Distribuerte evolusjonære mekanismer og organisering

Tore Halvorsen
Seksjon for intelligente systemer,
Institutt for datateknikk og informasjonsvitenskap, NTNU

6. september 2004

Sammendrag

Oppgaven tar for seg noen av problemstillingene som oppstår hvis man forsøker å distribuere en evolusjon ut i flere parallelle prosesser. Den ser først og fremst på de ulike måtene subpopulasjoner kan kommunisere, og hvordan parametre for kommunikasjonen kan påvirke hvor raskt man finner fram til gode resultater. Det skal være mulig å utnytte distribueringen til å bygge et super-skalart system. Det ble ikke funnet et super-skalart system i denne oppgaven.

Innhold

Figurer

Tabeller

1 Introduksjon

I denne oppgaven forsøker jeg å se på ulike egenskaper ved distribuerte evolusjonære systemer, med særlig basering i ulike migrasjonsmønster. Dette vil enkelt si at vi skal dele en evolusjon opp i en del mindre biter, og sende løsningsforslag (individer) mellom de ulike bitene. Hensikten med en oppdeling er at flere oppgaver da kan utføres i parallell, og følgelig bør en enkelt oppgave kunne utføres på mindre reell tid. Videre kan parallelliseringen utnyttes algoritmisk. Oppgaven skal ta for seg en variasjon av flere egenskaper ved distribuerte systemer, blandt annet migrasjonmønster, for å finne ut hvilke metoder som best utnytter denne parallelliseringen.

I oppgaven blir det brukt en god del engelske uttrykk på konsepter som er forholdsvis fag-spesifikke. Det er også en viss blanding av biologiske og informasjonsteknologiske konsepter, fordi denne oppgaven tar for seg en metode som er biologisk inspirert.

1.1 Hva er et evolusjonært system?

Et evolusjonært system er en heuristikk som bruker prinsippet om evolusjon for å finne gode løsninger på et spesifisert problem. Dette gjøres ved hjelp av et prinsipp som kalles naturlig utvalg (*natural selection*). Dette prinsippet kan ligne litt på prøv-og-feil (*trial-and-error)*, men er mye mer effektivt. Evolusjon har mange egenskaper som gjør søket mye mer effektivt enn et slikt tilfeldig søk. Det er vist at dette fungerer for enkle binærverdier i et monotont søkerom av Schema teoremet[4, 5][1].

Evolusjon vil i denne oppgaven bety en stokastisk iterativ prosess som baserer seg på rekombinasjon av del-løsninger (gener) for å finne en optimal løsning. At man over tid bygger sammen gener til et bedre resultat, er formulert i *Building Block Hypothesis.* Denne hypotesen beskriver hvordan evolusjon tenderer til å bygge sammen del-løsninger av mindre enheter. Disse del-løsningen

[1] Dette teoremet sier at evolusjon fungerer, altså at den prosessen vi kaller en evolusjon finner fram til svært gode resultater.

bygges igjen sammen til større del-løsninger helt til man får en komplett løsning. En biologisk ekvivalent kan ses på som ulike egenskaper ved organismer som kan settes sammen til en fornuftig helhet. (Se også 1.3.5), hvor gener brukes som byggeklosser som til slutt danner noe stort.

1.1.1 Utforske & Utnytte

Utforske (*Explore*) og utnytte (*Exploit*) *er* viktige begreper innen evolusjonær søking. De beskriver egentlig i hvor stor grad mulige løsninger kan forandre seg for hvert steg. Utforsking vil si i hvor stor grad man undersøker hele søkerommet. Et godt mål på utforsking er hvor stor bevegelsesfrihet en potensiell løsning har. Hvis en løsning har mulighet til store forandringer vil den kunne gjøre store hopp i søkerommet, og har med andre ord stor bevegelsesfrihet. Typisk for en stor utforskningsgrad er at mulige løsninger ligger rimelig jevnt fordelt i hele søkerommet.

Etterhvert bør denne bevegelsesfriheten begrenses så systemet konsenterer seg om områder i søkerommet hvor man har funnet gode løsninger. Dette gjøres ved at løsningene ikke lenger har muligheten til å gjøre store hopp i søkerommet. Potensielle løsninger får med andre ord bare lov til å forandre mindre deler av seg selv. Typisk for en stor utnyttingsgrad er at mulige løsninger har konsentrert seg i små klynger.

Hvis balansegangen mellom utforsking og utnytting er skjevt fordelt vil man typisk ikke finne de globalt beste løsningene. Dette har to separate grunner:

For lav utnyttingsgrad Systemet vil lete forholdsvis blindt i hele søkerommet og klarer ikke spesialisere løsningene sine. Selv om systemet kan være inne på riktig spor, mangler det spesialiseringen som trengs for en god løsning.

For lav utforskningsgrad Systemet forsøker for tidlig å spesialisere seg, og følger et spor som ikke kan lede fram til en god løsning. Dette fenomenet kalles *premature convergence*.

Graden av utnytting og utforskning kan gjerne forandre seg i løpet av en enkelt kjøring. I startfasen av problemløsningen bør man undersøke så mye som mulig av søkerommet før systemet utnytter de beste områdene som ble funnet. Denne overgangen er typisk rimelig gradvis ved at bevegelsesfriheten til potensielle løsninger sakte reduseres.

1.2 Gangen i en evolusjon

Evolusjon fungerer kort fortalt som følgende modell: Det genereres en startpopulasjon, alle individene i denne evalueres og deretter overføres disse individene på en eller annen måte til neste generasjon. Målet er at individene i snitt skal bli bedre for hver generasjonsovergang. Algoritmen fungerer ved at de beste individene har størst mulighet til å bli overført til neste generasjon, men dette er en stokastisk prosess og det vil derfor ikke være noen garanti for at de beste individene går videre til neste populasjon.

1.2.1 Generering av startpopulasjon

Det genereres et gitt antall tilfeldige mulige løsninger på problemet. Denne samlingen av mulige løsninger kalles en *population*, i dette tilfellet P_0, startpopulasjonen. Det forventes ikke at man finner en optimal løsning på dette trinnet. Deler av den optimale løsningen bør imidlertid være til stede allerede her. I følge *Building Block Hypothesis*[3] vil evolusjon virke på den måten at den bygger opp til en optimal løsning ved å bruke byggesteiner av løsningsforslag fra tidligere generasjoner.

1.2.2 Evaluering

Disse mulige løsningene testes/utprøves og får tildelt en verdi for hvor godt løsningen gjorde det. Dette skrittet kalles ofte evaluering *(evaluation)*. I mer kompliserte systemer kan det skilles mellom *genotype* (genmaterialet) og *fenotype* (genotypen uttrykt som en organisme). En phenotype testes da mot problemområdet, og resultatet registreres. Den noterte verdien kalles *fitness*.

Hensikten er å kunne rangere de ulike løsningsforslagene slik at det er mulig å selektere de beste. For denne oppgaven finnes det ingen faktisk forskjell mellom genotype og fenotype. I denne oppgaven slås derfor disse begrepene sammen til *individ*.

1.2.3 Overføring til neste generasjon

Etter at alle individer har blitt evaluert skal en ny populsjons bygges opp med den forrige som utgangspunkt. Vi er her interessert i å ta med oss de beste egenskapene. Vi skal med andre ord bruke P_0 til å lage P_1. Det finnes i det store og hele to metoder for at en populasjon kan få et nytt individ.

1. Kopiering
2. Kombinering

Derimot finnes det mange muligheter for å velge ut individer som skal brukes i disse trinnene.

1.2.3.1 Utvelging Fitness-verdien brukes som utgangspunkt for å plukke ut hvilke individer som skal ha en innvirkning på evolusjonen. Dette trinnet kalles utvelging *(Selection)*. Individer velges vanligvis ut på en stokastisk måte. Det finnes flere metoder for å velge individer. Her er noen eksempler:

Roulette Wheel Hvert individ får en bit av et lykkehjul tilsvarende hvor stor del av den totale fitnessen dette individet utgjør. Etter å ha satt opp lykkehjulet snurrer man dette og får et enkelt individ. Ulempen med denne metoden er at den er treg, man må nemlig gå gjennom populasjonen for hvert valgt individ. Videre takler den ikke negative fitness-verdier uten å skalere tallene på en eller annen måte.

Stochatic Universal Sampling (SUS) Som Roulette Wheel, men her snurrer man bare hjulet en gang. Så plukker man ut n punkter på hjulet og plukker ut de individene som inneholder disse punktene. Hensikten

er at selv individer med liten fitness skal komme med. Ulempen med dette er noe av det samme som for Roulette Wheel, men også at den har problemer med å kunne operere som en *stream*, altså at man enkelt når som helst kan spørre metoden om å plukke et nytt individ. Problemet oppstår ved at SUS må vite hvor mange individer man skal ha ut av seleksjonen på forhånd, og må derfor lagre mye mellom-resultater.

Random Man velger ut individer fullstendig på lykke og fromme. Ulempene her er at det ikke er noen tendens til å plukke ut gode individer.

Best Det beste individet velges ut. Kan utvides til å velge ut de n beste individene. Neste forespørsel til denne metoden kan da plukke ut det nest beste individet og så videre. Denne metoden brukes for å implementere *elitism* (forklart under).

Tournament Man velger ut to individer og så velger med en konstant sannsynlighet individet med høyest fitness. Denne metoden er rask, enkel å implementere og fungerer som en *stream*. Videre kan Tournament raskt utvides til å ta med flere individer (sette tre eller flere opp mot hverandre) eller kun ett enkelt. I ytterpunktene fungerer Tournament da både som Random (kun ett individ) og Elitism (alle individene, 100% sannsynlighet for å velge det beste).

I denne oppgaven har jeg i hovedsak valgt metoden som heter *Tournament Selection*. Denne metoden trenger ikke vite noe annet om populasjonen enn at det eksisterer en *random access*[2] til individene. Dette gjør også programmeringen enklere og mer effektiv.

1.2.3.2 Overføring Etter å ha valgt ut et individ, kommer beslutningen om hvordan informasjonen fra et individ skal påvirke den neste generasjonen.

[2]Vi kan få tilgang til et hvilket som helst individ til en hver tid. Alternativt eksisterer muligheten sekvensiell tilgang. En lenket liste passer med andre ord dårlig til å implementere populasjonen, fordi vi her bare har sekvensiell tilgang til individene.

Kopiering Individet kopieres over i neste generasjon P_{n+1}. Dette blir gjort for å bevare en del av individene på tvers av generasjoner. Man kan forestille seg at dette er individer som overlever, og hvor de som ikke kopieres inn i neste generasjon, dør.

Som spesialtilfelle kopieres de beste individene direkte over til neste generasjon. Dette er en deterministisk metode i motsetning til de andre, men en rent statisk. For hver generasjon P_n kopieres først de p% beste individene over til P_{n+1}. Dette blir gjort for å sikre at man aldri mister den beste løsningen man har funnet. Mekanismen som kopierer de beste individene, kalles *Elitism*.

Kombinering Individet kombineres med et annet individ, som også ble valgt ut, og resultatet av denne kombinasjonen legges til P_{n+1}. Denne mekanismen kalles tradisjonelt kryssing *(crossover)* [4], men jeg bruker det litt mer generelle begrepet "kombinasjon" i denne oppgaven. Begrepet kombinasjon brukes fordi det ikke forgår en kryssing på tradisjonelt vis, men heller en videre rekombinering av to individer. Det er nærliggende å bruke begreper som barn og foreldre når man snakker om en slik rekombinasjon.

