Examensarbete

HÖGSKOLAN
I SKÖVDE

VÄGPLANERING MED SLUMPMÄSSIG GENUSFÖRDELNING I GENETISKA ALGORITMER

PATHFINDING WITH RANDOM GENDER DISTRIBUTION IN GENETIC ALGORITHMS

Examensarbete inom huvudområdet Datalogi
Grundnivå 30 högskolepoäng
Vårtermin 2016

Stefan Nilsson

Handledare: Sanny Syberfeldt
Examinator: Mikael Johannesson

Sammanfattning

Den här studien beskriver en undersökning som tittar på prestationsskillnaden på en genetisk algoritm (GA) med kön, mot en GA utan kön. Den tittar även på prestationsskillnaden mellan en GA utan kön och en GA med slumpmässig könsfördelning. Problemet som algoritmerna är applicerade på innefattar vägplanering över en kontinuerlig, procedurell 2d-miljö som är uppbyggd av polygoner med olika färdkostnad. Den lämpligaste individen blir den med lägst färdkostnad, som alltså hittar den billigaste vägen mellan två fasta punkter. Observera att problemet är fullständigt statiskt genom alla körningar.

Utfallet tyder på att GA'n med kön producerar märkbart bättre resultat, oavsett om könsdistribueringen var konstant eller slumpmässig. De genetiska algoritmerna med kön uppvisade också en mindre tendens att konvergera, vilket tyder på att införandet av kön i algoritmen hade den hypotiserade effekten. Införandet av slumpmässig könsfördelning tycks ha resulterat i en mer konstant diversitet.

Vidare arbete skulle kunna innefatta en jämförelse mellan genetiska algoritmer med och utan kön för fler typer av problem. Detta skulle kunna leda till en allmän teori som sammanfattar könets påverkan på de genetiska algoritmerna allmänt. Vidare studier skulle även kunna titta på hur kön påverkar diversiteten i algoritmen, genom att göra samma jämförelser, men istället titta på hur den genetiska diversiteten utvecklas över tid.

Nyckelord: Genetiska algoritmer, kön, kontinuerlig vägplanering

Innehållsförteckning

1 Introduktion

Genetiska algoritmer används ofta i många olika situationer där antalet möjliga lösningar vida överstiger vad som rimligtvis kan jämföras med sedvanliga metoder (djupet-först, bredden-först sökningar). Då kan det istället vara lämpligt att använda en sökmetod som inte söker igenom hela lösningsrymden utan snarare tar stickprov på hela lösningsrymden för att hitta de bästa. Genetiska algoritmer gör just detta. Genom att kombinera existerande lösningar hoppas man att med hjälp av naturlig selektion få bättre lösningar, och på så vis ta fram en godtagbar lösning. Observera att genetiska algoritmer inte letar efter den optimala lösningen, utan snarare letar efter en lösning som kan godtas av användaren.

Ett problem som uppstår med genetiska algoritmer kallas för ”prematur konvergens”, när den genetiska diversiteten försvinner. Detta innebär att algoritmen fastnar i ett lokalt optima, en punkt i lösningsrymden där samtliga närliggande lösningar är sämre. Velazco & Bullinaria (2003) gjorde en studie på genetiska algoritmer där de bland annat införde genus i algoritmen. Deras resultat visade på att den genusbaserade algoritmen betedde sig bättre, i och med att den fick bättre resultat på kortare tid. Från detta kan man ställa en hypotes att införandet av genus generellt är bra för genetiska algoritmer. Detta arbete undersöker den hypotesen, men applicerar algoritmerna på ett annat problem: vägplanering i kontinuerlig 2d-miljö.

Miljön består av ett bestämt antal polygoner som var och en har en färdkostnad, exempelvis representativ för en maximal möjlig hastighet inom det området (i skog kan man exempelvis inte röra sig lika fort som på öppen mark). Varje lösning eller genom i den genetiska algoritmen symboliserar en rutt som navigerar från en startpunkt till en slutpunkt. Varje genpar i varje genom symboliserar en punkt i miljön. Varje genom symboliserar en serie punkter som då blir rutten för det specifika genomet. Den totala kostnaden för rutten blir då ett mått på hur bra just den lösningen är.

Miljön genereras procedurellt med hjälp av en algoritm vagt inspirerad av kortaste klyvningslinjemetoden som bland annat används i USA för att dela upp stater i röstningsdistrikt. Observera att algoritmen endast är vagt inspirerad av denna. Algoritmen utgår från en rektangel och klyver varje polygon i miljön i två nya polygoner. Detta upprepas tills dess att önskad mängd polygoner uppnåtts. Allteftersom polygoner skapas ges de även en färdkostnad inom ett förutbestämt intervall.

Utfallet i kapitel 5 tyder på att de genetiska algoritmerna med kön producerade märkbart bättre resultat, men även att de uppvisade en lägre tendens för konvergens. Detta påvisar att införandet av kön medgav en positiv effekt för den här typen av problem (vägplanering genom kontinuerlig miljö).

2 Bakgrund

I det här kapitlet diskuteras bakgrunden till den frågeställning som formuleras i kapitel 3. Frågeställningen utgår från innehållet i det här kapitlet. Här förklaras också de centrala koncept som problemformuleringen bygger på.

2.1 Genetik och evolutionsteknik

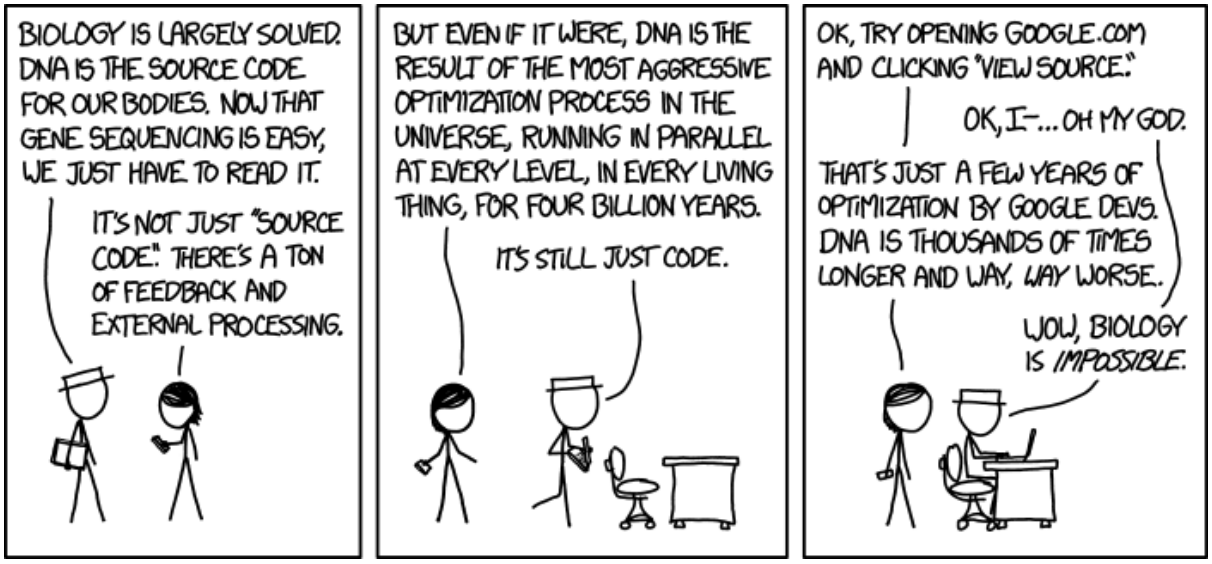

Illustration 1: xkcd 1605: DNA

I naturen råder naturens lagar, att endast de som är tillräckligt anpassade överlever miljön. Ett resultat av detta är att samtliga individer i en population ”tävlar” om att vara den som är bäst på att överleva. På så vis får vi en aggressiv optimeringsprocess som hela tiden strävar efter att förbättra individer. Då individer är kodade i DNA innebär detta att DNA't förändras över tid. Som vi kan se i xkcd 1605 (2016) ovan resulterar detta i att DNA är helt oläsbart, eftersom det har optimerats väldigt aggressivt, väldigt länge. Detta låter oss konstatera att komplicerade lösningar på abstrakta problem kan bestå av simpla byggstenar arrangerade i ett kanske oväntat mönster.

2.2 Genetiska algoritmer

En Genetisk Algoritm (GA) är en algoritm vars uppgift är att hitta lösningar till problem vars lösningsrymd är väldigt stor. Vi kan exempelvis prata om vanliga vägplaneringsalgoritmer, som Dijkstra's algoritm och A-stjärna. Dessa algoritmer söker igenom stora delar av problemrymden för att hitta den bästa lösningen till problemet. Nackdelen med dessa är just att de söker igenom stora delar av problemrymden för att hitta den absolut bästa lösningen: för vissa problem är problemrymden såpass stor att en genomsökning av endast en bråkdel av den skulle ta alldeles för lång tid för att vara praktiskt.

2.2.1 Handelsresandeproblemet

Ett problem vars sökrymd är för stor för vanliga vägplaneringsalgoritmer är handelsresandeproblemet. Formellt kan handelsresandeproblemet beskrivas som ”Finn den

billigaste hamiltoncykeln i en graf med bågkostnader", där noderna är städer, bågarna är vägar, och kostnaderna är väglängderna. Detta innebär att varje nod (stad) ska besökas exakt en gång, och den väg som bildas ska ha lägsta möjliga kostnad. Ett sådant problem får en lösningsrymd med storleken $\frac{m!}{n!(m-n)!}$ där m är antalet bågar och n är antalet noder. Under dessa förutsättningar får en fullt kopplad graf med 9 noder ca 94 miljoner olika lösningar. En fullt kopplad graf med 10 noder får över 3 miljarder lösningar. Detta visar tydligt att antalet lösningar ökar explosionsartat, vilket medför att lösningsrymden blir enorm. Därmed kan en bra metod vara att inte titta på samtliga lösningar, utan endast ta stickprov ur lösningsrymden, och försöka hitta en bra lösning utifrån dessa.

2.2.2 Populationen

Den centrala komponenten i en GA är populationen, som består av en samling element (också kallade individer eller genom). Varje genom representerar en unik lösning till problemet, och består av ett antal gener som kan representeras på olika sätt. Några vanliga sätt att representera generna i ett genom är med booleska värden (bitar), flyttal eller tecken. En genetisk algoritm itererar upprepade gånger över populationen. För varje iteration rensas oftast hela populationen från individer, och får en helt ny uppsättning individer. Varje sådan mängd med individer kallas för en generation. För varje iteration på populationen får varje enskilt genom ett fitness-värde, som är ett mått på hur bra motsvarande lösning löser problemet. Den här funktionen är alltid unik för varje problem.

