Insurance Referee Assignment Problem

Kenji Mah

Ira A. Fulton Schools of Engineering, ASU Online
Arizona State University
Tempe, Arizona
kmmah@asu.edu

***Abstract*—An insurance company is required to delegate referees to a given case to check if a customer's claims are justified. However, there is an exponential explosion for the number of possible case to referee assignments. In order to solve this problem efficiently, we will use Clingo, an answer set programming language that will allow us to use utilize knowledge representation and reasoning techniques to produce an optimal plan for assigning referees.**

***Index Terms*—Answer set programming, declarative programming, propositional logic**

I. Introduction

This problem originated from the ASP Challenge 2019 and was ran for the Technische Universität Wien (Austria) and the University of Genoa (Italy), in Spring 2019. The challenge is provided in the problem domains under "Insurance Referees Assignment Problem" [1]. An insurance company needs to send out referees to assess customer claims and create reports on the damages. However, there are many factors that can influence how the insurance company assigns the referees, such as costs for assigning a referee or if they are experienced enough to handle the case. For example, in our problem we must account for different types of referees, internal referees that work for the company and external ones that can be outsourced. We are also provided information regarding geological and personal preferences for each referee. We will need to take this information and formulate constraints to meet the insurance company's definition of the optimal solution. We can categorize these constraints into hard constraints and weak constraints. The hard constraints will define what assignments cannot be in a solution while the weak constraints model and influence the optimality of the solution.

The hard constraints are as follows:

- We need to ensure that the maximum number of working minutes of a referee must not exceed the total workload of that referee.
- Referees must not be assigned to a region they do not prefer.
- Referees must not be assigned to a case type that they do not prefer.
- Cases where the damage exceeds a certain threshold can only be handled by internal referees.

The weak constraints are as follows:

- Internal referees are preferred in order to minimize the costs of external ones.
- An external referee should have a balanced overall payments in the sense that he/she has a similar overall payment as another referee.
- All referees should have a balanced workload.
- Referees with higher preferences(region or case type) should handle the case.

In order to have a definitive optimal plan for the weak constraints we will aim to find the minimum of the following cost function:

$$cost = 16 \cdot c_A + 7 \cdot c_B + 9 \cdot c_C + 34 \cdot c_D + 34 \cdot c_E$$

where:

$$\begin{aligned}
c_A &= \textit{sum of all new payments of all external referees} \\
o_{rid} &= \textit{sum of all payments of an external referee} \\
avg_e &= \textit{average overall payments of all external referees} \\
c_B &= \textit{sum of } |avg_e - o_{rid}| \textit{ for all external referees} \\
w_{rid} &= \textit{total workload of a referee} \\
avg_w &= \textit{average overall workload of all referees} \\
c_C &= \textit{sum of } |avg_w - w_{rid}| \textit{ for all referees} \\
c_D &= \textit{sum of regional preferences } (3 - pref) \\
c_E &= \textit{sum of case type preferences } (3 - pref)
\end{aligned}$$

We are going to approach the solution as an answer set program, a form of declarative programming oriented towards difficult search problems [2]. This will be useful because there are r^n number of possible assignments where r is the number of referees and n is the number of cases. There are special constraints that we need to consider while we do the search which is why we approach this situation with a declarative program. We will formulate our search problem in the form that an answer set solver can compute to find stable models. Our solution is a positive program, so the stable model are the minimal models of the program. The minimal models are the atoms that satisfy all the rules of a program and no other model is a subset of those atoms. To find the stable models, we can view the program displayed in Appendix B as a set propositional logical statements to which we then apply a SAT solver to find the stable models.

Clingo Rules	Propositional Rules
:-	$\leftarrow$
comma in the body	$\wedge$
comma in the head	$\vee$
not	$\neg$
#false	$\bot$
#true	$\top$

TABLE I: Propositional rules to Clingo rules conversion table.

To understand the program in terms of the semantics of propositional logic we must use the following conversions in Table 1. Everything to the left of the ":-" symbol is considered the head and everything to the right of it is considered the body.

