Knowledge Representation and Reasoning Fall 2022

Course Project Milestone 4  
Individual Project Report

Ayush Joshi

Arizona State University  
ajoshi72@asu.edu

Abstract

This report is a part of a series of reports for the Knowledge Representation and Reasoning Fall 2022 course at Arizona State University, and includes details for the findings made while solving the “Referee Assignment” Answer Set Programming problem, as well as providing background information about the problem and solutions workings. Aside from this, the report attempts to provide a general breakdown of the approach used in the solution of the referee assignment problem, in terms that may be readily understood by someone with basic foundational knowledge in the field of knowledge representation and reasoning, as well as some rudimentary comprehension regarding answer set programming. This report also covers details regarding the solution to the Referee Assignment Problems solution that are built upon answer set programming concepts and fundamentals of using Clingo[3], the answer set programming solver developed at the Potsdam university as part of the Potassco[2] collection for answer set solving. The advanced usage of the Clingo[4] tools which allow for the solution of the Referee Assignment Problem are also highlighted and broken down in an easier to understand manner in this report.

Problem Statement*[[1]](#footnote-1)*

The goal of this section is to allow the reader to understand, in exact terms, the scope of the problem being tackled and how such information is being presented in the following sections.

The crux of the problem revolves around finding solutions in a search space problem, without obfuscating any intermediary inferences, as well as presenting all information in a format understandable by humans, even though the solution is produced programmatically. This is exactly the goal for answer set programming, and one of the solvers for answer set programming, Clingo[4], proposed as part of the paper “Multi-shot ASP solving with Clingo” [1], will be the main tool that will allow us to produce a solution to the referee assignment problem in the manner specified prior.

The use of this tool to implement our solution then presents further the problem of properly representing the domain of the referee assignment problem in terms that can maximally utilize the intricacies of Clingo[4] in order to produce optimal results. To support this, as part of the referee assignment problem itself, we are provided with “instances” which may be utilized to test and verify our solutions to the problem, which consist of preset parameters representing a particular search space within the bounds of a referee assignment problem, as well as the corresponding optimal output that is to be expected from a correct solution. With this structure at hand, we have available to us a sound criterion for evaluating the performance of our Clingo[4] program, which shall consist of a multifaceted approach to properly encode the essential “rules” that define an optimal solution to any “instance” of the referee assignment problem.

Given this basic layout of the problem that needs to be addressed by this project, we can expand on what the “rules” that will be utilized to solve the referee assignment problem via Clingo[4] consist of and represent. The “rules” in question essentially translate one-to-one into individual pieces of the Clingo[4] “code”. These same “rules” in turn have to be constructed such that all the interactions and constraints present within the referee assignment problem domain are accurately represented. These include but may not be limited to, the interaction between referees, the cases, and tertiary parameters such as individual preference, expertise and area constraints.

The section for the project background that follows will now display the depth of the Referee Assignment Problem such that all central aspects necessary to understand the implementation of its solution may be fully understood by the reader.

Project Background

This section of the report will provide to the readers, in detail, the required knowledge to understand the Referee Assignment Problem, as well as some basic concepts about Clingo[3], in order to clearly lay out how aspects of the referee assignment problem translated into the Clingo[4] code, which itself will be discussed in detail in the next section ‘Solution Approach’.

We shall first attempt to break down and present the information regarding the referee assignment problem, specifically regarding the representation of various entities involved within. Each of these entities is represented in Clingo[4] in the form of certain “facts”. These facts will later be utilized and combined in various ways to formulate our solution.

The “Referee” entity has attributes that pertain to its properties, listed as such, “rid” which is a unique identifier, “ref\_type” which specifies whether they are categorically “internal” or “external”, note that the instances provided for validation refer to these types as “i” and “e”, “max\_workload” specifying the upper limit of working time in minutes, “prev\_workload” is a sum of all efforts till this point of time, and “prev\_payment” which is non zero only for “external” type referees, and represents a sum total of all prior payments made to “external” referees.