2.2.3 Operatorer

Genetiska algoritmer använder en serie med operatorer, ofta en av varje typ av de tre typerna mutationsoperator, reproduktionsoperator och selektionsoperator. Dessa operatorer har i uppdrag att från en iteration, skapa en ny, bättre iteration. Operatorerna körs oftast i ordningen selektionsoperator, reproduktionsoperator, mutationsoperator.

Selektionsoperatorn väljer ut vilka genom som ska få gå vidare till reproduktionsoperatorn. Ett vanligt sätt som används för selektionsoperatorn är roulettehjulsprincipen, där chansen för ett genom att bli utvalt är dess fitness-värde dividerat med generationens individers summerade fitness-värden. På så vis får samtliga genom en bra chans att bli valda, samtidigt som bättre genom har en större chans.

Reproduktionsoperatorn kombinerar två genom till ett eller två nya genom. En vanlig metod som används är att man genererar ett slumpmässigt index som pekar på ett visst index i gensekvensen. Alla gener före det indexet i det första genomet tillsammans med alla gener efter det indexet i det andra genomet blir ett nytt genom. Det är även vanligt att de övriga generna bildar ett till genom. Denna metod kallas för enpunktsöverkorsning.

Mutationsoperatorns uppgift är att införa nya gener i populationen. Genom att slumpmässigt välja ut genom, och slumpmässigt modifiera dessa, exempelvis genom att slumpmässigt välja ut ett bestämt antal gener i genomet och för varje gen (om de är flyttal) addera ett slumpmässigt tal, och sedan anpassa det nya värdet till de begränsningar som markerats för gener.

2.3 GA för vägplanering

Velazco & Bullinaria (2003) införde genus och genusspecifika selektionsoperatorer i genetiska algoritmer, och jämförde med en genusfri genetisk algoritm på det klassiska handelsresandeproblemet. Deras resultat pekar på att införandet av genus i algoritmen gav bättre resultat, snabbare. Den genusbaserade genetiska algoritmen fick efter färre generationer bättre lösningar till problemet. Dock måste tilläggas att deras undersökning skiljer sig från denna på flera punkter. Bland annat tittade deras studie på en GA med kompetitiva och kooperativa operatorer på en jämt könsfördelad population, som dessutom applicerades på ett annat problem.

2.3.1 Kontinuerlig vägplanering med GA

Ett vanligt sätt att symbolisera gener i genetiska algoritmer är med flyttal. Detta passar utmärkt när man vill kartlägga genom till rutter i en kontinuerlig miljö. Man konverterar helt enkelt genernas värden som är i ett förutbestämt intervall, till det intervallet som skapas av miljöns kanter. På så vis kan man utläsa koordinater i miljön från genomen, och varje genpar blir då en koordinat, och genomet blir en serie med koordinater, en rutt genom miljön. Man kan här göra en jämförelse mellan att använda binära gener, och att använda flyttalsgener, precis som Cezary & Zbigniew (1991) gjorde. De konstaterade att flyttalsrepresentation är snabbare, och mer lämpligt i fall där flyttalsprecision är önskad, till exempel vid behov av kontinuerliga parametrar. Detta eftersom den booleska kodningen skulle bli extremt mycket mer komplicerad, och genomen skulle bli väldigt långa och svårbehandlade.

2.3.2 Populationsdiversitet

Om vi tittar på kontinuerlig vägplanering i en miljö uppbyggd av en serie polygoner, där vägen endast består av en bestämd startpunkt, en bestämd slutpunkt samt koordinaterna för en vägpunkt. Den vägpunkten består av två flyttalsvärden. Dessa två värden kan man se på som generna i ett väldigt kort genom. Lösningsrymden för ett sådant problem skulle då kunna kartläggas till en 3d-graf, med fitness-värdet för lösningarna på höjden. I Illustration 2 kan man se hur lösningsrymden skulle kunna se ut för ett sådant problem.

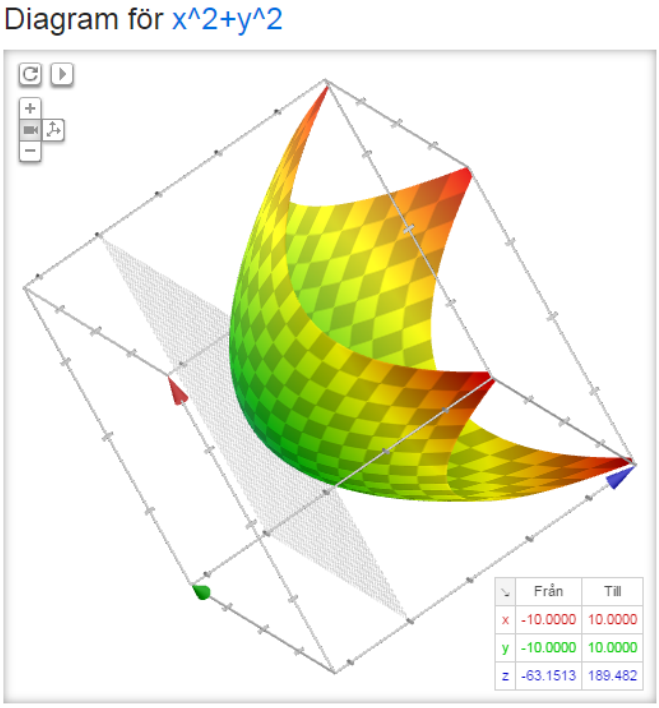

Illustration 2: Googles verktyg för grafer: ett lösningsrymdsexempel

I mer komplexa fall kan lokalt optimala punkter lätt uppstå, punkter i lösningsrymden där samtliga närliggande punkter allihop är sämre, men som fortfarande inte är den globalt optimala punkten. Dessa punkter kallas lokala optima, och är punkter där algoritmen kan fastna, om inte den genetiska diversiteten bevaras. Den genetiska diversiteten är ett mått på hur många unika lösningar det finns i populationen, eller, mer korrekt, ett mått på hur många olika unika gensekvenser det finns i populationen. När diversiteten sjunker kraftigt eller når en nollnivå säger man att algoritmen konvergerar. Detta är ofta dåligt, då algoritmen sällan kan ta sig ur en sådan punkt.

Nguyen & Nguyen (2008) pratar om hur man mäter diversiteten. Detta skulle kunna vara direkt användbart om man kopplar till exempel mutationsfaktorer till den genetiska diversiteten. På så vis skulle man kunna öka antalet mutationer och mutationsstorleken om algoritmen ser ut att konvergera.

2.3.3 Subpopulationer: en lösning till prematur konvergens

Herrera & Lozano (2000) skriver om ett sätt att lösa prematur konvergens, eller brist på genetisk diversitet. De föreslår flera subpopulationer, helt oberoende av varandra, som parallellt bearbetas av genetiska algoritmer, tillsammans med en migrationsmekanism som skapar ett genomutbyte mellan subpopulationerna. Genom att låta varje subpopulation itereras över av GA's med olika inställningar ges väldigt positiva resultat, som erbjuder bättre pålitlighet och träffsäkerhet. En annan, mindre komplicerad metod som ofta krediteras för att försena konvergens är att införa genus i algoritmen.

2.3.4 Genus i naturen

Genus i naturen medför att varje individ endast kan para sig med en del av populationen. Detta medför att de två bästa genomen inte nödvändigtvis kan para sig med varandra, då chansen finns att de är i samma genusgrupp. Därmed minskar tendensen för konvergens i algoritmen.

2.3.5 Genus i genetiska algoritmer för vägplanering

Genom att införa genus i genetiska algoritmer hoppas man bevara den genetiska diversiteten och samtidigt vidare efterlikna naturen.

3 Problemformulering

Som beskrivs av Bullinaria & Velazco, (2003) så är genusfria genetiska algoritmer mycket vanligare än genus-baserade genetiska algoritmer. Reproduktion sker vanligtvis genom att man korsar två individer, utvalda ur hela populationen, baserat på deras fitness-värde. Bullinaria & Velazco's slutsats fastställer att en algoritm som använder genus kan överträffa en genus-fri genetisk algoritm, iallafall för det klassiska problemet traveling salesman (vägplanering genom ett nätverk av städer. Varje stad ska besökas exakt en gång, ju kortare resa, desto bättre lösning).

Då undersökningen kommer göras på ett annat problem än Traveling Salesman, kan Bullinaria & Velazco's studie förefalla irrelevant, dock måste noteras att de konstaterar att genus-baserade GA's presterar bättre än genusfria GA's, vilket är högst relevant för den här studien. Baserat på deras slutsats kan man också formulera en hypotes att anledningen till att genusbaserade GA's presterar bättre är på grund av att införandet av genus i algoritmen medför en generellt högre genetisk mångfald, och därmed en lägre tendens för konvergens, alltså då algoritmens genom sammanfaller och blir kloner av varandra. Detta skulle innebära att den genetiska mångfalden bevaras över en längre tid, vilket leder till att algoritmen har en lägre tendens att fastna i lokalt optimala punkter i sökrymden. Då problem av den här typen ofta har lösningar som är bättre än alla liknande lösningar, men som fortfarande inte är bäst, då den bästa lösningen är väldigt olik, får vi punkter i sökrymden som är bättre än alla närliggande punkter. Dessa kallas för lokala optima. Genetisk mångfald förhindrar att algoritmen fastnar i sådana punkter.

Märk även att Bullinaria & Velazco använder en konstant balans mellan de två könen (50-50), vilket inte stämmer överens med naturen. När de parar två lösningar, individer i en population, tillverkar de alltid två nya lösningar, en av varje kön. Denna uppsats studerar en genetisk algoritm som, likt deras, är genusbaserad, men som, olikt deras, inte använder en konstant balans mellan de två könen. Algoritmen appliceras inte heller på det klassiska handelsresandeproblemet, utan används till vägplanering i en kontinuerlig miljö uppdelad i polygoner (fält) som har en symbolisk färdkostnad (representativ för exempelvis en hastighetsbegränsning/effektiv maximal hastighet, eller byggkostnad. De olika fälten kan ses som varande olika biomer, där till exempel bergstrakter har en låg maxhastighet/hög byggkostnad och slättland har hög maxhastighet/låg byggkostnad).