II. Solution

The data given to us comes in the form of facts which are have a schema shown in Appendix A. Our objective is to take these facts and create a program that outputs a set of predicates called assign that has an arity of two. The first argument will correspond to the case id while the second will correspond to the referee id. We will use a generate, define, test methodology for developing a solution [3].

A. Generate Step

In the generate step of our program we enumerate all possible assignments using the following Clingo rule:

```
{assign(Cid,Rid): referee(Rid,_,_,_,_)}=1 :-
    case(Cid,_,_,_,_,_).
```

This rule is all we need in our generation step because our problem is a static world problem in that it does not rely on a time variable. By creating all possible functions that maps the domain set of all cases to the codomain set of all referees. This is accomplished using a choice rule with a cardinality constraint in combination with local and global variables. This works by first grounding the global variables first, in this case the Cid of a case predicate. The groundings are the data given to us and defined with initialization facts as exemplified in Appendix C. After grounding the global variable, rules are produced for every Cid and will resemble the following:

```
{assign(Cid_1,Rid): referee(Rid,_,_,_,_)}=1.
{assign(Cid_2,Rid): referee(Rid,_,_,_,_)}=1.
        ...
{assign(Cid_n,Rid): referee(Rid,_,_,_,_)}=1.
```

In summary the choice rule with a cardinality of one will then the pair up every Cid with a single Rid. This step ensures that every case has a referee assigned to it which is one of the main goals of the insurance company.

B. Define Step

The define step for this problem is where we will define useful atoms that we can use later. The creation of atoms that help store values such as o_{rid} for the external referees or the avg_w of all referees will be useful in future computations. An example of such rule is as follows:

```
o_rid(Rid,X) :- referee(Rid,e,_,_,Prev_payment
    ), X=Prev_payment+N, N=#sum{Payment,Cid:
    case(Cid,_,_,_,_,Payment),assign(Cid,Rid)
    }.
```

In this case we will be using the sum aggregate. Like the choice rule, the global variables get grounded first in the sum aggregate and create new rules for each external referee.

```
o_rid(Rid_1,X) :- referee(Rid_1,e,_,_,
    Prev_payment_1), X=Prev_payment_1+N_1, N_1
    =#sum{Payment,Cid:case(Cid,_,_,_,_,Payment
    ),assign(Cid,Rid_1)}.
        ...
o_rid(Rid_n,X) :- referee(Rid_n,e,_,_,
    Prev_payment_n), X=Prev_payment_n+N_n, N_n
    =#sum{Payment,Cid:case(Cid,_,_,_,_,Payment
    ),assign(Cid,Rid_n)}.
```

In this case n will be the number of external referees. The sum aggregate will then sum over all payments for each case that was assigned to that referee. We then take that sum and add it to the previous payment of that referee to find the total payments made to that referee. It is important to accompany this number with the corresponding sum to the unique rid in order to differentiate the sums because Clingo will not differentiate multiple instances of the same fact. The avg_e can then be calculated by summing over all the o_{rid} facts and then dividing it by the number of external referees, which can be found by using the count aggregate. Similar techniques can be used to find the values for w_{rid} and avg_w.

C. Test Step

The test step is where we will be defining the constraints and eliminating any stable models that do not adhere to the specifications. This is also how we implement the optimization parameters. The hard constraints eliminate any stable model that conflicts with the rule. For example, the constraint where we need to ensure that a referee's workload does not exceed the total workload of that referee is implemented by the following rule:

```
:- referee(Rid,_,Max_workload,_,_),
    Max_workload-S<0, S=#sum{Effort: case(Cid,
    _,Effort,_,_,_),referee(Rid,_,_,_,_),
    assign(Cid,Rid)}.
```

This is a case where the rule has an empty head. In Clingo, this defaults to the rule to be equivalent to the propositional rule, $false \leftarrow body$. So, any stable model where the sum of the assigned cases is greater than the max workload is going to evaluate to false, therefore, excluding it from the solution set. All the hard constraints are constructed in this way.

