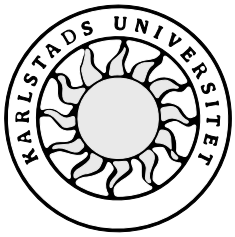


Datavetenskap

Jan Jönsson och Kristina Nilsson

Automatisk schemakonstruktion

Examensarbete, C-nivå

2000:33

Automatisk schemakonstruktion

Jan Jönsson och Kristina Nilsson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en kandidatexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Jan Jönsson

Kristina Nilsson

Godkänd, 2000-08-21

Handledare: Stefan Lindskog

Examinator: Stefan Lindskog

Sammanfattning

EuroTime AB [9] är ledande inom sitt område som avser tidmätning och registrering av tid. 1989 startade de med att utveckla ett system som registrerar tid. Systemet kallas TID2000 [6]. TID2000 är en applikation för Windows och det är en klassisk klient/serverapplikation vilken lagrar data i en relationsdatabas. TID2000 är en flexibel applikation och användaren kan ställa in till hur de är vana att arbeta med tid. Vidare finns det planer på att stödja Unix som plattform för databasmotorn.

Vår uppgift var att hjälpa schemaläggaren med en automatisk generering av ett schema. Det enda en schemaläggare ska behöva göra är skapa ett resursbehov och ett urval av personer att fylla detta resursbehov med. Schemaläggaren ska sen endast behöva trycka på knappen för att få programmet att generera ett schema. Vi kom efter en kortare diskussion överens om att hålla algoritmen enkel, lätt att förstå och lätt utbyggbar för extra funktionalitet. Anledningen till dessa intentioner var att EuroTime AB och vi ville försöka göra en implementation av algoritmen i C++. Vi var också tvungna att ta reda på de lagar som reglerar arbetsmarknaden. Dessa lagar blir regler som inte får brytas i vår algoritm. Vi skapade också ett antal möjliga regler som algoritmen kan bryta emot om schemaläggaren tillåter det. Dessa regler är av typen personliga önskningar, helger att arbeta och timmar kvar att arbeta. Typ och mängd av dessa regler kan variera från företag till företag. Schemaläggaren måste också ta i beaktande om han/hon vill ha poäng för sämre arbetstider såsom nätter och helger. Schemaläggaren har också möjligheten att ange prioritet på regler som inte är uttryckta i lag. Detta kan komma väl till pass när algoritmen kört fast och inte kan gå vidare. Vi har tagit allt detta under betänkande och tror att vårt förslag till lösning har en god möjlighet att bli en kommersiell produkt med lite ytterligare utvecklingsarbete och förfining. Vår intention om att implementera ett litet exempel blev begränsat till endast en kontroll (timmar kvar att arbeta) på grund av tidsbrist.

Automatic time schedule construction

Abstract

EuroTime AB is a leading company, which works with measurement of time and registration of time. In 1989 they started to develop a system that registers time. This system is called TID2000. TID2000 is an application for Windows. It is a classic client/server application, which stores data in a relational database. TID2000 is a flexible application and the users can adjust it to how they are use to work with time. There are some plans though to support Unix as a platform for the database engine.

Our task was to help the scheduler to automatize the generation of a schedule. The only thing a scheduler should have to do is to make an resource requirement and a selection of people to fill it with. Then he/she would just have to push the button and the program should generate a schedule. We agreed after some discussion to try to keep the algorithm simple, easy to understand and easy to add extra functionality. One reason for this was that EuroTime AB and we wanted to try to make an implementation of the algorithm in C++. One other thing we had to look into was the laws that regulate the labor market. These laws become rules that can’t be broken in the algorithm. We also made some possible rules that the algorithm could break if the scheduler allowed it. These rules are of the kind of personal wishes, weekends to work and hours left to work. The set of these rules can vary from company to company. The scheduler also has to consider whether he wants points for bad working hours such as nights and weekends. He/she also has the possibility to state the priority of rules not expressed by law. This can come in handy when the algorithm is stuck and can’t go forward. We have taken all this under consideration and we think that our proposal to the solution is good and possible to implement as a commercial product with a little more development and refinement. Our intention to implement a small example was limited to just one check (hours left to work) due to lack of time.

Innehållsförteckning

Figurförteckning

1 Introduktion

Det här dokumentet är ett resultat av ett examensarbete i ämnet datavetenskap. Examensarbetet omfattar 10 poäng och är en obligatorisk kurs på C-nivå på dataingenjörsprogrammet vid Karlstads universitet (KaU). Examensarbetet utfördes under våren 2000 vid EuroTime AB i Säffle.

1.1 EuroTime AB

EuroTime AB startades 1988 med utveckling av diverse elektronik. Under 1989 påbörjades utvecklingen av tidregistreringssystemet TID2000 för DOS. Man började först tillsammans med ICA-rörelsen därefter med landsting och industrier. Från våren 1990 har företaget marknadsfört konceptet TID2000.

EuroTimes kunder finns inom bl.a. vård och omsorg (d.v.s. kommun och landsting), handel, tjänsteföretag, bank, försvar, statliga verk, organisationer samt industri. Kunderna är var och en mycket kända inom sitt område. Förutom de egna programmen TID2000 för DOS och TID2000 för WINDOWS så har EuroTime som enda partner medverkat i utvecklingen av IBM:s personalprogram Palett. EuroTime har dessutom utvecklat ett Windowsbaserat schemabemanningssystem till Securitaskoncernen som finns i drift inom Sverige och Norge.

EuroTime utvecklar i C, C++, SQLWindows, Visual Basic och ASP (Active Server Page) mot relationsdatabaserna SQLBase och Oracle. När det gäller utrustning för ”tidsfångst” så har EuroTime varit det ledande företaget i utvecklingen av olika sätt för att fånga tid. Tidregistreringsterminaler/flexklockor finns i olika varianter och det har varit unikt att kunna ansluta dessa direkt till kundens datanätverk, både till Ethernet och till Token ring [5]. EuroTime har också en klockserverlösning som även den är unik eftersom den både kan kommunicera med såväl EuroTimes som konkurrenters program. Denna lösning ger kunden stor valfrihet. Genom nätanslutna flexklockor tillsammans med klockserver så kan stora globala klocknät skapas genom användandet av befintliga datanät. Talsvar är ytterligare en lösning där telefonen används för att ge eller att få besked när det gäller tid och PA-frågor. En WEB-lösning är det senaste som lanserats, detta för att ge kunden maximal valfrihet och kombinationsmöjlighet.

1.2 Bakgrund och mål

Idag är EuroTimes stora applikation TID2000 för Windows. Detta är ett Windowsprogram som lagrar data i en relationsdatabas, en klassisk klient/server-applikation. Denna applikation är flexibel och kunderna kan anpassa denna till hur de arbetar med tid. TID2000 har en väl utbyggd schemafunktion där användaren planerar sin personal utifrån de resursbehov man har. Det finns stöd i programmet att göra jämförelser med detta resursbehov för att snabbt och smidigt kunna avgöra om man är underbemannad eller har schemalagt mer personal än vad som behövs. Vad som kunderna saknat är att systemet på ett intelligent sätt själv kan skapa ett schemaförslag åt användaren, detta kallar EuroTime för automatisk schemakonstruktion.

Avsikten med examensarbetet är att ta fram en algoritm för schemakonstruktion. Med hjälp av denna algoritm ska sedan en prototyp tas fram.

2 Systembeskrivning TID2000

Detta kapitel beskriver systemet TID2000, dess innehåll och hur det fungerar. Sedan ges en mer detaljerad beskrivning av de viktigaste delarna i systemet som berör schemakonstruktionen.

2.1 TID2000

Systemet TID2000 består bland annat av en relationsdatabas som innefattar person-uppgifter, arbetsscheman, fackliga avtal och frånvaro. TID2000 är uppdelat i fyra delar:

Dagligt arbete: Här sköts det löpande dagliga arbetet såsom personalhantering, scheman, bemanning, bokningar, frånvarohantering, rättning av avvikelser, framtagning av rapporter.

Tabellunderhåll: Här läggs all grundinformation in som företagsnamn, kostnadsställen, personkategorier, försäkringskassor.

Behörighet: Här hanteras de som skall använda TID2000 och vad de får lov att göra. I behörighet finns även funktioner för att administrera databasen och göra program-uppdateringar.

Klockslagskalkylatorn: Detta är en enkel miniräknare som beräknar tider. Den kan räkna med timmar och minuter.