Det finnes systemer som lager to resultater i hver kombinasjon, dette blir ikke gjort i denne oppgaven, fordi ERX (se 1.3.4) ikke på en enkel måte lar seg adaptere til å lage to resultater.

1.2.3.3 Mutasjon En del av de mulige løsningene kan forandres noe. Dette kalles *Mutation*. Nøyaktig hvor dette utføres varierer en del mellom ulike programmer. Det kan stort sett gjøres på tre ulike trinn: før, under eller etter kombinasjonen. Alle de tre variantene har noen fordeler og ulemper.

Før Mutasjonen skjer i trinnet før individer velges ut for overføring til neste generasjon, men etter evaluering. Siden evalueringen allerede er gjort, og individet blir forandret etter dette er det ikke lenger samsvar mellom individet og fitnessverdien.

Under Mutasjonen skjer i samme trinn som utvelging. Først vil individer velges til en overføring. Så kan enten individet muteres før eller etter overgangen. Ved kopiering er denne forskjellen irrelevant, men for kombinering er dette forskjell på om foreldrene skal muteres før kombinasjon, eller om resultatet av en kombinasjon skal muteres før det plasseres inn i den nye generasjonen.

Etter Mutasjonen skjer etter at alle individer er overført til neste generasjon. Dette har omtrent samme effekt som å mutere under overføringen, men det kan komme til å mutere individer som ikke burde muteres, f.eks. individer som har blitt overført til neste generasjon ved hjelp av elitisme.

1.2.4 Avslutning

Skal evolusjonen fortsette slettes alle individene i P_n og individene i P_{n+1} brukes videre. Systemet sies da å ha gått over til neste generasjon. I mange tilfeller setter man noen stoppkritererier for evolusjonen. Enten ved at evolusjonen har brukt en gitt mengde ressurser (CPU-tid, antall generasjoner, antall timer) eller ved at systemet har funnet en løsning som oppfyller kravene.

1.2.5 Oppsummering

Evolusjon fungerer ved å lage en populasjon med mulige løsninger. Disse løsningene evalueres. Etter evalueringen brukes fitness-verdiene til å bygge opp en neste generasjon. Den gamle generasjonen slettes, og man evaluerer den nye. Slik fortsetter man til man enten har funnet en tilstrekkelig god løsning, eller man har funnet ut at man har holdt på lenge nok.

1.3 Travelling Salesman Problem

I denne oppgaven bruker jeg Travelling Salesman Problem (heretter TSP) som eksempel-problem. Jeg har valgt å bruke dette problemet fordi det er anerkjent som et problem med høy kompleksitet. Problemet i seg selv er

NP-hard, og å finne ut om det finnes en runde som er kortere enn X er NP-complete. [10] En forklaring av NP *(non-deterministic polynomial-time)* og kompleksitetsklasser faller utenfor denne oppgaven. Hovedpoenget er at TSP er et vanskelig og komplekst problem å løse.

TSP går i all sin enkelhet ut på å finne den korteste veien som går gjennom n punkter og tilbake til utgangspunktet. For denne oppgaven sin del skal vi finne veien gjennom n punkter på et plan, og avstanden mellom to punkter er den kartesiske avstanden mellom dem. Hvilken vei man går rundt ruta har ingenting å si i denne oppgaven.

Uttrykt mer matematisk går TSP ut på å finne den *Hamilton Cycle* i en *Symmetric Weighted Graph* som får minst mulig sum.

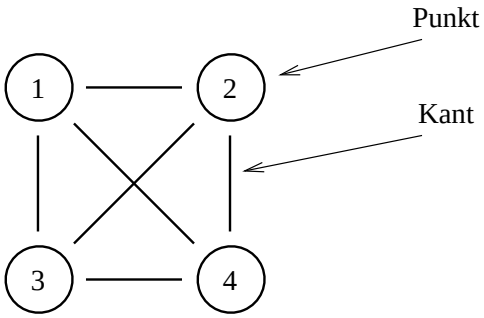
Problemet dukker ofte opp som et subproblem i mange andre problemer. Merrill Flood brukte dette først rundt 1940 for å sette opp kjøreplaner for skolebusser i USA[6]. Optimalisering av andre logistikk-problemer er nærliggende. Et klassisk eksempel er drilling av hull i kretskort hvor hullene blir punktene, og man vil finne den raskeste rekkefølge å borre dem. Det brukes videre i genforskning, operasjonsanalyse i økonomien, astrofysikk og mer. Stort sett alle steder hvor noe kan reduseres til et logistikk-problem.

Man kan godt gjøre dette problemet mer komplekst ved å utvide til flere dimensjoner, eller la vinkler også telle inn, sånn at man leter etter en rute hvor man også skal svinge minst mulig. Man kan også gi ulik vekt til avstanden avhengig av hvilken vei man går mellom to punkter.

Jeg har valgt å holde det enkelt for ikke å miste fokus på det evolusjonære systemet. Dette vil si at det ikke foregår noe lokalt søk ved hvert individ med ikke-evolusjonære algoritmer for TSP optimalisering (som 2opt, Lin-Kernighan eller lignende).

1.3.1 Representasjon.

For å lette representasjonen er hvert punkt gitt et unikt nummer. Hvert individ har så en vektor med alle disse numrene, og rekkefølgen er det eneste



Figur 1: Navn på ulike konsepter i grafer.

utslagsgivende for hvilken fitness-verdi et individ får. Merk at for n punkter vil det finnes $2 * n$ løsninger som er identiske. Løsningen kan forskyves uten at dette påvirker resultatet.

- (1,2,3,4) har samme rute som (2,3,4,1), (3,4,1,2) og (4,1,2,3)

Rekkefølgen kan også inverteres uten at det har noen innvirkning, siden vi ser på ruta som en ikke-direksjonell graf. Det vil si at retningen er uvesentlig. $1 \rightarrow 2$ har en retning fra 1 til 2, hvor $1 \leftrightarrow 2$ ikke har noen retning. For TSP vil dette si at det er like enkelt å flytte seg fra 1 til 2 som fra 2 til 1. Eksempler på hvor dette ikke er tilfellet vil være hvis man måler tiden man bruker for å forflytte seg opp og ned på et fjell, man vil typisk bruke mer tid opp enn ned.

- (1,2,3,4) har samme rute som (4,3,2,1)
- (4,3,2,1) får også samme rute som (3,2,1,4), (2,1,4,3) og (1,4,3,2)

For denne enkle ruten finnes det med andre ord åtte mulige representasjoner.

I mitt system er det ikke gjort noe forsøk på å redusere disse ulike representasjonsvariantene. Derimot behandles alle likt når det kommer til kombinasjonssteget.

Disse ulike variantene er like i den forstand at de representerer de samme kantene. Altså, (1,2,3,4) har kantene [1,2], [2,3], [3,4] og [4,1]. Siden denne

grafen er symetrisk er kanten [4,1] og kanten [1,4] identiske. Vi skal tilbake til utgangspunktet, derfor er kanten [1,4] også med. En annen mulig representasjon kunne benyttet kanter som den mest elementære bygge-enheten, men det er vanskeligere å holde kompleksiteten under kontroll. For n punkter finnes det $\sum_{k=1}^{n-1} k$ mulige kanter. Med 256 punkter vil vi ha over 32000 mulige kanter. Å bruke så mange kanter som representasjon vil kreve mer plass og prosessortid og det vil være kostbart å sjekke integriteten, altså om vi har en kontinuerlig tur gjennom alle punktene, samt at hvert punkt kun er representert en gang. Det er ikke med dette sagt at kantene ikke skal brukes, men at de ikke er veldig hensiktsmessige å bruke som gener (intern representasjon). Å bruke kantene som en beskrankning er derimot veldig greit.

Det er viktig at integriteten holder. Dvs. ingen mutasjon eller kombinasjon skal ha muligheten til å lage individer med flere eller færre enn n punkter, og alle punktene skal kun forekomme én gang. Dette avgrenser søkerommet betraktelig, og det går mye fortere å finne fram til en god løsning enn hvis vi også skulle vurdere ugyldige løsninger.

1.3.2 Evaluering

For å kunne gi hver løsning en score må individet evalueres. Dette gjøres enkelt ved å summere lengden av hver kant i individet. Dette gir et tall som beskriver hvor lang den aktuelle ruten er. Siden vi er interessert i å minimere denne avstanden, returneres avstanden som negativ. Alle individer vil med andre ord ha en negativ fitness. En kort avstand gir da den høyeste fitnessen.

1.3.3 Mutasjon

Mutasjon av TSP foregår i dette systemet på følgende måter.

1. Bytte om to punkter. Dette er enkelt og raskt å implementere. Introduserer som oftest fire nye kanter.
   Eksempel: (1,2,3,4,5,6) kan bli til (1,5,3,4,2,6) - bytter 2 og 5.
   Vi har her fjernet kantene [1,2],[2,3],[4,5],[5,6] og introdusert [1,5],[3,5],[2,4],[2,6]

2. Reversere en sekvens. Også rask og enkel å implementere. Introduserer to nye kanter.
   Eksempel: (1,2,3,4,5,6) kan bli til (1,4,3,2,5,6) - reverserer sekvensen fra 2 tom 4.
   Fjerner kantene [1,2] og [4,5] og introduserer [1,4],[2,5].

3. Flytte et punkt. Forholdsvis enkel å implementere, men tar noe lenger tid fordi deler av sekvensen må fysisk flyttes. Introduserer tre nye kanter.
   Eksempel: (1,2,3,4,5,6) kan bli til (1,3,4,5,2,6) - flytter 2 til posisjonen mellom 5 og 6.
   Fjerner kantene [1,2],[2,3],[5,6] og introduserer [1,3],[2,5],[2,6].

Som et spesialtilfelle av 1 eller 2, bytter man om to nabo-punkter - eller inverterer en sekvens av lengden to. Her introduseres to nye kanter.

I denne oppgaven brukes alle tre måtene med lik sannsynlighet. Det kunne vært interessant å se ulike mutasjonsrater for de ulike mekanismene, men dette er mer TSP spesifikt enn det er interessant for parallelle mekanismer.

1.3.4 Kombinasjon

Det finnes et utall mulige metoder for å kombinere to TSP-individer. Her brukes en metode som kalles Edge Recombination Crossover (ERX)[2]. Denne metoden går ut på å se hvilke kanter som uttrykkes av individers punkt-vektorer, for deretter å finne en rute som baserer seg på begge rutene.

Dette kan enklest forklares ved å tenke oss at vi legger begge rutene på hverandre. De fleste punktene vil da ha flere enn to kanter, enter tre eller fire. Ved å starte på en punkt og jobbe seg tilfeldig videre, kommer man til slutt gjennom alle punktene. og stort sett bare med kanter som fantes i de opprinnelige individene. Se 1.3.5.

Videre kan dette effektiviseres ved alltid å velge den kanten som fører til en punkt med lavest kardinalitet, altså det punktet som har færrest kanter. Dermed blir det vanligvis ikke nødvendig å innføre nye kanter i det hele tatt.

Grunnen til at dette kan skje er at man prøver å bli kvitt punkter med lav kardinalitet, siden disse er vanskeligst å bli kvitt. I det store og hele forsøker man seg på en *best-fit-first*-strategi. I det tilfellet består resultatet kun av kanter som også eksisterte i de to opprinnelige individene.

