Decision Management Community Challenge Nov-2018 - Vacation Days Advanced
A solution using Blaze Advisor

(Bob Moore, JETset Business Consulting, 9th Nov 2018)

1 Problem Statement (cut and pasted from the web site)

Every employee receives vacation days according to the following rules:

1. Every employee receives at least 22 vacation days.
2. Employees younger than 18 or at least 60 years, or employees with at least 30 years of service extra 5 days.
3. Employees with at least 30 years of service and also employees of age 60 or more, can receive extra 3 days, on top of possible additional days already given
4. If an employee has at least 15 but less than 30 years of service, extra 2 days can be given. These 2 days can also be provided for employees of age 45 or more.
5. A college student is eligible to 1 extra vacation day.
6. If an employee is a veteran, 2 extra days can be given.
7. The total number of vacation days cannot exceed 29.

2 A Naive Solution

The problem statement as it stands makes it a bit vague as to what exactly the challenge is. My simplistic interpretation was that given an employee and all their relevant characteristics determine how many vacation days they are entitled to.

I recently downloaded an evaluation of Blaze Advisor in response to a job prospect, so I thought I'd address this challenge using that as a tool. Since the problem is posed as a set of rules the obvious approach was to simply implement the solution as ruleset. So, the solution comprises three parts:

1. An object model – from the rules we see an employee has four characteristics:
   a) An age (**age** an integer)
   b) Their length of service (**yearsService** – an integer)
   c) If they are a college student (**collegeStudent** a boolean)
   d) If they are a veteran (**veteran** a boolean)

   We also add a few extra characteristics to hold the results:
   e) the 'basic' number of vacation days an employee is entitled to (**basicEntitlement** an integer)
   f) the 'extra' number of vacation days an employee is entitled to (**extraDays** an integer)
   g) an 'explanation' how the total number of vacation days was calculated (**explanation** a string)

2. Some boilerplate to create an employee instance, kick off the rules and print the outcome. In practise one would call the rules as a service, but here we just 'ask'

the user to input the relevant data, call the rules and print the result. In Blaze Advisor a 'main function' something like this will suffice[1]:

```
function main
is {
  // create the employee and ask the user for the input attributes
  theEmployee is some Employee initially an Employee.
  theEmployee.age = promptInteger("Give employee age").
  theEmployee.collegeStudent =
    promptBoolean("Is the employee a college student?").
  theEmployee.veteran = promptBoolean("Is the employee a veteran?").
  theEmployee.yearsService = promptInteger("Give employee years of service").

  // evaluate the rules
  VacationRules(theEmployee).

  // print the result
  print("Employee vacation allowance is "
    (theEmployee.basicEntitlement + theEmployee.extraDays) " days").
  print(theEmployee.explanation)
}
```

3. The rules themselves. Here the seven rules are translated pretty much literally from English into Advisor's SRL language, giving the following:

```
ruleset VacationRules
for {
  theEmployee : a Employee
}
is {

// 1. Every employee receives at least 22 vacation days.
rule Default
is
if  theEmployee.basicEntitlement is unknown
then {
  theEmployee.extraDays = 0.
  theEmployee.basicEntitlement = 22.
  theEmployee.explanation = "Employee has 22 days by default\n".
}

//  2. Employees younger than 18 or at least 60 years, or employees
//     with at least 30 years of service can receive extra 5 days.
rule YoungOrOldOrExperienced
is
if theEmployee.age < 18
or theEmployee.age >= 60
or theEmployee.yearsService >= 30
then {
  extraDays is an integer initially 5.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
  theEmployee.explanation =
    theEmployee.explanation "Employee gains " extraDays
    " days by virtue of YoungOrOldOrExperienced rule\n".
}
```