The “Case” entity is represented by the following attributes, with “cid” being a unique identifier for each case, “case\_type” which signifies the category of insurance claim (for example, industrial, personal, vehicle, or health claims, etcetera), “effort” which gives exact time in minutes a referee would require to spend effort on this case, “damage” defines the monetary value claimed in this case, “postC” represents the location, and “payment” illustrates the amount paid out in the situation where this case is handled by a referee of type “external”.

Aside from the core entities of “Referee” and “Case”, there are certain other “facts” that are critical to implement particular constraints within this problem domain. The property “externalMaxDamage” is a figure that sets a maximum limit on “damage” for a particular case, exceeding which that case can only be handled by an “internal” type referee. Apart from this there are 2 types of “preference” properties associated to referees via “rid”, symbolizing that referees’ preference for cases of a particular location “postC”: “prefRegion”, and for cases of a particular category “case\_type”: “prefType”. For both these attributes, a higher numerical value, the values represented as “region\_pref” and “case\_pref”, represents a higher preference of that referee to handle cases for that location and type respectively, and a value of zero for these subjecting a constraint that prevents the referee from taking cases in such locations and types respectively.

Now with these “facts” available to use, we can proceed to explore the concept of “hard constraints” in Clingo[4]. These “hard constraints” represent strict conditions that must be satisfied by each fact that Clingo[4] attempts to derive from the set of given facts, and these conditions are binary, that is, they do not have a ‘degree’ of satisfiability, but are either satisfied or not. The conditions of this type that we must satisfy in this problem are as follows. As explained prior, a referee with a “prefRegion” or “prefType” given as zero for a region or case category respectively, must never be assigned cases corresponding to that “postC” or “case\_type” respectively. A case with “damage” greater that “externalMaxDamage” must only be assigned to referees of type “internal”. Finally, the sum of “effort” of all cases assigned to a particular referee cannot exceed that referee’s “max\_workload”.

We can now proceed to discuss the concept of “weak constraints” and how they are represented with regards to Clingo[4]. Contrasted with the “hard constraints” from above, “weak constraints” do represent conditions with ‘degrees’ of preferability, and are used to identify optimal sets in between various sets of facts inferred by the Clingo[4] solver. The following “weak constraints” are applicable to our problem domain. Referees of type “internal” are preferred whenever possible, as a means to minimize cost incurred by paying the “payment” amounts to “external” referees for cases. The difference in payment to “external” referees must be minimal, and similarly, the workload of all referees should be close in value. Finally, the preference of each referee for any region or type of case should be values, such that referees with higher preference for a region or type of case should be preferentially assigned to such cases.

With all these critical components of the problem now defined, we will now explain in detail the approach taken to solve the Referee Assignment Problem.

Solution Approach

Having formed a solid foundational understanding of the problem domain, this section will present in greater detail, the specifics of a few core Clingo[4] code components, as well as a rough overview of the program as a whole, and direct the reader to the programmatic solution of the referee assignment problem in answer set programming. Note that the exact rules as well as instances used for these experiments are mentioned in the Appendix in the exact sequence in which they are covered in this section.

We will start off by delving into the basic requirements that our solution must satisfy. It must be able to accept “facts” for entities in the problem domain with attributes encoded within the form discussed in the Project Background sections, and output “facts” that it has deduced in the form of “assignments” like “assign(cid,rid)” where “cid” and “rid” correspond to case and referee IDs respectively. This formulations however, has to do with the basics of Clingo[3], and aspects of the code that have to do with simplistic applications of Clingo[4] like such will not be discussed in further detail in this report. Reference [3] has more information for the same.