ERX er ikke en lettvektsalgoritme som mange av de andre mulige kombinasjonsalgortimene. Den er litt komplisert, og krever en del prosessortid. I mitt program er det dette steget som tar mest tid. Til gjengjeld innfører den ikke nye gener i systemet. Dette er et ganske viktig punkt med hensyn til evolusjon generelt. Resultatet av en kombinasjon bør ligne best mulig på de individene den konstrueres på grunnlag av.

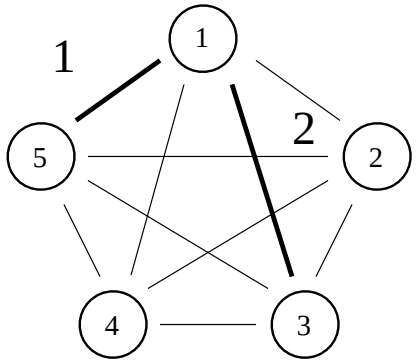
Det er nærliggende å bruke foreldre og barn som illustrasjon på henholdsvis de opprinnelige individene og resultat-individet av en kombinasjon.

1.3.5 Eksempel på evolusjon av TSP

For å illustrere gangen i en evolusjon skal vi forsøke å finne fram til den raskeste veien gjennom alle punktene i en regulær femkant. Vi betegner de ulike punktene med tallene 1 til 5. Et individ betegnes som (1,2,3,4,5), som også er den beste løsningen for dette enkle problemet. Det finnes mange løsninger som alle er isomorfe til denne løsningen. Alle har det til felles at de uttrykker de samme kantene (se 1.3.1).

Målet for hvor bra et individ gjør det, er lengden av ruta. Videre bruker vi det negative av denne lengden, for at den korteste ruta skal få et høyest, det vil i dette tilfellet si minst negativt, tall som fitness. Det finnes mange måter å skalere fitness på, men ingen brukes i denne oppgaven. En skalering er unødvendig fordi jeg kun bruker tournament-selection (se 1.2.3.1), som ikke tar hensyn til hvor stor forskjell det er mellom to fitness-verdier, men kun til hvilken som er høyest.

For å forenkle regningen brukes 1 som den korteste mulige avstanden, og 2 som en av de som krysser. Dette illustreres kort i figur 2. Vi får da et mulig spenn av fitness fra 5 til 10. En regulær femkant vil ha en omkrets på 5 og i et pentagram vil hele ruten utgjøre 10 enheter.



Figur 2: Illustrasjon av forenklet TSP-problem

1. Genererer et antall (5) tilfeldige løsninger på problemet.

2,3,4,1,5	4,5,3,1,2	3,5,2,1,4	3,5,2,1,4	2,4,1,4,3

2. Evaluerer disse og gir hver løsning en verdi.

2,3,4,1,5 = -7	4,5,3,1,2 = -8	3,2,5,1,4 = -7	3,5,2,1,4 = -8	2,4,1,3,5 = -10

3. Kopierer den beste løsningen til neste tidsskritt.

3,2,5,1,4 = -7

4. Vi kombinerer så noen av disse løsningene for å lage nye.

3,2,5,1,4	2,3,4,1,5	=>2,3,4,1,5
4,5,3,1,2	2,4,1,3,5	=>1,3,5,2,4
4,5,3,1,2	3,2,5,1,4	=>5,2,4,1,3
2,3,4,1,5	3,5,2,1,4	=>3,4,1,5,2

5. Så forandrer vi et lite utvalg av individene noe. Merk at vi senere vil ha mutasjonen på et annet steg, men for enkelhets-skyld har vi den her i dette tilfellet.

1,3,5,2,4	=>1,2,5,3,4
3,4,1,5,2	=>5,4,1,3,2

6. Nå sitter vi igjen med

3,2,5,1,4	2,3,4,1,5	1,2,5,3,4	5,2,4,1,3	5,4,1,3,2

7. Evaluerer disse og gir hver løsning en verdi

3,2,5,1,4 = -7	2,3,4,1,5 = -7	1,2,5,3,4 = -7	5,2,4,1,3 = -10	5,4,1,3,2 = -7

Gjennomsnittet av fitnessverdier er her bedre enn ved startposisjonen. At vi får et bedre resultat for hver generasjon følger ikke automatisk, men at gjennomsnittsfitnessen i snitt blir bedre for hver generasjon er et premiss for at evolusjon skal fungere.

8. Kombinerer, kopierer og forandrer vi videre, vil vi etter hvert ende opp med den optimale løsningen (1,2,3,4,5), som får verdien 5. Dette kan skje ved en kombinasjon av (1,2,5,3,4) og (5,4,1,3,2) og en god porsjon flaks, kanten [1,5] er nemlig ikke uttrykt av disse individene. Den kan derimot oppstå ved kombinasjon eller mutasjon.

For TSP kan man se på de uttrykte kantene som byggeblokker. Hvis man finner en god kant, bør denne bevares, det vil si komme videre til neste generasjon. Ved å sette sammen korte kanter til en koherent rute, finner man fram til gode løsninger. Dette er i tråd med building block hypothesis i den forstand at man bruker byggeklosser (korte kanter) til å lage en større konstruksjon (en koherent rute).

Evolusjonære systemer egner seg godt til å løse problemer uten noe klart definert mål, eller man kjenner målet men ikke vet hvordan man skal finne fram til det. I tillegg trengs det en metode for å se hvor god en mulig løsning er, noe som er forholdsvis enkelt i TSP.

1.4 Distribuering

En evolusjon kan deles opp på ulike måter for å la flere maskiner jobbe med den samme evolusjonen. Hver klient vil jobbe med sin del, og hensikten er å få utført evolusjonen raskere. Mange oppgaver kan distribueres for å kunne utføre mer på kortere tid. Det blir en måte å ligge foran Moore's Law [3] ved å bruke mange prosessorer. Hensikten er med andre ord effektivitet! Vi vil

[3]Prosessor-ytelsen dobles hver 12-18 måned. Litt avhengig av hvem du spør. Først postulert av Gordon Moore i 1964. Hver 12 måned holdt til rundt 1970 da doblingen gikk noe saktere og heller skjedde i løpet av 18 måneder. Gordon Moore var med å starte Intel i 1968.

utføre mer arbeid på samme tid, eller samme arbeid på kortere tid. Oppgaven vil stort sett ta for seg samme arbeid på kortere tid.

Stort sett alle distribusjonstrategiene har en eller annet form for master/slave forhold. En enhet har en form for kontroll over resten av systemet. Jeg vil i denne oppgaven kalle master for server og slave for klient. Hub/nav og spoke/eike kunne kanskje også fungert, men server/klient bør holde[4].

Tradisjonelt finnes det fire ulike distribueringsmodeller Global Master Slave, Island, Cellular og Hierarchical parallell Gentiske Algoritmer (GA) [7, 8, 9].

1.4.1 Ulike distribusjonsmodeller

En oppdeling av arbeid og kontroll kan gjøres på flere måter. De mest opplagte er omtalt under:

Global Master Slave Evalueringen foregår på klientene og kombinasjonen på en server. Denne modellen passer godt hvis det krever mye å evaluere et enkelt individ. Et eksempel kan være en omfattende fysisk modell som skal simuleres, hvor man forsøker å finne noen grunnleggende konstanter for dette systemet. Serveren kan da dele ut individer til klientene og få en fitness tilbake. Den forholdsvis enkle oppgaven med å kombinere og mutere kan utføres på serveren. Dette fungerer som en ikke-distribuert evolusjon, bare at mange individer kan evalueres parallelt. Sagt på en annen måte fungerer dette kun som en hardwareakselerasjon.

Cellular Populasjonen deles opp i små grupper som fordeles ut over et område. Seleksjon og kombinasjon kan kun finne sted ved nærliggende individer. Et eksempel vil være å legge alle subpopulasjoner ut over en matrise og la hver subpopulasjon kommuniserer nord, øst, vest og syd. Kommunikasjonen forgår ofte her ved at flere subpopulasjoner overlapper hverandre med tanke på kombinering, slik at det ikke finnes noen eksplisitt eksportering. Dette betegnes ofte som *fine-grained* GA. Det

[4]Det norske ordet for server er tjener, men jeg har valgt å bruke det engelske.

er også muligheter for å overlappe disse mindre bitene av populasjonen for å få litt interaksjoner mellom individer.

Island Hele populasjonen deles opp i mindre biter, og hver klient er sin egen evolusjon. Serveren har jobben med å koble ulike klienter til hverandre. Hver klient kan da sende og motta individer fra andre klienter. Denne teknikken kan utføres både synkront (alle klientene befinner seg i samme tidsskritt) eller asynkront (hver klient jobber så fort den kan). Dette kan også betegnes som en *coarse-grained* eller distribuert GA.

Hierarchical Denne modellen kombinerer Island med Cellular eller Global Master Slave.

Distinksjonen mellom *fine-grained* og *coarse-grained* er stort sett et mål på ratioen mellom komputasjon og kommunikasjon. Jo mer en modell kommuniserer, jo mer *fine-grained* kan den sies å være.

En subpopulasjon er en separat bit av den totale populasjonen. Dette kalles ofte *demes* (etter [11]) i litteratur om distribuert eller parallell GA. Jeg har valgt å holde meg til begrepet subpopulasjon.

I denne oppgaven brukes en synkron Island-modell.

1.4.2 Om Island-modellen

Subpopulasjon er en av modellene som gjør det mulig å utnytte det faktum at det faktisk **er** en distribuert evolusjon, og ikke separate populasjoner. Hver subpopulasjon som kjører på en separat maskin vil ofte kalles en node, da subpopulasjoner med migrasjonskanaler kan ses på som en graf.

Oppdelingen gjør det mulig å utnytte at det eksisterer en viss treghet i systemet. Hvis man tidlig i evolusjonen finner et uforholdsmessig fit individ, trenger ikke dette dominere den samlede populasjonen fordi mange subpopulasjoner ikke nødvendigvis har fått vite om dette. På den annen side har hver subpopulasjon muligheten til å fortsette i sitt eget spor. Dette kan forbedre utforske/utnytte modellen (se 1.1.1). Hver subpopulasjon har muligheten til

å følge et ganske snevert spor, hvor systemet totalt kan følge mange ulike retninger.

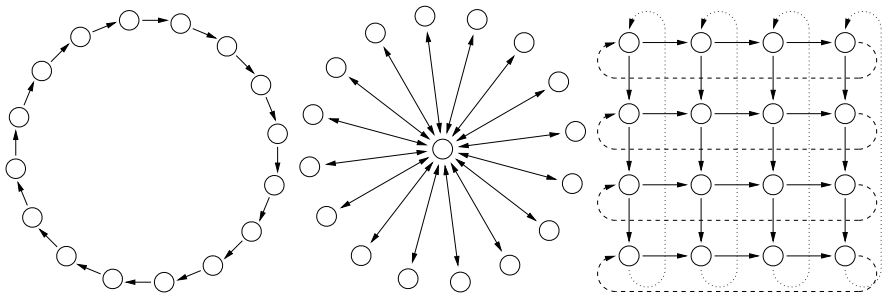
Subpopulasjoner i en evolusjon fører med seg noen flere muligheter. Først og fremst gjelder det her koblingene mellom de ulike subpopulasjonene, og hvordan flyten i disse går.

Eksportrate Hvor ofte flyttes/kopieres individer mellom subpopulasjonene?
Denne variabelen angir hvor ofte vi skal kopiere individer fra en subpopulasjon til en annen. Det kunne også vært mulig å flytte individet, slik at den opprinnelige noden mistet individet. Ved å senke eksportraten til null får man en parallell modell, mens den ved en veldig høy rate tilnærmer seg en enkelt stor populasjon.

