NTNU
Norwegian University of
Science and Technology

Air controller scheduling at Avinor

Torgeir Woldstad Skogen

Master of Science in Computer Science
Submission date: June 2017
Supervisor: Magnus Lie Hetland, IDI
Co-supervisor: Jan Hallvard Larsen, Avinor

Norwegian University of Science and Technology
Department of Computer Science

Assignment text

Avinor is in charge of the air traffic in Norway. Their work schedule needs to meet the demands at any time. This thesis relates to some problems in this scheduling. The goal is to implement a module that solves these problem. In particular, the chosen scheduling problems are about assigning shifts to controllers and assigning the air space sectors to controllers.

With rules regarding the duration of a shift, as well as the duration of work and break periods within the shift, the schedule must meet the forecasts for the demand throughout the day. Ideally the plan includes as few shifts as possible, and there will not be more time spent on break than necessary.

Controllers want to work in as many different sectors as possible, to ensure they stay relevant in all sectors - as traffic patterns, rules and regulations differ between sectors. Ideally, there should only be small changes to sector assignment during working periods. After breaks, the controllers can get a completely new set of sectors to work in.

Abstract

The air space in Norway is controller by Avinor. This work is done by air traffic controllers, who operate under strict rules regarding how this work is carried out. This has implications on how long they can work and how many breaks they need.

In this paper, I present problems related the planning of this work, as well as a proposed algorithm for each one. The first of this is scheduling shifts for one day to meet the given demands, which indicate how many controllers are required to be working at each time. The second problem is to assign sectors of air space to shifts, such that the air space is covered. How sectors are assigned changes similarly to demands, and ideally, controllers work in stable combinations of sectors. Lastly, the given demands are forecasts, and to have an appropriate schedule when the demands are different from expected, the schedule usually use upper estimates for demands. In cases where the demands are lower, it is desired to have a plan for which controllers should get breaks.

The first two of these problems are solved using branch and bound searches and the last one is solved using a greedy algorithm. For the first one, the complexity is high, so branch and bound does not seem to be an ideal solution strategy, while for the second one it seems much more promising. The greedy solution to the third problem has a short running time, but does not guarantee an optimal solution.

Sammendrag

Luftrommet i Norge er kontrollert av Avinor. Dette arbeidet blir gjort av flyveledere, som opererer under strenge regler angående hvorden dette arbeidet blir gjort. Dette har konsekvenser angående hvor lenge de kan jobbe og hvor mange pauser de trenger.

I denne oppgaven presenterer jeg problemer relatert til planleggingen av denne jobben, så vel som forselåtte algoritmer til hvert problem. Det første av disse problemene er å planlegge skift for en dag om gangen, slik at den gitte etterspørselen blir møtt. Det andre problemet er å tildele sektorer av luftrommet til flyvelederne, slik at hele luftrommet er dekket. Måten sektorene fordeles forandrer seg i løpet av dagen sammen med etterspørselen. Ideelt sett jobber flyvelederne i så stabile kombinasjoner av sektrorer som mulig. Til slutt, den gitte etterspørselen består av forhåndsestimater, og for å ha fornuftige planer i de tilfellene hvor den reelle etterpørselen spriker med estimatene, så er det vanlig å bruke høye estimater. For å håndtere lavere etterspørsel blir det planlagt hvilke skift som får ekstra pauser.

De første to av disse problemene har blitt løst ved bruk av branch and bound søketeknikken, og den siste er løst med en grådig algorithe. For det første problemet er kompleksiteten høy, slik at branch and bound virker å være en for treg løsningsstrategi. Det andre problemet virker på den andre siden virker som et bedre egner problem å løse med denne typen algorithe. Den grådige løsningen på det tredje problemet har lav kjøretid, men garanterer ikke en optimal løsning.

Contents

Chapter 1

Introduction

1.1 Avinor

Responsible for 45 state-owned airports in Norway, Avinor is a limited company owned by the state. The company is organized under the Norwegian Ministry of Transport and Communications.In addition to the airports themselves, Avinor is operates control towers, control centers and other technical infrastructure for safe air navigation.[1]

This thesis relates to the operation of control towers, specifically to scheduling of air traffic controllers. Before starting, three problems were presented, each relating to separate parts of the schedule. In the first problem, there are given demands that specify how many controllers are needed at each time during the day, in blocks of a given duration. Also given are rules about how much each controllers can work, including the duration of breaks periods and work periods. The goal is then to optimize the schedule for a full day so that as few controllers are needed. The second problem is to assign sectors of the air space to the controllers. Until the next break, the set of sectors each controller is responsible for should change as little as possible. Related to this problem, there are situations where it is beneficial to have plans for both optimistic and pessimistic estimates for the demands, as a result a related problem is to determine which shifts should be given time off when the demands are closer to the low estimates. The last problem is to assign shifts to employees. This problem has a longer time span, and is ultimately where one really discovers how many employees must be part of the staff to meet the requirements related to how much time off each employee needs before they can work another shift, and other requirements like which time of day the em-

ployees need to work. The focus of this thesis is the first two of these problems. This is because their complexity seems to be lower than the third, which means that exploring solution strategies will be more likely to yield results that can be useful.

Chapter 2

Problems

2.1 Shift assignment

Diving a full 24 hour day into equal segments, e.g. 30 minutes, The goal of this problem is to create a schedule of shifts, where there are a sufficient number of shifts that are scheduled to work at each segment. Table 2.1 has a list of parameters that can be part of the input to this problem.

