En-förälders genetisk algoritm med linjärt minskande mutationsfaktor

En jämförelse med standard genetisk algoritm

Jakob Vikström

Oscar Hansson

Institutionen för
data- och systemvetenskap

Examensarbete 15 hp
Data- och systemvetenskap
Kandidatprogram i Data- och Systemvetenskap (J.V 180 HP)
Kandidatprogram i Datavetenskap (O.H 180 HP)
Vårterminen 2023
Handledare: Pierre Arne Ingvar Wijkman
Engelsk titel: Single-Parent Genetic Algorithm with Linearly Decreasing Mutation Factor

Stockholms
universitet

Synopsis

Bakgrund

Att använda biologiskt inspirerade tekniker för att lösa stora och komplexa problem har undersökts sedan 1940-talet. En av dessa tekniker är genetisk algoritm. Denna algoritm hittar lösningar genom att undersöka mindre optimala lösningar och, precis som i naturen, tillämpa naturlig selektion och evolution för att avla en optimal lösning.

Problem

Med genetiska algoritmer finns det ingen garanti att ett globalt optimum kommer att upptäckas givet ett heuristiskt sökproblem. Ofta fastnar de i mindre optimala lösningar

Frågeställning

I vilka avseenden kan en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor skapa en bättre lösning än en standard genetisk algoritm?

Metod

Den empiriska forskningsstratergin experiment användes för att undersöka de olika genetiska algoritmernas prestation i flera handelsresandeproblem. Datainsamlingsmetoden som valdes var systematisk observation algoritmerna och observerades genom exporterade CSV-filer och grafer. Sedan valdes den statistiska metoden Mann-Whitney U test för att analysera data. Tillvägagånssättet för att skapa algoritmerna definierades och motiverades.

Resultat

De två algoritmerna utvecklades och exekverades på tio handelseresandeproblem. Resultatet redovisades i form av tabeller och grafer. Insamlad data analyserades sedan med ett Mann-Whitney U test. Testet visade på att den föreslagna algoritmen presterade bättre än en standard genetisk algoritm i samtliga testfall vilket bevisades signifikant.

Diskussion

I likhet med tidigare forskning visar denna studies resultat att det finns fördelar med en-förälders genetiska algoritmer. Studien visar att en en-förälders genetisk algoritm med linjärt minskande mutationsfaktor ökar lösningarnas precision vilket kan generaliseras och vidareföras till alla sök-optimeringsproblem där en lämplighetsfunktion kan definieras. Studien hade begränsningar i form av kontext, inställningar, tid och implementation av dessa algoritmer. Det är viktigt att resultatet från den här studien inte missbrukas på något sätt för att skada någon människa eller grupp av människor.

Sammanfattning

En genetisk algoritm är en evolutionär algoritm som använder biomimetiska principer för att efterlikna den darwinistiska evolutionen, i dessa grunder kan metaheuristiska problem lösas. En genetisk algoritm löser heuristiska optimeringsproblem genom att avla fram en bättre lösning genom mindre bra lösningar. En genetisk algoritm använder två föräldrar för skapandet av nya lösningar men både en förälder och fler än två föräldrar kan användas. Trots ständig utveckling av domänen är det inte alltid säkert att en genetisk algoritm hittar den optimala lösningen istället kan algoritmen fastna på en mindre optimal lösning. Genetiska algoritmer använder därför tekniken mutation vilket används som en lösning för att introducera glömda attribut in i populationen. Studien syftar till att undersöka om implementationen av en en-förälders-genetisk algoritm med linjärt minskande mutationsfaktor kan skapa bättre lösningar, lösningar, än en standard två-föräldrars-implementation. För att undersöka problemet skapades frågeställningen, vilket lyder, I vilka avseenden kan en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor skapa en bättre lösning än en standard genetisk algoritm? För att besvara denna frågeställning användes experiment som forskningsstratergi, observation som datainsamlingsmetod, och inferentiell statistik som analysmetodologi. Algoritmer med en förälder implementerades med linjärt minskande mutationsfaktor och algoritmer med två föräldrar implementerades med statisk mutationsfaktor, dessa användes för att lösa flera handelsresandeproblem. Data om algoritmernas prestanda sparades i CSV-filer och analyserades med Mann-Whitney U test. Grafer som illustrerade algortimernas beteende sparades även. Resultatet från testen visade att den föreslagna algoritmen presterade bättre än en standard genetisk algoritm i alla testfallen. En genetisk algoritm med en förälder och linjärt minskande mutationsfaktor gav lösningar med bättre precision vilket kan föras vidare till framtida studier om genetiska algoritmer och implementationer så som paketleveranser. Konsekvenserna av detta arbete kommer att ge en ökad förståelse för genetiska algoritmer och hur de kan implementeras för att underlätta i vardagliga problem, detta arbete är inte ämnat att skada människor utan endast ge kunskap och förbättra arbetsförhållanden.

Nyckelord: Evolutionär algoritm, Genetisk algoritm, Handelseresandeproblem, En-förälders-genetisk-algoritm, Optimering

Abstract

A genetic algorithm is a type of evolutionary algorithm that uses biometric principals to try to copy Darwinian evolution, this is used to solve heuristic search problems. A genetic algorithm solves problems by having a set of solutions and reproducing them to make better solutions. However, using a genetic algorithm on does not always yield the optimal solution. Although consistant research the algorithm can still get stuck on sub optimal solutions. A standard implementation of a genetic algorithm uses two parent solutions to create new solutions through reproduction but is possible to create new solutions using only one parent through mutation. This study will investigate if the implementation of a genetic algorithm with one parent and a linear decreasing mutation factor can create better solutions than a standard genetic algorithm implementation. The create a solution for these problems a research question was created. In what ways can a genetic algorithm with a single parent and a linearly decreasing mutation factor create a better solution than a standard genetic algorithm? To answer this question an experiment was used as the research strategy, observation was used as data collection method, and inferential statistics was used as methodology for analysis. The genetic algorithm with one parent was implemented using a decreasing mutation factor and the algorithm with two parents used a static mutation factor and both were used to solve different instances of the traveling salesman problem. The data concerning the algorithms performance was saved on CSV files and was analyzed with Mann-Whitney U tests. Graphs illustrating the behavior of the algorithms were also saved. The results from the tests showed that the proposed one parent algorithm performed better than the two-parent algorithm in all test cases. The genetic algorithm with one parent and a dynamically decreasing mutation factor was shown to give better solutions with better precision, this can be used in future studies concerning genetic algorithms and in practical applications such as package delivery. This study is not meant to harm any people but is instead meant to provide knowledge and better working conditions.

Keywords: Evolutionary algorithm, Genetic algorithm, Traveling Salesman Problem, Single-parent genetic algorithm, Optimization

Tack

Innan denna studie börjar vill vi tacka vår handledare Pierre Arne Ingvar Wijkman som guidat oss genom detta examensarbete. Utan dig skulle detta inte gått vägen!

Vi vill även tacka examinator, granskare och opponenter.

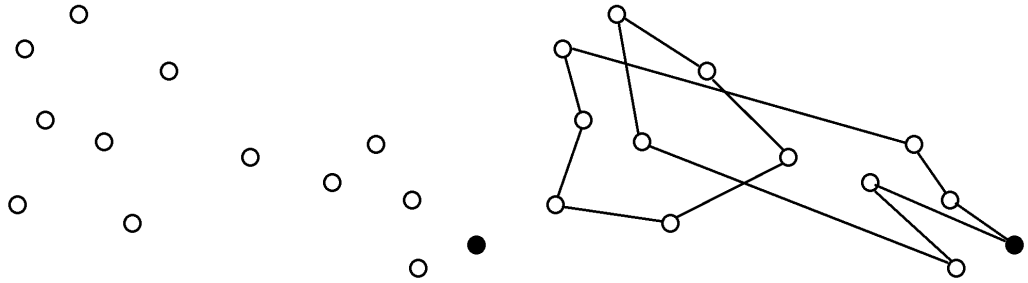
Innehåll

Figurer

Tabeller

1 Introduktion

1.1 Bakgrund



Figur 1: *Vänster: Planerad stadsdel med markerad punkt för postkontor Höger: Bervbärarens föreslagna postrutt som börjar och slutar vid postkontoret*

Stadsplaneringen för en ny stadsdel har precis godkänts av Flens kommunfullmäktige. Med en ny stadsdel krävs nya lösningar för samhällets infrastruktur. Därför har PostNord givits uppdraget att planera en ny postväg som ska omfamna alla nya bostäder i området samt ett nytt postkontor. En brevbärare anställd på PostNord i Flen får uppdraget att åka till platsen för den nya stadsdelen och skapa en postväg som all utdelning av post ska följa. Brevbäraren, utgår från platsen för det nya kontoret och vandrar mellan alla punkter, där byggnader kommer stå, och sedan tillbaka till kontoret. Den anställda markerar tydligt vägen som känns rätt och lämnar den till PostNords logistikchef. Chefen märker tidigt att något inte ser rätt ut med kartan. Det ser ut som att det borde finnas en bättre rutt. Därför beslutar logistikchefen att sätta sig ner och skapa en perfekt postväg. Men varje gång en nya postväg designats känner logistikchefen att det finns utrymme för förbättring. Efter att skapat femtiosju nya postvägar ger logistikchefn upp och väljer en godtycklig rutt som ska implementeras. Hur skulle denna process förbättras? Valet blev slumpmässigt vald då chefen inte hade en bra metod för att hitta den bästa postvägen. Processen som skapat en postvägen till den nya stadsdelen i Flen är väldigt lik en mindre precis process som en genetisk algoritm genomgår för att lösa liknande problem. Genom att använda en genetisk algoritm hade PostNord fått möjligheten till att använda en optimal postväg.

Att använda biologiska metaforer för att skapa nya tekniker, biomimetik, ger möjligheten att använda naturens lösningar replikerade i metaheuristiska algoritmer för att lösa stora komplexa optimeringsproblem. Många optimeringsproblem existerar utan det finns något exakt sätt att erhålla en optimal lösning. Vetenskapsmän har då undersökt implementationer från naturens principer för att lösa dessa problem (Sivanandam & Deepa, 2008). Optimering handlar om att hitta den bästa lösningen på ett problem, oftast genom att maximera eller minimera en funktion med flera variabler (Sivanandam & Deepa, 2008).

Många av dessa optimeringsproblem är NP-komplett. NP står för icke-deterministisk polynomialtid och refererar till en typ av problem, om problemet kan kan lösas av en algoritm och om lösningen kan verifieras i polynom tid så anses det vara NP (Eiben & Smith, 2015), NP-komplett däremot refererar till problem som är åtminstone lika svåra som alla problem i NP, alla problem i NP kan reduceras till ett NP-komplett problem i polynom tid (Eiben & Smith, 2015). Det innebär att om man lyckas lösa ett NP-komplett problem i polynom

tid kan man även lösa alla problem i NP i polynom tid (Applegate m. fl., 2006). Dessvärre är dessa oerhört stora och komplicerade matematiska problem. Det resulterar i att man istället måste förlita sig på andra tekniker som inte alltid hittar den optimala lösningen, globalt optimum, utan hittar lösningar som är nära nog, lokalt optimum. Ett sätt att verifiera om man har hittat en lösning som kan anses vara "nära nog" är att använda någon sorts metod för att hitta en nedre gräns på vad den kortaste vägen kan vara. Med tanke på att majoriteten av optimeringsproblem inte har någon algoritm som löser problemet på polynom tid och för att det finns flera lokala optimum är evolutionära algoritmer en bra kandidat för att hitta lösningar.

På 1940-talet undersökte Alan Turing om de biologiska principerna kring evolutionsteorin kan tillämpas i automatisk problemlösning. Turing föreslog en genetisk eller evolutionär sökning (Eiben & Smith, 2015). Idén implementerades aldrig. Denna domän utvecklades sedan mellan 1960-talet och 1990-talet, då fyra olika implementationer av liknande art framställdes. Dessa imiterade processer enligt det darwinistiska naturliga urvalet för att skapa heuristiska sökalgoritmer. Sedan 1990-talet har dessa implementationer uppfattats som olika discipliner inom samma domän, evolutionär beräkning, medan algoritmerna fick samlingsnamnet evolutionära algoritmer (Eiben & Smith, 2015). Genetisk algoritm, föreslagen av Holland (1992), är en välkänd variant av evolutionära algoritmer som använder "överlevnad för det mest lämpade" och evolution för att lösa komplexa problem. Algoritmen använder sig av, precis som naturlig evolution, en population med individer. Individer kan beskrivas som en kandidatlösning på problemet som den genetiska algoritmen försöker lösa (Sivanandam & Deepa, 2008). Individerna evalueras med en lämplighetsfunktion som mäter hur väl de presterar i förhållande till det problem som ska lösas. Genom evaluering kan de individer som presterat bäst väljas för att avla fram nya individer, denna process kallas selektion och individerna som valts benämns föräldrar. För att skapa dessa individer används teknikerna korsning och mutation. Korsning är tekniken för att avla fram nya individer med en eller flera föräldrar. En standard genetisk algoritm använder två föräldrar. De nya individerna erhåller en blandning av föräldrarnas attribut och populerarar en ny generation av lösningar. Mutation är en teknik för att bibehålla genetisk mångfald och undvika informationsförlust. För varje generation av populationen finns det en chans att mutera individers attribut. Chansen bestäms genom en mutationsfaktor som kan vara dynamisk eller statisk. En dynamisk mutationsfaktor förändrar chansen för mutation under varje generation. Sivanandam och Deepa (2008) beskriver att om korsning används för att skapa bättre lösningar genom att föda lösningar genom två föräldrar används mutation för att den genetiska algoritmen ska undersöka alla attribut. Dessa steg upprepas sedan tills ett definierat avslutsvillkor är uppfyllt (Eiben & Smith, 2015).

Holland (1992) förmedlar att nyttan med algoritmen kommer från att alla funktioner av ett problem och stegen som måste tas inte behöver förutses av en forskare istället kan ett program "avlas" genom genetiska algoritmer. Hollands teknik är en av de första implementationerna av biomimetik inom datavetenskapen och har använts flitigt de senaste åren för att lösa nya optimeringsproblem (Sivanandam & Deepa, 2008).

1.2 Problem

Med evolutionära algoritmer finns det ingen garanti att en optimal lösning kommer att upptäckas givet ett NP-komplett problem (Eiben & Smith, 2015). I Spears (2000) nämns evolutionära algoritmers tendens till att snabbt koncentrera sina resurser på lovande delar av sökutrymmet som en av dess mest attraktiva funktioner. Vidare beskrivs också detta som en potentiell bristfällighet genom att evolutionära algoritmer lätt tappar mångfald i populationen. Konsekvensen av förlusten är att algoritmen enkelt fastnar i lokala optimum. Återkommande i optimeringsproblem som ska lösas med evolutionära algorimter är därför balansen mellan att

avla individer genom föräldrar med gynsamma lösningar och undvika ogynsamma lösningar. Det är enklare sagt än gjort. Genetiska algoritmer använder därför tekniken mutation vilket används som en lösning för att introducera nya attribut in i populationen. Vanligtvis implementeras en statisk mutationsfaktor i dessa algoritmer. En linjärt minskande mutationsfaktor innebär att för varje generation i den genetiska algoritmen minskar chansen för att mutation inträffar i en kandidatlösning med ett förutbestämt värde. I Hassanat m. fl. (2019) kommer utredarna fram till att en minskande mutationsfaktor på en population av 100 kan ha en positiv effekt på algoritmens prestanda, dessa algoritmer använde dock två föräldrar för att skapa optimeringslösningar.

En förälders genetiska algoritmer har implementerats flera gånger för att undersöka olika optimeringsproblem. Cantó m. fl. (2009) förespråkade en asexuell genetisk algoritm som använder en förälder och endast mutation för att modifiera populationen i optimeringsproblem. I en litteraturstudie om dåtid, nutid och framtid för genetiska algoritmer beskrivs nuvarande svårigheter fortfarande vara att hitta en optimal mutationsfaktor och för den delen undvika lokalt optimum (Katoch m. fl., 2020). Amirghasemi och Zamani (2015) och Salesi m. fl. (2021) använder grund idén från Cantó m. fl. (2009) för att undersöka asexuella genetiska algoritmer i andra användningsområden, med mer än tio år av denna teknik och ständig utveckling består ändå samma problem.