Eksportseleksjon Hvordan velges individer som skal eksporteres?
Dette fungerer omtrent som seleksjon ved kombinasjon. Vi kan f.eks. velge dem ved hjelp av Tournament Selection. Den biologiske ekvivalenten vil si noe om hvilke dyr som forsøker å bytte flokk, eller som bytter område.

Importrate Hvor fort skal vi ta inn individer i subpopulasjonen? Hvor ofte og hvor mange?
Man kan sette en begrensning så ikke importering av individer dominerer. Hvis man har en node med mange inn-vektorer, kan det være fornuftig å begrense hvor mange individer utenfra som kan innføres ved hver generasjon. Individene vil da i mellomtiden ligge i en kø og vente. Det kan muligens være en fordel å ha noen "smeltedigler" i systemet, altså noen subpopulasjoner som tar i mot store mengder individer på hvert generasjonstrinn.

Migrasjonstruktur Hvordan skal nodene kobles sammen?
Hver node har mulighet til å koble seg til et antall andre noder, og vi har med dette muligheten til å lage en *directed graph*. Ut av dette kan vi lage mange ulike mønstre. Hvor fort genene spres i systemet er noe avhengig av dette, særlig hvor mange noder som i gjennomsnitt skiller



Figur 3: Ulike mulige strukturer
Til venstre et eksempel på RING, i midten en HUB og til høyre en GRID. Pilene illustrere eksportretningen. RINGen har en enkel lenking, HUB har dobbel (som den for så vidt må) og GRID har enkel lenking.

to vilkårlige subpopulasjoner.

1.4.3 Strukturer

Denne oppgaven vil ta for seg et lite utvalg mulige migrasjonstrukturer. Disse har fått betegnelsen ring, hub og grid. Lenkene mellom hver node kan også variere mellom å gå én vei eller begge.

Ring Hvor node er forbundet med to andre. Litt mer spesifikt er node n koblet til n-1 og n+1, bortsett fra første og siste noden, som er koblet til hverandre. Vi har da fått alle nodene koblet sammen i en lang kjede. Med enkel lenking vil n eksportere til n+1 og importere fra n-1. Dobbel lenking vil føre til en eksport og import begge veier. I denne strukturen tar det forholdsvis lang tid før informasjon fra en node treffer noen andre. En node er maksimalt $n - 2$ lenker unna en annen node i enkel lenking og $\frac{n}{2}$ for dobbel lenking.

Et annet aspekt som kan brukes ved en ring er å koble n til n+1, n+2...n+k. Ved å gjøre dette øker man kommunikasjonen, eller i verste fall minker destruktiv kommunikasjon fra en node, i det tilfellet en node fungerer som en data-sink.

Hub Alle nodene bortsett fra én er koblet med en dobbel lenke til en enkelt node. Denne ene noden fungerer som eneste kommuniksjonsledd, men dette leddet gir en svært rask kommunikasjon. Hver node er bare 2 lenker unna en annen node, så innen få generasjoner har alle nodene tilgang til informasjonen. Denne modellen alene kan fort få problemer med skalering siden senternoden har en trafikk like stor som alle de andre.

Grid Hver node er forbundet med fire andre. Kan også kalles en matrise. Gitt at man tegner opp alle nodene i en matrise, vil hver node ha en forbindelse nord, sør, øst og vest. Ytterkantene vil da ha en forbindelse til den motsatte ytterkanten. Ved enkel lenking vil man ha eksportering mot øst og syd - og følgelig importere individer fra nord og vest. Dobbel lenking eksporterer og importerer alle veier. Hvor langt unna en node kan befinne seg i denne strukturen har ganske mange variable, men det befinner seg rundt $\sqrt{n}$ og $\frac{\sqrt{n}}{2}$ lenker unna for henholdsvis enkel og dobbel lenking.

En grid kan også sies å være et mer generelt tilfelle av en ring. En ring kan være en $1xn$ grid. Her vil nord/syd kommunikasjonen være med seg selv og lenkene gir stort sett ingen mening.

Det finnes selvsagt også andre mulige strukturer.

Tilfeldig Hver node lenkes opp mot et tilfeldig, men begrenset, antall andre noder. Vi kan her enkelt variere hvor stor grad av binding vi vil ha, altså hvor mange noder en node i snitt skal koble seg opp til.

Kombinasjonstrukturer Vi kan godt kombinere ulike strukturer for å benytte det beste fra hver bit. F.eks. kan en tilfeldig struktur blandes med en ring slik at vi er garantert å ha en kobling mellom alle nodene, eller en ring sammen med en hub for å gi en raskere kommunikasjon gjennom senternoden.

1.5 Hva er et godt resultat?

Både fart og resultat kan optimaliseres. Jeg har vært mest opptatt av å optimalisere for fart, altså å kunne løse en oppgave raskere enn det som er mulig på en enkelt maskin. Dette kan være svært viktig for kompliserte problemer kan ta flere dager å løse. Å korte ned tiden for løsningen av disse problemene er en åpenbar fordel.

Vi skal se hvordan en tenkt stor maskin gjør det i forhold til et sett med noder. Denne tenkte maskinen skal være like "kraftig" som alle nodene til sammen. Man gir denne tenkte maskinen prosessortiden t og kjører en evolusjon med en populasjon på i individer. Ved distribuering forsøker man å ha n noder, som hver har en subpopulasjon på $\frac{i}{n}$ individer. Hver node skal da bruke prosessortiden $\frac{t}{n}$. Hver av modellene skal bruke like mye samlet prosessortid, men den parallelle modellen bruker mye mindre reell tid.

Prosessortiden som brukes ved kommunikasjon mellom ulike noder forutsettes ikke å ha noen innvirkning på målingene. Det betyr at en overhead grunnet distribuering ikke tas med, fordi nettverkskommunikasjonen knappest bruker noe CPU-tid. Vi antar at antall individer er proporsjonalt med prosessortiden, gitt at andre variable holdes konstant. Dette gir intuitivt mening, fordi alle de mest tidkrevende prosessene, først og fremst kombinasjon og evaluering, utføres et antall ganger lineært skalert i forhold til populasjonstørrelsen.

Dette gjøres ved å sammenligne fitnessverdier for hver generasjon både for et snitt av de distribuerte nodene, og for en superpopulasjon som har like mange individer som alle subpopulasjonene totalt. Sammenligningen kan gi tre ulike resultater:

Subskalart Distribuert gjør det dårligere enn en stor populasjon. Resultatet fra det distribuerte systemet er vesentlig dårligere enn fra et samlet. Oppgaven viser seg å være noe seriell av natur.

Skalart Distribuert gjør det like bra som en stor populasjon. Det er altså fult mulig å dele opp oppgaven og utføre den i parallell.

Superskalart Distribuert gjør det bedre enn en stor populasjon. Omstendigheter gjør at man utnytter tregheten og informasjonsdeling til å gjøre en bedre jobb enn hva som er mulig for et samlet system.

Et annet spørsmål er hvordan det går med flere store populasjoner, som går like fort, men gir et bedre resultat. Med andre ord å ikke redusere den absolutte tiden det tar å finne et resultat. Hvor man benytter den parallelle modellen til å la hver subpopulasjon ha i kontra $\frac{i}{n}$ individer. I dette tilfellet vil oppgaven ta nt prosessortid, men den samme reelle tiden.

2 Relatert litteratur

Det er gjort en del arbeid med parallelle evolusjonære systemer, men mange av disse har konsentrert seg mer om kjøring på en parallell maskin enn i et distribuert system, altså - at det kjører på en kraftig MIMD (Multiple Instructions Multiple Data) maskin kontra at selve oppgaven utføres som et samarbeid mellom mange enkelt-maskiner. I min oppgave er det noe overlapp mellom hva som er simulert distribuert og faktisk distribuert, men forskjellene vil uansett være forholdsvis små.

2.1 Muhlenbein

Muhlenbein[13] ga ut en artikkel i 1991 kalt "Evolution in time and space - the parallel genetic algorithm" hvor han ser på hvordan en parallellisering påvirker utviklingen av TSP og et *deceptive problem*[5]. Som illustrerende forsøk på sin Parallel Genetic Algortithm (PGA) bruker han DeJongs F5[6] med fire subpopulasjoner a ti individer hvor det eksporteres et enkelt individ hver tiende generasjon. Han har valgt å bruke migrasjonstrukturen han

[5]Deceptive problem: Vanligvis et problem der hvor den beste løsningen ligger i motsatt retning av en gradering for bitverdier kan dette være at et individ får en fitnessverdi lik antall bit som er satt til true, men hvor den beste løsningen har alle bit satt til false.

[6]Også kalt Shekel's foxholes

betegner som "ladder", som i denne oppgaven tilsvarer en enkelt-lenket ring-konfigurasjon. Argumentet for denne strukturen er at den i større grad gir subpopulasjonene mulighet til å spesialisere seg ved at det tar $O(n^2)$ steg før informasjon kan ha blitt spredd ut til alle subpopulasjonene, kontra $O(n)$ for strukturen han betegner som en $n * n$ torus, som i denne oppgaven har navnet enkelt-lenket grid.

Denne strukturen hjelper en seleksjonsprosess som kun velger ut potensielle kombinasjoner basert på avstand, altså kan et individ kun krysses med individer innenfor en viss nærhet. Her overlapper de ulike subpopulasjonene, slik at man får en overføring av informasjon fra en subpopulasjon til en annen. Grunnen til navnet torus er at en faktisk geometrisk figur konstruert etter grid-oppskriften vil se sånn ut - som et rør som går i sirkel. Bakgrunnen til at en $O(n^2)$ løsning er ønskelig fremfor en $O(n)$ har sammenheng med at de største kreftene i parallell evolusjon finner sted ved en migrasjon og de umiddelbare stegene etterpå. Dette har igjen sammenheng med at populasjonene helst ikke skal ha mye overlappende genmateriale, siden dette ikke vil gi noe utslag. Derfor er man interessert i å holde subpopulasjonene så langt ute av synkronisering som mulig, noe som gjøres ved "ladder"-konfigurasjonen.

Konklusjonen på artikkelen forteller oss at en parallellisering godt kan hjelpe utviklingen framover. Videre nevnes det at algoritmen enkel kan utvides med et lokalt søk, en ren hillclimbing, av de enkelt individer, sånn at det ikke er utelukkende evolusjon som driver resultatene fremover. Dette skrittet kombinerer evolusjonen med en form for læring. Et annet aspekt som tas opp, er å forandre populasjonen underveis i evolusjonen, eksempelvis å forandre størrelsen etter hvert.

Muhlenbein har senere gitt ut flere artikler[19, 20] hvor det vises at en parallellisering gir gode resultater. Disse artikkelene tar for seg omtrent det samme temaet som [13], men bruker noen andre problemer for å illustrere arkitekturen av PGA.

2.2 Baluja

En enda mer relevant artikkel er skrevet av Shumeet Baluja [14]. Også Baluja benytter seg av DeJongs funksjoner som en test av arkitekturen. Her argumenteres det med at distribuert evolusjon fungerer som "*punctuated equilibria*". I dette tilfellet vil det bety at hver subpopulasjon utvikler seg raskt og ganske stabilt, men treffer rimelig fort et platå. Individer fra subpopulasjoner som har nådd et høyt nivå, kan da ha stor fordel av å bli utsatt for et annet miljø slik at det kan fortsette å utvikle seg. Denne effekten skyldes ikke bare en endring i seleksjonstrykket, men også at det kan være andre gener som blir avgjørende for utviklingen. Miljøet som individet blir ført inn i kan alt ha utviklet et sett med gode gener som ikke overlapper de gode egenskapene til det importerte individet.