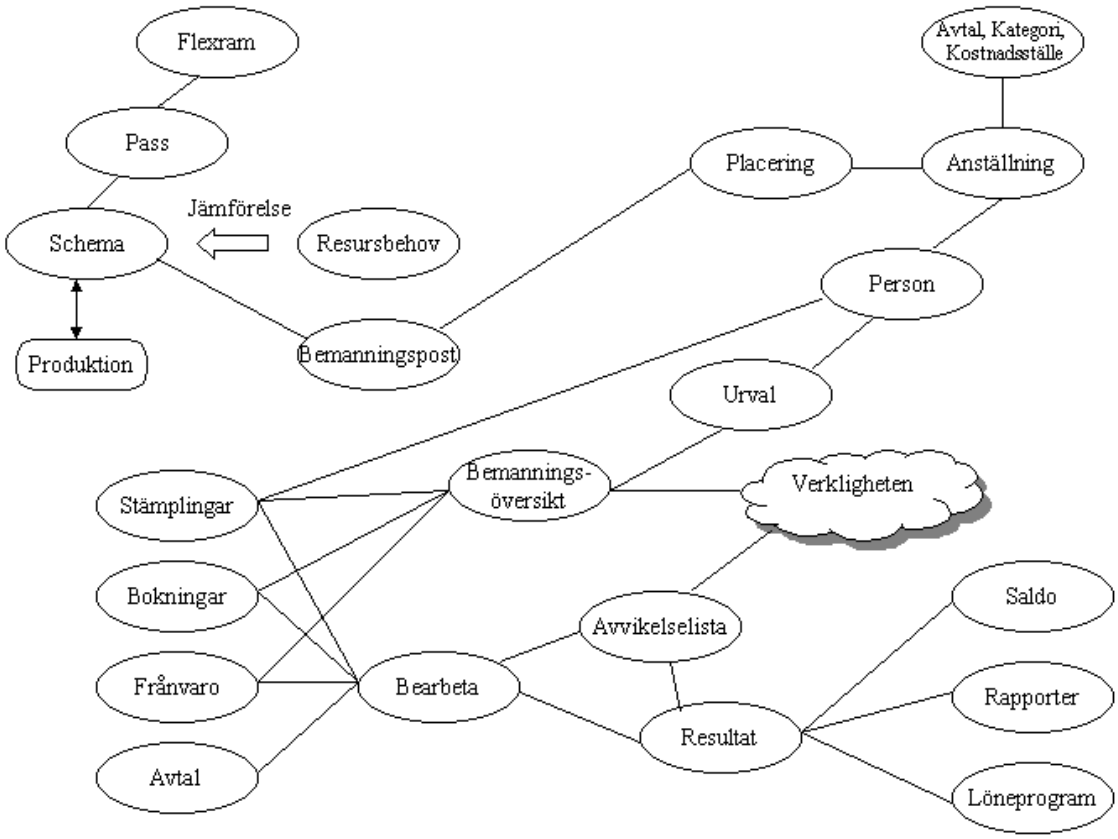
2.2 Klient/Server och SQL

TID2000 är gjort med en teknik som kallas klient/server och arbetar mot en SQL-databas. All tidsinformation lagras i en central databas och med hjälp av frågor plockas informationen fram från databasen. Frågespråket som används är SQL (Structured Query Language) [2].

Det fungerar i princip så att klienten, en arbetsstation, ställer en fråga till databasen. Databasen tar hand om frågan och sammanställer svaret i en tabell, och skickar tillbaka svaret till klienten. Detta sätt att hantera stora datamängder innebär att allt det tunga arbetet görs i databasen (servern) och din dator (klienten) enbart ställer frågor och presenterar svaren på ett begripligt sätt.

2.3 Systemöversikt

I figur 2.1 visas en bild över hur den del av systemet som berör scheman hänger ihop.



Figur 2.1 Systemöversikt

En person har bland annat personnummer, adress och telefonnummer. Anställning är knuten till en person som visar sysselsättningsgrad, avtal, kategori och kostnadsställe. En person kan ha flera anställningar. Varje person kan arbeta på olika avdelningar och tillhöra olika kostnadsställen, för att möjliggöra detta finns placeringar. Flera personer utgör en grupp som kallas ett urval. En person kan tillhöra många urval.

Ett schema består av arbetspass med flexramar. Schemat bemannas med placeringar vilket ger en bemanningspost. Schemat kan jämföras med resursbehovet. Ett schema sätts i produktion, men kan även tas ur produktion för att göra ändringar.

I bemanningsöversikten ses urvalets aktuella bemanning. Här syns också stämplingar, bokningar och frånvaro. Bemanningsöversikten kan även ses grafiskt. I bokningar görs byten av

arbetspass och redovisning av övertid. Under frånvaro registreras planerad och oplanerad frånvaro.

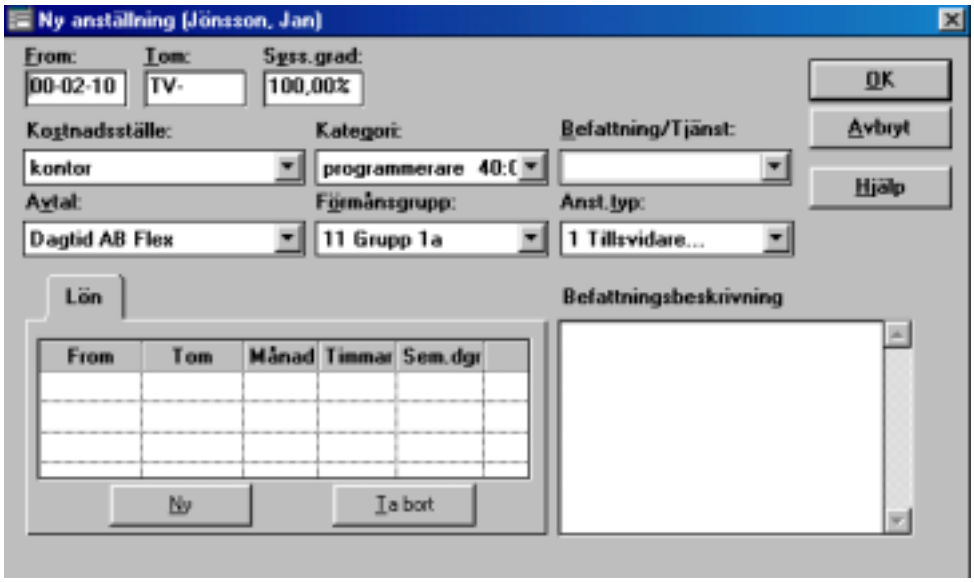
Avvikelselistan anropar bearbeta som kommer fram med ett resultat och som redovisas i avvikelselistan. Där ses felstämplingar och korrigeringar görs. Data från stämplingar, bokningar, frånvaro och avvikelser bearbetas med hänsyn till avtal. Detta redovisas som saldo, rapporter eller skickas direkt till ett löneprogram.

2.4 Detaljerad beskrivning

Vi kommer nedan att behandla en liten del av systemet TID2000. Det som berör schemakonstruktionen är person, resursbehov och schema.

2.4.1 Person

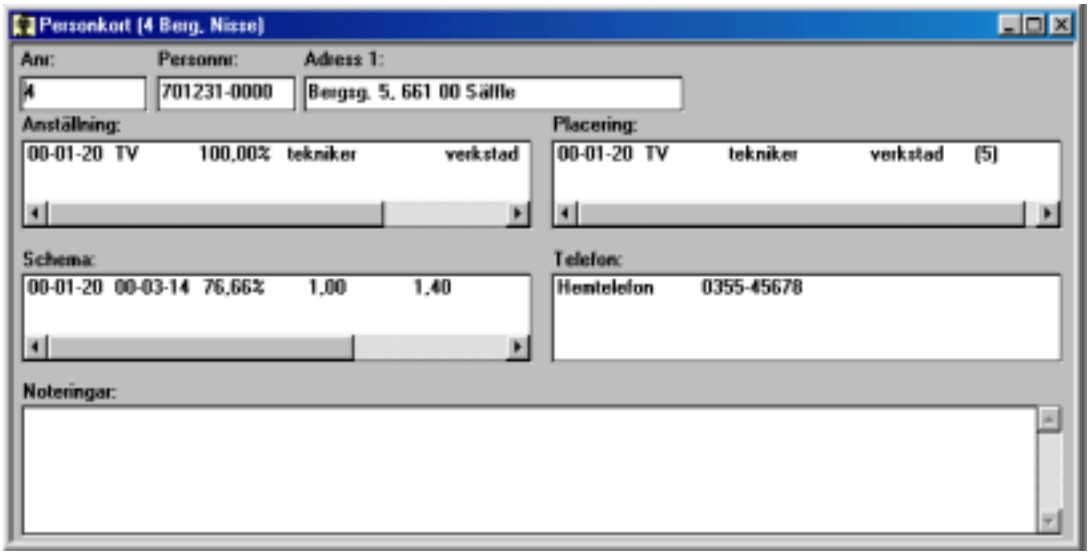
Vid ny anställning fylls tidsperioden i samt vilken sysselsättningsgrad personen kommer att ha, se figur 2.2. Dessutom fylls kostnadsställe, kategori, avtal, förmånsgrupp och anställningstyp i. Fälten befattning/tjänst, lön samt befattningsbeskrivning behöver ej fyllas i. Befattning/tjänst kan vara av typen personalansvarig för sin kategori.