Element	Description
Demand	List of necessary shifts for each segment
Duration	Specifies the minimal and maximal duration of shifts
On	Specifies the minimal and maximal duration of work periods
Off	Specifies the minimal and maximal duration of break periods
Night	Details about night shifts, when they start, end, how many normal shifts can overlap them and for how long
Admin	Details about admin shifts, when they can happen, how much normal work they can include
Ideal Shift Length	The ideal duration of shifts
Unpopular times	List of segments that should be avoided as a time to start a shift
Padding	List of segments where there should be a given number of shifts on break
Overstaffing	Whether to allow more staff than necessary
Run time	Specifies how long the algorithm is allowed to run, in total and after improving the solution
Prearranged	List of ready shifts to be included in the schedule

Table 2.1: Shift problem parameters

Due to strict rules, there are regulations on shifts regarding work schedules for air traffic controllers. This manifests itself in the input in terms of rules for duration of the overall shift, as well as breaks and work periods. On average, the shift duration should be about the same as other jobs. Night shifts have no breaks and last from about 10pm to 6am, so schedules include the night shifts started on the previous day. An overlap from night shifts into day shifts, or vice versa, can be allowed. What this means, is that a given number of shifts last into the next period. The purpose of this is to allow controllers to brief each other about the situation. For problems that are hard to optimize, some shifts can be converted into so called

admin time. What this means is that after doing regular work for a given number of segments, the rest of the shift consists of office work. This work is not related to the problem, it is just a way to avoid excessive breaks or having more than the required number of controllers. Admin time can only happen after normal work, because it is important that the controllers have a clear state of mind when on duty, and it might be confusing to work with future rules and regulations in the office and then translate into working with the current ones. The given demands are estimations of actual demands, and sometimes it it necessary to have people ready to work if the need arises. This is done via the padding, which specifies that a given number of workers should be on break at the times specified.

The goal of the problem is to find a schedule that uses as few shifts as possible. Furthermore, efficiency should be as high as possible, which means to avoid longer breaks than necessary. The shift durations should be as close to the ideal as possible. Overstaffing should be avoided if it is allowed. Shifts that start early, should be short. Additionally, it is preferred to avoid having multiple shifts start working periods at the same time. If they do, there will be more noise in the control center as a result of having multiple people coordinate the exchange of work duties.

2.2 Sector assignment

For all units of work defined in the schedule from the shift assignment the worker will be assigned one or multiple sectors of air space. Sectors are administrative units of the air space, controlled from the control center. The schedule must always cover all of the sectors. For the interests of this thesis, the sectors are already grouped together in combinations, and for each time slot in the schedule, there is a given configuration, which specifies which combinations are to be used. The input for this problem consists of this information about the configurations, as well as the work schedule. Furthermore, the input specifies for each sector, any potential qualifications the controllers need to have for them to be allowed to control that sector.

As long as controllers do not have breaks, they should continue working in the same sectors to the highest degree possible. It is not allowed to go from working in one combination directly into some other combination that has none of the same sectors as the first combination. Shifts should have as few qualifications as possible. This is achieved by avoiding placing sectors with different qualifications in the same shift. The reason for this is that some controllers do not have all the

qualifications, but they still have a need to work in as many sectors as possible to stay refreshed on the rules and regulations that apply there. By scheduling the same qualifications to the same shifts, the number of different shifts each worker can be assigned to is kept as high as possible. Additionally, after breaks, the sectors should be different from before the break. This reduces repetitiveness and allows the controllers to work in multiple sectors in the same shift.

2.3 Bystander problem

There can be varying estimates for the demands to the given problems. One way to deal with this is to use so called bystanders. This means that controllers who have been assigned work can be classified as bystanders. They are effectively given a break, but could get called in to work. The coordinator at the given time can use this schedule to know who to call in and who can be given a break if the level of demand is different from what was expected.

Input to this problem is a schedule outputted from the sector assignment as well as lists of high estimates and low estimates for how many workers are needed for each time slot. From this, there is a given number of workers that should be assigned as bystanders. Ideally, bystanders are assigned to the same worker for consecutive time slots when possible. The end of or start of working periods is also the best time to be assigned as bystander, because assigning bystanders in the middle of a working period would create the hassle of changing sector assignments more times than otherwise necessary. Lastly, shifts that are longer or closer to the middle of the day are also good candidates for the bystander assignment.

Chapter 3

Implementation

As part of the scheduling done at Avinor, they use computer systems to aid in the creation of plans. The system responsible for solving scheduling problems, such as ones discussed here, is written using the programming language called Java. Because of that, the algorithms that have been developed are written in Java. The input to the problems comes in the form of a string in JSON format. [2] This allows for a high degree of generality.

3.1 Shift problem

The problem is solved using a branch and bound algorithm. [3] A recursive function is used to solve the problem after choices have been made. There is a score associated with the solution, to indicate how well it satisfies the goal of the optimization, and whether it is a legal solution. In the context of this implementation, a choice is defined in the creation of a shift. The shifts are planned in chronological order, which means that for each time unit of the problem, and for the duration of the shift, it needs to be scheduled as working time, or break time. Ending the shift is another option that mus be evaluated as part of these choices. In cases where a working period or break period is required to be longer than one unit of time, there is no choice other than to let these working periods or breaks last for at least their minimal duration. As such, there is not necessarily a choice to be taken for all time slots for each shift. If at some time slot, all the shifts in the schedule have been assigned to something, but there is still unsatisfied demand, then a new shift is created. This is not a choice. However, in the case where overstaffing is allowed, then it is considered a choice to open a new shift.

The score for the solution is built along with the solution itself. The concept of branch and bound is to have an optimistic bound for how good the solution can potentially be. In the case where the solution built so far has a worse score than the best solution that has been found, then no further evaluation of that sub-problem is necessary. The scoring mechanism is adding a penalty score based on different properties, and the goal is to find a solution that has the lowest possible score. An important property for the scoring mechanism is that the score can only grow. This means that the score can be used as a lower bound for the best possible score of the given sub-problem.