Having created a search space consisting of “assign” statements in Clingo[4], we can discuss 2 of the most straightforward strict conditions or “hard constrains” implemented in Clingo[4] in the form of “integrity constraints” in accordance to the Clingo guide[3]. In case a referee of “rid” has an associated prefRegion value as zero for region “postC” then for any case with region “postC” cannot have their identifier “cid” assigned to that referee, that is “assign(cid,rid)” is invalid here. Similarly, In case a referee of “rid” has an associated prefType value as zero for category “case\_type” then for any case with category “case\_type” cannot have their identifier “cid” assigned to that referee, that is “assign(cid,rid)” is invalid here. Secondly, we have the required strict condition of assigning cases per the constraint determined by the property “externalMaxDamage”. To do so we need yet another simple comparision, if our current inferred solution set contains any “assign(cid,rid)” where the “damage” of case “cid” exceeds the value of “externalMaxDamage”, and the referee “rid” is of type “external”, then such assignment “assign(cid,rid)” is invalid. Moving forward, for ensuring no referee has their “max\_workload” exceeded, we will have to utilize the “#sum{}” aggregator function of Clingo[4] to formulate a strict condition. We can simply enumerate all the cases assigned to a particular referee, and add their efforts, and apply a condition to prevent such a sum from being greater than that referees “max\_workload” for the decided combination of case assignments.

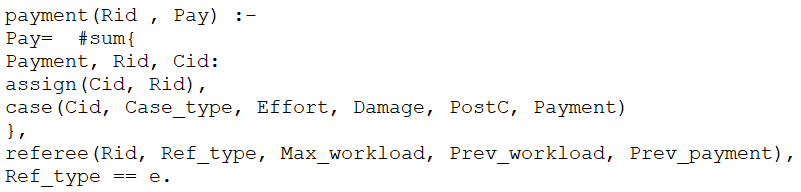
Having implemented the “hard constraints” for the problem, we can finally move on to the critical portion requiring slightly more nuanced usage and understanding of more advanced concepts of Clingo[4], involved in the implementation of “weak constraints”. As defined in the Clingo guide[3], the implementation of individual weak constraints is done via individual lines of code which act in a slightly different manner from code for “hard constraints”. In Clingo[4] “hard constraints” work essentially by dictating which derived “fact” have to be excluded from a solution, given that the condition specified in that constraint is satisfied. The “weak constraints” however, don’t directly cause the exclusion of any derived “facts”. They function by accumulating a “cost” value for a given solution each time their condition is satisfied by “facts” present in that solution, and at the end of the process of derivation, the solver identifies the solution with the least “cost” as the optimal one. However, as specified in the Clingo guide[3], there are optimization statements “#maximize” and “#minimize” that essentially act as collections of such “weak constraints”. These functions of Clingo[4], specially suit our needs for the conditions we need to implement.

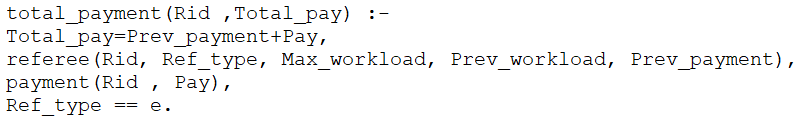
With the reader now sufficiently aware of these tools, we will now explain the process of implementing the optimization conditions that were covered in the Project Background section. For the simplest condition of “preferring internal referees”, we use the “#count{}” function prescribed in the Clingo guide[3] to find the number of “internal” referees, and subject this count to “#maximize”. Secondly, another simple condition, like with the situation of “hard constraints”, is to “allow referees to work on cases in preferred postC region; or, work on cases of preferred case\_type category” based on the present “facts” regarding prefRegion and prefType. This can also be done via the use of the “#maximize” optimizer, by using this function on the cumulative numerical values of “region\_pref” and “case\_pref” across all given “facts”.

With these basic optimizations cleared, we will discuss some complex optimizations which represent the remaining “weak constraints”. Part of the reason the following rules are complicated is due to the fact that they require some pre-processing, unlike how we directly use values given in the instances for the regional preferences and case category preferences, or do a simple #count operation, these rules will require us to first use Clingo[4] to derive some intermediary “facts”.

In order to balance the payments of “external” referees as required, we extract each payment made to an individual into a fact “payment(rid,pay)”, and calculate the total payment to that individual by summing up such payments into “total\_pay(rid,total\_pay)”. We then derive 2 more facts for minimum and maximum payment made to any referee as “min\_pay(min)” and “max\_pay(max)”, using a #count function to determine each, by ensuring that count of other payments lesser than “min” is zero, and similarly for “max”. Now we can optimize based on using #minimize on the difference between the maximum and minimum pay. We use a similar process to implement the optimization for “balancing the workload between referees”. Due to the complexity and criticality of these rules they are illustrated in the images at the end of this section. For context portions of the full program are present in the Appendix.