Figur 2.2 Exempel på hur man lägger in en nyanställd person

Med hjälp av personkortet, se figur 2.3, fås en snabb överblick av den anställde. Förutom personnummer, adress och telefonnummer visas också vilken typ av anställning och placering

personen har och vilket schema personen går på. Det finns även ett fält där speciella noteringar kan fyllas i.



Figur 2.3 Exempel på hur ett personkort kan se ut

2.4.2 Resursbehov

I resursbehovet anges hur många personer som verksamheten kräver av olika kategorier samt vilket kostnadsställe. Det finns tre olika typer av tidsintervall. Det är starttid, kontinuerlig och passvis.

Starttid: En grupp består av en eller flera perioder med starttid. Ett arbetspass startar inom en period. Därför är det bäst att passet börjar vid starttid.

Exempel:

Grupp: Namn på gruppen

Perioder: 00:00-08:00, 08:00-16:00, 16:00-24:00

Starttider: 00:00, 08:00, 16:00

Om ett pass börjar klockan 07:00 ligger den i perioden 00:00-08:00.

Kontinuerligt: Beskriver behov av antal personer för varje tidsperiod.

Datum, från klockan. – till klockan och antal personer (min – normal – max).

Exempel:

Datum: 2000-02-07, klockan 06:00-10:00, antal: två personer.

Datum: 2000-02-07, klockan10:00-14:00, antal: fyra personer.

Datum: 2000-02-07, klockan14:00-18:00, antal: tre personer.

Passvis: Beskriver behov av personal på varje pass per dag. Datum, pass och antal personer (min – normal – max).

Exempel:

Datum: 2000-02-07

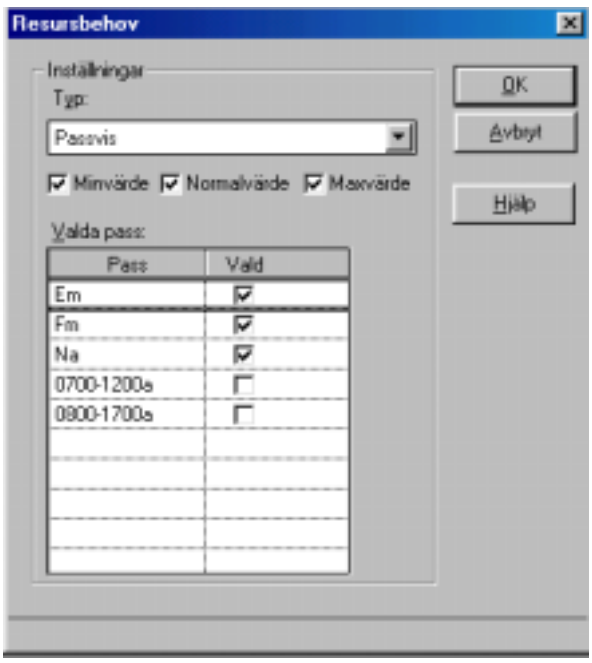
Pass:

Förmiddag: 06:00-15:00, antal: tre personer.

Eftermiddag: 14:00-23:00, antal: två personer.

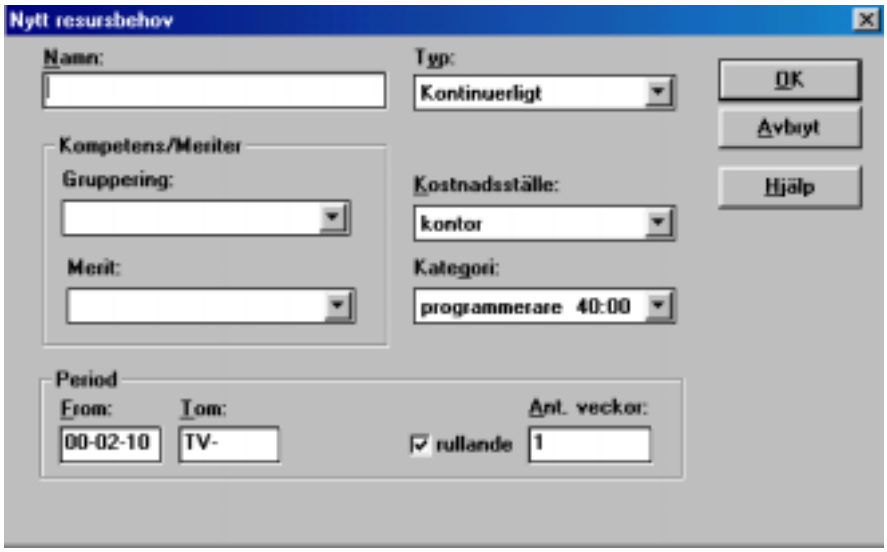
Natt: 22:00-07:00, antal: en person.

När resursbehov läggs in görs detta i tabellunderhåll i TID2000 och vissa inställningar görs. I fältet typ väljs vilken typ av resursbehov som ska gälla. Det finns som sagt tre olika typer: starttid, kontinuerligt och passvis. Resursbehovet kan visa minvärde, normalvärde och/eller maxvärde. Dessa värden anger minsta, normala respektive maximala behovet av arbetskraft. I figur 2.4 är passvis den valda typen och här görs val för de pass som skall användas i resursbehovet. När inställningarna är gjorda är det dags att logga in på dagligt arbete i TID2000.



Figur 2.4 Exempel på valda inställningar

Resursbehovet tilldelas ett namn som är en slags identifikation. Här anges vilken typ, kostnadsställe och kategori som resursbehovet ska ha. Kompetens/Meriter behöver inte vara med. Här fylls även i hur länge resursbehovet ska gälla, hur många veckor det avser och om det ska vara rullande, se figur 2.5.



Figur 2.5 Exempel på hur man gör ett nytt resursbehov

2.4.3 Schema

Först läggs ett tomt schema upp och detta fylls sedan med arbetspass. Ett arbetspass innehåller information om klockslag då arbetet ska starta och sluta samt rastens längd. Arbetspassen kan komma från passburken som innehåller fördefinierade pass. De kan matas in direkt i schemat eller kopieras från andra scheman. Arbetspassen placeras i schemats veckodagar för varje dag och varje schemarad. När ett schema är färdigt krävs det två åtgärder för att det ska bli aktiv, sätta schemat i produktion och bemanna. Om man väljer att sätta i produktion först kan man låta olika personer ha schemat under olika perioder. Om man å andra sidan väljer att bemanna först, gäller bemanningen automatiskt hela giltighetstiden, detta kallas att preliminärbemanna schemat.

Vid preliminärbemanning kopplas varje person till varsin schemarad. När ett schema fylls i skapas en bemanningspost. Varje person får en placering vid anställningen och egentligen är det bemanningsposten och placeringen som kopplas till varandra vid preliminärbemanning.

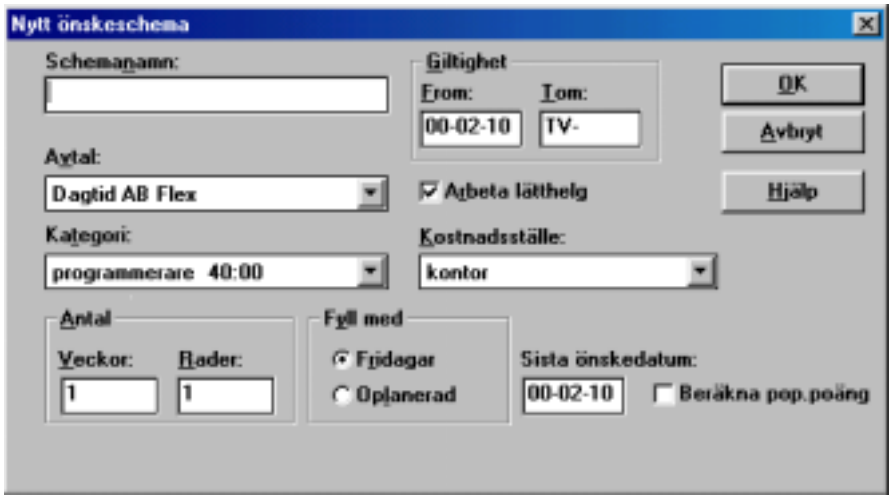
Det finns två sätt att ange flexramar:

Relativt: Flexramen anges relativt arbetspassets början och slut. Exempel ± 1 timme.
Absolut: flexramar anges i form av fasta klockslag.

Ett arbetspass består av uppgifter om klockslag då passet börjar och slutar samt längden på raster och när de ska inträffa om detta är förutbestämt. Alla olika arbetspass som används skapas och läggs i en så kallad passburk. Ur denna tas sedan de pass som önskas in i en schemarad. Det går också bra att skriva tider direkt i schemaraden.