The first thing this implementation does is to assign the night shifts at the end of the day. Normally it would assign shifts with chronological starting times, but this is an optimization that helps the branch and bound algorithm. Due to the number of total shifts being the most important goal of optimization, there is a big score factor associated with opening new shifts. The night shifts are always going to be added, and there is little to no choices to be made regarding night shifts. If these shifts had been assigned last, that would have meant that a relatively big but mandatory part of the final score is added at the end. When the algorithm has found a solution with a given number of shifts, and is evaluating other schedules with the same number of shifts, that could be more optimal due to higher efficiency etc, it should not be necessary to only discover that the solution is worse after the final night shift is added to the score. This is why the night shifts at the end of the day are defined first.

The main body of the recursive function is a for loop which iterates over the time slots of the problem. For reasons we will come back to, the first time slot is given as an argument to the function. It iterates until it reaches the point where only night shifts are allowed to work. This is defined as the start of the night shift plus the duration of the overlap between day shifts and night shifts. Should the problem include any prearranged shifts, those that start at the given time are added to the schedule at the beginning of this loop body. Then there is a while loop, which runs for as long as there are are shifts that have been created, but not assigned anything at the given time. These shifts are iterated over, and they are checked for what they can do. If a shift cannot do anything at the given time, then the problem is in an invalid state, and the function can return with an invalid result, which means that it return the biggest possible value for the int datatype. If the shift is only allowed to do one thing, then it can be scheduled to do that right away, along with any similarly forced options for subsequent time slots. Otherwise, a choice has to be made. The choice that is being made is done on the shift which has the fewest options.

When making a choice, for each option, a duplicate of the problem object is created, This duplicate of the problem is acted upon instead of the problem given as a function argument. The given shift is scheduled according to the option that is currently evaluated. Then, if the shift has none or one option for any subsequent time slots, this is taken care of right away. Then the recursive function is called with the given candidate as input. The score built up for the problem is another function argument. The function will return the score given to the best solution to the problem given all the decisions made up to that point. If this value is lower than the best score found, then it needs to be stored. The best found score is the last function argument. If the current score of any problem is higher than the best score, then that current problem can be ignored, either by returning or continuing the relevant loop. Since relaxing all the options includes solving the rest of the problem, the function is done after this has been done, and can return the relevant score. The actual best solutions is kept by letting the problem given as an argument inherit all the relevant information from the one that produced the best score.

However, there is more to this. In cases where all the shifts have been assigned to something, but there is still demand that has not been met, new shifts must be opened. Furthermore, if the problem fails because the demand could not be met, and it is a time where new shifts can not be opened, either because it is too close to the end of the day, or because it is a time specified as an unpopular start time, in those cases it might be necessary to open shifts that cause overstaffing immediately. This is another type of choice. The options here are the number of such extra shifts to be opened. The highest demand throughout the day is used as an upper bound. This is still within than for loop over the time slots, and this is the absolute last decision to be made about that time slot, so when the function is called recursively from this point, the function needs to move on to the next time slot. This is achieved by the function argument that specifies where the for loop starts. After the for loop, the only thing left to do is to check that any remaining shifts that are identified as not being ended, are allowed to end.

To keep track on the score used to evaluate the solution at any point during execution, the current score is passed as an argument. The argument variable is used directly in the code to keep track on any additions. When evaluating choices, each candidate starts with the current score, and any additions to the score based on the choice taken is kept in a separate variable, which is passed to the recursive function. The scores themselves are a combination of different factors. With the most weight, is the number of shifts. This weight is added every time a new shift is added to the schedule. Overstaffing is penalized with a smaller weight, however,

bigger overstaffing of bigger sizes are penalized more. The weight is multiplied by the sum of the squares of the overstaffing. The algorithm does this during the schedule creation by adding the nth term from equation 3.1 when opening a shift, or scheduling work to some shift results in the number of shifts with work at the given time is n bigger than the demand at that time.

$$a^2 = \sum_{n=1}^{a} (2n - 1) \tag{3.1}$$

To avoid having multiple working periods starting at the same time, a check is made before opening a new shift or assigning work to shifts that did have break or could have had break previously. If the assignment of work in this action means there will be more than one working period starting at that time, the corresponding weight is added to the score. To avoid letting early shifts get long, after any shifts gets assigned anything, the algorithm checks if it is an early shift, which means that it starts at 7am or earlier, which is typically an hour after the end of the night shift. If the duration of such shifts have been increased to above the median allowed shift duration, increase of the duration shift beyond this is multiplied with the corresponding weight and added to the score. To optimize for efficiency, which means to avoid breaks, each unit of break added to the schedule a weight to the score, this is the lowest weight.

3.2 Sector problem

This problem is also solved using a branch and bound algorithm. In this problem, the choices that represent branching paths in the search space are the choices regarding which combination of sectors are assigned to which shift. Because shifts are not allowed to do completely replace the sectors they are assigned to from one time slot to the next, the search space can be reduced by quickly eliminating illegal assignments.