[1] I make the hopefully not unreasonable assumption that Blaze Advisor's rule language – SRL – is sufficiently readable that no detailed explanation of the syntax and semantics are required

```
//  3. Employees with at least 30 years of service and also
//      employees of age 60 or more, can receive extra 3 days, on top
//      of possible additional days already given
rule OldOrExperienced
is
if theEmployee.age >= 60
or theEmployee.yearsService >= 30
then {
  extraDays is an integer initially 3.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
  theEmployee.explanation =
    theEmployee.explanation "Employee gains " extraDays
     " days by virtue of OldOrExperienced rule\n".
}

// 4. If an employee has at least 15 but less than 30 years of
//     service, extra 2 days can be given. These 2 days can also be
//     provided for employees of age 45 or more.
rule MidService
is
if theEmployee.age >= 45
or (theEmployee.yearsService >= 15 and theEmployee.yearsService < 30)
then {
  extraDays is an integer initially 2.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
  theEmployee.explanation =
    theEmployee.explanation "Employee gains " extraDays
    " days by virtue of MidService rule\n".
}

//  5. A college student is eligible to 1 extra vacation day.
rule CollegeStudent
is
if theEmployee.collegeStudent
then {
  extraDays is an integer initially 1.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
    theEmployee.explanation =
    theEmployee.explanation "Employee gains " extraDays
    " days by virtue of CollegeStudent rule\n".
}

//  6. If an employee is a veteran, 2 extra days can be given.
rule Veteran
is
if theEmployee.veteran
then {
  extraDays is an integer initially 2.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
  theEmployee.explanation =
    theEmployee.explanation "Employee gains " extraDays
    " days by virtue of Veteran rule\n".
}

//  7. The total number of vacation days cannot exceed 29.
rule OverallCap
is
if theEmployee.basicEntitlement + theEmployee.extraDays > 29
then {
  excessDays is an integer initially
```

```
        theEmployee.basicEntitlement + theEmployee.extraDays - 29.
      theEmployee.extraDays = theEmployee.extraDays - excessDays.
      theEmployee.explanation =
        theEmployee.explanation "Additional vacation reduced by " excessDays
        " days by virtue of OverallCap rule\n".
  }
  }
```

So, building a solution took maybe 15 minutes (tiding it up into a format suitable for presentation took rather longer). If we take a few sample cases:

An employee aged 17, with 0 years of service, not a collegeStudent, not a veteran, generates the following output:

```
Employee vacation allowance is 27 days
Employee has 22 days by default
Employee gains 5 days by virtue of YoungOrOldOrExperienced rule
```

An employee aged 18, with 1 years of service, not a collegeStudent, not a veteran, generates the following output:

```
Employee vacation allowance is 22 days
Employee has 22 days by default
```

(a bit sad, this employee now has a year's experience but has lost 5 vacation days!)

An employee aged 46, with 10 years of service, not a collegeStudent, but is a veteran, generates the following output:

```
Employee vacation allowance is 26 days
Employee has 22 days by default
Employee gains 2 days by virtue of MidService rule
Employee gains 2 days by virtue of Veteran rule
```

An employee aged 60, with 1 year of service, not a collegeStudent, not a veteran, generates the following output:

```
Employee vacation allowance is 29 days
Employee has 22 days by default
Employee gains 5 days by virtue of YoungOrOldOrExperienced rule
Employee gains 3 days by virtue of OldOrExperienced rule
Employee gains 2 days by virtue of MidService rule
Additional vacation reduced by 3 days by virtue of OverallCap rule
```

(age trumps experience!)

3 Discussion

Is this right? I'd argue strongly that this is a perfectly correct implementation of the provided rules, but some of the peculiarities of the outcomes suggest that what was asked for was not exactly what was wanted. For example, if you have 30 years' experience, rule 2 gives you 5 extra days every time, rule 3 gives you 3 extra days every time and then rule 7 says you have to give one of those back. From a business perspective it would make more sense simply to say if you have 30 years' experience (or are over 60) you get 29 days.

A hint as to what has gone wrong is in the second part of rule 3. This specifically says that these extra days are over and above some other days. But other rules simply say extra days 'can be given' or 'can be received' or the employee is 'eligible'. We have conditions which are necessary for the employee to be eligible for the extra days, but in the absence of constraints one doesn't have much choice but assume that they are sufficient as well.

If we compare this challenge to the original 2016 challenge, the equivalent of rule 4 has an extra sentence. It reads: “If an employee has at least 15 but less than 30 years of service, extra 2 days can be given. These 2 days can also be provided for employees of age 45 or more. *These extra 2 days cannot be combined with the extra 5 days*.”

Looking back to the output for an employee aged 60 above we see that rule 2 and rule 4 both fire because we have no such constraint in the updated problem statement (it doesn’t affect the outcome of course as we’ve already seen employees aged 60 always get the maximum number of days anyway).

