CODEN:LUTEDX/(TEIE-5459)/1-53(2021)

Electrical Engineering and Automation

Industrial

Optimering av sekvenser i produktionsplanering

Erik George Kronberg

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Optimering av sekvenser i produktionsplanering

Examensarbete av
Erik George Kronberg

För en civilingenjörsexamen inom Maskinteknik

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LUND TEKNISKA HÖSKOLA
AVDELNINGEN FÖR
INDUSTRIELL ELEKTROTEKNIK OCH AUTOMATION
Lund, Sverige

2021
Försvarad den 31 mars

FÖRORD

Detta arbete markerar slutet på flera månaders arbete, många års studier och utgör en stor milstolpe i mitt liv. Inte bara har det gett mig möjligheten att tillämpa mycket av det jag har lärt mig under årens många kurser, utan också att inhämta massor med nya kunskaper inom framförallt optimering. Det grundläggande problemet med produktionsoptimering har varit mycket intressant och det har varit roligt att ta fram något som kommer att vara till nytta för inte bara Novotek* utan förhoppningsvis många andra företag. Ett speciellt tack vill jag ge mina handledare på Novotek Jens Molin och Nils Lindqvist, som formulerade problemet och bistod med mycket hjälp och stöd. Jag vill också tacka min handledare på LTH Gunnar Lindstedt och min examinator Ulf Jeppsson som först introducerade mig till ämnet genom sina kurser i automation. Slutligen vill jag tacka alla medarbetare jag har fått träffa på Novotek som med sitt varma välkomnande har gett mycket motivation.

*Novotek Sverige AB, del av Novotek arbetar med lösningar för industriell IT och automation. De distribuerar ett flertal programvaror inom styrning, planering och övervakning från bland annat GE. Deras huvudkontor ligger i Malmö och koncernen har ett flertal bolag i bland annat Benelux, Irland, Schweiz och alla nordiska länder. Examensarbetet formulerades av Novotek eftersom de under en lång tid har noterat en efterfrågan från flera av deras kunder på ställtidsoptimering inom planering. Syftet med algoritmerna är att kunna implementera dem och enkelt modifiera efter olika kunders specifika behov.

SAMMANFATTNING

I det här arbetet implementeras två heuristiska optimeringsalgoritmer för att lösa ett speciellt fall av ett schemaläggningsproblem.

Schemaläggningsproblem av typen *parallella resurser* är vanligt förekommande i industrin. Ett antal operationer ska schemaläggas på ett antal resurser och sekvensen påverkar ställtid, leveransförseningar och dödtid. Schemat ska optimeras för att minimera dessa variabler och detta ska göras på en skälig tid och resultera i en jämn arbetsbelastning mellan resurserna. För att lättare kunna applicera kända optimeringsmetoder formuleras problemet först som ett handelsresandeproblem med flera säljare, där städer motsvarar operationer och säljare motsvarar resurser. En matematisk beskrivning som ett linjärt program tillkommer. Som optimeringsalgoritmer väljs en genetisk algoritm och simulerad kylning eftersom dessa är väldokumenterade och har tidigare visat goda resultat för just schemaläggningsproblem. Rapporten innehåller en utförlig presentation av dessa algoritmer gällande deras funktion, parametrar, för- och nackdelar. För att testa algoritmernas funktionalitet för det givna problemet genomförs en rad fallstudier med olika mål, mer specifikt resultat (fitness), exekveringstid och robusthet. Resultaten från fallstudierna visar att den genetiska algoritmen hittar bättre lösningar och är mer stabil men tar längre tid. Resultatet för den genetiska algoritmen blir dessutom bättre då antal individer ökas, ett beteende som simulerad kylning inte påvisar i någon större utsträckning. För exekveringar där antalet individer istället minskas presterar simulerad kylning bättre. Resultaten visar också att antal resurser inte påverkar exekveringstiden, att antal operationer har en ökande, linjär påverkan på tiden vid fixt antal individer samt en exponentiell ökning om både antal individer och antal operationer ökas. Sammanfattningsvis är ingen av algoritmerna klart bättre än den andra utan omständigheterna och användarens preferenser avgör vilken som borde användas.

ABSTRACT

In this thesis, two heuristic optimization algorithms are implemented to solve a special case of scheduling problems.

The industrial configuration with parallel resources is common. A number of operations are to be scheduled on a number of resources and the sequence of the operations affects the setup time, due dates and dead time. The schedule should be optimized by minimizing those variables and this should be done in a reasonable time and result in an equal work load between the resources. The problem is at first modelled as a multiple traveling salesman problem, where the cities corresponds to operations and the salesmen corresponds to resources. A mathematical formulation is included. A genetic algorithm and simulated annealing are used as optimization algorithms, which are both well documented and have been proven effective for this type of problem. The report includes a thorough description of the algorithms and their functions, parameters, pros and cons. To test the functionality of the algorithms for this problem, four different case studies are done which test the result, execution time and robustness. The results from the case studies show that the genetic algorithm finds better solutions and are more robust but requires more CPU time than simulated annealing. However, when the number of individuals however is low, simulated annealing finds a better solution. The genetic algorithm keeps finding better solutions if the number of individuals is increased, which simulated annealing does not. The results also show that execution time grows proportionally to the number of operations and as an exponential function if the number of individuals is increased as well. In summary, one of the algorithms can not be said to be better than the other, it depends on the preferences of the users.

INNEHÅLL

FIGURER

TABELLER

NOMENKLATUR

Fenotyp En avkodad genotyp. Representerar en fullständig lösning till problemet med både sekvens och allokering.

Fitness Ett värde som används för att jämföra individer. Värdet genereras av fitnessfunktionen.

Fitnessfunktion Används för att beräkna fitness. Baserad på målfunktionen men ofta manipulerad.

GA, genetisk algoritm En heuristisk optimeringsmetod vilken är baserad på teorin om det naturliga urvalet.

Gen En gen är den minsta beståndsdelen i en individ. En gen motsvarar en operation.

Genotyp Del av en individ som består av ett antal gener. På genotypen utförs olika genmodifikationer.

Individ Består av en genotyp, fenotyp och fitness. En individ representerar en lösning till problemet.

mTSP multiple Traveling Salesman Problem *sv. handelsresandeproblemet med flera säljare* - en generalisering av TSP med flera säljare, se TSP

Operation En process som ska utföras på en viss resurs. Varje operation har en processtid och är del av en order.

Order En beställning på en produkt och består av en eller flera operationer som krävs för att färdigställa denna. Operationerna i en order ska ofta utföras i en specifik ordning och är således sekvensberoende, vilket på engelska benämns som *routing*. I engelsk litteratur benämns order ofta som *Job*.

Resurs Ett konstnadsställe som till exempel kan vara en maskin såsom en svarv , en färspruta, eller en cell men också verktyg, transportband, människor, kylrum etcetera.

SA, Simulerad Kylning En heuristisk optimeringsmetod vilken är baserad på glödgningsprocessen (*eng. Simulated Annealing*).

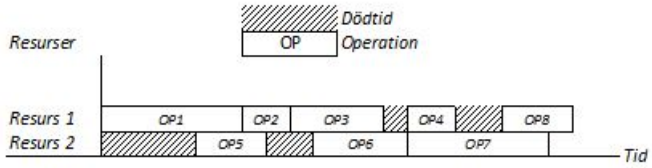
TSP Traveling Salesman Problem *sv. handelsresandeproblemet* - ett mycket känt matematiskt optimeringsproblem

Kapitel 1

INTRODUKTION

1.1 Bakgrund

Produktionsplanering är en viktig del av modern industri och har varit så under lång tid. Behoven av en effektiv produktion blir allt högre med växande konkurrens och ökade krav på hållbar tillverkning. Konsumtionssamhället har lett till en eskalerad produktion där 1000-tals produkter ibland tillverkas inom loppet på bara några minuter, där varje sekund är viktig. En bra planering kan spara tid och åstadkomma en effektivare produktion vilket i förlängningen blir en kostnadsbesparing och detta har gjort att planering har fått en central roll, tilldelad mycket uppmärksamhet och forskning. Schemaläggning är den del av planeringen som verkar på operationsnivå - här avgörs start- och stopptider för alla operationer samt vilken resurs de ska utföras på. Begreppet schemaläggning *eng. scheduling* används inte alltid konsekvent i litteraturen och det ska därför understrykas att betydelsen i denna rapport utgörs av den precis klargjorda. Schemaläggning är ett multiobjektivt optimeringsproblem som kan anses vara mycket komplext eftersom det ofta finns många villkor och parametrar att optimera. I detta fall är det ställtid, leveransdatum och dödtid men kan också vara väntetider, antal jobb i systemet, material etc. Redan med två resurser räknas problemet som ett NP-svårt problem, vilket innebär att det har en mycket komplex sökrymd och är mycket svårlöst [17, s. 517]. Med n operationer och m resurser blir antalet möjliga kombinationer att schemalägga $n!^m$ vilket växer oerhört snabbt och omöjliggör en systematisk genomgång av alla lösningar. En situation med 10 operationer och 2 resurser resulterar i ca 13 *biljoner* lösningar - vilket skulle ta 575 år för en processor som kan testa 1000 lösningar i sekunden. Därför används ofta heuristiska optimeringsmetoder såsom genetiska algoritmer eller simulerad kylning. Ett schemaläggningsproblem kan med fördel modelleras som ett handelseresandeproblem vilket möjliggör användning av litteratur och empiriska undersökningar inom optimering. Inom schemaläggning både i industrin och utanför är *ganttscheman* (se figur 1.1) ett mycket centralt begrepp och var den första vetenskapliga tekniken för planering och schemaläggning [17, s.152]. Ett ganttschema kan användas som ett komplement till en optimeringsalgoritm för att ge en grafisk representation av resultatet. En av fördelarna med dessa är just att användaren ges en överskådlig bild av hur schemaläggningen ser ut.



Figur 1.1: Ganttschema för två resurser

1.2 Syfte

Syftet med detta examensarbete är att implementera, testa och jämföra två olika algoritmer för att optimera ett speciellt fall av schemaläggning kallat *parallel machine model*. Optimeringen ska göras med avseende på ställtid, leveranstid och dödtid samt resultera i en jämn arbetsbeläggning mellan resurserna, på en skälig tid. Optimeringen förväntas inte hitta *den* optimala lösningen men en lösning som är tillräckligt bra. Algoritmerna ska simuleras för olika fall och utvärderas empiriskt. Resultatet ska påvisa hur algoritmerna fungerar vid växande komplexitet (fler operationer, fler resurser) och påvisa skiljaktigheter dem emellan samt, hur väl de presterar relativt varandra och på förhand uträknade basmått. Till detta ska en detaljerad utredning göras om vardera algoritms uppbyggnad, funktion och vilka parametrar som de påverkas av. Syftet är *inte* att göra någon djupgående teoretisk analys om hur och varför algoritmerna fungerar utan mer ur en praktisk synvinkel studera deras uppbyggnad och hur de kan implementeras för att lösa det givna problemet.

1.3 Metod

Inledningsvis ska en grundläggande litteraturstudie inom ämnesområdena optimering, optimeringsalgoritmer och schemaläggning göras. Därefter ska problemet definieras i termer av schemaläggning och baserat på problemets natur samt litteraturstudien ska två optimeringsalgoritmer väljas ut. Optimeringsalgoritmernas uppbyggnad och funktion ska studeras noggrant i relevant litteratur och därefter implementeras för lösa det givna problemet. Detta kommer att ske parallellt. Efter att båda algoritmerna har blivit implementerade och funktionstestade ska de utsättas för en eller flera fallstudier för att undersöka hur de presterar. Fallstudierna ska mynna ut i ett resultat som svarar mot arbetets syfte. Slutligen ska resultatet utvärderas och diskuteras.

1.4 Rapportstruktur

Inledningsvis följer en teoretisk bakgrund om schemaläggningsproblem med en detaljerad problembeskrivning av det givna problemet. I kapitel 2 modelleras problemet som handelsresandeproblem med en matematisk definition inkluderat. Därefter följer ett kapitel om optimeringsalgoritmer i vilket två stycken väljs ut, nämligen simulerad kylning och en genetisk algoritm. Dessa presenteras i de två följande kapitlen 4 och 5. Algoritmerna presenteras först med en bakgrundsbeskrivning som följs av en teknisk beskrivning vilken specifikt beskriver hur dessa är implementerade för just detta problem. Även ett avsnitt om fördelar/nackdelar och svårigheter med algoritmerna i fråga är inkluderat. Kapitel 6 innehåller fyra olika fallstudier i form av simuleringar med tillhörande beskrivning och resultat. Slutligen följer en diskussion i kapitel 7.