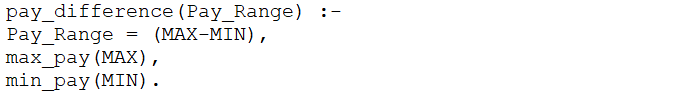
Following this process, we have successfully and completely implemented all constraints and optimizations for the solution of the Referee Assignment Problem. In the following sections we will briefly cover the results of this implementation and summarize miscellaneous details of this project.

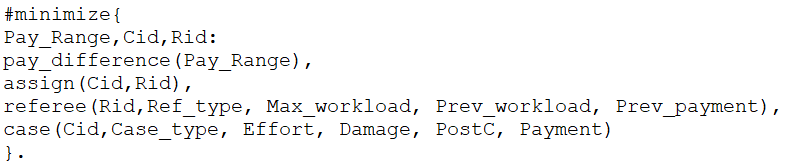
Rules for balancing payments (Note all these represent single code lines in the program, and names of all literals/variables have been kept the same as represented in the report as much as possible to preserve consistency): Step 1 create “payment” literals for each referee.

 Sum all per referee to get totals:

Calculate min pay and max pay:

|  |  |
| --- | --- |
|  |  |

Calculate difference:

And finally minimize based on this difference:

Main Results and Analysis

Up until this point this report has explained to the reader, with a high degree of detail, the domain of the Referee Assignment Problem, and how to implement a programmatic solution utilizing the capabilities of the Answer Set Programming solver, Clingo[4]. In this section we will provide to the reader a summarization of the results of using the solution described in the Solution Approach section, as well as supply the reader with a short analysis.

Using the code written with the guidelines and procedures from the prior section, we run it through the Clingo[4] solver paired with each of the test instances we have at hand, one at a time. The expected outputs of the instances, as well as the corresponding outputs are contained in the Appendix. After the execution of our code, we were able to verify that the optimal solution was being generated for each of the instances.

The code itself is clear and concise, and performs exceptionally on the provided test cases, as well as on a set of expanded complex test cases. The code does continue to yield optimal solutions for a variety of combinations of scenarios, which is a concrete testament to the success of the project as a whole. The creative usage of existing Clingo[4] functions for various rules, with correctness in mind, has allowed for this code to possess certain flexibility in possible future expansion and extension to other problem domains. However further analysis of the test cases and solution layout reveal the nature of the Referee Assignment problem to be an extremely simplified version of corresponding real-world problems.

Conclusion

As of writing this report, we have not only learned various concepts in the domain of Knowledge Representation and Reasoning, specifically pertaining to Answer Set Programing, but have also extensively explored both basic and advanced usage of Clingo[4], in order to solve problems emulating real-world scenarios, in a manner that allows all information to be conveyed in a manner understandable by humans across all intermediate steps while solving, in contrast to other methods popular in the related domain of Artificial Intelligence. We have successfully combined practical knowledge regarding the usage of such tools to present a working program for finding optimal solutions in a complex work space, with great efficiency and precision.

To conclude, we have laid out all necessary knowledge in this report for any reader with even foundational knowledge in this domain, and rudimentary experience with the toolset, to be able to replicate the results presented herein.

Opportunities for Future Work

The current work carried out in this project, and the Referee Assignment Problem as a whole, are a heavily restricted representation of solving any real-world problem. With the given flexibility of the solution produced by this project, there is scope to expand to solve more generic resource allocation scenarios, which will require capturing the intricacies of several real-world entity relationships into Clingo code, as well as representing the complicated constraints and optimizations as various compound rules.

References

[1] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, “Multi-shot ASP solving with clingo”, TPLP, 19(1), 27–82, 2019.