Tidigare studier har alltså undersökt och dragit slutsatsen att det finns implementationer av en-förälders algoritm som presterar bättre än standard genetiska algoritmer. Det finns alltså många aspekter att undersöka för att försöka skapa en bättre en-förälders genetisk algoritm. Givet att det fortfarande inte finns någon garanti för globalt optimum men att en en-förälders algoritm ändå visat på goda resultat är det en utmärkt utgångspunkt för att skapa en algoritm med syftet att närma sig ett garanterat globalt optimum. För att denna algoritm ska ge ännu bättre chans att nå globalt optimum bör mutationsfaktorn, Som beskrivits i ovannämnda studier, inte implementeras som statisk. Då Hassanat m. fl. (2019) visat på att en linjärt minskande mutationsfator ger god förutsättningar för att komma närmare en global lösning bör en en-förälders algoritm med linjärt minskande mutationsfaktor ge ännu bättre förutsättningar.

Problemet som ligger till grund för denna studie är att genetiska algoritmer enkelt hamnar i lokala optimum vilket resulterar i sämre lösningar. Därför undersöker denna studie en en-förälders algoritm med dynamisk mutationsfaktor som en del av lösningen på detta problem.

1.3 Frågeställning

I vilka avseenden kan en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor skapa en bättre lösning än en standard genetisk algoritm?

1.4 Centrala begrepp

1.4.1 Handelsresandeproblemet

Handelsresandeproblemt är ett optimeringsproblem och går ut på att hjälpa en resande handelsman givet en samling städer i en graf att hitta den kortaste vägen som besöker alla städer max en gång och slutar i samma stad som säljaren började i. Problemet är välkänt över hela världen inom matematik och datavetenskap dels för att det är relativt enkelt att begripa problematiken och dels för att det är svårare att lösa ju mer städer som ska besökas vilket kan lösa många reella problem (Applegate m. fl., 2006). Mycket av det tidiga arbetet med handelsresandeproblemet involverade praktiska tillämpningar som schemaläggning för bussar

eller sophämtning (Applegate m. fl., 2006) men på senare år har det använts mycket inom datavetenskapen och evolutionära algoritmer.

1.4.2 Korsning

Korsning inom genetiska algoritmer involverar traditionellt sett två föräldrar och är en process där individerna delar genetiskt material för att skapa nya individer som förs vidare till nästa generation av algoritmen, det finns flera olika metoder för att korsa föräldrarna men principen är alltid att välja ut två kandidater med bra genetiskt material för att sedan skapa förhoppningsvis bättre individer Sivanandam och Deepa (2008). Begreppet används i studien synonymt med transposition då skillanden endast är antal föräldrar som ingår i operationen. Nedan finns ett exempel på hur korsning ser ut i en genetisk algoritm med två föräldrar.

Förälder 1: [A, B, C, D, E, F]
Förälder 2: [G, H, I, J, K, L]
Barn: [A, B, C, J, K, L]

1.4.3 Mutation

Mutation är en process som sker efter korsningen, denna process är menad att hindra algoritmen från att fastna i lokala optimum och ska hjälpa algoritmen att undersöka större delar av sökutrymmet Sivanandam och Deepa (2008). Mutation kräver endast en individ och kastar om det genetiska materialet inom den egna kromosomen eller inte. Nedan visas ett exempel på mutation inom en individ.

Individ innan mutation: [A, B, C, D, E, F]
Individ efter mutation: [F, B, C, D, E, A]

1.4.4 Mutationsfaktorer

För att kontrollera till vilken grad en genetisk algoritm ändrar en individ i populationen använder man ofta olika grader på mutation ofta beskrivet i procent, denna procentenhet beskriver sannolikheten för att en individ antingen muterar den egna kromosomen eller inte Sivanandam och Deepa (2008).

1.4.5 Standard genetisk algoritm

En standard genetisk algoritm är ett begrepp som används i denna studie för att enklare skilja den genetiska algoritm vilket ska jämföras med en en-förälders algoritm. Denna algoritm är baserad på beskrivningar av genetiska algoritmer enligt Sivanandam och Deepa (2008). Den definierande faktorn vilket omfamnar detta begrepp är att den skapar nya individer i populationen genom korsning av två föräldrar. Mer information om de operationer som ingår i denna algoritm redovisas i avsnitt 2.2

1.4.6 En förälders genetisk algoritm

Det finns många definitioner av vad en en-förälder genetisk algoritm ämnar. I vissa studier används en-förälder genetisk algoritm synonymt med asexuell genetisk algoritm. I Cantó m. fl. (2009) beskrivs skillnaden mellan en asexuell genetisk algoritm och en standard genetisk algoritm vara avsaknad av korsning, den använder endast mutation. En annan studie implementerar en-förälders algoritmer med korsning men använder begreppet transposition då korsning som begrepp är vilseledande (Simões & Costa, 2000). Denna studie använder

begreppet för att beskriva en genetisk algoritm, närmast likt en standard genetisk algoritm, med kornsningsoperator, likt transposition, som skapar nya individer till populationen med endast en förälder istället för två.

1.5 Vetenskaplig förankring

Genetiska algoritmer är ett komplext område. Ett problem med en studie som ämnar undersöka implementationer av heuristiska sökalgoritmer är dess icke-garanterade precision under begränsade omständigheter. Detta kan göra det svårt att bevisa studiens resultat. Därför behöver studien bevisa validitet och reliabilitet. Johanesson och Perjons (2014) beskriver avsaknaden av validitet och reliabilitet som en fallgrop för studier. Uppfylls inte dessa två krav kan studien inte ge något vetenskapligt värde. För att undersöka denna domän måste studien därför vända sig till tidigare forskning inom ämnet. Detta med syfte att visa enhetlighet mellan tidigare forskning och denna studie.

Jämförelser mellan olika heuristiska sökalgoritmers nytta för att lösa problem så som handelseresandeproblem finns i överflöd. Çunkaş och Özsağlam (2009) jämför en genetisk algoritm med partikelsvärmoptimering och ser fördelar med båda algoritmerna. Studien beskriver nyttan med genetiska algoritmer som dess precision i att hitta globalt optimum. Den genetiska algoritmen var närmare globalt optimum än partikelsvärmoptimering. Li m. fl. (2008) jämförde en genetisk algoritm med mykolonioptimering och fann liknande slutsatser. Genetiska algoritmers nytta är dess precision. De förklarar även att nyttan minskade för handelseresandeproblem med fler än 30 städer beroende på vilket avslutningskrav som definierats.

I Simões och Costa (2000) skapas en genetisk algoritm som använder korsning inom en individs egna kromosom, det vill säga en genetisk algoritm som använder endast en förälder för att skapa nya lösningar, denna algoritm använder sig av asexuell transposition, transposition är en metod för korsning mellan två individer men denna algoritm gör det asexuellt vilket innebär att individen korsas med sig själv. Algoritmen testades på sju olika testfunktioner och alla test visade att asexuell transposition fungerade bättre än traditionell transposition i alla testfall. Då denna studie ämnar undersöka liknande förhållanden efterliknar den korsningsoperator vilket används i denna studie transposition med viss skillnad. En asexuell genetisk algoritm använder inte korsningsoperatorer utan endast mutationsoperationer. I Cantó m. fl. (2009) beskrivs en asexuell genetisk algoritm som har avsaknad av korsning mellan två föräldrar, för varje ny generation i algoritmen skapas istället en ny individ med genetiskt material från endast en förälder och behåller även föräldern till nästa generation ifall det skulle vara så att föräldern är bättre lämpad en barnet, algoritmen används för att lösa olika optimerings problem och blir även applicerad på problem inom astronomin. Studien fann att algoritmen hade fördelar gentemot en traditionell genetisk algoritm när det kommer till hur mycket uträkningar som krävs för att skapa en ny generation för att algoritmen inte använder dyrbara korsningsmetoder och utöver det så fann undersökningen att det oftast krävdes mindre antal generationer för att nå en optimal lösning. Denna typen av algoritm har implementerats och undersökts i flera användvändnings områden, i Amirghasemi och Zamani (2015) och Salesi m. fl. (2021) används en asexuell genetisk algoritm för implementation inom data mining respektive för att lösa ett kombinatoriskt optimeringsproblem, båda studierna visade lovande resultat. Det alla dessa studier om asexuell genetisk algoritm har gemensamt är att de finner positiva resultat både i jämförelse med genetiska algoritmer med två föräldrar och i olika implementationer inom olika ämnen. Dessa studier har valts som vetenskaplig förankring eftersom de belyser ett intressant ämne som fortfarande kräver mer forskning då algoritmerna ännu inte är optimala.

En annan viktig fråga är hur man ska hantera mutationsfaktor i genetiska algoritmer för att undvika lokala

optimum. I Hassanat m. fl. (2019) undersöks olika metoder för att välja mutations och korsningsfaktor i genetiska algoritmer. Utredarna skapade egna algoritmer och testade mot ett par fördefinerade algoritmer med statisk faktor på korsning och mutation, testet utfördes på olika varianter av handelsresandeproblemet. Av deras två egna algoritmer hade en av dem en hög mutationsfaktor som minskade dynamiskt och låg korsningsfaktor som ökade dynamiskt, och den andra var tvärtom (låg ökande mutationsfaktor och hög minskande korsningsfaktor), korsnings- eller mutationsfaktorn började på hundra procent och minskade till noll procent under körningstiden samtidigt som den andra höjdes. En dynamiskt ökande korsningsfaktor och minskande mutationsfaktor visade sig vara mer effektivt på stora populationer och en dynamiskt ökande mutationsfaktor och minskande korsningsfaktor visade sig vara mer effektivt på små populationer. Denna studie implementerade alltså dynamiska mutationsfaktorer på standard genetiska algoritmer. Resultatet är intressant men ändå ganska förväntat, en liten population innebär en mindre mångfald och i och med den ökande mutationen ökar mångfalden, och en stor population innebär att det redan finns en stor mångfald och den ökande korsningsfaktorn gör att föräldrarna skapar bättre lösningar genom att föröka sig. Tillsammans med studierna om asexuella genetiska algoritmer så utgör denna studien grundstommen i detta arbete då detta arbete ämnar att undersöka ett liknande ämne i en annan kontext, med en en-förälders-algoritm.

2 Metod

Detta kapitel ämnar att gå igenom och motivera vald forskningsstratergi samt metoder för insamling och analys av data samt överväganden av forskningsetiska aspekter och konsekvenser som kan komma till följd av arbetet.

2.1 Metodval

2.1.1 Forskningsstrategi

För att undersöka frågeställningen behövs en forskningsstratergi som under kontrollerade förhållande ger mätbara resultat (Johanesson & Perjons, 2014). Därför valdes experiment som forskningsstratergi för denna studie. Ett experiment beskrivs av Denscombe (2014) som en empirisk undersökning under kontrollerade tillstånd ämnad för att undersöka egenskaper och relationer mellan specifika faktorer. Johanesson och Perjons (2014) förespråkar att använda experiment för att bevisa eller motbevisa relationen mellan en faktor och ett utfall. För att mäta effekten av faktorerna används beroende och oberoende variabler. Det ger möjligheten att ändra de oberoende variablerna och undersöka vilken effekt det ger på de beroende variablerna. Experiment erhåller även isolation för dessa variabler som i sin tur ger det möjlighet till erhålla validitet av experimenten (Johanesson & Perjons, 2014).

I denna studie ska kandidatlösningarna som skapats undersökas för att nå bättre förståelse för vilken effekt linjärt minskande mutationsfaktor har på dem. Experiment ger möjligheten att jämföra flera resultat med linjärt minskande mutationsfaktor mot resultat från en standard genetisk algoritm. Experimenten använde därför resultatet av de genetiska algoritmerna som de beroende variablerna. Det användes för att se hur resultatet förändrades beroende på den oberoende variabeln, de genetiska algoritmerna.

2.1.2 Alternativ forskningsstrategi

En alternativ forskningsstrategi som undersöktes men inte valdes var fallstudie. Fallstudier används för att undersöka ett fenomen eller händelse i en specifik kontext (Johanesson & Perjons, 2014). Denscombe (2014) beskriver det som att upplysa det generella genom att undersöka det specifika. En forskare kan då undersöka hur två algoritmer presterar i en specifik kontext. Fallstudier kan alltså användas för att undersöka ett specifikt problem och vilka faktorer som påverkar problemet. Det ger nackdelen att en fallstudie inte kan generaliseras till andra problem. Då vår studie ämnade undersöka lösningar på generella kombinatoriska optimeringsproblem var metoden inte optimal för denna studie.

2.1.3 Datainsamlingsmetod

För att undersöka frågeställningen behövdes en datainsamlingsmetod där algoritmernas resultat kunde undersökas för att samla kvantifierbar data ämnat att besvara studiens frågeställning. Detta gav uppskov till att använda observationer. Observationer är en datasamlingsmetod där ett fenomen direkt observeras. Genom att observera resultatet av studien experiment kan en mer korrekt och detaljerad bild erhållas. Johanesson och Perjons (2014) beskriver observationer som användbart när forskningsstratergin är experiment. Specifikt valdes metoden systematisk observation.

Systematisk observation ämnar att förbättra pålitligheten, vilket är ett av de vanligaste problemen när det kommer till observationer (Johanesson & Perjons, 2014). Två personer kan tolka samma situation väldigt

olika, systematisk observation handlar om att hantera observationen objektivt istället för subjektivt, detta görs oftast med hjälp av ett observationsschema som är ett fördefinerat sätt att samla in händelser och interaktioner samtidigt som de sker (Johanesson & Perjons, 2014). I denna undersökningens fall passade systematisk observation bra eftersom data från körningen av algoritmerna samlas in och detta kan endast tolkas objektivt.

2.1.4 Alternativ datainsamlingsmetod

Det är viktigt att välja datainsamlingsmetod studiens syfte och frågeställning. Av de fem datainsamlingsmetoder som beskrivs i Denscombe (2014) är intervjuer, frågeformulär, fokusgrupper vanligtvis grundade i sociala studier som undersöker mänskliga faktorer. Datainsamlingsmetoden dokument används för att erhålla information genom redovisade källor. Frågeformulär övervägdes som datainsamlingsmetod, Denscombe (2014) beskriver frågeformulär som en metod för att samla in personers åsikter och upplevelser genom att använda en lista med nedskrivna frågor med relativt korta svar. Frågeformulär hade kunnat fungera om denna undersökningen hade kunnat utforma ett väl formulerat formulär och hittat ett bra urval av personer att delta, men denna process ansågs vara opraktiskt och datan som hade samlats in hade bestått av antingen fakta eller åsikter av personen som svarade på frågeformuläret (Denscombe, 2014), detta var inte av stort intresse och därav valde den här undersökningen ett mer praktisk angrepp som kan bidra med ny information.

2.1.5 Analysmetod

Observationer är bra för att samla in stora mängder kvantitativ data, denna datan kan omvandlas till siffror som sedan kan analyseras (Johanesson & Perjons, 2014). Inferentiell statistik är en kvantitativ analysmetodologi för att hitta mönster och relationer mellan olika datauppsättningar. Detta ger möjligheten till att generalisera fynden till ett större problemområde. Denna undersökning fokuserar på att jämföra skillnader på olika evolutionära algoritmer som samlas in med hjälp av systematisk observation och därför passade inferentiell statistik. Analysmetoden som valdes var Mann-Whitney U test. Testet är icke-parameteristiskt och används för att jämföra två stickprov av oberoende observerade populationer. Genom att mäta rangordningar av två olika dataset kan de observerade mätningarnas rangordningar undersökas, vilket ger ett u-värde, och bevisas signifikanta, vilket ger ett p-värde (Rey & Neuhäuser, 2011). Desto närmare noll u-värdet desto mer skillnad finns det mellan algoritmernas lösningar. Mann-Whitney U test fungerar väl med mindre stickprov. Testet är icke-parameteristiskt vilket innebär att det inte påverkas av skev data, testet kräver inte normalfördelad data. Då experimentet undersöker två algoritmer vilket genererar 20 lösningar, per algoritm, oberoende varandra ger denna metod stor nytta. Då handelseresandeproblem har optimala lösningar finns det stora chanser att insamlad data inte är normalfördelad vilket gjorde Mann-Whitney U test optimalt för denna studie.

