players. A natural extension of our models would be to define player choice heuristics in terms of argument measures available for each player.

Another possible next step is to reverse the problem and analyse the queries in terms of performance. For instance, we would be interested in knowing what types of queries give our strategies the hardest time. We could analyse them by measuring query-node parameters from ABA Graph. Such analysis may give ideas leading to better search heuristics.

Furthermore, we used admissible semantics in all of our experiments and validation. It would be useful to repeat the whole process with grounded semantics and compare the difference between trends learnt for admissible derivations.

Finally, we did not attempt to analyse the medical frameworks which may explain the output we obtained in Validation Chapter. It may also lead to another heuristics ideas. It may be useful to include medical frameworks in the training process as well.

Appendix A

Glossary

The definitions here will be largely informal. Formal definitions are given in the appropriate sections of the report. This is for quick reference only if the reader is lost in the amount of jargon used in the report.

ABA Graph: a graph representation of ABA Framework

ABA Framework: a set of assumptions, non-assumptions, inference rules and contraries which form an ABA representation of some knowledge base

backtracking: taking a different path in derivation (e.g. choosing different rule) after finding solution or failing

branching: generally refers to the fact that we may have a several different derivation paths which we may take and to which we have to later backtrack

body: usually a body of a rule, i.e. each rule is of the form: head :- body

choice point: a point in the derivation algorithm where there is more than one option on what to do next - for example we have more than one arguments and we have to select one

claim: usually an input query to ABA derivation procedure

conclusion: usually a head of a rule

culprits: assumptions used by the opponent and attacked by the proponent

decision function: a function of measurements which determines which option we should choose from a set of available alternatives at each choice point

decision value: a value computed by decision function

defences: assumptions used by the proponent and attacked by the opponent

derivation: see dispute derivation

derivation strategy: a strategy used to prove input query, specifies which player we choose, which sentences/rules/arguments proponent chooses and which sentences/arguments opponent chooses; in genetic algorithm context, a strategy is encoded using real numbers vector and is the same as a population member

derivation time: time elapsed between starting a derivation and getting all solutions for an input query

dispute derivation: a process of finding all solutions which support an acceptability of input query, i.e. deriving whether input query is acceptable and if it is finding all ways to prove it

eager selection: choose assumptions over non-assumptions if any available

filtering: a mechanism used in dispute derivation to ensure that it is acceptable and it terminates (for admissible semantics) by keeping track of used assumptions and making sure they are not used again

fitness: in our case an average derivation time of a population member, measured on 50-100 (framework, query) pairs

framework: see ABA Framework

(framework, query) pair: when an ABA Framework is synthetically generated we always test if its solvable within some specified time interval for some query, we then always use only that query on the framework in learning or benchmarking as we are guaranteed it is not trivial and it is solvable

goal: see query

head: usually a head of the, i.e. each rule is of the form: head :- body

lookahead (parameter): number of edges we travel from some given node to construct its lookahead graph;

lookahead subgraph (of a node): a part of ABA Graph located within lookahead distance from the given node

measure: usually refers to one of the measures used to select an option when the dispute reaches a choice point with many options (e.g. PageRank);

model: a set of measures and decision functions which specify how options are selected at choice points

model parameter: usually a weight associated with some measure, for example 'proponent argument choice degree-in weight';

option: usually refers to possible alternative we have when we reach a choice point; for example we have to choose between 3 arguments, each argument is refereed to as option

parameter: see model parameter

patient selection: choose non-assumptions over assumptions if any available

population: a set of strategies used in genetic algorithm, usually each strategy is a vector of real numbers (of weights)

population member: a derivation strategy encoded as a vector of real numbers used in genetic algorithm

potential argument: argument not yet fully expanded

premise: a body of a rule

query: an input sentence to a ABA derivation procedure which we want to prove

sentence: either an assumption or a non-assumption

strategy: see derivation strategy

unmarked set (of an argument): a set of assumptions and non-assumptions which we yet have to expand to get a full argument

weight: a random number between -1 and 1 which assesses the importance of a measure in decision function, each strategy / population member is a vector of weights

Appendix B

Full validation results

Before we present all strategies we have validated on test framework suit and the full output we have obtained. Unfortunately, the output was not available for some large frameworks due to memory-overflow issues which we were unable to solve.