The implication here is that the rules really ought to have said ‘one should not apply both rule 2 and rule 4’. And what about rules 5 and 6? Do we always add the extra days when these apply? Or only when the employees don’t already have ‘enough’ extra days? For example, does rule 6 really mean something like “*if the employee is a veteran, and the number of extra days they are entitled to according to other rules is less than two, then the number of extra days they are entitled to is two.*” There are several alternatives we might try to combine the rules, where some rules can be used together, and some cannot.

Another possibly implied but unstated idea for this challenge (going back to the original 2016 challenge) was it be implemented using decision tables. I’d argue that direct translation of the rules in English into rules in a ruleset provides an implementation which provides a much clearer mapping of what the business has specified.

4 A ‘Corrected’ Solution

I’ve noted before, almost everyone tackling a DMC challenge will eventually stray away from the original problem statement onto something of their own liking. Time to stray!!

The solution presented above meets the problem as stated but does not address the implied existence of additional constraints. Unfortunately, we have to guess at what these additional constraints should be. One approach is to go back to the business and get them to rationalise their rules. This is not a matter of changing the policy, just re-expressing it in a manner that they themselves can better understand. For example, we might get them to agree to the following restatement:

1. Every employee receives at least 22 vacation days.
2. Employees younger than 18 receive an extra 5 days.
3. Employees with at least 30 years of service and employees of age 60 or more, receive an extra 7 days
4. If an employee has at least 15 but less than 30 years of service or if they are aged between 45 and 59, they receive an extra 2 days
5. A college student is eligible to 1 extra vacation day in addition to any extra days received due to age or years of service.
6. If an employee is a veteran, they are entitled to at least 2 extra days over the minimum number of vacation days.
7. The total number of vacation days cannot exceed 29.

Apart from Rule 6, things behave exactly as before, but we’ve made things a bit simpler by separating the treatment of the very young and the very experienced (or at least mature) employees, so rule 2 is only about young people, while rule 3 handles the

implications of both Rule 2 and 3 in the original formulation with regard to grey beards. We've also removed the redundant firing of Rule 4 for the mature employees and made the applicability of Rule 5 more explicit.

Should we state Rule 4 in the manner of the original 2016 challenge? I'd strongly argue not. It's very simple to do this – we add a condition that the rule about experienced or mature employees should not also hold, so the conditions of Rule 4 look something like:

```
rule MidService
is
if (theEmployee.age >= 45
or (theEmployee.yearsService >= 15 and theEmployee.yearsService < 30))
and not YoungOrOldOrExperienced
then {
  …
}
```

However, apart from the fact the original statement of the rule is plain perverse (why give upper and lower limits for experience and then just forget to give an upper limit for age?), putting exception cases in rules makes them harder to understand. If we manage to get the rules restated, get them restated using good practise. For myself, I would be suggesting the business actually change the rules to something more like:

- Employees with at least 30 years of service and employees of age 60 or more, receive an extra 5 days
- Employees with at least 15 years of service and employees of age 45 or more, receive an extra 2 days

This gives the same result[2] but is easier to understand for both employer and employee. With this formulation it is clear that an employee is rewarded for being experienced and/or mature and the reward increments over time as employees become more experienced and/or mature.

Rule 6 is rather different. Given the idea extra days aren't given out wholesale, it provides a plausible interpretation of the phrase 'can be given' which ensures veterans always get *at least* 2 extra days[3]. However, these extra days need not be in addition to other extra days already allowed. So, for example a veteran who is also a college student only gets 2 extra days, not 3. Likewise, two veterans one aged 25 with 2 years' experience, the other 50 with 20 years' experience both get 2 extra days, the first gets the extra days by Rule 6, but the second would have had them by Rule 4 even if they were not a veteran.