2.1.6 Alternativ analysmetod

Ett annat test som kunde använts i denna studie är t-test. T-test kunde användas för att testa om det fanns någon signifikant skillnad mellan de två algoritmerna, anledningen till detta är för att t-test fungerar bra på små undersökningar med provstorlek under 30 (Denscombe, 2014). I ett T-test kalkyleras medelvärdet och standardavvikelsen från två uppsättningar data för att bestämma om skillnaderna är slumpmässiga eller inte (Denscombe, 2014). Problemet med testet är att det krävde normalfördelning av data. Då förhoppningarna för detta experiment var att algoritmernas lösningar kommer vara mycket nära optimal lösning kunde nor-

malfördelning av data avskrivas. Förutom inferentiell statistik finns även metoden deskriptiv statistik. Denna används för att beskriva och dra slutsatser för de dataset som experimenterats på (Johanesson & Perjons, 2014). Detta gör att resultatet inte kan användas för att dra slutsatser om populationer utanför testdatan. En dataanalys utförd med deskriptiv statistik är därför inte intressant i denna studie.

2.1.7 Forskningsetiska aspekter

När forskningspapper ska publiceras är det viktigt att överväga etiska och samhälleliga aspekter som kommer påverka forskningsprojektets helhet. Detta reflekterar hur metoderna kommer påverka både deltagare och samhället i sin helhet. Forskningsmetoderna som användes i denna studie involverade inga externa deltagare och inkluderade ingen känslig information. Därför kunde denna forskningen härledas som säker ur ett forksningsetiskt perspektiv.

2.2 Metodtillämpning

I detta experiment undersöktes resultaten av olika implementationer av genetiska algoritmer för att lösa handelsresandeproblemet. Avsikten med experimentet var att testa I vilka avseenden en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor kan skapa en bättre lösning än en standard genetisk algoritm. I experimentet användes algoritmerna som oberoende variabel och kandidatlösningarna som beroende variabel. Innan experimentet började behövdes flera aspekter bestämmas.

2.2.1 Handelsresandeproblemet

De problem som valts för implementationen är hämtade från TSPLIB (Reinelt, 2013) och inhämtades i XML format. Dessa problem är väl undersökta och det resulterar i att det existerar optimala lösningar till de problem som valts ut. Varje stad hade fördefinierade kostnader till alla andra städer i problemet. Den optimala lösningen blev då den minsta kostnaden för vandringen mellan första staden, alla andra städer och sedan tillbaka till den första staden.

Namn	**Städer**	**Optimal lösning**
burma14	14	3323
gr21	21	2707
brazil58	58	25395
ftv70	70	1950
kroA100	100	21282
bier127	127	118282
gr202	202	40160
lin318	318	42029
fl417	417	11861
pa561	561	2763

Tabell 1: *Redovisning av tio valda TSPLIB handelsresandeproblem*

2.2.2 Programeringsspråk

Programmeringsspråket som valdes för experimentet var Python. Det valdes med motivationen att det är en lättläst kod som tidigare använts av båda studenterna.

2.2.3 Populationsstorlek

För att generera individer till populationen fanns frågor kring storleken. Sivanandam och Deepa (2008) beskriver att möjligheten att uppnå globalt optimum är beroende av storleken på populationen. Större populationer ger enklare utforskande av sökutrymmet. Studier har underökt komplexiteten för en genetisk algoritm att konvergera och resultatet gav att det krävs O(nlogn) tid. Det ger en linjär ökning av tid då populationen blir större. Större population tar alltså längre tid att konvergera (Sivanandam & Deepa, 2008). Då menar Sivanandam och Deepa (2008) att rent praktiskt bör 100 individer vara en bra populationsstorlek. Därför valdes denna storlek för experimentet.

2.2.4 Kromosomrepresentation

Då problemen inhämtades av TSPLIB och redan hade en implementerad kromosomrepresentation fanns inget annat alternativ än det TSPLIB erbjöd (Reinelt, 2013). Implementationen är vägrepresentation det vill säga en lista med alla städer i den ordningen som de besöks, där varje stad representeras av ett heltal. Till exempel kommer en väg som går genom städerna i ordningen 5-4-3-1-2 representeras som [5, 4, 3, 2, 1] (Larranaga m. fl., 1999).

2.2.5 Lämplighetsfunktion

Då handelsresandeproblem i TSPLIB hade fördefinierade kostnader mellan städer och angav globala optimum på dess problem fanns det inget annat alternativ än deras lämplighetsfunktion (Reinelt, 2013). Deras funktion var endast addition av kostnaderna mellan alla städer i en lösning. Detta gav en enkel beräkning för att evaluera alla individer.

2.2.6 Selektionsmetod

Turneringsselektion valdes som strategi för selektion, denna metod håller en turnering för en slumpvist utvald delmängd ur populationen där den individ som är bäst enligt lämplighetsfunktionen kommer att segra, detta upprepas tills nästa generation är fylld med nya individer (Sivanandam & Deepa, 2008). Valet av turneringsstorlek sattes till fem procent av populationen vilket innebär fem individer per turnering.

2.2.7 Korsningsoperatorer

För att undvika faktorer som inte ämnades undersökas i denna studie valdes metoden tvåpunktskorning. Denna metod kunde implementeras både i en genetisk algoritm med en förälder och två. Från korsning genererades två individer enligt en standard genetisk algoritm (Sivanandam & Deepa, 2008). Valet att använda tvåpunktskorsning ger fördelar då en-förälders algoritmen med enpunktskorsning ger chanser att en av de skapade individerna är en exakt kopia av föräldern. Tvåpunktskorsning garanterar att individerna inte är exakta kopior. Nedan visas en grafisk representation för hur en förälder kan skapa två olika kandidatlösningar (barn) samt hur två föräldrar skapar två olika kandidatlösningar.

En-förälders tvåpunktskorsning

Förälder: [0, 1, 2, 3, 4, 5, 6, 7, 8]
Delar: [0, 1, 2] & [3, 4, 5] & [6, 7, 8]
Barn: [6, 7, 8, 0, 1, 2, 3, 4, 5] & [3, 4, 5, 0, 1, 2, 6, 7, 8]

Standard tvåpunktskorsning

Förälder: [0, 1, 2, 3, 4, 5, 6, 7, 8] & [8, 7, 6, 5, 4, 3, 2, 1, 0]
Delar: [0, 1, 2] & [3, 4, 5] & [6, 7, 8] & [8, 7, 6] & [5, 4, 3] & [2, 1, 0]
Barn: [0, 1, 2, 5, 4, 3, 6, 7, 8] & [8, 7, 6, 3, 4, 5, 2, 1, 0]

Korsningfaktor bestämdes till 1.0, Det garanterar att korsning alltid sker i alla fall, vilket motiveras med att föräldrarkandidatlösningar sparas och kan följa med till nästa generation om dess fitnessvärde är bättre än den nya genererade populationen. På detta vis finns det mindre chans för tapp av mångfald i populationen eller förlust av bra kandidatlösningar.

2.2.8 Mutationsoperatorer

Mutationsoperationen som implementerades i de genetiska algoritmerna var bytesmutation (swap mutation). Denna metod bytte plats på två slumpmässigt valda städer i två individer. En fördel med denna mutationsoperation var hur operationen gav möjligheten att även skapa två olika barnkandidatlösningar genom en förälderkandidatlösning, detta likt tidigare nämnda operationen transposition.

Mutationsfaktorn för algoritmen med en förälder började på 0.5 och minskade med 0.00002 per generation medans mutationsfaktorn för algoritmen med två föräldrar ligger konstant på 0.03 vilket är ett standardvärde på mutation enligt litteraturen (Hassanat m. fl., 2019). Detta innebär att mutationsfaktorn för algoritmen med en förälder kommer att vara 0 efter 25 000 generationer.

2.2.9 Avslutningskriterie

Det finns många olika metoder för att stoppa algoritmen och några av dessa listas i Sivanandam och Deepa (2008), den här undersökningen använder sig av en av dem, nämligen när en algoritm inte registrerar någon sänkning av lämplighetsvärdet (ökning av lämplighet) i någon av kandidatlösningarna på ett förutbestämt antal generationer (ingen förändring av lämplighet). Det förutbestämda antalet generationer är satt till 5000, detta på grund av tidsbegränsning.

2.2.10 Datainsamlingsmetod

Datainsamlingsmetoden som valdes var observation. Detta ger att resultatet av experimentet dokumenterades under körningen och kunde observeras när all exekvering genomförts. All önskevärd data exporterades till CSV-filer. Utöver informationen given av CSV-filer genererades även grafer vilket illustrerar algoritmernas beteende med generationer i x-led och lämplighetsvärde i y-led. Varför graferna valdes skildra generationer istället mutationsfaktor eller tid är pågrund av den information som det ger studien. Mutationsfaktorn är linjär och tiden är baserad på hårdvara och kodeffektivitet. Därför ger en visuell representation av standardalgoritmens prestation, på alla handelseresandeproblemen, en enklare förståelse för dessa algoritmers verkan.

2.2.11 Analysmetod

Experimentet genererade data som sedan undersöktes för att se hur algoritmerna presterade. Detta genomfördes genom Mann-Whitney U test. Ett sådant test kalkylerar rang för två uppsättningar data för att utföra ett statistiskt test av signifikans. Om det finns en signifikant skillnad mellan datauppsättningarna så visas detta genom att p-värdet är mindre än 0.05 och u-värdet representerar summan av rangpositioner för

den mindre av de två dataset. Det kan bevisa att resultatet inte är slumpmässigt (Rey & Neuhäuser, 2011). I den här undersökningens kördes de båda algoritmerna 20 gånger på varje problem och sen analyserades kandidatlösningarnas medelvärde och standardavvikelse. Verktyget som användes var Pythonbiblioteket SciPy funktion för Mann-Whitney U test. En anledning till att testet valdes var för att det fungerar bra för små undersökningar med dataset som är mindre än 30 (Rey & Neuhäuser, 2011). Frågeställningen kunde besvaras genom att undersöka sannolikheten av att skillnaderna på dessa resultat inte var slumpmässiga (Rey & Neuhäuser, 2011).

2.3 Forskningsetiska aspekter

Denscombe (2014) påpekar att forskare förväntas följa flera etiska principer för att erhålla forskningsetik. (i) Ingen som medverkat i studien ska lida pågrund av deras involvering. (ii) Involvering i studien ska vara frivilligt. (iii) Forskare ska vara öppna och ärliga till de involverade i studien. (iv) forskare står inte över lagen. De tre första punkterna är för denna studie inte relevanta. Studien använde inte persondata och involverade inte personer. Den sista punkten följdes genom att forskningen undvek olaglig eller skadlig verksamhet genom att se till att inget bröt mot någon lag eller skadade någon på något sätt. Konsekvensen blev att det inte fanns någon risk för att studien kunde anses som oetisk.

3 Resultat

Detta kapitel ämnar beskriva skapandet av de två genetiska algoritmerna, redovisning av resultatet för experiment som utförts, samt analys av resultat. Allt detta ger möjlighet till att besvara frågeställningen.

3.1 Utveckling av algoritmer

Med de inställningar som beskrivits i avsnitt 2.2 skapades två algoritmer, en en-förälders genetisk algoritm och en standard genetisk algoritm. Dessa två algoritmer kördes i samma miljö med samma programmeringsspråk, Python, och efterliknade varandra i den formen att alla bestämda icke-avvikande inställningar var identiska mellan de två algoritmerna. Endast i mutationsoperationen och korsningsoperationen fanns skillnader i koden.

De två genetiska algoritmerna skapades i Python där ett flertal funktioner representerade operationer vilket bearbetade de datapunkter som importerats från TSPLIB (Reinelt, 2013). Algoritmerna startades genom ett start-funktion som styrde alla operationer genom en loop. Funktionerna load_xml och load_graph användes för att importera handelseresandeproblemens datapunkter och distanser från xml-filer. De datapunkter som importerats användes sedan i funktionen populate_problems. Denna funktion använde de importerade städerna och skapade hundra initiala kandidatlösningar med slumpvis vald ordning. Vid skapandet av kandidatlösningar beräknades automatiskt lämplighetsvärdet genom att addera alla distanser mellan datapunkter. De kandidatlösningar som skapades sparades i en lista där kandidatlösningarna bearbetades under varje generation. Funktionen tournament_selection skapades för att selektera de kandidatlösningar med lägst fitnessvärde. Detta genom en loop som kördes antalet kandidatlösningar i populationsmängden gånger. För att modifiera de valda kandidatlösningar skapades funktionen two_point_crossover vilket använde de selekterade kandidatlösningarna erhållna genom tournament_selection. Listan med kandidatlösningar användes i funktionen för att slumpvis hämta en eller två kandidatlösningar, beroende på vilken algoritm som exekverade, för att skapa två nya kandidatlösningar (barn). Alla de nyskapade kandidatlösningarna sparades i listan med tidigare kandidatlösningar. Funktionen swap _mutation användes i båda algortimerna för implementera bytesmutation. Genom att slumpvis, per kandidatlösning, antingen låta sekvensen av datapunkter i kandidatlösningen vara oförändrad eller byta en datapunkt med en annan slumpvis vald kandidatlösning. När alla modifikationer på populationen är över användes tournament _selection återigen för att välja ut de bästa kandidatlösningarna som skulle vidare till nästa generations bearbetningar. Denna selektion gav möjligheten för populationen att undvika förluster av förälderkandidatlösningar som har bättre lämplighetsvärde än dess barnkandidatlösningar. I slutet av varje generation letades hela lista av kandidatlösningar igenom för att hitta det bästa lämplighetsvärdet och det sparas för jämförelse med nästa generations bästa lämplighetsvärde. Detta för att antingen inkrementera en variabel som innehåller antalet generationer utan kandidatlösning med bättre lämplighetsvärde eller nollställa denna variabel då nytt bästa lämplighetsvärde hittats. Slutligen inkrementerades en variabel som övervakade hur många generationer algoritmen genomgått.

I en while-loop, med konvergenskriterie enligt avsnitt 2.2.9, utfördes stegen i de genetiska algoritmerna. Nedan redovisas de steg algoritmerna genomförde under exekveringen.

Exekveringssteg

i Turneringsselektion

ii Tvåpunktskorsning

iii Bytesmutation

iv Turneringsselektion

v Bästa lämplighetsvärde sparas

vi Generationsindex inkrementeras

vii Om lämplighetsvärdet är det samma som det tidigare lämplighetsvärdet inkrementeras variabeln som kontrollerar while-loopen. Om lämplighetsvärdet inte är det samma ändrades variabeln till noll

Tio handelseresandeproblem undersöktes med de två algoritmerna och kördes tjugo gånger per problem. De inhämtade observationerna, i enighet med avsnitt 2.2, ger möjlighet till att besvara frågeställningen.

3.2 Algoritmernas lösningar

Namn	**Optimal lösning**	**En-förälders algoritm**	**Standard algoritm**
burma14	3323	3337	3468
gr21	2707	2876	3189
brazil58	25395	29641	37969
ftv70	1950	2700	3498
kroA100	21282	31294	40232
bier127	118282	171199	187012
gr202	40160	61804	67432
lin318	42029	95397	114274
fl417	11861	32361	81001
pa561	2763	6203	7816

Tabell 2: *Medelvärde av bästa lösning av 20 körningar på varje algoritm*

Ovan visas en tabell (Tabell 2) över körningarna av varje handelsresande problemen där varje problem redovisas med sin optimala lösning och medelvärdet för en-förälders och standard algoritmen, enligt lämplighetsfunktionen. Denna data är bearbetad för enklare redovisning. Rå-data för exekveringarna hittas i bilaga A.

Namn	**U-värde**	**P-värde**	**Resultat**
burma14	44.5	5.370195558179742e-06	Signifikant
gr21	58.0	6.200937257646842e-05	Signifikant
brazil58	11.0	1.707787846751859e-07	Signifikant
ftv70	2.0	4.586386355828241e-08	Signifikant
kroA100	3.0	5.322844918751244e-08	Signifikant
bier127	73.0	0.0003109987933734979	Signifikant
gr202	52.0	3.305223244351078e-05	Signifikant
lin318	34.0	3.788690529863132e-06	Signifikant
fl417	0.0	3.39780756408668e-08	Signifikant
pa561.tsp	2.0	4.586386355828241e-08	Signifikant

Tabell 3: *Mann-Whitney U-testresultat för 10 problem*

Ett Mann-Whitney U-test utfördes för att undersöka vilken algoritm som gav minst lämplighetsvärde och

om dess resultat var signifikant eller slumpartat. Mann-Whitney U-testet visade att en-förälders-algoritmen gav små u-värden för samtliga handelseresandeproblem. Detta kan visa på att en-föräldersalgoritmen presterat bättre än standardalgoritmen. I det här Mann-Whitney U-testet så kan man utläsa att p-värdet var mindre än 0.05 för samtliga handelsresandeproblem, vilket innebär att det finns en signifikant skillnad mellan algoritmerna i alla problemen.