Roupec (2011) har gjort en studie där bland annat genus-baserade genetiska algoritmer studeras. Dock är slutsatsen väldigt otydlig, och nämner inte någon av de olika implementationsvariationerna som används. Roupec använder sig däremot av ungefär samma genus-arrangemang som är planerat att användas: när nya individer genereras tilldelas de ett av de två könen manligt och kvinnligt. Därmed kommer populationen alltid vara uppdelad på två, och överkorsning (parning) blir väldigt simpelt: välj ut en individ från varje del, och överkorsa dessa. Fortsätt tills den nya generationen är av lämplig storlek (lika stor som den föregående).

Två frågeställningar föreslås:

1. Kommer, i de flesta fall, en genus-baserad genetisk algoritm prestera bättre jämfört med en genusfri genetisk algoritm vid vägplanering i en kontinuerlig miljö?
2. Kommer en genus-baserad genetisk algoritm där könsfördelning sker slumpmässigt, prestera bättre än en jämnt balanserad genetisk algoritm i samma miljö?

För båda frågorna appliceras samtliga algoritmer på ett exempelproblem, där algoritmernas uppgift är att planera en väg genom en kontinuerlig 2d-miljö bestående av polygoner med färdkostnad. Algoritmerna utgår från samma punkt, och har samma målpunkt. Varje genpar i genomen kartläggs till koordinater i 2d-miljön. Vägen blir då den rutt som bildas om man drar raka streck mellan varje koordinat, i följd, om man utgår från startpunkten, och slutar på slutpunkten. Prestationen hos algoritmerna mäts genom att titta på hur samtliga fitness-värden för vardera population förändras över tid. För en bättre algoritm skulle fitness-värdena förändras med en högre hastighet i positivt led. Detta skulle innebära att värdena i ett tidigare stadium når höga värden. En algoritm som uppfyller detta beskrivs som att den uppnår ett acceptabelt resultat på kortare tid.

3.1 Metodbeskrivning

För att kunna studera de nödvändiga aspekterna skall ett experiment genomföras, där ett program implementeras som kör två olika genetiska algoritmer vars fitness-funktion är hur bra det löser problemet att vägplanera över en procedurellt genererad, kontinuerlig miljö bestående av polygoner, där varje polygon har en resekostnad. Ju billigare lösning, desto bättre fitness-värde. Kartläggningen från genom till väg sker genom att ett genom består av en serie med flyttal. Varje par av flyttal symboliserar en koordinat i en 2d-rymd. Dessa serier av koordinater som bildas för varje genom kartlägger då en väg, med de olika koordinaterna som vägpunkter. Under varje körning samlar programmet in fitness-värden för samtliga individer över samtliga generationer för samtliga algoritmer. För att kunna titta på den genetiska diversiteten sparas även varje enskilt genom undan. Genom och fitness-värde sparas ihop så att eventuell nödvändig koppling kan göras utan besvär.

3.1.1 Analys

Då en intressant punkt att studera är hur tidigt vardera algoritm konvergerar, kan man för varje algoritm titta på hur den genetiska diversiteten utvecklas över tid. För att göra detta kan man se genomen som vektorer i en n-dimensionell rymd, där n är antalet gener i ett genom. Genomsnittet för alla genom blir då en medelpunkt i samma rymd. Den genetiska diversiteten kan då definieras som det genomsnittliga avståndet från varje genom, till medelpunkten.

Från en sådan jämförelse kan man sedan eventuellt konstatera vilken algoritm som har störst tendens för prematur konvergens.

En annan intressant analys är att titta på hur fitness-värden ändras över tid. Genom att plotta en graf över hur genomsnittsvärdet, minimivärdet och maxvärdet för fitness för varje generation över tid, kan man titta på hur fitness-värden utvecklas, och därifrån eventuellt dra en slutsats om vilken algoritm som presterar bäst, där bättre är definierat som att den positiva förändringen hos fitness-värdena är högre. Det är även intressant att titta på hur standardavvikelsen förändras över tid.

3.1.2 Metoddiskussion

Nackdelen med den här mätmetoden är att den inte tar hänsyn till hur rak vägen är, vilket skulle vara relevant för exempelvis järnvägssträckor, då tågen måste sakta ner i kurvor. För detta skulle man även behöva ta hänsyn till summan av absolutbeloppet på vinklarna som

uppstår vid varje sväng. Det är inte relevant att räkna med negativa vinklar, då tågen måste sakta ner lika mycket, oavsett åt vilket håll de svänger.

Alternativt skulle man kunna mäta både sträckans färdkostnad (som kan symbolisera byggkostnaden för järnvägen) och sträckans ”rakhet” det vill säga hur mycket den svänger, för att ta fram den ultimata järnvägsrutten. Dock skulle man i ett sådant fall förmodligen vilja vikta sträckans rakhet, då det är en kostnad över tid, medan byggkostnaden är en engångskostnad.

4 Genomförande

I det här kapitlet beskrivs implementationsprocessen för artefakten. Kapitlet börjar med en förstudie där utvecklingsmiljö och kollisionsdetektering tas upp. Då kollisionsdetekteringen är en komplicerad, och viktig del i artefakten, är det även en stor del av förstudien. Efter förstudien följer en redovisning av implementationsprocessen då produkten implementerades. Här tas även designbeslut som tagits under processen upp, och diskuteras. Kapitlet avslutas med en beskrivning av en pilotstudie, vars syfte är att visa att produkten är tillräcklig för att utföra studien som beskrivs i tidigare kapitel.

4.1 Förstudie

4.1.1 Val av utvecklingsmiljö

Slutprodukten har inga som helst grafiska krav eller behov, men för att förenkla implementationsprocessen var det nödvändigt att ha någon form av grafisk representation av hur de två algoritmerna betedde sig i miljön. Därför valdes en utvecklingsmiljö med högt stöd för grafik. Vidare ställde slutprodukten väldigt få krav på möjligheter i utvecklingsspråket, vilket innebar att det kunde väljas efter behag. Därför gjordes implementationen i c#, som är ett högnivåspråk men som fortfarande har många möjligheter för egen implementering, och spelmotorn Unity (Unity Technologies, 2005-2016), som tillhandahåller många avancerade grafiska aspekter. Alternativt hade till exempel Java varit ett alternativ, då de har väldigt grundläggande (men tillräckliga) grafiska implementationsmöjligheter i ett standardbibliotek. Vidare hade även c++ kunna varit ett alternativ, på grund av de möjligheter språket i sig ger. Dock finns det få grafiska möjligheter i det språket, utan ett tredjepartsbibliotek hade då med största sannolikhet varit nödvändigt.

4.1.2 Kollisionsdetektering

Kostnaden för en väg kan beräknas som: för varje polygon, summera sträckan vägen går igenom polygonen, multiplicera med områdets färdkostnad, och lägg till detta till den totala vägkostnaden. Detta kan lättare beskrivas om man tittar på en hypotetisk situation som kan

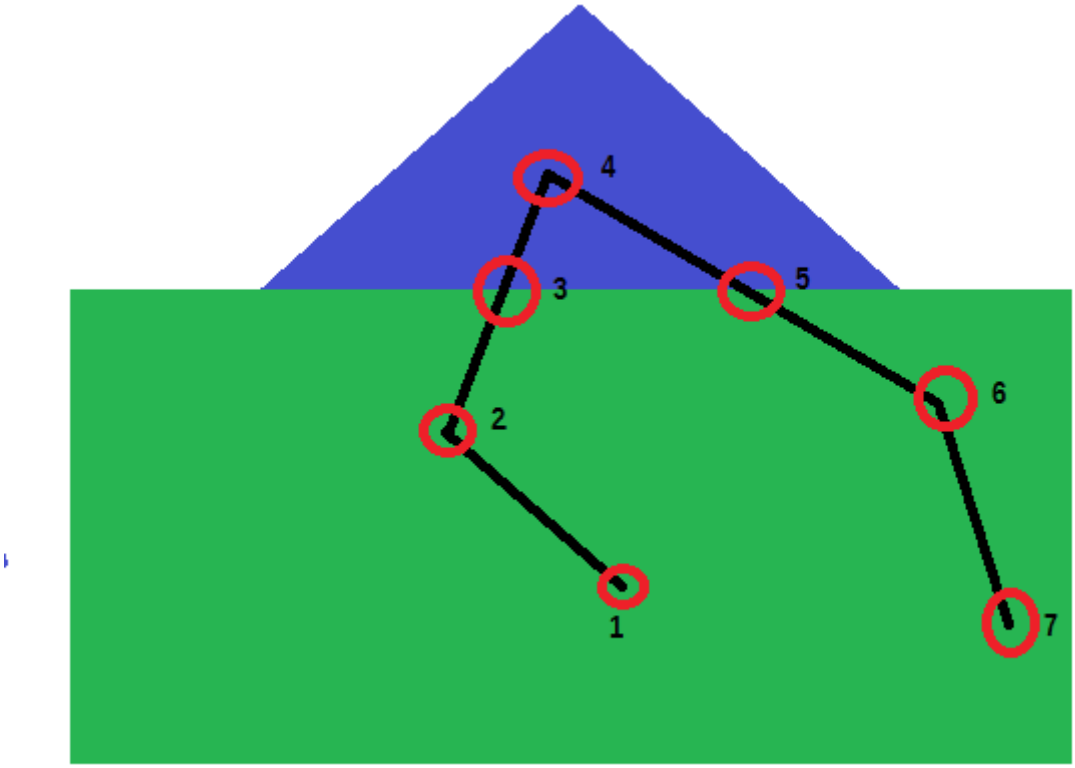

Illustration 3: väguppdelning i polygoner

uppstå (se Illustration 3). I figuren finns en svart väg, med ett antal punkter inringade i rött och numrerade. Kostnaden för vägen som syns kan beräknas som: beräkna avståndet mellan punkt 1 och 2. Multiplicera med färdkostnaden för den gröna polygonen, och lägg till vägens totala kostnad. Gör samma sak för sträckan mellan punkterna 2 och 3. Upprepa för 5 och 6 samt 6 och 7. Gör sedan samma sak för den andra polygonen, det vill säga, beräkna avståndet mellan punkt 3 och 4, multiplicera resultatet med färdkostnaden för den blåa polygonen, och lägg slutresultatet till vägens totala kostnad. Gör sedan samma sak för sträckan mellan punkt 4 och 5.