ABA Simple

Strategy description	Weights Used	Small	Medium	Large
Low degree	Degree -1	8394	37100	46684
Low pagerank	PageRank -1	11585	44699	86692
Low unmarked set size	Unmarked Set Size -1	13717	49982	92184
High degree	Degree +1	19973	60000	120000
High pagerank	PageRank +1	19956	60000	120000
High unmarked set size	Unmarked Set Size +1	20000	60000	120000
Low degree, low pagerank	Degree -1, PageRank -0.5	15317	43329	66252
High degree, high pagerank	Degree +1, PageRank +0.5	19960	60000	120000
Best performing strategy learnt	All measures negative	15215	44008	73824

Table B.1: Different ABA Simple strategies benchmarked on small, medium and large frameworks. The results are report in milliseconds.

ABA Generic

Strategy description	Weights Used	Small	Medium
Low degree	Respective measure -1, all other measures 0	7619.1761904762	28473
Low closeness	Respective measure -1, all other measures 0	9493.6571428572	35571
Low betweenness	Respective measure -1, all other measures 0	10162.44	37723
Low eigen	Respective measure -1, all other measures 0	10217.9230769231	36351
Low pagerank	Respective measure -1, all other measures 0	12078.2	41091
Low hub	Respective measure -1, all other measures 0	14795.4555555556	48987
Low authority	Respective measure -1, all other measures 0	12455.1555555556	40147
High degree	Respective measure +1, all other measures 0	20000	37077
High closeness	Respective measure +1, all other measures 0	20000	38989
High betweenness	Respective measure +1, all other measures 0	20000	60000
High eigen	Respective measure +1, all other measures 0	20000	60000
High pagerank	Respective measure +1, all other measures 0	20000	60000
High hub	Respective measure +1, all other measures 0	17692.05	42178
High authority	Respective measure +1, all other measures 0	19921.7	40498
Best performing strategy learnt	Negative degree, eigen, pagerank and betweenness	13120.65	32123

Table B.2: Different ABA Generic strategies benchmarked on small, medium and large frameworks. The results are report in milliseconds.

ABA Simple Extended

Description		Proponent Priority			Opponent Priority		
Strategy description	Weights Used	Small	Medium	Large	Small	Medium	Large
Reduces branching by minimising degree-out	Negative non-assumption out-degree (-1) and negative unmarked set size (-0.25) for all choice points	7729	37828	94013	4260	26289	66668
Increases branching by maximising degree-out	Positive non-assumption out-degree for all choice points	19918	60000	120000	17366	54000	120000
Increases branching for proponent and reduces branching for opponent	Positive non-assumption out-degree for all proponent choices, negative for all opponent choices	10520	49020	106667	7074	34880	93353
Picks popular assumptions for proponent and unpopular for opponent	Positive assumption in-degree and out-degree for proponent choices, negative for opponent	14003	53260	120000	10200	41222	66683
Picks unpopular assumptions for proponent and popular for opponent	Negative assumption in-degree and out-degree for proponent, positive for opponent	11801	46523	106674	8888	41099	106666
Picks popular assumptions for both players	Positive assumption in-degree and out-degree for all choice points	13916	53403	x	10190	41189	x
Best performing learnt strategy: minimises branching	Mostly negative non-asm deg-out, positive asm in-deg and out-deg	4550	28317	51317	4610	25458	80000
Best strategy made explicit: minimises branching and picks most popular assumptions	Negative non-asm deg-out (-1), positive asm in-deg and out-deg (+0.5)	7314	38774	89366	4010	26504	36991

Table B.3: Different ABA Simple Extended strategies benchmarked on small, medium and large frameworks. The results reported are **average derivation times in milliseconds**.