This kind of rule throws a bit of a spanner in the works. We cannot determine if we need to modify the number of extra days a veteran has until we've applied the other rules. Rules 2 to 5 are quite independent of one another, but Rule 6 is not. In practise if we rewrite the original ruleset using the new formulation in a naive manner we get:

```
ruleset VacationRules1
for {
  theEmployee : a Employee
```

[2] At least in almost all cases, but I confess I've not sat down and checked every combination

[3] It's not a great deal for veterans, but they always start on *at least* 24 days of vacation regardless of age and experience

```
}
is {

// 1. Every employee receives at least 22 vacation days.
rule Default
is
if  theEmployee.basicEntitlement is unknown
then {
  theEmployee.extraDays = 0.
  theEmployee.basicEntitlement = 22.
  theEmployee.explanation = "\tEmployee has 22 days by default\n".
}

//  2. Employees younger than 18 receive an extra 5 days.
rule Young
is
if theEmployee.age < 18
then {
  extraDays is an integer initially 5.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
  theEmployee.explanation =
    theEmployee.explanation "\tEmployee gains " extraDays
    " days by virtue of Young rule\n".
}

//  3. Employees with at least 30 years of service and employees
//     of age 60 or more,  receive an extra 7 days
rule OldOrExperienced
is
if theEmployee.age >= 60
or theEmployee.yearsService >= 30
then {
  extraDays is an integer initially 7.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
  theEmployee.explanation =
    theEmployee.explanation "\tEmployee gains " extraDays
     " days by virtue of OldOrExperienced rule\n".
}

// 4. If an employee has at least 15 but less than 30 years of service
//    or if they are aged between 45 and 59 they receive an extra 2 days
rule MidService
is
if theEmployee.age is between 45 and 59
or theEmployee.yearsService is between 15 and 29
then {
  extraDays is an integer initially 2.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
  theEmployee.explanation =
    theEmployee.explanation "\tEmployee gains " extraDays
    " days by virtue of MidService rule\n".
}
```

```
//  5. A college student is eligible to 1 extra vacation day in addition to
//     any extra days received due to age or years of service.
rule CollegeStudent
is
if theEmployee.collegeStudent
then {
  extraDays is an integer initially 1.
  theEmployee.extraDays = theEmployee.extraDays + extraDays.
    theEmployee.explanation =
    theEmployee.explanation "\tEmployee gains " extraDays
    " days by virtue of CollegeStudent rule\n".
}

//  6. If an employee is a veteran, they are entitled to at least 2 extra days
//     over the minimum number of vacation days.
rule Veteran
is
if theEmployee.veteran and theEmployee.extraDays < 2
then {
  extraDays is an integer initially 2.
  theEmployee.extraDays = extraDays.
  theEmployee.explanation =
    theEmployee.explanation "\tEmployee is entitled to " extraDays
    " days by virtue of Veteran rule\n".
}

//  7. The total number of vacation days cannot exceed 29.
rule OverallCap
is
if theEmployee.basicEntitlement + theEmployee.extraDays > 29
then {
  excessDays is an integer initially
    theEmployee.basicEntitlement + theEmployee.extraDays - 29.
  theEmployee.extraDays = theEmployee.extraDays - excessDays.
  theEmployee.explanation =
    theEmployee.explanation "\tAdditional vacation reduced by " excessDays
    " days by virtue of OverallCap rule\n".
}
}
```