The function, which is recursive, has a for loop that iterates over all the time slots in the schedule. There is a while loop which repeats itself as long as the number of combinations that are unassigned is greater than one. Because when there is only one unassigned combination, that combination can be trivially assigned to the only shift that does not have a combination assigned to it for that time slot. Of course that is assuming that this assignment is legal. To make actual choices, the algorithm iterates over all pairs of shifts and combinations, to evaluate the number of sectors that are different between the combination assigned to on the previous

slot and the candidate combination. Shifts that did not work during the previous time slot are skipped in this part. Whichever pair has the lowest difference will be greedily used, but only if the difference is lower than a given number, like three sectors. When combinations are imagined as sets of sectors, this difference is equal to the size of the symmetric difference between these sets. However, if the size of the intersection of the sets is equal to zero, which means the sets are disjoint, then the difference is said to be infinity. This is a property that is used because complete change is not allowed. Notice that in cases where the difference is zero, which means, the same combination is used two times in a row, if the worker that was assigned that to that combination previously, and continues to work, they must also be assigned with that combination in the following time slot. This is repeated until no more pairs have a low enough difference. The function is called recursively to solve the rest of the problem given each possible assignment of the first combination.

The scoring system for this problem uses three factors. For each assignment, there is a potential value that is added to the score. Any new qualifications that get added to the shift will result in a score increase. Which means, if the combination that is being assigned includes any sectors where qualifications are required, if any of those qualifications have not already been added to the shift, then the number of qualifications is increasing with this assignment. The penalty from qualifications is using a harmonic sum, see equation 3.2. This means that each subsequent new qualification gives a smaller increase to the score. The score for this problem is a floating point number because of this. The other parts of the score depend on whether the shift was working during the previous time slot or not. If they were then the difference in sectors discussed earlier is used. Otherwise, a similarity of sectors is used. These concepts are opposites of each other. The point is that after breaks, the shift should be assigned different sectors that differ from those they were assigned to before the break. If they continue working, they should ideally continue working in the same sectors. Each of these parts of the score are multiplied by respective weights. Before any assignments take place, the score adjustment is calculated, which is used for checking if the assignment is legal, and if the new score is higher than the best score.

$$\sum_{n=1}^{N} \frac{1}{n} \tag{3.2}$$

3.3 Bystander problem

Bystanders are assigned using a greedy algorithm. For each time slot, bystanders may be necessary. If they are, they are assigned according to four parameters. If the shift was assigned as bystander during the previous time slot, then this shift is most likely going to be selected as a bystander. Next, if that time slot on the shift is right next to a break, then that is the second highest priority. Should there be a demand for bystanders to be assigned for the all the time slots until the shift has a break, then there is considered to be a break next to the work unit that is potentially converted to bystander time. The first two factors exists so that the work does not get more fragmented than otherwise required. Should someone get a work period interrupted by bystander time in the middle, they would have to coordinate the change of sector combinations one more time than otherwise necessary. The last two factors are the shift duration and distance to midday. The distance to midday is calculated using the middle of the shift and comparing that to the middle of the whole schedule. Longer shifts are more preferable, and then shifts closer to the middle of the day are used to break ties.

Chapter 4

Problem solution

In essence, a solver module has been created, which can solve three kinds of problems. Two of them are solved using branch and bound algorithms, while the third is solved using a greedy algorithm. In this section, examples of solutions to given problem instances are presented.

4.1 Shift Assignment

A small and simple problem that can be used to demonstrate the problem is presented as follows.

- Demands: 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
- Duration
  - Min: 12
  - Max: 18
- Work period duration
  - Min: 2
  - Max: 4
- Break period duration
  - Min: 2

  - Max: 3

- Night Shift
  - Morning
    - End: 12
    - Overlap: 1
    - Count: 1
  - Evening
    - Start: 44
    - Overlap: 0
    - Count: 0

- Overstaffing: false

This problem has been solved using the following parameters for penalties.

- Number of shifts: 1000
- Overstaffing weight: 100
- Stagger weight: 10
- Early shift too long weight: 10
- Break weight: 1

Note that since the problem input specifies that overstaffing is not allowed, the corresponding weight is irrelevant for this instance.

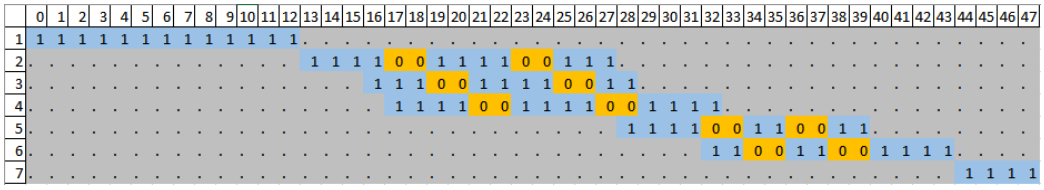

Figure 4.1: Shift schedule solution

Figure 4.1 shows the solution to the problem. This solution has a score of 7020, and the algorithm terminated after about 17 seconds. In the process of

finding this solution, which is the most optimal one, 15 other valid solutions were also found, with scores ranging from 8240 to 7021.

Changing the input, such that working periods are allowed to last between one and four units of time, instead of two and four like specified above, changes the performance of the algorithm. The solution is the same. Now, instead of 15 other solutions, the algorithm discovered 12 other solutions before finding the best one, and the worst of these has a score of 8130, which is a better solution than the first one found previously. All that really means is that in the order the algorithm makes choices, there is a solution where a shift has a working period that lasts for only one time unit, which has a better score than the the first valid solution that is found using the same search order, but when the only difference is that working periods need to be at least two units long. However, another important fact about this change of the input, is that the running time was longer. With this input, the algorithm spent about 55 minutes before terminating.

An alternative change to the input has similar results, this time allowing overstaffing while the work periods remain constrained to be at least two units long. A total of 19 valid solutions were found, where the best one is still the same one presented earlier. The first one has a score of 8430. Also worth noting is that the second solution has a score of 8241, which at first glance looks very similar to the score that the first version of this problem got, the score is only one point higher. However, that does not mean that the solution is similar. The fact that the first problem found when avoiding overstaffing had a better score means that this solution, as well as the one with the score of 8340, contain overstaffing. The running time for this input was around 95 minutes.