En schemagrupp är en grupp av schemarader som används för att schemalägga en grupp av personer samtidigt där personerna ska ha individuella scheman. En rad i schemat för varje person. Denna schemagrupp kan sedan jämföras mot resursbehovet för samma grupp. När schemagruppen bemannas, vilket ofta görs först och sedan sätts i produktion, hanteras varje rad för sig.

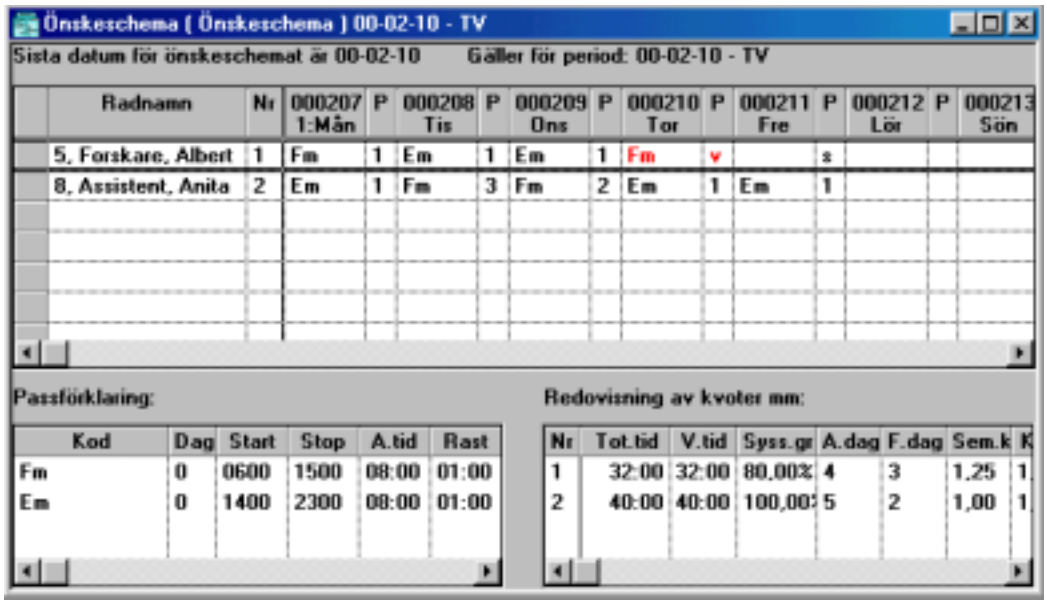
I ett önskeschema får personalen själva fylla i de arbetspass som önskas. Dessa önskningar jämförs sedan med det resursbehov som verksamheten kräver. Först skapas ett nytt schema med det datumintervall som perioden skall omfatta och en rad i schemat för varje person, se figur 2.6.



Figur 2.6 Exempel på hur man gör ett nytt önskeschema

Sedan preliminärbemannas schemat. Med det menas att varje rad i schemat knyts ihop med en viss person. Detta blir en mall som personalen själva får fylla i med sina önskningar. Arbetstiderna läggs in som pass eller genom att ange tiden i klockslag. Varje dag i önskeschemat kan anges med prioritet, hur viktigt det är att vara ledig eller arbeta en dag. Prioriteten kan

anges med v = mycket viktig, 1, 2, 3 = lägre grader av angelägenhet, s = semester, t = tjänstledighet, se figur 2.7. När det är klart och datumet för önskningar har gått ut tas önskeschemat till schemagrupp, denna schemagrupp är en kopia av önskeschemat. Eventuella ändringar görs manuellt i schemagruppen. När schemat överrensstämmer med resursbehovet skall schemat sättas i produktion. Beräkning av popularitetspoängen fungerar endast för tidsintervallet kontinuerligt.



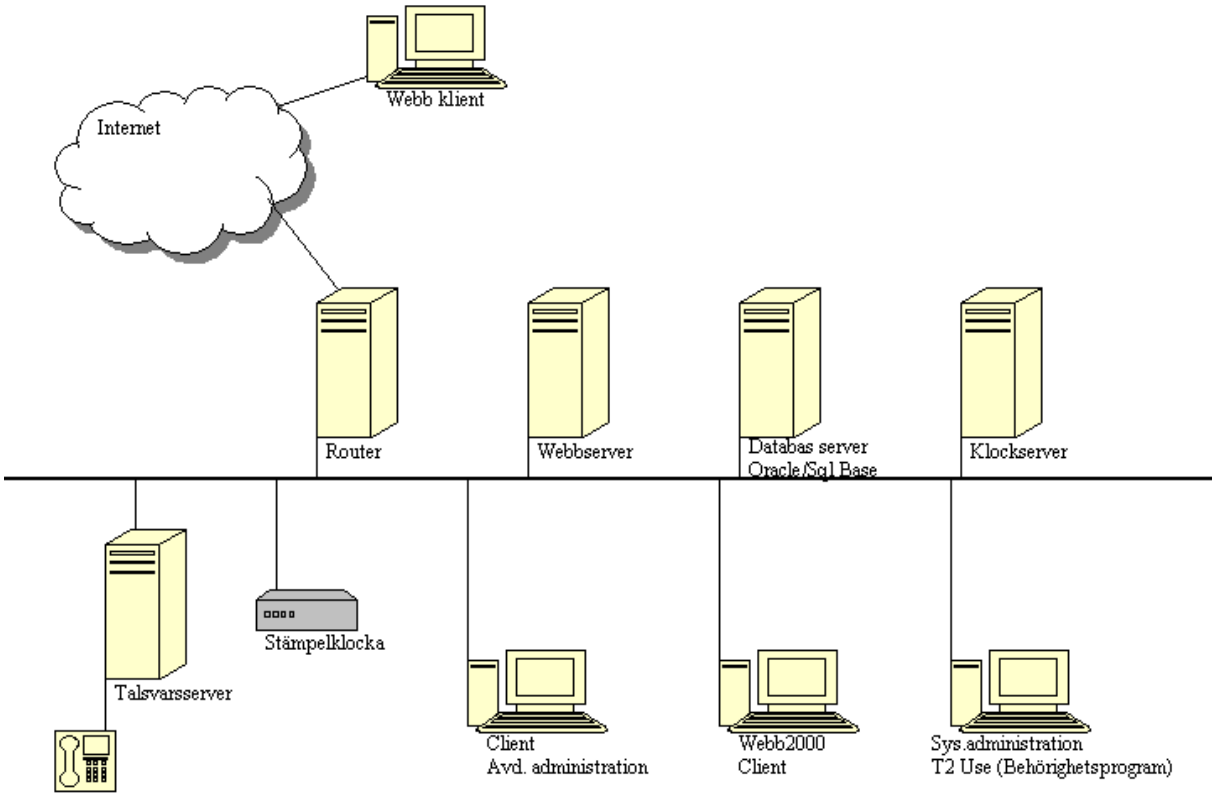
Figur 2.7 Exempel på hur ett önskeschema kan se ut

Vid schemaläggning finns kontroller för veckovila, sysselsättningsgrad och överlappande pass. Veckovilan är minst 36 timmar enligt ATL (Arbetstidslagen) [7] men kan justeras i lokala avtal. Personens arbetstid kontrolleras mot dennes sysselsättningsgrad.

2.5 TID2000 installation

En vanlig installation av TID2000 ser ut som i figur 2.8. Avdelningsadministratören sköter om den lokala delen av TID2000 när det gäller schemaläggning och dylikt. Avdelningsadministratören får sina administrationsrättigheter från systemadministratören som använder ett program för att registrera behörighet för olika användare. Stämpel-klockan har en knappsats samt en magnetkortsläsare för bland annat registrering av in/ut stämplingar, frånvaro etc. Instämplingarna registreras sedan i klockservern. Det finns även möjligheter att registrera sig via telefon. Registrering sker då via en talsvarsserver som rings upp. En relativt ny och om-

tyckt tjänst är möjligheten att via Internet/intranet ansluta sig till en webbserver för att registrera sig. Möjligheten att registrera sig via webben kan vara bra om man befinner sig på resande fot. De databashanterare som TID2000 idag använder är SQL Base samt Oracle.



Figur 2.8 Exempel på en TID2000 installation

3 Förutsättningar och krav

Nedan beskrivs vilken data användaren är ansvarig att mata in till programmet samt de krav som ställs på algoritmen. Kapitlet tar även upp de avgränsningar som gjorts under arbetets gång.

3.1 Schemaläggningsförutsättningar

Schemaläggaren lägger ett resursbehov och gör ett urval, och bestämmer därefter vilka personer som ska schemaläggas. Schemaläggaren bestämmer också poängen på passen och vilken prioritet reglerna ska ha. En mer noggrann beskrivning av reglerna ges i kapitel 4.