För att kunna göra detta behöver man kunna avgöra om en punkt är innanför en polygon. Mecki (2008) beskriver en enkel metod för att göra just detta. Den går ut på att man drar en linje från en punkt som absolut är utanför polygonen, till den punkt som skall undersökas. Sedan räknar man hur många gånger den linjen träffar kanter som tillhör polygonen. Är antalet träffar udda, är punkten innanför polygonen. Detta fungerar för alla typer av polygon. En demonstration kan ses i Illustration 4 nedan, där metoden har applicerats på två olika punkter (numrerade i bilden som punkt 2 och 3). Vi ser att en linje dragits mellan en punkt som vi vet är utanför polygonen (punkt 1) till de punkter vi vill undersöka. Till punkt två ser vi att en kollision inträffar. Punkten är alltså inuti polygonen, eftersom 1 är ett udda tal. Punkt 3 däremot, är utanför polygonen, då vi får ett jämnt antal kollisioner (punkt 5 och 6, två).

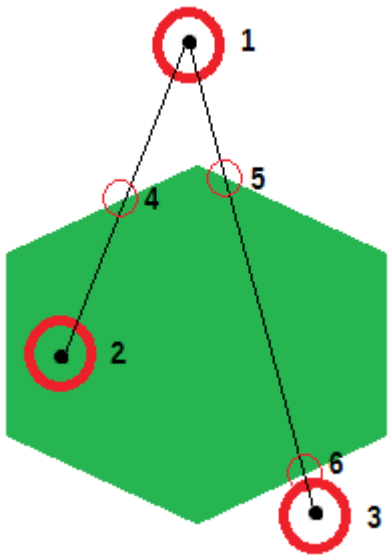

Illustration 4: kollisionscheck mellan två olika punkter och en polygon

För att kunna beräkna den här typen av kollisionsdetektering mellan en punkt och en polygon, måste man kunna räkna ut om två sträckor kolliderar. För kollisionsdetektering i exemplet ovan bör man, för punkt 2, för varje sida på polygonen, titta på om sidan kolliderar med sträckan mellan punkt 1 och punkt 2. Kollision mellan sträckor kan ses som en ekvation. Låt den ena sträckans ändpunkter benämnas v1 och v2. Om v ses som en vektor får den då värdet v2 – v1. Låt den andra sträckans ändpunkter benämnas u1 och u2. Samma relation gäller för dem, u, sedd som en vektor, blir differensen mellan u2 och u1. Låt s och t vara två variabler. Ekvationen kan ses som att kollision sker då $v_1 + s * v = u_1 + t * u$ förutsatt att s

och t båda är i intervallet [0, 1]. Notera att om s har värdet 1, skall $v_1 + s*v = v_2$, vilket även gäller för t och u. Denna ekvation kan sedan ses som ett ekvationssystem, en ekvation per koordinataxel. De två ekvationerna blir då

$$\begin{Bmatrix} (1)\, v_{1x} + s*v_x = u_{1x} + t*u_x \\ (2)\, v_{1y} + s*v_y = u_{1y} + t*u_y \end{Bmatrix}$$

där beteckningen x och y på variablerna indikerar vilken komponent i punkten/vektorn de tillhör. Genom att modifiera en av dessa ekvationer så att s eller t står ensamt på sin sida, exempelvis som i ekvation 3 nedan, och därefter byta ut s:et i den andra ekvationen mot det där, så att det blir som i ekvation 4, uppnås en ekvation där det är möjligt att få t ensamt och endast beroende på komponenterna i de olika vektorerna, som i ekvation 5 nedan.

$$(3)\, s = \frac{(v_{1x} + t*v_x - u_{1x})}{u_x}$$

$$(4)\, u_y + \frac{(v_{1x} + t*v_x - u_{1x})}{u_x} * u_y = v_{1y} + t*v_y$$

$$(5)\, t = \frac{(v_{1y} - u_{1y} - v_{1x} * (\frac{u_y}{u_x}) + u_{1x} * (\frac{u_y}{u_x}))}{(v_x * (\frac{u_y}{u_x}) - v_y)}$$

Därefter kan s beräknas från t med hjälp av ekvation 3, och sedan kan s och t jämföras för att se om de är inom det godkända intervallet [0, 1].

4.2 Implementation

Implementationen gjordes i c# med spelmotorn Unity. Programmet körs i 2d, då hela studien bygger på koncept i en 2d-rymd. På grund av detta är hela projektet i 2d. All rendering görs med Unitys inbyggda grafikbibliotek GL.

4.2.1 Övergripande

Programmet innehåller sammanlagt 8 filer med programkod som inte producerats av unity eller visual studio. Dessa listas och förklaras kort nedan:

1. Edge.cs: en fil innehållande en klass som representerar en sträcka (två punkter i rymden)
2. Field.cs: en fil innehållande en klass som representerar en polygon
3. GenderedGeneticAlgorithm.cs: en fil innehållande klass som representerar den genetiska algoritmen med kön. Ärver från GeneticAlgorithm
4. GeneticAlgorithm.cs: en fil innehållande en klass som representerar en standard genetisk algoritm
5. Path.cs: en fil innehållande en klass som representerar en väg (serie med vägpunkter)
6. PolygonGenerator.cs: en fil innehållande en klass som handhar genereringen av polygonerna

7. Renderclass.cs: en fil innehållande en klass som innehåller alla programmets programkonstanter

8. World.cs: en fil innehållande en klass som representerar världen som simuleras

Första steget i programmet är att generera en värld. Därefter slumpas populationer i de genetiska algoritmerna ut, vägar kartläggs till genomen, och genomen får fitnessvärden utefter hur bra vägarna är. Därefter händer allt i en loop, där de två genetiska algoritmerna itereras, och vägarna uppdateras

4.2.2 Polygongenerering

Polygongenereringen är en komplex del av programmet, som endast körs i början av varje körning, då den är ansvarig för genereringen av världen, och världen inte uppdateras under körning. Den representeras av en klass som bör, på grund av sin design, instantieras på nytt varje gång en ny omgång med polygoner ska genereras. När objektet instantieras anges hur många polygon som önskas. Polygonerna genereras i det 2-dimensionella intervallet [0, 1]. Detta innebär att en polygon kan vara som max 1 enhet på x-axeln, och 1 enhet på y-axeln. Ett gemensamt intervall för polygoner och vägar är nödvändigt då de hanteras tillsammans, i samma rymd. Polygonerna hade kunnat genereras i intervallet som utgörs av fönstret, men då fönstrets önskade storlek kan förändras, och då det egentligen inte är relevant för studien, valdes ett mer normaliserat intervall. Första steget i genereringen är att skapa den första polygonen, som då innefattar hela världen. Ingen polygon kommer någonsin vara utanför den ursprungliga polygonen på något vis. Den ursprungliga polygonen blir då en kvadrat med sidan 1.

Genereringen går till så att den delar upp den största existerande polygonen i två mindre delar tills dess att önskad mängd polygoner uppnåtts. Uppdelningen går till så att algoritmen hittar de två längsta sidorna på polygonen, slumpar ut två punkter, en på vardera sida, och drar en sida mellan dessa två punkter. På så vis delas polygonen upp i två nya polygoner. Sista steget i skapandet av nya polygoner är att tilldela dem en färdkostnad, vilken slumpas ut i ett programkonstant intervall.

4.2.3 De genetiska algoritmernas uppdatering

En genetisk algoritm består som nämnts tidigare av en population med genom. Resultatet av varje iteration av dessa kallas en generation. Varje iteration går till så att algoritmen genererar en helt ny population utifrån den förestående. Detta går till genom att algoritmen använder en selektionsoperator, som särskiljer sig från vardera algoritm, till att välja ut två föräldrar. Dessa två föräldrar används sedan till att skapa ett nytt genom, genom att slumpmässigt välja en punkt längs genomets gensekvens. Alla gener före den punkten tilldelas från den ena föräldern, och alla gener efter den punkten tilldelas från den andra föräldern.

Efter att den nya generationen har genererats, itereras samtliga nya genom igenom, och för varje genom slumpas ett värde ut. Är värdet över en konstant som kallas mutationschansen kommer genomet genomgå en mutation. Mutationen går till så att en slumpmässig gen väljs ut, som modifieras slumpmässigt baserat på hur mycket den kan modifieras (genen måste hålla sig inom intervallet) samt en konstant som kallas mutationsfaktorn. Efter detta ersätts den gamla generationen med den nya, och populationens fitnessvärden uppdateras.

4.2.4 Genom: mer specifikt

Genomet består av en gensekvens, som representeras som en serie med flyttal. Dessa flyttal hålls för enkelhetens skull i intervallet [0, 1]. Det är vanligt att genom representeras som bitsträngar, men då problemet som skall lösas har en kontinuerlig natur, är flyttal ett bra alternativ. Då talen kan kartläggas om till vilket intervall som helst valdes det mest grundläggande intervallet: [0, 1]. När den första generationen skapas har inga genom föräldrar. Då slumpas gensekvenserna ut. Observera att längden på gensekvenserna är en programkonstant som alltid är delbar med två. Syftet med den begränsningen härrör sig i att gensekvenserna ska kartläggas till 2-dimensionella punkter, vilket kan läsas mer om i 4.2.5.

Det övervägdes att låta de två olika genetiska algoritmerna ha varsin typ av genom, så att en klassstruktur i likhet med den som syns i Illustration 5 nedan skulle uppnås. Dock ansågs detta öka komplexiteten onödigt, eftersom könsfördelningen ansågs kunna hanteras annorlunda, så idén övergavs.

Det övervägdes även hur fitnessvärden skulle sparas som smidigast, då de är okända vid skapning, och då genomen i sig inte är inblandade i utvärderingen. Det bestämdes att fitnessvärdena skulle sparas i genomen, då detta överensstämmer med objektorienterad design.

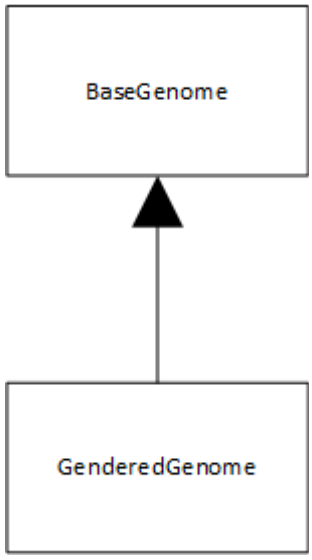

Illustration 5: koncept klassdiagram genom

4.2.5 Från genom till vägpunktsserie

När kartläggningen från genom till väg sker, tolkas varje par av gener i genomet om till en koordinat i rymden. Observera dock att den rymden är i samma intervall som generna, så kartläggningen sparar helt enkelt genernas värden som 2d-punkter, och använder dessa för att konstruera en väg.

