Genetisk algoritme til hospitals vagtplanlægning

Studenteropgave

Software P1-projektrapport

Gruppe

Software A312a

Dato

17. december 2020

AALBORG UNIVERSITET
STUDENTERRAPPORT

Første Studieår - Software
Strandvejen 12-14
9000 Aalborg
http://www.tnb.aau.dk

Titel:

Overskrift

Projekt:

P1-projekt

Projektperiode:

Oktober 2020 - December 2020

Projektgruppe:

A312a

Deltagere:

Emil Normann Brandt
Phillip Mørk
Mathias Rasmussen
Jens Emil Fink Højriis
Louis Nørdam
Hassan Hammoud
Muslem Mohamad Chikhu

Vejleder:

Mathias Ruggaard Pedersen

Sidetal: 72
Afsluttet: 17. december 2020

Abstract

The method used for schedule planning on hospitals is suboptimal and can lead to health problems, both physical and psychological. We have researched the scope of schedule planning problems on hospitals and how we can design and program an automatic scheduling planner in C, to combat these issues. The program is automated through a genetic algorithm, that is designed with crossover sections to mutate individuals from generation to generation, and incorperates both hard and soft constraints, to generate a fitness score for each plan being generated.

Forord

Skriv forord her

Indholdsfortegnelse

Kapitel 1 Definitioner **1**

Kapitel 2 Introduktion **2**

Kapitel 3 Problemanalyse **4**

3.1 Udfordringer ved vagtplanlægning 4

3.2 Arbejdsbelastning 5

3.2.1 Stress 5

3.2.2 Nattearbejdes effekt på sygeplejersker 6

3.2.3 Lange arbejdsdage 7

3.2.4 Døgnrytme 7

3.2.5 Arbejds- og privat-livsbalancen 8

3.3 Regler og love bag vagtplaner 9

3.3.1 Frokost pauser 10

3.3.2 Helligdage, FO-dage og søgnehelligdage 10

3.4 State of the art 11

3.4.1 Softwareløsninger 11

3.4.2 Manuel vagtplanlægning 14

3.5 Opsummering af krav til vagtplan 15

Kapitel 4 Afgrænsning og problemformulering **17**

4.1 Opsummering af problemanalysen 17

4.2 Afgrænsning 17

4.3 Problemformuleringen 18

Kapitel 5 Løsning **20**

5.1 Tekniske krav til programmet 20

5.1.1 Funktionelle krav 20

5.1.2 Non-funktionelle krav 21

5.2 Teori om løsning af vagtplaner 22

5.2.1 Genetiske algoritmer . . . 22
5.2.2 Tidlig konvergens . . . 23
5.3 Design af genetisk algoritme . . . 26
5.3.1 Valg af DNA . . . 26
5.3.2 Start population . . . 27
5.3.3 Mutation . . . 27
5.3.4 Generering af ny generation . . . 28
5.3.5 Crossover . . . 28
5.3.6 Evalueringen af vagtplan. . . . 29
5.4 Strukturering af programmet . . . 30
5.4.1 Data input/output . . . 31
5.4.2 Program funktionaliteter . . . 32

Kapitel 6 Implementering **34**
6.1 Datastrukturer . . . 34
6.1.1 struct Worker . . . 34
6.1.2 struct Schedule . . . 35
6.1.3 struct RequiredWorkers . . . 35
6.1.4 struct BlockSchedule . . . 36
6.2 Funktioner . . . 36
6.2.1 main . . . 36
6.2.2 make_schedule . . . 37
6.2.3 generate_initial_population . . . 40
6.2.4 evaluate_schedule . . . 42
6.2.5 Combine_Schedule . . . 48

Kapitel 7 Test af program **51**
7.1 Test af fitness funktion . . . 51
7.2 Test af genererede vagtplaner . . . 53
7.2.1 Valide vagtplaner . . . 55
7.3 Evaluering . . . 57
7.4 Tidstagning . . . 58

Kapitel 8 Konklusion **60**

Kapitel 9 Perspektivering **62**
9.1 Mulige tekniske forbedringer til programmet . . . 62
9.2 Autentisk brug af produkt . . . 64
9.2.1 Kultur på arbejdspladser . . . 64

9.2.2 Effekt af produktmangler 64
9.3 Refleksion over program 65
9.4 Brug af genetisk algoritme 65

Litteratur **66**

Bilag

Bilag 1: Programmets kildekode . . . . .
Bilag 2: Medarbejder liste A 74
Bilag 3: Eksempel på eksporteret vagtplan 75
Bilag 4: Individuel brugervenlig vagtplan 76
Bilag 5: Medarbejder liste B 77
Bilag 6: Bedste vagtplan lagt ud af de 1000 vagtplaner 78

1 | Definitioner

Igennem denne rapport bruger vi nogle termer, som kan være tvetydige. Derfor så definerer vi helt specifikt, hvad vi mener med disse termer her.

Hospitaler og sygehuse Vi bruger disse to ord som synonymer.

Hårde krav Hårde krav til en vagtplan er krav som skal overholdes af vagtplanen før at den kan være valid.

Bløde krav Bløde krav til en vagtplan er krav som ikke er nødvendige at overholde for at vagtplanen skal bruges, men det gør vagtplanen bedre hvis de bløde krav overholdes.

Valid vagtplan En valid vagtplan er en vagtplan, som overholder alle de opsatte hårde krav.

Optimal vagtplan En optimal vagtplan er en valid vagtplan, men som overholder så mange bløde krav, som er overhovedet muligt. Der findes derfor flere forskellige optimale vagtplaner, hvis de overholder det samme antal ønsker.

2 | Introduktion

Vagtplaner bliver brugt mange steder på arbejdsmarkedet og i mange forskellige former. Nogle eksempler på dette er vagtplanlægning på et hospital, på skoler og pilot-planer.

Vagtplanlægning dækker derfor over mange forskellige problemer og problemtyper, alle vagtplaner har det ultimative mål at de vil lave en optimal plan ud fra de kriterier der er blevet opstillet. De forskellige former for vagtplaner har mange fælles kriterier og mange unikke kriterier for den enkelte branche. Vi har givet et hurtigt overblik over nogle af disse kriterier i tabel 2.1.

Skibe	**Pilot-planer**	**Dagligvarebutikker**	**Hospitaler**
Skiftplanlægning	Lovmæssige forhold	Sæsonændringer	Uddannelse
Udmattelse og årvågenhed	Tid imellem afgange	Kvalificerede medarbejdere	Kompetencer
Lovmæssige forhold	Industri-forhold	Lovmæssige forhold	Døgnvagter
	Træning og kvalifikationer	Udlån af personale	Aftalte besøg
	Sprogegenskaber		Overenskomst

Tabel 2.1. Eksempler på vagtplanlægning i følgende brancher: Vagtplan på skibe (Australian government, 2020) (Baumler et al., 2020), pilot-planer (Kohl og Karish E., 2004), dagligvarebranchen (PDC, 2020b) og sundhedsbranchen (Danske Regioner, 2016) (Danske Regioner et al., 2018).

Vagtplanlægning er derfor en kompleks størrelse. Der er forskellige udfordringer og behov for hver virksomhed, og både størrelse og branche kan også have stor betydning for hvordan en virksomhed vælger at planlægge deres vagter. Virksomheder kan have forskellige filosofier og værdier, som både kan komme til udtryk i medarbejderne, men også i selve virksomhedens arbejdskultur og derved også deres vagtplaner. Dette ser man især også i hospitalsbranchen som vi kommer til at gennemgå senere i dette afsnit.

Vores fokus er vagtplaner på hospitaler. Hvis man ser på sygeplejersker, så kan man især se at der er en kultur for lange vagter. (Stimpfel og Aiken, 2013) I artiklen har de fundet ud af at den mest almindelige vagt for en sygeplejerske, er 12-13 timer. Dette påvirker kvaliteten af det job de udfører, deres helbred og patientsikkerheden. Vagtplanlægning på sygehuse er derfor et komplekst emne, som endnu ikke er helt løst. Der er mange ting som kan forbedres, såsom lange vagter. Udover dette så er der også mange forskellige typer af

medarbejdere, som skal være på arbejde bestemte tidspunkter af dagen. Så der er meget at tage højde for, når man laver et program til at generere en vagtplan.

3 | Problemanalyse

Det er svært at lave en valid vagtplan, da der er mange krav der skal tages hensyn til. Når vagtplanlæggere skal lave vagtplaner på hospitaler, så vil der være ønsker og krav fra personalet, som en vagtplan skal tage hensyn til, da der på sygeplejerskernes arbejdsplads indgår variabler, som natarbejde, stress og beredskabsvagter. Et hospital er et essentielt område i samfundet, derved er det vigtigt patienter har mulighed for at kunne blive behandlet 24 timer i døgnet. Derfor skal hospitaler være bemandet 24 timer i døgnet, men samtidig skal der leves op til kvalitetskrav som f.eks. Danmarks stramme overenskomster.(Overbjerg, 2019) Vi vil undersøge hvilke krav der kan være vigtige, når man skal sammensætte en vagtplan.

3.1 Udfordringer ved vagtplanlægning

At finde en valid vagtplan er ikke det samme som at finde en optimal vagtplan. En optimal vagtplan overholder alle kravene. Det kræver mange tusinde variabler, og meget avanceret matematik for at finde en god vagtplan matematisk.(Overbjerg, 2019) Det er fordi vagtplanlægning, er et np-fuldstændig problem (B. Cooper og H. Kingston, 1995), hvilket betyder at det tager meget computerkraft at lave en valid vagtplan og endnu mere computerkraft for at lave en optimal vagtplan. Sværhedsgraden af problemet bliver kun større når man tilføjer flere medarbejdere, krav og ønsker. Især i sundhedsbranchen er det derfor meget svært, da der er mange variabler at tage forbehold for. Derfor er vi nødt til at undersøge hvilke krav, der er mest relevante, sådan at vi kan minimere mængden af krav. Man kan betragte det som, at jo færre krav vi har, jo flere medarbejdere og ønsker vil det endelige program kunne håndtere. Når man laver en vagtplan, er det også vigtigt at tænke på medarbejderne og deres forhold. Både for deres velvære, men også for virksomhedens effektivitet. Det vil vi komme ind på hvordan man gør igennem resten af problemanalysen.

3.2 Arbejdsbelastning

Når der udarbejdes en vagtplan, så skal der sørges for, at medarbejderne ikke bliver overbelastet. Derfor skal der undersøges, hvilke krav der skal være til en vagtplan, for at medarbejderne har det bedst muligt, psykisk og fysisk. Det er det, vi vil undersøge i dette afsnit.

3.2.1 Stress

På et hospital skal der være bemanding 24 timer i døgnet. Dette betyder, at der altid er nogen, som skal arbejde om natten. Det gør man gennem skifteholdsarbejde, hvor hold af medarbejdere skiftes til at arbejde hele døgnet rundt. Når man arbejder skifteholdsarbejde, hvor arbejdstiderne ændrer sig ofte, skaber det psykosocialt stress. (Puttonen et al., 2010)

Psykosocialt stress er, når en person føler, at deres sociale status bliver udfordret. Det er altså en form for stress, hvor man hele tiden er usikker på ens sociale situation, og er bange for, at den bliver værre. (Kogler et al., 2015) Ifølge et studie omkring skifteholdsarbejde (Puttonen et al., 2010), så er der tre overordnede grunde, til at psykosocial stress kan opstå igennem skifteholdsarbejde. (1) Manglende kontrol over ens arbejdstimer. (2) Dårlig familie- og arbejdslivs balance. (3) For lidt tid til at komme sig efter en dårlig arbejdstime, som f.eks. natarbejde. I samme studie konkluderer forskerne at psykosocialt stress, i sidste ende, fører til hjerteproblemer. Psykosocialt stress er ikke det, som skaber hjerteproblemer direkte. Psykosocialt stress fører til andre problemer som overvægt og rygning, hvor disse problemer kan medføre hjerteproblemer. Generelt er stress bare noget som arbejdsgiverne helst skal undgå. Ifølge Det Europæiske Arbejdsagentur, stammer halvdelen af alle sygedage fra stress. (Det Europæiske Arbejdsagentur, 2020) Derfor er det vigtigt, at man så vidt som muligt prøver på at undgå de tre problemer gennem vagtplanen.

For at mindske stress, især for mødre, så skal en vagtplan være forudsigelig. Et studie (Lozano et al., 2016) undersøgte hvordan en abnormal vagtplan påvirker stress for mødre og fædre. En abnormal vagtplan kan f.eks. være med huller midt på dagen eller arbejde på andre tidspunkter der ligger uden for normen (hvor normen er at arbejde en gang efter morgen, og få fri før aften). I studiet fandt de frem til, at hvis man brugte en forudsigelig abnormal vagtplan, så kunne det mindske stress for mødre. De fandt ud af at mødre med en forudsigelig abnormal vagtplan havde 50% mindre chance for at være stresset. Hvis vagtplanerne ikke var forudsigelige, så var der kun 40% mindre chance. Studiet kunne kun

konkludere hvilken procentdel, der var stresset og hvilken procent der ikke var. De kunne ikke finde en sammenhæng mellem arbejdstiderne og mængden af stress. De fandt ud af at imens mødre fik mindre stress af abnormale vagtplaner, fik fædre derimod mere stress af det. Udover det kan vi konkludere at en vagtplan ikke nødvendigvis skal være indenfor normale arbejdstimer for ikke at stresse, og at forudsigeligheden af en vagtplan også er en vigtig faktor.

3.2.2 Nattearbejdes effekt på sygeplejersker

En undersøgelse omkring nattearbejde for sygeplejersker, har brugt et spørgeskema til at vurdere nattearbejdets indflydelse på deres sundhed. Resultaterne fra denne undersøgelse kan ses i tabel 3.1. Tabellen viser, at størstedelen af de deltagende sygeplejersker svarede, at de bruger stimulanser såsom koffein til at holde sig vågen, når de har nattevagter. Brugen af stimulanser kan have en negativ indflydelse på medarbejdernes sundhed. Derfor bliver brugen af stimulanser betragtet, som noget negativt ved nattearbejde. En af de største problemer, som blev konkluderet fra spørgeskemaet, var mangel på søvn, hvor størstedelen mente at de fik mindre end 6 timers søvn hver nat. Nogle af de værste sundhedsmæssige komplikationer som søvnmangel kan forårsage er angst, kognitiv svækkelse, svækket immunforsvar og fedme. Dette bliver understøttet af de spørgsmål, der var til sidst i spørgeskemaet, hvor 41% havde oplevet udmattelse og manglende søvn. Derudover havde 30% indikeret, at nattearbejde havde haft en negativ indflydelse på deres sundhed. (Brooks et al., 2020)

Variable	**Mean**	**SD**
I do not like rotating shifts because it affects my health.	3.17	0.94
I would rather work straight night shift schedule than rotate shift.	3.15	1.05
I use caffeine to keep myself awake at night.	3.15	1.25
I fall asleep withen the first 2 hours of returning home from a shift.	2.48	1.43
I get at least 6 hours of sleep each day/night.	1.85	1.29
I have fallen asleep at the wheel while driving home in the morning after night shift work.	1.74	1.32
I use sleep aids such as over-the-counter or prescriped sleep medication.	1.64	1.58

Tabel 3.1.
Ud fra spørgsmålene på venstre siden kunne sygeplejerskerne svare på en skala fra 1 til 5, hvor 1 er "Jeg er stærkt uenig", hvor 5 er "Stærk enig". "Mean"er den gennemsnitlige besvarelse og SD er standardafvigelsen. (Brooks et al., 2020)

Derudover for at bevare sygeplejerskernes arbejdsglæde og undgå stress skal ledelsen hjælpe dem ved at håndtere og planlægge pauser, og sørge for at de bliver overholdt. En løsning som er blevet præsenteret fra sygeplejerskers side, er at have en mere fleksibel vagtplan. Fleksibel vagtplan betyder i denne kontekst at vagtplanen skal afspejle de individuelle ønsker som medarbejderne giver og de ønsker afdelingen har. Denne fleksible vagtplan skal også give tid til pauser, så deres arbejde ikke går ud over deres helbred.

Derved minimerer man de menneskelige fejl der opstår hvis en sygeplejerske har arbejdet for meget eller sovet for lidt eller lignende. En løsning for at forbedre arbejdsforholdet ved nattearbejde, som fremlagt i deres rapport, er at forbedre kommunikation mellem sygeplejerskerne og deres umiddelbare overordnede, da det er blevet vist at f.eks. informative debriefinger hjælper sygeplejerskerne med at afstresse og forbedre deres arbejdsglæde. En debriefing i dette tilfælde betyder at man snakker med ens overordnede om de job relaterede oplevelser man har haft i løbet af dagen. (Savic et al., 2019)

3.2.3 Lange arbejdsdage

Når man planlægger en vagt, så er det svært at undgå lange dage for nogle medarbejdere. Det kan dog være skadeligt mentalt for medarbejdere og derudover laver man flere fejl når man er træt. (Wagstaff og Lie, 2011). En undersøgelse fra Stockholm: Scandinavian Journal of Work, Environment & Health, konkluderede at en arbejdsdag længere end 8 timer øger risikoen for fejl, og allerede ved 12 timer er der dobbelt så stor chance for fejl som ved 8 timer.(Wagstaff og Lie, 2011). Der sker også færre fejl hvis man arbejder udelukkende om natten i stedet for at have en vagt der går fra aften til nat.(Wagstaff og Lie, 2011). Derfor skal vi prøve at inkorporere disse elementer i en løsning for at mindske antallet af fejl.

3.2.4 Døgnrytme

Udover lange vagter kan det også være skadeligt for ens helbred hvis vagterne er dårligt placeret. Hvis ens døgnrytme bliver ændret for ofte, især roterende vagtskifte, kan det give problemer. Man har ofte roterende vagter på hospitaler fordi der skal være personale 24 timer i døgnet. Et studie sammenlignede 75.000 sygeplejersker (Gu et al., 2015), som viste, at når man arbejder i mere end 15 år med roterende nattevagter, så har man mellem 19-23% større chance for at få hjerte-sygedomme. Chancen for at få lungekræft stiger også med 25%. Derfor er det vigtigt at man laver nattevagter på den mindst skadelige måde. Dansk sygeplejeråd har lavet nogle anbefalinger, til hvordan man kan forstyrre døgnrytmen mindst muligt (Dansk sygeplejeråd, 2012). En af deres råd er at man med forskellige nattevagter, skal prøve at skubbe ens døgnrytme frem i stedet for tilbage, det vil sige i stedet for at arbejde (nat, aften, dag, morgen), så vil det være bedre at arbejde (nat, morgen, dag, aften). På den måde forstyrrer man døgnrytmen mindre. Derudover foreslår de også, at man kun har en eller to nattevagter i træk.

Det er vigtigt at tænke over, når man laver en vagtplan for at skabe færrest muligt

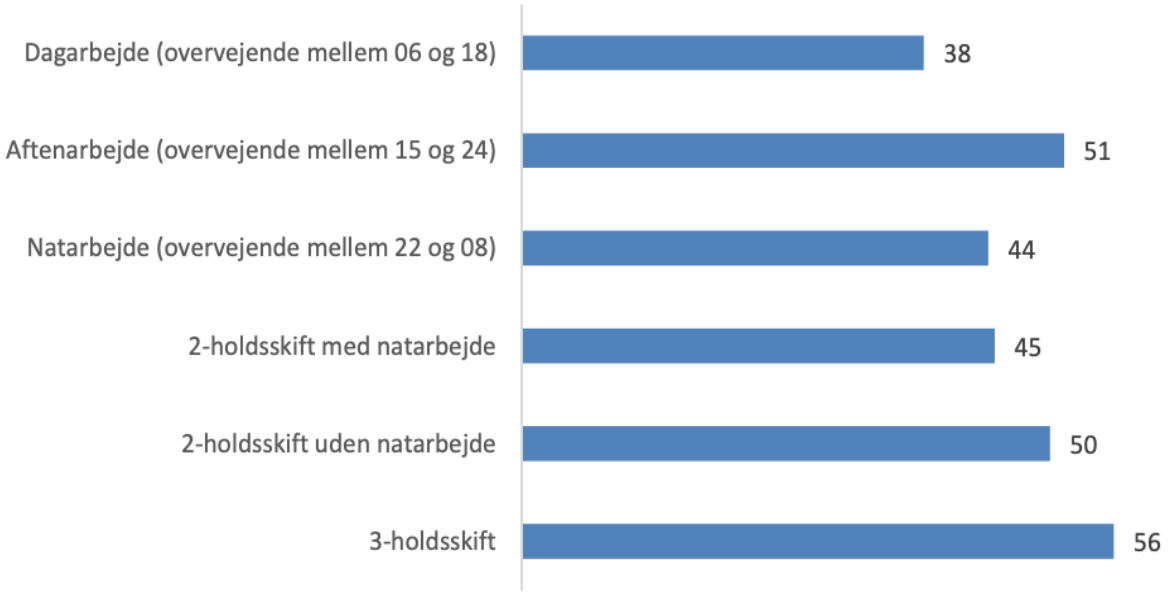
helbredsproblemer, for dem der arbejder.

3.2.5 Arbejds- og privat-livsbalancen

Rigshospitalet har oplevet konflikter med de 4.600 sygeplejersker der arbejder der. Grunden til konflikten er at sygeplejerskerne er utilfredse med deres lange vagtplan, og at deres overenskomst er blevet brudt. Sygeplejerskerne mener, at de skal have en ordentlig vagtplan, hvor en vagt hverken skal være for kort eller for lang. (Juulsager, 2020). Ifølge Dorte Steenberg fra Dansk Sygeplejeråd kan sygeplejerskerne ikke nyde deres privatliv, da fritiden ofte kan blive afbrudt af tilfældige vagter, som tvinger dem på arbejde. Dette skaber utilfredshed blandt sygeplejerskerne, da de hverken har indflydelse eller kontrol over deres skema.(Bille, 2017).