Poängen är till för att en person inte ska få för många icke-populära pass. Om ett pass är populärt eller inte bestämmer alltså schemaläggaren. Schemaläggaren kan även välja att inte ha någon poäng alls och då blir alla pass likvärdiga. Hög poäng betyder att passet inte är populärt. Poängsättningen kan även automatiseras genom att schemaläggaren endast behöver ange om han vill ha poäng eller inte. Om han anger ja genom ett val vid resursbehovets skapande så kan passen tilldelas fördefinerade värden baserade på inom vilken tidsperiod passen ligger, t.ex. alla pass som börjar på vardagar efter klockan 22.00 tilldelas 3 poäng. Om detta pass legat på en lördag eller söndag skulle det ge en högre poäng, exempelvis 5 poäng.

Prioriteten kommer till användning då algoritmen kört fast och måste ”ta bort” en regel eller ett block av regler beroende på implementation och börja om för att få fram ett schemaförslag.

Schemaläggaren ska kunna mata in följande data:

- Namn på schema (någon slags identifikation).
- Tidsperiod (fr.o.m. – t.o.m.).
- Antal veckor (hur länge schemat ska gälla).
- Antal rader (hur många personer schemat ska innehålla).
- Kostnadsställe (exempel kontor eller verkstad).
- Om schemaläggaren vill använda sig av poäng eller ej.

- Prioriteter (d.v.s. vissa parametrar är viktigare än andra).
- Gränser för avvikelse från sysselsättningsgrad (antalet timmar +/-).

Schemaläggaren ska också kunna skriva in regler för personer vid anställning som t.ex.:

- Att varje person ska arbeta t.ex. var tredje helg.
- En person arbetar ej fredagar.
- Kan ej samarbeta med viss person.

3.2 Krav

Algoritmen ska konstruera ett schemaförslag så att resursbehovet uppfylls och att schemat blir så bra som möjligt för samtliga personer. Algoritmen ska också ta hänsyn till tidigare scheman. Som utdata ska förutom ett schemaförslag också finnas någon slags poängsättning på personer som tas med till nästa konstruktion.

En prototyp ska implementeras. Den kan innehålla ett mindre antal parametrar än vad en fullversionsimplementation av algoritmen skulle ha innehållit. Prototypen ska vara lätt utbyggbar för att kunna implementera ytterligare parametrar för framtida behov. Ett antal olika tester ska sedan utföras på prototypen beroende på vilka parametrar man valt att ha med i prototypen.

Det finns dock vissa regler som baserar sig på avtal som ska vara uppfyllda för personer:

- Veckovila 36 timmar i följd ska vara uppfyllt (kan ändras genom lokala avtal).
- Dygnsvila 11 timmar i följd (kan ändras genom lokala avtal).
- Max arbetstid 12 timmar i följd.

Om vi skulle komma fram till att det inte är möjligt att göra en algoritm för automatisk schemakonstruktion, ska detta dokumenteras. Orsaker om varför vi inte lyckades ska i så fall specifieras.

3.3 Avgränsningar

TID2000 är ett stort program som tar hänsyn till många olika inparametrar. Vid schema-konstruktion har vi gjort vissa avgränsningar. Vi tar **inte** hänsyn till följande:

- Jour- och beredskapsarbete.
- Flextid.
- Resursbehovet läggs enligt typen starttider och kontinuerligt, se figur 2.4.
- Min- och maxbehov av arbetskraft på pass, se figur 2.4.

4 Regler

I det här kapitlet beskrivs först vilka regler som algoritmen ska ta hänsyn till. Därefter ges en mer detaljerad beskrivning av hur arbetsstyrningsreglerna kontrolleras.

4.1 Allmänt

Det finns regler som ska kontrolleras innan en person kan schemaläggas på ett pass. Reglerna delas in i två kategorier:

- Personregler

  Regler som gäller specifikt för en person t.ex. arbetar ej X-pass, planerad frånvaro, X ska arbeta denna dag.
- Arbetsstyrningsregler

  Regler som kontrollerar en persons tillgängliga timmar, max arbetstid, dygnsvila, veckovila, antal poäng och antal helgdagar.

4.2 Tillgängliga timmar

För att en person ska kunna bli schemalagd på ett pass så måste den ha några timmar kvar att schemalägga. Detta kontrolleras genom att varje person tilldelas ett antal timmar varje vecka som den ska arbeta baserad på sin sysselsättningsgrad. Timmarna summeras och läggs i ”tillgängliga timmar”. I tillgängliga timmar läggs också eventuella resttimmar eller ”tillgodotimmar” från föregående schemaperiod. När personen har blivit schemalagd på ett pass minskas tillgängliga timmar med passets antal timmar. När hela perioden är schemalagd ska differensen mellan antalet timmar som personen ska arbeta och antalet timmar som personen blivit schemalagd helst vara noll. Alltså parametern ”tillgängliga timmar” ska vara noll. Algoritmen tillåter en avvikelse och denna avvikelse bestämmer schemaläggaren innan schemaläggningen sätts igång. Om det förekommer en avvikelse (± antal timmar) sätts denna som en parameter i arbetsstyrningregler och dessa timmar förs över till tillgängliga timmar i nästa schemaperiod.

4.3 Maximal arbetstid

Om personen har tillgängliga timmar så kontrollerar algoritmen ifall personen förekommer på något annat pass samma dag. Detta görs för att se till att personen inte får överlappande pass och överskrider 12 timmar, vilket är max arbetstid. Algoritmen börjar med att schemalägga det första passet på dagen. Allteftersom den arbetar sig nedåt i passen så måste den kontrollera uppåt i de pass som blivit schemalagda samma dag. Är personen schemalagd på ett pass samma dag, kontrollerar algoritmen start- och stopptid för passet algoritmen står på. Detta görs så att personen inte blir schemalagd på två överlappande pass. Är det ett överlappande pass får personen bara det pass som den redan var schemalagd på. Är det däremot inte överlappande pass summerar man antalet timmar på detta pass och med det pass personen redan är schemalagd på. Är summan av antalet timmar av passen mindre än eller lika med tolv kan personen arbeta båda passen. Är det mer än tolv timmar får personen bara det pass som den redan var schemalagd på.

4.4 Dygnsvila

Under ett dygn ska det finnas 11 lediga timmar mellan föregående dags pass slut och nästa dags pass början. Detta ska algoritmen kontrollera genom att gå 12 timmar bakåt respektive framåt från passets början respektive slut av det pass som ska schemaläggas för att se om dygnsvilan på 11 timmar i följd uppfylls.

4.5 Veckovila

Under en vecka ska det finnas 36 timmar i följd då en person ska vara ledig. Veckovilan kontrolleras på samma sätt som dygnsvilan. 84 timmar kontrolleras från passets början respektive slut för att se om det finns 36 timmar i följd inom detta intervall.

4.6 Antal poäng

Varje gång en person schemaläggs på ett pass får personen ett antal poäng beroende på hur många poäng passet är värt. Poängen summeras till personens övriga poäng. Hög poäng betyder att passet inte är populärt. En person som har hög poäng har alltså inte lika stor chans att få ett mindre bra pass än en person som har låg poäng.

4.7 Antal helgdagar

För att det ska bli rättvist med antal helgdagar som personerna ska arbeta måste en parameter sättas till en person som indikerar hur många helgdagar personen har schemalagts på. När denna parameter visar det antal helgdagar som personen ska schemaläggas på kan personen inte schemaläggas någon mer helgdag under tidsperioden.

5 Algoritm

I det här kapitlet beskrivs en möjlig algoritm för automatisk schemakonstruktion.

5.1 Algoritmbeskrivning

Vi har försökt göra algoritmen så att den lätt kan byggas ut med ytterligare funktioner för kontroll av ytterligare parametrar. Detta har vi gjort genom att lägga kontroller på en person på samma plats i algoritmen, d.v.s. vi kan lägga till parametrar till en person och funktioner för att kontrollera dessa utan att behöva ändra algoritmens uppbyggnad.

Algoritmen ska börja med att göra en jämförelse mellan resursbehovet och de personer som ska schemaläggas för att kunna avgöra om man kommer att bli underbemannad eller kommer att schemalägga mer personal än vad som behövs. En person tilldelas ett antal timmar varje vecka som den ska arbeta baserad på sin sysselsättningsgrad. Timmarna summeras och läggs i tillgängliga timmar. I tillgängliga timmar läggs också resttimmar från föregående schemaperiod. När algoritmen ska jämföra resursbehov och urval summerar den antalet tillgängliga timmar för varje person och jämför dessa timmar med det totala antalet timmar i resursbehovet. Den kommer att tillåta en ställbar differens mellan resursbehov och urval. Stämmer resursbehovet överens med tilltänkta personer som ska schemaläggas så ska den automatiska schemaläggningen sättas igång, annars skrivs ett meddelande ut på skärmen som talar om att resursbehovet är för stort alternativt för lite jämfört med urvalet.