As we can see from this experimentation with the input parameters, even a small problem can increase in complexity by orders of magnitude. For reference, some of the problem instances Avinor deals with can have two to three times as high demand during the day compared to this example. This means that the number of shifts required in those schedules might be multiplied by similar factors. For big problems, this algorithm will take an unreasonable amount of time to finish, and thus it would be more appropriate to use other solution strategies.

Figure 4.2: Sector assignment solution

4.2 Sector assignment

Figure 4.2 shows the solution given from the sector assignment problem for a particular input set. The input to this problem includes a shift schedule which corresponds to the output with work and break times indicated. In the output, break times are indicated in the same way as time slots outside of the shift. The rest of the input concerns the sector set and configurations. The sectors in the sector set are identified using the following numbers. For this problem, each sector has its own qualification.

- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 20
- 21
- 22

The sectors are grouped together in combinations, which are grouped together in partitions. All sectors are present in exactly one combination within each partition. The partition can be represented as follows.

- C1
  - ALLO
    - 9, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, 22

- C2
  - 9T2EB
    - 9, 10, 11, 12, 20, 21
  - 3T7S
    - 13, 14, 15, 16, 17, 22
- C3
  - 9T2EB
    - 9, 10, 11, 12, 20, 21
  - 34
    - 13, 14
  - 567S
    - 15, 16, 17, 22
- C30
  - 9T2
    - 9, 10, 11, 12
  - 3T7
    - 13, 14, 15, 16, 17
  - OEBS
    - 20, 21, 22
- C40
  - 9T2
    - 9, 10, 11, 12
  - 34
    - 13, 14
  - 567
    - 15, 16, 17
  - OEBS

        * 20, 21, 22

- C4W0
    - 902
        * 9, 10,12
    - 3467
        * 13, 14, 16, 17
    - 5
        * 11, 15
    - OEBS
        * 20, 21, 22
- C5WS
    - 902
        * 9, 10, 12
    - 3467
        * 13, 14, 16, 17
    - 15
        * 11, 15
    - OEB
        * 20, 21
    - OS
        * 22
- C5S
    - 9T2
        * 9, 10, 11, 12
    - 34
        * 13, 14
    - 567

        * 15, 16, 17
    - OEB
        * 20, 21
    - OS
        * 22

- C5W0
    - 902
        * 9, 10, 12
    - 34
        * 13, 14
    - 67
        * 16, 17
    - 15
        * 11, 15
    - OEBS
        * 20, 21, 22

- C6WS
    - 902
        * 9, 10, 12
    - 34
        * 13, 14
    - 67
        * 16, 17
    - 15
        * 11, 15
    - OEB
        * 20, 21

- OS
  - 22

The input indicates at which times each partition is to be used, this is indicated on the output displayed on figure 4.2. When solving this problem, the algorithm parameters are as follows: Weight for qualification contribution is 10, the weight for the difference in sector in subsequent work slots is 10, and the weight for similarity before and after a break is one. The maximum difference for which the algorithm will make the greedy choice is three.

The algorithm terminated after about one second. The given solution was the $15th$ solution found by the algorithm, and it has a score of 676.326118326118. Meanwhile, the first valid solution had a score of 753.0559163059161. Because all the sectors have exactly one qualification on them, the number of qualifications required by each shift in such a solution is equal to the number of unique sectors the shift has been assigned throughout the schedule. Table 4.1 has a rundown of how many qualifications each shift has in this solution.

Shift	Qualifications
0	12
1	8
2	7
3	6
4	7
5	9
6	4
7	5
8	9
9	5
10	7
11	7
12	12
13	9
14	6
15	12

Table 4.1: Number of qualifications for each shift

Reducing the threshold for the greedy choice to one, such that the only assignments that are chosen greedily are those that have exactly the same combinations

as the preceding ones, changes both the running time and the solution. Now, the optimal solution has a score of 673.1158008658006, and this is now the $61st$ valid solution found by the algorithm. The first one had a score of 906.2622655122653. The algorithm terminated after about 35 seconds.

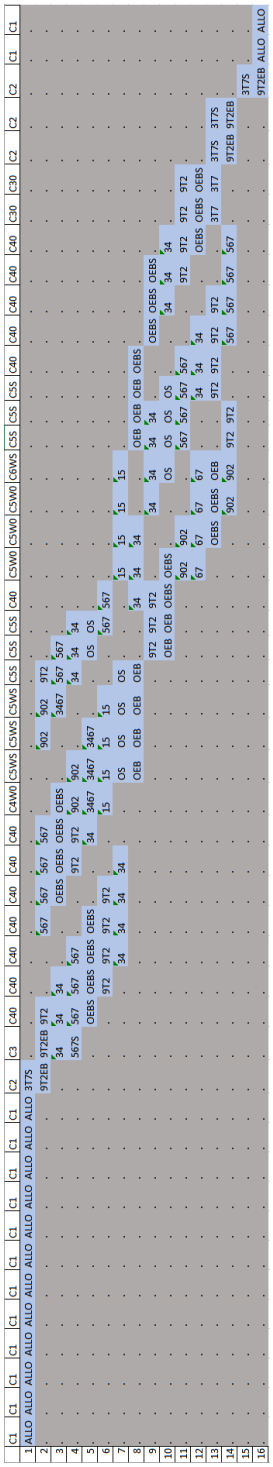

Figure 4.3: Improved Sector assignment solution

Figure 4.3 shows the new schedule. The difference stem from an early choice that was made differently, which propagated through the schedule by swapping corresponding combinations. In particular, in the time slot where the partition $C3$ is used, the shifts that ere assigned with the combinations 34 and $567S$ have had their assignments swapped. Note how in the third shift, in the last working period, in the new schedule, the combination changes from 3456 to 567 instead of 34. The latter of which has a difference of two, and would have been chosen with the more forgiving threshold for the greedy choice. The difference in terms of score in the new schedule, is that it receives about 6 points more in terms of qualifications and one point more in terms of sector similarity. In terms of sector difference, it receives 10 points less, which in total accounts for the about three points lower score compared to the first alternative. If the greedy threshold is set to two instead, the same solution would still be found, but in about four seconds, and the algorithm would have found 21 other solutions before the most optimal one.