För samtliga handelseresandeproblem redovisades lägre medianlämplighetsvärden för en-förälders-algoritmen. U-värdet för alla handelseresandeproblem bevisades väldigt låga. Ett lägre U-värde innebär att den första gruppen, det vill säga en-förälders algoritmen, generellt hade lägre lämpligehtsvärde för lösningarna. Genom p-värdet per test kunde resultaten av experimentet bevisas vara signifikanta för alla handelsresandeproblem. En tabell och grafer skapades för att visa skillnader i algoritmernas olika beteenden.

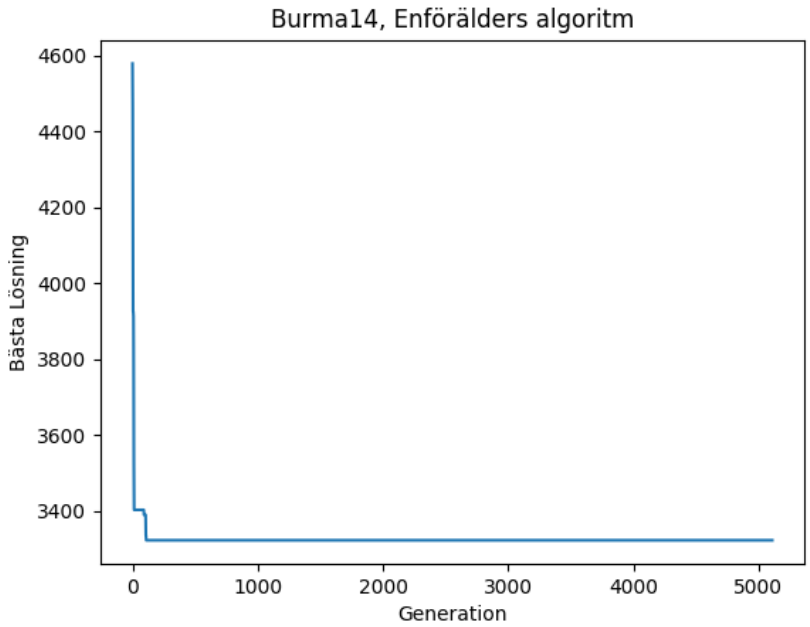
3.3 Algoritmernas beteende

Här ämnas resultatet av hur algoritmerna närmar sig konvergens och sedan konvergerar, när en bättre lösning inte hittas, undersökas. Tidigare har ett avslutningskriterie definierats i avsnitt 2.2.9. När en algoritm inte hittar en bättre lösning under femtusen generationer avslutades den. För att undersöka denna aspekt skapades därför en tabell över medelvärdet för antalet generationer som krävdes för konvergens. Utöver det genererades grafer. Detta för att visualisera hur algoritmerna hittar bättre lösningar under körningarna och hur mängden generationer ökar när algoritmen närmar sig konvergens. Graferna består av endast en körning per algoritm och problem.

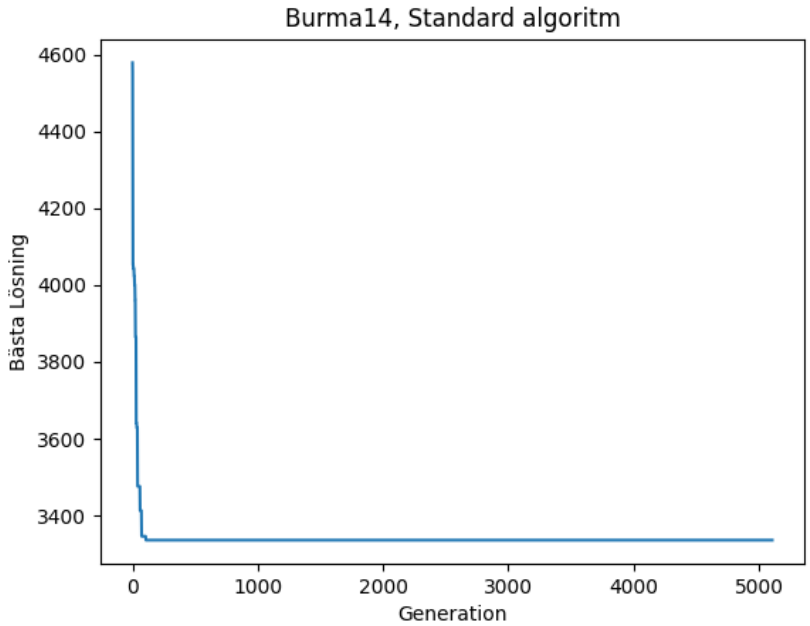
Namn	**En-förälders algoritm**	**Standard algoritm**
burma14	5484	5087
gr21	5395	5551
brazil58	13371	8697
ftv70	8096	8915
kroA100	13947	18652
bier127	10712	26085
gr202	10003	74298
lin318	13471	131432
fl417	17747	248828
pa561	24829	228983

Tabell 4: *Medelvärde av antal generationer som krävdes för att nå fram till en slutgiltig lösning av 20 körningar på varje algoritm*

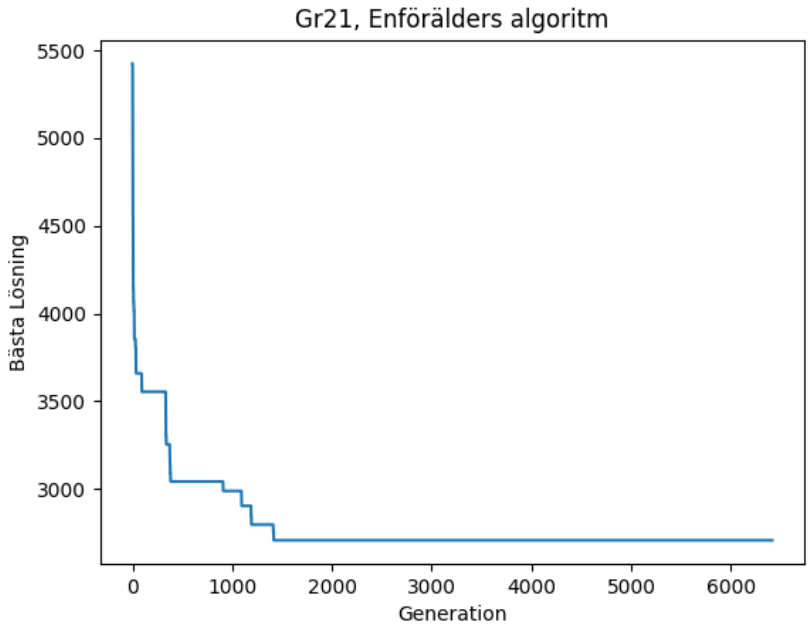
Tabell 4 visar medelvärdet för hur många generationer som krävs för konvergens. För handelseresandeproblemen burma14 och brazil58 konvergerade standard algoritmen tidigare än enförälders-algoritmen. För de andra problemen konvergerade en-förälders-algoritmen tidigare. För problem burma14 och gr21 finns det en marginell skillnad mellan medelvärdet för konvergens medans för problem bier127, gr202, lin318, fl417, pa561 var skillnaden mellan medelgeneration markant. pa561 visar att det tog runt 25000 generationer för en-förälders algortimen medans det tog runt 228000 generationer för standard algortimen. Det genomsnittliga värdet var som tidigare beskrivet mindre för en-förälders-algoritmen vilket gör det noterbart att tabellen visar att lösningarna oftast hittades tidigare än för standard algoritmen.



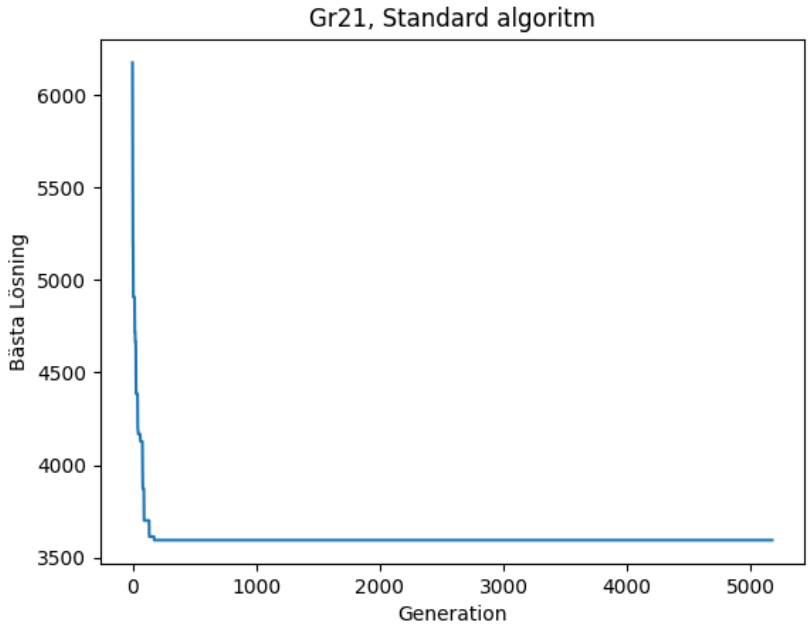
Figur 2: *Visuell representation av en-förälders algoritmens exekvering av burma14*



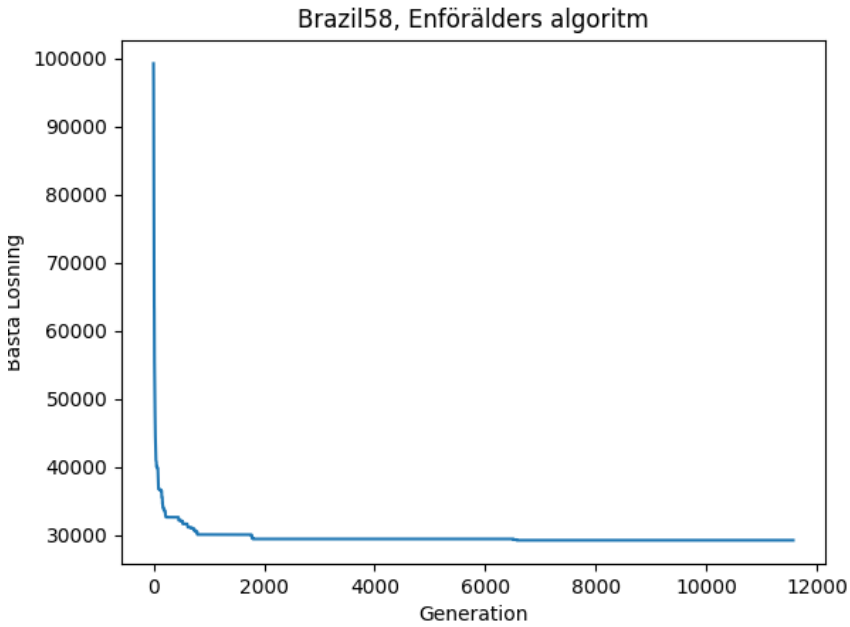
Figur 3: *Visuell representation av standard algoritmens exekvering av burma14*



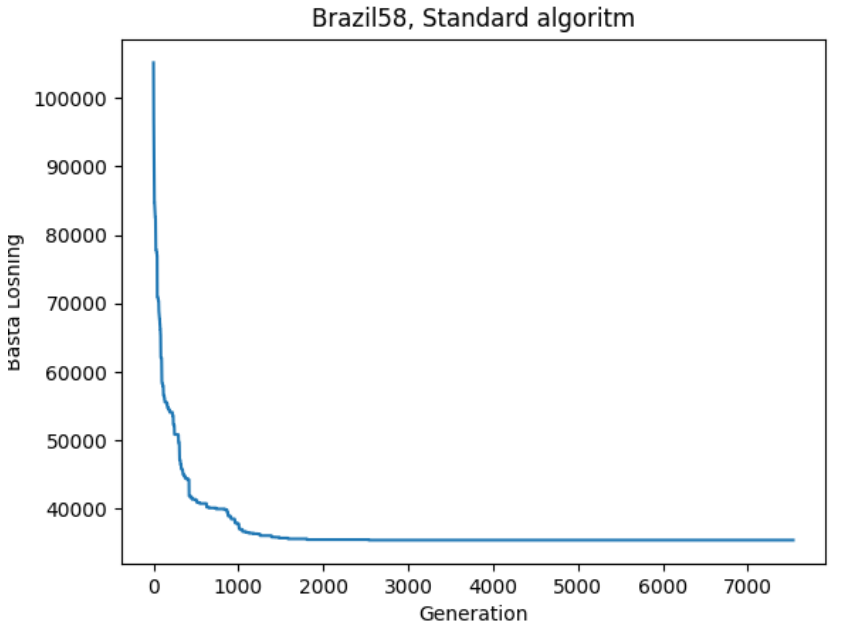
Figur 4: *Visuell representation av en-förälders algoritmens exekvering av gr21*



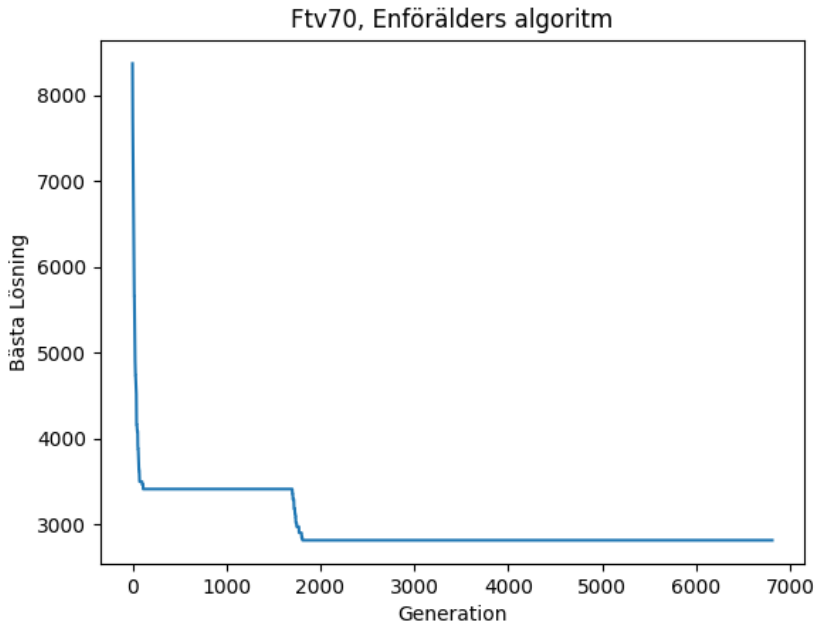
Figur 5: *Visuell representation av standard algoritmens exekvering av gr21*



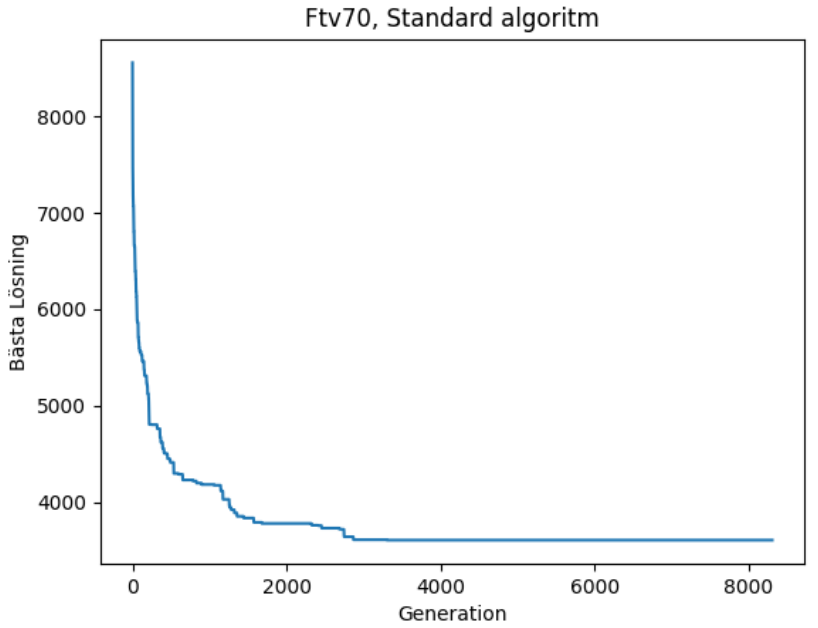
Figur 6: *Visuell representation av en-förälders algoritmens exekvering av brazil58*