Kapitel 2

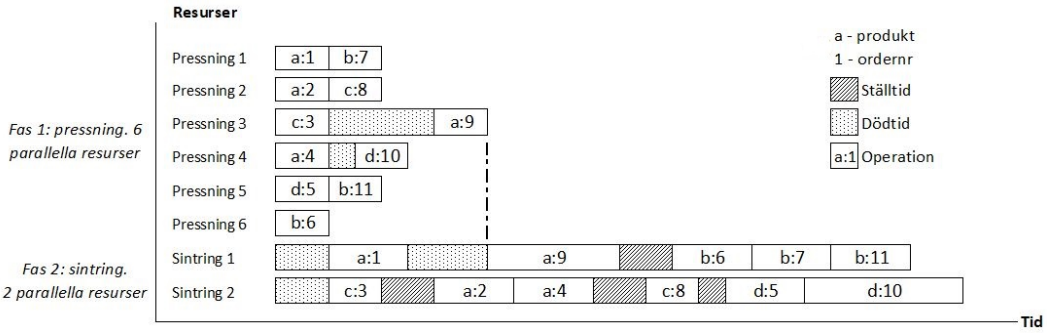
TEORETISK BAKGRUND

2.1 Problembeskrivning

Schemaläggning inom produktion är ett brett begrepp och det finns många olika fall beroende på hur produktionen i fabriken är upplagd. Den tillämpas på såväl diskret som kontinuerlig tillverkning samt mindre och större fabriker. Pinedo redogör för ett 20-tal olika fall i sin bok *Scheduling - Theory, Algorithms, and Systems* [18]. Fallet det här arbetet fokuserar på är identifierat och definierat utifrån hans bok och benämns som *parallel machine model*, vilket är en uppsättning med ett flertal likartade maskiner (resurser) som arbetar parallellt och operationerna som ska utföras kan allokeras fritt mellan dessa maskiner. Detta är en vanlig uppsättning i industrin [18, s.111]. Modellen som behandlas här är deterministisk vilket innebär att den bortser från alla slumpmässiga faktorer såsom haveri och avvikande processtider, vilka tas med i en stokastisk modell. Figur 1.1 kan utgöra ett exempel på två parallella resurser, förutsatt att resurserna är likartade.

Ur ett större perspektiv är modellen en del av en *flexible flow shop* (FFc), vilken är modellen av hela fabriken där det kan finnas flera stationer med olika uppsättningar av parallella resurser och produkterna följer ett flöde genom fabriken i en bestämd sekvens - till exempel ett löpande band. Vid varje station allokeras produkten till en resurs där operationen ska utföras. En FFc med två stationer (faser) demonstreras i figur 2.1. Figuren föreställer ett schema för en metallurgisk produktion med sex parallella pulverpressar och två parallella sintringsugnar. Sintringsugnarna utgör en flaskhals vilket gör dem till ett bra mål för optimering.

Mycket av litteraturen som finns tillgänglig om schemaläggning fokuserar på hela fabriken där optimeringsmetoder syftar till att schemalägga hela ordrar. Så är inte fallet med detta arbete, vilket istället riktar sig mot delar av produktionen där parallella resurser förekommer, till exempel *fas 2* i figur 2.1. Speciellt är fallet tillämpbart på flaskhalsar som kan optimeras och bli utgångspunkten för resten av schemaläggningen.



Figur 2.1: Ganttschema för "flexible flow shop"

Problemet består sammanfattningsvis av att ett antal operationer ska schemaläggas på ett antal resurser på ett optimalt sätt. Resurserna antas vara identiskt lika och varje resurs kan utföra alla operationer i vilken ordning som helst. En fullständig operation måste genomföras på en och samma resurs och kan alltså inte avbrytas och flyttas. Varje operation har information om bland annat:

- Dess processtid *d* eller den tid den kräver av resursen för att slutföras. Denna är oberoende av resurs.
- Ordern den tillhör och därmed vilken operation den (eventuellt) måste föregås av samt vilken som kommer efter.

Med schemaläggningen önskas följande mål uppnås, där a-c utgör mål för optimeringen och d är ett bivillkor. Punkt e är mer att betrakta som ett önskemål.

a. Minimera ställtiden.
b. Minimera leveransförseningar.
c. Minimera att någon resurs svälter (dödtid).
d. En jämn arbetsbeläggning på resurserna.
e. En skälig beräkningstid

a. Ställtid är sekvensberoende och uppstår när ett produktbyte sker. I praktiken kan det vara till exempel byte av färg i en färgspruta eller omställning av dimensioner i ett sågverk. Det kan förekomma flera olika byten som ger upphov till ställtid. I sågverksexemplet kan till exempel byte av trädslag från ett mjukt till ett hårt kräva byte av klinga som tar en viss tid, medan att ställa om dimensioner för samma trädslag tar en annan tid. Upprepade operationer av samma produkt innebär således att ingen ställtid uppkommer. Ställtiden illustreras i figur 2.1.

b. Varje operation är knuten till en order som har en bestämd leveranstid. Det är önskvärt att schemaläggningen är utförd så att inga leveransförseningar uppstår. På engelska benämns förseningar som *tardiness*.

c. En operation kan behöva föregås av en annan operation på en annan station. Till exempel måste ett hål borras innan det kan gängas, grovsvarvning måste ske före finsvarvning och slipning innan målning. Optimeringen ska ta hänsyn till föregående operationer, vilket också kan beskrivas som materialtillgång. I väntan på att den föregående operationen ska slutföras kan det uppstå en *dödtid* vilken också kan vara kostsam. På engelska benämns detta villkor som *precedence constraint* vilket kan översättas som startvillkor och dödtid benämns som *idle time*. Notera att föregående operationer inte alltid är aktuella - de operationer som är föremål för optimeringen kan också vara först i kedjan. Dödtid och startvillkor illustreras i figur 2.1, se speciellt order “a:9”.

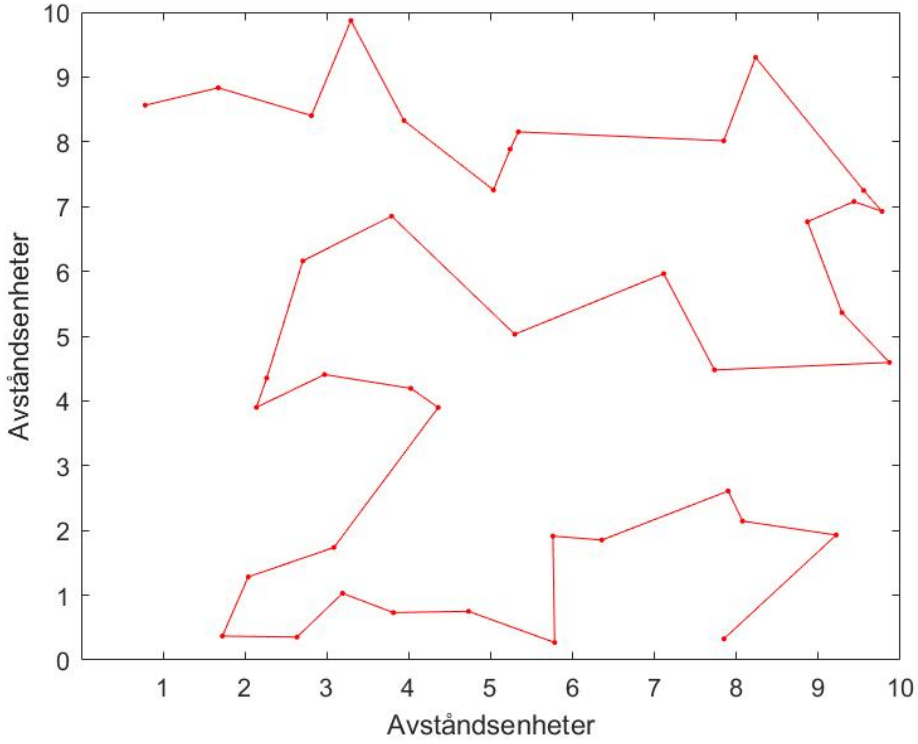
d. En jämn beläggning är eftersträvansvärt. Genom en jämn beläggning minskar också den längsta processtiden, det som på engelska benämns som *makespan*.

e. En skälig exekveringstid beror på omständigheterna runtomkring. En algoritm som tar mer än 1 minut att exekvera anses vara långsam. Om programmet kan köras i bakgrunden kan det tillåtas ta längre tid men om en produktionsplanerare väntar på att algoritmen ska bli klar kan 30 sekunder upplevas som lång tid. Ofta avvägs exekveringstiden mot optimeringens resultat.

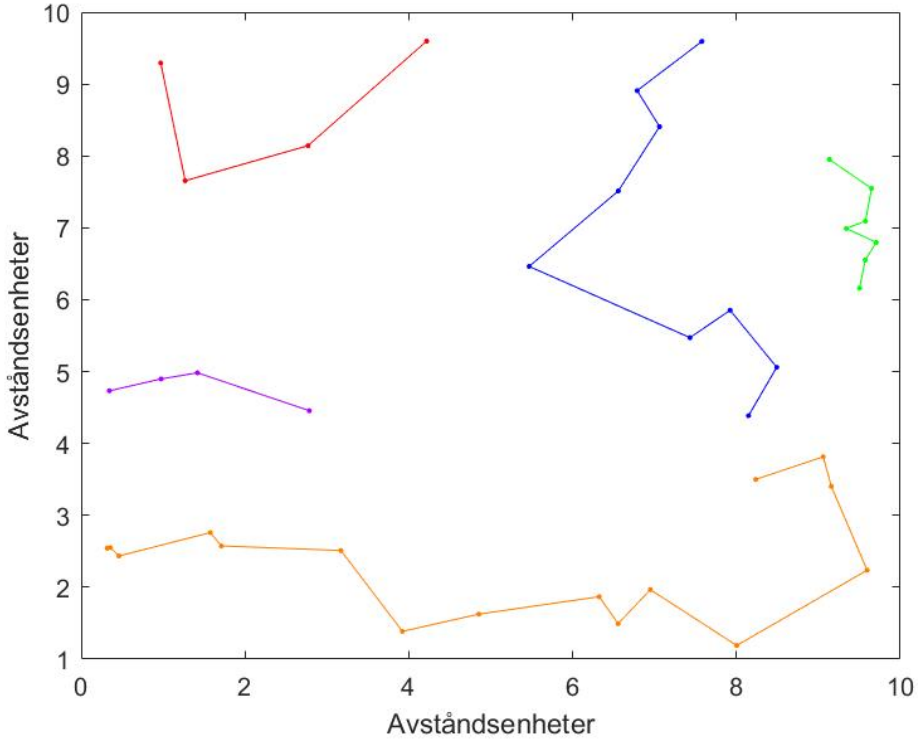
2.2 Modellering med handelsresandeproblemet

Schemaläggningsproblem i FFc är mycket vanliga att modellera som handelsresandeproblem *eng. Traveling salesman problem* (TSP) [12, s.254]. TSP är ett mycket känt matematiskt optimeringsproblem [8] som går ut på att hitta den bästa möjliga sekvens (det kortaste avståndet) för en säljare att resa mellan ett antal n städer. Ett klassiskt TSP har $n!$ möjliga lösningar vilket redan för $n = 10$ är svårt att lösa.

Handelsresandeproblemet med flera säljare *eng. multiple Traveling Salesman problem* (mTSP) är en generalisering av TSP vilket kan användas för att modellera problem där både sekvens och allokering är av betydelse [16]. I mTSP finns istället *flera* säljare m som ska besöka ett antal städer. Precis som för TSP kan problemet definieras på olika sätt; till exempel kan en bestämd start- och slutpunkt för alla säljare användas, s.k. *depot*. I detta fall används godtyckliga och olika start-och slutpunkter och problemet benämns då som öppet (mTSPo). Varje stad besöks endast *en* gång av *en* säljare och det summerade reseavståndet för alla säljare ska minimeras. Villkoren åskådliggörs i figur 2.2b för mTSP samt i figur 2.2a för TSP.



(a) TSPo



(b) mTSPo med fem säljare

Figur 2.2: Lösningar (rutter) till TSP och mTSP

Eftersom mTSP har mer än en säljare blir tillämpningarna också fler enligt Király och Abonyi [11]. Ett exempel de tar upp i sin artikel är planeringen av skolbussar där det totala avståndet ska minimeras samtidigt som varje buss har ett begränsat antal platser och ett bestämt tidsfönster att hämta barnen inom.