ABA Lookahead

Description		Proponent Priority			Opponent Priority		
Strategy description	Weights Used	Small	Medium	Large	Small	Medium	Large
Chooses most central nodes	Positive degree-in and degree-out in all choice points	19849	60000	×	16243	54109	108018
Maximises degree-in, minimises degree-out	Positive degree-in and negative degree-out in all choice points	19849	60000	×	14252	51732	53340
Minimises degree-in, maximises degree-out	Negative degree-in and positive degree-out in all choice points	19833	60000	×	15252	50433	120000
Reduces branching by minimising subgraph size	Negative subgraph size in all choice points	11141	49777	×	5676	38943	80001
Increases branching for proponent reduces for opponent	Positive subgraph size for all proponent's choice points, negative for opponent's	4772	34882	×	8628	44674	106666

Table B.4: Different ABA Lookahead strategies benchmarked on small, medium and large frameworks. The results are report in milliseconds.

Assumption Lookahead

Description		Proponent Priority			Opponent Priority		
Strategy description	Weights Used	Small	Medium	Large	Small	Medium	Large
Considers all observed trends - proponent priority	Neg subgraph size for opp, pos for prop (+1), pos prop deg-in (0.25), neg opp deg-in (0.75), pos prop deg-out, neg opp (0.5), neg number of attacks for both (-0.25)	4851	27658	59277	×	×	×
Considers all observed trends - opponent priority	Neg subgraph size (-1) and num of attacks (-0.5), pos deg-in and deg-out for prop (0.25), neg for opp (-0.25)	×	×	×	7888	42611	120000

Table B.5: Different Assumption Lookahead strategies benchmarked on small, medium and large frameworks. The results are report in milliseconds.

Assumption Lookahead

Description		Proponent Priority			Opponent Priority		
Strategy description	Weights Used	Small	Medium	Large	Small	Medium	Large
Chooses aggressive assumptions for proponent and peaceful for opponent	Positive degree-out in all proponent's choice points, negative degree-out in all opponent's choice points	8119	31929	×	9236	41477	×
Chooses peaceful assumptions for proponent and aggressive for opponent	Negative degree-out in all proponent's choice points, positive in all opponent's choice points	19915	60000	×	16504	52001	×
Chooses victimised assumptions for proponent and untroubled assumptions for opponent	Positive proponent degree-in, negative opponent degree-in in all choice points	6282	31779	×	10123	47219	106666
Chooses untroubled assumptions for proponent and victimised assumptions for opponent	Negative proponent degree-in, positive opponent degree-in in all choice points	19833	60000	×	13514	41561	80000
Minimises number of attacks for both: proponent and opponent	Negative number of attacks for both players in all choice points	11140	40830	×	7678	42182	×
Maximises number of attacks for both players	Positive number of attacks for both players in all choice points	19823	60000	×	17158	52757	×
Reduces branching by minimising subgraph size	Negative subgraph size in all choice points	11490	45999	×	5972	39454	×
Increases branching for proponent reduces for opponent	Positive subgraph size for all proponent's choice points, negative for opponent's	4935	35244	58301	8908	45774	×

Table B.6: Different Assumption Lookahead strategies benchmarked on small, medium and large frameworks. The results are report in milliseconds.

Assumption Dynamics

Description		Proponent Priority			Opponent Priority		
Strategy description	Weights Used	Small	Medium	Large	Small	Medium	Large
Reduces branching by minimising subgraph size	Neg subgraph size in all choice points	10589	58043	×	4492	28341	57168
Increases branching by maximising subgraph size	Pos subgraph size in all choice points	19775	60000	×	17887	55373	×
Increases branching for proponent reduces for opponent	Pos subgraph size for all prop's choice points, neg for opp's	4077	28564	×	9309	47564	80000
Moves toward D/C	Pos D/C membership and number of D/C in subgraph weights	19544	60000	×	17566	54923	×
Moves away from D/C	Neg D/C membership and number of D/C in subgraph weights	11526	56408	×	7703	36410	×
Proponent goes towards D/C, opponent goes away	Pos D/C weights for proponent, neg for opponent	8706	59336	×	10563	43965	×
Proponent goes away from D/C, opponent goes towards	Neg D/C weights for proponent, pos for opponent	19775	60000	×	13844	45231	×
Minimises assumptions	Neg 'number of assumptions' weight for all choice points	10182	52137	×	7261	37471	80000
Maximises assumptions	Pos 'number of assumptions' weight for all choice points	19726	59742	×	16720	54789	112111