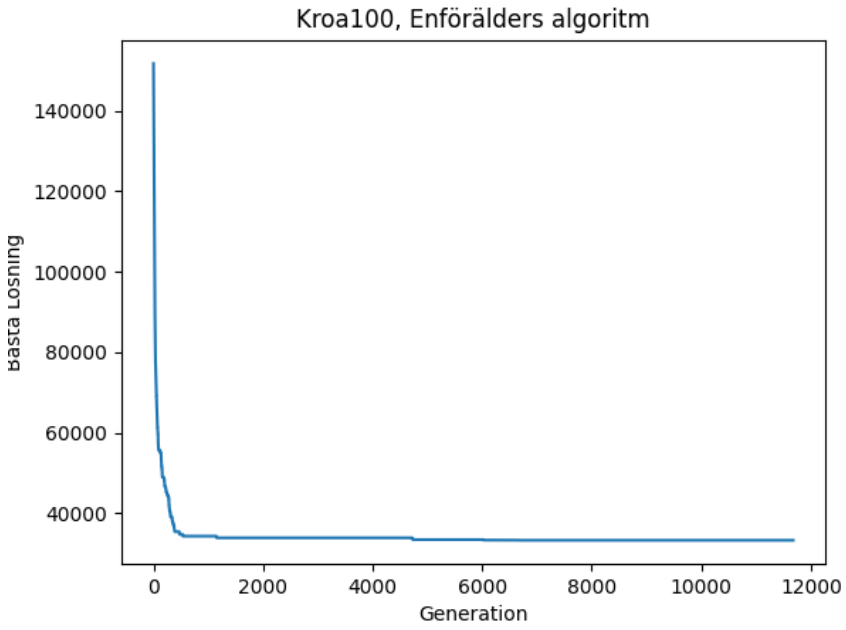
Figur 7: *Visuell representation av standard algoritmens exekvering av brazil58*



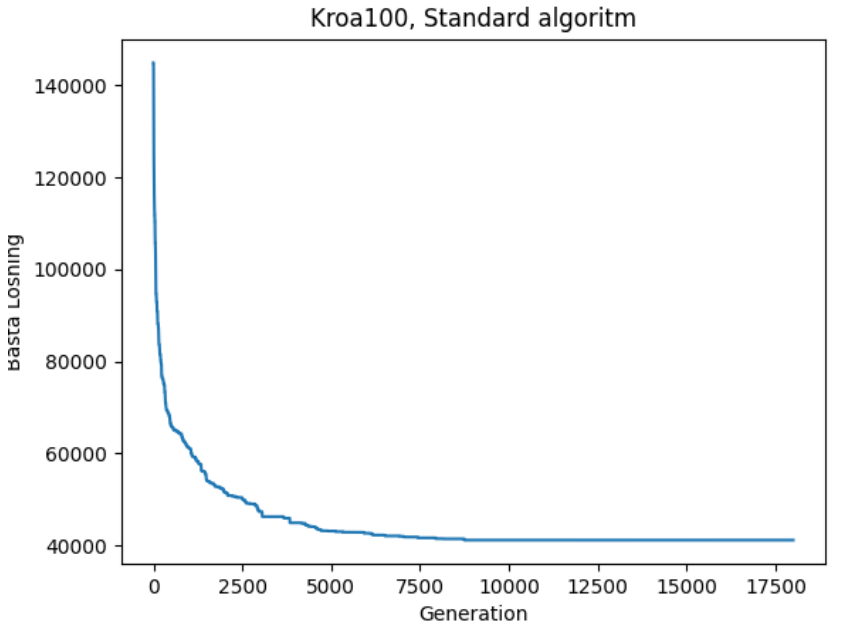
Figur 8: *Visuell representation av en-förälders algoritmens exekvering av ftv70*



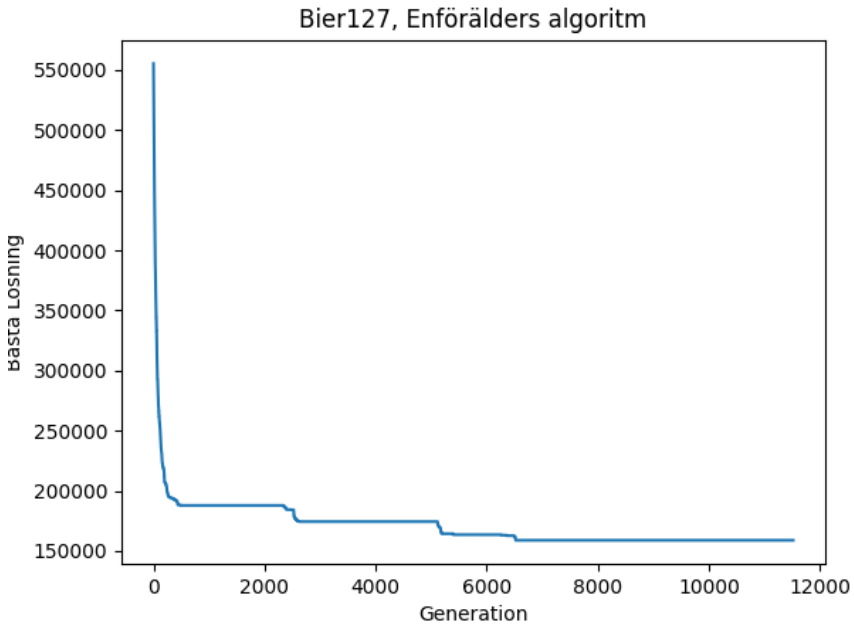
Figur 9: *Visuell representation av standard algoritmens exekvering av ftv70*



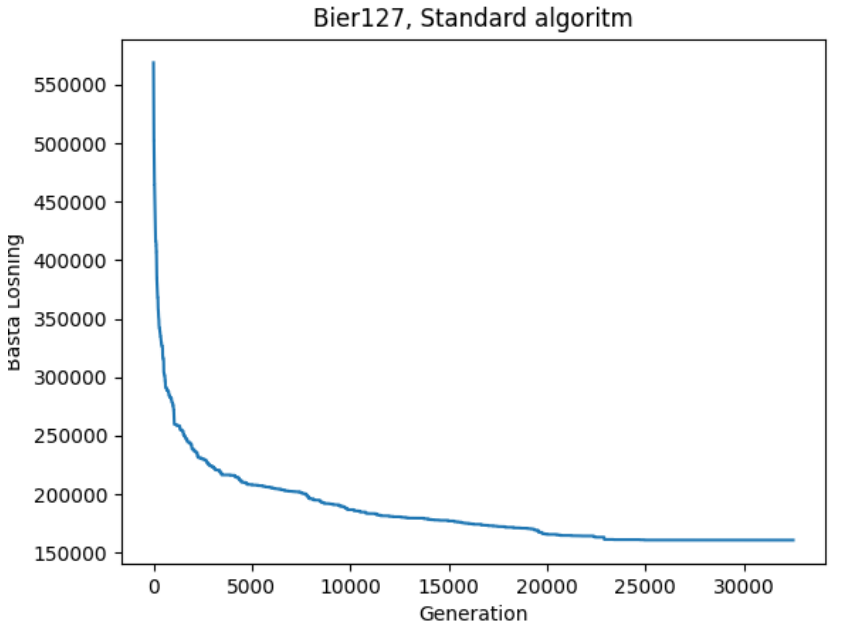
Figur 10: *Visuell representation av en-förälders algoritmens exekvering av kroa100*



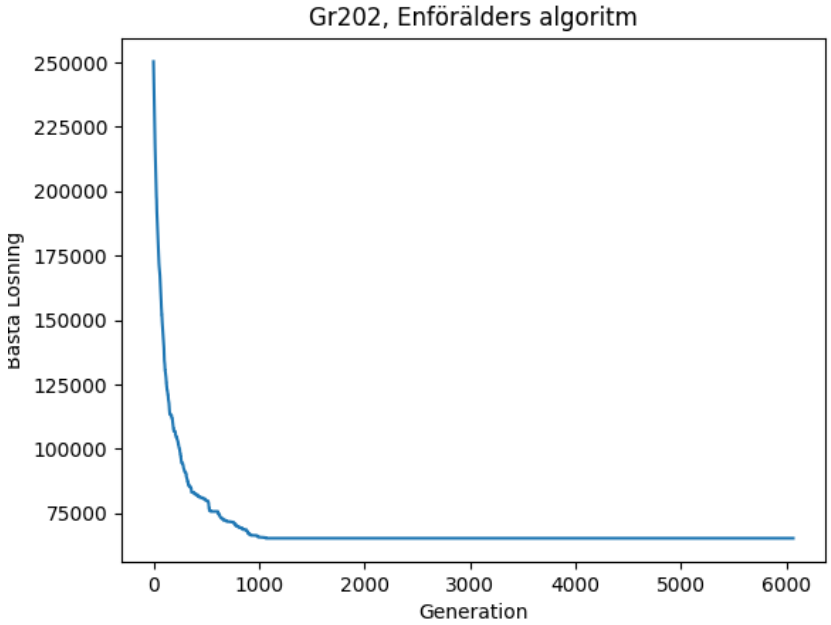
Figur 11: *Visuell representation av standard algoritmens exekvering av kroa100*



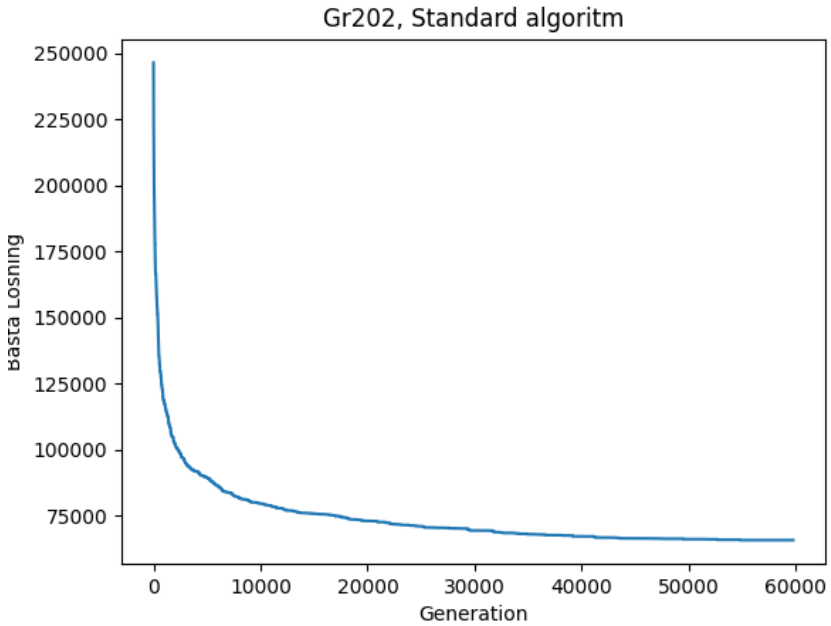
Figur 12: *Visuell representation av en-förälders algoritmens exekvering av bier127*



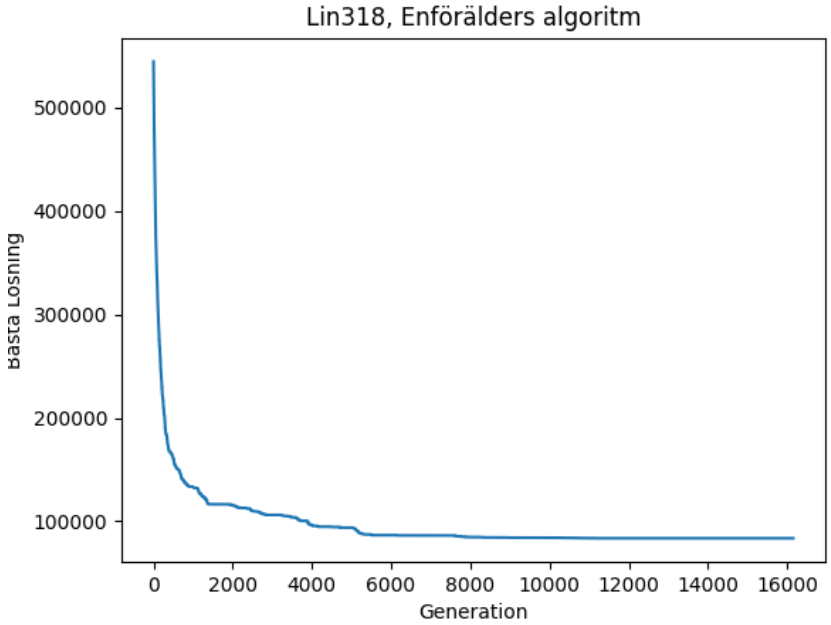
Figur 13: *Visuell representation av standard algoritmens exekvering av bier127*



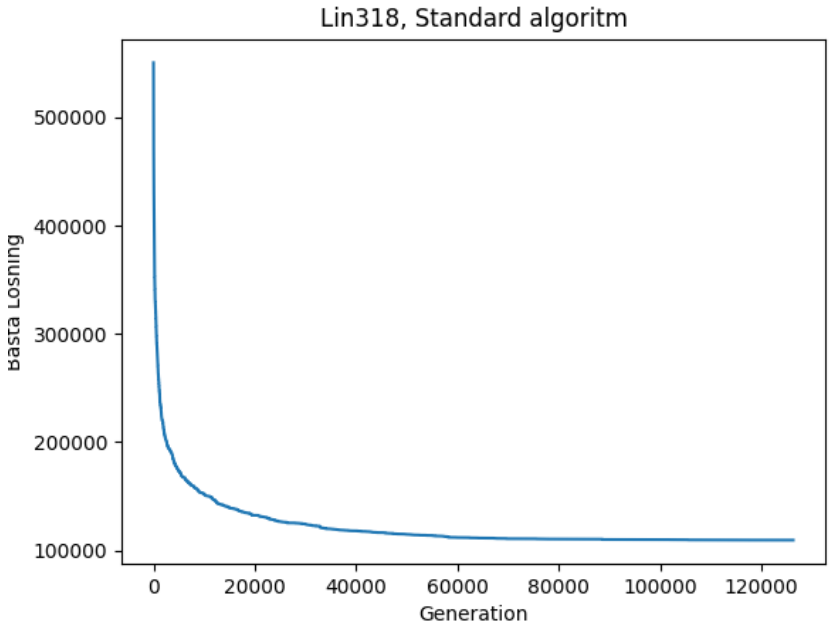
Figur 14: *Visuell representation av en-förälders algoritmens exekvering av gr202*



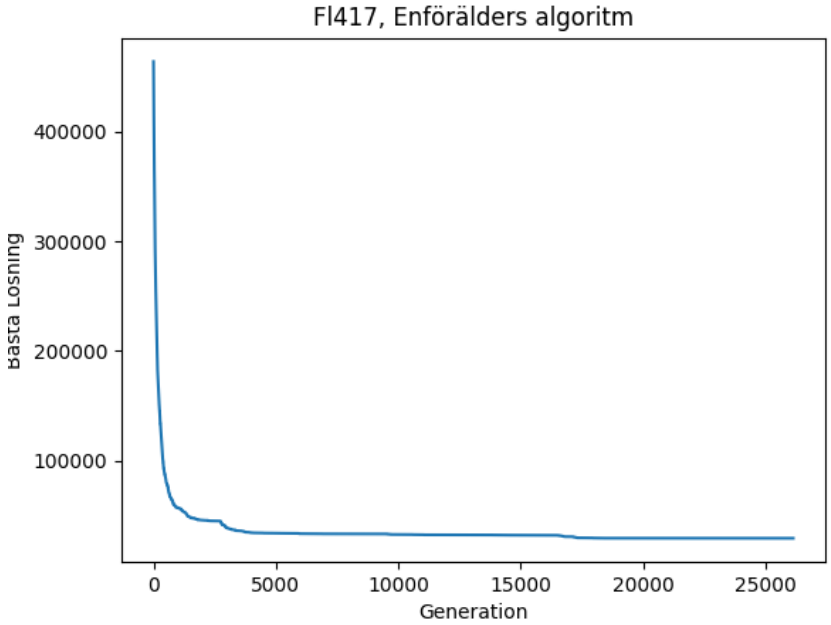
Figur 15: *Visuell representation av standard algoritmens exekvering av gr202*