Genom att modellera schemaläggningsproblemet som ett mTSP kan ett flertal olika väldokumenterade algoritmer användas för att lösa det. Király m.fl. har till exempel skapat en genetisk algoritm för att lösa ett problem som de beskriver som ett mTSP där produktion med ett flertal produktionsanläggningar ska schemaläggas för att minimera spill och därmed negativ miljöpåverkan [12]. I sitt resultat påvisar de att liknande planeringsproblem med fördel kan beskrivas som mTSP. Moon m.fl. tar även de fram en genetisk algoritm för att lösa ett liknande problem med mål att utvärdera allokering mellan olika resurser samt sekvenser för att minimera ställ- och transporttid [16].

I det här fallet modelleras säljarna som resurser och städerna som operationer. Avståndet mellan städerna (operationerna) svarar mot en kostnad som uppstår till följd av specifikationerna *a-c* i avsnitt 2.1. Varje stad har en viss besökstid och den totala besökstiden ska fördelas jämnt mellan säljarna vilket svarar mot en jämn arbetsbelastning mellan resurserna (jämför specifikation *d* i avsnitt 2.1. Ett förtydligande av begreppen som används i denna modell redogörs för i tabell 2.1.

Produktionsplanering		mTSP		Matematisk beskrivning
Operation	↔	Stad	↔	Nod
Resurs	↔	Säljare		-
Kostnad	↔	Distans	↔	Båge
Processtid	↔	Besökstid		-

Tabell 2.1: Begrepp

Modellering med mTSP kan sammanfattas enligt följande, där problemdefinitionen beskrivs jämte varje punkt inom parentes.

- Ett antal städer ska besökas av en eller flera säljare (ett antal operationer ska utföras av en eller flera resurser).
- Alla städer måste besökas en och endast en gång av *en* säljare (en operation utförs endast en gång på en resurs).
- Den totala besökstiden ska fördelas jämnt mellan säljarna. (den totala processtiden ska fördelas jämnt mellan resurserna. En jämn arbetsbeläggning eftersträvas).
- Säljarna börjar och slutar sin resa i olika, godtyckliga punkter (en operation kan bara utföras en gång, därför måste de vara olika).
- Den totala distansen ska minimeras (distansen motsvarar kostnaden för ställtid, dödtid och leveransförseningar).

2.3 Matematisk formulering

mTSP kan beskrivas som en graf enligt följande: $G = (N, B)$. Tillhörande denna finns en mängd noder N där antalet $|N| = n$ samt en mängd bågar B. Noderna representerar städer vilka ska besökas av m säljare och bågarna är vägarna som förbinder städerna. Till varje båge finns en kostnad C = $c_{i,j}$ där index i och j betecknar två godtyckliga noder. Observera att $c_{i,j} \neq c_{j,i}$, d.v.s. kostnaden av att resa från stad i till j är inte nödvändigtvis samma som att resa från j till i och kostnadsmatrisen är därmed asymmetrisk. Kostnaden c_{ij} kan beskrivas som:

$$c_{ij} = ST_{ij} \cdot c_1 + LF_{ij} \cdot c_2 + DT_{ij} \cdot c_3 \tag{2.1}$$

där ST är ställtid, LF är leveransförseningar och DT är dödtid räknat i timmar och c_{1-3} är deras motsvarande kostnad per timme.

De binära variablerna x_{ijk} och y_{ik} införs så att:

$$x_{ijk} = \begin{cases} 1 & \text{om säljare } k \text{ använder bågen från nod } i \text{ till } j \\ 0 & \text{annars} \end{cases} \tag{2.2}$$

$$y_{ik} = \begin{cases} 1 & \text{om säljare } k \text{ besöker noden } i \\ 0 & \text{annars} \end{cases} \tag{2.3}$$

Dessutom införs tidsvariablerna t_m och t_k, där d_i är besökstiden i staden i så att:

$$t_m = \frac{1}{m} \cdot \sum_{i}^{n} \sum_{k}^{m} d_i \cdot y_{ik} \tag{2.4}$$

$$t_k = \sum_{i}^{n} d_i \cdot y_{ik} \quad \forall k \tag{2.5}$$

Härefter kan problemet formuleras som ett heltalsproblem.

$$\text{minimera} \quad \sum_{i}^{n} \sum_{j}^{n} \sum_{k}^{m} c_{ij} \cdot x_{ijk} \tag{2.6}$$

$$\text{då} \quad \sum_{i}^{n} \sum_{j}^{n} \sum_{k}^{m} x_{ijk} = n - k \tag{2.7}$$

$$\sum_{j}^{n} \sum_{k}^{m} (x_{ijk} + x_{jik}) \leq 2 \quad \forall i \tag{2.8}$$

$$\sum_{k}^{m} y_{ik} = 1 \quad \forall i \tag{2.9}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \quad \forall S \subset L \subset N \tag{2.10}$$

$$\frac{|t_m - t_k|}{t_m} < 0.2 \quad \forall k \tag{2.11}$$

$$x_{ijk} \in \{0, 1\}, \forall (i, j) \tag{2.12}$$

Tidsvariabeln t_m i ekvation 2.4 är den summerade besökstiden för alla säljare dividerat med antalet säljare och utgör därmed en exakt fördelning av besökstiden säljarna emellan, medan t_k i ekvation 2.5 är den summerade besökstiden en enskild säljare upplever under sin rutt.

Ekvation 2.6 är målfunktionen som summerar den totala resekostnaden. Eftersom problemet ska formuleras som ett öppet mTSP (mTSPo) ska lösningen bestå av ett bestämt antal bågar = $n - m$, vilket påvisas i bivillkor 2.7. Alla noder tillåts att ha en valens på maximalt två, vilket anges i bivillkor 2.8 (valens är antalet bågar som är anslutna till en viss nod). Detta innebär att varje nod kan ha antingen en valens - nämligen frånresande *eller* tillresande vilket motsvarar start- resp. slutnod, alternativt två valenser - en s.k. mellanlandning med både tillresande *och* frånresande. Eftersom en säljare inte får starta sin rutt i en nod där en annan säljare slutade sin införs bivillkor 2.9 som förhindrar att en stad besöks av mer än en säljare. Bivillkor 2.10 förhindrar att det inte förekommer några cykler (subturer) för någon säljare genom att ställa krav på en godtycklig delmängd $S \subset L$, där delmängden L motsvarar alla noder som blivit besökta av säljare k. Ekvation 2.11 anger att besökstiden för en enskild säljare inte får avvika mer än 20% från en exakt uppdelning. Sammanfattningsvis utgör ekvationerna 2.7 - 2.12 bivillkoren för optimeringen och alla utom de två sista åskådliggörs i figur 2.2b.

m	antal säljare
n	antal noder
i, j	index för noder. $i, j = 0, 1, .., n$
k	index för säljare. $k = 1, 2, .., m$
x_{ijk}	binär variabel, för def. se ekv. 2.2
y_{ik}	binär variabel, för def. se ekv. 2.3
d_i	besökstid i stad i
t_m	total besökstid fördelat på säljare. se ekv. 2.4
t_k	summerad besökstid för en säljare. se ekv. 2.5
N	mängd med alla noder
L	mängd med alla noder besökta av k
ST	ställtid
LF	leveransförseningar
DT	dötid

Tabell 2.2: Variabler

Kapitel 3

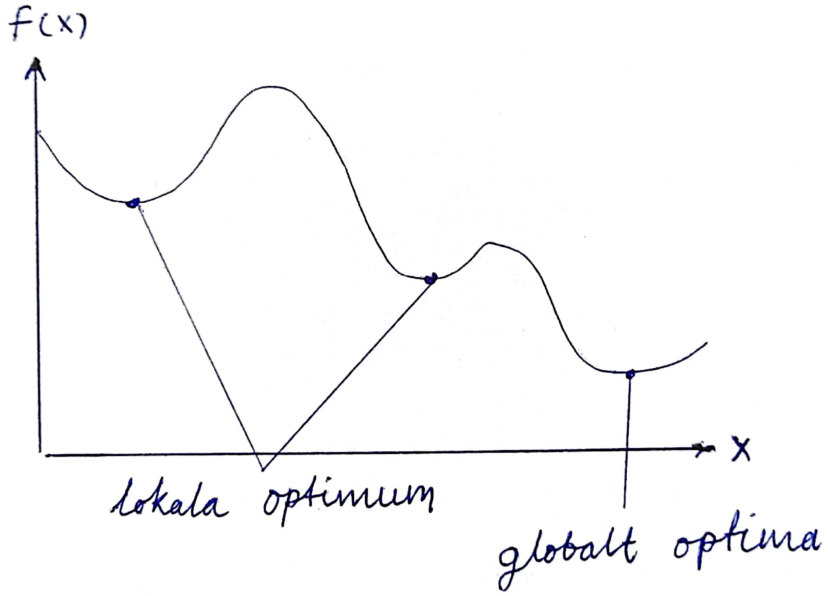
OPTIMERINGSALGORITMER

3.1 Val av algoritmer

I inledningen nämndes att redan för två resurser är schemaläggningsproblem ofta NP-svåra och för denna kategori av problem används ofta *heuristiska* optimeringsmetoder [8, s. 411]. Dessa metoder hittar nära-optimala eller tillräckligt bra lösningar till problem vars komplexitet gör att de inte kan lösas inom rimlig tid, som till exempel TSP [6]. Vanligtvis har de någon (ofta enkel) strategi för att söka sig till tillfredställande lösningar (utan att söka igenom alla) men de garanterar inte att *den* optimala lösningen hittas [3, s.421].

Både Holmberg [8] och Hromcovič [3] beskriver utförligt i sina böcker simulerad kylning och genetiska algoritmer. Hromcovič beskriver dessa som kända och robusta [3, s. 432]. Inom schemaläggning är simulerad kylning (SA) vida använd både inom akademin och industrin med goda resultat [19, s.457]. Genetiska algoritmer (GA) är även de vanligt förekommande inom schemaläggningsproblem, se till exempel [16], [11] och [15]. De har dessutom fördelen att de kan implementeras och användas utan djupare analys av problemets struktur [19, s.461]. Simulerad kylning och genetiska algoritmer tycks vara de bäst dokumenterade och mest använda och väljs därför som mål för denna utredning. Andra förekommande algoritmer som har noterats är till exempel ACO - ant colony optimization, PSO - particle swarm optimization och tabusökning.

Både simulerad kylning och genetiska algoritmer är skapade utifrån naturliga processer och båda är delvis beroende av slump [3]. Detta leder i förlängningen till att algoritmerna kan (kommer att) ge olika resultat vid upprepade försök. Algoritmerna är uppbyggda helt olika, den genetiska algoritmen utgår ifrån en population med individer medan simulerad kylning utgår från en enskild ledarindivid. När det gäller heuristiska algoritmer i allmänhet är ett problem för tidig konvergens - att ett lokalt optimum hittas istället för globalt (se fig 3.1). Båda algoritmerna utgår ifrån individer, vars definitioner presenteras nedan.



Figur 3.1: Lokala och globala optima

3.2 Algoritmernas grundläggande byggnadssten: en individ

En individ består av en genotyp, en fenotyp och ett fitnessvärde vilka beskrivs nedan och illustreras i figur 3.2.

Genotyp

En individ representerar en lösning till problemet och består av en *genotyp* som är uppbyggd av i detta fall en men ibland flera kromosomer. Varje kromosom i sin tur består av ett antal gener. I den här implementeringen är genotypen representerad av en vektor. Varje gen motsvarar en operation och genotypen innehåller alla operationer som ska optimeras samt information om vilken sekvens dessa ska utföras i. En genotyp med 15 gener illustreras i figur 3.2. Det är på genotypen alla operatorer som används både i SA och GA verkar.

Fenotyp

En genotyp kan avkodas till en *fenotyp* vilket är den fysiska skepnaden av lösningen, det vill säga en exakt redogörelse för vilken sekvens operationerna ska utföras i och vilken resurs de ska belasta och kan därmed ses som en fullständig lösning. I avkodningen görs uppdelningen av operationer mellan resurserna i enlighet med bivillkor 2.11. Observera att antalet operationer per resurs därför kan skilja.