En undersøgelse fra dansk sygeplejeråd viser at det er især de unge, der har deres sværest med at få hverdagen til at hænge sammen grundet deres vagtplaner. Ca. 46 pct. af 30-39-årige sygeplejersker mener, at de ofte eller jævnligt har oplevet konflikt mellem arbejde og privatliv. I 2015 mente 36 procent af sygeplejerskerne på tværs over alle aldersgrupper, at de ofte oplevede konflikt mellem arbejde og privatliv grundet deres travle vagtplaner. (Bille, 2017)

Treholdsskift er vagter hvor medarbejderne arbejder i 3 hold hvor der er et hold om morgenen, et andet om dagen og et tredje om natten. En uge kan en medarbejder eksempelvis arbejde dag og en anden uge kan den samme medarbejder arbejde nat.(HK, 2020). Sygeplejersker oplever at vagter med fast aften, og treholdsskift skaber ubalance mellem deres arbejdsliv og privatliv. (Dansk sygeplejeråd, 2016b) Treholdsskift er problematisk, da det gør det sværere at planlægge sociale aktiviteter uden for arbejdet. (Holmboe, 2018). Dansk sygeplejeråd har lavet to undersøgelser i 2012 og i 2015 om sygeplejerskers psykiske arbejdsmiljø og helbred. Undersøgelserne er sket via spørgeskema, hvor der i 2012 var besvarelser fra 3.496 aktive danske sygeplejersker og i 2015 var der 2.212. Sygeplejerskerne blev stillet spørgsmål omkring balancen mellem deres arbejdsliv og privatliv, hvor de skulle vurdere i hvor stor grad det forårsagede problemer i arbejds-privatlivs balancen. Respondenterne skulle give hvert spørgsmål point imellem 1 og 5, hvor 1 er 'Jeg er stærkt uenig' og 5 er 'Stærk enig'. De fleste spørgsmål har 5 svarkategorier, hvor man i beregningen af en score har givet hver svarkategori henholdsvis, 100, 75, 50, 25 og 0 point. De har derefter udregnet den gennemsnitlige score af spørgsmålene for hver person og derefter gennemsnittet for alle respondenter. En score på 0 er lav og 100 er høj som betyder et belastende arbejdsmiljø. (sygeplejeråd, 2016).



Figur 3.1. Figuren viser balancen mellem arbejdsliv-privatlivsbalancen fordelt i arbejdstider. Målingerne er i point hvor højere tal betyder mere ubalance. (Dansk sygeplejeråd, 2016b)

På den måde er de kommet frem til figur 3.1, som viser at hvis man ser på arbejdstider, så oplever sygeplejersker med fast 3-holdsskift mest ubalance når det kommer til deres arbejdsliv-privatliv. Dagarbejde mellem 06 og 18 skaber mindst ubalance for sygeplejerskerne. Ud fra figuren 3.1 kan man helt klart konkludere at sygeplejersker mener at dagarbejde skaber mindre ubalance mellem arbejdsliv og privatliv. Resultater viser også at højere ubalance mellem arbejdsliv og privatliv, fører til mere stress og højere sandsynlighed for et jobsskifte. (Dansk sygeplejeråd, 2016b) Undersøgelsen viser også at sygeplejersker, som har mere ubalance mellem deres arbejdsliv-privatliv, har et dårligere selvvurderet helbred end dem med en bedre balance. (Dansk sygeplejeråd, 2016b)

3.3 Regler og love bag vagtplaner

Når man skal lave vagtplaner, er det vigtigt at den vagtplan man har lavet er valid. Det betyder at den lever op til alle de lov- og industrimæssige krav. Det er det vi har kaldt de hårde krav. Hvis planen bryder regler og love, så kan det give store problemer for arbejdspladsen, derfor kigger vi nu på de relevante regler og love.

Inden for hospitalsbranchen, især for sygeplejersker, er der mange regler, der inkluderer hviletids-, fridøgn- og afspadseringsregler.(Dansk sygeplejeråd, 2016a) Disse regler er lovpligtige, og skal derfor overholdes, men de er lavet så de er relativt fleksible. En god vagtplan skal derfor ikke bare overholde disse regler men samtidigt også prøve at opfylde

andre ønsker og krav.

Arbejdet som sygeplejerske omfatter også nattevagter, hvor at der også er specifikke krav og regler, som skal overholdes. Der skal føres en helbredskontrol af ansatte hvert 3. år, ens daglige vagter må ikke overstige 8 timer per. døgn gennemsnitligt i en periode på 4 måneder og arbejderne har generelt krav på mindst en 11 timers hvileperiode mellem hver vagt. (Forbundet af offentlige ansatte, 2020)

Disse regler medfører fleksibilitet, som tillader, at arbejdsugerne ikke er konstante, og at man kan have forskellige antal timer hver uge. Disse regler hænger sammen med den regel EU vedtog i 2003 der hedder "48 timers reglen". Reglen siger at hvis man går 4 måneder tilbage, så må man i gennemsnit ikke have arbejdet mere end 48 timer om ugen (Danskerhverv, 2019). Dette skal vagtplanen også tilrettelægges efter. Den ugentlige arbejdstid kan være længere end normalt i en periode, hvis det så bliver kompenseret for i de andre uger, så gennemsnittet ender med at være på de 48 timer. Man kan af den årsag godt ende ud med overarbejde, hvilket på den ene side gør vagtplanen nemmere at lave, og på den anden side har de førnævnte negative effekter.

Ansatte på hospitalet over 18 år skal have mindst et fridøgn for en 7 døgns periode. Det betyder i praksis, at personalet skal altid have fri en dag i løbet af ugen. Et fridøgn er en hvileperiode på 35 timer i stræk. Specifikt for hospitaler så kan fridøgn blive udskudt 5 døgn, siden hospitaler falder indenfor pasning af mennesker. Hviletiden kan også nedsættes til 8 timer i stedet for 11 timer to dage om ugen og ikke to dage i træk. (Arbejdstilsynet, 1997)

3.3.1 Frokost pauser

I forbindelse med frokostpauser er sygeplejersker, med en offentlig overenskomst, sikret en frokostpause på mindst 29 minutter. Et EU-direktiv er kommet med en melding som lyder at ”Pausen skal placeres inden for arbejdstiden således, at formålet med pausen tilgodeses,” (Kjeldsen, Susanne Bloch, 2005). Det vil sige at pausen skal ligge på et fornuftigt tidspunkt, hvor det er naturligt for medarbejdere at spise. (Kjeldsen, Susanne Bloch, 2005)

3.3.2 Helligdage, FO-dage og søgnehelligdage

Hvis en helligdag falder på en hverdag kaldes det en søgnehelligdag. Søgnhelligdage har nogle regler som skal overholdes når der skal lave vagtplaner. Hvis der er en søgnhelligdag

i ens normperiode, så skal ens normperiode nedskrives, så at man skal arbejde færre timer i den periode. Man skal arbejde det der svarer til en arbejdsdag mindre end normalt. I praksis vil det sige at man skal arbejde 4/5 af de timer i en uge med en søgnhelligdagdag. Hvis der er en søgnhelligdag i en normperiode, så har man også ret til en FO-dag hvilket betyder at man har friperiode på 35 timer fra man tager hjem fra arbejdet til man skal på arbejde igen. (FOA, 2020)

3.4 State of the art

Her vil vi gennemgå nogle af de løsninger og tiltag der er i brug i den danske sundhedssektor til at lave vagtplaner. Formålet med en redegørelse og analyse af state of the art, er at få en oversigt over hvilke ønsker, som forbrugeren har, som nuværende software dækker, og hvilke mangler der eventuelt kunne være på nuværende løsninger.

3.4.1 Softwareløsninger

Vagtplaner er blevet forsøgt løst på mange måder, og der findes meget forskelligt software, som der laver vagtplaner. Hospitalsbranchen har dog rigtigt mange parametre og forhold der skal tages forhold til, derudover er det meget vigtigt at vagtplanen fungerer, som den skal på grund af patientsikkerhed. Derfor er det relativt få former for automatisk vagtplanlægnings-software der er i brug i hospitalsbranchen. Af samme årsag er der mange hospitaler der ikke er gået over til software og i stedet laver vagtplaner via Excel eller papir. (Danske Regioner, 2016) Derudover skriver Region Danmark at selv når et vagtplanlægningssoftware er i brug, virker det kun som understøttelse, så vagtplanen skal stadig udarbejdes i fællesskab mellem softwaren og nogle vagtplans-ansvarlige.

3.4.1.1 MinTid og Tjenestetid

MinTid og Tjenestetid er to stykker software udviklet af Silkeborg Data. Programmerne er lavet til at blive brugt sammen. MinTid er en form for skema hvor medarbejderne f.eks. kan se afdelingens skema, ønske fridage og se ens omsorgsdage.(Silkeborgdata, 2020a) Tjenestetid er software til vagtplanlæggeren hvor de kan se ferie-, afspadsering- og norm timeregnskabet for medarbejderne. Derudover er lovmæssige- og overenskomst-krav inkorporeret så vagtplanen opfylder de forskellige krav.(Silkeborgdata, 2020b). Tjenestetid kan ikke lægge vagtplaner automatisk, det bruges kun til at gøre vagtplanlægning

nemmere. Tjenestetid er også lavet til at gøre lønninger nemmere(Silkeborgdata, 2020b), men det er ikke særligt relevant for vores fokus.

Konflikter ved og evaluering af MinTid Regionmidtjylland har den 15. oktober 2014 udgivet en evaluering fra møder, hvor de diskuterede deres oplevelse med MinTid. Ulemperne som medarbejderne oplevede var således:

- Medarbejderne følte at regler og lovene når det kom til arbejdstid og overenskomsten, var lidt af en barriere, når de havde specifikke ønsker.(Aarhus Universitetshospital, Afdeling M, Risskov, 2014)
- Det var svært at ramme den rigtige vagt, når man ønskede vagter.(Aarhus Universitetshospital, Afdeling M, Risskov, 2014)
- Medarbejderne savnede funktionen til at bytte vagter.(Aarhus Universitetshospital, Afdeling M, Risskov, 2014)
- MinTid mangler en funktion hvor man nemt kan overføre vagtplanen til outlook, så man har den i kalenderen.(Aarhus Universitetshospital, Afdeling M, Risskov, 2014)
- Aftaler med vagtplanlæggeren dukker ikke op når man vil lave særaftaler eller præferencer.(Aarhus Universitetshospital, Afdeling M, Risskov, 2014)

3.4.1.2 Lignende software

Der findes meget forskelligt software med omtrent den samme funktionalitet som MinTid og Tjenestetid. De eksempler vi har fundet som bliver brugt i branchen er: PlanDay som bliver brugt steder som Frederiksberg Hospital, Bispebjerg Hospital og Falck.(PlanDay, 2020) PDC Plan bliver brugt i Region Hovedstaden.(PDC, 2020a) TimePlan bliver brugt af private sundhedspleje virksomheder som Aleris Hamlet.(TimePlan, 2020) Derudover er Silkeborg Data som der har lavet MinTid og TjenesteTid i gang med et nyt produkt; EG Optima som der skal fungere som en bred softwareløsning for det offentlige. (Klar, 2020) Softwarepakken har også funktionaliteter ud over vagtplanlægning og er ment til at fungere som en samlet softwareløsning til hospitaler. Nogle af disse funktionaliteter er: Økonomi, aktivitet og diverse analyseværktøjer. Det er baseret på et allerede eksisterende system, som bliver brugt på 90% af engelske sygehuse og nogle steder i Sverige og Tyskland.

Disse forskellige former for software deler næsten alle deres funktionaliteter på mange områder. En af punkterne hvor der er forskel er lovkravsmæssigt. De har alle sammen implementeret kontraktforhold i en hvis grad. Det er dog ikke alt softwaren der har implementeret de branchemæssige og hospitalsunikke krav. Vi kan se på det udbudte software og at der bliver brugt så meget forskelligt i branchen at valget af software

afhænger af præferencer på det enkelte hospital. Det at der er mange forskellige løsninger på det samme problem er selvfølgelig positivt da hospitaler kan vælge den løsning der passer bedst til dem. Samtidigt vil der dog være en fordel i at de fleste hospitaler bruger det samme stykke software, på den måde kan man sikre sig at det software man bruger har alle de funktioner der skal til for at optimere driften.

En analyse af Rigsrevisionen viser at størstedelen af de afdelinger de har undersøgt ikke udnytter deres medarbejderes arbejdstid maksimalt. Derudover konkluderer de at mange af afdelingerne ikke bruger data fra vagtplanen til at optimere den så meget som de kunne. (Rigsrevisionen, 2015)

Derfor er det vigtigt at den brugte Software giver mulighed for at udnytte de forskellige overenskomster, giver data fra de brugte vagtplaner og lignende. Derudover er der også en fordel for medarbejderne at forskellige hospitaler bruger den samme software. På den måde vil ansatte i sundhedssektoren alle lære at bruge det samme stykke software og hvis man skifter arbejdsplads vil der være en mindre omstillingsperiode. Især for dem der er inddraget i vagtplanlægningen.

Rigsrevisionen foreslår i deres rapport at der skal bruges et it-system til udarbejdelsen af vagtplanerne. Både af de førnævnte årsager og fordi at en transparent vagtplan har stor værdi ifølge deres analyse. (Rigsrevisionen, 2015) En transparent vagtplan betyder her at det er let at se ens egen vagtplan som man f.eks. kan i en app ved brug af disse software løsninger.

3.4.1.3 Matematiske modeller

I Sydvestjysk Sygehus er der blevet lavet en matematisk model til at lave vagtplaner. Der indgår mange forskellige forhold i denne model, som skal justeres, til alle de steder den skal bruges, i rapporten giver de eksemplerne fair vagtfordeling, maksimal opfyldning af friønsker og minimal brug af vikarer.(Overbjerg, 2020) (Danske Regioner, 2016) En af dem der arbejder med at udvikle en matematisk model siger at der for en 4 ugers plan vil være imellem 4-5 tusind variabler. Dog har brug af denne matematiske model stadigvæk gjort at tiden brugt på vagtplanlægning er blevet halveret. Det indebærer både vagtplanlægning, ønsker fra ansatte, aflæsning og opdatering af de matematiske modeller bag. (Danske Regioner, 2016) Denne matematiske model kræver dog allerede løbende opdateringer for at blive brugt på ét hospital, for at udrulle det til samtlige hospitaler ville kræve meget arbejde fordi modellen skal tilpasses til de enkelte steder. Hvilket er en af årsagerne til det ikke er brugt mange steder.

3.4.2 Manuel vagtplanlægning

Der er mange forskellige muligheder for software, matematiske modeller og algoritmer der laver vagtplaner. Men nogle steder har man i stedet beholdt den manuelle vagtplanlægning og optimeret den. I Regionshospitalerne i region NordJylland begyndte de i 2011 at have en vagtplanlægger til at lave vagtplan for alle hospitalets afdelinger. (Danske Regioner et al., 2018) Vagtplanerne bliver stadigvæk indtastet i software der hjælper med at overholde arbejdstidsregler og lignende, men forskellen er, at der ikke er automatisk vagtplansgenerering, og at der er mere fokus på, hvem der laver vagtplanen. I stedet for at en læge, laver en vagtplan i hver afdeling, er der én læge der står for at lave den overordnede vagtplan på tværs af afdelingerne i samarbejde med de enkelte afdelinger.

Nogle af de fordele som der er, ved at have én læge til at lave vagtplanen er, at lægen kan planlægge så sygeplejeskernes tid i ambulatoriet bliver optimeret, give yngre læger god mulighed for, at lære deres speciale og sikre at specialiserede lægers vagter er godt planlagt for at de dækker de tidspunkter, hvor der er brug for dem.

Det at én læge laver vagtplanen for alle afdelinger, gør at i stedet for at hver afdeling har deres egen optimale vagtplan, er der en samlet vagtplan, som der overordnet er mere effektivt end de enkelte vagtplaner uden det samarbejde. Ifølge Rigsrevisionens rapport er en af de store grunde til ineffektivitet på danske hospitaler manglende samarbejde imellem de forskellige afdelinger og faggrupper.(Rigsrevisionen, 2015) Mange hospitaler har også forsøgt at optimere hvordan man arbejder med vagtplaner i samarbejde med deres respektive HR-afdelinger for at prøve at opnå det samme mål. (Nicolajsen, 2017)

Med manuel vagtplanlægning har man erfaret at man kan planlægge længere ud i fremtiden hvilket er praktisk af flere årsager. Det gør det nemmere at planlægge hvornår man kan tage patienter og ansatte kan få deres afspadsering, fridage og ferie at vide længere tid på forhånd.(Danske Regioner et al., 2018)

Automatisk vagtplanlægning har sværere ved langsigtede vagtplaner og skal stadigvæk udarbejdes i samarbejde med et menneske for at sikre, at det kan fungere og for at bytte rundt på nogle vagter. (Danske Regioner et al., 2018)

Det negative ved manuel vagtplanlægning er, at det for større hospitaler kræver mindst en fuldtidsplanlægger og mange flere vil blive inddraget. Det kræver mange menneskelige ressourcer, hvoraf mange kunne blive sparet ved en automatisk vagtplan.

På Sygehus Sønderjylland indgår der 45 planlæggere i skabelsen af vagtplanen. Selvom ikke alle 45 udelukkende arbejder med vagtplanen er det stadigvæk mange, i forhold til hvor mange der er involveret i verifikationen af en automeret vagtplan. (Danske Regioner

et al., 2018)

Derudover er det nødvendigt, at den hovedansvarlige har et godt overblik, over hvad der sker på hospitalet, og hvordan de forskellige afdelinger fungerer. En vagtplanlægger der ikke har indblik i de forskellige kulture, i forskellige afdelinger, vil højst sandsynligt lave en suboptimal vagtplan. På grund af det vil det give store problemer for hospitalet hvis den hovedansvarlige for vagtplanlægning stopper, har sygdom eller lignende.

3.5 Opsummering af krav til vagtplan

Nu har vi undersøgt emnet og fundet frem til vigtige krav for at en tidsplan er god for medarbejderne.

I afsnit *3.2.1 Stress* og *3.2.2 Nattearbejdes effekt på sygeplejersker* fandt vi frem til 5 overordnede krav for at minimere sygeplejerskerne stress

- Man skal føle at man har indflydelse over, hvornår man skal arbejde.
- Find en måde at balancere familie- og arbejds-livsbalance.
- Giv medarbejdere tid til at komme sig efter en hård vagt såsom nattearbejde.
- Sørge for at vagtplanen er forudsigelig og ikke ændrer sig for meget.
- En vagtplan skal tage hensyn til sygeplejersker med nattearbejde, ved at indsætte pauser.
- En vagtplan skal tage højde for individuelle præferencer.

I afsnit *3.2.3 Lange arbejdsdage* fandt vi frem til 1 krav for at minimere fejlbehandling på grund af træthed:

- Arbejdsdage skal helst ikke være længere end 8 timer.

I afsnit *3.2.4 Døgnrytme* fandt vi frem til 1 krav for at minimere døgnrytme problemer.

- Sørg for at roterende vagtskift kører fremad, så enkelte ansatte tager vagter i rækkefølgen (nat, morgen, dag, aften).

I afsnit *3.2.5 Arbejds- og privat-livsbalancen* fandt vi frem til 2 krav for at forbedre arbejde- og privatlivs balancen.

- Vagtplanen skal opbygges sådan at den skaber mere balance mellem arbejdet og privatlivet.
- Vagtplanen skal opbygges sådan at arbejde ikke går unødvendigt ud over sygeplejerskernes privatliv.

I afsnit *3.3 Regler og love bag vagtplaner* fandt vi 5 relevante krav til vagtplanen for at overholde lovene og regler:

- 48 timers reglen - Overarbejde og merarbejde.
- 11 timers reglen.
- Ugentlig fridøgn.
- Arbejdstider, herunder natarbejde, frokost og pauser.
- Helligdage, FO-dage og søgnehelligdage.

I afsnit *3.4 State of the art* fandt vi frem til 3 krav for at forbedre den overordnede effektivitet og mindske stress for sygeplejerskerne.

- Tidsplanen skal være transparent gennem et godt brugerinterface, sådan at medarbejderne nemt kan tjekke deres arbejdstimer.
- Tidsplanen skal forsøges at blive lavet i samarbejde imellem de forskellige afdelinger.
- Vagtplanen skal laves for så lang tid som muligt, så den er forudsigelig.

Her har vi opsat kravene igen i tabel 3.2 for overskuelighed:

Formål	Krav	
Mindske stress	Selvinflydelse	Langsigtet- og forudsigelig vagtplan
	Hviletid efter natarbejde	Pauser
	Vagter roterer fremad	Transparent vagtplan
Optimere udbytte af vagter	Undgå arbejdsdage over 8 timer hvis muligt	Samarbejde imellem afdelinger
Lovkrav	48 timers reglen	11 timers reglen
	Fridøgn	Helligdage
	Arbejdstider, natarbejde og pauser	

Tabel 3.2. Tabel over vores krav

4 | Afgrænsning og problemformulering

Til vores problemformulering har vi afgrænset mange af de ting vi har fundet ud af i analysen. Grunden til at vi afgrænser os så meget, er at det er meget svært at lave et program, der opfylder alle kravene og udover det er vi også tidsbegrænset, derfor medtager vi kun de vigtigste og mest opnålige mål.

Først vil vi gennemgå de ting vi ikke kommer til at medtage og hvorfor. Derefter kommer vores problemformulering.

4.1 Opsummering af problemanalysen

I problemanalysen har vi fundet frem til, at sygeplejersker ofte er udsat for psykiske og fysiske helbredsproblemer grundet en suboptimal metode til vagtplanlægning. Vagtplanlægningen er ofte suboptimal, da den er irregulær, og især nattevagter kan have en stor betydning for sygeplejerskernes sundhed. Ud over dette er der også en række lovmæssige krav som skal overholdes, for at tillade at produktet vil kunne fungerer på arbejdsmarkedet, og en række krav, som vil tilfredstille de mangler, som sygeplejerskerne selv mener, at de nuværende løsninger, og andre vagtplanlægningsmetoder, ikke dækker godt nok.

4.2 Afgrænsning

Vi har valgt at gøre så arbejdsdagende er maksimalt 8 timer lange, det betyder at en medarbejder ikke får tildelt mere end 48 timer om ugen. Det gør vi for at mindske kompleksiteten af opgaven så vi kan nå at blive færdige.