Efter att ha jämfört resursbehov och urval fortsätter algoritmen med att ställa sig på första dagen och första passet i resursbehovet. Algoritmen tar fram personer som ska och kan arbeta detta pass genom att kontrollera personerna i urvalet mot personreglerna och arbetsstyrningsreglerna. Personerna som klarar alla kontroller lagras tillfälligt i en datastruktur. När alla personer i urvalet har testats så kopieras den tillfälliga datastrukturen.

Den tillfälliga datastrukturen används för att plocka ut personer och lägga in dem i en ”permanent” datastruktur. Den ”permanenta” datastrukturen ska sedan användas vid schemaläggning av personer på pass. Den bestämmer i vilken ordning som personerna ska schemaläggas. Först tas personer som ska arbeta. Personerna läggs in enligt följande kriterier:

- Personen med lägst poäng
- Vid lika poäng så slumpas ordningen för personer fram.

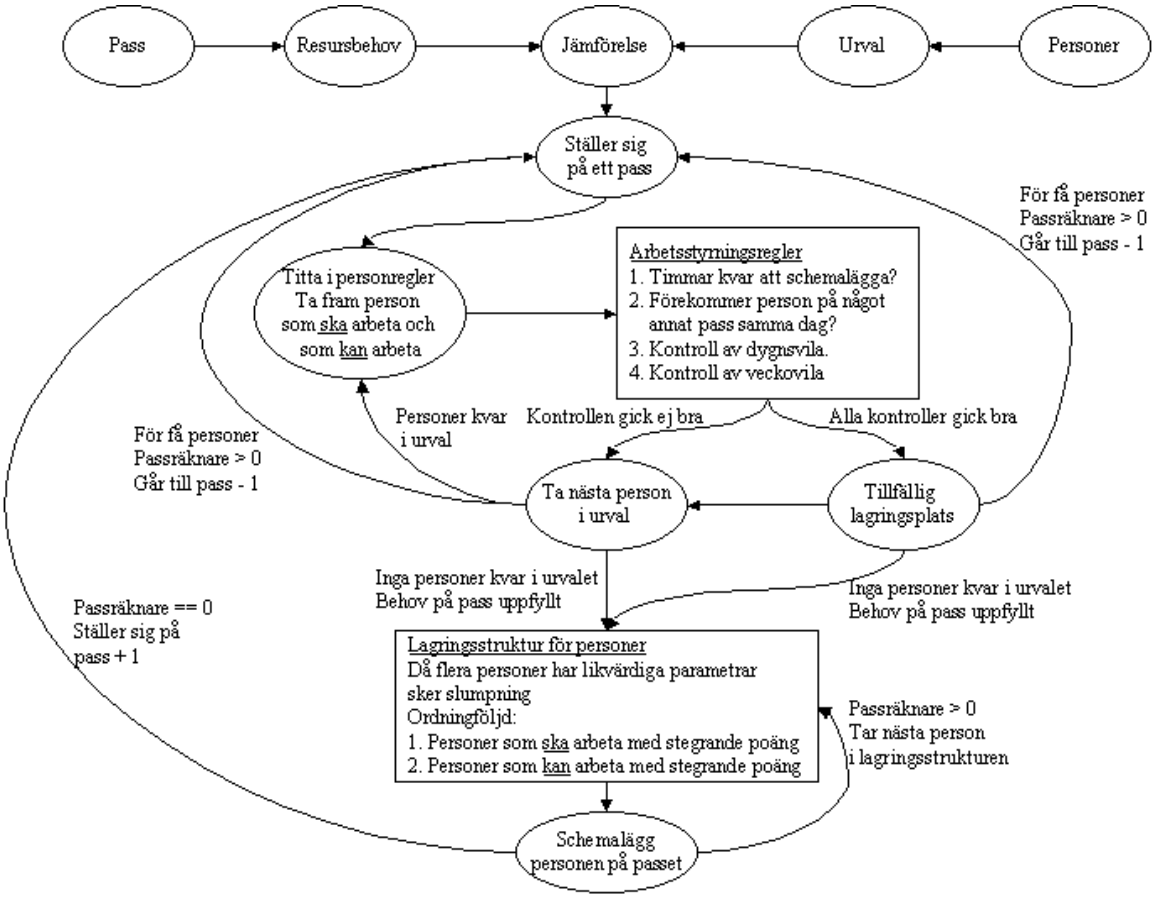
Därefter görs samma procedur för personer som kan arbeta detta pass.

Schemaläggningen startar därefter från första position i den ”permanenta” datastrukturen. Så länge som det finns pass att schemalägga går algoritmen tillbaka till den ”permanenta” lagringsstrukturen och tar nästa position. När passräknaren har nått noll går algoritmen ur loopen och går till nästa pass. När dagens alla pass är schemalagda går den vidare med nästa dag. Detta gör den tills den nått slutet på schemaperioden.

Om antalet personer som klarat kontrollen är mindre än behovet på passet så går algoritmen till föregående pass och söker i den ”permanenta” datastrukturen. Den person som blivit schemalagd byts ut mot nästa person i den ”permanenta” datastrukturen för att på sätt se om man på detta vis kan gå vidare med schemaläggningen. Denna så kallade ”backtracking” görs så länge som det finns personer kvar i den ”permanenta” datastrukturen. Om det efter att man gått till sista position i den ”permanenta” datastrukturen fortfarande inte går att komma vidare så går man tillbaka ytterliggare ett pass och upprepar samma procedur.

Algoritmen kan komma till ett skede då den har gått igenom alla möjliga fall utan att fått fram ett schemaförslag. Schemaläggaren får då uppmaningen att släppa på ett eller flera parameterkrav för att börja om schemaläggningen från allra första början. Vilken eller vilka parametrar som inte ska kontrolleras har schemaläggaren tidigare angivit genom prioritet.

I figur 5.1 illustreras hur algoritmen ska fungera.



Figur 5.1 Algoritmflöde

5.2 Alternativa lösningar

Under arbetets gång har vi försökt förfina vår lösning. Här följer några idéer som vi tidigare har funderat på innan vi bestämde oss för den beskrivna algoritmen.

Baserat på egna erfarenheter tror vi att en person som schemaläggs på en lördag även skulle vilja bli schemalagd söndagen samma helg. Vi har tagit fram en tänkbar lösning när det gäller schemaläggning av helger. Om en person schemaläggs på en lördag så försöker algoritmen att schemalägga söndagen med samma person. Personerna som blir schemalagda på lördagen sparas i ett ”lördagsurval” för att kunna användas på söndagen. Om personerna i ”lördags-urvalet” inte kan arbeta söndagen eller om det inte finns tillräckligt med personer i ”lördagsurvalet” för att täcka söndagens behov så tar algoritmen fram nya personer på samma sätt som den tar fram personer till vardagarna.

Vi funderade länge på om algoritmen skulle schemalägga de sämsta passen först för att på så sätt ha många personer i urvalet att välja bland. Vi antog att de sämsta passen var helg- och nattpass. Algoritmen skulle då börja med att schemalägga helger först och sedan vardagarna. Algoritmen skulle också börja schemalägga nattpassen först för en dag och sedan schemalägga de bästa passen sist. Den här lösningen förkastade vi med tanke på att det var vi som hade bestämt vilka pass som skulle vara dåliga respektive bra. Det kanske är så att helg- och nattpassen är populära pass på ett företag t.ex. passen ger bra ersättning. Vi tyckte det var fel att vi skulle bestämma om ett pass skulle vara populärt eller inte. Nu låter vi schemaläggaren istället få bestämma passens popularitet genom att få ange poäng på passen. Själva tankesättet är nog inte helt fel att använda sig utav i den algoritm vi beskrivit. Eftersom schemaläggaren kan få poängsätta passen så vet vi om ett pass är sämre än ett annat och på så sätt schemalägga pass med hög poäng först.

En lösning som vi också höll kvar vid länge var att algoritmen skulle slumpa fram personer och sedan göra en parameterkontroll på personen. Om personen inte gick igenom kontrollen skulle en ny person slumpas fram o.s.v. Vi kom fram till att detta inte var någon bra lösning. Eftersom algoritmen då inte tog någon hänsyn till om det fanns personer som verkligen skulle arbeta en viss dag eller inte.