Fitness

Till varje fenotyp kan ett fitnessvärde beräknas, vilket är det värde som kommer att kännetecknа individen och avgöra hur bra den är. I detta fall är fenotypens fitness proportionell mot den totala kostnaden som lösningen ger upphov till och beror således på sekvensen operationerna utförs i. Fitness beräknas med fitnessfunktionen (*eng. fitness function*) vilken svarar mot målfunktionen, ekvation 2.6. Fitnessfunktionen kan vara exakt samma som målfunktionen men är ofta manipulerad av olika anledningar (se mer nedan). Fitnessfunktionen definieras för en individ υ som $f(\upsilon) = f_1(\upsilon) + f_2(\upsilon) + ... + f_n(\upsilon)$ där $f_i(\upsilon)$ är funktionen för optimeringsvariabel i. Fallet som utreds här har tre optimeringsvariabler och således tre funktioner vilka är de för ställtid, leveransförseningar och dödtid (jämför ekvation 2.1 samt specifikationerna $a - c$ i avsnitt 2).

Som nämndes i det föregående stycket manipuleras ofta fitnessfunktionen för att få algoritmen att bete sig på ett önskvärt sätt, vilket kallas *skalning*. De olika variablerna som ska optimeras kan vara av olika storleksordning och av olika betydelse - till exempel kan en minskning från 10 till 5 timmar ställtid betyda mycket medan om en produkts försening minskar från 105 till 100 timmar inte spelar någon avsevärd roll. Därför är det ofta aktuellt att införa någon form av skalning [4, s.66]. Detta problem undgås i den genetiska algoritmen (se avsnitt 4.2: Urval) men är aktuellt för simulerad kylning.

Individ υ med 15 gener

Genotyp med 1 kromosom à 15 gener

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

Fenotyp efter avkodning. Resurser a, b och c

a	1	2	3	4	5	
b	6	7	8	9	10	11
c	12	13	14	15		

Fitness beräknat på 3 variabler

$$f(\upsilon) = f_1(\upsilon) + f_2(\upsilon) + f_3(\upsilon)$$

Figur 3.2: En individ

3.3 Smarta startpunkter

Både den genetiska algoritmen och simulerad kylning använder *smarta startpunkter*, vilket innebär att de börjar i en bestämd och inte slumpmässig punkt. Detta kan hjälpa algoritmen att snabbare konvergera mot optimum [3, s. 453]. Det kan ses som en metod där algoritmerna får hjälp att hitta optimum. En annan praktiskt fördel med detta är att användaren kan göra manuella ändringar i schemat vilka algoritmen sedan tar hänsyn till. Det innebär också att om algoritmerna körs flera gånger efter varandra kan resultatet förbättras ytterligare eftersom lösningen från den förra exekveringen används som startpunkt i nästa.

Kapitel 4

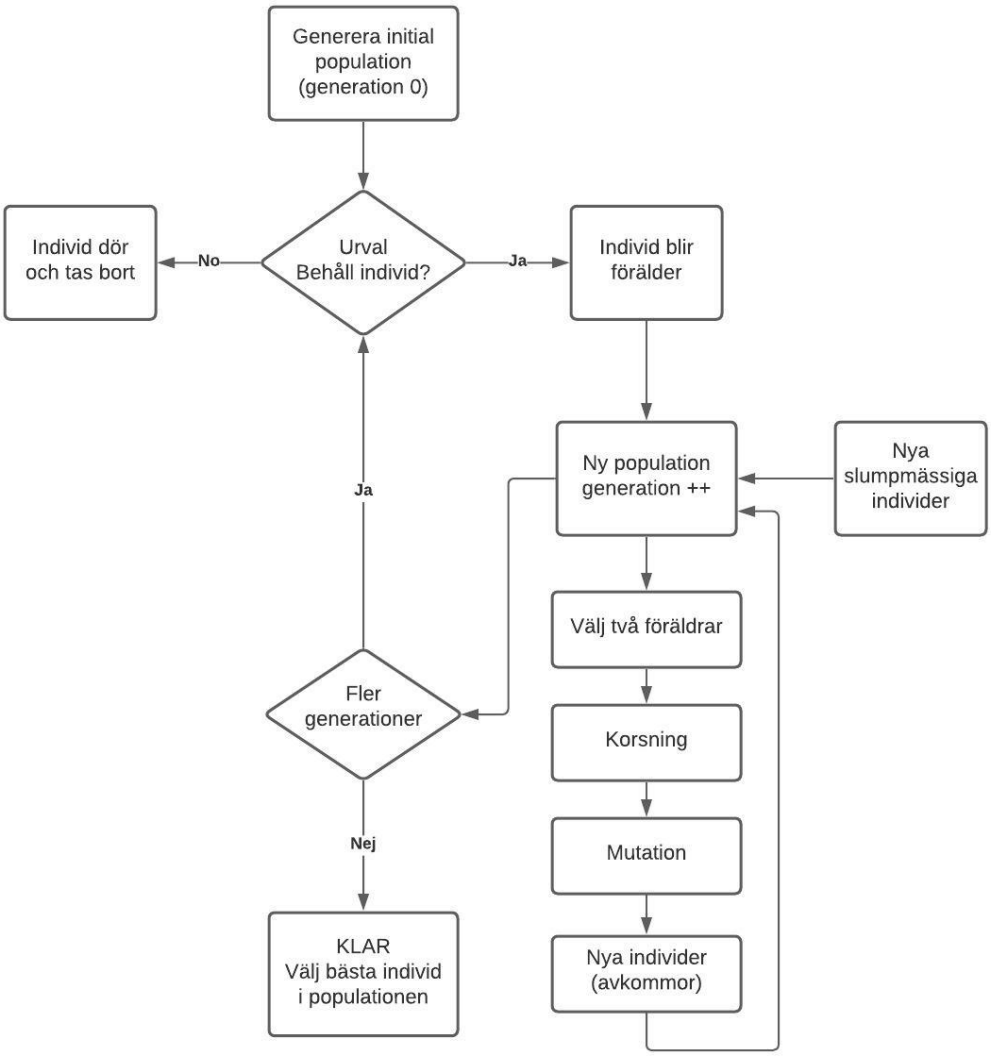
GENETISK ALGORITM

4.1 Om genetiska algoritmer

Genetiska algoritmer tillhör gruppen evolutionära algoritmer och är uppbyggda på principen om hur en population förändras genom evolution. Algoritmens grundläggande funktion bygger alltså på Darwins evolutionsteori och det naturliga urvalet samt Gregor Mendels upptäckter om hur olika egenskaper ärvs genom mixade anlag från föräldrarna (Neo-Darwinism). Algoritmen arbetar med en ständig population av individer där somliga dör och andra blir föräldrar som får avkommor, vilka skapas med ett antal genetiska operatorer såsom korsning och mutation. De genetiska algoritmerna började användas på 1960- och 1970-talet. J.H. Holland presenterade en djup teoretisk analys på flera tillämpningsområden bland annat ekonomi, spelteori och psykologi i sin bok *Adaption in natural and artificial systems* [7], vilket var den första systematiska genomgången av algoritmens beteende. Genetiska algoritmer har en uppsjö med tillämpningsområden och i det närmaste otaliga utföranden. Några andra exempel på användningsområden är optimering av fackverksutformning och pipelinelokalisering [5, s. 125]. Genetiska algoritmer är mer avancerade än simulerad kylning, implementeringen är mer tidskrävande men dess många parametrar ger större möjligheter att påverka resultatet, även om individrepresentationen är fix [3, s. 452].

Algoritmen arbetar med en population av bestämd storlek med en andel föräldrar och en andel andra individer. Populationen utvecklas under ett antal generationer, som närmast kan ses som iterationer. I varje generation genomförs en process bestående av tre faser - urval, korsning och mutation. I urvalet beslutas vilka av individerna i populationen som ska bli föräldrar och skapa nya avkommor till nästa generation samt vilka individer som ska dö och tas bort. Det är individernas fitness som avgör dess öde. Avkommorna skapas från dess föräldrar med *genetiska operatorer*, i detta fall *korsning* och *mutation*. När alla generationer har genomgåtts kvarstår en population där den bästa individen förhoppningsvis är motsvarar en tillfredställande lösning.

I algoritmen används en mängd med parametrar vilka redogörs för i tabell 4.1 och ett flödesschema för hela algoritmen återges i figur 4.1. För att få en bättre förståelse av flödesschemat rekommenderas att läsa avsnitt 4.2 parallellt.



Figur 4.1: Genetisk algoritm

G_t	antal generationer
R_t	storlek av populationen
P_t	antal föräldrar
C_t	antal avkommor
Q_t	övriga individer
υ	individ
δ_υ	trängselavstånd
$c_r \in \{0, 1\}$	korsningsfaktor
$m_r \in \{0, 1\}$	mutationsfaktor
$q \in \{0, 1\}$	slumpmässigt genererat tal
$f(\upsilon)$	fitnessfunktion

Tabell 4.1: Parametrar i en genetisk algoritm

4.2 Funktion och implementation

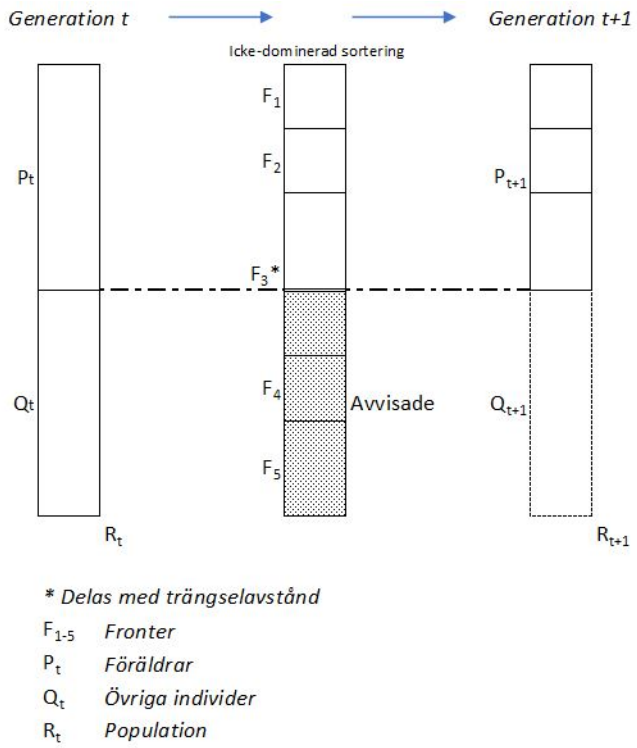
Fas 1: Urval med NSGA-II

Ett problem som kan uppstå i urvalet är dominans - en eller flera individer med mycket bra fitnessvärde dominerar och algoritmen konvergerar för fort. Detta är ett problem om urvalet sker enbart med avseende på fitness. Ett sätt att undvika detta är att använda en NSGA-algoritm, som istället för att evaluera individerna efter deras fitness, jämför variablerna var för sig. Mer om detta följer.

Urvalet är processen i ett generationsskifte där det avgörs vilka som ska bli föräldrar och leva vidare i nästa generation samt vilka som ska avvisas och tas bort. Detta är en komplicerad process och en rad nya begrepp kommer att introduceras. Urvalet sker med en algoritm som kallas NSGA-II (Elitist, Non Dominated Sorting Genetic Algorithm) vilken presenterades av Kalyanmoy m.fl [10]. Algoritmen gör urvalet inte baserat på fitness utan jämför värdet på varje variabel för sig. Detta är en stor fördel eftersom någon skalning inte behöver göras. Evalueringen görs med Paretos teori om optimalitet för flera variabler, om vilken kan läsas i till exempel [9].