[2] University of Potsdam. “Potassco, the Potsdam Answer Set Solving Collection”. [Online] Available: https://potassco.org/

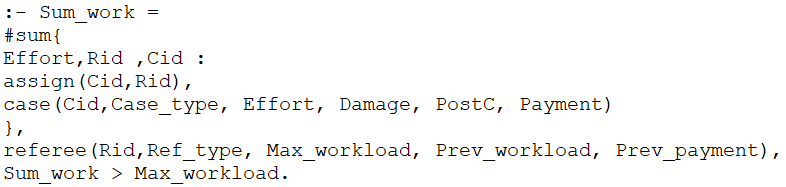
[3] Github:Potassco. (Jan 15, 2019). “Potassco guide version 2.2.0” [Online] Available: https://github.com/potassco/guide/releases/tag/v2.2.0

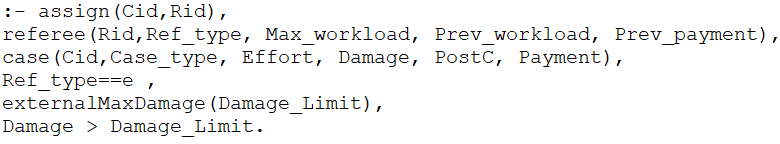
[4] University of Potsdam. “Clingo, Answer set programming solver” [Online] Available: https://potassco.org/clingo/

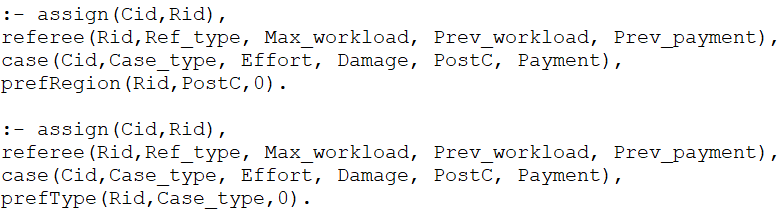
Appendix

Section 1 [CODE]: note that we attempt to keep the naming of elements of code as consistent as possible with explanations included in the report

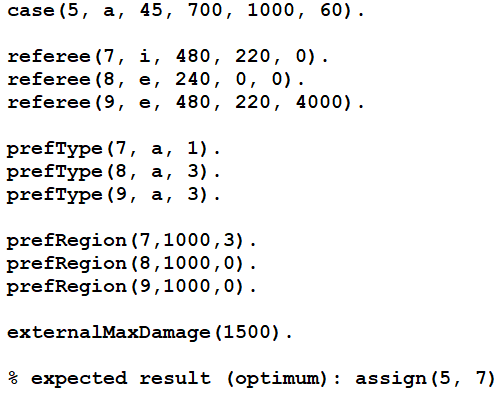
General code to create set of “assign” atoms:

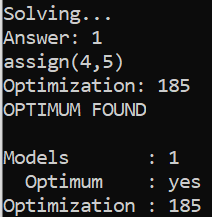
Varoius “Hard constraints”; Max\_workload not be exceeded:

Not exceeding damage limit for external referee:

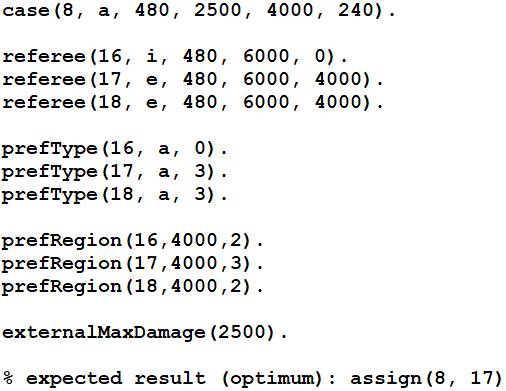
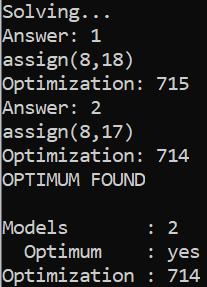
Preferences for region or case\_type:

Section 2 [Results]: we provide the contents of the instance file used in the test along with the corresponding output:

Test case 1:

Output:

Test Case 2:

Output:

1. Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved. [↑](#footnote-ref-1)