Another example of a hard constraint is the rule where referees cannot take cases in regions that they are not responsible for. For this rule we create a rule where we evaluate if an assignment has a fact where the referee's preference equates to zero. We also need to account for the absence of a preference fact because we are to assume that the referee's preferences default to zero. We can accomplish this by adding the following rule:

```
:- assign(Cid,Rid) , {prefRegion(Rid, Postc,
    Pref)}=0, case(Cid,_,_,_,Postc,_), referee
    (Rid,_,_,_,_).
```

By setting the cardinality to zero we can eliminate any models where the assignment of a referee does not have an accompanying regional preference fact. These rules can also be implemented for the case type preferences as well.

Weak constraints need to be handled differently from hard constraints. If the body of a weak constraint is satisfied, instead of eliminating it from the set of stable models, it is given a score. By default, the minimal score is the optimal score. We can also choose to have specific constraints to have a precedence over other constraints by including a level to the score where higher levels take more precedence over lower levels. The weak constraint reflecting the minimization of payments to referees is as follows:

```
:~ C_a=#sum{Payment,Cid,Rid:case(Cid,_,_,_,_,
    Payment), assign(Cid,Rid), referee(Rid,e,_
    ,_,_)}. [16*C_a]
```

This constraint differs syntactically from the hard constraints in that ":-" is changed to ": " and there is an extra term at the end of the rule. The default level of this constraint is zero and this is what we will be using to store the value of our cost function. If another constraint shares the same level, the cost will be added to that level. We can take advantage of this mechanism when defining the constraint c_B. The following is the rule for the $7 \cdot c_B$ term in the cost function:

```
:~ averageE(A), o_rid(Rid,X). [7*|A-X|,Rid]
```

Instead of defining a separate fact to store the value of c_B we can directly apply arithmetic in the weak constraint's cost to the values in the facts created in the define step and add it to the cost. Weak constraints act similarly to the sum aggregate in that we must indicate that we want to sum over, in this case all rid's. The same logic can then be applied to the rest of the weak constraints to reflect the final cost of a stable model.

III. Results

After running all the test cases provided in the problem domain, the program proved to output the correct optimal assignments. The output of the program will display the stable models with all the predicates. The number of predicates could be overwhelming because all we really want to know is the assignment, so to filter the other predicates out we can add the following line of code to our program:

```
#show assign/2.
```

In attempts to see if any optimizations could be made to improve the time it takes for the program to complete, we created trial runs where the program was tested with differing number of threads and recorded it in Table 2.

In most cases there is no significant speed up. In fact, it seems that adding more threads slows down the completion of the program. This is probably because during the generation step, none of the test cases exceed 64 assignments, which is relatively low for today's modern computing power.

	Threads			
Test Case	t=1	t = 4	t = 8	t = 16
1	0.011	0.01	0.007	0.008
2	0.006	0.007	0.007	0.008
3	0.008	0.01	0.01	0.012
4	0.007	0.008	0.009	0.01
5	0.007	0.007	0.008	0.009
6	35.62	35.695	60.746	79.17
7	0.292	0.206	0.213	0.314
8	0.04	0.044	0.046	0.049
9	0.031	0.024	0.026	0.027
10	1.258	1.063	1.084	1.219

TABLE II: The table represents the time for the program to complete in seconds

We can also run the program with the additional argument, `--opt-mode=enum 0`, in order to enumerate all stable models with their corresponding scores. This can be useful for analytics or possibly if the insurance company decides not to use the optimal assignment for some reason.

The solution provided may not be the most optimal in terms of computation efficiency. Efficiencies to reduce or avoid storing values for atoms, like the ones in the define step, may help computation times. However, the solution is presented this way to keep the solution clear and comprehendible.

IV. Lessons Learned

By doing this project I learned many concepts in the field of AI. Answer set programming was a new method for me when it comes to solving complicated search problems. In learning ASP, I also gained a concrete understanding of propositional logic, first order logic, and how it can be utilized to solve real world problems.