Det argumenteres her mot en direkte fast eksportering av individer til en annen subpopulasjon hver n'te generasjon, og anbefales heller å ha noen overlappende populasjoner for å redusere effekten av at individer som introduseres ikke er helt ulike populasjonen de kastes inn i. En viss overlapp vil gi mulighetene til at subpopulasjoner som er nær hverandre, har noe til felles, men at det fremdeles finnes ulikheter ved større distanser. Her brukes det 4096 subpopulasjoner og noen ulike overlappinger som gjerne omfatter rundt 8 subpopulasjoner. Derimot er det kun ti individer per subpopulasjon.

Konklusjonen kan enklest uttrykkes ved at samspillet mellom nesten isolerte populasjoner finner mange gode resultater. Arkitekturen som er brukt, finner også bedre løsninger enn en enkel parallell genetisk algoritme, som her hadde 40 subpopulasjoner. Noe av konklusjonen viser da til at "*punctuated equilibria*" som utviklingsprinsipp fungerer godt, og designet med svært mange noder for å la dette fenomenet forekomme ofte, viser seg å ha vært et godt valg.

2.3 Whitley, Rana & Heckendorn

Det er også gjort analyse av island-modellen i Darrell Whitley, Soraya Rana, Robert B. Heckendorn artikkel "The island model genetic algorithm: on sep-

arability, population size and convergence"[15]. Island modellen vil her si det å eksportere de top 5% av hver subpopulasjon hver n'te *evaluering i hver subpopulasjon* i en ring-konfigurasjon. Migrasjonen foregår i forhold til evalueringer og ikke generasjoner for å jevne ut forskjellene mellom modellene.

Det eksperimenteres både med en total populasjonsstørrelse på 5000 og 500, med og uten migrasjon og mutasjon. Denne totale populasjonen deles opp i ulike antall subpopulasjoner. Problemene som brukes som illustrasjon, er enten et deceptive problem og separable funksjoner. Resultatene viser stort sett at oppsett med mange subpopulasjoner ofte gir bedre resultater enn en enkelt samlet populasjon. Samtidig viser det seg at mindre populasjoner (500) har en stor nytte av mutasjon for å tilnærme seg en større populasjon.

Hovedargumentet hviler på at flere subpopulasjoner hjelper til med å holde et genetisk mangfold, hvor en samlet populasjon gjerne vil nå et platå det ikke finnes noen enkel vei utenom. Dette er konsistent med Balujas forsøk over.

2.4 Alba, Cotta & Troya

Med genetisk mangfold som basis ser Enrique Alba, Carlos Cotta og Jose M. Troya videre på hvilke fordeler en parallellisering kan gi, i to artikler[17, 16]. Her måles det genetiske mangfoldet ved å se på en gjennomsnittlig entropi for den totale populasjonen, og videre også seleksjonstrykket. Både *coarse* og *fine-grained* genetiske algoritmer (GA) blir testet, men varierende grad av kobling, ulike migrasjonstrategier, samt både synkrone og asynkrone modeller. Målet er å se hvilke kurver entropien følger i utviklingen for de ulike systemene. I utgangspunktet har en cellular (*fine-grained*) GA et mye mindre entropisk fall enn distribuerte algoritmer. Av ulike distribuerte algoritmer faller solid-state GA drastisk nedover i forhold til en generational GA.

Hovedpoenget i disse artiklene er at det genetiske mangfoldet er avhengig av forholdsvis løs kobling / isolasjon av subpopulasjonenene i en island-modell. En merkbar forandring i effektiviteten fant sted ved en eksportering hver $16 * totale\,populasjonen$ evaluering. Både mindre og mer eksportering førte

til mer CPU-bruk for å finne tilsvarende resultater. I akkurat dette forsøket ble det brukt en asynkron modell som ikke tar for seg antall generasjoner, men heller ser på hvor mange evalueringer som blir gjort. Å gjøre om antall evalueringer til antall generasjoner er for så vidt en heller enkel oppgave. En hyppigere eksportfrekvens enn det fører til en mangel av genetisk mangfold og følgelig dårlige utviklingsgrad, selv om dette tilnærmer seg hvordan en seriell eller samlet populasjon vil ha utført oppgaven. Det finnes mange indikasjoner på at en løst koblet island-modell kan bidra til å løse problemer hvor det ikke nødvendigvis hjelper å øke populasjonstørrelsen.

2.5 Andre & Koza

Et system implementert av David Andre og John R. Koza har fått konsistent super-linear skalering[18]. Denne artikkelen tar for seg genetisk programmering på en rekke problemer. De genetiske programmene ble begrenset til 2500 punkter i program-treet, altså 2500 instruksjoner, som gjør at hver evaluering kan ta så mye tid at det ikke er praktisk å gjennomføre kjøringen på en seriell maskin.

Et aspekt som betraktes, er her er byggingen av et hardwareoppsett som skal maksimere ytelse i forhold til pris. Resultatet er, ikke overraskende, at det lønner seg å bygge et cluster av billige maskiner - i dette tilfellet en prosessor tilsvarende en 486/33 - som i dag er svært utdatert, men prinsippene gjelder fremdeles.

Selve evolusjonen fungerer som separate islands med en eksporteringskø og en importeringskø som tar seg av en asynkron migrasjon av individer mellom tilsluttede noder. Alle nodene er koblet til fire andre i en gridløsning (torus). Derimot blir individer flyttet og ikke bare kopiert.

Som første illustrasjonsproblem benytter de seg av en boolsk *even-5-parity* funksjon, altså en funksjon som er sann hvis et like antall av de fem argumentene er sanne og returnerer usann ellers. Fitness for et individ baseres på hammingdistansen mellom resultatet fra et individ og det korrekte resultatet. Med en total populasjon på 32000 (500 individer på i hver av de

64 subpopulasjonene) individer viste det seg at en eksportrate på 5% gir et korrekt resultat etter minst evalueringer. Dette vil altså si 5% eksport i hver av de fire retningene hver generasjon.

Videre brukes samme program og konfigurasjon til vanskeligere oppgaver, deriblant problemer fra *cellular automata*, molekylær biologi og kretskortdesign. I disse tilfellene fant systemet løsninger som hadde like god, eller bedre, suksessrate enn tidligere løsninger skrevet av mennesker. De fikk med andre ord løsninger på menneskelig nivå. Dette har for så vidt mer med evolusjonære systemer generelt å gjøre enn med distribuering, men det viser igjen at en distribuering ikke bare er mulig, men til og med en god ide.

2.6 Oppsummering

Det har blitt brukt mange ulike modeller og mekanismer for å lage parallelle evolusjonære systemer. Forskjellene kan deles opp etter mange akser, kommunikasjonsrater, kommunikasjonsmodeller, synkronitet og om systemet har en faktisk distribuering. Det som imidlertid er felles for de fleste gode funn er at de forsøker å utnytte "*punctuated equilibria*"[14]. Balansegangen mellom for lite og for mye informasjonsutveksling løses på ulike måter, men målet er stort sett det samme: å få laget ulike subpopulasjoner som får en solid dytt i ryggen ved tilførsel av nytt og godt genmaterialet. Hvis subpopulasjonene klarer å holde seg forholdsvis ulike, klarer dette fenomenet å gi parallell GA en superskalar ytelse.

Hensikten med denne oppgaven er å forsøke å finne hvilke parametre som gir best ytelse med det moduldesignet som er valgt. Nettopp dette designet gjør at rammebetingelsene er noe forskjellig fra systemene beskrevet over. Fra tidligere arbeid vet vi at hver subpopulasjon bør få muligheten til å utvikle sitt egen "miljø" for å kunne utnytte distribueringenen til sitt fulle.

3 Programmet

Et tradisjonelt system vil gjerne benytte seg av noen få variable som *elitism*, *crossover* og *mutation probability*, gjerne kombinert med ulike seleksjoner og skaleringsmekanismer. Et system går gjerne gjennom disse etter tur for å lage neste populasjon. Eksempelvis velger det først et par individer til elitism, skalerer fitness for resten med en eller annen funksjon, og det bruker *roulette wheel selection* for å kombinere resten. Denne prosessen kan generaliseres ved bruk av moduler.

Modulene er implementert som subklasser av en abstrakt klasse med en 'execute' *member function*[7]. Videre lagres alle modulene for en populasjon i en vektor - og alle modulene i denne vektoren kalles etter tur ved overgang fra et tids-skritt til det neste. Vi kan med et slikt system lage moduler for *elitism*, *mutation* og en form for *combination*. Ved å legge slike moduler i den tidligere nevnte vektoren kan man enkelt bygge sammen vilkårlige overganger. Denne løsningen gjør det svært enkelt å bygge opp et ønsket system.

Modulsystemet gjør det enkelt å legge til to moduler som tar seg av *Import* og *Eksport* av *Individer* til andre populasjoner. Disse to modulene er da ansvarlige for å bevege individer mellom subpopulasjonen. Man kan plassere import og eksport av individer hvor man vil i overgangen fra et populasjonstrinn til den neste.

Merk at import i dette programmet erstatter individer i populasjonen. Dette gjøres ved at individer importeres i steget før kombinasjon og kopiering, følgelig blir det mindre plass til individer fra den vanlige subpopulasjonen. Et alternativ kan her være å midlertidig øke populasjonstørrelsen til å inneholde den vanlige størrelsen og de importerte individene, slik at importerte individer må kjempe i første trinn for å bli med videre i evolusjonen.

Etter at alle moduler er lagt til, trenger programmet kun gå gjennom alle modulene i rekkefølge. Etter modulene har gjort jobben sin, jobber vi videre på den nye populasjonen.

Kilde for programmet ligger på http://www.stud.ntnu.no/~torehalv/d6.src.bz2

[7] C++ har member functions. Java bruker navnet method for dette.

3.1 Optimalisering

Systemet er skrevet i C++ og er forsøkt å gjøre så effektivt som mulig. Noen av teknikkene som er brukt følger:

3.1.1 Dobbel populasjon

I programmet som brukes eksisterer det til en hver tid to fulle populasjoner. Den nåværende generasjonen og en delt forrige/neste. Dette bruker dobbelt så mye minne, men sparer svært mange *alloc[8]. Minneforbruket blir da følgelig dobbelt så stort som en enkelt populasjon.

En annen løsning er å kun bruke en populasjon, og ved nøye manipulering bruke individer fra den nåværende populasjonen som mål for nykonstruerte individer. Det kan først velges ut en kø av individer som skal benyttes som grunnlag til neste tidsskritt. Fra denne køen er det mulig å trekke ut individer som ikke skal brukes videre. Nye individer, altså de som skal eksistere ved neste tidsskritt, kan da lages på de som ikke lenger brukes. Ved å gjøre dette kan man klare å nesten halvere minnebruken. Siden vi maksimalt vil bruke alle individene, vil ett av disse bli "ledig" etter andre kombinasjon. I worst-case tilfellet med kun kombinering som overføringsmekanisme, og alle individene brukes som kilde eksakt to ganger, vil man få individer som ikke skal brukes igjen ved andre tidsskritt - hvor man har brukt ett av individene to ganger. Man trenger med andre ord til tider to ekstra individer som buffer, men i praksis skal ikke dette være nødvendig. Denne mekanismen krever en del jobbing både fra programmerer og prosessor for å fungere. Derfor lar teknikken seg vanskelig kombinere med et fleksibelt modulsystem, fordi modulene ikke vet om hverandre. Ved å innføre en felles kø og flere skritt gjennom ulike deler av hver modul kunne en enkel populasjon fungert, men det er altså ikke gjort i dette tilfellet.