Ett viktigt begrepp i sorteringen är dominans. En individ sägs dominera en annan då den är minst lika bra sett till *alla* optimeringsvariabler samt definitivt bättre sett till minst en variabel. En individ som är dominerad kallas recessiv. Matematiskt kan detta formuleras enligt följande där f_i är funktionen för en optimeringsvariabel i och υ_1 dominerar υ_2.

$$f_i(\upsilon_1) \leq f_i(\upsilon_2) \quad \forall i \in \{1, 2, ...n\}$$
$$f_j(\upsilon_1) < f_j(\upsilon_2) \quad \exists j \in \{1, 2, ...n\}$$



Figur 4.2: Icke-dominerande sortering med fem fronter

Algoritmen utgår ifrån en population R_t som består av en andel föräldrar P_t och en andel övriga individer Q_t, se figur 4.2. Det första den gör är en *icke-dominerande* sortering, genom att för varje variabelfunktion f_i jämföra vardera individ med alla andra i populationen och kartlägga vilka individer som dominerar andra. Med denna information skapas ett antal paretofronter med paretooptimala lösningar, se figur 4.3a där tre stycken fronter med två variabelfunktioner illustreras. Lösningarna i front 1 dominerar alla lösningar i front 2 och 3, lösningarna i front 2 dominerar de i front 3 men domineras av front 1. Front 3 består endast av recessiva individer. De individer som ligger på samma front är paretooptimala - ingen kan avgöras vara definitivt bättre än någon annan. En förflyttning mellan individerna på en paretofront kommer leda till en förbättring av någon variabel men också en försämring av någon annan. När individerna är indelade i fronter tilldelas de en rang efter den front de tillhör och sorteras sedan efter deras rang. Eftersom variablerna i detta fall ska minimeras är fronterna som ligger närmast origo bättre och erhåller en högre rang. De individer som ligger i de sämsta fronterna avvisas och tas bort. Ofta går denna indelning inte jämnt ut vilket kräver ytterligare en beslutsvariabel - trängselavstånd δ (*eng. crowding distance*).

Trängselavståndet för en individ i definieras enligt följande:

$$\delta_\upsilon = \frac{d_1}{f_1^{max} - f_1^{min}} + \frac{d_2}{f_2^{max} - f_2^{min}} + ... + \frac{d_n}{f_n^{max} - f_n^{min}}$$

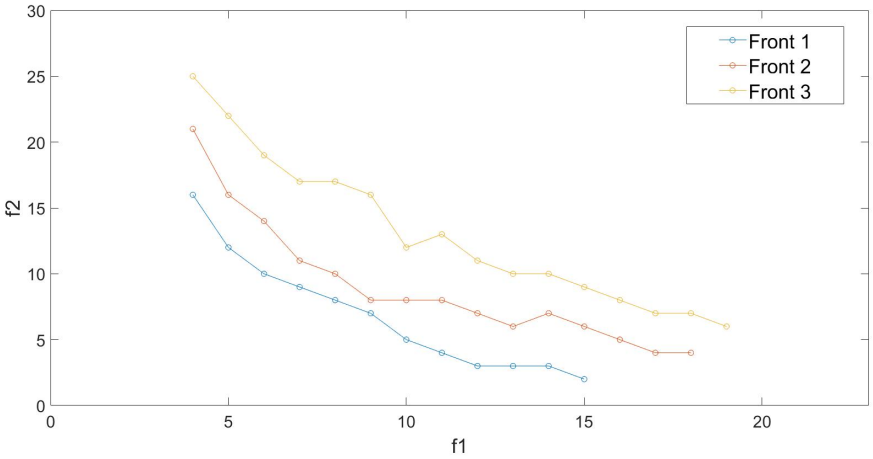
där

$$d_i = f_i(\upsilon + 1) - f_i(\upsilon - 1) \quad \forall i \in \{1, 2, ...n\}$$

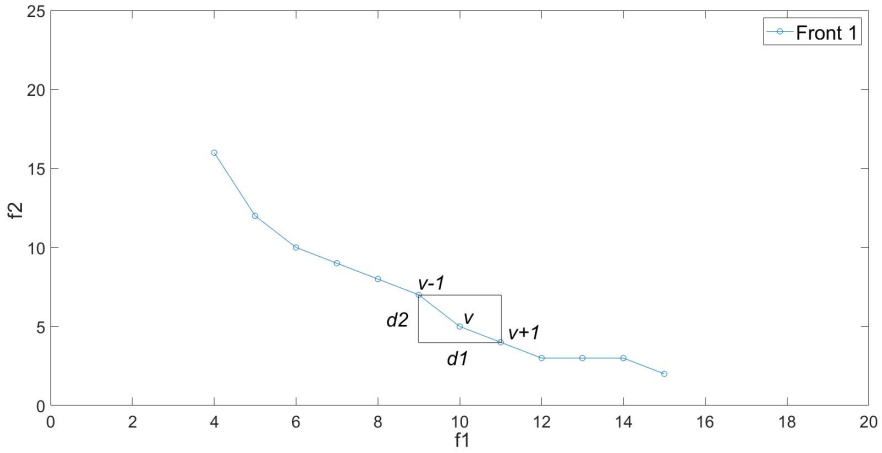
Trängselavståndet är alltså ett avstånd beräknat mellan de närmsta punkterna i förhållande till det totala avståndet, summerat för alla variabler. Avståndet illustreras i figur 4.3b. Syftet med denna beslutsvariabel är att utesluta individer som ligger nära varandra och på så sätt använda en större del av sökrymden.

Om en front måste delas upp görs detta med avseende på trängselavstånd - individerna med större avstånd blir de som förs över till nästa generation (jämför F_3 i figur 4.2). När alla föräldrar är utvalda är nästa steg att para ihop dessa. Detta görs med en tävling (*eng. tournament*). I tävlingen väljs två (ibland fler) slumpmässiga föräldrar ut ur föräldrapopulationen P_t. Dessa tävlar mot varandra och den med högst rang vinner. Om de har samma rang vinner den med högst trängselavstånd. I urvalet används också ett begrepp som kallas *elitism* vilket innebär att den bästa individen får en garanterad plats i nästa generation utan att korsas eller muteras för att säkerställa att den bästa lösningen inte försvinner i nästa generation. Den bästa lösningen är den med högst rang och högst trängselavstånd.

När alla föräldrar är ihopparade initieras fas 2 - korsning.



(a) Paretofronter med 2 variabler



(b) Trängselavstånd δ_i för i

Figur 4.3: Urval med paretooptimalitet för två variabler

Fas 2: Korsning

Två föräldrar paras genom en s.k. korsning (*eng. crossover*) där en eller två nya individer (avkommor) skapas genom att ta gener från båda föräldrar. Denna process illustreras i figur 4.4 där samma individ som i föregående exempel med 15 gener används. I det här fallet används en *one-point crossover*, vilket innebär att endast en korsningspunkt används. Korsningspunktens position i kromosomen avgörs av korsningsfaktorn k_r och ett för varje gen genererat slumpmässigt tal $q \in \{0, 1\}$. En kromosom itereras igenom och för varje gen ställs villkoret $k_r < q$. När detta först blir sant är korsningspunkten nådd. Operatorn kopierar först de gener som finns i den första föräldern fram till korsningspunkten och lägger dessa i den första avkomman. Därefter fyller den på med de resterande gener som saknas från den andra föräldern. Den andra avkomman blir inversen av den första. Om korsningsfaktorn $c_r = 1$ eller 0 innebär det således att avkommorna blir kloner av sina föräldrar. Korsningen ger upphov till både omallokering och sekvensändring av operationerna, vilka kan var större eller mindre beroende på var korsningspunkten infinner sig.

förälder 1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
avkomma 1	1	2	3	4	5	6	7	8	**15**	**14**	**13**	**12**	**11**	**10**	**9**
(avkomma 2)	**15**	**14**	**13**	**12**	**11**	**10**	9	10	11	12	13	14	15		
förälder 2	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**

|| = *korsningspunkt*

Figur 4.4: Korsningsoperator

Fas 3: Mutation

När avkommorna är skapade muteras dessa genom att slumpmässiga gener byter plats. Mutationen genomförs för att hålla populationen diversifierad genom att skapa nya genotyper som inte tidigare har blivit evaluerade, vilket gör att algoritmen inte fastnar i lokala optima [7, s. 111]. Den mutationsoperatorn som används i denna implementering är en mixad variant som slumpmässigt väljer mellan tre olika mutationsoperatorer; *swap, insert* eller *revert*, vilka illustreras i figur 4.5. Swap innebär att två slumpmässiga gener väljs ut och byter plats (jämför 4 och 9). I insert plockas en gen bort och sätts in på en annan plats vilket gör att alla efterföljande operationer flyttar ett steg (jämför 13 med efterföljande). I revert väljs två stycken gener ut slumpmässigt, och alla gener däremellan byter plats genom att bli inverterade (jämför 2 och 7 och allt däremellan). Valet av gen(er) beror på mutationsfaktorn m_r. En gen väljs som offer för mutation då $q < m_r$. En mutation kommer alltid att ge upphov till sekvensändringar och ofta även omallokeringar. I alla de fall som exemplifieras i figuren sker resursbyte. När mutationerna är klara läggs de nya individerna till i populationen, se Q_{t+1} i figur 4.2. Populationen kan också kompletteras med nya, slumpmässigt skapade individer för att säkerställa en diversifierad population.

swap	*före*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	efter	1	2	3	**9**	5	6	7	8	**4**	10	11	12	13	14	15
insert	*före*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	efter	1	2	3	**13**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	14	15
revert	*före*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	efter	1	**7**	**6**	**5**	**4**	**3**	**2**	8	9	10	11	12	13	14	15

| = *resursindelning*

Figur 4.5: Mutationsoperatorer

4.3 Mer om den genetiska algoritmen

Det finns otaliga sätt att implementera en genetisk algoritm. Det som kan skilja sig mycket är bland annat korsnings- och mutationsoperatorerna samt individkodningen. Király och Abonyi presenterar till exempel en ny multi-kromosom [11]. Naseri och Afshari presenterar ett flertal nya sätt att både korsa och mutera i sin artikel *"A hybrid genetic algorithm for integrated process planning and scheduling problem with precedence constraints"* [2].

Kunapareddy m.fl. presenterar en genetisk algoritm med korsnings- och mutationsoperatorer som först angriper allokeringen och därefter sekvensen vilket innebär att de använder två korsningsoperatorer och två mutationsoperatorer i varje parning. Denna implementering visar sig vara effektiv när den jämförs med en mer traditionell implementation av algoritmen och även SA och andra vanligt förekommande algoritmer inom området [15].

En av fördelarna med GA jämfört med SA är att algoritmen tar fram en hel population med bra individer - det vill säga en mängd med bra lösningar, vilket i vissa lägen kan vara fördelaktigt [3]. Till exempel kan den innehålla lösningar där optimeringsvariablerna har blivit olika bra optimerade. Korsnings- och mutationsoperatorn gör också att algoritmen hela tiden jobbar i *hela sökrymden* och tittar inte endast på närliggande punkter (grannar).

Den genetiska algoritmen innehåller ytterligare parametrar att ställa in vilket ger den större flexibilitet. Några betydande parametrar att justera är:

- Storlek av populationen och antal generationer.
- Urval av populationen (hur många som ska bli föräldrar).
- Mutations och korsningsoperatorer.

Genom att öka storleken på populationen eller antalet generationer ökar också sannolikheten att hitta ett globalt optimum men jämte det även beräkningstiden. Som för alla heuristiska optimeringsmetoder är valen av parametrar en avvägningsfråga mellan resultat och beräkningstid och vad som är bäst beror på användarens preferenser. För att GA ska vara konkurrenskraftig tidsmässigt har empiriska tester påvisat att populationsstorleken behöver hållas relativt liten [3, s. 453].

K a p i t e l 5

SIMULERAD KYLNING

5.1 Om simulerad kylning

Simulerad kylning (*eng. Simulated annealing*, SA) är en heuristisk optimeringsmetod vars funktion liknar den process som mikrostrukturen i ett uppvärmt material såsom stål eller glas genomgår då den svalnar långsamt och kontrollerat (glödgning). Metoden presenterades 1983 av S. Krikpatrik m.fl [13] som beskriver algoritmen enligt följande. *"Annealing, as implemented by the Metropolis procedure, differs from iterative improvement in that the procedure need not get stuck since transitions out of a local optimum are always possible at nonzero temperature. A second and more important feature is that a sort of adaptive divide-and-conquer occurs. Gross features of the eventual state of the system appear at higher temperatures; fine details develop at lower temperatures"*.

Hromkovič beskriver SA som en mer utvecklad variant av *local search* med bättre verktyg för att undvika att fastna i lokala optima [3, s. 433]. *Local search* sv. lokalsökning är en metod som söker i närliggande punkter och till skillnad från SA endast accepterar lösningar som är bättre. Detta resulterar i att lokalsökningen ofta hittar just ett lokalt men inte alltid globalt optimum [8, s. 422]. Vidare menar han att att algoritmens funktion mycket förenklat kan beskrivas som följande: för att hitta den lägsta punkten i ett område bestiger en person först ett högt berg där hon kan kika efter "lovande" dalar. Därefter besöker personen den dal hon fann mest lovande och fortsätter med en rekursiv metod att söka efter den lägsta punkten i denna dal.