Antal af medarbejdere

Vi har valgt at brugeren skal bestemme hvor mange medarbejdere der skal være på

arbejde hver vagt. På den måde skal vores program ikke udregne, hvorfor der skal være et antal medarbejdere. Det undgår unødvendigt kompleksitet og gør det også nemmere for brugeren. Det er også nemmere for en erfaren læge, at udregne hvor mange der skal på arbejde end vores program.

Regler og love

For at produktet skal have være brugbart, skal det følge lovgivningen. Dog ser vi bort fra f.eks. regler om hellig- og feriedage.

Ferie og helligdage

Vi har valgt ikke at medtage ferie, helligdage og lignende. Selvom evnen til at tage ferie og få helligdage er vigtigt i planlægningen af en vagtplan, så mener vi ikke at det er værd at medtage på grund af den kompleksitet det inddrager. Derudover uden kontakt til et hospital er vores viden omkring hvordan de vil håndtere det meget lav.

Pauser

Vi kan ikke styre hvornår sygeplejerskerne har tid til at holde pause, og det ville være besværligt, hvis brugeren skulle ændre vagtplanen, hver gang der sker noget som flytter pausen. Derfor vurderer vi, at det er bedst, at medarbejderne eller vagtplanlæggeren styrer, hvornår der skal holdes pauser, og programmet styrer, hvornår de skal være på arbejdet.

Samlede krav til en software løsning

- Give medarbejderne tid til at komme sig over en nattevagt.
- Vagtplanen skal være forudsigelig, det betyder at vagtplanen skal være givet i god tid og ikke ændre sig for meget.
- Arbejdsdagene må ikke være mere end 8 timer lange.
- Vagtplanen skal overholde relevante lovkrav.
- Roterende vagtplan skal gå i den rette rækkefølge, altså skal vagterne cyklisk gå fremad.
- Vagtplanen skal være nem at tjekke, og modtagelig for ønsker fra sygeplejerskerne.

4.3 Problemformuleringen

Skæve arbejdstider og dårlige vagtplaner skaber mange problemer for medarbejderne på hospitaler. Hvordan kan vi reducere stress hos sygeplejersker og skabe en bedre arbejds- og privatlivsbalance, igennem et automatisk vagt-

planlægningsprogram der kan lægge en vagtplan til hospitaler, der overholder de førnævnte krav?

5 | Løsning

5.1 Tekniske krav til programmet

Vi har dannet nogle krav ud fra hvad vi har lært igennem problemanalysen. Vi har valgt ikke at kontakte et rigtigt sygehus fordi det her kun er et 1. semester projekt, derfor vil det ikke være inden for scopet af dette projekt at kontakte et sygehus for at få input på hvilke krav de ville have til et vagtplanlægnignsprogram.

Derfor så har vi fremsat krav, som vi mener er realistiske og relevante. Vores krav til vagtplanen kommer fra problemformuleringen og vores andre krav er hvad vi mener giver mening i forhold til selve programmet.

5.1.1 Funktionelle krav

For at et krav er funktionelt, kræver det at kravet omhandler en bestemt funktionalitet af programmet. Det betyder at kravene er relativt nemme at vurdere, da programmet enten lever op til kravet eller ej. (Indexed.dk, 2020)

Bløde krav

Bløde krav i vores kontekst betyder at vagtplanen ikke behøver at overholde dem, men det er en bedre vagtplan hvis den gør. Kravene kommer fra problemanalysen hvor vi har fundet ud af hvad der er bedst for medarbejderne.

- Programmet skal tage højde for præference af hvilken vagt, sygeplejerskerne helst vil have, morgen, aften eller nat
- Programmet skal prøve at indfri medarbejderes ønsker om en fridag

Hårde krav

Hårde krav er de krav som skal overholdes for at en vagtplanen kan klassificeres som valid.

- Vagterne skal gå cyklisk fremad. Det betyder at hvis man har en natte-vagt, så skal man have enten en natte-vagt eller en dag-vagt. Hvis man har haft en dag-vagt, så skal man have en aften-vagt eller en dag-vagt. Hvis man har haft en aften vagt så skal man have en aften vagt, da man ikke kan få en natte-vagt på grund af 11-timers reglen. Hvis man gerne vil bryde den cyklus, så skal man have en dag hvor man ikke arbejder.
- En person må maksimalt have to nattevagter i træk
- Programmet skal overholde disse lovkrav:
  - Ugentligt fridøgn
  - 11 timers reglen

Input / output krav

Vores program skal kunne gemme og indlæse filer. Det skal gøres i et bestemt fil format der er passende. Vi har valgt at bruge CSV filer da de er nemme at redigere og læse. Det er en simpel måde at repræsentere data på, som let kan blive lavet via regnearks-programmer eller simple tekst redigerings programmer. En nærmere forklaring af CSV filer kommer i sektion 5.4.1. Derudover laver vi en pæn version af vagtplanen til hver medarbejder, den fil bliver gemt som en '.txt' fil.

- Programmet skal kunne eksportere en vagtplan til en CSV fil
- Man skal kunne læse medarbejdere ind gennem en CSV fil
- Programmet skal kunne indlæse en vagtplan i CSV format og lave pæne tekst vagtplan filer til de enkelte medarbejdere
- Programmet skal outputte en uge af gangen
- Programmet skal acceptere CSV-filer som bruger semikolon eller komma som separator

Programspecifikationer

Programspecifikationer er krav, som strukturering og implementeringeng af programmet følger.

- Programmet skal lave vagtplanen efter de tidsperioder som kan ses i tabel 5.1
- Navne har en maksimum længde i dette program, som ikke kan overskrides
- Programmet skal kunne håndtere vilkårligt mange medarbejdere

5.1.2 Non-funktionelle krav

Vores non-funkionelle krav er dem som ikke kan vurderes objektivt. Derfor så kan vi kun teste disse ud fra flere folks meninger og teste dem kvalitativt. (Indexed.dk, 2020)

Navn	Start	Slut
Nat	00:00	8:00
Dag	8:00	16:00
Aften	16:00	24:00

Tabel 5.1. En tabel over de 3 forskellige arbejdstider som programmet opdeler dage i

- Programmet skal være brugervenligt.
- Programmet skal kunne lave en vagtplan indenfor en acceptabel tidsperiode

5.2 Teori om løsning af vagtplaner

Problemet med at lægge vagtplaner er som beskrevet tidligere et np-fuldstændigt problem. Det betyder at det tager meget computerkraft at finde en løsning til. Derfor har vi kigget på hvad andre har gjort for at gøre denne proces effektiv. Vi er kommet frem til at en genetisk algoritme er den bedste løsning til vores optimerings problem. (Burke et al., 2004)

5.2.1 Genetiske algoritmer

Vi vil nu gennemgå et eksempel hvor andre har prøvet at løse problemet med en genetisk algoritme og hvad de har gjort for at få algoritmen til at løse problemet. I et koreansk studie fra 2001, undersøgte forskere hvordan at man kunne planlægge et vagtskema til sygeplejersker ved hjælp af en genetisk algoritme(Tsuruoka, 2001).

I første trin af algoritmen allokeres der plads til det minimum antal arbejdere, som skal være til stede ved en given vagt. Herefter er der en række kvoter, som skal opfyldes. Dette kunne f.eks. være at en medarbejder ikke må have mere end 8 nattevagter på en måned. Hvis det skulle ske at en medarbejder f.eks. fik 9 nattevagter på 1 måned, ville 1 af disse vagter blive flyttet, til en ny medarbejder i systemet, som har ubesluttede vagter. Herefter allokeres der også plads til fridage i vagtplanerne, ud fra de ubesluttede vagter i skemaet.

Herefter gennemgår individerne en evalueringsfunktion, hvor at individernes fitnessniveau bliver evalueret. I dette studie valgte man at bruge metoden, hvor at man brugte bestemte individer til mutation, i modsætning til at "klippe"i vagtplanerne. Studiet viste at genetiske algoritmer kan være en effektiv og tilfredstillende måde, at løse vagtplanlægningsproblemer, med de metoder som studiet tog i brug.

En genetisk algoritme er inspireret af evolutionsteorien (Mitchell Melanie, 2020). En genetisk algoritme fungerer ved, at man har en population af forskellige individer. De

elementer som et individ består af i en genetisk algoritme er ofte omtalt som dna. Man sætter individerne op imod hinanden i en funktion, og finder de bedste, bedømt ud fra en fitness funktion. Herefter vil forskellige individer bytte elementer med hinanden. Det er det man kalder for crossover. Det der afgør hvilke elementer der skal blandes kalder man for crossover-punktet. Hvilke individer der blander deres dna er forskelligt fra algoritme til algoritme, nogle vælger at slå den bedste og værste sammen, andre sammensætter de to bedste og så videre. Grunden til at man gør det på så mange forskellige måder er fordi at der ikke er en metode der altid er bedst. Som regel vil man bruge de bedste skemaer til det da ideen er at de har godt dna siden de har en høj fitness.

Nogle af de ting man vil prøve at undgå når man laver crossover er at individerne bliver for homogene og at populationen for hurtigt bliver færdigudviklet før at individerne er blevet så gode som muligt. Når populationen bliver for homogen til at der sker flere ændringer, så kalder man det "tidlig konvergens", det skriver vi om i 5.2.2.

Der findes mange metoder til at undgå disse ting. Vi vil ikke komme nærmere ind på metoder vi ikke benytter os af. Men en ting alle algoritmer justerer er crossover og crossover-point. Måden individer skaber et nyt individ fra deres dna er som beskrevet også afhængigt af algoritmen. Hvis man ser et individ som en bitstreng kan man vælge enten at have et cut-off punkt hvor f.eks. halvdelen af dna'et, altså de enkelte bits i strengen i dette eksempel, bliver byttet imellem de to individer. Man kan også vælge at hver anden bit bliver byttet og så videre, og med mere komplicerede individer end en bit streng kan man lave endnu flere forskellige metoder til crossover.

Den næste teknik der bliver brugt til at ændre DNA i genetiske algoritmer er mutation. Det vil sige at noget dna i de nye individer tilfældigt bliver ændret. Dette gør man for at undgå at undgå tidlig konvergens imod noget bestemt og for at beholde diversitet blandt de forskellige individer. Det vi forstår ved, at en genetisk algoritme konvergerer, er at løsningerne går hen imod en specifik løsning eller en løsningsmetode. Det forklarer vi nærmere i afsnit 5.2.2.

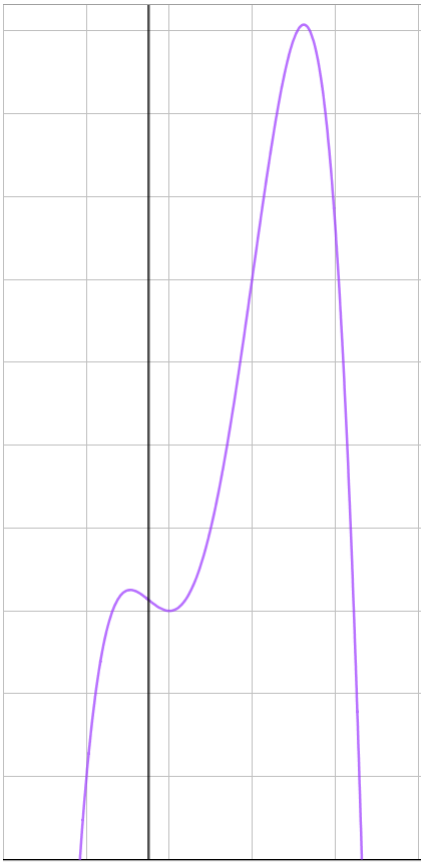
5.2.2 Tidlig konvergens

Genetiske algoritmer gør at individerne i populationen udvikler sig generation efter generation. Ideen bag dette er at algoritmen udforsker forskellige måder at løse problemet på. Nogle problemer kan have mange forskellige løsninger som er af forskellig kvalitet.

Derfor er det farligt hvis der for eksempel i en genetisk algoritme er to forskellige løsninger hvor den ene er god som udgangspunkt men ikke kan forbedres markant og den anden er

dårlig som udgangspunkt men har meget potentiale hvis der bliver lavet nogle ændringer i senere generationer. I den algoritme er der en fare for at fitnessfunktionen vurderer at den første metode er den bedste og den derfor bliver spredt til alle individerne. På den måde kan den dårlige med meget potentiale blive kasseret. Hvis det sker vil man i resultatet ikke kunne se at den endelige løsning ikke er den egentlige bedste løsning. (Simona Nicoară, 2009) (Yee Leung et al., 1997)

Det er det man inden for genetiske algoritmer kalder for tidlig konvergens. Det vil man gerne undgå, for at øge chancen for at man finder den bedst mulige løsning. I forbindelse med det bruger man også begreberne lokalt- og globalt maksimum. Det deler stort set betydning med udtrykkene fra matematik. Lokalt maksimum vil sige at det er den højeste fitness et individ har af de genererede individer i en kørsel af algoritmen. Globalt maksimum er den højeste fitness et individ kan opnå i algoritmen. (Womersly, 2008) Det kan bedst visualiseres ved hjælp af en graf, hvor der er forskellige maksima fordelt rundt på x- og y-planet, vi har en graf i figur 5.1.



Figur 5.1. Graf til illustration af lokal og global maksima

I dette eksempel er den lilla graf en fitnessfunktion. Den vertikale sorte linje er en af funktionernes vendepunkter. Det vil sige at hvis en stor del af en generations individers fitness ligger til venstre for den sorte linje vil den næste generation konvergere imod det

lokale maksimum til venstre for vendepunktet. Hvis algoritmen er begyndt at konvergere imod det maksimum skal der blive dannet individer der ligger til højre for vendepunktet for at begynde at konvergere imod det globale maksimum.

Den ændring der skal være fra den ene generation til den anden, skal altså være ret stor. Dette er blot et eksempel. En fitness funktion kan se ud på mange måder og operere i mange dimensioner. I dette eksempel skal der enten være en heldig mutation eller lignende eller også vil slutresultatet være det lokale maksimum som der ikke er den bedste løsning.

På en anden måde kan tidlig konvergens forklares, ved at hvis algoritmen befinder sig på et lokalt maksimum, og der kommer crossover, men det ikke ændrer den næste generation markant. Så kan man risikere at generationerne, bevæger sig omkring det samme lokale maksimum, men aldrig flytter sig nok til at nå det globale maksimum.

Efter crossover er de nye individer den næste generation. Når man laver en genetisk algoritme, skal man altså beslutte hvor mange individer, man starter ud med og så kan algoritmen blive afsluttet af flere forskellige metoder. Man kan stoppe når et individ opnår en hvis fitnessværdi, et bestemt antal generationer, tjekke om det bedste individ ændrer sig meget lidt i et hvis antal generationer eller man kan benytte en blanding. (Safe et al., 2004) (Jan et al., 2000) (Mallawaarachchi, 2017)

5.2.2.1 Population

Populationens størrelse har en effekt på hvor succesfuld en genetisk algoritme er, hvis populationen er for lille, så er der en risiko for at algoritmen ikke vil kunne finde en god løsning, fordi der ikke er nok individer til at udforske de mulige løsninger. For at få de bedste løsninger vil fremgangsmåden derfor være at have en relativt stor population. At have en større population er dog ikke udelukkende positivt da det gør at algoritmen bliver langsommere fordi at der skal laves flere trin per generation. Hvis det er muligt så kan algoritmen blive bedre hvis den første generation allerede har nogle individer med god fitness. Derfor så skal man så vidt muligt prøve på at give nogle gode individer i starten. (Diaz-gomez og Hougen, 2007)

5.2.2.2 Selektion

Selektion er den del af den genetiske algoritme der udvælger de individer som skal blive parret og hvilke der skal fjernes fra populationen. Det er vigtigt i forhold til diversiteten af populationen at vælge nogle dårlige individer nogle gange, fordi ellers ender man med tidlig konvergens. Selektionen sker ved at vi vurderer hvert individ med en fitness funktion.

Denne funktion giver en talværdi der beskriver hvor godt individet er. Det kan man så bruge til at vælge de bedste individer til parring. (Shyalika, 2019)

5.3 Design af genetisk algoritme

Nu har vi kigget på den generelle teori bag en genetisk algoritme. Nu vil vi bruge det til at lave en genetisk algoritme som kan løse vores vagtplanlægnings-problem. Vi definerer *Vagt*, som det tidspunkt på døgnet hvor at vagten forekommer, en *Dag*, som i hvilken dag på ugen vagten eksisterer, og en *Blok*, som som er en kombination af *Vagt* og *Dag*. Disse begreber er visualiseret i figur 5.2.

				Dag						
	Tidspunkt			Mandag	Tirsdag	Onsdag	Torsdag	Fredag	Lørdag	Søndag
Vagt	Nat	00:00	08:00							
	Dag	08:00	16:00							
	Aften	16:00	00:00			Blok				

Figur 5.2. Her kan man se hvad vi brugere de forskellige ord til at repræsentere. Vi bruger dag til at beskrive en dag, vi bruger vagt til at beskrive hvad tidspunkt på dagen det er og blok til at beskrive en bestemt vagt på en bestemt dag.

5.3.1 Valg af DNA

Vi har valgt at et individ i vores algoritme svarer til en vagtplan for en uge. Derudover har vi valgt at en blok svarer til et stykke DNA. Grunden til dette er fordi vi har vurderet at den bedste måde at lave crossover er ved at bruge blokke som DNA. Det er blandt andet fordi en blok for sig selv ikke kan vurderes som invalid, så længe den samme medarbejder ikke er der to gange. Vi har overvejet at bruge en dag eller en vagt til vores DNA, disse vil dog danne komplikationer, som vi nu gennemgår.

Hvis vi havde valgt *vagter* som DNA, så vil der kun være tre stykker DNA, og vi vil derfor ikke drage særligt stor nytte af en genetisk algoritme da det ville være svært at blande DNA'et og stadigvæk have en valid vagtplan. Det giver heller ikke mening for mutation at skulle ændre så stor en del af vagtplanen. Siden at vagtplaner også hænger meget sammen med de tidligere vagter, vil det ikke give mening at tage en hel *vagt* uden de tidligere dage. Det er både fordi at det går imod det bløde krav og fordi vi har fokus natarbejde og roterende døgnrytme og fordi vagtplanen på den måde også vil have en stor chance for at det samme menneske får to vagter på den samme dag.

Hvis vi havde brugt *dag* som DNA, så ville det også bringe komplikationer. Der ville være 7 DNA stykker, hvilket er bedre end ved valget af *vagter*. Hvis det er *dage* som bliver brugt som DNA, så er der en større risiko for at kravet for at en person ikke må have flere natte vagter end to i streg bliver brudt. Derudover gør det også at hvis en dag er dårlig eller invalid vil den eneste måde at gøre den bedre på være en heldig mutation. På den måde kunne man opleve at have nogle gode vagtplaner hvis fitnessvurdering falder fordi at en invalid dag skifter rundt imellem vagtplanerne.

Vores valg af at bruge blokke som DNA har også nogle problemer. Det er f.eks. at det kan være meget let at komme til at lave en invalid vagtplan f.eks. fordi at den samme medarbejder kan ende med at arbejde 3 gange på en dag. Vi har dog vurderet at det giver bedst mening fordi at man ved ændring af blokke giver algoritmen mulighed for at opbygge den bedst mulige vagtplan, som giver mange flere muligheder end ved valg af vagter/dage. Derudover er det også den DNA som giver mest mening at mutere da det i vores situation er det mindste DNA af de beskrevne muligheder, som vi ser som de eneste fornuftige muligheder der er.

5.3.2 Start population

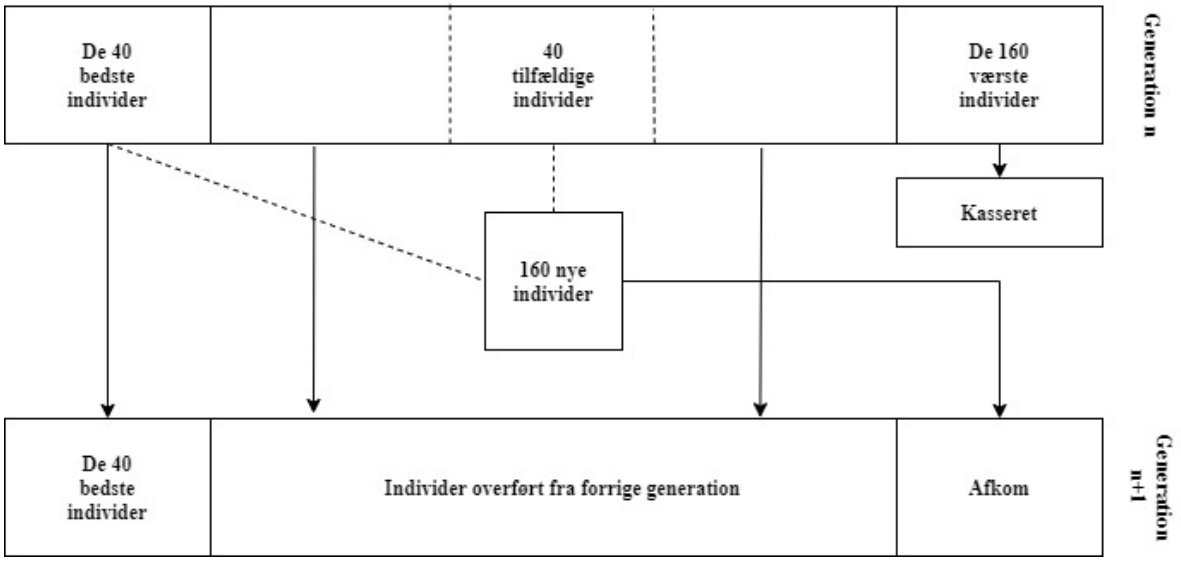
I vores algoritme bliver start population genereret tilfældigt. Som vi har beskrevet tidligere i sektion 5.2.2.1, så er det bedst for de endelige resultater hvis start populationen har en god fitness score. Derfor giver det mening at bruge en heuristisk algoritme der kan generere nogle individer som der lever op til nogle simple kvalitetskrav. Det vi har valgt at gøre er at sørge for at der ikke er en medarbejder der skal arbejde to gange den samme dag, samt at kravet til antal af medarbejderne i blokkene er opfyldt. Vores analyse lægger op til at algoritmen ville fungere bedst hvis start populationen bestod af udelukkende valide vagtplaner, men det at generere valide vagtplaner ville i sig selv være et større program. Vi mener at den heuristiske algoritme vi benytter er tilstrækkelig for vores mål og algoritme.