4.3 Bystander assignment

The input to the bystander problem is effectively the output schedule from the sector assignment problem, along with arrays which hold information about how many shifts that should get their work converted into bystander time. Figure 4.4 shows the solution to one such problem. The creation of this solution took about 60 milliseconds.

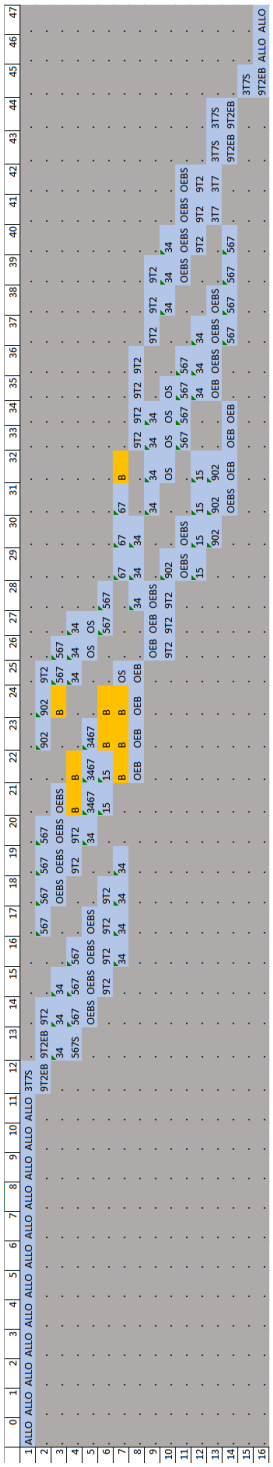

Figure 4.4: Solution to bystander asignment

Chapter 5

Discussion

5.1 Shift Assignment

Complexity

The search space for this problem is very big. As an example, we can examine a typical problem instance. The shifts are required to have a duration between 12 and 18 slots, work periods are required to last between two and four slots and breaks are required to last either two or three slots.

Ideally the shift should be as efficient as possible, which means that it should have as many units of work as possible relative to the number or break units. It should not be hard to see that for shifts that last between 12 and 16 slots, they are required to have at least four slots of break. Furthermore, in shifts longer than 16, there can not be more units of work than 12, which is already possible for a shift with a duration of 16. Effectively, since efficiency is to be optimized, the longest shifts only happen in the interest of adjusting the working periods to fit the schedule, not to allow more work. Meanwhile, shifts with durations between 12 and 16 can increase the efficiency in cases where this allows fewer shifts to be used. Tables 5.1, 5.2, 5.3 and 5.4 are examples of shifts with a minimal number of break units for different shift durations.

x	x	x			x	x			x	x	x

Table 5.1: Example of 12 duration schedule with four break units

x	x	x	x			x	x	x	x			x	x	x	x

Table 5.2: The only 16 duration schedule with four break units

x	x	x	x				x	x	x	x				x	x	x	x

Table 5.3: Example 1 of 18 duration schedule with six break units

x	x	x			x	x	x			x	x	x			x	x	x

Table 5.4: Example 2 of 18 duration schedule with six break units

A shift with a given duration and a given number of break periods can be scheduled in many different ways. Table 5.5 is an illustration of this for shifts of duration 12 with two breaks. In essence, the shift is incomplete, and there are still some units of work and break that can be distributed. The working periods that can be expanded are marked with $?$ and the breaks that can be expanded are marked with $\%$. Each working period can only hold up to three more units of work. And the slots for breaks can only hold one additional break unit. In this schedule, seven time units are mandated, and the rest need to filled in around those. Between five and eleven units must be added.

x	?	?	?			%	x	?	?	?			%	x	?	?	?

Table 5.5: Possibilities of 12 duration schedule with two breaks

This problem can be generalized as the following. Given n unique bins each with a capacity c_i, how many ways are there to distribute k balls between these bins. Note that bins are allowed to have separate capacities, which is useful in our case because both work periods and break periods have variable durations. This can be solved using dynamic programming.[4] Given a list of capacities, and a number of units to distribute among those capacities, the number of ways to achieve this can be solved recursively by placing each possible number of units into the last container, and then for each of these possibilities, the problem is solved recursively with the corresponding reduction of the number of units and the sub-list of the capacities that excludes the last one. The result is the sum of each of these recursive sub-problems. The default cases of the problem are as follows: First, if the number of units to be distributed is equal to zero, then the answer is

one. Second, if the sum of capacities in the containers is less than the number of units to distribute, then the answer is zero. Third, if the sum of capacities is equal to the number of units, then the answer is one. Fourth, if there is only one container, then the answer is one. If none of these apply, then the answer uses recursion. Dynamic programming is used to compute and store the solution to all sub-problems before they are used. In my specialization project that I worked on during the fall of 2016[1], the same problem was a central piece of the computation. Through experimentation, it was shown that the number of configurations in this problem scales exponentially in terms of the number of bins.