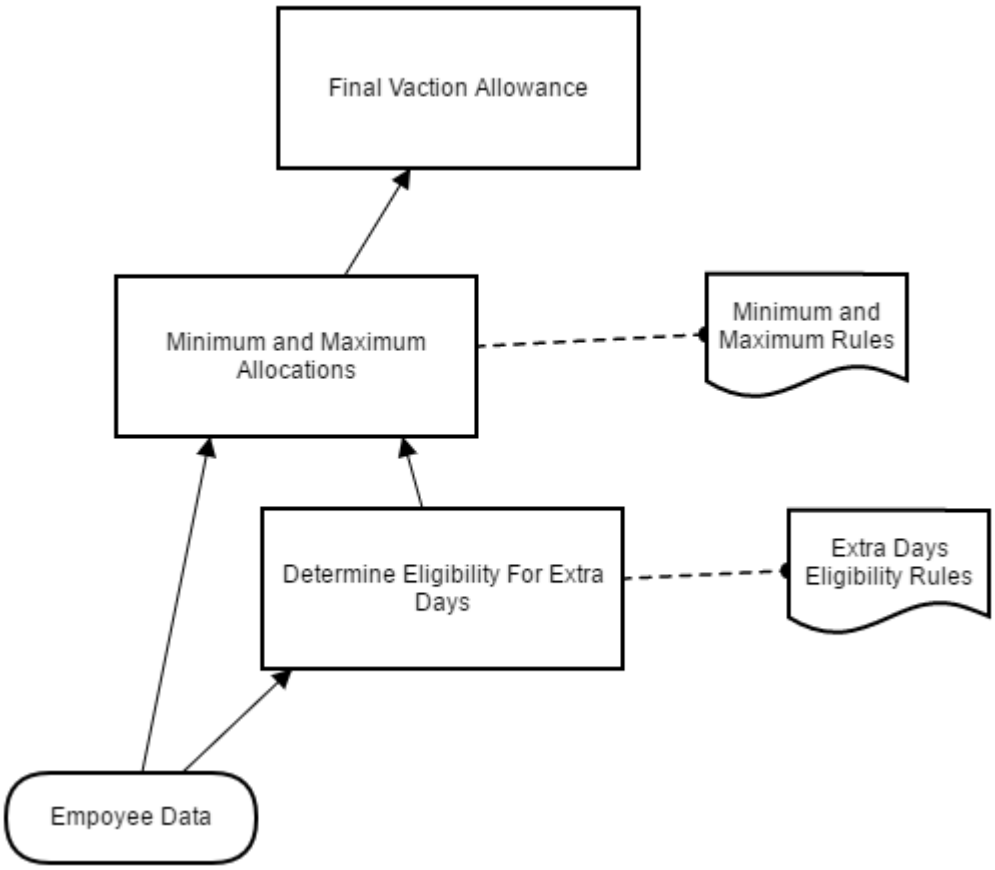
And this ruleset actually works perfectly correctly, but this is a bit of an accident. By default, Blaze Advisor processes rules using a version of the RETE algorithm, but as with most RETE algorithms, the default order of execution in the absence of inference and/or missing data, is that rules are executed top to bottom. So, by virtue of its position in the ruleset, Rule 6 will only fire after the other rules which add extra days, and thus works properly. If it is positioned at the top of the ruleset the outcome would be to two extra days for veterans over and above other extra days, rather than only if they don't get at least two extra days.

One often gets away with ignoring this problem in a small ruleset, but if more rules where there are interdependencies are added things can break – consider for example adding a new rule about an extra 2 days for people called 'Bob' after Rule 6. Any veterans called Bob end up with 4 extra days not 2.

One approach to making sure the solution is not dependent on the rule ordering is simply to model the decision-making process as comprising two stages, first we apply the rules which are independent of each other (Rules 2 to 5) and then those which

depend on the outcome of these rules (Rules 6 and 7). We can capture the approach with a simple DRD (see below). To 'implement' the DRD in Blaze Advisor, we could use a rule flow, or simply modify our boilerplate logic to call first the 'Extra Days Eligibility Rules', followed by the 'Minimum and Maximum Rules'.

The naming of the second knowledge source is based on the observation that Rule 6 (in the updated formulation) is actually a rule about the minimum number of extra days for a certain class of employees (veterans). More minimum and/or maximum rules could be envisaged.



The two rulesets now read as follows:
Firstly the 'Extra Days Eligibility Rules':

```
ruleset ExtraDaysEligibility
for {
	theEmployee : a Employee
}
is {

// 1. Every employee receives at least 22 vacation days.
rule Default
is
if  theEmployee.basicEntitlement is unknown
then {
	theEmployee.extraDays = 0.
	theEmployee.basicEntitlement = 22.
	theEmployee.explanation = "\tEmployee has 22 days by default\n".
}
```

```
//  2. Employees younger than 18 receive an extra 5 days.
rule Young
is
if theEmployee.age < 18
then {
       extraDays is an integer initially 5.
       theEmployee.extraDays = theEmployee.extraDays + extraDays.
       theEmployee.explanation =
              theEmployee.explanation "\tEmployee gains " extraDays
              " days by virtue of Young rule\n".
}

//  3. Employees with at least 30 years of service and employees
//      of age 60 or more,  receive an extra 7 days
rule OldOrExperienced
is
if theEmployee.age >= 60
or theEmployee.yearsService >= 30
then {
       extraDays is an integer initially 7.
       theEmployee.extraDays = theEmployee.extraDays + extraDays.
       theEmployee.explanation =
              theEmployee.explanation "\tEmployee gains " extraDays
               " days by virtue of OldOrExperienced rule\n".
}

// 4. If an employee has at least 15 but less than 30 years of service
//     or if they are aged between 45 and 59 they receive an extra 2 days
rule MidService
is
if theEmployee.age is between 45 and 59
or theEmployee.yearsService is between 15 and 29
then {
       extraDays is an integer initially 2.
       theEmployee.extraDays = theEmployee.extraDays + extraDays.
       theEmployee.explanation =
              theEmployee.explanation "\tEmployee gains " extraDays
              " days by virtue of MidService rule\n".
}

//  5. A college student is eligible to 1 extra vacation day in addition to
//     any extra days received due to age or years of service.
rule CollegeStudent
is
if theEmployee.collegeStudent
then {
       extraDays is an integer initially 1.
       theEmployee.extraDays = theEmployee.extraDays + extraDays.
              theEmployee.explanation =
              theEmployee.explanation "\tEmployee gains " extraDays
              " days by virtue of CollegeStudent rule\n".
}
}
```