5.3.3 Mutation

Når en ny generation bliver genereret er der 1/3 chance for at en af de 21 blokke (1 for hver vagt på 7 dage), bliver muteret. Den muterede blok er altid valid i sig selv da vi sørger for at den samme medarbejder ikke kan være der flere gange. Vi benytter os af mutation for at bekæmpe tidlig konvergens og for at give algoritmen en mulighed for at rette op på en uheldigt genereret blok der holder individernes fitness tilbage.

5.3.4 Generering af ny generation

I vores algoritme bliver vores population på 1.000 individer genereret ved at nogle udvalgte individer agerer som "forældre". Ved crossover bliver de to forældres dna blandet sammen. Når den første generations individer er blevet genereret, så parrer vi de 40 bedste individer med hver deres tilfældige individ. Hvert par får fire børn som der alle, sandsynligvis, er forskellige på grund af måden vi laver crossover og mutation på (se 5.3.3 og 5.3.5. Derefter kasserer vi de 160 værste individer da hver generation genererer 160 nye individer i alt. Populationen vil altid have den samme mængde individer, da den genetiske algoritme på den måde kan køre et vilkårligt antal generationer.



Figur 5.3. Visualisering af selektion af dna

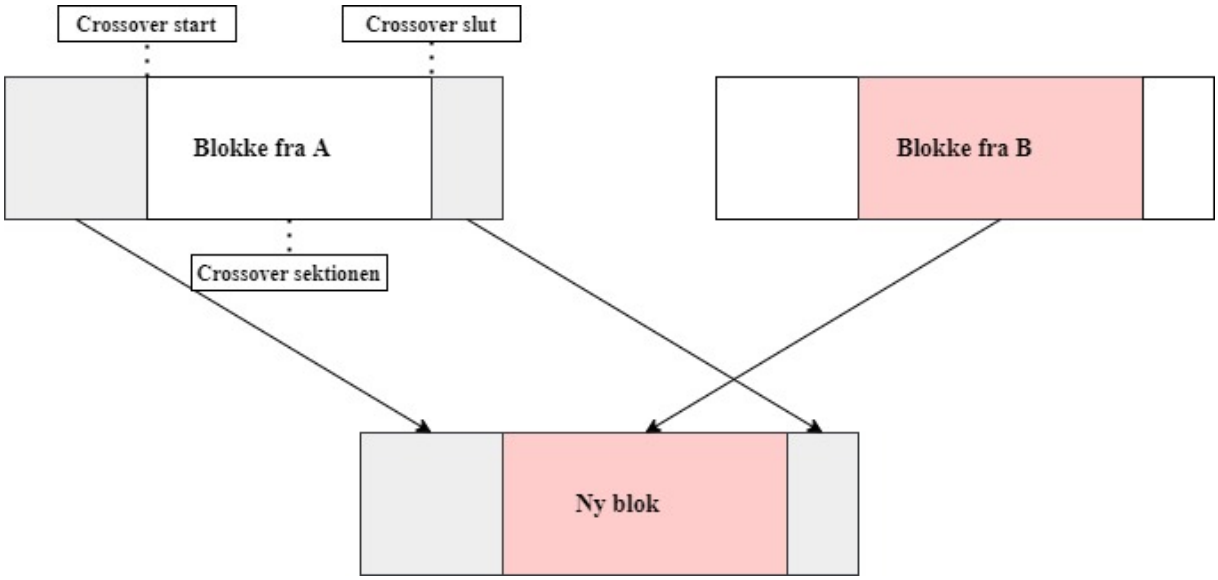
5.3.5 Crossover

Når vi har to individer til crossover, så skal vi blande deres DNA. Som beskrevet i 5.3.1 består et individ af 21 stykker dna som er de 21 blokke. Derfor er det altså de blokke der skal byttes for at lave crossover. Den måde vi har valgt at blande DNA imellem generationer, er ved at bruge et tilfældigt crossover punkt på DNA'et.

Vi vælger vores crossover punkt ved først at generere et tilfældigt tal imellem 0 og 20 derefter genererer vi endnu et tal som er større end det første tal og højst 21. Den DNA indekseret fra det det første tal til og med det sidste tal er den DNA der skal laves crossover på.

Barnet bliver lavet som en kopi af det første individ og derefter bliver det valgte dna fra det andet individ kopieret over i barnet. I vores program vil det sige at det første skema,

med en af de 40 højeste fitness værdier, er det individ som barnets DNA bliver inden crossover. Vi har visualiseret det i figur 5.4, hvor vi har valgt vilkårlige crossover start- og slutpunkter.



Figur 5.4. Crossover på Blok A og Blok B

5.3.6 Evalueringen af vagtplan.

Vi har valgt at tillægge de forskellige individer point ud fra en fitness-funktion, som er en naturlig del af en genetisk algoritme.

De point som funktionen giver et individ, er baseret på vores hårde og bløde krav. En vagtplan bliver tildelt bonuspoint, hvis at medarbejderne har deres foretrukne vagt. Derudover giver den en lille straf hvis en medarbejder ikke har fået opfyldt deres fridagsønske.

Hvis et individ ikke opfylder et hårdt krav bliver der tildelt en stor mængde minuspoint. Antallet af minuspoint for ikke at opfylde et hårdt krav er markant højere end antallet af pluspoint for at opfylde et blødt krav. Det er baseret på at før en vagtplan kan anses som valid, kræver det at alle hårde krav er opfyldt, hvilket vil sige at en vagtplan der ikke bryder nogle hårde krav, ikke kan ses som i samme kategori som en vagtplan der bryder hårde krav ligegyldigt hvor mange bløde krav den opfylder.

Ifølge vores program er det bedste individ det individ med det højeste antal point. I praksis vil det altså sige det individ der bryder færrest hårde krav og derefter bedst opfylder de bløde krav.

Vores fitnessfunktion giver specifikt -1.000 point for hvert hårdt krav der bliver brudt. Hvis det giver en medarbejder deres foretrukne vagt giver det +1 point og hvis en medarbejder ikke har deres ønskede fridag giver det -2 point.

Med det antal medarbejdere vores program er ment til at håndtere, vil en vagtplan ikke kunne få nok pluspoint til at nærme sig 1.000. Det ville dog være simpelt at gøre minuspointene en størrelsesorden større og være f.eks. 10.000 minuspoint.

Ud fra dette system kan de forskellige individer sammenlignes, ud fra den mængde af points de har, derved finder vi de bedste vagtplaner fra hver generation.

For at komme med et eksempel med en simpel generation:

	Individ 1	Individ 2	Individ 3
Points	-910	32	60

Tabel 5.2. Simpel vurdering af en generation af 3 individer

Hvis vi vurderer generationen i 5.2, så observeres der at individ 1 har et negativt antal points, dette er fordi individet ikke opfylder alle hårde krav, læg mærke til at der stadigvæk er blevet tildelt points for de bløde krav der er blevet opfyldt. Antallet af points som vores fitnessfunktion giver er vores vurdering af et skemas kvalitet, det vil sige jo flere point et individ har, jo bedre anses det til at være. Derfor ville individ 3 anses som bedst i dette eksempel.

5.4 Strukturering af programmet

Vi har nu defineret hvad vi vil have vores program skal og defineret nogle krav som vores program skal opfylde. Nu vil vi beskrive nærmere hvordan vi vil designe programmet ved hjælp af flowcharts og beskrivelser.

Flowcharts er en teknik, som bruges til f.eks. algoritmer og funktioner så man kan tydeliggøre hvilken procedure der skal følges. Det gør det også mere klart at se, hvilke beslutninger der skal tages uden forbehold for det endelig resultat.

Et flowchart er en metode til at visualisere struktureringen af et program, eller en del af et program. I et flowchart inddeler man figurer til forskellige egenskaber, betydningen af indholdet i disse figurer varierer. Derfor definerer vi her hvordan vi bruger dem i denne rapport, baseret på en artikel(programiz, 2020):

- Firkant = En procces

- Rombe = Input/output
- Diamant = To forskellige handlinger valgt efter et udtryk
- Firkant med runde hjørner = Start eller slut på funktion

Vi bruger "divide and conquer"metoden til at programmere programmet. Her deler man større opgaver op i små dele indtil delene bliver små nok til at kunne blive implementeret.(Smith, 1983) Det er via den metode at vi har delt vores program op og lavet forskellige hjælpefunktioner i vores program.

5.4.1 Data input/output

Som vi har beskrevet i vores krav, så skal vores program benytte CSV (Comma seperated values) filer til input af medarbejdere og til output af vagtplan. En CSV fil er en simpel form for fil format, som der kan redigeres med regneark- og tekstredigerings programmer, hvilket er godt for brugeren, derudover er det også simpelt at indlæse i c. CSV filer er som navnet siger en fil med komma sepererede værdier. I filen svarer et komma til kolonneskift i et regneark og linjeskift svarer til rækkeskift. (Computer Hope, 2020)

Det betyder også at hver celle ikke kan indholde linjeskift eller kommaer. Her kunne man f.eks. implementere sådan at hvis en værdi er inkapsuleret med citationstegn (") så kan man godt bruge komma og linje skift. Det har vi i vores program valgt ikke at implementere fordi det er en unødvendig komplikation siden at navne ikke indeholder kommaer eller linjeskift og det ikke er relevant i de andre elementer der skal indtastes.

5.4.1.1 Medarbejder liste format

For at vi kan tage en medarbejder liste som CSV, så skal vi bestemme hvilken kolonne; navn, ønsket fridag, ønsket vagt og UUID skal stå i. Her har vi besluttet at det kommer i denne rækkefølge:

navn, ønsket fridag, ønsket vagt, UUID

Vi har givet et eksempel på en medarbejderliste i tabel 5.3.

Shannon Casady	tirsdag	nat	0
Tanika Burstein	lørdag	aften	1

Tabel 5.3. Eksempel på en medarbejder liste tabel

I Snippet 5.1 vises der hvordan CSV filen vil se ud.

```
1 Shannon Casady,tirsdag,nat,0
2 Tanika Burstein,lørdag,aften,1
```

Snippet 5.1. CSV formattering af tabel 5.3

5.4.1.2 Vagtplan format

Når vi har lavet vores vagtplan med vores program, så skal vi gemme skemaet i en CSV fil. Her skal vi også beslutte os om en måde at få det ind i en tabel på. Her har vi besluttet os for at bruge følgende format:

Dag, Vagt, navn1.UUID1, navn2.UUID2, navn3.UUID3, ..., $

Her har vi først dag og vagt som beskriver hvilken blok denne linje beskriver. Det skal ikke blive brugt til noget af programmet fordi den forventer at alle 21 blokke kommer i rækkefølge. Dette bliver mest brugt sådan at vagtplanlæggeren nemmere kan lave manuelle ændringer i vagtplanen. Efter det så har vi navn og UUID på de personer som skal arbejde i den blok og til sidst et $ som markerer slutningen på den liste. Navn og UUID er enkodet med et navn først så et punktum og til sidst deres UUID. I virkelighed skal programmet kun bruge UUID til at referere til den rigtige person. Navnet er der kun sådan at vagtplanlæggeren selv nemt kan se hvem der er sat på hver blok. Et eksempel på en vagtplan tabel kan ses i bilag 3. Det her er dog ikke den færdige version som skal gives til medarbejderne. Det er en fil som bruges af vagtplanlæggeren og programmet. Man kan bruge programmet til at lave en individuel vagtplan til hver medarbejder som kun viser deres vagter. I stedet for at være formateret som CSV, er det en brugervenlig fil som medarbejderne intuitivt kan læse. Et eksempel på en individuel intuitiv vagtplan er i bilag 4.

5.4.2 Program funktionaliteter

Programmet skal have tre forskellige funktioner.

- Skab en vagtplan
- Test en vagtplan
- Print brugervenlig version af vagtplaner

Den første del skaber en vagtplan ud fra den medarbejder liste som blev beskrevet tidligere og ud fra hvor mange der skal arbejde ved hver vagt. Programmet læser `medarbejder.csv`

som medarbejder listen og beder vagtplanlæggeren om at indtaste hvor mange der skal arbejde på hver vagt.

Den anden funktion af programmet vurderer en vagtplan. Her bliver alt input taget igennem program parametre hvor brugeren skriver i kommandopromten `vagtplanlægger.exe test <navn på vagtplan>` for at teste en bestemt vagtplan. Derefter giver programmet en fitness score til den givne vagtplan.

Den sidste funktionalitet af programmet er at printe en brugervenlig version af vagtplanen ud. Den fungerer også via program parametre. I dette tilfælde skal brugeren skrive `vagtplanlægger.exe print <navn på vagtplan>`, for at printe de individuelle vagtplaner til hver medarbejder. Derefter så gemmer programmet en individuel vagtplan til hver medarbejder inde under mappen 'output'. Mappen 'output' skal være lavet før man kører programmet.

6 | Implementering

Nu hvor vi har besluttet os for den algoritme vi vil bruge til at skabe et skema, så kan vi begynde at lave flowcharts og skrive kode til det. Vi vil nu beskrive de vigtigste funktioner i vores program.
Vi har nu defineret krav og hvordan programmet overordnet skal fungere. Derudover har vi beskrevet nærmere hvordan nogle af vores funktioner kommer til at fungere. I dette afsnit kommer vi til beskrive nogle udvalgte funktioner nærmere både hvordan vi har skrevet dem, hvorfor vi har valgt at gøre som vi har gjort og derudover hvordan de fungerer. Først vil vi beskrive de datastrukturer, som vi har gjort brug af da de er vitale for forståelsen for programmet.

I dette afsnit har vi markeret variabler, enums, structs og lignende med en speciel skriftype (`eksempel`) for at forbedre læseevnen da mange af vores variabler har passende navne og derfor kan forveksles når de bliver brugt i sætninger. Derudover har vi taget en beslutning om at variabelnavne og lignende er på engelsk, mens output og lignende er på dansk.

6.1 Datastrukturer

6.1.1 struct Worker

Denne struct repræsenterer de enkelte medarbejdere i vores program, hvor at vagtplanlæggeren har mulighed for at tildele værdier til nogle af medlemmerne igennem en input CSV fil.
Worker har følgende medlemmer:
`name`: navn, maksimalt 50 karakterer langt
`desired_day_off`: ønsket fridag
`desired_shift`: ønsket `vagt`
`last_block`: sidste blok hvor de arbejdede
`consecutive_night_shifts`: antal nætter de har arbejdet i træk

`day_off`: variabel der der viser om en medarbejder har haft fridøgn
`UUID`: et unikt id for hver medarbejder

`UUID` er et unikt id, som hver medarbejder har, og som også vil være synligt på selve vagtplanen. Denne variabel eksisterer for at kunne differentiere imellem flere medarbejdere med samme navn, derudover er det også nemmere at indeksere forskellige arbejdere med et tal end ved en streng.

`desired_shift` og `desired_day_off` er variabler som programmet indlæser ved hjælp af CSV-filen. Det er de variabler der beskriver en medarbejders ønsker.

De andre variabler bliver brugt internt til både generering af start population som beskrevet i 5.3.2 og 6.2.3 og til at udregne deres fitness, som vi beskriver i 5.3.6 og 6.2.4.

6.1.2 struct Schedule

Denne struct bliver brugt til at beskrive et færdigt skema.

Det har følgende medlemmer:
`blocks`: Et array med 21 elementer af typen `BlockSchedule` som er et skema for en bestemt blok

`score`: Skemaets score ifølge vores fitnessfunktion

De 21 elementer i arrayet er sorteret, hvilket vil sige det første element bliver set som skemaet til den første blok altså mandag nat.

6.1.3 struct RequiredWorkers

Denne struct viser, hvor mange medarbejdere der skal være til de forskellige `Shift`. Som vi har forklaret tidligere har vi defineret at vores skemaer skal have et antal medarbejdere i hvert `Shift` lig antallet af krævede medarbejdere. Det er altså det værdierne i `RequiredWorkers` afgør.
Den har følgende medlemmer:
`night_workers`: Antal af krævede medarbejdere til `SHIFT_NIGHT`
`day_workers`: Antal af krævede medarbejdere til `SHIFT_DAY`
`evening_workers`: Antal af krævede medarbejdere til `SHIFT_EVENING`

Hvor `SHIFT` værdierne er de enumeration navne vi har valgt til at beskrive de forskellige vagter.

6.1.4 struct BlockSchedule

Denne struct beskriver et skema til en blok.
Den indeholder kun et element:
`workers`: Et array af `Worker` pointers
Hver pointer i arrayet peger på en `Worker` som arbejder i den blok. På grund af den måde structen er opbygget på hører den ikke til en specifik blok. Det er udelukkende dens position i et array der afgør det.

Antallet af elementer i skemaet er lig `RequiredWorkers` til den passende vagt.

6.2 Funktioner

6.2.1 main

Vores main funktion, tager imod 2 parametre som kan ses på linje 1 i snippet 6.1.

`argc` er det antal af elementer som er i `argv`. `argv` skal ses som et array af strenge.

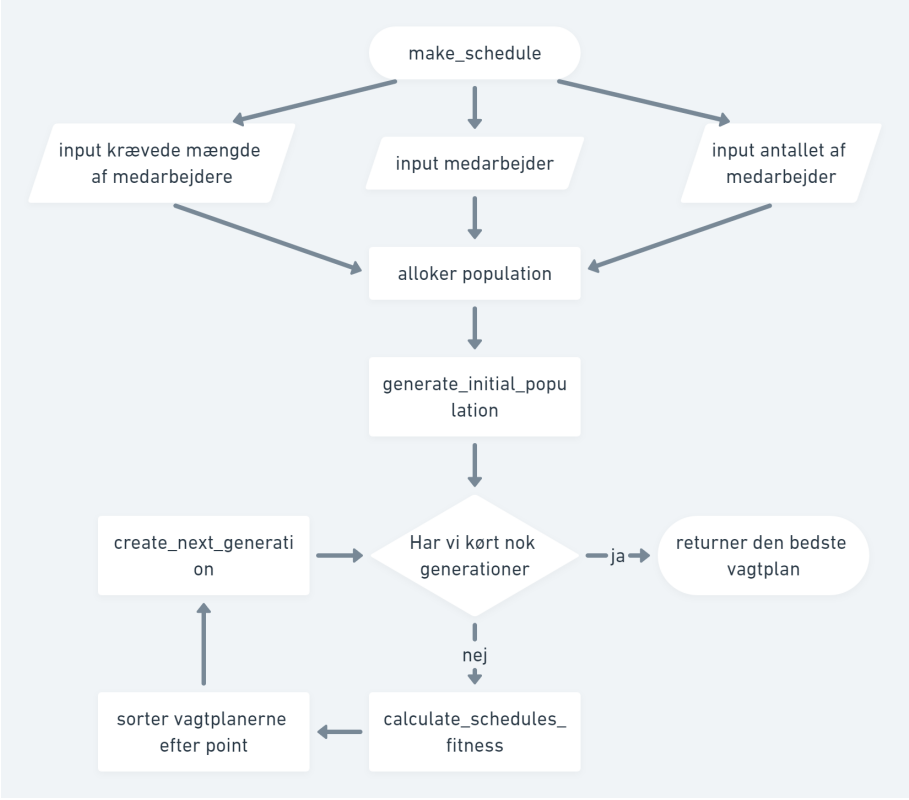
```
1  int main(int argc, char** argv) {
2    if (argc >= 3) {
3      if (strcmp(argv[1], "test") == 0) {
4        test_vagtplan(argv[2]);
5      } else if (strcmp(argv[1], "print") == 0) {
6        print_vagtplaner(argv[2]);
7      } else {
8        printf("Forkert parameter, du kan bruge test eller print\n");
9        return EXIT_FAILURE;
10     }
11   }
12   else {
13     skab_vagtplan();
14   }
15   return 0;
16 }
```

Snippet 6.1. Main funktionen i programmet

Inde i selve main funktionen, har vi en selektiv kontrolstruktur, et if-statement, som bliver eksekveret når `argc` er større eller lig med 3. Det betyder at brugeren har indtastet mere end 2 programparametre. Det første parametre i argv er hvad exe filen hedder, derfor så er der altid mindst et programparameter. Siden brugeren har givet 2 eller flere programparametre, så vil brugeren enten teste vagtplanen eller prøve at printe deres individuelle vagtplaner. Derfor så tester vi på linje 10 om brugeren har skrevet "test"og på linje 12 og brugeren har skrevet "print". Hvis brugeren skriver test, så bruger vi funktion `test_vagtplan` funktionen med programparameter 2 som er filnavnet på den vagtplan som skal testes. Hvis brugeren har skrevet "print", så gør vi præcist det samme bare med `print_vagtplaner`. Hvis brugeren har skrevet en ukendt kommando i programparametre 2, så giver programmet en fejlbesked på linje 15 og returner en fejlkode på linje 16.

Hvis brugeren ikke har givet 2 eller flere programparametre, så bruger vi funktionen `skab_vagtplan`, som skaber en vagtplan ved hjælp af det genetiske algoritme.

6.2.2 make_schedule



Figur 6.1. Flowerchart over vores make_schedule funktion

`make_schedule` funktionen står for at køre den genetiske algoritme. Det er den funktion som står for at holde styr på populationen, lave nye generationer og holde styr på hvad fitness værdien er for populationen. På figur 6.1 er vores flow diagram som i store træk

beskriver `make_schedule` funktionen. Først tager vi de 3 formelle parametre som input til funktionen. Derefter så allokerer vi nok plads til populationen, som er en liste af vagtplaner.

```
Schedule *population = malloc(sizeof(struct Schedule) *
    POPULATION_SIZE);
```

Her allokerer vi nok plads til populationen som er en liste af `Schedule`. Vi har en macro definition af population størrelsen sådan at det er nemmere at ændre senere hvis vi får brug for det.

Derefter kan man se på figur 6.1 at vi så kalder en funktion `generate_initial_population`, som skaber de første tilfældige individer til vores population. Den funktion beskriver vi nærmere ved sektion 6.2.3.

Derefter starter vi en løkke som kører en generation hvert omløb. Denne løkke kører indtil vi har kørt nok generationer. I vores program har vi valgt at den skal køre 100.000 generationer og det har vi vurderet til at være nok. Efter noget tid, så bliver individer mere og mere ens, derfor så bliver hver generation mindre effektiv og fitness scoren forbedres ikke særlig meget mere. Derfor så har vi vurderet at 100.000 generationer er en god grænse.