Table 5.6 is used for this purpose to find the number of ways to schedule a shift with two breaks. Because seven units are spent on the structure of having two breaks, the shift has a remaining number of units ranging from five to 11 that can be spent on additional work units or break units. What that means is that in the presented table, there are multiple cells that hold numbers of legal shift configurations. The cells in the rightmost column in the rows labeled 5 through 11 each hold numbers of unique legal shifts. In other words, there are a total of 174 ways to structure a shift with the given rules that has two break periods. Of course there are ways to have more breaks in a shift, which means that to solve the problem for all shift configurations, the table must be extended. Note that when expanding for one more break period, additional time units get bound to the structure of the shift, so fewer units can be used in legal shifts. When there are three breaks, the rows in which the legal shifts reside are 2 through 8. Which would equal up to 1314. For four breaks, the number is 1132. And finally for five breaks, the number is 73. So in total, there are 2693 ways to schedule one shift.

[1]Project title: Solving Thrill Digger, a version of Minesweeper

Units	work	break	work	break	work
0	1	1	1	1	1
1	1	2	3	4	5
2	1	2	5	8	13
3	1	2	7	12	25
4	0	1	7	14	38
5	0	0	5	12	46
6	0	0	3	8	46
7	0	0	1	4	38
8	0	0	0	1	25
9	0	0	0	0	13
10	0	0	0	0	5
11	0	0	0	0	1

Table 5.6: Dynamic programming table

As if the number of shift structures was not enough, we must also take into consideration the starting time of the shift. In simple term, the shift can be scheduled in all the different ways it can be structured, and it can start at any legal starting time. Besides the times of day reserved for night shifts, and any potential times defined as unpopular, most of the schedule remains open as starting time for each shift. Granted, any shift will only start on the first slot where there is demand for more work. So to define a shift completely, we we have to look at the remaining demand to set the starting time and then shape the shift. Unless the demand is low enough, there is a high degree of freedom for each shift. The complexity of creating a full schedule must be said to be exponential in terms of the number of the number of shifts that are necessary. The base of this exponential would be the number of shift structures, which can already be said to be exponential in terms of the number of periods the shift consist of, where this expression in turn uses the number of options for the periods as the base.

In terms of spatial complexity, the algorithm duplicates the partial solutions for every choice. This means that when the algorithm search is in one particular place in the search tree, all the parent nodes in the tree will hold a copy. At most, the number of copies that need to be stored in memory at the same time will be equal to the maximal depth of the search tree. The height of a tree is logarithmic in terms of the number of nodes. Since we have established that the number of nodes scales exponentially, then the height of the tree scales linearly.

Algorithm properties

Although overstaffing is generally allowed, it is an undesired property and most optimal solutions do not feature overstaffing. Disallowing overstaffing completely is an option that is allowed through the problem input, and doing so might change both the solution to the problem, and the search space. For the outlined algorithm, doing so reduces the search space significantly. By pruning choices that lead to any overstaffing immediately, the algorithm does not need to search deep enough to accumulate the score required to see that the overstaffing was not desired, alternatively it start searching for the more optimal solutions earlier. If the solver is given limited running time, then for complicated problems, there might not be enough time to find solutions without overstaffing, so allowing it in those cases would be a possible way to produce any solutions.

Other algorithms might not see the same benefit of disallowing overstaffing. In local search algorithms, one finds a feasible solution, and then perform a series of improvements until a local optimum has been found. Restricting the search space for such an algorithm is going to increase the difficulty of finding a first feasible solution, and reduce the feasible neighborhood.

5.2 Sector Assignment

Complexity

The algorithm used to solve the sector assignment problem also scales exponentially, it is similar to the algorithm that solves the shift assignment problem. However, the number of choices is different. Since the input to the sector assignment includes the output of the shift assignment, one can say that the complexity of the sector assignment exists within the provided shift schedule. Again, the complexity scales exponentially in terms of the number of choices that need to be made. A choice happens in two cases. The most prominent one is when two or more working periods start at the same, which is what is called stagger in the shift assignment problem. The other opportunity for choices to appear is when a sector combination is split during the working period to which it has been assigned. Effectively, when a controller is working with one combination, then if that controller is working during the next time slot as well. Then if the sector configuration that is used at that time has combinations such that the sectors the controller was working with placed in more than one combination, then this controller is allowed to work in any of these combinations.

The algorithm has a parameter that allows the second type of choice to be taken in a greedy fashion depending on sector difference. This can be used to shrink the search tree, at the possible expense of solution quality. Effectively, as long as there are choices that are better than a given level of quality, the best of these are are chosen greedily. How much this impacts the complexity depends on the sector configurations in the input. If the combinations change often between time slots, then the problem is harder.

Staggered starting of work periods are defined from the shift schedule. The presented algorithm deals with these by exploring all possibilities. At any point in the schedule where n working periods start at the same time, there are $n!$ assignments to these assuming that all other shifts can get combinations assigned to them using the greedy algorithm. So in terms of the problem complexity, stagger has an exponential contribution. Luckily, stagger is already a property that should be avoided when possible due to how it affects the working environment of the controllers. This means that the complexity of the problem is kept to a minimum.

Because sectors are assigned to the same schedule that was produced during shift assignment, and because these problems are part of the same problem pipeline, it is natural to compare their complexities. The choices in scheduling shifts can be said to happen at all time slots during shifts where the current period is greater than or equal to the minimum duration and less than the maximum duration for that kind of period, where the periods can be either working periods or break periods. Meanwhile, in the sector assignment problem, the choices can most prominently be attributed to the places in the schedule that have stagger, which only happens at the start of work periods specifically, and include more than one shift at a time. Substantial change in sector configuration can also lead to choices happening that include shifts in the middle of their working periods. As a result, the schedule has more opportunities for choices during the shift assignment phase rather than in the sector assignment phase. There is a factorial term in the complexity of the sector assignment problem. However, in comparison with the shift scheduling problem, this is likely only going to contribute for schedules with particularly large staffing demands. While for the shift assignment problem, the duration of the schedule remains the most important factor. This duration is more likely to change in terms of the resolution of the schedule rather than the actual duration of 24 hours, which has implications on the number of possible lengths of each variable period of the shift.