Ett alternativ hade varit att spara vägarna i en annan rymd, alltså med ett annat intervall för koordinaterna. Då även polygonerna, som nämns i stycke 4.2.2, sparas i intervallet [0, 1] övergavs den idén, då det skulle medföra onödiga komplikationer med att konvertera från ett intervall till ett annat.

4.2.6 Genetisk algoritm med respektive utan kön: skillnader och likheter

De två genetiska algoritmerna implementerades som två separata klasser, där den genetiska algoritmen med kön ärver från den utan kön. Den enda funktionaliteten som skiljer, är funktionen för urval av genom för föräldrar till den nya generationen. I originalalgoritmen implementeras den som en roulettehjulsalgoritm. Detta innebär att samtliga genom i populationen har en chans att bli utvald, dock med lägre chans ju lägre fitnessvärde de har. Samtliga fitnessvärden i populationen summeras. Ett tal slumpas ut mellan noll och summan av alla fitnessvärden. Därefter görs för varje genom: räkna ut summan av alla fitnessvärden för genomen som redan är kollade, plus fitnessvärdet för det här genomet. Är slumptalet lägre än det uträknade värdet, väljs genomet, och returneras. För den genetiska algoritmen med kön fungerar det likadant, förutom att algoritmen helt ignorerar ena halvan av populationen. Vilken halva som ignoreras beror på hur många gånger funktionen har körts. Vid jämnt antal körningar ignoreras andra halvan, vid udda, första halvan.

4.2.7 Designdiskussion

I det här stycket diskuteras ett urval av intressanta och relevanta designbeslut som togs under utvecklingsprocessen.

När polygongeneratorn och världen skulle utvecklas och implementerades behövde världens storlek bestämmas. En lösning var att definiera världen som lika stor som skärmen, vilket följer en logik, då det är där den renderas. Dock är skärmen inte en konstant storlek (alla skärmar är inte lika stora) och i själva verket helt orelaterad till studien. En annan lösning som övervägdes var att låta världen vara exakt 1*1 enhet, vilket skulle innebära att alla polygon har hörn i intervallet [0, 1]. Det här intervallet övervägdes då det är normaliserat, och kräver färre operationer för konvertering till ett annat intervall (för konvertering mellan exempelvis [0, 2] och [0, 3], krävs 2 operationer, dividera talet med 2 och multiplicera talet med 3), vilket eventuellt kan spara tid och minska/undvika komplikationer. Den andra lösningen valdes då det kan minska antalet operationer, samtidigt som studien, som nämnts ovan, är helt oberoende av vilket intervall som används.

Vid implementation av multipla genetiska algoritmer finns det olika lösningar till klasshierarkin: de två som övervägdes illustreras i Illustration 6 och Illustration 7 nedan.

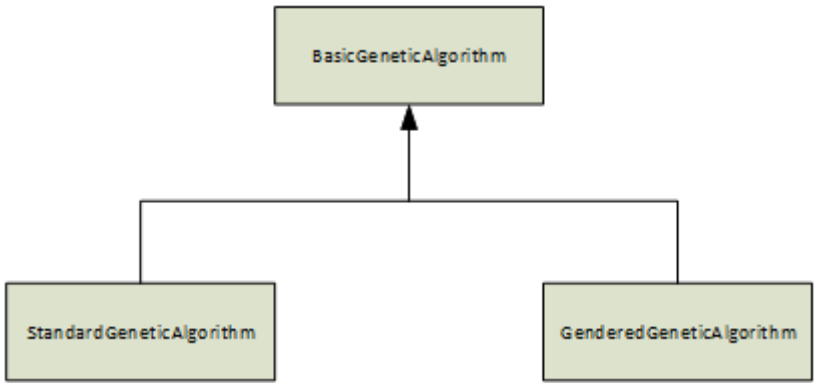

Illustration 6: exempel på klasshierarki för genetiska algoritmer: en basklass och två subklasser

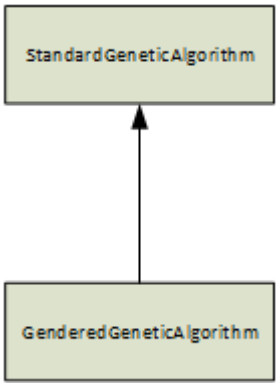

Illustration 7: Exempel på klasshierarki för genetiska algoritmer: en virtuell basklass som symboliserar en grundläggande algoritm, och en subklass som modifierar den

Den första lösningen beskrivs som en abstrakt superklass som endast implementerar det mest grundläggande och gemensamma för de två genetiska algoritmerna. Två klasser som ärver från den representerar de två genetiska algoritmerna och implementerar algoritmspecifik funktionalitet, vilket bland annat innefattar en selektionsoperator, då den är unik för varje algoritm. Den andra lösningen innnefattar en mer simplistisk hierarki, där det finns en grundläggande genetisk algoritm som konkret superklass, som alltså implementerar standardalgoritmen. Viss funktionalitet kan skrivas om i en subklass, men det mesta är spikat. Subklassen representerar den könsbestämda genetiska algoritmen och skriver över viss funktionalitet från superklassen, vilket bland annat innefattar selektionsoperatorn. Den andra lösningen valdes, då problemet kräver ytterst lite algoritmspecifik funktionalitet, som enkelt kan skrivas över i en subklass.

Genomen till algoritmerna kunde även de implementeras på olika sätt. Den första lösningen som övervägdes var att låta genomen implementeras med en liknande klasshierarki som visas i Illustration 6. Detta skulle innebära en abstrakt superklass där de centrala begreppen implementeras, samt en subklass per genetisk algoritm, där de algoritmspecifika egenskaperna hos genomen implementeras. Det andra alternativet som övervägdes skulle ge en klasshierarki som liknar den i Illustration 7, alltså en konkret superklass som representerar genomen för den klassiska genetiska algoritmen, samt en subklass som representerar genomen för den könsbestämda genetiska algoritmen. I det fallet skulle könet på genomen sparas i genomen, och selektionsoperatorn i den könsbestämda genetiska algoritmen skulle använda det värdet för att bestämma vilket genom den skulle välja. Det tredje alternativet som övervägdes innefattade endast en klass för genom, där könet inte sparas i genominstansen. Könet bestäms istället av vilken del av populationen den befinner sig i. Genom i den första halvan av populationen klassas som manliga, de i den andra halvan klassas som kvinnliga. Implementationen av genom innefattar då endast överkorsningsoperatorn, samt gensekvensen. Då skillnaderna i implementationerna av de olika klasserna i lösning 1 och 2 är ytterst få, valdes lösning 3 istället, då den är simplast, samtidigt som den löser problemet fullständigt.

4.3 Pilotstudie

För att säkerställa att produkten kan användas för att utföra den studie som beskrivs i kapitel 3, har ett pilottest gjorts, där alla data samlats in för ett problem på mycket mindre skala. Simuleringen kördes under kortare tid, och lägre värden på programkonstanter användes, i syfte att minska mängden data. Då pilotsstudien endast ska visa på att produkten uppfyller kraven, kan fokus läggas på det, istället för att faktiskt utföra en studie i full skala. Observera att endast programkonstanter har ändrats. Dessa modifieras enkelt till en större skala, utan behov av omkompilering eller liknande.

Vid körning av programmet genereras två textfiler med namnet ”originalGAOutput.txt” och ”genderedGAOutput.txt”. Dessa filer hamnar i mappen output i samma mapp som .exe-filen. I dem finns data för varje enskilt genom för respektive algoritm, över hela körningstiden. Datan som sparas är genomets gensekvens, följt av genomets fitnessvärde. Efter det kommer ett ”;” (semikolon) och en radbrytning som skiljer varje genom åt. Varje generation delas upp med en radbrytning, så varje generation är i sitt eget stycke.

Denna data kan sedan evalueras och studeras. I Illustration 8 syns en graf över resultatet för en exempelkörning: genomsnittsfitnessvärde för en genetisk algoritm med kön över tid. Observera att körningen gjordes med en kort gensekvens, ett fåtal genom, samt få polygoner för att minska mängden data. I Illustration 9 kan man se samma sak för en genetisk algoritm utan kön. Då även själva gensekvenserna är med kan man också titta på den genetiska spridningen för algoritmerna, om det skulle vara intressant.

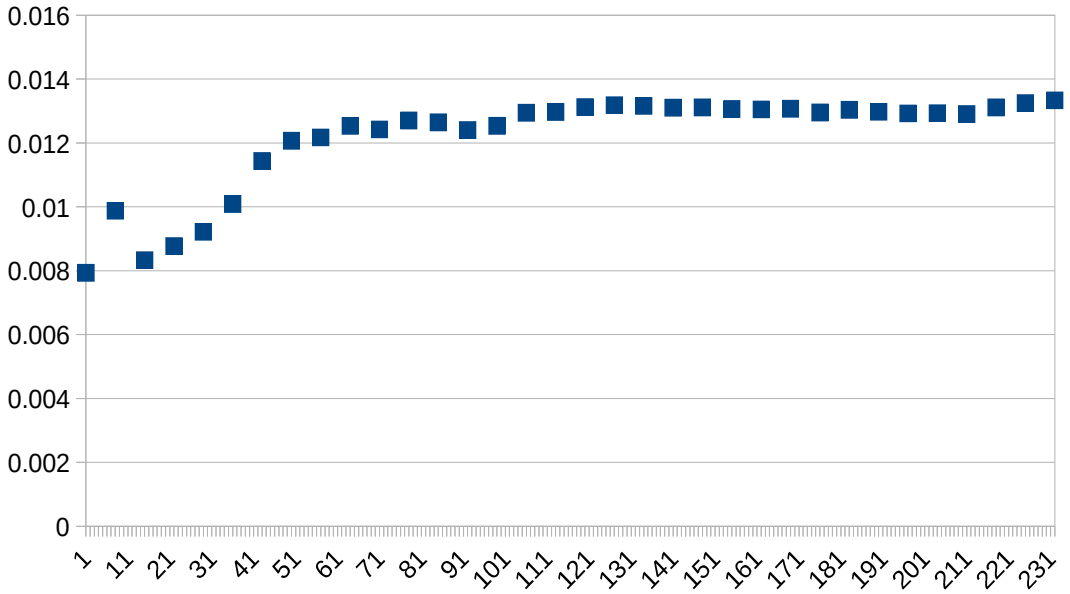

Illustration 8: genomsnitligt fitnessvärde över tid för genetisk algoritm med kön