Den første ting vi gør i løkken er at udregne fitness værdien af hver vagtplan. Det gør vi gennem `evaluate_schedule` som udregner fitness-værdien for et skema. Den er beskrevet nærmere ved sektion 6.2.4

```
1 for (i = 0; i < POPULATION_SIZE; i++) {
2   population[i].score = evaluate_schedule(&population[i],
        required_workers, workers, worker_count);
3 }
```

I snippet 6.2.2 går vi over hver skema i populationen og giver dem en fitness evaluering.

Derefter sorterer vi skemaerne efter deres fitness værdi ved hjælp af `qsort` funktionen fra standard C biblioteket.

```
  qsort(population, POPULATION_SIZE, sizeof(struct Schedule),
      compare_schedule);
```

Her bruger vi så en funktion der hedder `compare_schedule` til at sammenligne to vagtplaner. `compare_schedule` funktionen kan man se på snippet 6.2.

```
int compare_schedule(const void* a, const void* b) {
  const  Schedule* sa = a;
  const  Schedule* sb = b;

  return sb->score - sa->score;
```

```
}
```

Snippet 6.2. `compare_schedule` funktionen

Vi har allerede udregnet fitness så `compare_schedule` bruger det felt i `Schedule` structen som hedder `score`. Så `compare_schedule` funktionen trækker bare de to `score` værdier fra hinanden og finder ud af hvilken der er størst eller om de er ens ved at trække de to tal fra hinanden.

```
1 for (i = 0; i < AMOUNT_OF_BEST_INDIVIDUALS; i++) {
2   int random = random_number(AMOUNT_OF_BEST_INDIVIDUALS,
       POPULATION_SIZE - AMOUNT_OF_BEST_INDIVIDUALS * AMOUNT_OF_CHILDREN);
3   unsigned int j;
4   for (j = 0; j < AMOUNT_OF_CHILDREN; j++){
5       combine_schedule(workers, worker_count, required_workers,
           &population[i], &population[random],
           &population[POPULATION_SIZE - AMOUNT_OF_BEST_INDIVIDUALS *
           AMOUNT_OF_CHILDREN + i * 4 + j]);
6   }
7 }
```

Snippet 6.3. Delen som genererer næste generation

Nu hvor vi har sorteret vagtplanerne, så kan vi begynde at skabe næste generation. I sektion 5.3.4 viser figur 5.3 hvordan den næste generation skabes. Det skal vi så bare implementere i C. Selve koden kan ses i snippet 6.3, her starter vi på linje 1 med at lave en for løkke som skal køre `AMOUNT_OF_BEST_INDIVIDUALS` gange. `AMOUNT_OF_BEST_INDIVIDUALS` er en makro som vi har defineret til 40. På den her måde kører vi igennem de første 40 individer, og siden at listen er sorteret, så kører vi igennem de 40 bedste.

Derefter skal vi vælge et tilfældigt individ som skal parres med det `i` bedste individ. det gør vi via en funktion der hedder `random_number` som laver et tal mellem et minimum og maksimum. på linje 2 vælger vi et tal som er udenfor de individer som bliver kasseret og udenfor de 40 bedste. Så tallet der bliver lavet er mellem `AMOUNT_OF_BEST_INDIVIDUALS` og `POPULATION_SIZE - AMOUNT_OF_BEST_INDIVIDUALS * AMOUNT_OF_CHILDREN`.

Derefter så kører vi en for løkke som kører `AMOUNT_OF_CHILDREN` gange. Den er defineret til 4 i vores program, så hver af de 40 bedste individer får 4 børn. derefter så bruger vi `combine_schedule` funktionen til kombinere disse to skemaer og få et nyt et ud. Den nye genererede vagtplan bliver så gemt der, hvor de kasserede vagtplaner ligger lige nu, nemlig bagerst i arrayet. Når det så er blevet gjort, så har vi en ny generation. Vi har altså ikke to forskellige arrays til den nye generation, de bliver bare gemt i den gamle.

6.2.3 generate_initial_population

For at genopfriske hvordan en genetisk algoritme fungere, så vil en genetisk algoritme kunne lave et vilkårligt antal af generationer. En generation vil være afhængige af den generation der kom før. Derved for at kunne danne fremtidige generationer, så skal der laves en start population. Den bliver dannet ud fra en funktion, som vi har kaldt `generate_initial_population`, som nu vil blive beskrevet.

Først beskriver vi funktionens input og output:

```
1 void generate_initial_population(
2     struct RequiredWorkers required_workers,
3     struct Worker *worker[],
4     unsigned int worker_count,
5     struct Schedule schedules[],
6     unsigned int population_size
7 )
```

Snippet 6.4. Prototype for funktionen `generate_initial_population`

Funktionen skal have en liste af de medarbejdere som input, hvorefter funktionen laver skemaer ud fra de medarbejdere der er blevet givet, denne liste er et array af `Worker` pointers, som funktionen bruger til at tilgå de forskellige medarbejdere og tildele dem et skema.

En generation består af 1.000 individer. Derfor skal der genereres 1.000 skemaer til start-populationen. Dette har vi gjort ved at lave en hjælpefunktion som der laver et skema, funktionen hedder `generate_random_schedule` som nu vil blive beskrevet.

```
1 void generate_random_schedule( Worker* workers[],
2 const unsigned int worker_count,
3 const  RequiredWorkers required_workers,
4 Schedule* schedule);
```

Snippet 6.5. Prototype for funktionen `generate_random_schedule`

`generate_random_schedule` tager imod disse formelleparametre som kan ses i snippet 6.6.

Det bemærkes at funktionen er af typen void, det vil sige at funktionen ikke returnerer nogen værdi. I denne funktion er det opsat sådan at der er input parametre og et output parameter. Output parameteren i denne funktion er `schedule`, resten af de formelleparametre er input parametre.

```
1 for (day = 0; day < 7; day++) {
2     unsigned int workers_top = worker_count;
```

```
3     int shift;
4     for (shift = 0; shift < 3; shift++) {
5       int required_workers_for_shift =
            get_required_for_shift(required_workers, (enum Shift) shift);
6       unsigned int worker_index;
7       schedule->blocks[day * 3 + shift].workers =
            malloc(required_workers_for_shift * sizeof(struct Worker*));
8       if (schedule->blocks[day * 3 + shift].workers == NULL) {
9         fatal_error("Kunne ikke allokere mere hukommelse");
10      }
11
12      for (worker_index = 0; worker_index < required_workers_for_shift;
            worker_index++) {
13
14        int random_index = random_number(0, workers_top);
15        Worker* tmp = NULL;
16        if (workers_top <= 0) {
17          fatal_error("Ikke nok medarbejdere til at lave en valid plan
                for en dag");
18        }
19
20        schedule->blocks[day * 3 + shift].workers[worker_index] =
              workers[random_index];
21
22        tmp = workers[random_index];
23        workers[random_index] = workers[workers_top - 1];
24        workers[workers_top - 1] = tmp;
25        workers_top--;
26      }
27    }
28  }
```

Snippet 6.6. kode til `generate_random_schedule`

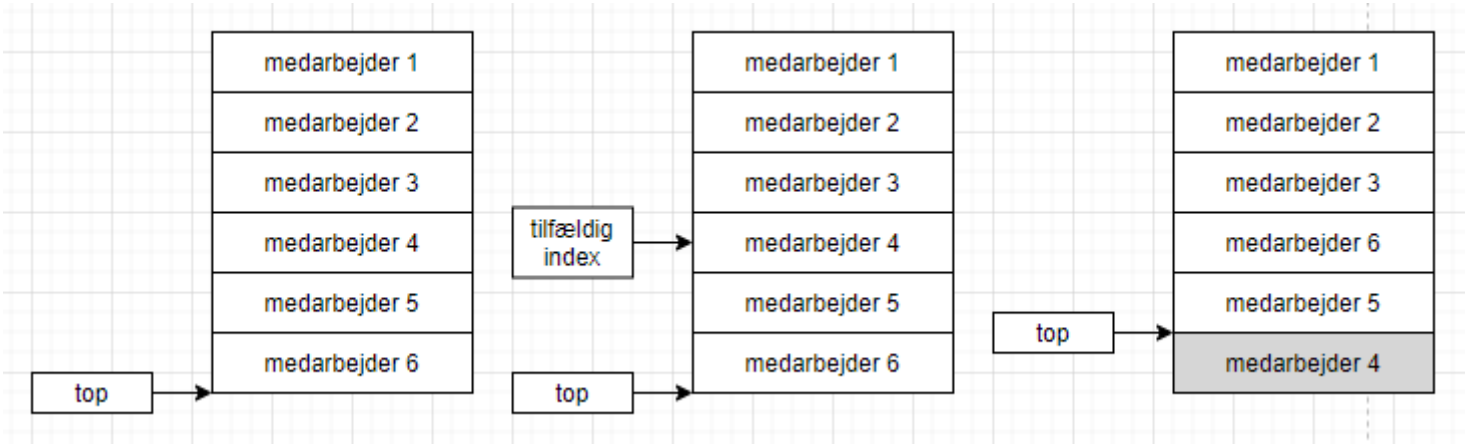
Metoden som bliver brugt til at danne skemaet er således: Der startes med en forløkke der itererer over alle ugens dage som kan ses i snippet 6.6 på linje 1.

På linje 6 i snippet 6.6 ses der en forløkke som itererer over de 3 vagter der er på en dag.

I forløkken på linje 15 i snippet 6.6, gennemløbes kroppen af forløkken indtil der er blevet tilføjet nok medarbejdere. Vi har tidligere sagt at når start-generationen bliver genereret så skal den samme medarbejder ikke arbejde to gange på en dag. Derfor så må vi nødt til at lave et form for algoritme til at gøre dette effektivt. Først så vælger vi et indeks mellem 0 og `workers_top`. Top er blevet defineret på linje 3 til at være `worker_count`. Derefter så tjekker vi om `workers_top` er 0 på linje 20, i så fald er vi løbet tør for medarbejdere

til denne dag og så giver vi en fejlmeddelelse.

Derefter så bruger vi det tilfældige indeks vi har lavet `random_index` som indekset til en tilfældig medarbejder. Derefter tilføjer vi den medarbejder til den nuværende blok. Så trækker vi 1 fra `worker_top`. Så bytter vi den tilfældige medarbejder ud med den ved `workers_top - 1`. Nu kan vi ikke vælge den medarbejder igen. Dette er også illustreret på figur 6.2.



Figur 6.2. Selektion af unik medarbejder

Dette algoritme kræver at vi kan flytte rundt på medarbejderne i medarbejderlisten uden at invalidere andre blokke. Det er så grunden til at vores worker array er en array af worker pointers istedet for bare at være workers.

For at vende tilbage til hoved funktionen `generate_initial_population`, så har vi nu vist hvordan man laver et individ. For at generere en hel population, så sættes `generate_random_schedule` i en forløkke, som kører 1000 gange. Dette kan ses i snippet 6.7. Dette vil sige når funktionen `generate_initial_population` har kørt, så vil der være genereret 1.000 individer.

```
int i;
for (i = 0; i < population_size; i++) {
  generate_random_schedule(worker, worker_count, required_workers,
      &schedules[i]);
}
```

Snippet 6.7. Forløkke til at generere 1.000 individer

6.2.4 evaluate_schedule

Vores fitnessfunktion har vi kaldt evaluate_schedule. Prototypen til funktionen ser sådan her ud:

```
double evaluate_schedule(Schedule* schedule, const RequiredWorkers
    required_workers, Worker* worker[], unsigned int amount_of_workers);
```

Snippet 6.8. Prototype for evaluate_schedule

Funktionens output er en double der beskriver kvaliteten af det skema der bliver givet som input. Al evaluering i vores fitnessfunktion er diskret hvilket vil sige at funktionen ligeså godt kunne returnere en integer eller lignende. Det første formelleparameter er schedule som der er det skema der skal evalueres givet som pointer. Typen er `Schedule` som vi har beskrevet tidligere.

Det næste parameter er `required_workers` som der er af typen `RequiredWorkers` som forklaret tidligere. Det parameter viser hvor mange medarbejdere der er til hver vagt. Det næste parameter er `worker` som er et array af `Worker` pointers. Det hænger sammen med det sidste parameter `amount_of_workers` som der er antallet af `Worker` pointers i `worker` arrayet.

Målet med fitnessfunktionen er at undersøge om hvorvidt vores bløde og hårde krav bliver overholdt i det givne skema. Derfor er vores funktion mere eller mindre delt op i en del til hvert krav.

Vores funktion er opbygget i for løkker til at løbe igennem skemaet. Som forklaret tidligere ser vi en uge som 21 blokke og et skema er 21 blokskemaer. For at give et bedre overblik og nemmere programmering har vi delt det op i dage og vagter i funktionen. Den første forløkke er i linje 12 i Snippet 6.9.I Snippet 6.9 er den del af koden der sørger for løkkerne og de variabler der bliver tildelt værdier i hver løkke med. Der kommer flere Snippets til de forskellige dele af funktionen der håndterer de forskellige krav.

```
1  double evaluate_schedule(Schedule* schedule, const RequiredWorkers
       required_workers, Worker* worker[], unsigned int amount_of_workers){
2    unsigned int day, shift, worker_number;
3
4    unsigned int worker_i = 0;
5    for (worker_i = 0; worker_i < amount_of_workers; worker_i++) {
6      worker[worker_i]->last_block = -10;
7      worker[worker_i]->consecutive_night_shifts = 0;
8      worker[worker_i]->day_off = -1;
9    }
10
11   schedule->score = 0;
12   for (day = 0; day < 7 ; day++) {
13
14     for (shift = 0; shift < 3; shift++) {
15       unsigned int block_number = day * 3 + shift;
```

```
16        unsigned int workers_needed =
              get_required_for_shift(required_workers, shift);
17        Worker** current_worker_array =
              (schedule->blocks[block_number]).workers;
18
19        for (worker_number = 0; worker_number < workers_needed;
              worker_number++) {
20          Worker* current_worker = current_worker_array[worker_number];
21          enum Day last_day;
22
23          if (current_worker->day_off == -1) {
24            current_worker->day_off = 0;
25          }
26
27          if (current_worker->last_block < 0) {
28            last_day = DAY_INVALID;
29          } else {
30            last_day = current_worker->last_block / 3;
31          }
32                  [Kode til de enkelte krav som bliver gennemgået senere]
33        }
34      }
35    }
36    for (worker_i = 0; worker_i < amount_of_workers; worker_i++) {
37      if (worker[worker_i]->day_off == 0) {
38        if (!(worker[worker_i]->last_block > 0 && 21 -
              worker[worker_i]->last_block > 5)) {
39          schedule->score -= 1000;
40        }
41      }
42    }
43
44    return schedule->score;
45  }
```

Snippet 6.9. Forløkker og initialisering af variable i `evaluate_schedule`

Den kører altså 7 gange og `day` styrer den dag der bliver tjekket. Der er altså lighed imellem `enum Day` og `day` hvilket vi udnytter i funktionen. Inde i den løkke er der endnu en løkke som er på linje 14 i Snippet 6.9. Den løkke kører altså 3 gange og styrer hvilken vagt der bliver tjekket og passer ligeledes med `enum Shift`. Med de to løkker bliver ting inden i den sidste løkke gennemløbet 21 gange, hvilket altså passer til det antal blokke vi gerne vil igennem. For at holde styr på den nuværende blok har vi lavet variablen `block_number`.

Vi bruger den til at indeksere `schedule.blocks` hvilket vil sige at den starter fra 0, altså er det reele bloknummer lig `block_number + 1`.

Inde i `shift`-løkken har vi en løkke der går alle medarbejderne fra den bloks skema igennem. Som beskrevet tidligere er antallet af medarbejdere i en blok lig `required_workers` til det `Shift` som blokken har.

`requried_workers` har tre medlemmer: `night_workers`, `day_workers` og `evening_workers`. Vi har en funktion, `get_required_for_shift`, som der returnerer værdien i `required_workers` fra et `Shift`.

Den tredje løkke er på linje 19 i Snippet 6.9.

Nu har vi beskrevet hvordan løkkerne fungerer. Alle de krav vi tjekker bliver gjort inde i det inderste loop det vil sige at de bliver kørt det samme antal gange som der er medarbejdere der arbejder på en uge. Den eneste undtagelse er et enkelt tjek som der ikke ligger i en løkke som vi beskriver nærmere senere.

I vores kode har vi en variabel som er en pointer til den Worker som der bliver arbejdet med i den sidste løkke. I de følgende beskrivelser kommer den også til at blive refereret til. Det er `current_worker` på linje 20 i Snippet 6.9.

6.2.4.1 Hårde krav

Her vil vi gennemgå den del af funktionen der tjekker de hårde krav. Medmindre andet er skrevet ligger det inden i den tredje forløkke som starter på linje 19 i Snippet 6.9.

Den del af funktionen der tjekker om 11-timers reglen er overholdt ser sådan her ud:

```
1 if(block_number - current_worker->last_block <= 2 &&
     current_worker->last_block >= 0){
2 schedule->score -= 1000;}
```

Snippet 6.10. 11 timers reglen

11 timers reglen betyder i forhold til blokke at en vagt skal være mindst 3 blokke fra den forrige, altså 16 timer fra den forrige vagt, det er det den første del af det logiske udtryk viser. Det andet udtryk er for at tjekke om medarbejderen har arbejdet før denne uge.

En Workers `last_block` starter med at være -10 så derfor bliver vi nødt til at tjekke om den er mindst 0. Hvis `last_block` er mindre end 0 betyder det at det er arbejderens første arbejdsdag og derfor kan de ikke overtræde 11-timers reglen. Hvis den if-sætning returnerer 1 uddeler vi minus 1.000 point for at overtræde et hårdt krav.

Vi tjekker også om en medarbejder opfylder de cykliske krav. De cykliske krav betyder at hvis man arbejder om natten må man ikke arbejde aften den næste dag. I den situation er antallet af blokke imellem de to vagter lig 5. Hvis man arbejder om dagen må man ikke arbejde om natten den næste dag. Men det må man heller ikke ifølge 11 timers reglen da der kun er en vagt imellem de to, derfor behøver vi ikke tjekke det her. Hvis man arbejder om morgenen må man ikke arbejde nat eller dag næste dag. Begge de ting er allerede udelukket af 11 timers reglen. Det vil sige det eneste vi skal tjekke er hvis antallet af blokke imellem `block_number` og `current_worker->last_block` er lig 5 og at `current_worker->last_block` var den forrige dag, så skal der gives minuspoint. Den del af funktionen ser sådan her ud:

```
1 if(block_number - current_worker->last_block <= 2 &&
     current_worker->last_block >= 0){
2 schedule->score -= 1000;}
```

Snippet 6.11. Tjekker efter cykliske brud

Den næste del af funktionen tjekker om en medarbejder har haft for mange nattevagter i træk. Ifølge vores krav må man maksimalt have to nattevagter i træk. Til at hjælpe med at gøre det effektivt har vi indført felt `consecutive_night_shifts` til `Worker` struct'en som har værdien af antallet af natte-vagter medarbejderen har haft i træk. Vores funktion tjekker først om det nuværende `shift` er nat. Hvis det er det, tjekker den om medarbejderens forrige vagt også var nat. Hvis den var det bliver der tjekket om `consecutive_night_shifts` er mindst to, da de vil betyde at denne vagt bliver den tredje vagt i træk hvilket giver minus-point. Hvis ikke den forrige vagt var en nattevagt bliver `consecutive_night_shifts` sat til 0. Derefter ude af if-else sætningen incrementer vi `consecutive_night_shifts` med en på grund af den nuværende vagt. Hvis ikke `shift` er nat bliver `consecutive_night_shifts` sat til 0. I c-kode ser det sådan her ud:

```
1  if (shift == SHIFT_NIGHT) {
2    if (current_worker->last_block == block_number - 3){
3      if (current_worker->consecutive_night_shifts >= 2){
4        schedule->score -= 1000;
5      }
6    }
7      else{
8        current_worker->consecutive_night_shifts = 0;
9        }
10     current_worker->consecutive_night_shifts += 1;
11 }
12 else{
13   current_worker->consecutive_night_shifts = 0;
```

```
14 }
```

Snippet 6.12. Tjekker nattevagter i træk

Grunden til at vi har kontrolstrukturen i linje 2 er for at tjekke om medarbejderes sidste dag var den forrige dag. Hvis at en medarbejder arbejder f.eks. en morgen vagt bliver `consecutive_night_shifts` sat til 0. Men hvis medarbejderen har et fridøgn forbliver `consecutive_night_shifts` uændret. Men ifølge vores krav svarer en fridag til at have en anden vagt hvor man altså gerne må have to nattevagter i træk efter et fridøgn.

Det næste vi tjekker er fridøgn. Et fridøgn er 35 timer uden nogle vagter. Det vil sige at en medarbejder har haft et fridøgn hvis de har haft mindst 6 blokke i træk uden nogle vagter. Det vil sige vi bare kan tjekke om `block_number` - `current_worker->last_block` er mindst 5. Dog siden at en medarbejders `last_block` er lig -10 hvis de ikke har arbejdet i løbet af skemaet skal vi istedet tjekke om `block_number` er over 4. Igen er grunden til at det er 4 og ikke 5 fordi at `block_number` er 0 indekseret. Vi har programmeret det således:

```
1 for (worker_i = 0; worker_i < amount_of_workers; worker_i++) {
2     if (worker[worker_i]->day_off == 0) {
3       if (!(worker[worker_i]->last_block > 0 && 21 -
            worker[worker_i]->last_block > 5)) {
4         schedule->score -= 1000;
5       }
6     }
7   }
```

Snippet 6.13. Fridøgn tjek

Grunden til at der bliver lagt 1 til, i linje 1, hvis medarbejderen ikke har haft en vagt denne uge, er fordi at `block_number` er 0-indekseret. Det betyder altså at hvis en medarbejder har deres første vagt i `block_number` 5 har de haft et fridøgn. Det behøver vi ikke tage højde for når `current_worker->last_block` er højere end -1 da de begge er 0-indekseret.