V. Conclusion

In conclusion, one of the benefits of using an ASP program is mutability for different problem sets. Let's say we want to add additional constraints regarding a scheduling aspect of our problem. We can simply add specific parameters to our problem to account for conflicting time slots, as well as the information for each case with additional facts, like the preference information.

We could also change this static world problem to an online, live assignment system where the assignments and availability of referees change. This would require more difficult changes to the solution however.

We could have implemented hard constraints in terms of weak constraints by assigning the hard constraints to a level higher than the weak constraints. This would be useful if there were no stable assignments for a given initialization and the insurance company wanted to implement a different criterion for this type of situation.

References

[1] Technische Universität Wien, the University of Genoa, "Problem Domains," sites.google.com, 2019. [Online]. Avaliable: https://sites.google.com/view/aspcomp2019/problem-domains?authuser=0

[2] V. Lifschitz, "What Is Answer Set Programming?" in Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (2008), Chicago, Illinois, USA, July 13–17, 2008. pp. 1594-1597.

[3] M.Gebser, R.Kaminski, B.Kaufmann, M.Ostrowski, T.Schaub, S.Thiele, ”A User’s Guide to gringo, clasp, clingo, and iclingo,” October 4, 2010, Avaliable: http://wp.doc.ic.ac.uk/arusso/wp-content/uploads/sites/47/2015/01/clingo_guide.pdf. [Accessed Mar. 2, 2021].

VI. Appendix A

A. Referees are given as facts of form:

referee(rid, type, max_workload, prev_workload, prev_payment).

where

- rid is a positive integer that servers as unique identifier of the referee,
- type defines the type of the referee and is one of the constants internal or external,
- max_workload is an integer in the range [0,720] that defines the maximum working minutes of the referee per day,
- prev_workload is an integer that represents the previous workload of the referee (i.e., the sum of the efforts of all cases handled so far),
- prev_payment is an integer that represents the sum of previous payments of external referees (is 0 for internal ones).

B. Cases are given as facts of form:

case(cid, type, effort, damage, postc, payment).

where

- cid is a positive integer that servers as unique identifier of the case,
- type is a constant that defines the domain of the case (e.g. passenger car, truck, etc)
- effort is an integer that defines the expected number of working minutes needed to handle the case,
- damage is an integer that defines the amount of damage in Euros,
- postc postal code of the location of this case,
- payment is an integer that defines how much payment an external referee receives for handling this case.

C. The maximum damage of cases assigned to external referees is given by a single fact of form

externalMaxDamage(d).

where

- d is an integer. Cases with a damage greater than d must be handled by internal referees.

D. Preferences of regions are given as facts of form

prefRegion(rid, postc, pref).

where

- rid is a referee id,
- postc is the postal code of a region,
- pref is an integer value in the range [0,3].

It encodes that rid should take a case in region postc with preference pref, where a higher value of pref denotes higher preference. The special case pref=0 means that rid is not allowed to take a case in region postc at all (hard constraint). We assume that if for a certain pair of rid and postc no pref is specified, then it is 0 by default.

E. Preferences of case types are given as facts of form

prefType(rid, caset, pref).

where

- rid is a referee id,
- caset is a case type,
- pref is an integer value in the range [0,3].

It encodes that rid should take a case with type caset with preference pref, where a higher value of pref denotes higher preference; the special case pref=0 means that rid is not allowed to take a case with type caset at all (hard constraint). We assume that if for a certain pair of rid and caset no pref is specified, then it is 0 by default.