While the ‘Minimum and Maximum Rules’ are as follows[4]:

[4] There is a strong case for putting the ‘Default’ rule in the ‘Minimum and Maximum Rules’ instead of the ‘Extra Days Eligibility Rules’. But it involves shuffling round some of the initialisation code too, and I was too lazy to do this

```
ruleset MinAndMax
for {
	theEmployee : a Employee
}
is {

//  6. If an employee is a veteran, they are entitled to at least 2 extra days
//      over the minimum number of vacation days.
rule Veteran
is
if theEmployee.veteran and theEmployee.extraDays < 2
then {
	extraDays is an integer initially 2.
	theEmployee.extraDays = extraDays.
	theEmployee.explanation =
		theEmployee.explanation "\tEmployee is entitled to " extraDays
		" days by virtue of Veteran rule\n".
}

//  7. The total number of vacation days cannot exceed 29.
rule OverallCap
is
if theEmployee.basicEntitlement + theEmployee.extraDays > 29
then {
	excessDays is an integer initially
		theEmployee.basicEntitlement + theEmployee.extraDays - 29.
	theEmployee.extraDays = theEmployee.extraDays - excessDays.
	theEmployee.explanation =
		theEmployee.explanation "\tAdditional vacation reduced by " excessDays
		" days by virtue of OverallCap rule\n".
}
}
```

5 Comparison with Other Approaches

At the time of writing there are no other solutions to this challenge, but there are a couple of other sources worth exploring. Firstly, the solutions to the original 2016 challenge and then a couple of presentations by Jacob Feldman from Decision Camp 2018[5], where the problem considered is some way between this challenge and the original 2016 version.

Two key aspects of the original challenge compared to the current one are:

- The solutions were explicitly expected to use decision tables
- There is no upper limit on the total vacation time

Some of the 20(!) solutions to the original challenge are quite elegant but (to my mind at least) most 'cheat'. These solutions simply expand out the rules and effectively just enumerate the possibilities and pre-calculate the vacation allowances.

[5] See https://decisioncamp2018.files.wordpress.com/2018/09/decisioncamp2018-jacobfeldman1.pptx and https://decisioncamp2018.files.wordpress.com/2018/09/decisioncamp2018-closingnotes-jacobfeldman.pdf

Examples of this are the OpenRules Solution 1[6] and the Corticon Solution 1[7]. Effectively the decision table designers are manually performing a kind of 'compilation' task one would hope the implementation engine could achieve. Quick examination of the rules shows that we have only 4 age ranges (0-17,18-44, 44-59 and 60+) and 3 experience ranges (0-14, 15-29 and 30+), and if we stick with these categorisations of the employees, then no matter how complicated we then make the rules we only need a decision table with 12 rows at most to work out an employee's vacation eligibility[8]. So, for a 'toy' sized problem like this such an approach is perfectly viable. Even for the current 'advanced' problem we have no more than 48 possible cases (including the unrealistic ones where the experience exceeds the age), so using brute force is still not hideously onerous.

We shouldn't be too sniffy about this kind of brute force approach for small problems, but its down side is it doesn't scale and (in my humble opinion), even for a problem this simple, it makes it more challenging for the business to see that what is implemented corresponds to what they originally specified.