Table B.7: Different Assumption Dynamics strategies benchmarked on small, medium and large frameworks. The results are report in milliseconds.

Assumption Dynamics

Description		Proponent Priority			Opponent Priority		
Strategy description	Weights Used	Small	Medium	Large	Small	Medium	Large
Proponent reduces branching, goes away from D/C, maximising assumptions, opponent reduces branching, goes toward D/C, minimising assumptions	Neg subgraph size for all choice points, negative D/C weights for proponent, positive D/C weights for opponent, positive number of assumptions for proponent, negative for opponents	10384	49693	×	4748	25799	57168
Increases branching for proponent, reduces for opponent, proponent goes towards D/C, opponent goes away from D/C	Negative subgraph size (-1) for opponent, positive (+1) for proponent, positive D/C measures for proponent (+0.25), negative for opponent (-0.25)	5269	42208	106667	×	×	×
Reduces branching for both players, both players move away from D/C and minimise assumptions	Negative subgraph size (-1) for both players in all choice points, negative D/C weights and 'number of assumptions' weights (-0.25) for both players	×	×	×	4750	28777	63492
Best performing strategy learnt - proponent priority	Maximises subgraph size for prop non-assumption choice, minimises subgraph size for all other, goes away from assumptions and D/C	3229	25963	47274	×	×	×
Best performing strategy learnt - opponent priority	Minimises subgraph size for all choice points, goes toward assumptions and D/C	×	×	×	5629	32834	69191

Table B.8: Different Assumption Dynamics strategies benchmarked on small, medium and large frameworks. The results are report in milliseconds.

Bibliography

[CTC^{+}12] R. Craven, F. Toni, C. Cadar, A. Hadad, and M. Williams, "Efficient argumentation for medical decision-making," in *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

[DKT09] P. Dung, R. Kowalski, and F. Toni, "Assumption-based argumentation," *Argumentation in Artificial Intelligence*, pp. 199–218, 2009.

[Dun95] P. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial intelligence*, vol. 77, no. 2, pp. 321–357, 1995.

[Hol75] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* University of Michigan Press, 1975. [Online]. Available: http://books.google.co.uk/books?id=YE5RAAAAMAAJ

[ICS] ICS AB, "Sicstus prolog." [Online]. Available: http://sicstus.sics.se/

[Jon88] K. Jong, "Learning with genetic algorithms: An overview," *Machine Learning*, vol. 3, no. 2-3, pp. 121–138, 1988. [Online]. Available: http://dx.doi.org/10.1007/BF00113894

[MD89] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proceedings of the eleventh international joint conference on artificial Intelligence*, vol. 1. San Mateo, CA, 1989, pp. 762–767.

[Mit98] M. Mitchell, *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press, 1998.

[New10] M. Newman, *Networks: an introduction.* Oxford University Press, Inc., 2010.

[Obj] Object Refinery Limited, "JFreeChart." [Online]. Available: http://www.jfree.org/jfreechart/

[SK93] S. Schulze-Kremer, "Genetic algorithms for protein tertiary structure prediction," in *Machine Learning: ECML-93.* Springer, 1993, pp. 262–279.

[Sta] Stanford University, "Stanford Network Analysis Platform." [Online]. Available: http://snap.stanford.edu/snap/

[Str12] F. Strub, "Improving computation in assumption base argumentation by using graph theory and machine learning"," 2012.

[Ton12] F. Toni, "A generalised framework for dispute derivations in assumption-based argumentation," *Artificial Intelligence*, 2012.