Det er den første del af den del af funktionen der tjekker om der har været fridøgn. Den anden del er ude af forløkkerne på linje 36 til 42 i Snippet 6.9. Som forklaret tidligere er værdien af en medarbejders medlem `day_off` lig 1 hvis de har haft et fridøgn, den er 0 hvis de har haft mindst en vagt i skemaet og -1 hvis de ikke har haft nogle vagter. Vi indsætter den del af funktionen for læsbarhed:

```
1 for (worker_i = 0; worker_i < amount_of_workers; worker_i++) {
2   if (worker[worker_i]->day_off == 0) {
3       if (!(worker[worker_i]->last_block > 0 && 21 -
            worker[worker_i]->last_block > 5)){
```

```
4           schedule->score -= 1000;
5       }
6   }
7 }
```

Snippet 6.14. Tjekker overtrædelse af fridøgn

If-sætningen på linje 3 tjekker om en medarbejder har haft et fridøgn i slutningen af ugen og ikke har haft en vagt siden. Hvis de ikke har haft en vagt siden er værdien af medarbejderens `day_off` ikke blevet opdateret.

6.2.4.2 Bløde krav

Vores bløde krav er meget simple at tjekke og det er kun nogle få linjer inden i den tredje forløkke. Når vi tjekker de bløde krav med ønsket fridag og foretrukne vagt tjekker vi om medarbejderens nuværende vagt er deres foretrukne, i hvilket fald scoren bliver 1 højere. Derefter tjekker vi om de arbejder på deres ønskede fridag hvor scoren bliver 2 mindre hvis de gør. Koden ser sådan her ud:

```
1 if (current_worker->desired_shift == shift){
2   schedule->score += 1;
3 }
4 if (current_worker->desired_day_off == day){
5   schedule->score -= 2;
6 }
```

Snippet 6.15. Tjekker efter foretrukne vagter og ønskede fridage

Når alle disse krav er blevet testet for alle medarbejdere i alle blokke har vi tjekket alle de krav vi vil. Det eneste der er tilbage i funktionen er at give minus point for manglende fridøgn og at returnere den endelige score. På dette tidspunkt i funktionen er hele scoren registreret. Så vi returnerer den værdi. Da scoren også er i `Schedule` medlemmet score kan scoren også blive genfundet senere.

6.2.5 Combine_Schedule

Denne funktion har formålet at lave crossover på de individer, som der bliver genereret i programmet. Prototypen for funktionen ser således ud:

```
1 void combine_schedule(Worker* workers[], size_t worker_count,
    RequiredWorkers required_workers, const  Schedule* a, const
    Schedule* b,  Schedule* out);
```

Snippet 6.16. Prototype for funktion combine_schedule

I funktionen starter vi med at initialisere 2 heltals variabler, som vi tildeler en tilfældig værdi med vores `random_number` funktion, som kommer til at være start og slutpunktet for individernes crossover. Herefter assigner vi `a_block` og `b_block` til `blocks` medlemmet fra skemaerne `a` og `b` i `blocks`. Vi initialiserer også en variabel som er en pointer der peger på det skema som vi gerne vil lave. Det er medlemmet `blocks` i `out` som er et outputparameter. Nu bruger vi en iterativ kontrolstruktur, en for-løkke til at kopiere data fra a og b blokkene, over i vores output skema.

```
1 for (j = 0; j < 21; j++) {
2     if (j >= crossover_start && j < crossover_end) {
3       memcpy(out_block[j].workers, b_block[j].workers,
            get_required_for_shift(required_workers, j % 3) *
            sizeof(Worker*));
4     } else {
5       memcpy(out_block[j].workers, a_block[j].workers,
            get_required_for_shift(required_workers, j % 3) *
            sizeof(Worker*));
6     }
7   }
```

Snippet 6.17. Kopiering fra x_block til out_block

For-løkken kører 21 gange for de 21 blokke i et skema. Den sørger for at lave crossover imellem de to individer. Det nye skema bliver lig de blokke fra `a_block` hvor at bloknummeret ikke er inde for intervallet `crossover_start` - `crossover_slut`. De blokke der ligger i intervallet bliver lig blokkene fra `b_block`. Det vi bruger til at kopiere er funktionen 'memcpy' som der kopierer et antal bytes fra en pointer over til en anden. Vi kopierer data fra enten `a_block` eller `b_block` som beskrevet før. Antallet af bytes er det antal som der skal arbejde den vagt. `get_required_for_shift` finder det krævede antal medarbejdere for en vagt hvor det ene parameter er en variabel af typen `RequiredWorkers` og den anden er et tal imellem 0 og 2 som der svarer til en vagt.

Den blok der skal tildeles en ny værdi er blok nummer `j` så den nuværende vagt er `j` % 3. Så kan vi lave funktionskaldet `get_required_for_shift(required_workers, j % 3)` for at finde det antal medarbejdere der skal være i blokken og derefter gange det med `sizeof(Worker)` for at få det antal bytes som skal kopieres.

Nu har funktionen lavet crossover. Den næste del af funktionen giver en chance for mutation. Vi har valgt at der skal være en tredjedelschance for mutation. Vi har derfor lavet en if-sætning der tjekker om et tilfældigt tal % 3 er lig 0.

```
if (rand() % 3 == 0) {
```

Vi fortsætter nu med en for-løkke, som siger følgende.

```
for (i = 0; i < needed_workers; i++)
```

Hvis vi kommer ind i den if-sætning er der en for-løkke som der sørger for mutation. Det vi gør er først at generere et tal imellem 0 og 21 som der afgør hvilken blok der muterer. Derefter finder vi ud af hvor mange medarbejdere der er i blokken. Forløkken kører det antal gange.

Inde i forløkken går den de nuværende medarbejdere igennem og erstatter dem med en tilfældig medarbejder fra `workers` arrayet imellem 0 og variablen `top` som er blevet initialiseret til at være lig `worker_count`. Efter en tilfældig medarbejder er blevet sat ind i blokken bliver den valgte medarbejder og medarbejder nummer `top - 1` i `workers` byttet rundt. Derefter bliver `top` decrementet med 1.

Det fortsætter indtil at alle de krævede medarbejdere er blevet erstattet med en tilfældig arbejder fra `workers`. Hele den del af funktionen kan ses i snippet 6.18

```
1    if (rand() % 3 == 0) {
2    int random_block_index = random_number(0, 21);
3    int i;
4    int top = worker_count;
5    int needed_workers = get_required_for_shift(required_workers,
         random_block_index % 3);
6    for (i = 0; i < needed_workers; i++) {
7      int random_index = random_number(0, top);
8      Worker* tmp = NULL;
9      if (j <= 0) {
10       fatal_error("Ikke nok medarbejdere til at lave en valid plan
             for en dag");
11     }
12     out_block[random_block_index].workers[i] = workers[random_index];
13     tmp = workers[random_index];
14     workers[random_index] = workers[top - 1];
15     workers[top - 1] = tmp;
16     top--;
17   }
18 }
```

Snippet 6.18. Delen af `combine_schedule` funktionen der stå for at mutere en blok

7 | Test af program

Nu har vi implementeret hele programmet, så nu vil vi teste programmet for at se om det overholder de krav som vi satte i starten af processen.

7.1 Test af fitness funktion

Når man skal teste funktioner og programmer hvori der indgår tilfældighed er det svært definitivt at konkludere hvorvidt at programmet fungerer som ønsket. Man kan forsøge at gøre det på flere forskellige måder. En af de måder vil være at teste de enkelte funktioner og argumentere for at de tilfældige elementer i funktionerne overholder de krav der er for funktionen, og på den måde har man sandsynliggjort at funktionen virker efter hensigten. Den metode vi har valgt at bruge på grund af den måde vores program er opbygget på er at teste vores fitness-funktion. Det kan vi relativt simpelt gøre ved at inputte et antal skemaer som vi manuelt har udregnet fitness-værdien på og teste om den aktuelle værdi er lig den forventede værdi, altså en form for blackbox-testing.

Fitness funktionen testes indtil det virker som det skal og derefter kan vi køre vores algoritme til at generere skemaer. Vi kan få programmet til at vise fitness-værdierne af skemaerne for hver generation. Hvis vores fitness-funktion viser at skemaerne generelt bliver bedre fra hvordan de startede i mange tests har vi sandsynliggjort at vores algoritme fungerer korrekt. Det gode ved denne metode er at man hurtigt kan se hvorvidt at den skrevne algoritme giver fornuftige resultater. Derudover er den slags test nemmere og hurtigere at lave. Hvilket er grunden til at vi har valgt denne testtype da vores tidshorisont i projektet er relativt kort. Ulempen ved denne metode er at den potentielt kan give negativt resultat, fordi at skemaerne ikke bliver bedre, men grunden til dette er ikke på grund af fejl i programmet, men derimod at det program man har designet ikke fungerer teoretisk set.

Til test af vores fitness-funktion benytter vi os af to skemaer. Det ene overtræder ikke nogle hårde krav og overholder et antal af bløde krav.

Til dette har vi lavet 12 tilfældige medarbejdere som skal bruges til at lave disse 2 skemaer, listen kan ses i bilag 2.

I tabel 7.1 kan den dårlige vagtplan ses. Der er nogle af medarbejderne som har samme navn, derfor så har vi noteret UUID sammen med navnet for at gøre det muligt at kende forskel.

	Mandag	Tirsdag	Onsdag
00:00 - 08:00	Shannon Casady.0 Donette Worthey.9	Tanika Burstein.1 Nicol Pasko.8	Shannon Casady.0 Quintin Bratt.2
08:00 - 16:00	Tanika Burstein.1 Quintin Bratt.2 Celestine Mcclean.4	Tanika Burstein.1 Quintin Bratt.2 Nicol Pasko.8	Tanika Burstein.1 Celestine Mcclean.4 Clarisa Satchell.5
16:00 - 24:00	Donette Worthey.6 Donette Worthey.9	Celestine Mcclean.4 Donette Worthey.9	Clarisa Satchell.5 Donette Worthey.9

Torsdag	Fredag	Lørdag	Søndag
Shannon Casady.0 Shaniqua Canez.7	Shannon Casady.0 Shaniqua Canez.7	Shannon Casady.0 Celestine Mcclean.4	Shannon Casady.0 Celestine Mcclean.4
Tanika Burstein.1 Quintin Bratt.2 Nicol Pasko.8	Tanika Burstein.1 Celestine Mcclean.4 Nicol Pasko.8	Rita Hursh.3 Quintin Bratt.2 Shaniqua Canez.7	Tanika Burstein.1 Clarisa Satchell.5 Shaniqua Canez.7
Celestine Mcclean.4 Donette Worthey.9	Rita Hursh.3 Clarisa Satchell.5	Celestine Mcclean.4 Tanika Burstein.1	Quintin Bratt.2 Rita Hursh.3

Tabel 7.1. Dårlig vagtplan ud fra de 12 medarbejdere i bilag 2

Så har vi vurderet dette skema via de krav som vi har stillet op i sektion 5.3.6. Udfra det, er vi kommet frem til en forventet fitness på -17995, efter vi testede det med programmet, så kom vi frem til samme score.

Det næste skema vi lavede var et godt skema som overholdte alle de hårde krav og hvor vi sørgede for at overholde en del bløde krav også.

	Mandag	Tirsdag	Onsdag
00:00 - 08:00	Donette Worthey.9 Roseanna Tutson.19	Donette Worthey.9 Celestine Mcclean.4	Clarisa Satchell.5 Sallie Richart.18
08:00 - 16:00	Quintin Bratt.2 Clarisa Satchell.5 Shaniqua Canez.7	Quintin Bratt.2 Nicol Pasko.8 Shaniqua Canez.7	Quintin Bratt.2 Nicol Pasko.8 Shaniqua Canez.7
16:00 - 24:00	Tanika Burstein.1 Rita Hursh.3	Tanika Burstein.1 Rita Hursh.3	Tanika Burstein.1 Donette Worthey.6

Torsdag	Fredag	Lørdag	Søndag
Donette Worthey.9 Celestine Mcclean.4	Donette Worthey.9 Celestine Mcclean.4	Sallie Richart.18 Roseanna Tutson.19	Sallie Richart.18 Roseanna Tutson.19
Quintin Bratt.2 Nicol Pasko.8 Shaniqua Canez.7	Quintin Bratt.2 Clarisa Satchell.5 Shaniqua Canez.7	Clarisa Satchell.5 Nicol Pasko.8 Celestine Mcclean.4	Clarisa Satchell.5 Donette Worthey.9 Celestine Mcclean.4
Tanika Burstein.1 Rita Hursh.3	Tanika Burstein.1 Rita Hursh.3	Donette Worthey.6 Rita Hursh.3	Donette Worthey.6 Nicol Pasko.8

Tabel 7.2. God vagtplan ud fra de 12 medarbejdere i bilag 2

Ved den gode vagtplan (tabel 7.2) fik vi en fitness score på 35 når vi udregnede det i hånden. Programmet gav også en fitness score på 35.

Ud fra det konkluderer vi at vores fitness funktion er implementeret korrekt i forhold til det vi definerede at den skal gøre.

7.2 Test af genererede vagtplaner

Eftersom fitness-funktionen er blevet valideret. Skal programmet som helhed testes, via endnu en blackbox test. Formålet med denne test er at teste om programmet kan finde en god vagtplan ud fra en liste af medarbejdere. Da programmet kan håndtere vilkårligt mange medarbejdere, så vil denne test give en indikation på om programmet kan generere en valid vagtplan. Selvom programmet kan håndtere vilkårligt mange medarbejdere, så har vi valgt at have et lille antal af medarbejdere på kun 10 for at programmet kan køre hurtigere og derved nå at generere en stor nok mængde data. Dog skal det noteres at programmet ikke er blevet testet med flere end 10 medarbejdere, derved kan kvaliteten af de genererede vagtplanner ikke garanteres, hvis antallet af medarbejdere er over 10. Listen af medarbejdere kan ses i bilag 5. Antallet af medarbejderne er udvalgt til at der hver dag vil være et konstant antal af medarbejdere, hvor medarbejderne er fordelt således: Nat 2, dag 3 og aften 2.

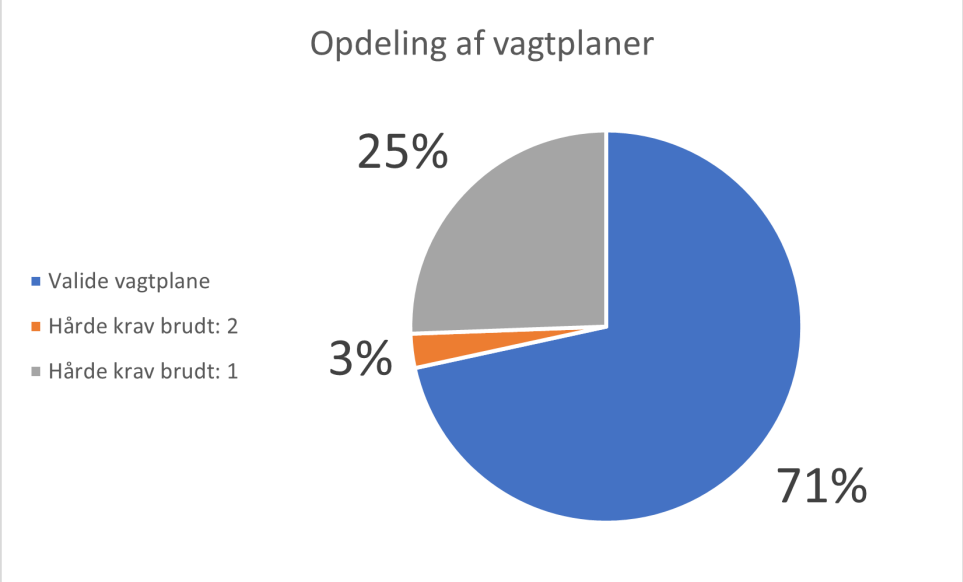
Testen af programmet forgår således: Programmet får 10 medarbejdere som medarbejder

listen. Programmet laver 100.000 generationer, hvor den bedste vagtplan fra den sidste generation bliver gemt i en CSV fil.

Følgende ting bliver også gemt:

- Filnavn hvor kørselsnummer indgår i navnet (kørselsnummer er det antal gange programmet har kørt +1)
- Scoren for vagtplanen
- Hvor mange hårde krav der bliver brudt
- Antal af opfyldte ønskede vagter
- Ønskede fridage der ikke blev opfyldt

Dette bliver gjort 1.000 gange, derved vil der være 1.000 vagtplaner som der laves en analyse af. Det skal noteres at listen af medarbejdere er den samme hver gang.



Figur 7.1.

Ud fra det genererede data, er der blevet lavet et cirkel diagram ud fra hvor mange valide og ikke valide vagtplaner der fremkommer i dataen. Cirkeldiagrammet ses i figur 7.1. Chancen for at programmet genererer en valid vagtplan er 71%. Dette vil betyde at cirka hver fjerde gang programmet producerer en vagtplan, så vil vagtplannen ikke være valid. Det skal bemærkes, at størstedelen af de vagtplaner der bryder hårde krav, bryder kun et hårdt krav. Hvor kun 3% bryder mere end et hårdt krav.

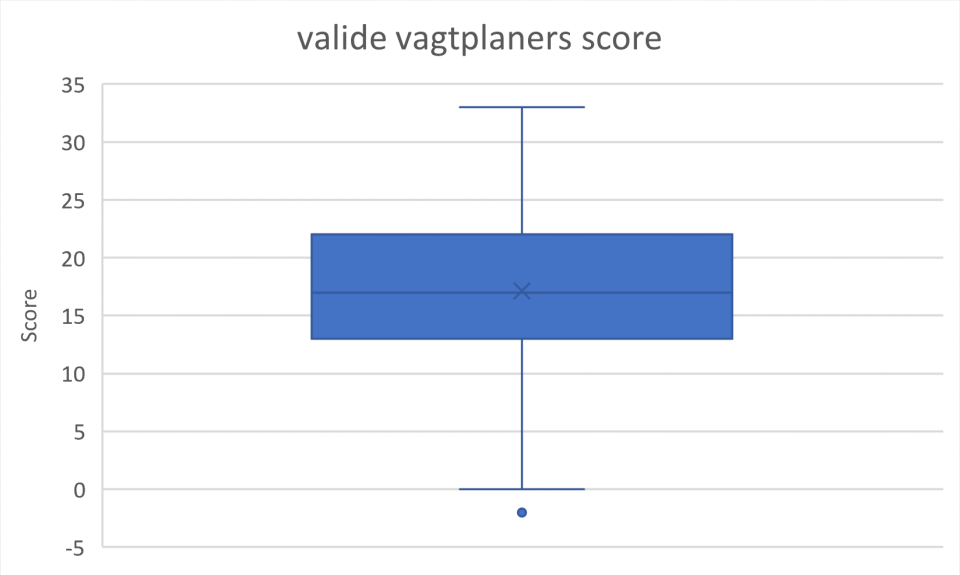
Antal vagtplaner	Gennemsnits score	Minimum score	Q1	Median	Q3	Maks
1000	-301.85	-1996	-977	14	20	33

Tabel 7.3. Minimum er den mindste værdi i datasættet. Mellem Minimum og Q1, svares til 25% af alle scores. Q1 til median svarer til 50%. Median til Q3 svarer til 75%. Q3 til Maks svarer til 100%

Ud fra tabellen 7.3, det bemærkes at en stor del af dataen ligger tæt i intervallet 14-33. Eftersom der en betydelig forskel imellem de valide vagtplaner og de ikke valide vagtplaner, så analysers der med fokus på de valide vagtplaner.

7.2.1 Valide vagtplaner

I dette afsnit vil de valide vagtplaner blive analyseret.

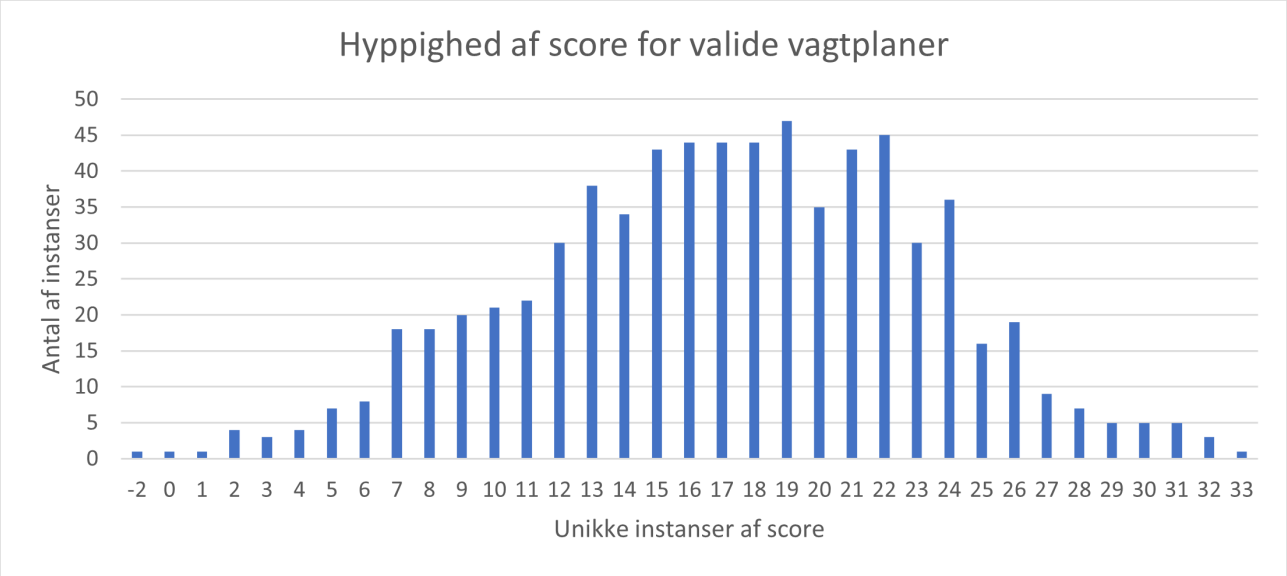


Figur 7.2. Boksplot over valide vagtplanners score

Antal vagtplaner	Gennemsnits score	Minimum score	Q1	Median	Q3	Maks
711	17.15	-2	13	17	22	33

Tabel 7.4. Statistik over valide vagtplaners score

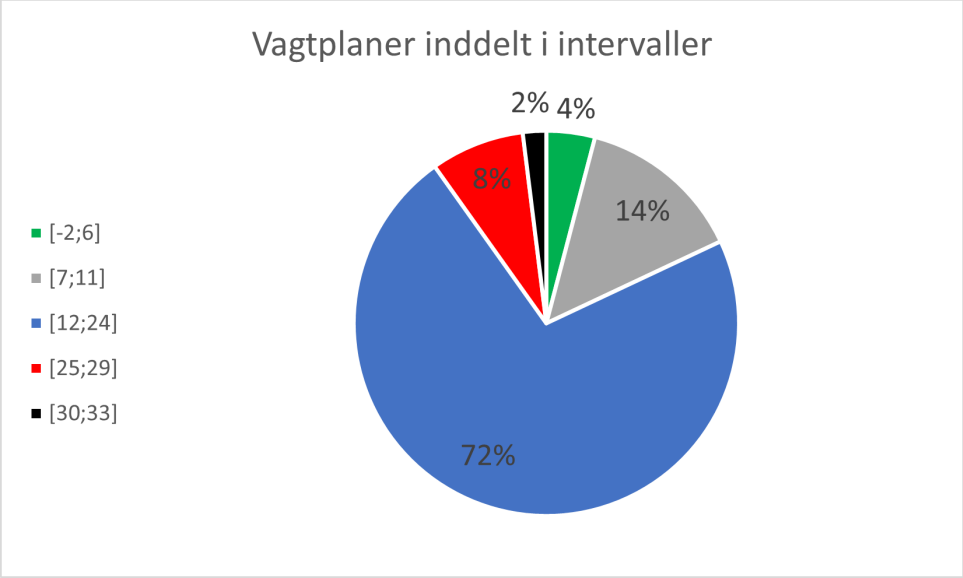
Det bemærkes at i tabellen 7.4 at gennemsnits scoren og medianen er identiske, hvis der afrundes til nærmeste heltal. Det er svært at finde ud af hvor gode skemaerne er hvis vi bare kigger på alle. Derfor kan man kigge på dem der opfylder alle de hårde krav så man kan få et bedre indblik i hvor mange point skemaer får. Intervallet for de valide vagtplaners score er imellem -2 til 33, dette er visualiseret i 7.2, her kan det ses at 50% af alle de valide vagtplaners score ligger i intervallet 13 til 22. Dette betyder programmet har stor sandsynlighed for generere en vagtplan med en score imellem 13 til 22, med de input vi har givet i denne test. Dette understøttes af hyppigheden af scoren for de valide vagtplaner som kan ses her på figur 7.3.