5.3 Bystander problem

The algorithm used to solve the bystander problem is greedy, which means that the solution is not guaranteed to be the best solution. However, to find a more optimal solution, one would have to define what that means. In using a greedy algorithm, one performs the choice which in isolation is the best one. That is, if the entire problem consisted of that choice, that choice would be the optimal solution. However, that can lead to a less optimal of the entire problem. As an example, the algorithm might choose to assign a bystander slot to the very last working slot of some shift, based on the fact that this would not be in the middle of a working period and the shift might be longer and closer to the middle of the day when compared to the other shifts. However, when there is also a bystander which needs to be assigned on the subsequent time slot, there might not be any shifts which can get this bystander slot without it being placed in the middle of a working period. If the first of these bystander slots were instead assigned to a shift that is just starting their work periods, then those two bystander slots could be assigned to the same shift, which is considered as good. Additionally, that assignment would then not be in the middle of a working period, which is also good. However, to properly define the best solution, there are many more cases that must be prioritized.

Chapter 6

Conclusion

In this thesis, I have presented three problems in the work scheduling pipeline at Avinor and algorithms that can solve them. These problems, called shift assignment, sector assignment and bystander assignment are related to each other because the output from one of these problems feeds into the next as input.

Shift assignment deals with structuring a schedule of shifts to meet given demands that vary throughout the day. There are also given rules to how shifts look, both in general and in terms of when they start during the day. A solution to this problem using a branch and bound algorithm has been presented. This is a solution strategy that is guaranteed to the find the most optimal solution if the algorithm has enough time to terminate. However, due to the complexity of this problem, the algorithm is likely to take too long to finish for bigger problem instances.

Sector assignment matches shifts with sectors of the air space such that all the sectors are covered at all times during the schedule. With the partitioning of the sectors being defined in the input to the problem, the main task in this problem is to decide which shift gets which combination of sectors. When a shift has been assigned to work in consecutive time slots, the shift must be assigned similar combinations such that the change is kept small. After breaks, the shift shift should be assigned to different sectors from before, but the third point is that most shifts should require as few qualifications as possible, qualifications being tied to the sectors. This problem is also solved using a branch and bound algorithm. In this case, the complexity seems much more promising, which indicates that branch and bound is a better fit for this problem.

Bystander assignment is about having schedules for multiple scenarios. The demands given to earlier problems are predictions, which might turn out to not be accurate. Therefore it is relevant to plan for change in demands. Typically this

happens by planning for high estimates for demands, and then assigning some shifts with so called bystander time. This means that whoever is on that shift is given break time, but can get called in to work if necessary. To solve this problem, a greedy algorithm has been used. This is a type of algorithm that gives a solution much quicker, but is not guaranteed to give an optimal solution.

6.1 Further work

The scoring mechanism for all of the algorithm can potentially be improved upon as Avior learns more about how the algorithms perform. Both in terms of the parameters like the weights given to different parts of the score, but there is also room to add or remove parts, as well as changing how they work.

Shift assignment

To enhance the performance of this algorithm, there are a couple of possible avenues. One option is to change the search order, not only in the presented structure, but instead of making decisions in chronological order, is is possible complete each shift before defining the next. By doing it this way, it would be possible to explore options for each shift in an order that is determined to be as optimal as possible. For example, it would be possible to explore all the options where the time spent on breaks in each shift is minimal before exploring local solutions that are less optimal. Next, it would be possible to front load more of the score. For example, the minimal number of break units in a shift is defined purely from the input, so any shift essentially contributes with this number number of break units by virtue of being a shift in the schedule. Once it it clear that an additional shift is necessary, this has implications on how many break units in total the schedule must have. Front loading information like this means that the effect of the branch and bound can be increased, since information that is not unique to one branch is discovered before the branching happens.

Sector assignment

This branch and bound algorithm also has the potential of front loading more information. When the partitions for subsequent time slots include the same sector combination that was assigned to a shift that is also working during those subsequent time slots, that assignment can be done straight away. This has the potential

of saving some computation, because decisions are made in chronological order, one such stretch of the same combinations could be interrupted by a choice regarding another shift. However, unlike in the first algorithm, this only saves duplicate work, and will not help prune solutions faster. The reason is that when shifts are assigned with the same combination as the preceding time slot, there is no penalty occurring.

Another opportunity would be in regards to the greedy parameter. Maybe running the algorithm first with a setting where more choices are made greedy, then after that run finishes, using the found score and a more strict threshold in another run, maybe time could be saved.

Bystander assignment

Like discussed earlier, this problem is solved using a strict greedy algorithm that uses a hierarchy of the desired solution qualities. To find better solutions, one option would be to implement a scoring system and solve this problem using a branch and bound algorithm as well. Additionally, the algorithm used for this problem is also running in chronological order. After the algorithm has found a solution, it would be possible to perform a local search which looks for stretches of bystander time that can be combined, which would mean that bystander time that was assigned early based on shift duration and how close it is to the middle of the day, those bystander slots can instead be assigned to the same shifts that are assigned bystander time at later times.

Bibliography

[1] Avinor, about the company. https://avinor.no/en/corporate/about-us/the-avinor-group/about-the-company. Accessed: 2017-05-10.

[2] Ben Smith. *Beginning JSON*. Apress, 2015.

[3] L. G. Mitten. Branch-and-bound methods: General formulation and properties. *Operations Research*, 18(1):24–34, 1970.

[4] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.