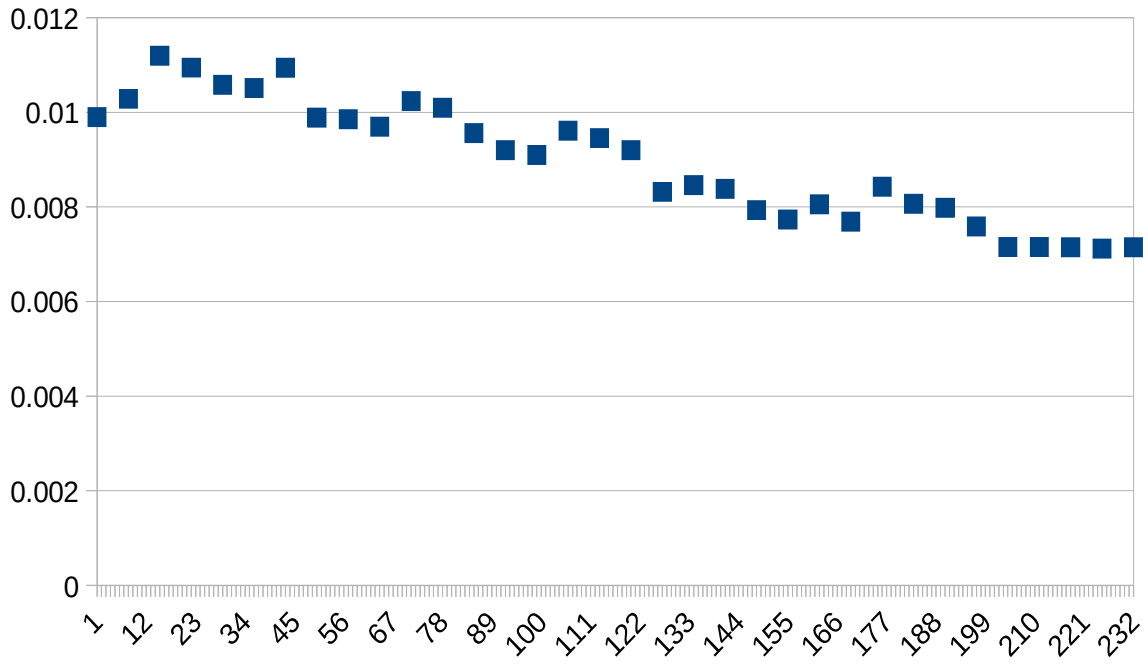

Illustration 9: genomsnittligt fitnessvärde över tid för genetisk algoritm utan kön

4.3.1 Analys

Grafen i Illustration 8 tyder på att den genetiska algoritmen med kön började med sämre lösningar och tidigt hade en kraftig stigning i lösningskvalitét för att snabbt stanna av på ett värde över 0.012. I Illustration 9 syns data från den genetiska algoritmen utan kön, och där syns en svag nedstigning ända från början, ner till under 0.008, vilket tyder på att kvalitén på lösningarna långsamt försämrades.

4.3.2 Diskussion

Baserat på den data som presenteras, kan det konstateras att den genetiska algoritmen med kön snabbt producerar bättre resultat än vad som började med, men även snabbt konvergerar, då fitness-värdet snabbt slutade utvecklas, vilket kan tyda på att algoritmen stannat. Då den genetiska algoritmen med kön snabbt producerade resultat som var bättre än de från den genetiska algoritmen utan kön kan det konstateras att den genetiska algoritmen med kön ger bättre lösningar på kortare tid, men att den också konvergerar fortare. Observera att denna data är väldigt bristfällig, och med största sannolikhet inte alls tillåter en korrekt slutsats, då fitnessvärdena för den genetiska algoritmen utan kön faktiskt sjunker med tiden. Detta är inte alls ett typiskt beteende för genetiska algoritmer, och tyder på att den genetiska diversiteten är för låg (för få individer i populationen). Då genetiska algoritmer bygger sin framgång på genetisk diversitet kan icke-positiva resultat förväntas om den genetiska diversiteten är för låg.

En körning gjordes med större värden på programkonstanterna (längre gensekvens, fler antal polygoner, samt större population). Tiden mättes på denna körning, som hade de programkonstanter som kommer användas för studien, med den skillnaden att körningen inte avslutades. Totalt tog körningen ca 81 sekunder, vilket tyder på att tidskomplexiteten på artefakten är dålig (O(n^2) eller värre), men att en fullständig körning likväl är möjlig att utföra, om än tidskrävande. Med detta menas att en komplett körning kan genomföras på mindre än en dag. Värt att notera är också att den körningen genererade ca 60 MB av data,

vilket tyder på att en fullständigt komplett körning skulle generera många gånger mer data, vilket kan bli tidskrävande att processera.

5 Utvärdering

I det här kapitlet presenteras den undersökning som gjorts utförligt, till stor del i form av grafer. Observera att endast den relevanta datan presenteras, då undersökningen fokuserar på prestation (alltså hur bra algoritmerna löser problemet) läggs all fokus på hur fitness-värdena utvecklas över tid. Den genetiska diversiteten visas inte upp i det här kapitlet, men senare kapitel diskuterar hur diversitet skulle kunna mätas för ett sådant här problem. I det här kapitlet tas även en analys av datan upp, där körningen beskrivs i ordform. I slutet av kapitlet redovisas de slutsatser som kan dras från datan.

5.1 Presentation av undersökningen

Två simuleringar gjordes, en där en genetisk algoritm utan kön kördes tillsammans med en genetisk algoritm med kön där könsfördelningen var fast. I samtliga fall bestod populationen av 1000 individer, vars respektive gensekvens bestod av 10 värden. Varje population itererades över exakt 2185 gånger, vilket resulterade i 2185 generationer per algoritm och körning. Hur fitness-värdena för de två algoritmerna i den första körningen utvecklades över tid, kan studeras i diagrammen i Illustration 10, Illustration 11 och Illustration 12 nedan.

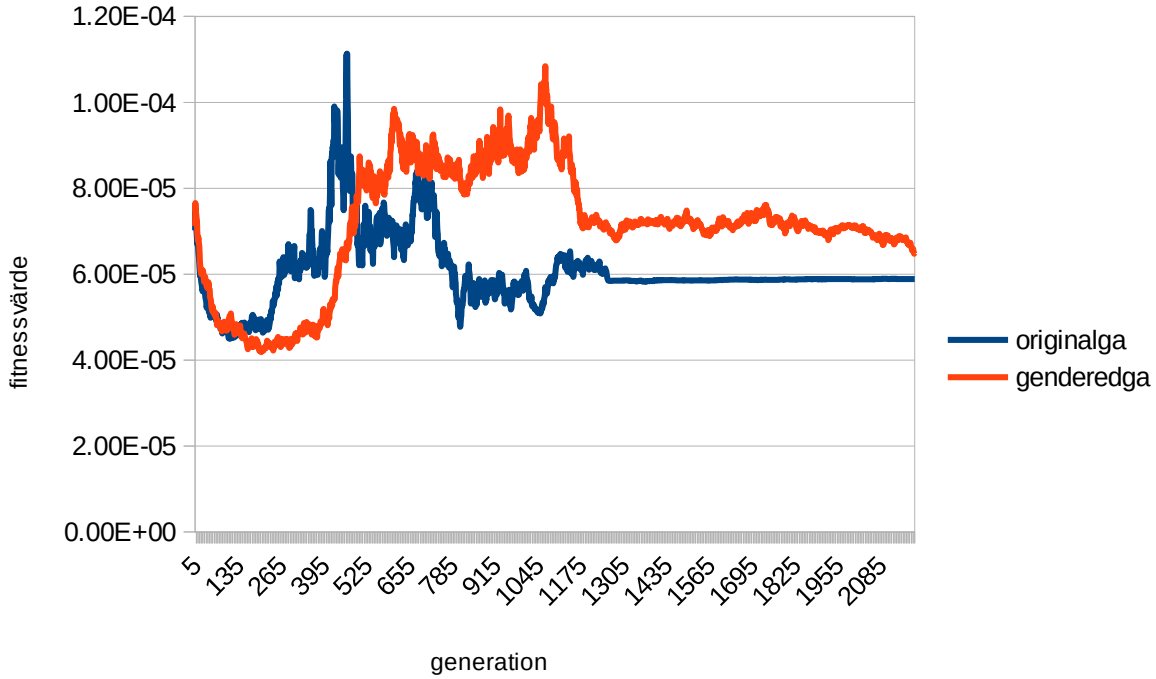

Illustration 10: genomsnitsfitnessvärde för GA med och utan kön med fast könsdistribuering