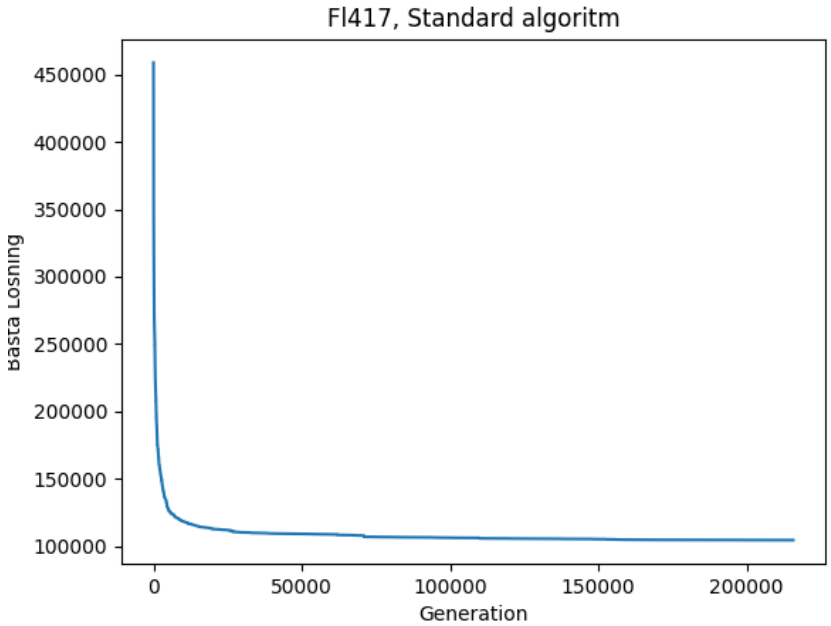
Figur 16: *Visuell representation av en-förälders algoritmens exekvering av lin318*



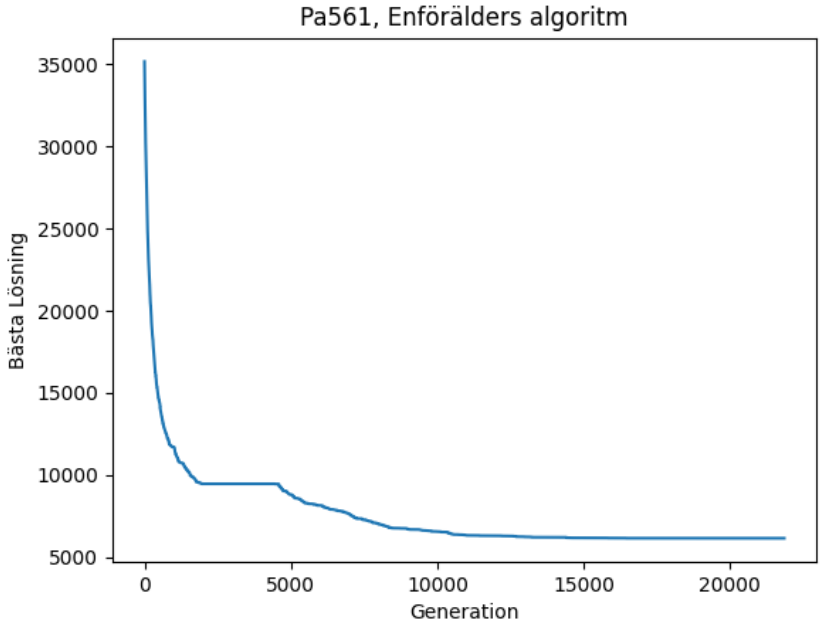
Figur 17: *Visuell representation av standard algoritmens exekvering av lin318*



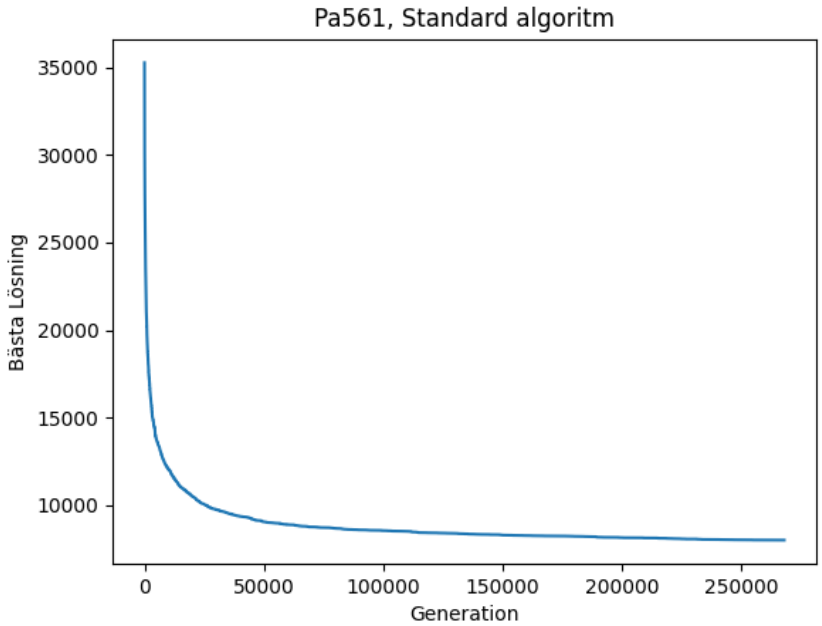
Figur 18: *Visuell representation av en-förälders algoritmens exekvering av fl417*



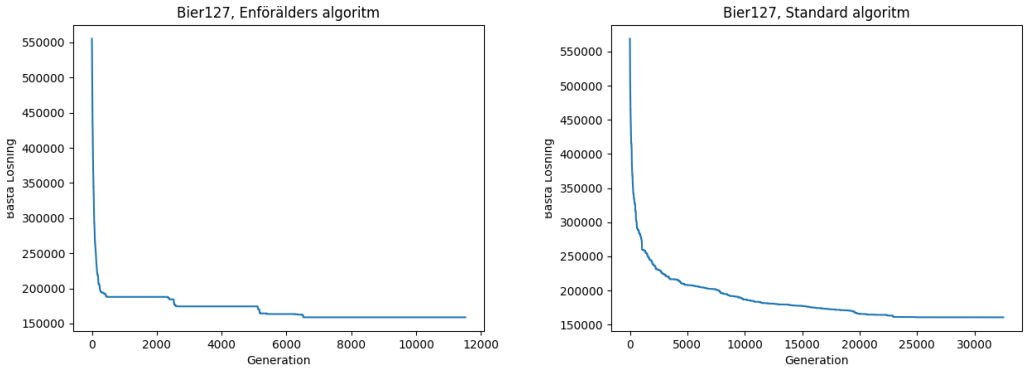
Figur 19: *Visuell representation av standard algoritmens exekvering av fl417*



Figur 20: *Visuell representation av en-förälders algoritmens exekvering av pa561*



Figur 21: *Visuell representation av standard algoritmens exekvering av pa561*



Figur 22: *Bier127 visar att algoritmerna erhåller olika konvergenskurvor*

Samtliga grafer visar på att de två algoritmerna hittade bättre lösningar med olika beteenden. För handelseresandeproblem med mindre än 70 städer illustrerar graferna att algoritmernas beteende liknar varandra. Då graferna efterliknar varandra finns bevis att standard genetiska algoritmens mutationsfaktor verkade under bättre förhållanden, alltså fungerade korsningsoperatorn och mutationsoperatorn effektivt för att lösa problem under färre än 70 städer. En-förälders algoritmen, vilket efterliknade standard algoritmen, visade på samma fördelar. Notera att en-förälders algoritmen hittar mer precisa lösningar vilket bevisats signifikant genom Mann-Whitney U test.

Två-föräldrarsalgoritmens grafer illusterade för handelseresandeproblem till och med mer än 70 städer en rundare kurva. En mjuk kurva tyder på låg mutationsfaktor, vilket implementerades, konsekvensen av detta ledde till gradvis och jämn konvergens vilket krävde fler generationer och missade bättre lämplighetsvärden. Grafen för en-föräldrars algoritmen efterliknade mer en trappa. Detta kan förklaras med högre mutationsfaktor och endast en förälder i korsningsoperatorn. Konsekvensen av dessa inställningar gav en algoritm som tidigt introducerade en större mångfald av lösningar som sedan undersöks, korsas och muteras för att snabbare hitta bättre lösningar.

4 Diskussion och Slutsats

4.1 Diskussion

Denna studie har grundats i domänen evolutionära algoritmer men specifikt i genetiska algoritmer. Genom att undersöka en en-förälders-genetisk algoritm med dynamisk mutationsfaktor och jämföra den med en standard genetisk algoritm finns utrymme till att bidra till denna forskningsdomän. Figur 1 till och med figur 20 erbjuder en grafisk representation där man klart och tydligt kan se hur konvergenskurvorna skiljer sig mellan de två algoritmerna, dessa grafer kompletterades med tabeller med genomsnittliga värden som erhölls av 20 körningar på varje algoritm och problem. En-förälders algoritmen, till skillnad från standard algoritmen, hittar lägre lämplighetsvärde tidigare. Denna skillnad illustrerar hur den linjärt minskande mutationsfaktorn tidigt förbättrade undersökningsförmågan av en-föräldersalgoritmen. Resultaten visade att en en-föräldersalgoritmen gav bättre resultat än en standard genetisk algoritm i alla testfall. Det man kan utläsa av tabell 2 från avsnitt 3.2 är att en-föräldersalgoritmen gav bättre medellämplighetsvärde på tjugo körningar i alla instanser av handelsresandeproblemen, data från körningarna redovisas också i större detalj i bilaga A och B. Testerna som utfördes (Tabell 3) visar att en-föräldersalgoritmen skapade mer precisa lösningar i alla testfall, det kunde bevisas genom u-värden och p-värden givet Mann-whitney U test. Vidare så kunde dessa skillnader visas vara signifikanta genom att alla test visade ett p-värde under 0.05. Tabell 4 visar att en-föräldersalgoritmen kräver mindre antal generationer i alla handelsresandeproblem förutom två (burma14 och brazil58), detta visar på att en-föräldersalgoritmen finner en slutgiltig lösning snabbare med de angivna avslutningsvilkoren. Tidigare forskning har visat att dynamiska mutationsfaktorer kan hjälpa genetiska algoritmers prestanda, dock har detta implementerats med standard genetiska algoritmer (Hassanat m. fl., 2019). Deras slutsats är att dynamiskt minskande mutationsfaktor är bättre för större handelseresandeproblem och dynamiskt ökande mutationsfaktor är bättre för mindre handelseresandeproblem. Detta undersöktes inte specifikt i vår studie, det finns en möjlighet till att det även kan stärka våra resultat då studien efterliknar liknande vissa implementationer, linjärt minskade mutationsfaktor, i vår studie och slutade i liknande resultat. Det minsta u-värdet som observerades av för handelseresandeproblemet fl417 med 0, vilket var det näst största handelseresandeproblemet. Även det största handelsresandeproblemet pa561 resulterade i lågt u-värde. Detta visar på att resultatet för studien är till mestadels överens med Hassanat m. fl. (2019). Resultatet från denna studien stärker tidigare resultat från studier som har undersökt en-förälders genetiska algoritmer (Cantó m. fl., 2009; Simões & Costa, 2000) och har samtidigt bidragit med ytterligare en potentiell förbättring som inte har undersökts tidigare nämligen en linjärt minskande mutationsfaktor.

4.1.1 Framtida forskning

Ytterligare forskning krävs för att se hur optimeringen av mutationsfaktorn kan ge ännu bättre resultat. Till exempel kan efterföljande studier fokusera på att utveckla och testa den här studiens algoritm mot en liknande med en adaptiv mutationsgrad eller statisk mutationsgrad. Den här studien har tydligt visat att det går att skapa bättre och mer optimerade lösningar ur ett avståndsperspektiv genom att använda den föreslagna algoritmen jämfört med att använda en standard genetisk algoritm, men studien har haft begränsningar i form av tid och resurser, detta har påverkat studiens omfattning genom att handelsresandeproblemen som användes för att testa algoritmerna valdes på grund av storlek, efterföljande studier kan därför utöka denna studien med att använda större problem-storlekar för att ytterligare validera algoritmens prestanda. En intressant studie som bygger på dessa resultat är i vilka avseenden en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor kan skapa en bättre lösning än en en-förälders genetisk algoritm med

statisk mutationsfaktor. Detta kan hjälpa undersöka om det finns någon förbättrande och signifikant skillnad mellan dessa algoritmer.

4.1.2 Etiska och samhälleliga konsekvenser

Genetiska algoritmer är en form av artificiell intelligens som är ett kontroversiellt ämne i dagens samhälle. Människor runt om i värden blir allt mer beroende av datorer för att utföra dagliga sysslor och det finns en oro för att vissa arbetsroller kan automatiseras med hjälp av artificiell intelligens. Därför så är det viktigt att resultatet från den här studien inte missbrukas på något sätt för att skada någon människa eller grupp av människor. Positiva samhälleliga konsekvenser påverkas genom implementationer inom vård, produktdesign, finans med mer. Då en ökad precision ger mer tillit till algoritmen kan mer komplexa domäner förbättras.

4.2 Begränsningar

För att ge denna studie transparens och trovärdighet bör studiens begränsningar kritiskt granskas.

4.2.1 Validitet och Reliabilitet

Mäter vi det som frågeställningen kräver och hur bra mäter vi det? Den frågan undersöks av validitet (Hjelm m. fl., 2014). Frågeställningen kräver ett experiment som undersöker i vilka avseenden en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor kan skapa en bättre lösning än en standard genetisk algoritm. I denna studie användes den formuleringen för att undersökta de två algoritmernas lägsta lämplighetsvärde och på så sätt se till vilken grad algoritmerna presterar jämt emot varandra. För att undvika jämförelsen av icke-intressanta aspekter av dessa algoritmer definierades alla inställningar de båda algoritmerna delade. Tydligt förklarades de skillnader som ämnades undersökas. Dessa var den beroende variabeln lämplighetsvärdet samt den oberoende variabeln de genetiska algoritmerna, alltså antal föräldrar och mutationsfaktorn. Denna studie har alltså beskrivit alla skillnader och likheter mellan de två algoritmerna och kan på så sätt höja validiteten. Då validitet är svårt att mäta kan detta bedömas med tidigare forskning som stödjer den mätning studien resulterade i. Denna studies resultat efterliknar de tidigare studierna.

Reliabilitet ämnar mäta kvaliteten på studiens mätningar. Hjelm m. fl. (2014) beskriver att reliabiliteten är bevisad om en eller flera andra undersökningar kommer fram till samma resultat oavsett vem som undersöker. Reliabilitet kan alltså stärkas genom tidigare forskning. Då denna studie undersöker heuristisk sökoptimering kan detta vara svårt att replikera exakt samma resultat utan studiens egenskrivna algoritmer. Då informationen om optimal lösning existerar för alla handelseresandeproblem i TSPLIB gick det att jämföra de skapade algoritmernas bästa lösning med dessa. TSPLIB kan då användas för att jämföra kvaliteten på studiens mätningar. Då flera handelseresandeproblem resulterade i optimal lösning stärker det studiens reliabilitet. Då tidigare forskning drar samma slutsatser om algoritmerna stärker det denna studie. När det kommer till de instrument som har använts för att erhålla resultatet så måste både mänskliga och tekniska faktorer räknas in, resultatet kan ha påverkats av implementationen av algoritmerna och efterföljande test, det kan även ha påverkats av de tekniska instrument som användes för att räkna ut resultatet, detta i sin tur kan ha en inverkan på reproducerbarhet och reliabilitet. Därför var den här undersökningen noga med att följa de steg som redovisades i avsnitt 2.2 i ett försök till att undvika att resultatet skulle kunna påverkas av dessa faktorer. Genom att vara medveten om dessa faktorer och se till att vidta åtgärder för att undvika

deras inverkan på resultatet strävade studien efter att upprätthålla reliabilitet. Det är viktigt att tolka dessa resultat med en viss försiktighet då dessa faktorer kan ha påverkat studiens reliabilitet.