Part of the need to 'cheat' simply comes down to the rules involving 'or' connectives, Decision tables don't deal with conditions involving 'or' connectives very well[9]. To handle 'or' connectives, you either have to create multiple decision tables, or you have to break down the various combinations of truth values for each clause in the 'or' conditions (or 'cheat' as I have rather pejoratively termed it).

Gary Hallmark[10] and Jacob Feldman[11] both provide what I found more elegant and scalable solutions. These better preserve the original statement of the problem, by implementing each rule as a separate decision table and then using an additional piece of logic to combine them.

Adding in the additional constraint of a cap to the vacation allowance is of course trivial for the brute force approach (if one is prepared to ignore the effort of taking a look at every output row in the decision table)[12] and the solution still comprises a single decision table. For the split approaches illustrated by Gary and Jacob, it does imply yet another extra piece of logic (which you'd also need to accommodate Rule 6 as reformulated in section 4 above).

---

[6] See https://dmcommunity.files.wordpress.com/2016/01/vacationdays1.jpg

[7] See https://dmcommunity.files.wordpress.com/2016/01/vacationdays-corticon1.jpg

[8] Actually, we really only need 9 rows taking into account the observation that under 18s cannot realistically have as much as 15 years' service, nor people under 45 as much as 30 years' experience. With the given three rules provided it turns out 8 rows suffice.

[9] Rules handle 'or' conditions very well, unfortunately human being often struggle with them. I generally discourage people from using them unless (as I think is the case here) the usage is sufficiently simple as to be unambiguous in meaning and that their use suitably simplifies the problem statement.

[10] See https://dmcommunity.files.wordpress.com/2016/01/vacationdays-garyhallmark.jpg

[11] See https://dmcommunity.files.wordpress.com/2016/01/vacationdays3.jpg

[12] This emphasizes that maintenance of a brute force approach can be hard work

Overall, I have to say that I found a straightforward rules-based implementation required less thought to build and looks easier to verify and explain than using decision tables. Given their more expressive power, adding in new and exotic exceptions is obviously easier using rules rather than decision tables. However, I'd certainly expect a more disciplined and structured approach to the statement of the business logic would level the playing field somewhat. Even simply restating the 'mid service' rule in a rational manner makes life a lot simpler.

Finally, my interest was piqued at the rather different approach Jacob explored in the second of his two presentations at DecisionCamp 2018[13], considering if a constraint-based approach could be used, to provide an optimal solution incorporating the overall cap of 29 days. The idea here is to try an incorporate the constraints explicitly into an overall 'Model-Based' solution.

While I really liked the general idea, my reading of Jacob's solution as presented does suggest it subtly imposes some rather nasty (but not at once obvious) constraints. For example if you are over 60 you are entitled by the first rule to 5 days, and by the second to 3, but this takes you over the 29-day limit, and the logic seems to dictate that you have to lose *either* all the 3 days or all the 5 days, meaning you can get at best only 27 days.

In line with the 'can receive' qualification in the challenge statement (which to be fair is missing from Jacob's statement of the rules), one would expect (for example) that you get 'up to' 3 days from Rule 3, not either all 3 days or no days at all. This issue can be obviously be circumvented by applying the 29-day limit after calculating the number of days but splitting out the exceptions in this way was something which this approach was intended to avoid.

Keeping with the spirit of what was intended it would seem that the optimisation problem needs to treat the number of extra days awarded as variables as well as the eligibility rules, so the constraint problem could be formulated along the following lines[14]:

> $x_t \in \{0,1\}$ = 1 if an employee is eligible for extra days of type t, = 0 if otherwise
> $y_t \in \{0,1, 2, \ldots max_t\}$ max_t being the maximum extra days permitted of type t
>
> Maximize $Total_e = 22 + \sum_t y_{t1} * x_{t1}$
> Subject to $Total_e <= 29$

An interesting challenge with this constraint-based approach may be how to 'explain' the outcome if the total is capped, something which using rules (even if it might be considered more 'Method-Based' by Jacob) manages straightforwardly.

---

[13] See pages 5 & 6 of https://decisioncamp2018.files.wordpress.com/2018/09/decisioncamp2018-closingnotes-jacobfeldman.pdf

[14] Of course, it possibly already is – clearly the whole solution is not on a single PowerPoint slide and I may be misunderstanding all that was intended to be conveyed by it!