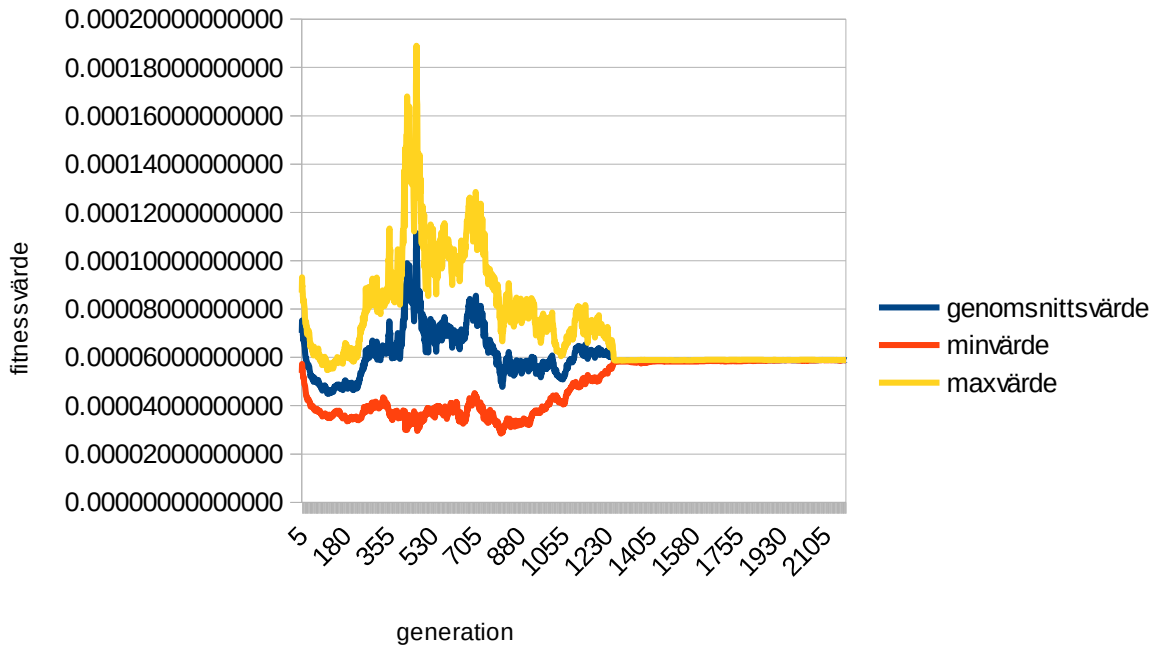

Illustration 11: standardavvikelse för GA utan kön, första körningen

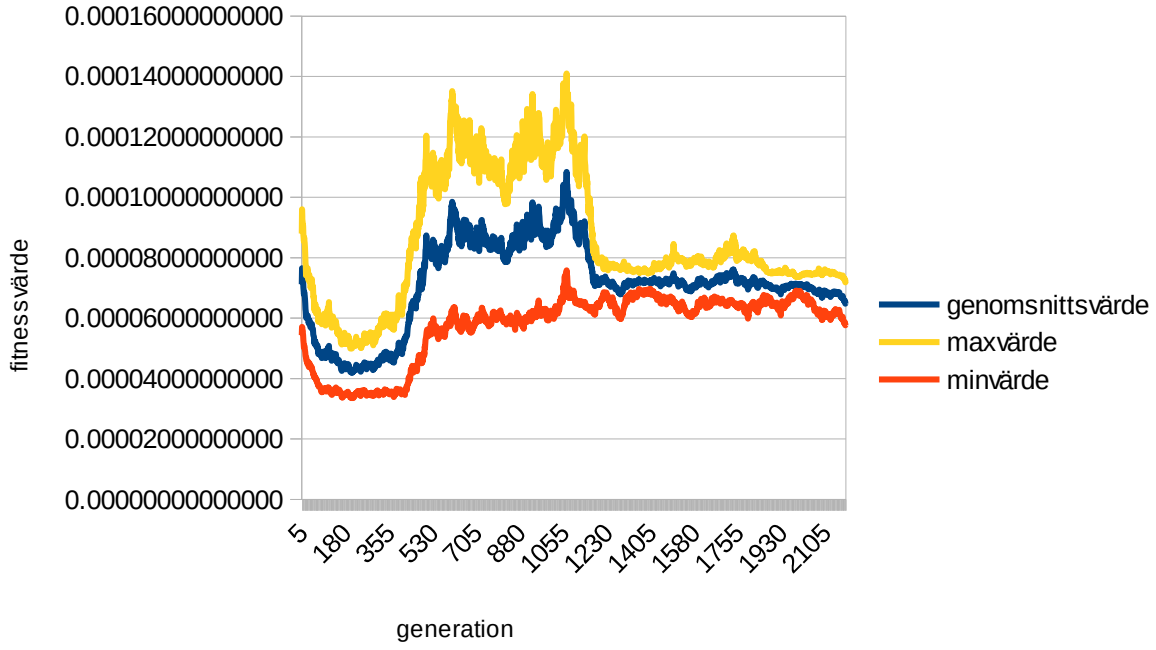

Illustration 12: standardavvikelse för GA med kön och fast könsfördelning

Hur fitnessvärdena för algoritmerna i den andra körningen utvecklades, kan ses i Illustration 13, Illustration 14 och Illustration 15 nedan.

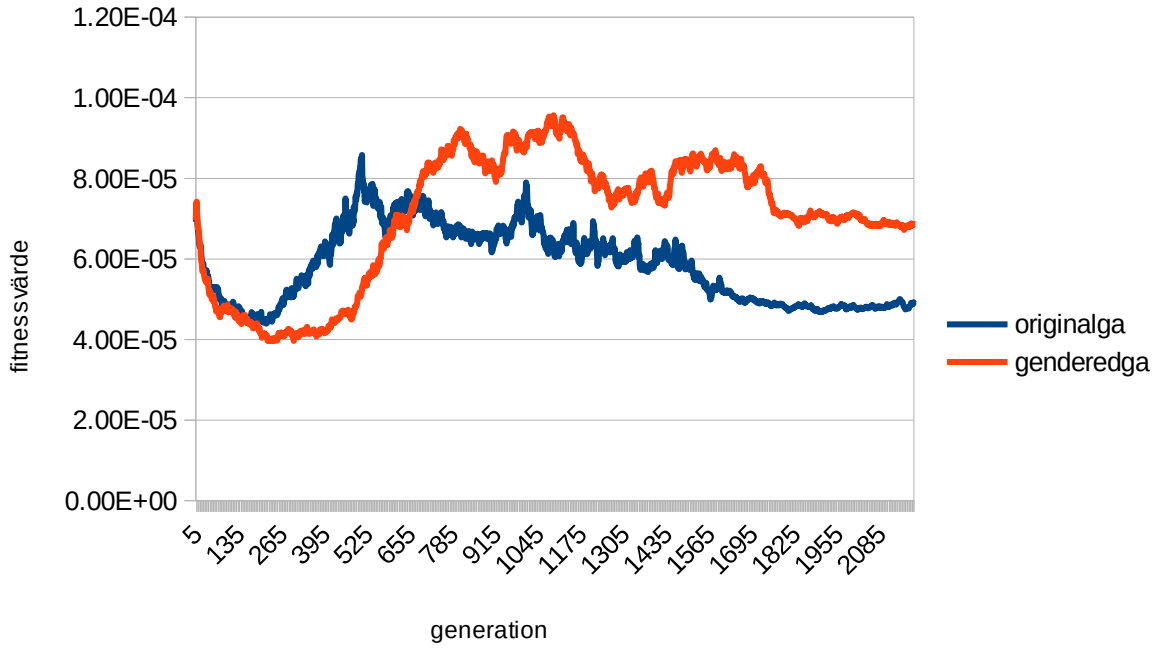

Illustration 13: genomsnittsfitnessvärden för GA utan kön, samt GA med slumpmässig könsfördelning

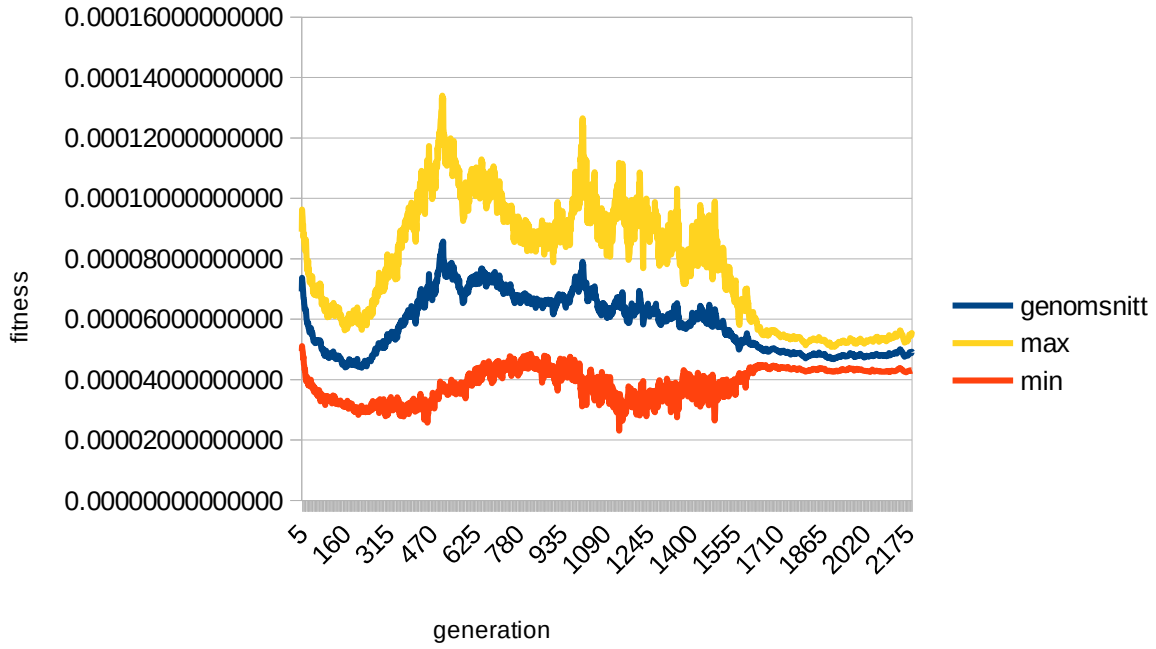

Illustration 14: standardavvikelse för GA utan kön, andra körningen

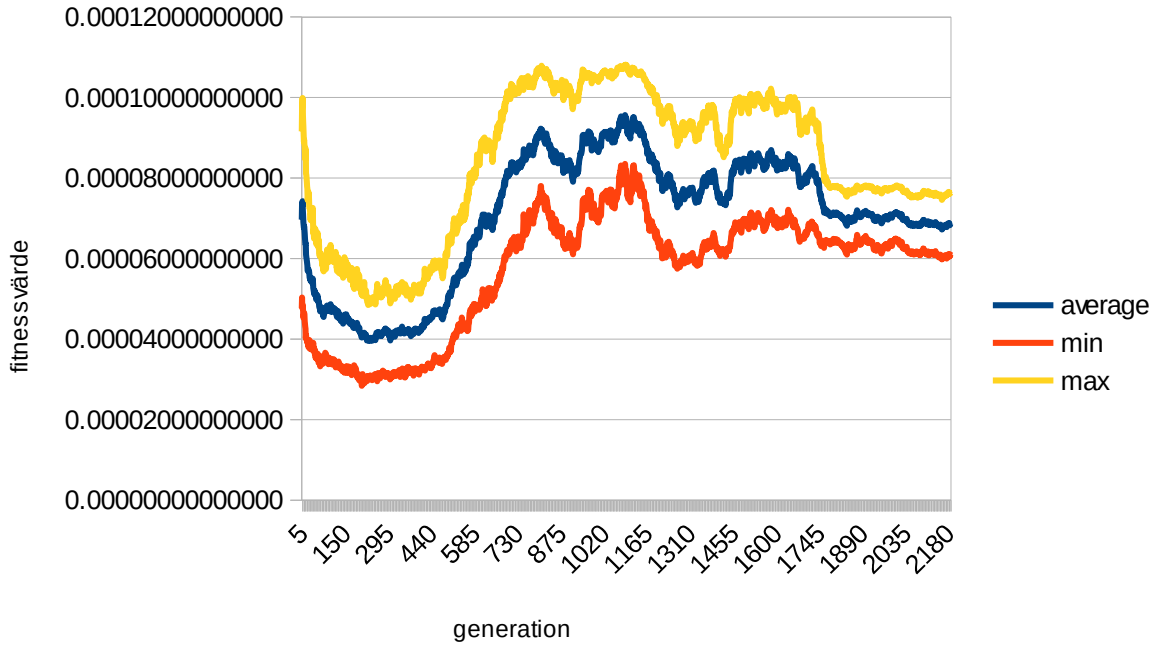

Illustration 15: standardavvikelse för GA med kön och slumpmässig könsfördelning

5.2 Analys

Som kan ses i Illustration 10, Illustration 11 och Illustration 12 ovan, nådde standardavvikelsen en ungefärlig lägstanivå ungefär samtidigt för båda algoritmerna (ungefär vid den 1230:e generationen). Att standardavvikelsen sjönk så drastiskt tyder på att algoritmerna konvergerade. Då ingen av algoritmerna faktiskt nådde en standardavvikelse på 0, stannade ingen av dem.