4.2.2 Metodreflektion

Forskningsstrategin, datainsamlingsmetoden och analysmetoden som valdes var väl ämnade för vad studien hade för avsikt att undersöka och utgjorde tillsammans en bra grund för att arbetet skulle kunna få ett intressant resultat. Däremot så finns det en del saker i tillämpningen av metoden som hade kunnat gjorts på ett annat sätt, en potentiell brist i tillämpningen kan ha berott på utredarnas oerfarenhet inom evolutionära algoritmer och kan ha påverkat tidskomplexiteten hos algoritmerna, detta i sin tur kan ha påverkat hur många olika handelsresandeproblem och hur stora handelsresandeproblem som kunde undersökas.

En brist, vilket implementerat skulle ge en större reproducerbarhet, är de egenskrivna algoritmerna. Det existerar Pythonbibliotek vilket tillhandahåller färdigskrivna genetiska algoritmer med ett flertal inställningar till sitt förfogande. Fördelen med denna implementation är den garanterade korrektheten av koden. Biblioteket användes dock inte, då den saknade inställningar som krävdes för den förespråkade en förälders genetiska algoritmen.

Konvergenskriteriet som valdes reflekterar en brist i denna studie. Som beskrivet i avsnitt 2.2.9 sattes ett förutbestämt avslutningskriterie på 5000 generationer utan förbättring av lämplighetsvärdet. Konsekvensen av detta medför att algoritmerna fortfarande har risken att fastna i lokalt optimum. Det kunde lösas med ett ännu högre avslutningskritere. Problemet med högre avslutningskriterie är exekveringstiden och prestandan på den dator, i och med tidsspannet denna studie skapades under, som utförde exekveringen. Med en bättre dator kunde exekveringar ske snabbare och med det skulle avslutningskriteriet möjligen höjas. Resultatet av detta är inte undersökt men troligen skulle algoritmerna ytterligare närma sig globalt optimum. Alternativ kunde ett adaptivt avslutningskriterie introduceras, där avslutningskriteriet är relativt till antalet städer i handelseresandeproblemet.

En potentiell brist i studien är att Python biblioteket SciPys Mann-Whitney U test metod inte kontrollerades med testvärden innan dataanalysen. Genom att kontrollera biblioteket innan de riktiga testen utförs så kan man säkerställa att implementationen fungerar korrekt.

En brist med detta arbete, som kunde gett denna studie mer insikt, är tidsbristen som påverkade hur många algoritmer som borde skapats. Det finns intresse att jämföra dessa algoritmer med en tredje en-föräldersalgoritm med statisk mutationsfaktor. Denna algoritm hade ökat kunskapen om hur en-föräldersalgoritmens minskande mutationsfaktor förhåller sig till en en-förälders algoritm med statisk mutationsfaktor.

4.2.3 Reproducerbarhet

Denscombe (2014) beskriver att en stor fördel med experiment är dess reproducerbarhet. Genom att dokumentera alla relevanta variabler involverade i experimentet stärks detta. I avsnitt 2.2 beskriver studien tydligt de inställningar vilket definierar de två algoritmerna. Det hjälper framtida forskare reproducera studiens experiment. Då heuristiska sökalgoritmer inte följer specifika exekveringsspår resulterar det i att även om experimentet reproduceras finns det låg sannolikhet att exakt samma resultat kan finnas. Med denna transparens är det endast givet att framtida forskares resultat efterliknar denna studies resultat.

4.2.4 Generaliserbarhet och vidareförbarhet

Denna studie undersökte implementationen av en en-förälder genetisk algoritm med linjärt minskande mutationsfaktor. Definitionen av algoritmen ligger i dess inställningar. Det heuristiska sökproblemet som använts för att undersöka lämpligheten av de två implementationerna av genetisk algoritm var handelseresandeproblemet. Studien har bevisat att inom dess kontext (inställningar) finns det fördelar med att använda en en-förälders algoritm. Generellt kan denna algoritm användas för att lösa alla heuristiska sökproblem vilket kan definiera en lämplighetsfunktion. Då all relevant information kring inställningar och kontext är väl definierade finns utrymme föra denna studies resultat och slutsatser vidare till framtida forskning

4.2.5 Trovärdighet

Denna studie är av kvantitativ natur vilket Denscombe (2014) beskriver som den mest trovärdiga förhållningssättet till forskning. Denna studie har redovisat all relevant data och hur den var insamlad samt beskrivit i detalj hur dess data bearbetats.

4.3 Slutsats

Denna studie undersökte i vilka avseenden en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor kan skapa en bättre lösning än en standard genetisk algoritm?. Syftet var att undersöka om den linjärt minskande mutationsfaktorn kunde bidra till att skapa bättre kandidatlösningar i optimeringsproblem gentemot en standard genetisk algoritm.

För att besvara frågeställningen *I vilka avseenden kan en genetisk algoritm med en förälder med linjärt minskande mutationsfaktor skapa en bättre lösning än en standard genetisk algoritm?* utfördes tjugo experiment på tio handelsresandeproblem med två olika genetiska algoritmer. Analysen visade att i alla problem gav en-förälders-algoritmen med dynamisk mutationsfaktor signifikant bättre resultat. Skillnaden i medelvärde var av störst intresse för den här uppsatsen. Det visade sig vara lägre för en-förälders algoritmen i alla testfall. Resultatet visades vara att den föreslagna algoritmen skapade bättre optimerade lösningar jämfört med standard algoritmen. Den visades alltså vara mer effektiv. Detta bevisades genom att alla undersökta handelseredandeproblems slutliga lämplighetsvärden som undersöktes var statistiskt signifikanta genom Mann-Whitney U test. Då en-förälders algoritmen skapade bättre optimerade lösningar och undvek lokalt optimum medans standard algoritmen oftare fastnade i lokalt optimum, bevisade algoritmen att den mer effektivt bevarade mångfalden av en genererad population av kandidatlösningar. Genom att gradvis minska mutationsfaktorn och skapa nya kandiatlösningar genom asexuell korsning bevarades flera olika varianter av kandidatlösningar. Detta gav mindre chans att fastna i lokalt optimum. Till sist visade den föreslagna algoritmen ge bättre kostnadsseffektivitet. Eftersom algoritmen skapade bättre optimeringslösningar under färre generationer krävdes mindre kostnader för att lösa problemen.

Referenser

Amirghasemi, M., & Zamani, R. (2015). An effective asexual genetic algorithm for solving the job shop scheduling problem. *Computers Industrial Engineering*, *83*, 123–138. https://doi.org/10.1016/j.cie.2015.02.011

Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2006). *The Traveling Salesman Problem: A Computational Study*. Princeton University Press. https://doi.org/10.1515/9781400841103

Cantó, J., Curiel, S., & Martínez-Gómez, E. (2009). A simple algorithm for optimization and model fitting: AGA (asexual genetic algorithm). *501*, 1259–1268. https://doi.org/10.1051/0004-6361/200911740

Çunkaş, M., & Özsağlam, M. Y. (2009). a comparative study on particle swarm optimization and genetic algorithms for traveling salesman problems. *Cybernetics and Systems*, *40*(6), 490–507. https://doi.org/10.1080/01969720903068435

Denscombe, M. (2014). *The Good Research Guide for Small-scale Research Projects*. Open University Press/McGraw-Hill Education.

Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer. https://doi.org/10.1007/978-3-662-44874-8

Hassanat, A., Almohammadi, K., Alkafaween, E. A., Abunawas, E., Hammouri, A., & Prasath, V. S. (2019). Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, *10*(12). https://doi.org/10.3390/info10120390

Hjelm, M., Lindgren, S., & Nilsson, M. (2014). *Introduktion till samhällsvetenskaplig analys*. Gleeprups Utbildning AB. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A709861&dswid=-2906

Holland, J. H. (1992). Genetic Algorithms, 297–314. https://www.jstor.org/stable/10.2307/24939139

Johanesson, P., & Perjons, E. (2014). *An Introduction to Design Science*. Springer. https://link.springer.com/book/10.1007/978-3-319-10632-8

Katoch, S., Chauhan, S. S., & Kumar, V. (2020). A review on genetic algorithm: past, present, and future. *Multimed Tools Appl*, 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., & Dizdarevic, S. (1999). Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial intelligence review*, *13*, 129–170. https://doi.org/10.1023/A:1006529012972

Li, K., Kang, L., Zhang, W., & Li, B. (2008). Comparative Analysis of Genetic Algorithm and Ant Colony Algorithm on Solving Traveling Salesman Problem. *IEEE International Workshop on Semantic Computing and Systems*, 72–75. https://doi.org/10.1109/WSCS.2008.11

Reinelt, G. (2013). *TSPLIB*. Universität Heidelberg. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html

Rey, D., & Neuhäuser, M. (2011). Mann-Whitney Test. I M. Lovric (Red.), *International Encyclopedia of Statistical Science*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-04898-2_616

Salesi, S., Cosma, G., & Mavrovouniotis, M. (2021). TAGA: Tabu Asexual Genetic Algorithm embedded in a filter/filter feature selection approach for high-dimensional data. *Information Sciences*, *565*, 105–127. https://doi.org/10.1016/j.ins.2021.01.020

Simões, A., & Costa, E. (2000). Using Genetic Algorithms with Asexual Transposition. *Proceedings of the 2000 Congress on Evolutionary Computation*, *2*, 1196–1203. https://doi.org/10.1109/CEC.2000.870785

Sivanandam, S. N., & Deepa, S. N. (2008). *Introdution to Genetic Algorithms*. Springer. https://doi.org/10.1007/978-3-540-73190-0

Spears, W. M. (2000). *Evolutionary Algorithms: The Role of Mutation and Recombination*. Springer Science & Business Media. https://link.springer.com/book/10.1007/978-3-662-04199-4

Bilaga A

Rådata för bästa lösningar av en-förälders genetisk algoritm

burma14	gr21	brazil58	ftv70	kroA100
En förälder	En förälder	En förälder	En förälder	En förälder
3323	2707	29717	2967	30027
3323	3197	28745	2720	34836
3323	2707	32364	2503	29407
3323	3180	27517	2592	32019
3323	2833	29518	2815	33455
3346	2878	28941	2246	36761
3323	2709	29584	2515	28508
3323	2801	28743	2718	28885
3527	3007	26725	2462	27903
3323	2897	33994	2952	34731
3346	2707	27849	2911	35294
3323	2707	31024	2921	32212
3323	2707	31769	2373	30695
3323	2707	29089	2511	28811
3323	3100	28410	2527	27707
3323	2801	31763	2486	30371
3371	2998	27649	3200	28625
3323	3242	28465	2850	32565
3323	2931	28900	3054	31527
3323	2709	32068	2680	31545

Tabell 5: *20 körningar av en-förälders genetisk algoritm på fem handelsresandeproblem*

bier127	gr202	lin318	fl417	pa561.tsp
En förälder	En förälder	En förälder	En förälder	En förälder
210201	68599	89874	26977	5794
169521	64623	85217	42130	6263
179493	57036	102313	41333	6664
161036	61305	103239	25127	6570
161672	68093	86160	30237	7444
150225	61243	109240	37166	6824
147487	53444	88635	33769	6294
165869	60070	85830	35749	6335
167522	60768	84629	33900	6041
171070	68297	96351	33341	6630
177502	58181	93706	25033	5500
150164	65327	81926	33249	6170
163420	59090	82717	27114	6226
221633	63988	98422	35987	6036
174440	60600	116169	29137	5239
179453	61269	116253	37099	5811
164489	61776	103655	27409	5930
152110	57145	94056	30518	5558
184261	64687	90469	35938	5850
172420	60552	99085	26007	6894

Tabell 6: *20 körningar av en-förälders genetisk algoritm på fem handelsresandeproblem*

Bilaga B

Rådata för bästa lösningar av standard genetisk algoritm

burma14	gr21	brazil58	ftv70	kroA100
Standard	Standard	Standard	Standard	Standard
3371	2930	44418	3418	43153
3371	3158	36267	3300	38018
3553	2968	35998	3586	47579
3371	3262	39552	3612	37573
3448	3374	45548	3321	39374
3530	3185	41346	3407	36032
3323	3328	32243	3435	46964
3455	3196	32797	3551	40073
3778	3630	42451	3797	40072
3517	3105	40176	3381	35947
3496	3462	37696	3715	41541
3336	3290	44289	3519	37683
3461	3303	30660	4027	43456
3323	2707	35796	2996	38184
3778	3587	38674	3597	45282
3448	3009	35358	3503	38303
3553	3043	33444	3246	36694
3336	3249	34034	3560	42510
3461	3027	32711	3566	38215
3461	2967	45931	3436	37990

Tabell 7: *20 körningar av en standard genetisk algoritm på fem handelsresandeproblem*

bier127	gr202	lin318	fl417	pa561
Standard	Standard	Standard	Standard	Standard
183570	64608	114905	80382	7107
202969	63244	113918	72978	8096
172295	73682	115972	73341	7902
176301	70304	108308	83788	8122
188275	67922	112307	78974	7963
183415	68301	107646	57511	8187
206961	65248	117586	81567	7759
174048	65933	111508	88093	7420
192817	71832	106639	81384	7595
177846	62336	118913	78978	7880
202858	62009	108394	95936	7481
165350	68332	104894	88711	7633
179053	65879	125738	77624	8184
196978	66297	125932	82989	7997
189152	69109	115601	79042	7993
185668	67205	103069	103644	7786
185113	66454	115698	89036	7460
203508	68371	120204	64470	7873
188006	71645	117668	85186	7854
186068	69937	120598	76387	8038

Tabell 8: *20 körningar av en standard genetisk algoritm på fem handelsresandeproblem*

Bilaga C

Rådata för antal generationer för att nå en lösning, en-förälders genetisk algoritm

burma14	gr21	brazil58	ftv70	kroA100
En förälder	En förälder	En förälder	En förälder	En förälder
5011	5624	11984	6765	7159
5197	5486	6927	9427	18486
6867	5382	20376	9048	11601
5025	5072	14219	6699	16123
5036	5456	10298	5509	10912
5032	5115	11452	11066	8056
5016	6243	7495	9716	27225
5226	5248	8437	14350	13822
5007	5156	13521	10650	12343
5700	5246	9129	6174	11605
5011	6616	12501	5361	12350
5099	5243	22060	5412	13917
5183	5181	10307	12451	20972
5050	5155	9197	6350	9071
5036	5019	22047	10222	8646
9771	5155	8848	9106	16241
5067	6237	18013	5272	14431
5025	5031	19666	5298	13846
5175	5153	18572	6676	15598
6150	5097	12375	6370	16540

Tabell 9: *20 körningar av en en-förälders genetisk algoritm på fem handelsresandeproblem*

bier127	gr202	lin318	fl417	pa561
En förälder	En förälder	En förälder	En förälder	En förälder
5425	6377	11987	21425	25810
8037	7659	21522	13159	29350
9634	10450	20195	10346	20556
10460	12678	14128	16586	24574
6841	7198	16210	21730	11174
7137	9550	8565	22144	19955
17739	12637	11483	23036	24134
15235	9606	14331	14029	28535
18400	7267	19928	23648	23079
6962	13400	13434	12735	16188
7032	9162	15812	14921	30765
19079	7076	12533	18131	22269
17481	13307	13947	25864	22927
9939	6577	17957	9765	23604
5926	9364	6761	11460	40774
8051	13650	6595	23702	31206
10159	14182	7802	13261	24010
16175	10952	10659	29558	36348
7580	9382	11811	15696	27656
6947	9604	13776	13746	13683

Tabell 10: *20 körningar av en en-förälders genetisk algoritm på fem handelsresandeproblem*