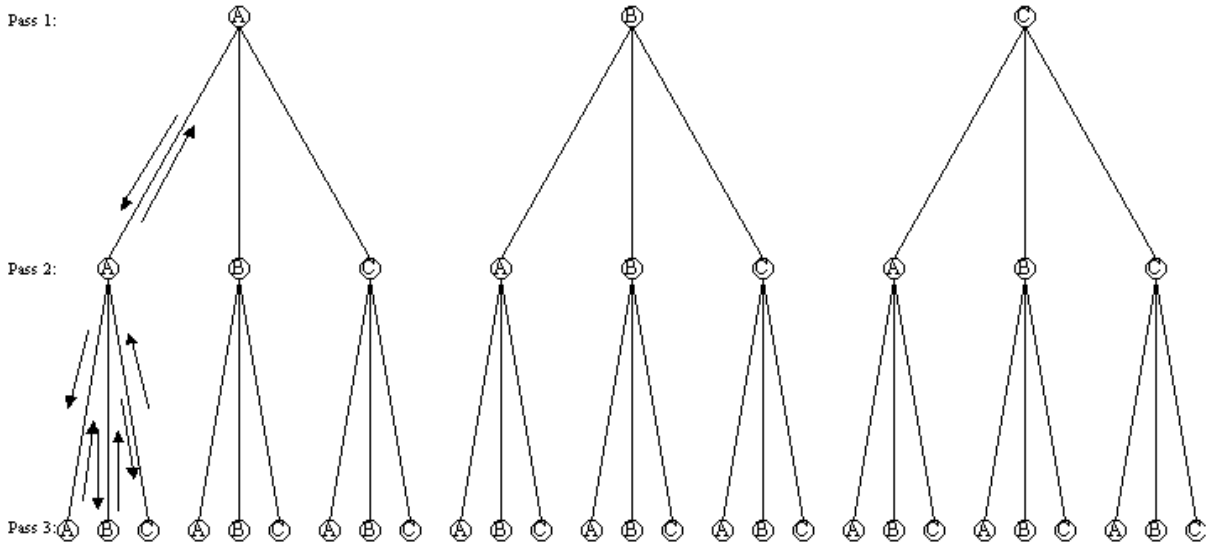
Vi hade flera tänkbara lösningar om algoritmen skulle komma till ett skede då den inte kan schemalägga någon person. Den ena var att släppa på något parameterkrav för att kunna gå vidare. En annan lösning var att generera flera scheman parallellt eller seriellt. Ett tredje alternativ var att algoritmen skulle rekursivt försöka gå tillbaka och byta ut en person mot en alternativ lösning. Det sista alternativet var att algoritmen skulle använda sig av en trädstruktur och gå tillbaka till närmast föregående framkomliga väg och fortsätta nedåt i trädet därifrån. Till slut så blev det alltså en blandning av dessa alternativa lösningar.

6 Analys av algoritmen

I detta kapitel beskrivs ett räkneexempel på algoritmen då den måste gå igenom alla möjliga fall.

6.1 Räkneexempel på algoritmen

Vi har tänkt att algoritmen ska producera flera parallella temporära scheman där varje schema ska fungera som en trädstruktur. Om algoritmen inte kan schemalägga en person på ett pass ska den gå tillbaka till föregående pass och försöka med nästa person som vi beskrivit tidigare. Vi ska illustrera detta med ett exempel, se figur 6.1.



Figur 6.1 Trädstruktur på räkneexempel

Vi ska ta fram ett ”värsta fall” d.v.s. då algoritmen måste gå igenom alla möjliga fall till det sista för att få fram ett schema. I exemplet som vi nu ska visa har vi bara tagit med tre personer och algoritmen ska bara schemalägga tre pass. Vi kallar personerna A, B och C och sätter passen till 1, 2 och 3. Vi har gjort en del begränsningar i exemplet. Vi har inte tagit med några speciella regler, inte satt några timmar på passen och inte talat om vilka dagar passen hör till. Pass 1, 2 och 3 kan höra till samma dag eller ligga på varsin dag. Vi börjar med pass 1. Vi testar med person A och ser ifall personen kan arbeta. Person A kan arbeta och schema-

läggs på passet. Nästa pass som ska schemaläggas är pass 2. Vi testar med person A. Person A kan arbeta detta pass också, den schemaläggs på passet och algoritmen går vidare till pass 3. Testar person A. Eftersom vi ska ta fram ett ”värsta fall” d.v.s. gå igenom alla möjliga fall så vore det inte bra att säga att person A kan arbeta detta pass eftersom då har vi schemalagt alla pass och bara fått fram ett fall. Vi säger att person A inte kan arbeta pass 3. Algoritmen går tillbaka till föregående pass och testar ifall person B kan arbeta pass 3. Person B kan heller inte arbeta detta pass eftersom då skulle vi bara få fram två fall och vi vill få fram alla möjliga fall. Algoritmen går då tillbaka till föregående pass igen och testar med person C. C kan heller inte. Nu får algoritmen gå tillbaka allra första passet och därifrån testa om B kan arbeta pass 2. Person B kan arbeta pass 2 och algoritmen går vidare till pass 3. Så här fortsätter algoritmen tills den fått fram alla möjliga fall och det har vi fått då person C blivit tilldelad alla pass.

Genom att se på figur 6.1 får vi fram följande
Välja ut en person till pass 1
Antal fall: $3 = 3^1$
Välja ut en person till pass 2
Antal fall: $3 * 3 = 3^2 = 9$
Välja ut en person till pass 3
Antal fall: $3 * 3 * 3 = 3^3 = 27$

Vi kan nu se ett samband mellan antal pass och antal personer och får fram formeln som gäller för att beräkna antal fall vid ”värsta fall” lösning [3].

$f(x) = x^y$ där x = antal personer
och y = antal pass

27 fall låter kanske inte så mycket men då ska man ta hänsyn till att vi bara har använt oss av tre personer och ska bara schemalägga tre pass. Ett riktigt schema innehåller vanligtvis både fler personer och pass. Vi kan med hjälp av formeln se vad vi får om vi istället tar 5 personer och 30 pass.

$5^{30} = 931322574615478515625$ fall

Nu ser man ganska tydligt hur stort trädet blir. Ändå är inte 5 personer och 30 pass mycket, det är egentligen ett ganska litet schema. Utav fallen som vi fått fram finns det visserligen flera som man kan räkna bort. T.ex. en person kan knappast bli schemalagd på 30 pass i en

följd om det inte är väldigt korta pass förstås. Men om man skulle räkna på 8-timmars pass så skulle aldrig en person kunna få två pass i rad som ligger på samma dag.

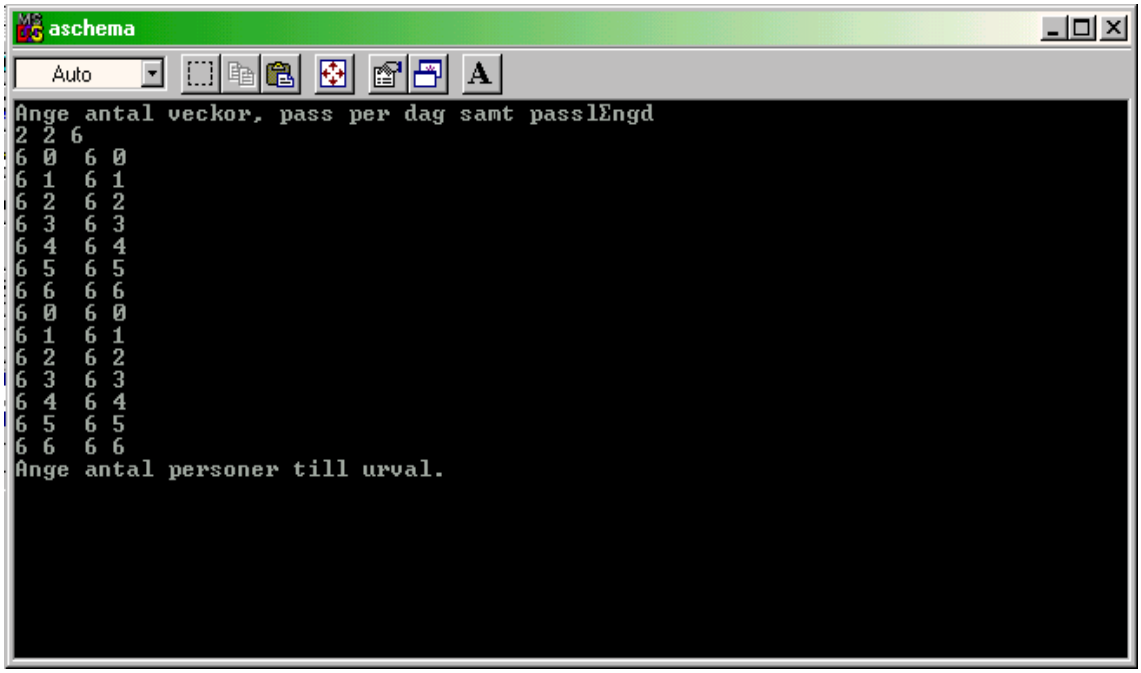
Med tanke på att det blir väldigt stort så måste man nog göra vissa begränsningar i antalet fall per pass. I den algoritm som vi har tagit fram gör vi inga begränsningar gällande antalet möjliga fall.