Løsningen brukt i dette systemet er mer direkte effektivt, men kan bare holde på halvparten så mange individer. Jeg vet ikke om dette er noen stor ulempe.

[8] *alloc: malloc, calloc, realloc, free og lignende. Vanlige systemkall for å allokere minne. Systemkall er normalt litt trege.

For TSP-problemet brukt i denne oppgaven er ikke minneforbruket så stort at forskjellen mellom enkel og dobbelt populasjon hadde gjort noe forskjell.

3.1.2 Static combine-data

ERX bruker en Nx4 matrise for å holde orden på alle kantene. Denne tabellen slettes aldri. Det er da viktig å sikre at tabellen ikke inneholder noen verdier etter en ferdig kombinasjon. Barnet av to individer slettes heller aldri, men kopieres til et individ i neste generasjon. Minnebruken vil da være forholdsvis konstant under hele kjøringen, og man slipper overheadet av *alloc (se fotnote 8).

3.1.3 Combination

Nx4 matrisa i ERX (Se 1.3.4) er sitt eget objekt med en del optimaliseringer for tilstedeværelse av verdier samt verdier i seg selv. Det brukes lookup-tabeller for alle bitoperasjoner, som telling av antall bit og for å finne først satte bit. Ved å prekalkulere slike verdier spares mye tid på disse operatsjonene. Matrisen er implementert som en konstant-størrelse array uten noen form for sjekk, bortsett fra C-style makro asserts som enkelt kan fjernes i compile-time.

ERX dikterer også at man skal velge noden med lavest kardinalitet for hvert steg. Dette gjøres ikke den første gangen for å slippe å vurdere alle nodene. Siden man uansett skal gjennom alle nodene har det ingenting å si hvor man starter.

4 Enkel kjøretest

For å verifisere at systemet fungerer, kjører jeg en liten test. Her tar jeg for meg et enkelt TSP problem, som vi enkelt kan se den beste løsningen på, nemlig ved å sette alle nodene i en sirkel med radius 1. Her vil den beste

løsningen tilbakelegge en strekning tilnærmet -2π, avhengig av hvor mange noder som brukes.

4.1 Først en enkelt node

Sekvensen som brukes for utviklingen er som følger.

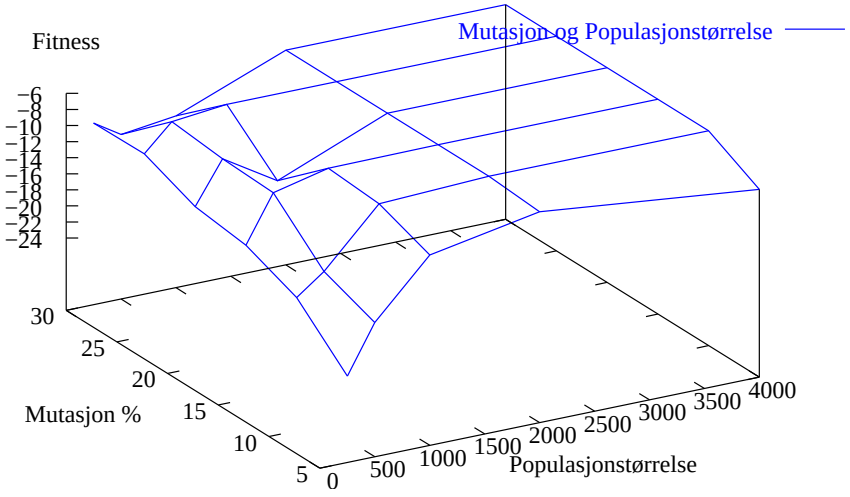
1. 1 elitism
2. n% mutate
3. 50% tournament (60%) copy
4. 50%(rest) tournament (60%) combine

Innledende brukes en 256-noders sirkel med radius 1. Dette er en ren hill-climbing[9]. På den annen side er den enkel å analysere i ettertid, og gitt at dette kun er for å få et grovt overslag over om systemet fungerer, vurderes den som bra nok. Den følger Building Block Hypothesis rimelig bra, og systemet vet ikke at oppgaven er triviell.

Vi skal her forsøke å finne noen gode parametre for resten av oppgaven, og varierer populasjonstørrelse og mutasjonsrate. Mutasjon trenger ikke nødvendigvis brukes. Faktisk vil mutasjon gjøre resultatene mer usikre, fordi det kan plukkes fram gener som ikke eksisterer i den opprinnelige populasjonen, men som ligger nært opptil. En større populasjon vil ha muligheten til omtrent det tilsvarende.

Etter noen innledende runder viste det seg at en populasjonstørrelse på 2000 individer og over 10% mutasjon gav konsistent gode resultater. Se figur 4. Dette stemmer bra med [1] som mener en optimal populasjonstørrelse for TSP ligger mellom $\log_2 n!$ og $2\log_2 n!$ med n=antall noder. For 256 noder bør man med andre ord ha 1600-3200 individer i populasjonen.

[9]Hillclimbing: En hver forandring som er positiv inkorporeres i et videre løsningsforslag. Man har da til slutt kommet til den optimale løsningen, hvis løsningsrommet er monotont stigende.

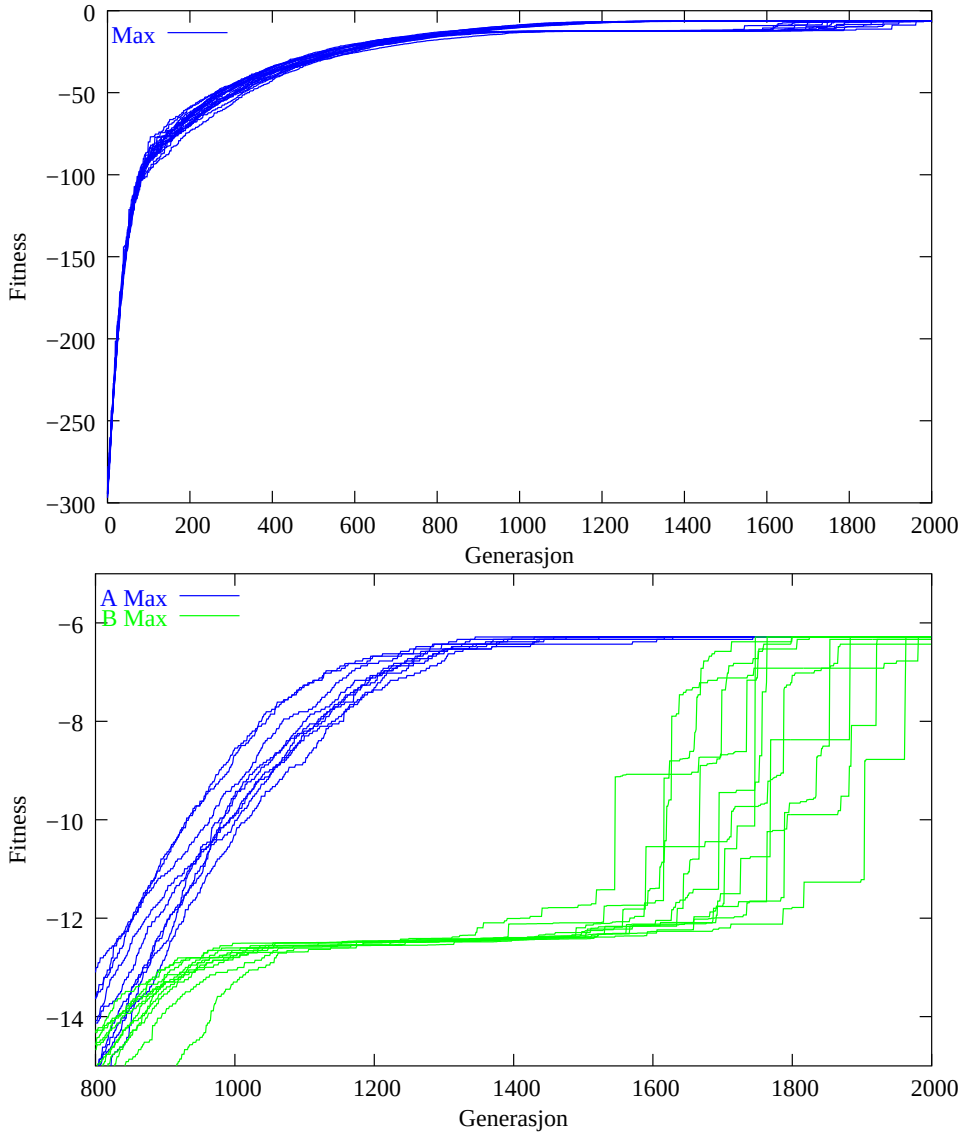


Figur 4: Mutasjonsrate mot Populasjonstørrelse (en enkelt kjøring)

Det viser seg at dette fungerer tålelig bra, som vist ved 20 kjøringer i figur 5. Vi ser den karakteristiske asymptotiske kurven hvor man nærmer seg en optimal løsning. Det mest spennende med figur 5 er området fra 800 til 2000 generasjoner, hvor omtrent halvparten av populasjonen stagnerer. I tilfellene hvor man ikke ledes inn på et sidespor, finner man en nær optimal løsning rundt den 1300 generasjon.

Løsningene som ikke fører helt fram, er slike som først forsøker å gå rundt sirkelen to ganger, før de presses til å prøve en enkel runde. Løsningen stopper opp et øyeblikk på -4π for så å jobbe seg videre til -2π. Hvis man går nøye inn noe tidligere i evolusjonen, kan man se den samme tendensen rundt -6π, men da i mye mindre grad. Dette problemet utmerker seg mest i så enkle problemer som en sirkel, men selve konseptet om en dobbel runde kan muligens generaliseres til mer ulineære problemer.

I det store og hele viser dette at systemet fungerer i den forstand at det klarer å finne gode løsninger. Det sier også noe om tidsskalaen. Rundt 2000 generasjoner er nok til å finne en svært god løsning. For dette enkle forsøket fant det den optimale.



Figur 5: Maks fitness / 20 kjøringer av en 256 noders sirkel : Delt opp etter sluttstrategi.

4.2 Enkle distribuerte forsøk

Jeg gjorde noen enkle forsøk for å se nærmere på noen strukturelle egenskaper. I forsøkene har jeg brukt tre noder med to strukturer. Den ene en sirkel, der hver node har en kobling til en annen node. Den andre er en linje, hvor den siste noden ikke kobler seg videre. Forsøket med linjen gir informasjon om hvorvidt en distribuering har innvirkning. Den siste noden i linjen bør gjøre det bedre enn de andre, fordi den får importert gode gener fra de to andre.

I begge forsøkene benyttes 700 individer. Det tilsvarer omtrent $\frac{1}{3}$ av populasjonen som jeg påviste var nødvendig i den innledende runden. Ved å bruke en fraksjon av en optimal populasjonstørrelse kan man sammenligne med en større populasjon for å se effektiviteten til det distribuerte systemet. I begge forsøkene brukes elitism som eksporteringsmekanisme.