Figur 7.3. Søjlediagram for hyppighed af unikke scores

Figuren 7.3 viser hyppighedden af alle de unikke scores for de valide vagtplaner. Det bemærkes at der ikke er mange instanser af scores i intervallet [-2;6] eller i intervallet [27;33]. Størstedelen af instanserne findes i intervallet [7;26]. Dette kan dog indelles mere hvor det interval hvor der er størst sandsynlighed for en score i intervallet [12;24].

På bagrund af hyppighedden i søjledigrammet 7.3 så er der blevet lavet et cirkel diagram ud fra nogle intervaller, disse intervaller er lavet med figur 7.3 i mente, hvor intervallerne er opdelt således: [-2;6] [7;11] [12;24] [25;29] [30;33].



Figur 7.4. Cirkel diagram for inddeling for valide vagtplanner i udvalgte intervaller

Figur 7.4 viser procent inddeling af de scores, der er blevet genereret af programmet. De procenter giver en indikation over hvilken sandsynlighed for at programmet genererer en

vagtplan med en score inden i en af intervallerne. Eftersom vores mål er at finde den bedst mulige vagtplan som programmet kan generere, hvor det interval med de højeste scores kun udgør 2%. Dette giver en indikation over hvor sjældent programmet genererer en vagtplan der er af meget høj kvalitet. Selv med denne lille chance for at generere den bedste vagtplan vi har set er der ingenting der viser at det reelt er den bedste vagtplan.

Figur 7.4 viser hvordan de forskellige intervaller inddeles i et lagkagediagram. Ud fra det kan vi se at de fleste vagtplaner der bliver genereret er indenfor intervallet [12;24]. Derfor kan vi sandsynliggøre at programmet generere gode vagtplaner.

7.3 Evaluering

Nu hvor vi har fundet ud af at vores fitness funktion virker og den genetiske algoritme virker som forventet. Derfor vil vi nu se om fitness funktionen og det genetiske algoritme rent faktisk laver gode skemaer i forhold til hvad vi fandt frem til i problem analysen.

	Mandag	Tirsdag	Onsdag
00:00 - 08:00	Roseanna Tutson.9 Cyndy Graziano.5	Mittie Swindoll.1 Roseanna Tutson.9	Sunday Bove.2 Cyndy Graziano.5
08:00 - 16:00	Gregoria Willcutt.6 Gayla Shattuck.8 Sunday Bove.2	Gayla Shattuck.8 Gregoria Willcutt.6 Clement Reichel.7	Gregoria Willcutt.6 Gayla Shattuck.8 Clement Reichel.7
16:00 - 24:00	Linda Burdett.0 Yang Saner.3	Linda Burdett.0 Gayla Shattuck.4	Gayla Shattuck.4 Yang Saner.3

Torsdag	Fredag	Lørdag	Søndag
Mittie Swindoll.1 Cyndy Graziano.5	Yang Saner.3 Mittie Swindoll.1	Yang Saner.3 Cyndy Graziano.5	Gayla Shattuck.4 Cyndy Graziano.5
Sunday Bove.2 Clement Reichel.7 Linda Burdett.0	Gayla Shattuck.8 Gregoria Willcutt.6 Clement Reichel.7	Gregoria Willcutt.6 Gayla Shattuck.8 Sunday Bove.2	Sunday Bove.2 Gregoria Willcutt.6 Clement Reichel.7
Roseanna Tutson.9 Gayla Shattuck.4	Gayla Shattuck.4 Roseanna Tutson.9	Roseanna Tutson.9 Linda Burdett.0	Gayla Shattuck.8 Linda Burdett.0

Tabel 7.5. Vagtplan lagt af vores program ud fra de 10 medarbejdere i bilag 5

På figur 7.5 er vores bedste vagtplan som blev genereret i de 1000 forsøg som vi brugte i sektion 7.2. De individuelle skemaer for hver medarbejder for dette skema ligger i bilag 6. Hver af skemaerne har rimelig stabile vagter. udover det er vagterne cyklisk som vi gerne ville have. Folk der arbejder om natten gør det ikke mere end 2 dage i streg, og de har også en dag til at komme sig over vagten hvis det er den skifter til noget andet. Herfra kan vi konkludere at vores fitness funktion vurdere det bedste skema korrekt til at være et

godt skema. Derfor så kan vi sandsynliggøre at vores fitness funktion er en god indikator for om et skema er godt eller dårligt.

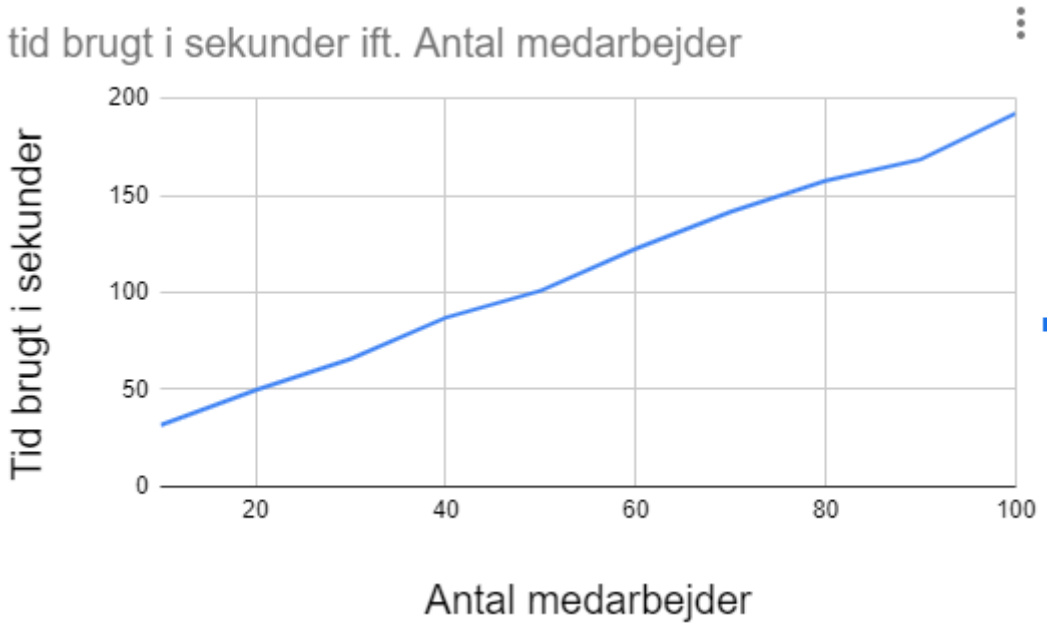
7.4 Tidstagning

Vores program tager rigtig mange input variabler, derfor så er det svært at tage tid på hvor langt tid det tager at kører. Vi har derfor valgt at teste nogle forskellige variabler. Vi prøver derfor at teste variablerne ganget med en faktor, så vi starter med 10 medarbejdere og 2 om natten, 3 om dagen og 2 om aftenen. Testen efter bliver så 20 medarbejdere og 4 om natten, 6 om dagen og 4 om aftenen og så videre. Testen bliver udført på følgende processor `CPU Intel(R) Core(TM) i7-4790K CPU @ 4.00GHz`. Programmet kører kun på en kerne og mængden af hukommelse programmet bruger er ikke relevante i de størrelser af medarbejdere vi tester.

Antal medarbejder	Natarbejdere	Dagarbejdere	Aftenarbejdere	tid brugt i sekunder
<10	2	3	2	31.644
20	4	6	4	49.695
30	6	9	6	65.671
40	8	12	8	86.976
50	10	15	10	100.897
60	12	18	12	122.446
70	14	21	14	141.555
80	16	24	16	157.622
90	18	27	18	168.516
100	20	30	20	192.301

Tabel 7.6. Tabel over hastigheden af programmet til at lægge vagtplaner af forskellige størrelser

På tabel 7.6 er dataen fra testen, vi har kun testet en gang for hver størrelse siden at programmet egentligt bare kører 100.000 generationer og derfor så bliver det genetiske algoritme kørt en masse gange alligevel.



Figur 7.5. Graf over tid i forhold til antal medarbejdere

På figur 7.5 kan man se at vores program overordnet stiger lineært med mængden af medarbejdere inden for den mængde af medarbejdere vi har testet.

8 | Konklusion

Igennem denne rapport har vi gået igennem hvilke krav og udfordringer der er ved at lave vagtplaner til hospitaler. Ved hjælp af det, så skabte vi følgende problem formulering:

Skæve arbejdstider og dårlige vagtplaner skaber mange problemer for medarbejderne på hospitaler. Hvordan kan vi reducere stress hos sygeplejersker og skabe en bedre arbejds- og privatlivsbalance, igennem et automatisk vagtplanlægningsprogram der kan lægge en vagtplan til hospitaler, der overholder de førnævnte krav?

For at forsøge at løse dette problem har vi lavet et program der kan lægge vagtplaner til sygeplejersker på hospitaler. Her valgte vi at bruge en genetisk algoritme til at løse problemet. Så undersøgte vi hvordan man kunne tilrettelægge en algoritme, til at optimere vagtplaner. Herefter så kunne vi bygge et program op omkring dette. Efter programimplementeringen var færdig, så testede vi hvor godt det virkede i sektion 7. Her kom vi frem til at vagtplanlægnings programmet fungerede godt, men vi testede kun den genetiske algoritme for 10 medarbejdere hvor der skulle være sat 2 om natten, 3 om dagen og 2 om aftenen. Formålet med testen var at se om programmet kunne komme frem til en valid vagtplan med de givne medarbejdere, der kom vi frem til at der var 71% chance for programmet kunne generere en valid vagtplan. Dette betyder at der er en stor sandsynlighed for, at når programmet kører, at der bliver genereret en valid vagtplan.
Ud fra det kan vi sige at hvis man vil bruge vores program til at generere en vagtplan er det en god ide at køre den flere gange for at få større chance for at den laver en vagtplan der er valid og opfylder så mange krav som muligt.

Derfor er det svært at sige om programmet ville fungere i den virkelige verden. Dog så kunne vi se ud fra de vagtplaner programmet skabte, at vagtplanerne opfylder vores hårde krav. Vi kunne også se at programmet mellem 10 medarbejdere og 100 medarbejdere, havde et lineært forhold mellem tiden, brugen og antallet af medarbejdere.

Alt dette betyder at vores program opfyldte vores krav, men uden flere tests kan vi ikke vise at programmet virker ved en større liste af medarbejdere. Derfor er det svært at sige om vores program ville virke i den virkelige verden hvor vi har mange flere medarbejdere

og mere varierede mængder af medarbejdere hver vagt. Derudover har vi også lavet simplificeringer i vores program som gør det mindre brugbart i virkeligheden.

9 | Perspektivering

I dette kapitel vil vi perspektivere over flere områder i vores rapport. Vi vil have henblik på hvordan vi kunne have forbedret vores program hvis vi havde haft mere tid, hvordan at vi er blevet klogere på at skabe et produkt, hvordan at den teori vi har brugt er relevant og hvordan at vores produkt ville fungere i den virkelige verden.

9.1 Mulige tekniske forbedringer til programmet

Vi har nu færdigkodet et vagtplanlægningsprogram i C. Vi har i forbindelse med det gjort os nogle overvejelser over hvilke funktioner vi ville have medtaget i vores program hvis vi havde mere tid.

I en vagtplan ville det være relevant at der kan blive tage hensyn for arbejdernes pauser, for at undgå at alle sygeplejersker på en afdelingen tager deres pause samtidigt, en funktion der kunne tilføjes til programmet kunne være at der blev tildelt pauser ind i en blok, men samtidig sørge for at der vil være et minimum af sygeplejersker tilrådighed på afdelingen. Dog som diskuteret før, ved dem i afdelingen selv bedst hvornår de kan holde pause. En måde at gøre det på kunne f.eks. være at give nogle tidspunkter hvor et hvis antal mennesker holder pause og medarbejderne selv fordeler det.

I vores afgrænsning havde vi valgt ikke at tage hensyn til ferie- og helligdage på bagrund af størrelsen af projektet. Hvis der var afsat mere tid til projektet, så kunne der implementeres sådan at programmet ville kunne tage hensyn til ferie- og helligdage. Samtidig havde vi valgt at programmet kun skulle kunne planlægge en uge af gangen, men hvis ferie og helligdage også skulle inddrages, så vil der også være behov for at planlægge mere end en uge af gangen, dette ville også skulle implementeres. Samt så ville dette også opfylde kravet om at vagtplanen skal være forudsigelig, bedre hvis der blev planlagt mere en uge af gangen.

Vi havde også planer om at implementere en måde, hvor at programmet kunne læse

en tidligere vagtplan, og planlægge den næste vagtplan ud fra den, og derved på den måde kunne lave langsigtede vagtplaner. Denne funktionalitet har vi ikke på baggrund af manglende tid.

I det nuværende program er det fastlagt af vagtplanlæggeren, hvor mange der skal arbejde hver vagt af ugens 7 dage. Dette er ikke hensigtsmæssigt hvis arbejdsbyrden ikke er den samme hver dag. Derfor kunne der med fordel implementeres sådan at lægen kan bestemme hvor mange der skal arbejde i hver blok på en uge.

Ifølge vores krav til en software løsning, så skulle programmet kunne tage imod ønsker fra sygeplejerskerne, dette krav bliver overholdt ved, at sygeplejerskerne kan ønske en fridag, samt hvilken vagt de bedst kan lide. Dog så kan dette udvides til at nogle sygeplejersker f.eks ikke vil have en fredag aftenvagt, osv. Altså implementere sådan at sygeplejerskerne kan ønske fri på specifikke blokke og ønske mere end en fridag, f.eks. lørdag og søndag.

Vores program, tager forbehold for at hver medarbejder bliver lagt på en vagtplan individuelt. Hvis man ville have et langt større antal medarbejdere med i en vagtplan. Kunne det være interessant, at implementere en funktion, der gør det muligt at kategorisere flere arbejdere, til en gruppe, som gør det muligt at tildele en hel gruppe, som har fælles karakteristika, til en bestemt vagt.

Det kunne være relevant, især for hospitaler overordnet set, at implementere forskellige former for medarbejdere i programmet. Dette kunne f.eks. være at man havde rangeringer af arbejdere, som f.eks. "Overlæge"og "Sygeplejerskeassistent", som ville have indvirkning på hvor mange af de forskellige medarbejdere der skal bruges og derudover ting som hvem der ville arbejde bedst sammen.

Set fra vagtplanlæggerens brug af programmet, kan de hospitalsspecifikke regler, såsom nedsætning af 11-timers reglen 3, også være relevante at implementere hvis vi havde mere tid.

Som beskrevet under afsnit 7.1 og i afsnit 8 har programmet en chance for at lave en invalid vagtplan og burde derfor blive kørt flere gange for at sikre bedst resultater. Derfor kunne man implementere at programmet laver flere forskellige vagtplaner per kørsel og enten kun viser den bedste eller giver flere forskellige muligheder til vagtplanlæggeren.

Det kunne også laves så vagtplanlæggeren f.eks. giver et antal af vagtplaner som programmet skal lave eller en hvis mængde tid programmet skal køre og derefter giver den bedste vagtplan produceret,

9.2 Autentisk brug af produkt

9.2.1 Kultur på arbejdspladser

Programmet kan potentielt bruges i virkeligheden, men det er relativt begrænset i dets funktioner og der findes langt mere brugervenlige programmer. Dog som et program der genererer vagtplaner, har det nogle funktioner, som eksempelvis MinTid og tjenestetid ikke har. Sammenlignet med nogle af de andre genetiske algoritmer som er blevet lavet er den meget simpel, men ud fra vores research i 3.4 er der ikke sådanne algoritmer i stor brug i den danske sundhedsbranche. (Burke et al., 2004) Vi har forsøgt at tilfredsstille vores brugervenlighedskrav, ved at bruge CSV-filer. Disse filer er brugervenlige, da de er redigerbare, og at vagtplanlæggerne har mulighed for at ændre i disse filer på en nem måde. Programmet kan også godt bruges andre steder i samfundet, da kravene også er gældende. Vagtplanlæggerens input er ikke begrænset til sygeplejersker kun så det er fint nok hvis det bruges andre steder. Det kan dog diskuteres, at programmet ville skulle forbedres, og gøres unikt til forskellige hospitaler, da der er stor forskel kulturmæssigt på hospitaler 3.

9.2.2 Effekt af produktmangler

Programmet tager ikke forbehold for f.eks. helligdage, og ferier. Da dette er tilfældet, vil det også have en stor virkning på hvordan at programmet vil blive brugt, og manglerne vil også kunne mærkes her, igennem utilfredshed fra medarbejderne. Samtidigt planlægger vores program kun vagtplaner til en uge ad gangen. Det er derfor umuligt at lave vagtplaner, som kan tage forbehold for, hvordan at forrige ugers vagtplaner blev lavet. Dette kan også have have en negativ effekt på, hvor modtagelige at medarbejderne er for dette program. Utilfredshed, kunne også opstå i den form, at der med henblik på gennemsnitligt arbejde over en periode på 4 måneder 3, kan være en kultur for arbejderes præference, for at arbejde et bestemt antal timer i en hvis periode. Programmet tager heller ikke forbehold for, om der er en forskellig mængde arbejde på en vilkårlig dag på ugen. Dette kan også være et problem, da der ville være for mange, eller for lidt medarbejdere, på en vilkårlig ugedag.

9.3 Refleksion over program

Med henblik på vores arbejdsproces, er vi blevet klogere på hvordan at man kan bruge en ukendt teori, og applikere dette, til et program, som skal laves. Selve implementeringen,har også vist sig, at være en mindre del af projektet, end forventet. En stor del af projektet, viste sig at omhandle selve ideerne bag programmet og vores programdesign. Vi har fået kendskab til genetiske algoritmer, og hvordan at man potentielt kan løse et kombinatorisk problem med denne teori. Vi har også fået kendskab til hvilken størrelse, der gemmer sig bag, beskrivelsen af et kombinatorisk problem, og hvor kompliceret dette kan blive.

9.4 Brug af genetisk algoritme

Brug af genetiske algoritmer, i forhold til vagtplanlægning, er set før (Burke et al., 2004). Der er en journal (Burke et al., 2004), som gennemgår en række studier, der gennemgår forskellige metoder til vagtplanlæging, som introducerede f.eks. (Cheng et al., 1997), en måde at vagtplanlægge på, hvor at der bliver taget forbehold for både "soft"og "hard"constraints, som vi har draget inspiration fra. Denne metode er også blevet applikeret, til den virkelige verden. I denne journal, er der også blevet beskrevet, hvordan at mange studier, har prøvet at inkorporere genetiske algoritmer ind i vagtplanlægning. Der har dog været delte resultater, hvor at nogle studier har haft en for simpel algoritme, som ikke har været mulig at bruge i virkeligheden, da antallet af variabler ikke har været nok. Vi har draget inspiration fra dette, da det vi ville undersøge, om det kunne være muligt, at skabe en optimal måde at gøre dette på, hvis der blev tillagt flere variabler. Vores program er simpelt, men det kunne have været interessant at inkludere flere variabler, og gøre det mere avanceret, for at se hvordan at dette vil have indflydelse på autentisk brug på et hospital.

Litteratur

Aarhus Universitetshospital, Afdeling M, Risskov, **2014**. Aarhus Universitetshospital, Afdeling M, Risskov. *Evaluering af MinTid - LMU, Afdeling M.* `https://www.rm.dk/api/NewESDHBlock/DownloadFile?agendaPath=%5C%5CRMAPPS0221.onerm.dk%5CCMS01-EXT%5CESDH%20Data%5CRM_Internet%5Cdagsordener%5CHMU_Psykiatri_og_Soc%202015%5C08-12-2014%5CAaben_dagsorden&appendixId=87600`, 2014. Besøgt 23-10-2020.