5.2 Funktion och implementation

I förklaringen som följer används begreppen *ledare* och *granne* vilka båda är individer av de slag som definieras i avsnitt 3.1. En granne är en närliggande punkt till ledaren. Individerna evalueras med avseende på deras fitness. Algoritmen kan följas stegvis i flödesschemat i figur 5.1. Läsaren rekommenderas följa denna figur parallellt med beskrivningen som följer.

SA är en algoritm som bygger på slump. Den grundläggande funktionen är att den genererar en granne n till ledaren p och kommer med viss sannolikhet att acceptera denna granne som ny ledare *även* om den är sämre (i detta fall högre fitness), vilket gör att algoritmen kan flytta sig från lokala optima [6]. Sannolikhetsfunktionen $P(T(i))$ som avgör sannolikheten för att acceptera en granne som ny ledare utgör grunden till SA och definieras i ekvation 5.1. Till en början när temperaturen är hög är sannolikheten att acceptera en granne som ny ledare också hög och minskar därefter i takt med temperaturen. Även differensen i fitness mellan två individer dE (ekvation 5.2) påverkar sannolikheten för acceptans. Temperaturen minskar enligt kylfunktionen $T(i)$ som definieras i ekvation 5.3 och beror på vilken iteration i algoritmen befinner sig i.

En granne som är bättre accepteras *alltid*. För att ett byte ska ske till en granne med sämre fitness ska villkoret $q < P(T(i))$ gälla för ett visst värde på i, där q är ett för varje i slumpmässigt genererat tal $q \in \{0, 1\}$. $P(T(i))$ kommer alltid att vara avtagande eftersom dE alltid kommer att vara negativ när grannen har högre fitness. I takt med att $P(T(i))$ avtar, minskar också sannolikheten att $q < P(T(i))$ uppfylls och därmed sannolikheten att flytta till ett sämre tillstånd. Algoritmen stoppas då alla iterationer har exekverats och temperaturen således har nått sin sluttemperatur T_{slut}.

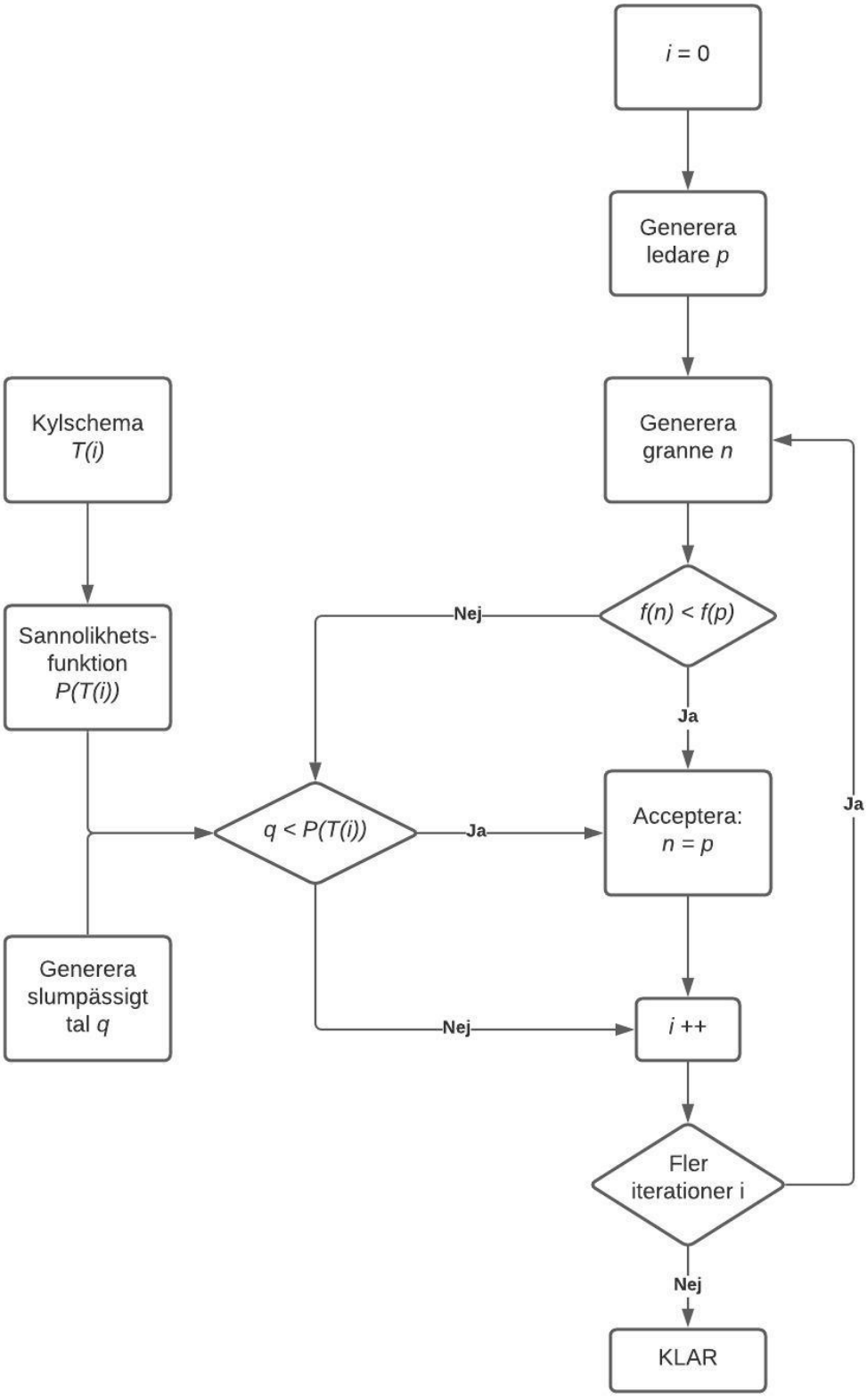
$$P(T(i)) = e^{\frac{dE}{T(i)}} \quad dE < 0 \tag{5.1}$$

$$dE = f(p) - f(n) \tag{5.2}$$

$$T(i) = T_{slut} + T_{start} \cdot \alpha^i \tag{5.3}$$

$P(T(i))$	sannolikhetsfunktion
$f(\upsilon)$	fitnessfunktion
$T(i)$	kylfunktion
i	iteration
n	granne (individ)
p	ledare (individ)
$\alpha \in \{0, 1\}$	kylfaktor
T_{start}	starttemperatur (hög)
T_{slut}	sluttemperatur (låg)
$q \in \{0, 1\}$	slumpmässigt genererat tal

Tabell 5.1: Parametrar i simulerad kylning



Figur 5.1: Simulerad kylning

Grannskap

Algoritmen genererar en granne på samma sätt som en mutation sker i den genetiska algoritmen (se avsnitt 4.2, Mutation). Det innebär att grannen kan genereras på tre olika sätt, genom swap, insert eller revert vars funktion redogörs utförligt i det föregående kapitlet. En av dessa operatorer utförs på ledaren och grannen är därmed genererad. Vilken av dessa operatorer som används i varje fall beror på slumpen. Oavsett vilken operator som används kommer operationssekvensen att ändras. Ofta omallokeras också flera operationer, se till exempel hur insert i figur 4.5 byter plats på tre operationer (13, 11 och 5). Grannskapet (*eng. neighborhood*) definieras som alla de individer som kan räknas som en utvald individs granne. Eftersom en granne kan ändras relativt mycket från ledaren, både med sekvensändringar och omallokeringar anses grannskapet mycket komplext och någon närmare beskrivning av detta görs inte i denna rapport.

5.3 Mer om simulerad kylning

Simulated Annealing har sedan dess introduktion 1983 blivit en välanvänd och framgångsrik algoritm, känd framför allt som en relativt enkel och lättimplementerad algoritm som tar fram robusta lösningar för nästan alla optimeringsproblem [3, s. 430].

Det finns flera problem med SA, bland annat att ställa in dess parametrar och att kartlägga dess uppförande [14, s. 670]. För att ställa in dem krävs en grundlig analys av den data som ska optimeras och även en del kvalificerade gissningar [6]. De parametrar som behöver definieras och ställas in och vilka också anses som problematiska är:

- Val av granne n
- Val av starttemperatur T_{start}

Kodningen av funktionen som genererar grannar är en annan aspekt som kan påverka algoritmens resultat avsevärt: genom att söka efter grannar i ett större grannskap ökar möjligheterna att hitta bra lösningar - men också beräkningstiden [20]. Ett större grannskap innebär att genmanipulationen är större - fler operationer byter plats.

T_{start} bör vara så pass hög att algoritmen till en början kan acceptera precis alla lösningar och borde inte vara högre då det resulterar i onödig belastning på processorn [6]. Om alla lösningar ska accepteras till en början ska dE vara maximalt, d.v.s. differensen mellan det högsta och det lägsta möjliga fitnessvärdet. Detta är givetvis problematiskt eftersom både det lägsta och det högsta fitnessvärdet är okänt - faktum är att det lägsta är det som eftersöks och det högsta torde vara lika krävande att hitta. Approximativa värden kan tas fram genom att analysera data från en s.k. *random search*, en slumpmässig genomsökning av sökrymden där det högsta och lägsta fitnessvärden som registreras används för att beräkna dE. Detta ska tilläggas är en tidskrävande process och resultaten är endast giltiga för det specifika fallet.

I detta fall används en exponentiell kylfunktion vilken i litteraturen ofta är ansedd att vara bäst på grund av att den ger en bra avvägning mellan snabbhet och resultat [20]. Det finns andra varianter på kylfunktioner till exempel linjära eller hyperboliska.

Kapitel 6

FALLSTUDIER OCH RESULTAT

6.1 Testmiljö

För att testa algoritmernas funktionalitet och effektivitet byggdes en testmiljö i planeringsprogrammet ROB-EX vilket är ett avancerat, multifunktionellt planeringsprogram avsett för tillverkande industri. Programmet genererar i huvudsak ganttscheman och kan integreras mot olika affärs- och styrsystem. För den intresserade läsaren finns mer information på ROB-EX hemsida [1]. Testmiljön utformades som ett bryggeri med 15 produktionslinjer och ett varierande antal fyllningslinjer (3 - 7) som var målet för optimeringen. Bryggeriet hade 15 olika drycker att producera. En dynamisk ställtidsmatris (se tabell 6.1) upprättades med en slumpmässig ställtid mellan 0.5 och 1.5 timmar. Varje order tilldelades ett leveransdatum. Bryggeriet är delvis baserat på ett verkligt fall och kan anses vara ett trovärdigt scenario.

	Carlsbe	Champi	Coke 50	Corona	Eriksber	Fanta 5	Harboe	Mariest	Millenit	Norrlan	Peroni !	San mig	Sofiero	Sprite 5	Staropr
Carlsberg 500	0.00	1.00	0.33	1.35	0.55	1.35	1.47	1.03	1.50	1.10	0.75	1.45	1.08	0.45	1.45
Champis 500	1.50	0.00	1.02	0.98	1.33	1.00	0.38	1.40	0.58	0.77	0.60	1.27	1.17	1.20	1.33
Coke 500	0.72	0.75	0.00	1.03	0.68	1.00	0.47	1.43	0.52	1.30	1.20	1.22	0.35	1.05	1.47
Corona 500	0.78	0.33	0.55	0.00	0.78	1.10	0.72	1.17	1.05	1.02	0.62	0.83	1.32	0.35	1.17
Eriksberg 500	0.73	1.43	0.78	0.80	0.00	0.90	1.42	0.93	0.37	0.98	0.77	0.80	1.20	0.75	0.35
Fanta 500	0.42	0.62	0.82	0.50	1.18	0.00	1.23	0.65	1.28	0.47	1.17	1.35	1.37	0.47	0.68
Harboe 500	0.78	0.52	1.45	0.63	0.70	0.75	0.00	0.65	1.42	0.92	1.40	0.95	0.73	1.40	1.12
Mariestad 500	1.48	0.70	1.43	0.67	0.35	0.52	1.08	0.00	0.83	0.92	0.97	1.10	0.65	1.00	0.42
Millenium 500	1.23	0.35	0.82	0.93	1.22	1.40	0.48	1.43	0.00	1.07	0.57	1.48	0.35	1.32	1.30
Norrlands guld 500	0.87	1.22	0.80	0.87	1.23	0.50	0.92	0.77	0.38	0.00	1.33	0.45	0.93	1.18	0.80
Peroni 500	0.93	1.00	1.37	0.55	1.08	1.35	0.50	1.17	0.38	1.50	0.00	0.45	0.87	1.28	0.37
San miguel 500	1.48	0.98	0.67	0.87	1.17	0.35	1.32	0.95	1.10	1.30	1.27	0.00	0.33	0.58	0.63
Sofiero 500	1.35	0.60	1.00	0.50	1.27	0.72	0.73	1.13	1.27	1.43	0.82	1.47	0.00	0.85	0.67
Sprite 500	0.55	0.97	1.07	0.75	0.78	1.10	0.63	1.25	1.10	0.52	0.93	1.32	1.17	0.00	0.75
Staropramen 500	0.47	1.08	1.22	1.37	1.08	0.38	0.65	0.35	1.43	0.62	1.00	0.68	1.32	0.90	0.00

Tabell 6.1: Ställtidsmatris

6.2 Sammanfattning av fallstudier

För att kartlägga algoritmernas beteende genomfördes fyra fallstudier med olika mål, vilka sammanfattas nedan. Emellanåt används notationen operationerXresurser, till exempel innebär 30x3 ett fall med 30 operationer och 3 resurser.