4.2.1 Sirkel

Populasjon per node 700

Mutation 20%

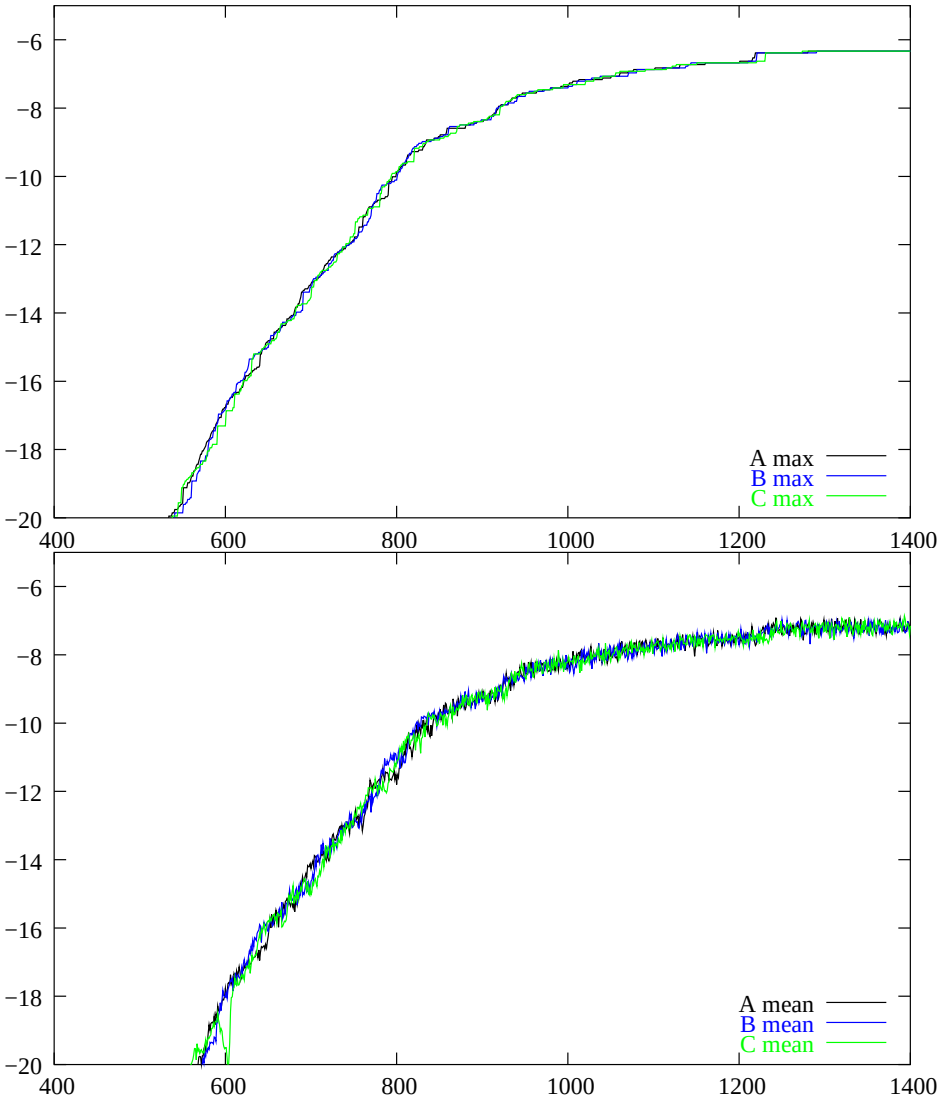
Struktur A→B→C→A

Eksport 10 Individer hver 10. generasjon

Import 1 Individ per generasjon

Maksimalverdiene følger hverandre svært tett. Det er ikke overraskende fordi de beste individene eksporteres ved hvert femte tidsskritt. Maksimalverdien starter ved rundt -300 og gjennomsnittet ved rundt -350. Grafene er kuttet litt for å illustrere se ulikhetene tydligere. En optimal løsning finnes rundt generasjon 1200.

Vi ser at middelverdien til noden C har et lite hakk rundt generasjon 600. Mest trolig lager de importerte individene fra node B i snitt dårlige løsninger



Figur 6: Ringstruktur - 3 noder a 700 individer.
Øverst kan man se en graf over de beste løsningene hver node har funnet.
Under er det plottet gjennomsnittet av løsningene.

når de kombineres med de lokale individene. Derimot fortsetter maksimalverdien oppover i samme tidsrom. Dette skyldes at noen kombinasjoner gir bedre resultater.

Testen gir oss en indikasjon på at systemet fungerer. Det er ingen tendenser til at systemet skal henge seg opp, og resultatet er en optimal løsning. Gjennomsnittet er muligens litt for nære den beste løsningen, utviklingen har et lavt standardavvik, og dette gjør at vi fort kan få en *premature convergence*. Denne tendensen kan være ødeleggende for senere forsøk. En mulighet for å motvirke dette er å gjøre mutasjonsraten omvendt proporsjonal med standardavviket. Vi vil da øke mutasjonen mot slutten av evolusjonen.

4.2.2 Linje

Populasjon per node 700

Mutation 20%

Struktur A→B→C

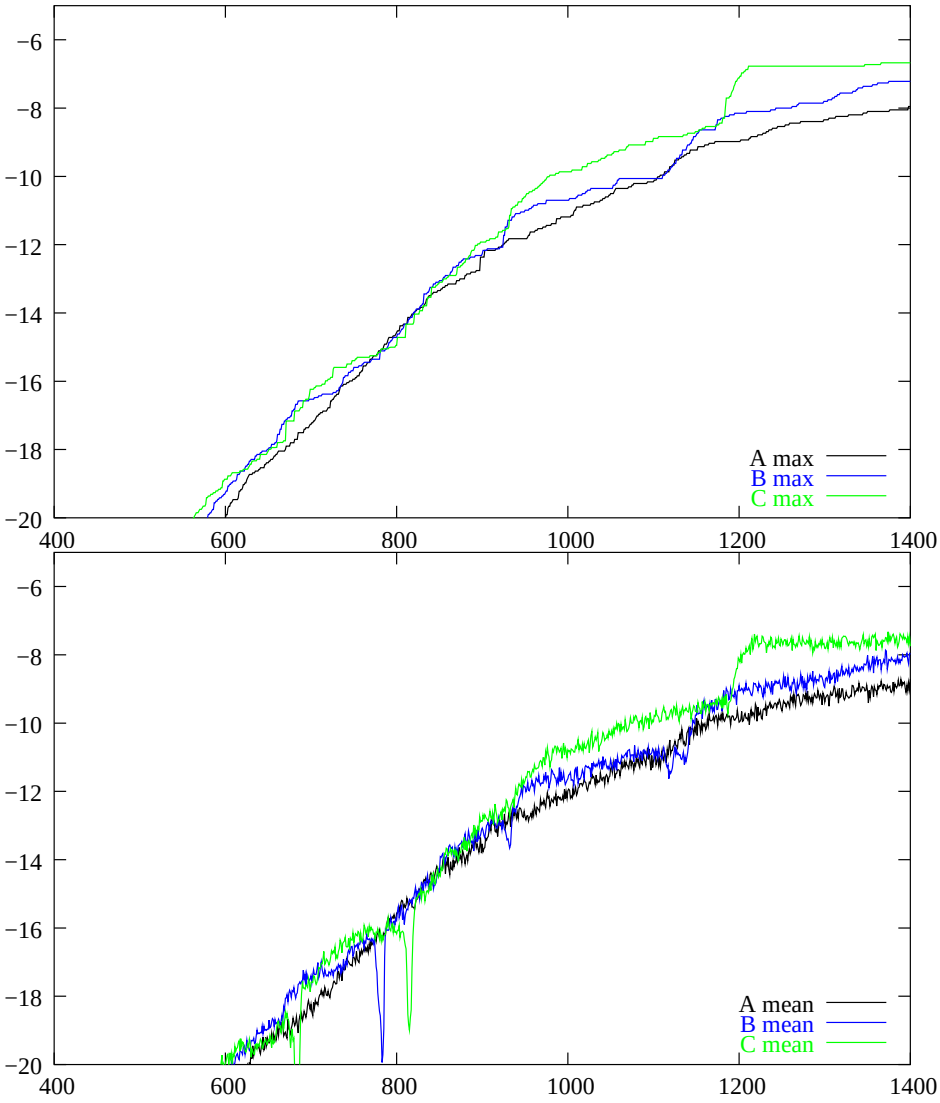
Eksport 10 Individer hver 10. generasjon

Import 1 Individ per generasjon

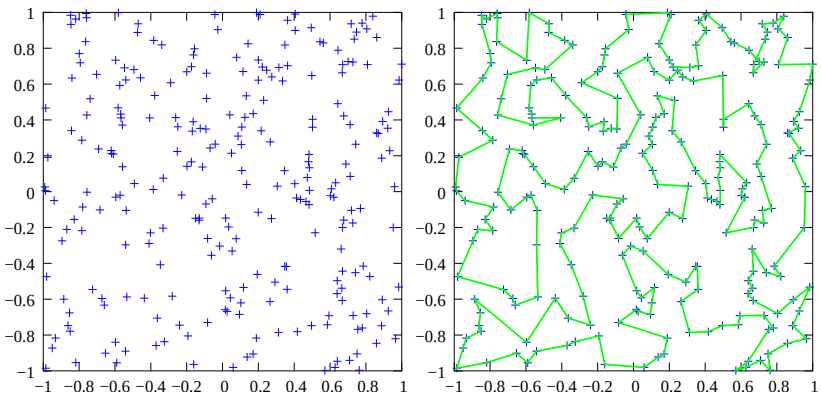
Gode egenskaper vil forplante seg til den siste noden.

Det eneste, og kanskje enkleste vi kan se av figur 7, er at en subpopulasjon stort sett har fordeler av å få tilsendt nye gode individer.

Det er spennende å se på de plutselige stupene i gjennomsnittsverdien rundt 800. generasjon. Det virker som A har eksportert noe som senker gjennomsnittet betraktelig (noe som i seg selv har en god løsning, men som ikke lar seg enkelt kombinere med de daværende løsningene. Dette er kanskje verdt å gå nærmere inn på, men jeg har en mistanke om at det stort sett er TSP spesifikt). Uansett, så ser vi at det samme "droppet" skjer ti-tyve generasjoner senere i C populasjonen, noe som tyder på at nå har B eksportert disse



Figur 7: Linjestruktur - 3 noder a 700 individer
Øverst kan man se en graf over de beste løsningene hver node har funnet. Under er det plottet gjennomsnittet av løsningene.



Figur 8: Tilfeldig nodenett for TSP og et eksempel på løsning

forstyrrende genene. Mest trolig har A eksportert genene i 790, så har B eksportert dem i 800 - og rundt 805 har de blitt plukket opp.

I det store og hele ser vi at en kobling mellom subpopulasjoner bidrar til å finne bedre resultater.

5 Forsøk

Jeg skal i dette kapittelet se på hvordan systemet klarer å løse et tilfeldig TSP problem. Dette gjøres ved å lage 256 noder innenfor området [-1,1][-1,1]. Det faktiske nodenettet jeg bruker er vist i figur 8. 256 er valgt for å redusere minnebruken. Her kan man bruke en enkelt byte for å representere en "by". Ved mer enn 256 vil minnebruken øke til nesten det dobbelte.

256 noder er veldig lite, men siden mange av forsøkene må kjøres flere ganger er, dette forholdsvis fornuftig.