I Illustration 13, Illustration 14 och Illustration 15 ovan ser man som nämnt hur fitnessvärdena evolverade över tid för den andra körningen. Man kan se i Illustration 13 att den genetiska algoritmen med kön hade ett mycket högre genomsnittligt fitnessvärde än den genetiska algoritmen utan kön I Illustration 14 och Illustration 15. Ovan ser man ytterligare tecken på att detta stämmer då den undre gränsen för standardavvikelsen i Illustration 15 är över 0.00006, medans den övre gränsen för standardavvikelsen i Illustration 14 är under samma värde. I Illustration 14 och Illustration 15 ovan ser man att den genetiska algoritmen utan kön förlorade diversitet väldigt kraftigt, medan den genetiska algoritmen med kön höll en till synes väldigt jämn diversitet över samtliga generationer.

5.3 Slutsatser

Utgående från den analys som gjordes i kapitel 5.2, kan man konstatera att för det här problemet presterade den genetiska algoritmen utan kön sämst av alla tre, om man tittar på konvergens. Samtliga algoritmer förutom den med slumpmässig könsdistribution konvergerade till en viss grad. Detta tyder på att införandet av kön och fast könsfördelning

direkt medförde en lägre tendens för konvergens. Dock är detta inte den aspekten som är intressant, då den här studien siktar på att svara på om algoritmen ger generationer med högre fitnessvärde. Om man tittar på den aspekten är de genetiska algoritmerna med kön överlägset bättre än den utan kön, då båda slutade på ett ungefärligt genomsnittsfitnessvärde på 7.0E-05, medan den utan kön slutade på ungefär 5.0E-05 respektive 6.0E-05. Detta visar att införandet av kön gav en positiv effekt, medan införandet av slumpmässig könsdistribuering inte medgav någon märkbar effekt på fitnessvärdena.

6 Avslutande diskussion

I det här kapitlet sammanfattas studien, från bakgrund, till utvärdering. Sammanfattningen går kort igenom vad studien bygger på, vad frågeställningen är, hur genomförandet gick till och vad arbetet resulterade i. Vidare följer ett delkapitel som diskuterar problemets och resultatets nytta i samhället, men som även gör kopplingar till andra studier. Diskussionskapitlet tar även upp de etiska aspekter som berör både frågeställningen, problemet, och resultatet. Slutligen följer delkapitlet framtida arbete, som fokuserar på hur man går vidare med den här studien, och bygger vidare på resultaten.

6.1 Sammanfattning

Genetiska algoritmer tillåter ett program att hitta en kanske inte optimal lösning till ett problem, men åtminstone en tillräckligt bra lösning till ett problem, vars lösningsrymd är alltför stor för att man ska kunna hitta en optimal lösning inom rimlig tid. Den här rapporten studerar vad som händer om man modifierar en genetisk algoritm genom att lägga till en könsaspekt, likt vad Velazco & Bullinaria (2003) gjorde i sin studie, att införa att varje gång ett nytt genom skapas, måste en förälder vara av det ena könet, och en förälder vara av det andra könet. Syftet med denna modifiering var att förhindra prematur konvergens, och utfallet som redovisas i kapitel 5 visar att förändringen resulterade i en lägre tendens för konvergens. Modifieringen medförde även att bättre resultat producerades, vilket var önskvärt. Skillnaden mot Velazco & Bullinarias studie är att denna även tittar på konsekvenserna av att införa slumpmässig könsbalansering, så att varje genom tilldelas ett kön slumpmässigt, vilket gav väldigt omärkliga konsekvenser, om man jämför med en genetisk algoritm utan kön.

6.2 Diskussion

En intressant aspekt att titta på i resultatet är när fitness-värdena sammanfaller i den genetiska algoritmen med kön och fast fördelning. Detta beskrivs i studien som en konvergens, algoritmens alla individer blir endast kloner av varandra, och vidare iterationer resulterar i lite eller inget nytt. Algoritmen har fastnat i ett lokalt optima. Observera att det vi ser i grafen inte nödvändigtvis är en algoritm som fastnat i ett lokalt optima, då lösningsrymden inte nödvändigtvis har ett lokalt optima på just den nivån, det kan finnas flera. Detta visar att fitness-värden inte kan visa allt om en genetisk algoritm. För att faktiskt kunna fastställa om en algoritm fastnat i ett lokalt optima måste man titta på den faktiska genetiska diversiteten. Hur detta kan göras diskuteras i kapitlet om framtida arbete.

Då samtliga verktyg som användes i arbetet antingen är gratis i icke-kommersiella syften, eller är egentillverkade, och eftersom allt arbete gjordes helt utan utomstående kontakter och påverkan, är det svårt att applicera etiska argument på arbetet. Dock kan resultatets påverkan på samhället diskuteras, men även här måste nämnas att eftersom problemet är helt abstrakt, och de genetiska algoritmerna endast kan ses som en finslipning av ett verktyg, kan de inte ses som något oetiskt, då verktyget redan från början var för abstrakt för att kunna klassas på en sådan skala, och inte blev mindre abstrakt av vidareutvecklingen. Naturligtvis kan algoritmen användas till att lösa oetiska problem, men det kan knappast ses

som ett argument att förhindra den här typen av studier, då så gott som alla verktyg kan användas i oetiska ändamål.

Man kan även diskutera kring hur etiskt det är att bygga teknologi baserat på hur naturen fungerar. Genetiska algoritmer bygger på naturen, och införandet av kön medför ytterligare likheter. Detta skulle kunna ses som oetiskt, om man tänker sig att alla former av artificiellt liv är oetiskt. Dock är den här typen av algoritm så långt ifrån liv, att det blir ett väldigt långsökt argument, med låg vikt.

Tittar man på det här arbetet ur ett samhälleligt perspektiv är det tydligt att den här typen av studier endast kan utgöra en hörnsten i samhällets kunnande. Då den producerade artefakten är specialiserad för den här typen av problem, är den helt värdelös för andra syften. Detta innebär att det är studien i sig som är värdet i det här arbetet. Studien å andra sidan kan motiveras att indirekt vara en väldigt viktig del i samhällets kunnande om genetiska algoritmer, då den tittar på den mycket grundläggande aspekten vad införandet av kön medför för konsekvenser, för en specifik typ av problem. Notera att resultatet kanske kan ses som till stor del oviktigt i sig, då det endast fokuserar på en viss typ av problem, lösta med en viss typ av verktyg, men värdet hos studien består i att den kan utgöra en grund för vidare studier.

6.3 Framtida arbete

En intressant studie som skulle kunna göras är att titta på hur den genetiska diversiteten utvecklas över tid, och hur kön påverkar den genetiska diversiteten. En metod för att mäta den genetiska diversiteten i en generation skulle kunna vara att se alla gensekvenser som en n-dimensionell geometrisk vektor, där n är gensekvensens längd. På så vis kan man se gensekvensen som en punkt i n-rymden, från vilket man kan räkna ut en genomsnittlig punkt, och mäta avståndet från den punkten, till punkten som representeras av genomet. Genom att titta på hur det avståndet evolveras över tid kan man eventuellt dra slutsatser om hur den genetiska diversiteten utvecklas. Om man sedan jämför hur den genetiska diversiteten utvecklas över tid för en genetisk algoritm utan kön, och en genetisk algoritm med kön och fast könsfördelning, kan man se vad införandet av kön har för faktiska effekter på den genetiska diversiteten. Sedan kan man göra samma jämförelse mot en genetisk algoritm med kön och slumpmässig könsfördelning.

Vidare kan man även titta på vad införandet av genom har för effekt på genetiska algoritmer överlag, det vill säga om man applicerar dem på andra typer av problem, och försöker dra en generell slutsats. En sådan studie skulle troligen peka på samma sak som denna, då en förändring av problem som algoritmen appliceras på troligen inte resulterar i förändring i hur algoritm beter sig, då problemet alltid appliceras till en lösningsrymd med en viss dimension.

Referenser

Cezary, Z. J. & Zbigniew, M (1991). An experimental comparison of binary and floating point representations in genetic algorithms. I Belew, R. K & Booker, L. B. (red.) *Proceedings of the fourth international conference on Genetic algorithms*. International Conference on Genetic Algorithms. San Diego, USA juli 1991, ss. 31-36.

Herrera, F. & Lozano, M. (2000). Gradual distributed real-coded genetic algorithms. *IEEE Transactions on Evolutionary Computation,* 4(1), ss. 43-63.

Mecki (2008). How can I determine whether a 2d point is within a polygon?. *Stackoverflow* [forum], 20 oktober. http://stackoverflow.com/a/218081 [2016-04-04]

Nguyen Thi, H & Nguyen Xuan, H. (2008) A Brief Overview of Population Diversity Measures in Genetic Programming. I Pham, T.l., et al. (red.) *Proceedings of the third Asian-pacific Workshop on Genetic Programming*. Asia-pacific workshop on genetic programming. Hanoi, Vietnam 12-14 oktober 2006, ss. 128-139.

Sánchez-Velazco, J & Bullinaria, J. A. (2003). Sexual selection with competitive/cooperative operators for genetic algorithms. I Castillo, O. (red.) *Proceedings of the IASTED International Conference on Neural Networks and Computational Intelligence*. IASTED International Conference on Neural Networks and Computational Intelligence. Cancun, Mexico maj 19-21 2003, ss. 191-196.

Unity Technologies (2005-2016). *Unity* (Version 5.2.3) [datorprogram]. Tillgänglig på Internet: https://unity3d.com/

Xkcd 1605 (2015). a webcomic of romance, sarcasm, math and language. Tillgänglig på Internet: https://xkcd.com/1605/ [hämtad 1 augusti 2016].