1. Hur väl algoritmerna presterar jämfört med på förhand uträknade basmått. Fall: 30x3.
2. Hur antalet individer påverkar resultatet och exekveringstiden. Fall: 60x4 med ökande antal individer från 50 000 till 200 000.
3. Hur antal operationer och resurser påverkar exekveringstid. Fall: från 50x3 till 200x7.
4. Hur robusta algoritmerna är. Fall: 50x5.

Alla mätvärden som återges i fallstudie 1-3 är medelvärden av *minst* 3 exekveringar. För att så rättvist som möjligt jämföra algoritmerna, ställdes dess parametrar in så att antalet evaluerade individer var lika. Dessa varierades mellan 50 000 och 200 000 vilka var de gränser som gav upphov till vad som kan anses som rimliga exekveringstider. En närmare överblick av parametrarna återges i tabell 6.2. Notera åter att fitness är ett värde som ska minimeras och lägre fitness således är fördelaktigt.

	Antal individer	50000	100000	150000	200000
GA	Generationer	1000	1000	1000	1000
	Population	100	200	300	400
	Antal föräldrar	50	100	150	200
	Antal avkommor	50	100	150	200
	c_r	0.95			
	m_r	0.05			
SA	Iterationer	50000	100000	150000	200000
	T_{start}	60-200			
	T_{slut}	1			
	α	0.95			

Tabell 6.2: Parametrar som användes för olika antal individer

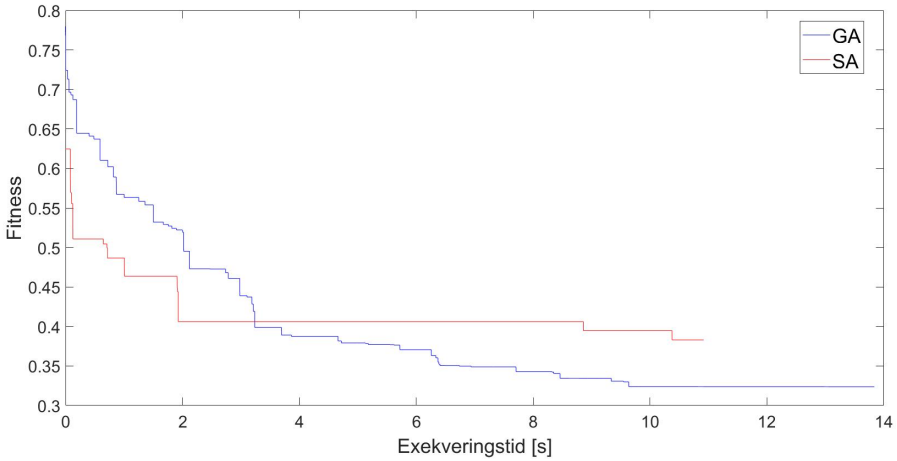
6.3 Fallstudier

Fallstudie 1

Syftet med fallstudie 1 var att för ett realistiskt fall undersöka hur bra algoritmerna optimerade vardera variabel. Fallet gjordes med 30 operationer och 3 resurser och för varje variabel beräknades ett basmått - det minsta möjliga värdet variabeln skulle kunna nå. Basmåtten beräknades enligt nedan, och återges i tabell 6.3. Antal individer ställdes in till 100 000.

Resultatet återges i tabell 6.3 och åskådliggörs i figur 6.1. Den genetiska algoritmen presterade bättre både avseende ställtid och förseningar. Gällande den tredje variabeln dödtid, hittade båda algoritmer en optimal sekvens. SA gjorde en 5.5 sekunder snabbare optimering än GA. Figur 6.1 visar de båda algoritmernas arbete för att hitta den optimala lösningen. Här tydliggörs att SA konvergerar snabbare, vid t = 2 sekunder medan GA konvergerar vid t = 10 sekunder. Basmåtten som användes som referens beräknades enligt följande:

- För ställtid beräknades basmåttet som medelvärdet av ställtidmatrisen · (antal operationer - antal resurser). Resurserna subtraheras eftersom ingen ställtid uppkommer för de första operationerna (eftersom ingen operation kommer före de första behövs inte maskinerna rengöras eller ställas in). Detta motsvarar ställtiden av en helt slumpmässig planering.
- Genom att bestämma en fast tidpunkt t och tilldela hälften av operationerna som skulle planeras ett försenat leveransdatum t-12h och den andra hälften ett leveransdatum t+12h (som kan uppnås), kunde den totala förseningen beräknas i den bästa möjliga planeringen.
- Dödtiden beräknades till 0.



Figur 6.1: Fallstudie 1: konvergens

Variabel	Basmått	GA	SA
Ställtid [h]	23.5	8.3	10.7
Förseningar [h]	516	571	600
Dödtid [h]	0	0	0
Exekveringstid [s]		16.8	11.3

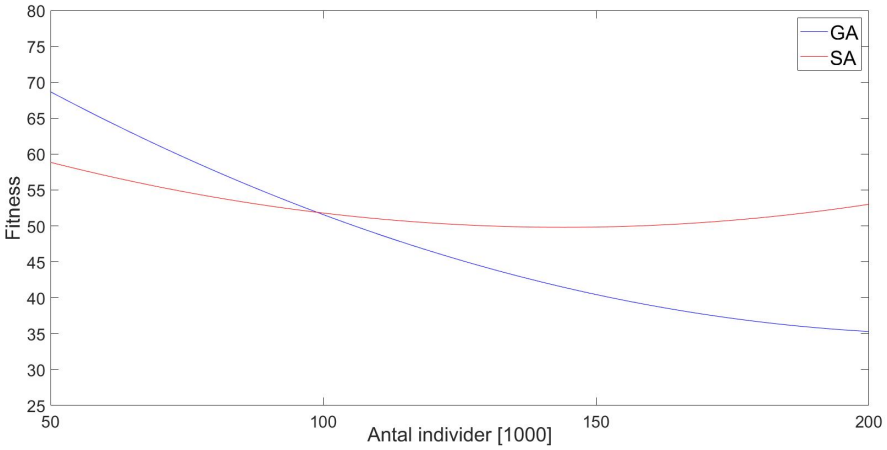
Tabell 6.3: Resultat Fallstudie 1

Fallstudie 2

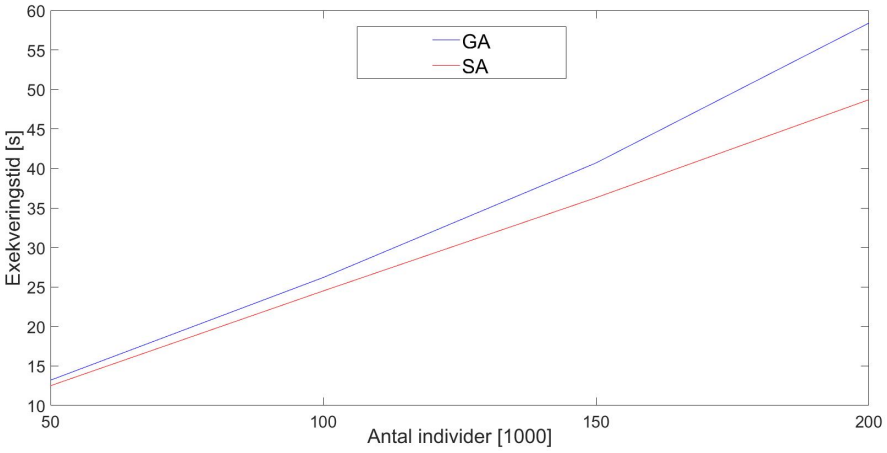
I fallstudie 2 undersöktes hur antalet individer påverkade resultat och exekveringstid för ett fixt antal operationer och resurser nämligen 60 respektive 4. Varje algoritm testades för 50 000 - 200 000 individer i fyra steg.

Resultatet från fallstudie 2 redovisas i figur 6.2. I figur 6.2a har fitness av de båda algoritmerna avbildats som funktion av antalet individer. SA har bättre fitness för individer < 100 000 men konkurreras därefter ut av GA. SA påvisar heller inte någon markant förbättring av ett ökat antal individer utan ger till och med ett sämre resultat då individer > 150 000. Den genetiska algoritmen tycks hitta allt bättre lösningar vid ökat antal individer. Dess funktion är svagt exponentiell och tycks konvergera efter 200 000 individer.

Angående exekveringstiden som illustreras i figur 6.2b uppförde sig algoritmerna mycket likartat. Tiden det tar för algoritmerna att finna en lösning ökar linjärt med antal individer. Den genetiska algoritmen påvisade en något högre derivata och exekveringstiden växte därmed fortare för denna.