7 Implementering och test

Vi skulle försöka göra en prototyp av algoritmen som vi tagit fram. Vi valde att som implementationsspråk använda C++ [4]. I C++ använde vi oss av STL (Standard Template Library) [1] [8] för att slippa skriva alla de datastrukturer vi behövde för att lagra och söka i när algoritmen körs. Vi hoppades att detta skulle bespara oss arbetet med att göra egna länkade listor, dynamiska vektorer och dylikt, vilket det också gjorde.

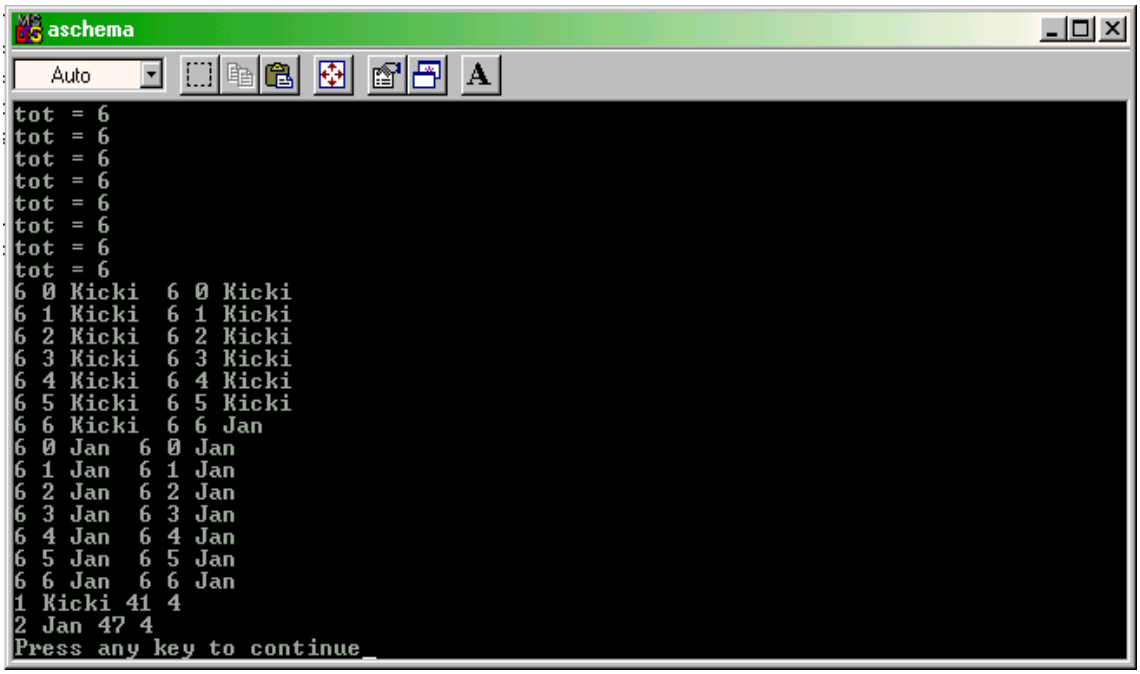
Vi började med att strukturera upp olika klasser för person, personregler, urval, dag, pass, resursbehov, schema och schemaregler. Eftersom det var en prototyp vi skulle implementera begränsade vi oss till att passens timmar var lika för alla pass och antal pass per dag var lika. Personer som skulle kunna tänkas vara med i urvalet automatgenererades från en array med namn.

Vi fortsatte med att göra en inmatningsfunktion där schemaläggaren får ange antal veckor, pass per dag och antal timmar per pass. Detta ska vara resursbehovet och ligga till grund för schemaläggningen.



Figur 7.1 Inmatning för resursbehov

Sedan får schemaläggaren mata in hur många personer som önskas till detta resursbehov. Efter att ha matat in antal personer så skapas dessa personer. Personerna får slumpvis valda timmar tilldelade genom en slumpfunktion som slumpar tal mellan 20 och 60 timmar. Dessa timmar ska utgöra parametern tillgängliga timmar. I tillgängliga timmar finns de timmar en person ska arbeta baserad på sysselsättningsgraden och eventuella resttimmar från föregående period, därav slumpningen mellan 20 och 60 timmar.



Figur 7.2 Schemaresultat

Vi fortsatte med att göra en funktion som summerade timmarna i resursbehovet och timmarna för de utvalda personerna. Totalsummorna jämförs med varandra. Vi lät algoritmen tillåta en differens mellan resursbehov och urval eftersom det kanske är helt omöjligt att få att timmarna stämmer exakt och på så sätt aldrig kan börja schemalägga på grund av att det fattas en timme. I prototypen har vi satt denna gräns till ±10 %. Om differensen är högre eller lägre än den tillåtna så skrivs ett meddelande ut att det är för få eller för många personer och programmet bryts.

Enda testen vi hann att implementera var en kontroll på tillgängliga timmar. Vi försökte även att implementera en kontroll på max arbetstid, men på grund av tidsbrist var vi tvungna att avbryta för det tog för mycket tid att spåra ett stegfel i implementationen. Trots detta enda

test tycker vi dock visar att vår algoritm vid en utbyggnad med välimplementerade funktioner för att kontrollera parametrar har en god möjlighet att generera ett schema.

8 Erfarenheter och rekommendationer

Uppgiften vi blev ålagda att utföra var ganska komplex d.v.s. det fanns mycket man var tvungen att ta hänsyn till bl.a. arbetstidslagar och hur TID2000 ser ut idag. Vi grep oss dock an uppgiften med stor optimism, men upptäckte snart att vi såg på problemet på två olika sätt. Detta ledde till ett antal diskussioner om hur uppgiften skulle lösas. Efter diskuterande och med ökad förståelse för hur den andre tänkte lyckades vi med att kombinera våra olika infallsvinklar till en som vi tycker bra och fungerande algoritm.

Det hade också varit bra om kravspecifikationen varit starkare. Nu fick vi själv ställa upp en del krav som vi tyckte var nödvändiga. Att kravspecifikationen var lite vag var inte enbart dåligt för det gav oss lite fritt utrymme för egna funderingar på vad som kunde krävas av en kravspecifikation.

En annan erfarenhet som vi fått genom att göra detta examensarbete är att det är väldigt svårt att göra en tidsplan som fungerar. Den tid vi avsatte för implementering och test blev alldeles för kort.

9 Slutsatser

Examensarbetets syfte var att se om det var möjligt att ta fram en algoritm för automatisk schemakonstruktion. Algoritmen skulle helst generera ett schema som var så rättvist som möjligt för samtliga personer som skulle schemaläggas.

Vi kom fram till att vi aldrig kan göra en algoritm som får fram ett schema som är helt rättvist för alla anställda. Ett schema kan bli så rättvist som möjligt beroende på antalet parametrar algoritmen väljer att använda. Generellt gäller att ju fler parametrar algoritmen väljer att kontrollera, ju mer rättvist blir schemat men det kan då bli svårt att generera ett färdigt schema.

Vi tycker att detta examensarbete varit roligt att göra. Även om uppgiften var svår hade vi möjlighet att göra vissa begränsningar så att det var möjligt att slutföra den inom den tid vi hade till vårt förfogande. Vi tycker att vi har lyckats med uppgiften. Det hade dock varit roligt om vi lyckats få med fler kontroller i implementationen av algoritmen.

Slutligen har vi även fått en liten försmak av hur det kan vara att arbeta med programutveckling gentemot en beställare av ett program.

Referenser

[1] Borland C++ Programmer's Guide, Borland International, ISBN: 0672309238, Version 5, 1996

[2] Date C.J, An Introduction to Database System, Longman Higher Education Division, ISBN: 0201385902, first edition, 1999

[3] Grimaldi Ralph P, Discrete and Combinatorial Mathematics, Addison-Wesley Publishing Company, ISBN: 0201600447, third edition, 1994

[4] Stroustrop, Bjarne, The C++ Programming Language, Addison-Wesley Longman Publishing Co., ISBN: 0201889544, July 1997

[5] Tanenbaum, Andrew S, Computer Networks, Prentice Hall, ISBN: 0133342481, third edition, 1996

[6] Användarhandbok. EuroTime AB TID2000 för windows, 1997-10-31

[7] ATL (Arbetstidslagen) http://www.jit.se/lagbok/982673t.html

[8] MSDN Library – October 1999 http://msdn.microsoft.com/

[9] http://www.eurotime.se

A Defintioner och begrepp

Appendixet ger en lista med förklaringar av olika definitioner, begrepp och förkortningar som förekommer i denna examensrapport.

ASP – Active Server Page
Scriptspråk för dynamiska webbsidor.

ATL – ArbetstidsLagen

SQL – Structured Query Language
Frågespråk för databaser.

STL – Standard Template Library
Ett ANSI/ISO bibliotek med olika fördefinierade datastrukturer.