Bilaga D

Rådata för antal generationer för att nå en lösning, standard genetisk algoritm

burma14	gr21	brazil58	ftv70	kroA100
Standard	Standard	Standard	Standard	Standard
5276	5343	7765	11394	15442
5045	5259	8574	9534	16214
5057	5150	8489	7569	22326
5073	8064	7389	9693	17906
5030	5506	6856	7825	13370
5143	5176	9778	10630	20336
5104	5443	9349	9733	14789
5129	5140	12151	10138	19862
5094	5186	8532	7486	16317
5046	5163	9837	10416	30888
5098	8843	7452	7110	18061
5077	5156	8706	7848	20398
5086	5260	11015	7289	12529
5084	5206	7944	9293	18882
5063	5138	10683	7434	20697
5099	5215	8781	7113	21185
5032	5144	8358	10584	19514
5127	5204	7321	10550	16064
5081	5144	7910	7238	18571
5015	5298	7051	9440	19683

Tabell 11: *20 körningar av en standard genetisk algoritm på fem handelsresandeproblem*

bier127	gr202	lin318	fl417	pa561
Standard	Standard	Standard	Standard	Standard
32152	62260	139489	192861	304775
23160	133679	102068	202649	210348
17996	72460	139032	261545	220843
32543	74834	144549	218245	165388
28414	41079	125519	148384	181567
18021	68017	148428	371203	283762
18265	89274	131314	325937	213729
18802	56758	106455	223800	254025
26703	96775	115108	210518	185119
27339	65213	102594	348994	189466
17790	62306	152296	172170	206949
21891	85673	184221	239718	232774
22092	66171	97573	360255	235072
24982	80773	99866	290086	209394
23926	49434	161140	255397	247774
46909	67635	114100	237252	302518
41173	60753	173547	213794	259192
23797	51382	126172	249791	237423
27135	72022	162307	237843	238959
28622	129478	102881	216117	200587

Tabell 12: *20 körningar av en standard genetisk algoritm på fem handelsresandeproblem*

Bilaga E

Kod från genetiska algoritmer

```
def two_point_crossover(winners):
  new_individuals = []
  for individual in winners:

    first_cut, second_cut = sorted(random.sample(range(1, len(individual.path)), 2))
    genes = []

    first_part = individual.path[:first_cut]
    second_part = individual.path[first_cut:second_cut]
    third_part = individual.path[second_cut:]
    genes = [first_part, second_part, third_part]

    result1 = third_part + second_part + first_part
    result2 = second_part + first_part + third_part

    new_indi1 = Individuals(result1)
    new_indi2 = Individuals(result2)
    new_individuals.append(new_indi2)
    new_individuals.append(new_indi1)

  return new_individuals
```

Listing 1: En-förälders genetisk algoritm two point crossover

```
def swap_mutation(new_generation, mutationFactor):

  for individual in new_generation:
    if random.randint(0, 100) <= mutationFactor:
      firstNumber = random.randint(0, len(individual.path) - 1)
      secondNumber = random.randint(0, len(individual.path) - 1)

      firstIndividual = individual.path[firstNumber]
      secondIndividual = individual.path[secondNumber]

      individual.path[firstNumber] = secondIndividual
      individual.path[secondNumber] = firstIndividual

      individual.fitness_calculator()

  return new_generation
```

Listing 2: En-förälders genetisk algoritm swap mutation

```
def two_point_crossover(winners):
  new_individuals = []
  middle = len(winners)/2
  winners1 = winners[:int(middle)]
  winners2 = winners[int(middle):]

  for first_parent, second_parent in zip(winners1, winners2):
      first_cut, second_cut = sorted(random.sample(range(1, len(first_parent.path)), 2))
```

```
9
10      first_parent_first_part = first_parent.path[:first_cut]
11      first_parent_second_part = first_parent.path[first_cut:second_cut]
12      first_parent_third_part = first_parent.path[second_cut:]
13      second_parent_first_part = second_parent.path[:first_cut]
14      second_parent_second_part = second_parent.path[first_cut:second_cut]
15      second_parent_third_part = second_parent.path[second_cut:]
16
17      result1 = first_parent_first_part + second_parent_second_part +
    first_parent_third_part
18      result2 = second_parent_first_part + first_parent_second_part +
    second_parent_third_part
19
20      result1 = remove_duplicates(result1, first_parent.path)
21      result2 = remove_duplicates(result2, first_parent.path)
22
23      new_indi1 = Individuals(result1)
24      new_indi2 = Individuals(result2)
25      new_individuals.append(new_indi2)
26      new_individuals.append(new_indi1)
27
28  return new_individuals
```

Listing 3: Standard genetisk algoritm two point crossover

```
1  def swap_mutation(new_generation, mutationFactor):
2
3    for individual in new_generation:
4      if random.randint(0, 100) <= mutationFactor:
5        firstNumber = random.randint(0, len(individual.path) - 1)
6        secondNumber = random.randint(0, len(individual.path) - 1)
7
8        firstIndividual = individual.path[firstNumber]
9        secondIndividual = individual.path[secondNumber]
10
11       individual.path[firstNumber] = secondIndividual
12       individual.path[secondNumber] = firstIndividual
13       individual.fitness_calculator()
14
15   return new_generation
```

Listing 4: Standard genetisk algoritm swap mutation

```
1  def tournament_selectionx(problem, newProblem):
2    winners = []
3    population = problem + newProblem
4
5    for i in range(n_population):
6      contenders = random.sample(population, n_contenders)
7      winner = contenders[0]
8      for contender in contenders:
9        if contender.fitness <= winner.fitness:
10         winner = contender
11     winners.append(winner)
```

```
  return winners
```

Listing 5: Tournament selection

```
def init_tsp(problem, i):
  start_time = time.time()
  counter = 0
  minValueBefore = 10000000000000000000000000000000000000000000000000
  minValueAfter = 100000000000000000000000000000000000000000000000000
  generation = 0
  mutationFactor = 50
  global generations
  global best_solutions
  while(counter < 5000):
    winners = tournament_selection(problem.populations)
    for indi in winners:
      if indi.fitness < minValueBefore:
        minValueBefore = indi.fitness

    crossover_generation = two_point_crossover(winners)
    crossover_generation = swap_mutation(crossover_generation, mutationFactor)
    crossover_generation = tournament_selectionx(crossover_generation, winners)
    problem.populations = crossover_generation

    for indi in crossover_generation:
      if indi.fitness < minValueAfter:
        minValueAfter = indi.fitness

    if minValueAfter >= minValueBefore:
      counter += 1
    else:
      counter = 0

    generations.append(generation)
    best_solutions.append(minValueAfter)

    generation += 1
    mutationFactor = mutationFactor - 0.002
    data.append([i, "en-f r lders genetisk algoritm", problem.name, best_solutions[len(
    best_solutions) - 1], time.time() - start_time, generation, mutationFactor])

  minValueBefore = 10000000000000000000000000000000000000000000000000
  minValueAfter = 100000000000000000000000000000000000000000000000000
  counter = 0
```

Listing 6: En-förälders genetisk algoritm Main loop

```
def init_tsp(problem, i):
  start_time = time.time()
  counter = 0
  best_solutions = []
  generations = []
  minValueBefore = 10000000000000000000000000000000000000000000000000
  minValueAfter = 100000000000000000000000000000000000000000000000000
  generation = 0
  mutationFactor = 3
```

```
while(counter < 5000):
  winners = tournament_selection(problem.populations)
  for indi in winners:
    if indi.fitness < minValueBefore:
      minValueBefore = indi.fitness

  crossover_generation = two_point_crossover(winners)
  crossover_generation = swap_mutation(crossover_generation, mutationFactor)
  problem.populations = tournament_selectionx(crossover_generation, winners)

  for indi in crossover_generation:
    if indi.fitness < minValueAfter:
      minValueAfter = indi.fitness

  if minValueAfter >= minValueBefore:
    counter += 1
  else:
    counter = 0

  generations.append(generation)
  best_solutions.append(minValueAfter)

  generation += 1
data.append([ i, "standard genetisk algoritm", problem.name, best_solutions[len(
  best_solutions) - 1], time.time() - start_time, generation, mutationFactor])

minValueBefore = 1000000000000000000000000000000000000000000000000000
minValueAfter = 1000000000000000000000000000000000000000000000000000
generations = 0
counter = 0
```

Listing 7: Standard genetisk algoritm Main loop

Bilaga F

Reflektion Jakob Vikström

Examensarbetet anser jag fyller de krav som krävs för att examinera detta examensarbeteskurs. Jag och Oscar Hansson har tydligt och uppmärksamt evaluerat de betygskriterium som beskrivits i exjobbsanvisningarna. Överlag har den information given av Pierre Arne Ingvar Wijkman och exjobsanvisningarna följts till punkt och prick. På så sätt har processen att skriva detta arbete förenklats. Studien är baserad på relevant tidigare forskning vilket har givit ett problem som bestått efter flera årtal av forskning. Utöver det har en bakgrund till denna domän givits. Detta har resulterat i en intressant frågeställning som kunde undersökas i mån om att förbättra en del av det större problemet för genetiska algoritmer. Frågeställningen skapades med hjälp av handledare Wijkman, intresse för domänen samt i mån om att skapa något som skulle kunna utveckla domänen ytterligare. Forskningsstratergin är beskriven väl i definition såväl som implementation med motivation utifrån experimentet. Det gäller även datainsamlingsmetoden samt analysmetoden. Alternativa datainsamlingsmetoder och analysmetoder har undersökts och beskrivits. Samt samhälleliga och forskningsetiska aspekter. Då denna studie använder forksningsstratergin experiment är transparens av utformningen av experimentet en fallgrop. Jag anser att alla inställningar algoritmerna implementerades med var väldefinierade och metoder för att mäta och analysera data förklarades på ett enkelt och lättförstått sätt. Resultatet, utveckling av algoritmer och mätning av dessa algoritmer, och analysen har givit information vilket kunde lyftas vidare till en intressant diskussion där tidigare forskning visat liknande resultat. En diskussion om studiens tillvägagångssätt har även givit intressant information om vad som kunde gjorts bättre samt vilka aspekter som kan vara intressant för framtida studier. Slutsatsen kunde där efter utformas på ett relevant och grundat sätt. Då studien använt oss av peer review för alla delar av arbetet har förbättringar genom insikter av andra studenter utformats i en välskriven, väl strukturerad och relevant studie. Det kan påpekas att vissa betygskriterier på högre nivå inte prioriterats. Det har resulterat i att vissa delar av studien inte är av, enligt exjobsanvisningarna, högsta klass. Till exempel kan det vara svårt att evaluera originalitet och signifikans på den nivån att denna studie skulle kunna presenteras på en workshop eller vetenskaplig konferens. Detta visas även i vetenskaplig förankring där en omfattande litteraturstudie saknas. Men en någorlunda djupgående och kritisk diskussion om tidigare forskning existerar men som kanske inte når kravet för 2 poäng enligt betygsunderlagen. Denna studie har inte försökt att nå 3 poäng i någon av betygskriterierna. Detta hade så-klart givit en mer kunskapsrik studie. Då denna studie fick förlängd deadline kunde planeringen för detta arbete förbättrats. Det tog lång tid innan första milstenen nåddes för detta projekt då vi hamnade i en intressant domän där vi inte hade någon förkunskap specifikt om genetiska algoritmer. Kunskapsluckans konsekvens var att inget i första delen av arbetet skrevs på egenhand, då det krävdes långa diskussioner av vad begrepp innebar. När denna kunskapslucka fylldes gick arbetet i bra takt, dock hjälpte inte det med den tid vi missat. Det misstag som påverkats mest var att studien skrevs på heltid. Om denna studie skrivits på deltid hade mer tid lagt på att så tidigt som möjligt förstå domänen innan arbetets början. Då Oscar Hansson går datavetenskapliga programmet och jag går data- och systemvetenskapliga programmet fanns en vilja att skapa ett arbete inom datavetenskap. Detta enligt mitt intresse för sökalgoritmer och programmering vilket skapades genom kurser så som Programmering 1 och 2, Algoritmer och Datastrukturer samt vissa inslag om NP problem från Data Mining. Intresset för algoritmer gav mig hösten 2023 ett jobb på ett stor finansteknologiskt företag där jag tidigt i anställningsprocessen fick visa mina färdigheter inom utveckling och metodologi inom datavetenskap. Även om genetiska algoritmer kanske inte är ett ämne jag kommer arbeta med i framtiden har jag bevisat att jag på kort tid kan bli expert inom ett komplext ämne

och i vissa fall kanske denna kunskap kan ge mig lite nytta i arbetslivet. Denna studie är jag nöjd med. Jag och Oscar Hansson har arbetat tight för att skapa ett så bra arbete som möjligt. Det fanns dock utrymme att skapa ett mycket mer omfattande arbete vilket vi beskrivit i framtida studier och brister. Att bli expert inom ett sådant ämne som genetiska algoritmer, som tidigt var väldigt förvirrande, har gjort mig otroligt stolt.

Bilaga G

Reflektion Oscar Hansson

Under arbetets gång så fokuserade vi mycket på att uppnå de mål som var angivna för varje kapitel, dessa har varit i vårt bakhuvud under hela arbetets gång och vi har varit väldigt noga med att nå alla mål som har angivits. Däremot så var det målen som motsvarade en poäng som prioriterades i första hand och de övriga har antingen uppfyllts eller inte blivit prioriterade. Till exempel så kanske vi inte har varit tillräckligt djupgående eller kritiska i vissa delar av uppsatsen för att uppnå en högre poäng. Planeringen för examensarbetet fungerade under omständigheterna bra, i början var det ganska kaotiskt då vi inte hade något förutbestämt att skriva om och det var svårt att läsa in oss på ett område som vi var intresserade av, vi gick igenom flera olika idéer som inte kom någon vart, i slutändan så fick vi hjälp av vår handledare att hitta ett ämne som han tyckte passade. Det som jag tycker var svårt i början och som hade gått att göras bättre hade varit om det fanns tydligare riktlinjer för hur man skulle komma igång och om det hade funnits fler idéer på examensarbeten i Sci-pro, när vi väl kom igång med vårt arbete så hade det redan gått en månad på kursen och det kändes omöjligt att hinna färdigt med arbetet i tid. För övrigt så fungerade vår interna planering och uppdelning av arbetet bra, vi hade regelbundna möten och delegerade uppgifterna mellan oss. Något som vi hade kunnat bättre var att vi båda var väldigt ivriga att börja skriva algoritmerna och få ut testresultat innan vi var helt säkra på vad vi ville ha för data från körningarna, detta ledde till att vi var tvungna att köra om testerna några gånger och eftersom det tog lång tid för att köra algoritmerna så försvann det ganska mycket tid på det, men samtidigt så kanske det är svårt att få allt rätt från första början. Vi hann inte med att lämna in vårt färdiga arbete innan terminsslut, detta kan ha berott på att vi kom igång sent och att vi var tvungna att läsa in oss på ett ämne som ingen av oss hade någon tidigare erfarenhet inom. Det hade varit bra med ytterligare handledning och tydligare riktlinjer i början. Detta examensarbete relaterar till utbildningen eftersom att det är inom datavetenskapen och har en koppling till flera kurser som till exempel ALDA eftersom arbetet kort går in på komplexiteten på algoritmerna, sen har jag haft nytta av alla programmeringskurser som var till hjälp när vi skulle implementera algoritmerna, från metod kapitlet och framåt var METOD kursen oerhört användbar, VESK var självkart också användbar genom hela arbetet. Jag valde ett ämne som jag tyckte var intressant att skriva om och även fast jag inte kommer studera vidare efter kandidatexamen så är jag väldigt nöjd med resultatet och det kommer att vara till stor hjälp i mitt framtida jobb. Jag har fått ett arbete inom industrin där jag kommer att jobba mycket med effektivisering och automatisering för produktion och jag känner att genom denna uppsats så har jag lärt mig mycket om just det och jag hoppas att jag kan implementera evolutionära algoritmer i mitt framtida arbete. Jag är nöjd med genomförandet, det som funkade mindre bra var inledningen, det var svårt att komma igång och vi hade nog behövt mer vägledning i början. Efter att vi kom igång så flöt allting på bra och vi hade ett bra samarbete. Jag är ganska nöjd med resultatet också men jag hade jag fått göra om arbetet igen så hade jag ändrat om en hel del, exempelvis så hade jag gärna skapat några ytterligare algoritmer att jämföra med för att kunna dra mer slutsatser av resultatet. I det stora hela är jag dock väldigt stolt över detta arbete och jag tycker att vi gjorde ett bra jobb.