Arbejdstilsynet, **1997**. Arbejdstilsynet. *Ugentligt fridøgn.* `https://at.dk/media/2837/at-meddelelse-ugentligt-fridoegn-5-01-2.pdf`, 1997. besøgt 29-10-2020.

Australian government, **2020**. Australian government. *Managing crew fatigue.* `https://www.amsa.gov.au/vessels-operators/domestic-commercial-vessels/managing-crew-fatigue`, 2020. Besøgt 16-10-2020.

B. Cooper og H. Kingston, **1995**. Time B. Cooper og Jeffrey H. Kingston. *The complexity of timetable construction problems.* Practice and Theory of Automated Timetabling, 1153, 1995.

Baumler et al., **2020**. Raphael Baumler, Bikram Bhatia og Momoko Kitada. *Ship first: Seafarers' adjustment of records on work and rest hours.* Marine Policy, 104186, 2020. doi: 10.1016/j.marpol.2020.104186.

Bille, **2017**. Maya Bille. *Arbejdet som sygeplejerske udfordrer privatlivet.* https://dsr.dk/politik-og-nyheder/nyhed/arbejdet-som-sygeplejerske-udfordrer-privatlivet, 2017. Besøgt 22-10-2020.

Brooks et al., **2020**. Candie Brooks, Leon c. Coody og Sam Abraham. *Night Shift Work and Its Health Effects on Nurses.* The Health Care Manager, 39(9), 122–127, 2020. doi: 10.1097/HCM.0000000000000297. URL

https://journals.lww.com/healthcaremanagerjournal/Fulltext/2020/07000/Night_Shift_Work_and_Its_Health_Effects_on_Nurses.3.aspx.

Burke et al., 11 2004. Edmund Burke, Patrick De Causmaecker, Greet Vanden Berghe og Hendrik Van Landeghem. *The State of the Art of Nurse Rostering.* J. Scheduling, 7, 441–499, 2004. doi: 10.1023/B:JOSH.0000046076.75950.0b.

Cheng et al., 1997. B. M. W. Cheng, J. H. M. Lee og J. C. K. Wu. *A nurse rostering system using constraint programming and redundant modeling.* IEEE Transactions on Information Technology in Biomedicine, 1(1), 44–54, 1997. doi: 10.1109/4233.594027.

Computer Hope, **2020**. Computer Hope. *How to create a CSV file.* https://www.computerhope.com/issues/ch001356.htm, 2020. Besøgt 15-12-2020.

Dansk sygeplejeråd, **2016a**. Dansk sygeplejeråd. *Arbejdstid, når du er ansat i regionen.* https://dsr.dk/loen-og-arbejdsvilkaar/overenskomster-og-aftaler/regionale-overenskomster-og-aftaler/arbejdstid-naar, 2016a. Besøgt 22-10-2020.

Dansk sygeplejeråd, **2016b**. Dansk sygeplejeråd. *NOTAT Sygeplejerskers arbejdsliv-privatlivsbalance*, Dansk sygeplejeråd, https://dsr.dk/sites/default/files/50/notat_sygeplejerskers_arbejdsliv-privatlivsbalance_sath_2015_opdat.pdf, 2016b. Figur 4. Arbejdsliv-privatlivsbalancen (score) fordelt på arbejdstid, 2015.

Dansk sygeplejeråd, **2012**. Dansk sygeplejeråd. *Forebyg brystkræft når du arbejder om natten.* https://dsr.dk/loen-og-arbejdsvilkaar/arbejdsmiljoe/fysisk-arbejdsmiljoe/natarbejde/forebyg-brystkraeft-naar-du, 2012. Besøgt 13-10-2020.

Danske Regioner, **2016**. Danske Regioner. *Optimal anvendelse af personaleressourcer på hospitaler*, Danske Regioner, 2016. URL https://www.regioner.dk/media/3257/personaleressourcer-rapport.pdf. Om vagtplaner i afsnit 6.

Danske Regioner et al., 2018. Danske Regioner, Region Nordjylland, Region Syddanmark, Region Sjælland, Region Midtjylland, Region Hovedstaden og Moderniseringsstyrelsen. *nspirationskatalog om vagtplanlægning og personaleanvendelse.* https://modst.dk/media/30159/inspirationskatalog-vedr-vagtplanlaegning-og-personaleanvendelse.pdf, 2018. Case nummer 5, 8, 9 og 13 er dem der bliver refereret til.

Danskerhverv, **2019**. Danskerhverv. *Hvad tæller med i 48-timers reglen?* https://www.danskerhverv.dk/presse-og-nyheder/nyheder/hvad-taller-med-i-48-timers-reglen/, 2019. Besøgt 15-10-2020.

Det Europæiske Arbejdsagentur, **2020**. Det Europæiske Arbejdsagentur. *Psykosociale risici og stress på arbejdspladsen.* https://osha.europa.eu/da/themes/psychosocial-risks-and-stress, 2020.

Diaz-gomez og Hougen, **2007**. Pedro Diaz-gomez og Dean Hougen. *Initial Population for Genetic Algorithms: A Metric Approach.* https://www.researchgate.net/publication/220862320_Initial_Population_for_Genetic_Algorithms_A_Metric_Approach, 2007. 30-11-2020.

FOA, **2020**. FOA. *Helligdage, FO-dage og søgnehelligdage.* https://www.foa.dk/raad-regler/i-job/arbejdstid/helligdage-fo-dage-soegnehelligdage, 2020. 29-10-2020.

Forbundet af offentlige ansatte, **2020**. Forbundet af offentlige ansatte. *Natarbejde.* https://www.foa.dk/raad-regler/i-job/arbejdstid/natarbejde, 2020. besøgt 28-10-2020.

Gu et al., **2015**. Fangyi Gu, Jiali Han, Francine Laden, An Pan, Neil E. Caporaso, Meir J. Stampfer, Ichiro Kawachi, Kathryn M. Rexrode, Walter C. Willett, Susan E. Hankinson, Frank E. Speizer og Eva S. Schernhammer. *Total and Cause-Specific Mortality of U.S. Nurses Working Rotating Night Shifts.* American Journal of Preventive Medicine, 48(3), 241 – 252, 2015. ISSN 0749-3797. doi: https://doi.org/10.1016/j.amepre.2014.10.018.

HK, **2020**. HK. *NATARBEJDE, SKIFTEHOLDSARBEJDE OG RÅDIGHEDSVAGT.* https://www.hk.dk/raadogstoette/arbejdstid/natarbejde, 2020. 02-11-2020.

Holmboe, **2018**. Rikke Holmboe. *LIVET MED TREHOLDSSKIFT – DET SOCIALE LIV?* https://fadlforlag.dk/2018/11/02/livet-med-treholdsskift-det-sociale-liv/, 2018. Besøgt 28-10-2020.

Indexed.dk, **2020**. Indexed.dk. *Sådan udformes en kravspecifikation til IT-projekt.* https://indexed.dk/blog/kravspecifikation/, 2020. Besøgt 04-12-2020.

A Jan, M Yamamoto og A Ohuchi, 2000. A Jan, M Yamamoto og A Ohuchi. Evolutionary algorithms for nurse scheduling problem. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, volume 1, pages 196–203 vol.1. IEEE, 2000. ISBN 9780780363755.

Juulsager, **2020**. Jens Jacob Juulsager. *Konflikt mellem sygeplejersker og Rigshospitalet spidser til*. https://www.tv2lorry.dk/kobenhavn/konflikt-mellem-sygeplejersker-og-rigshospitalet-spidser-til, 2020. Besøgt 22-10-2020.

Kjeldsen, Susanne Bloch, **2005**. Kjeldsen, Susanne Bloch. *Vigtige 29 minutter*. https://dsr.dk/sygeplejersken/arkiv/sy-nr-2005-4/vigtige-29-minutter, 2005. 29-10-2020.

Klar, **2020**. Peter Klar. *Fremtidens sundhedssektor: Behovet sætter holdet*. https://eg.dk/nyheder/2020/april/fremtidens-sundhedssektor-behovet-satter-holdet/, 2020. Besøgt 28-10-2020.

Kogler et al., **2015**. Lydia Kogler, Veronika I Müller, Amy Chang, Simon B Eickhoff, Peter T Fox, Ruben C Gur og Birgit Derntl. *Psychosocial versus physiological stress - Meta-analyses on deactivations and activations of the neural correlates of stress reactions*. PubMed, 119, 235–51, 2015. doi: 10.1016/j.neuroimage.2015.06.059. URL https://www.rm.dk/socialomraadet/hmu-for-psykiatri-og-social/dagsordener-og-referater/?SelectedYear=2014&SelectedMeetingId=484#EvalueringafMinTid(20min). Besøgt 22-10-2020.

Kohl og Karish E., **2004**. Niklas Kohl og Stefan Karish E. *Airline Crew Rostering: Problem Types, Modeling, and Optimization*. Annals of Operations Research, 2004.

Lozano et al., **2016**. Mariona Lozano, Dana Hamplová og Céline Le Bourdais. *Non-standard work schedules, gender, and parental stress*. Demographic Research, 34 (9), 259–284, 2016. doi: 10.4054/DemRes.2016.34.9. URL https://www.demographic-research.org/volumes/vol34/9/.

Mallawaarachchi, **2017**. Vijini Mallawaarachchi. *Introduction to Genetic Algorithms — Including Example Code*. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3, 2017. Besøgt 27-11-2020.

Mitchell Melanie, **2020**. Mitchell Melanie. *Genetic algortithms: An overview*. https://onlinelibrary.wiley.com/doi/pdf/10.1002/cplx.6130010108, 2020. Besøgt 14-12-2019.

Nicolajsen, **2017**. Lisbeth Nicolajsen. *Projekt Bedre Vagt- og Arbejdstidsplanlægning for læger*. https://www.rm.dk/siteassets/om-os/organisation/koncern-hr/koncern-hr-udvikling/konsulentydelser/organisationsudvikling/bedre-vagt-og-arbejdstidsplanlagning-2017.pdf, 2017.

Overbjerg, **2019**. Lotte Overbjerg. *Vagtplaner kræver avanceret matematik.* https://www.welfaretech.dk/nyheder/2019/december/vagtplaner-kraever-avanceret-matematik, 2019. Besøgt 16-12-2019.

Overbjerg, **2020**. Lotte Overbjerg. *Det skal være nemmere at bruge matematiske modeller til vagtplanlægning.* Detskalv\T1\aerenemmereatbrugematematiskemodellertilvagtplanl\T1\aegning, 2020. Besøgt 28-10-2020.

PDC, **2020a**. PDC. *Vagtplanlægning i sundhedsvæsenet.* https://www.pdc.com/da/solution/vagtplanlaegning-sundhed/, 2020a. Besøgt 28-10-2020.

PDC, **2020b**. PDC. *VagtPlan strømliner hele vagtplanprocessen.* https://www.pdc.com/da/solution/personalestyring-detailhandel/personalestyring-vagtplan/, 2020b. Besøgt 20-10-2020.

PlanDay, **2020**. PlanDay. *Personalehåndtering og vagtplanlægning i sundhedsvæsenet.* https://www.planday.com/da/industri/sundhedspleje/, 2020.

programiz, **2020**. programiz. *Flowchart In Programming.* https://www.programiz.com/article/flowchart-programming, 2020. Besøgt 15-12-2020.

Puttonen et al., **2010**. Sampsa Puttonen, Mikko Härmä og Christer Hublin. *Shift work and cardiovascular disease — pathways from circadian stress to morbidity.* Scandinavian Journal of Work, Environment & Health, 36(2), 96–108, 2010. ISSN 03553140, 1795990X. URL http://www.jstor.org/stable/40967836.

Rigsrevisionen, **2015**. Rigsrevisionen. *Beretning til Statsrevisorerne om hospitalernes brug af personaleresurser*, Rigsrevisionen, https://www.rigsrevisionen.dk/media/2033513/hospitalernes-brug-af-personaleresurser.pdf, 2015.

Martín Safe, Jessica Carballido, Ignacio Ponzoni og Nélida Brignole, *2004*. Martín Safe, Jessica Carballido, Ignacio Ponzoni og Nélida Brignole. On Stopping Criteria for Genetic Algorithms. In Ana L. C. Bazzan og Sofiane Labidi, editor, *Advances in Artificial Intelligence – SBIA 2004*, pages 405–413, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-28645-5.

Savic et al., **2019**. Michael Savic, Rowan P. Ogeil, Megan J. Sechtig, Peta Lee-Tobin, Nyssa Ferguson og Dan I. Lubman. *How Do Nurses Cope with Shift Work? A*

Qualitative Analysis of Open-Ended Responses from a Survey of Nurses. International Journal of Environmental Research and Public Health, 2019. doi: https://doi.org/10.3390/ijerph16203821. URL `https://www.mdpi.com/1660-4601/16/20/3821/html`.

Shyalika, **2019**. Chathurangi Shyalika. *Genetic Algorithms -Selection*. `https://medium.com/datadriveninvestor/genetic-algorithms-selection-5634cfc45d78`, 2019. Besøgt 14-12-2020.

Silkeborgdata, **2020a**. Silkeborgdata. *MinTid*. `https://www.silkeborgdata.dk/produkt/mintid`, 2020a. Besøgt 22-10-2020.

Silkeborgdata, **2020b**. Silkeborgdata. *Tjenestetid*. `https://www.silkeborgdata.dk/produkt/tjenestetid`, 2020b. Besøgt 22-10-2020.

Simona Nicoară, **2009**. Elena Simona Nicoară. *http://bmif.unde.ro/docs/20091/12NICOARA$_S$IMONA* 2009. Besøgt 16-12-2020.

Smith, **1983**. R. Douglas Smith. *The Structure of Divide-and-Conquer Algorithms*. `https://www.researchgate.net/publication/272506022_The_Structure_of_Divide-and-Conquer_Algorithms`, 1983. 30-11-2020.

Stimpfel og Aiken, **2013**. Amy Witkoski Stimpfel og Linda H Aiken. *Hospital staff nurses' shift length associated with safety and quality of care*. Journal of nursing quality, 28(2), 122 – 129, 2013.

sygeplejeråd, **2016**. Dansk sygeplejeråd. *Sygeplejerskers arbejdsmiljø - SATH-databasen*. `https://dsr.dk/loen-og-arbejdsvilkaar/arbejdsmiljoe/sygeplejerskers-arbejdsmiljoe-trivsel-og-helbred-sath-1/`, 2016. Data bag undersøgelse.

TimePlan, **2020**. TimePlan. *Personaleadministration og vagtplan for institutioner og sundhedssektoren*. `https://www.timeplan-software.com/da/info/kunder-dk/personaleadministr og-vagtplaner-for-institutioner/`, 2020. Besøgt 28-10-2020.

Tsuruoka, **2001**. H. Kawanaka; K. Yamamoto; T. Yoshikawa; T. Shinogi; S. Tsuruoka. *Genetic algorithm with the constraints for nurse scheduling problem*. `https://ieeexplore.ieee.org/document/934317`, 2001. 27-11-2020.

Wagstaff og Lie, **5 2011**. Anthony Sverre Wagstaff og Jenny-Anne S. Lie. *Shift and night work and long working hours - a systematic review of safety implications*. Scandinavian journal of work, environment & health, 37(3), 173–85, 2011. URL `https:`

//search.proquest.com/docview/866447151?accountid=8144. Copyright - Copyright Scandinavian Journal of Work, Environment & Health May 2011; Last updated - 2014-04-03.

Womersly, **2008**. Bob Womersly. *Local and Global Optimization Formulation, Methods and Applications*. https://web.maths.unsw.edu.au/~rsw/lgopt.pdf, 2008. Besøgt d. 4-12-2020.

Yee Leung et al., **1997**. Yee Leung, Yong Gao og Zong-Ben Xu. *Degree of population diversity - a perspective on premature convergence in genetic algorithms and its Markov chain analysis*. IEEE Transactions on Neural Networks, 8(5), 1165–1176, 1997. 10.1109/72.623217.

Bilag

Bilag 1: Programmets kildekode

Al data forbundet med programmet er indeholdt heri.
Se elektronisk bilag.

Bilag 2: Medarbejder liste A

Navn	Ønsket vagt	Ønsket fridag	UUID
Tanika Burstein	lørdag	aften	1
Quintin Bratt	mandag	dag	2
Rita Hursh	onsdag	aften	3
Celestine Mcclean	mandag	nat	4
Clarisa Satchell	tirsdag	dag	5
Donette Worthey	torsdag	aften	6
Shaniqua Canez	torsdag	dag	7
Nicol Pasko	fredag	dag	8
Donette Worthey	mandag	nat	9
Sallie Richart	tirsdag	nat	18
Roseanna Tutson	tirsdag	dag	19

Bilag 3: Eksempel på eksporteret vagtplan

mandag	nat	Shannon Casady.0	Donette Worthey.9	$	
mandag	dag	Tanika Burstein.1	Quintin Bratt.2	Celestine Mcclean.4	$
mandag	aften	Donette Worthey.6	Donette Worthey.9	$	
tirsdag	nat	Tanika Burstein.1	Nicol Pasko.8	$	
tirsdag	dag	Tanika Burstein.1	Quintin Bratt.2	Nicol Pasko.8	$
tirsdag	aften	Celestine Mcclean.4	Donette Worthey.9	$	
onsdag	nat	Shannon Casady.0	Quintin Bratt.2	$	
onsdag	dag	Tanika Burstein.1	Celestine Mcclean.4	Clarisa Satchell.5	$
onsdag	aften	Clarisa Satchell.5	Donette Worthey.9	$	
torsdag	nat	Shannon Casady.0	Shaniqua Canez.7	$	
torsdag	dag	Tanika Burstein.1	Quintin Bratt.2	Nicol Pasko.8	$
torsdag	aften	Celestine Mcclean.4	Donette Worthey.9	$	
fredag	nat	Shannon Casady.0	Shaniqua Canez.7	$	
fredag	dag	Tanika Burstein.1	Celestine Mcclean.4	Nicol Pasko.8	$
fredag	aften	Rita Hursh.3	Clarisa Satchell.5	$	
lørdag	nat	Shannon Casady.0	Celestine Mcclean.4	$	
lørdag	dag	Rita Hursh.3	Quintin Bratt.2	Shaniqua Canez.7	$
lørdag	aften	Celestine Mcclean.4	Tanika Burstein.1	$	
søndag	nat	Shannon Casady.0	Celestine Mcclean.4	$	
søndag	dag	Tanika Burstein.1	Clarisa Satchell.5	Shaniqua Canez.7	$
søndag	aften	Quintin Bratt.2	Rita Hursh.3	$	

Bilag 4: Individuel brugervenlig vagtplan

```
+-------------+------+-------+------+-------+------+------+------+
|             |Mandag|Tirsdag|Onsdag|Torsdag|Fredag|Lørdag|Søndag|
+-------------+------+-------+------+-------+------+------+------+
|00:00 - 08:00|      |       |      |       |      |      |      |
|08:00 - 16:00|######|#######|######|       |      |      |######|
|16:00 - 24:00|      |       |      |#######|      |      |      |
+-------------+------+-------+------+-------+------+------+------+
```

Bilag 5: Medarbejder liste B

Linda Burdett	fredag	nat	0
Mittie Swindoll	lørdag	nat	1
Sunday Bove	tirsdag	dag	2
Yang Saner	torsdag	nat	3
Gayla Shattuck	lørdag	nat	4
Cyndy Graziano	tirsdag	nat	5
Gregoria Willcutt	torsdag	dag	6
Clement Reichel	mandag	dag	7
Gayla Shattuck	torsdag	dag	8
Roseanna Tutson	onsdag	nat	9

Bilag 6: Bedste vagtplan lagt i løbet af de 1000 forsøg

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |       |       |       |       |       |       |
|08:00 - 16:00|       |#######|#######|#######|#######|       |#######|
|16:00 - 24:00|       |       |       |       |       |       |       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 1. Vagtplan for Clement Reicehel.7

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|#######|       |#######|#######|       |#######|#######|
|08:00 - 16:00|       |       |       |       |       |       |       |
|16:00 - 24:00|       |       |       |       |       |       |       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 2. Vagtplan for Cyndy Graziano.5

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |       |       |       |       |       |#######|
|08:00 - 16:00|       |       |       |       |       |       |       |
|16:00 - 24:00|       |#######|#######|#######|#######|       |       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 3. Vagtplan for Gayla Shattuck.4

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |       |       |       |       |       |       |
|08:00 - 16:00|#######|#######|#######|       |#######|#######|       |
|16:00 - 24:00|       |       |       |       |       |       |#######|
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 4. Vagtplan for Gayla Shattuck.8

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |       |       |       |       |       |       |
|08:00 - 16:00|#######|#######|#######|       |#######|#######|#######|
|16:00 - 24:00|       |       |       |       |       |       |       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 5. Vagtplan for Gregoria Willcut.6

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |       |       |       |       |       |       |
|08:00 - 16:00|       |       |       |#######|       |       |       |
|16:00 - 24:00|#######|#######|       |       |       |#######|#######|
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 6. Vagtplan for Linda Burdett.0

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |#######|       |#######|#######|       |       |
|08:00 - 16:00|       |       |       |       |       |       |       |
|16:00 - 24:00|       |       |       |       |       |       |       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 7. Vagtplan for Mitie Swindoll.1

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|#######|#######|       |       |       |       |       |
|08:00 - 16:00|       |       |       |       |       |       |       |
|16:00 - 24:00|       |       |       |#######|#######|#######|       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 8. Vagtplan for Roseanna Tutson.9

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |       |#######|       |       |       |       |
|08:00 - 16:00|#######|       |       |#######|       |#######|#######|
|16:00 - 24:00|       |       |       |       |       |       |       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 9. Vagtplan for Sunday Bove.2

```
+-------------+-------+-------+-------+-------+-------+-------+-------+
|             |Mandag |Tirsdag|Onsdag |Torsdag|Fredag |Lørdag |Søndag |
+-------------+-------+-------+-------+-------+-------+-------+-------+
|00:00 - 08:00|       |       |       |       |#######|#######|       |
|08:00 - 16:00|       |       |       |       |       |       |       |
|16:00 - 24:00|#######|       |#######|       |       |       |       |
+-------------+-------+-------+-------+-------+-------+-------+-------+
```

Snippet 10. Vagtplan for Yang Saner.3