(a) Fitness - individer



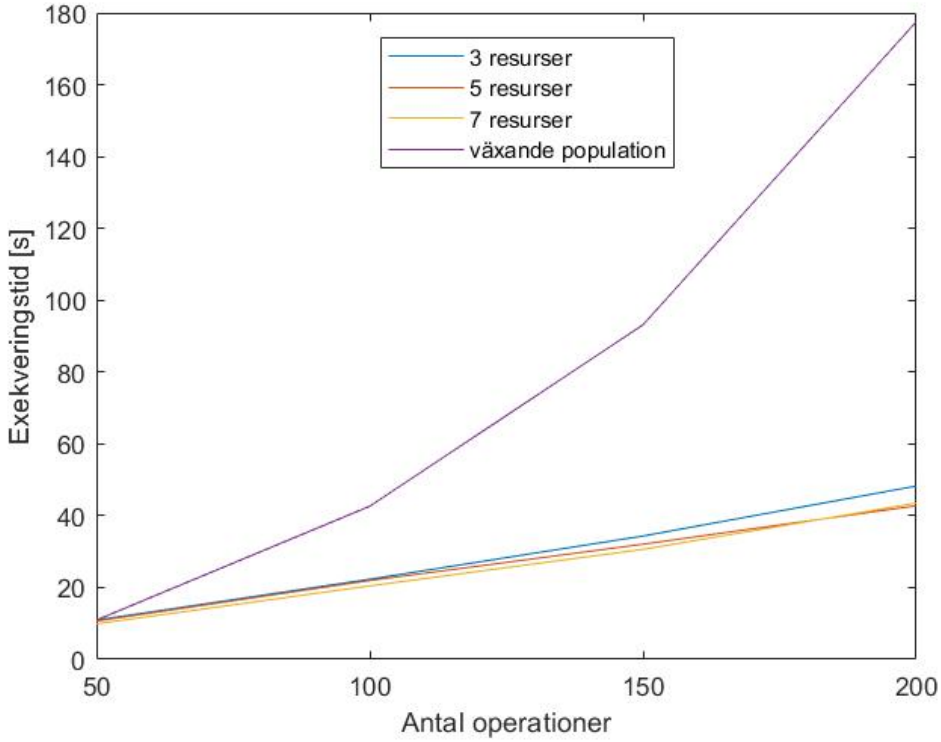
(b) Tid - individer

Figur 6.2: Fallstudie 2: Påverkan av antal individer

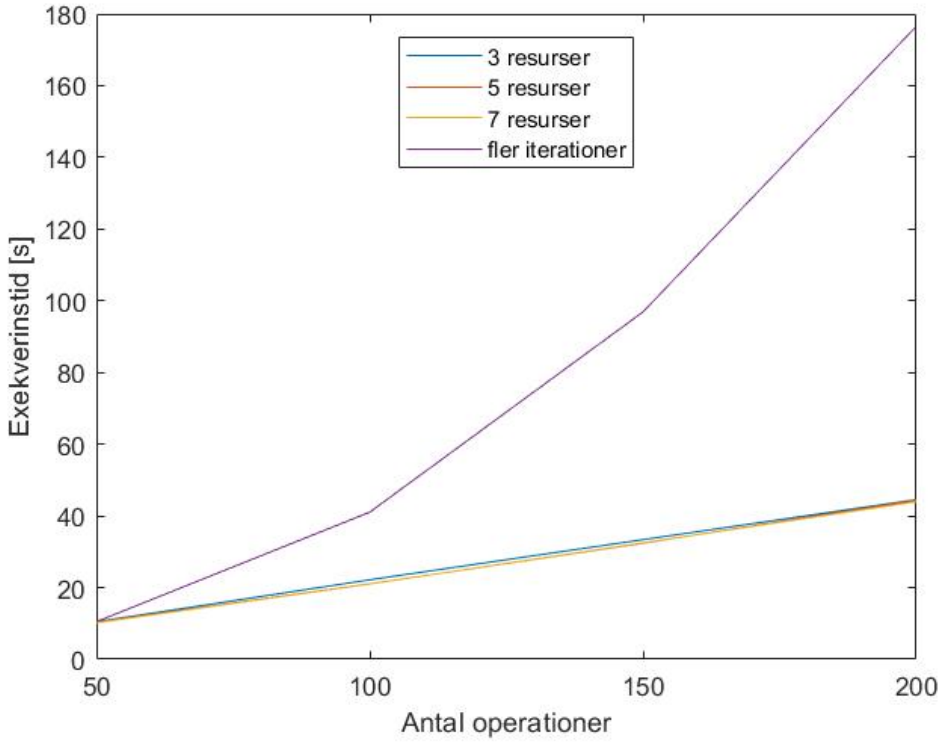
Fallstudie 3

Fallstudie 3 utfördes för att analysera hur antal operationer och resurser påverkade exekveringstiden för ett fixt antal individer. I takt med att dessa ökar blir problemet mer komplext och för att algoritmerna fortfarande ska kunna prestera bör dessa kompenseras med fler individer. Dessutom ändras *dE* för SA, vilken måste kompenseras med högre initialtemperatur T_{start}. Därför gjordes även en simulering där algoritmerna kompenserades med detta. Simuleringar genomfördes med 50 - 200 operationer i fyra steg, fördelat på 3, 5 och 7 resurser. Endast exekveringstiden noterades.

Resultatet åskådliggörs i figur 6.3. Den genetiska algoritmen 6.3a uppvisar ett mycket likartat beteende med SA 6.3b. Antalet resurser tycks inte ha någon påverkan på exekveringstiden medan antalet operationer orsakar en linjär ökning. Om däremot algoritmerna kompenseras med ett ökat antal individer i takt med ökat antal operationer växer exekveringstiden exponentiellt, fortfarande mycket likartat. Ett fall med 200 operationer och 200 000 individer tog de båda algoritmerna ca 3 *minuter* att utföra.



(a) GA



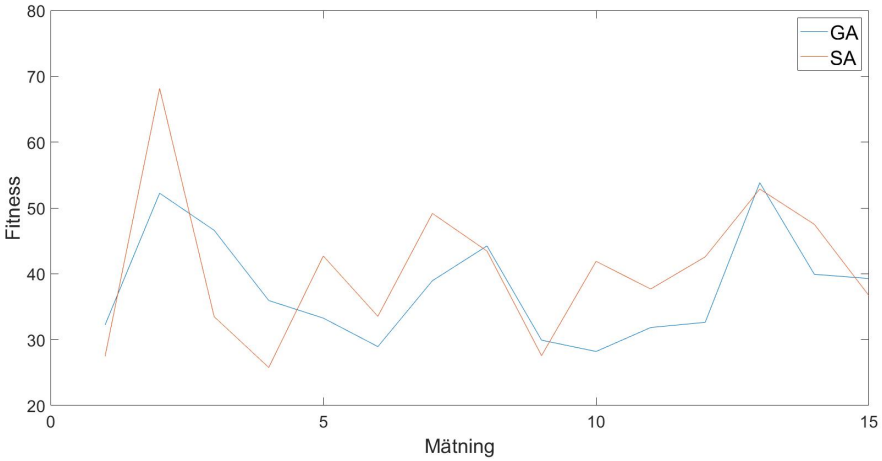
(b) SA

Figur 6.3: Fallstudie 3: Tid - operationer - resurser

Fallstudie 4

Fallstudie 4 genomfördes för att studera hur algoritmernas resultat skiljde sig åt under upprepade exekveringar för samma fall med oförändrade parametrar. 15 stycken exekveringar gjordes för vardera algoritm för ett fall med 50 operationer och 5 resurser. Under tidigare fallstudier noterades att exekveringstiden är relativt stabil - därför togs ett beslut om att inte göra någon kartläggning över exekveringstidens variation.

Resultatet illustreras i figur 6.4 och återges i tabell 6.4. GA påvisar ett mer stabilt beteende med en standardavvikelse på 8.70 jämfört med 11.14 för SA och har också ett lägre medelvärde och median. Däremot hittade SA den bästa lösningen - med minsta fitness = 25.77.



Figur 6.4: Fallstudie 4: Robusthet

Fitness	GA	SA
Median	35.94	41.89
Min	28.20	25.77
Medel	37.87	40.7
Standardavv.	8.70	11.14

Tabell 6.4: Resultat Fallstudie 4

Kapitel 7

SLUTSATS OCH DISKUSSION

En sammanställning av resultaten från fallstudie 1 - 4 redogörs nedan:

1 GA optimerar bättre avseende alla variabler men har en längre exekveringstid.

2 För färre individer hittar SA en bättre lösning men GA vinner vid ökat antal. Exekveringstiden växer likartat, linjärt.

3 Antal resurser påverkar ej exekveringstiden. Däremot ökar den linjärt med antal operationer och exponentiellt om antal individer ökas i takt med operationerna.

4 För upprepade exekveringar uppvisar GA ett något mer robust beteende än SA.

Fallstudie 1, 2 och 4 visar att den genetiska algoritmen för det mesta ger ett bättre resultat men med en längre exekveringstid. SA kan producera en bra lösning snabbt men ger inte möjlighet att hitta påtagligt bättre lösningar vid ökat antal individer, vilket speciellt visualiseras i fig. 6.2a. Det finns inget stöd i teorin om att SA skulle generera sämre lösningar vid ännu fler individer och beteendet får därför tolkas som att den når ett lokalt optimum. Orsaken till detta beteende och även orsaken till att den ger ett sämre resultat än GA kan vara på grund av definitionen av det så kallade grannskapet. Det är mycket komplext att beskriva hur grannskapet ser ut. Det kan sägas vara relativt stort eftersom det i vissa fall innebär flera förändringar från ledaren. Detta hade kunnat vara föremål för en fördjupad studie för algoritmen. Även parametrarna, i synnerhet starttemperaturen och kylfunktionen kan vara en orsak till beteendet. Det finns mer avancerade metoder för att ställa in parametrarna för såväl SA som GA. Algoritmerna kan emellertid producera bra lösningar även med icke-optimalt inställda parametrar.

Vilken algoritm som är att föredra är högst beroende av den rådande situationen. I vissa fall kan det vara av högsta vikt att exekveringstiden minimeras för vilka SA skulle vara förstahandsvalet. I andra fall där resultatet är viktigare skulle GA vara att föredra. Dessutom har GA fördelen att den hittar flera bra lösningar som användaren kan välja emellan.

Utöver resultatet från fallstudierna är SA betydligt lättare att implementera - däremot är parametrarna, i synnerhet T_{start}, svårare att ställa in än de för GA. En annan fördel med GA är att ingen hänsyn behöver tas till skalning - d.v.s. variablerna kan tillåtas vara av helt olika storleksordning, som till exempel leveransförseningar och ställtid i fallstudie 1 (jämför tabell 6.3).

Optimeringsalgoritmerna bedöms fungera väl för fall som anses vara av normal storleksordning - ett 30 tal operationer och 2-4 resurser, vilket bevisades framför allt med basmåtten i fallstudie 1. För större fall gjordes inga mätningar mot basvärden eftersom de blir väldigt tidskrävande och komplexa att beräkna. Det kan dock antas att de presterar väl även här med kompenserat antal individer. En av svårigheterna med optimeringsalgoritmer är att bevisa att den lösningen som har hittats är den optimala - eller hur långt ifrån den optimala lösningen den är.

I en fortsatt utveckling av algoritmera skulle fler villkor kunna tas med. Till exempel är det inte ovanligt att vissa produkter inte kan utföras på alla resurser, eller kan det finnas vissa önskemål från produktionen att vissa produkter ska utföras på specifika resurser. En annan funktionalitet som är efterfrågad är att få med underhåll i planeringen. Ofta ska resurserna underhållas med jämna mellanrum - till exempel oljebyte varannan vecka eller liknande. Det är fördelaktigt att utföra dessa underhåll i samband med produktbyte eftersom resurserna ändå måste försättas i vila. Det kan finnas många andra krav på planeringen att ta hänsyn till men i denna utredning är de vanligaste och kanske viktigaste kraven utredda.

LITTERATURFÖRTECKNING

[1] URL https://rob-ex.com/en/.

[2] M. R. Amin-Naseri and A. J. Afshari. A hybrid genetic algorithm for integrated process planning and scheduling problem with precedence constraints. *International Journal of Advanced Manufacturing Technology*, 59:273–287, 2012. doi: 10.1007/s00170-011-3488-7.

[3] J. Hromkovič. *Algorithmics for Hard Problems*. Springer, Germany, second edition, 2004. ISBN 978-3-642-07909-2.

[4] R. Colin. *Genetic Algorithms*, pages 55–82. Springer US, Boston, MA, 2003. ISBN 978-0-306-48056-0. doi: 10.1007/0-306-48056-5_3. URL https://doi.org/10.1007/0-306-48056-5_3.

[5] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, first edition, 1989. ISBN 0201157675.

[6] P. Brinch Hansen. Simulated annealing. *Electrical Engineering and Computer Science - Technical Reports*, 170, 6 1992.

[7] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, USA, 1975. ISBN 0472084607.

[8] K. Holmberg. *OPTIMERING - Metoder, modeller och teori för linjära, olinjära och kombinatoriska problem*. Liber, 2010. ISBN 9789147099351.

[9] A. Jahan. *Multi-criteria Decision Analysis for Supporting the Selection of Engineering Materials in Product Design.* Elsevier, 2016. ISBN 0081005369.

[10] D. Kalyanmoy, A. Samir, Amrit P., and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, pages 849–858, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-45356-7.

[11] A. Király and J. Abonyi. Optimization of multiple traveling salesmen problem by a novel representation based genetic algorithm. *Köppen M., Schaefer G., Abraham A. Intelligent Computational Optimization in Engineering*, pages 241–269, 2011. doi: https://doi.org/10.1007/978-3-642-21705-0_9.

[12] A. Király, M. Christidou, T. Chovan, E. Karlopoulos, and J. Abonyi. Minimization of off-grade production in multi-site multi-product plants by solving multiple traveling salesman problem. *Journal of Cleaner Production*, 111: 253–261, 2015. doi: http://dx.doi.org/10.1016/j.jclepro.2015.05.036.

[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983. URL https://www.jstor.org/stable/1690046.

[14] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson/Addison-Wesley, Boston, Massachusetts, first edition, 2006. ISBN 0321295358.

[15] A. Kunapareddy and G. Allaka. An improved genetic algorithm for production planning and scheduling optimization problem. In *Intelligent Manufacturing and Energy Sustainability*, pages 157–171, Singapore, 2020. Springer Singapore.

[16] C. Moon, Y. Seo, Y. Yun, and M. Gen. Adaptive genetic algorithm for advanced planning in manufacturing supply chain. *Journal of Intellegent Manufacturing*, 17:509–522, 2006. doi: 0.1007/s10845-005-0010-0.

[17] G. Olsson and C. Rosen. *Industrial Automation - Applications, Structures and Systems*. Department of Industrial Engineering and Automation, Lund University, Sweden, 2005.

[18] M. L. Pinedo. *Scheduling - Theory, Algorithms, and Systems*. Springer, New York, third edition, 2008. ISBN 9780387789347.

[19] M. L. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer New York, second edition, 2009. ISBN 9781441909091.

[20] W. Xia and Z. Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers Industrial Engineering*, 48(2):409–425, 2005. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2005.01.018.