VII. Appendix B

```
% program for finding optimal referee assignment
#const maxWorkload = 720.

%create assign function domain(cases) codomain(refs)
{assign(Cid,Rid): referee(Rid,_,_,_,_)}=1 :- case(Cid,_,_,_,_,_).

%%%%%%%%%%%%  Hard Constraints %%%%%%%%%%%%%%%%
% The maximum number of working minutes of a referee must not be exceeded by the actual workload, where the
     actual workload is the sum of the efforts of all cases assigned to this referee.
:- referee(Rid,_,Max_workload,_,_),Max_workload-S<0, S=#sum{Effort: case(Cid,_,Effort,_,_,_),referee(Rid,_,
    _,_,_), assign(Cid,Rid)}.

% A case must not be assigned to a referee who is not in charge of the region at all
:- assign(Cid,Rid) , {prefRegion(Rid, Postc,Pref)}=0, case(Cid,_,_,_,Postc,_), referee(Rid,_,_,_,_).
:- assign(Cid,Rid) , prefRegion(Rid, Postc,0), case(Cid,_,_,_,Postc,_), referee(Rid,_,_,_,_).

% A case must not be assigned to a referee who is not in charge of the type of the case at all
:- assign(Cid,Rid), {prefType(Rid, Caset, Pref)}=0, case(Cid,Caset,_,_,_,_), referee(Rid,_,_,_,_).
:- assign(Cid,Rid), prefType(Rid, Caset, 0), case(Cid,Caset,_,_,_,_), referee(Rid,_,_,_,_).

% Cases with an amount of damage that exceeds a certain threshold can only be assigned to internal referees
    .
:- assign(Cid,Rid), case(Cid,_,_,Damage,_,_), referee(Rid,e,_,_,_), Damage > X, externalMaxDamage(X).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% Weak Constraints %%%%%%%%%%%%%%%%

% A. Internal referees are preferred in order to minimize the costs of external ones.
:~ C_a=#sum{Payment,Cid,Rid:case(Cid,_,_,_,_,Payment), assign(Cid,Rid), referee(Rid,e,_,_,_)}. [16*C_a]

% B. The assignment of cases to external referees should be fair in the sense that their overall payment
    should be balanced
o_rid(Rid,X) :- referee(Rid,e,_,_,Prev_payment), X=Prev_payment+N, N=#sum{Payment,Cid:case(Cid,_,_,_,_,
    Payment),assign(Cid,Rid), referee(Rid,e,_,_,_)}.
averageE(A) :- A=S/N, S=#sum{X,Rid: o_rid(Rid,X)}, N=#count{Rid: referee(Rid,e,_,_,_)}.
:~ averageE(A), o_rid(Rid,X). [7*|A-X|,Rid]

% C. The assignment of cases to (internal and external) referees should be fair in the sense that their
    overall workload should be balanced.
w_rid(Rid,X) :- referee(Rid,_,_,Prev_workload,_), X=Prev_workload+N, N=#sum{Effort,Cid:case(Cid,_,Effort,_,
    _,_),assign(Cid,Rid)}.
average(A) :- A=S/N, S=#sum{X,Rid: w_rid(Rid,X)}, N=#count{Rid: referee(Rid,_,_,_,_)}.
:~ average(A),w_rid(Rid,X). [9*|A-X|,Rid]

% D. Referees should handle types of cases with higher preference.
:~ S=#sum{3-Pref,Cid,Rid: assign(Cid,Rid), referee(Rid,_,_,_,_), case(Cid, Caset, _, _, _, _), prefType(Rid
    ,Caset,Pref)}. [34*S]

% E. Referees should handle cases in regions with higher preference.
:~ S=#sum{3-Pref,Cid,Rid: assign(Cid,Rid), referee(Rid,_,_,_,_), case(Cid, Caset, _, _, _, _), prefRegion(
    Rid,Postc,Pref)}. [34*S]

#show assign/2.
```

VIII. Appendix C

```
% example test case (case 10)
% optimal assignment: assign(1, 1) assign(2, 1) assign(3, 3)
case(1, a, 120, 1600, 1200, 73).
case(2, a, 440, 1000, 1190, 91).
case(3, b, 120, 800, 1190, 31).
referee(1, i, 600, 220, 0).
referee(2, e, 360, 140, 2800).
referee(3, e, 240, 40, 700).
prefType(1, a, 1).
prefType(1, b, 2).
prefType(2, a, 2).
prefType(3, b, 2).
prefRegion(1, 1200, 1).
prefRegion(1, 1190, 1).
prefRegion(2, 1200, 2).
prefRegion(2, 1190, 1).
prefRegion(3, 1190, 1).
externalMaxDamage(1500).
```