Hvert punkt vil ligge rundt $\frac{1}{8}$ fra sitt nærmeste andre punkt. Dette anslaget gjøres på grunnlag av at det finnes 16*16 punkter, som fordeles på en 2*2 stor flate. Reduseres problemet til én dimensjon, blir det 16 punkter fordelt på avstanden 2, altså langs en linje. Da punktene er vilkårlig plassert, skal de for store mengder tilsvare en jevn fordeling. Vi kan da beregne at de fyller ut hele avstanden, og middelavstanden til det nærmeste punktet blir da $\frac{2}{16} = \frac{1}{8}$. Dette bør kunne generaliseres til to dimensjoner. Den optimale

turen bør da ligge noe under 32 enheter, faktisk en del under. I *worst case* er punktene helt jevnt fordelt og kun i dette tilfellet vil den korteste turen være 32 enheter lang. Systemet bør følgelig finne løsninger som er bedre enn dette ved å utnytte at flere punkter ligger i klynger. Dette er selvsagt et ganske grovt estimat, men bør gi en viss indikasjon på om ting stemmer. Den beste løsning som ble funnet, var -25.023.

Gitt samme argumentasjon, kan vi gjette oss til at en tilfeldig valgt løsning vil ha en total på rundt 260, fordi hvert punkt i snitt har avstanden 1 til et vilkårlig annet punkt. Dette stemmer også rimelig bra. En initialpopulasjon har gjerne fitnessverdier mellom -240 og -290.

5.1 Sirkelstruktur med kontrollnoder

Populasjon

0,1,2 700

C 700

T 2100

Mutation 20%

Struktur 0→1→2→0, C og T er uavhengige.

Eksport 10 Individer hver 10. generasjon

Import 1 Individ hver generasjon.

T og C er her kontrollnoder. C er en uavhengig node, som er tatt med for å vise hvordan en sammenkobling påvirker nodene. Noden C er identisk med 0,1 og 2, bortsett fra at den ikke er koblet til noen annen node. T er en annen uavhengig node som har en populasjon lik alle de distribuerte, i dette tilfellet $3 * 700 = 2100$.

Forsøket gir ikke nevneverdig ny informasjon i forhold til hva man skulle forvente. Det at viser tre noder a 700 individer har en raskere utvikling enn

en enkelt node på 700 individer. I figur 9 er det vist et enkelt eksempel på dette forsøket. Figuren viser er at distribueringen har en positiv innvirkning på evolusjonen, samt at det ikke er fullt så bra som en samlet populasjon i dette tilfellet. T holder faktisk på å spore av et sted rundt generasjon 400, men henter seg raskt inn igjen. Mulig vi her ser en stagnering på ca tre runder? Med en runde tenker jeg her på den korteste veien, det finnes et lokalt optimal rundt hele multiple av denne beste løsningen. (Se 4.1)

Den enkle noden C gjør det i det store og hele svært bra. Det kunne være interessant å se når den tar igjen 0,1,2 og T, men dette blir ikke gjort ikke i denne oppgaven.

5.2 Skalering

Populasjon

0,1,2 n

T 3n

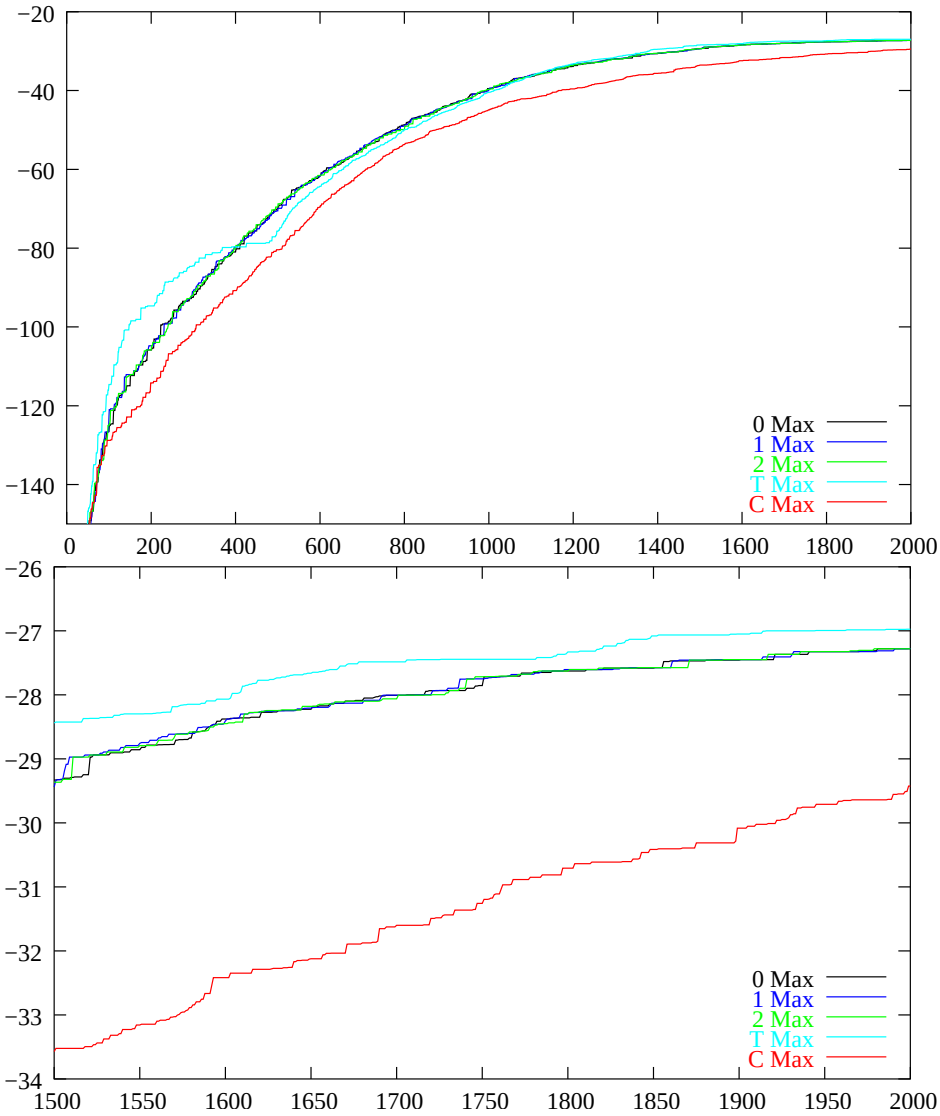
C n

Mutation 20%

Struktur 0→1→2→0, T og C er uavhengige.

Eksport 10 Individer hver 10. generasjon

I dette forsøket skal jeg se hvordan en distribuert evolusjon forholder seg til en samlet evolusjon når man varierer populasjonstørrelsen, hvor den samlede evolusjonen har like mange individer som alle de distribuerte nodene. Hensikten er å se hvordan dette forholder seg til populasjonstørrelsen. Vi vet av [1] at en samlet populasjon bør ligge mellom 1600 og 3200 for dette problemet. Siden det er kjent at populasjonstørrelser over 3200 ikke gir særlig økning, testes populasjoner med 3000, 2250, 1500, 750, 300 og 150 individer. De distribuerte nodene får da henholdsvis 1000, 750, 500, 250, 100 og 50



Figur 9: 3 runs med sirkelstruktur på tilfeldig nodenett. 3+1 node a 700 individer.

individer hver. Jeg tar også med noen høyere tall for å se hvordan det går i disse tilfellene, litt for å se om det oppstår bedre løsninger i løpet av samme tidsperiode grunnet parallelliseringen.

Det brukes samme vilkårlige nodenett som tidligere, og antatt beste løsning ligger noe over -26.

Vi kan betrakte trioen 0,1,2 som én modell, altså den distribuerte modellen (D), og noden T som en annen modell, som inneholder like mange individer som D. Til slutt har vi også noden C, som er lik en enkelt node fra D, men uten koblinger. Resultatene finnes i tabell 1.

Her kan det forventes at jo færre individer vi har totalt, jo større vil forskjellene mellom de ulike modellene være.

Figur 10 tar for seg maksimal fitness ved subpopulasjonstørrelser på 500 og 1000. Vi ser her følgende fenomener:

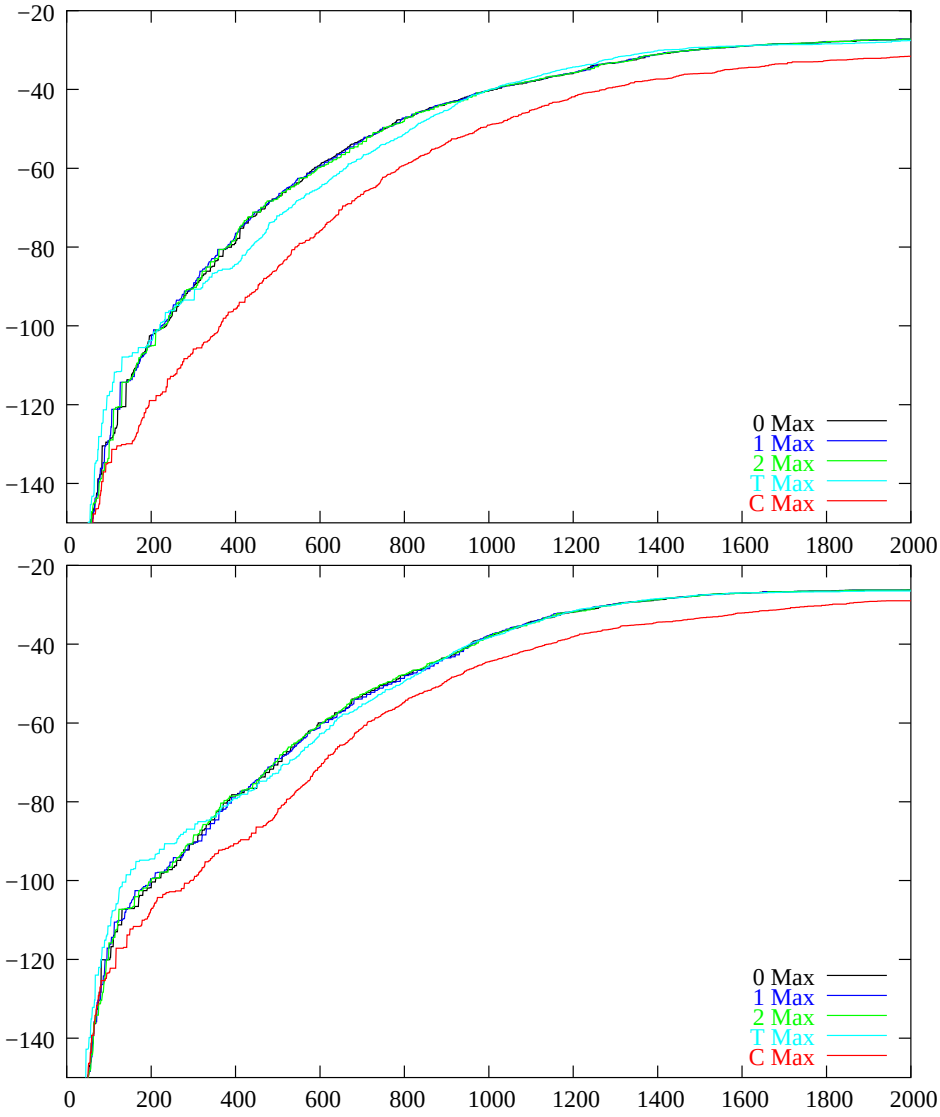
1. Den distribuerte modellen D får stort sett like gode resultater som T. I noen tilfeller kommer T inn på et noe heldigere spor og får en mer gunstig utvikling, men blir gjerne slått av den distribuerte modellen. I de fleste tilfellene hvor den samlede populasjonen er større enn et minimum finner D en bedre løsning enn T.

2. C gjør det konsistent dårligere enn både T og 0,1,2.

3. Ved svært små populasjoner gjør T det ofte best, mest trolig fordi subpopulasjonene er langt under det minimum som kreves.

4. Det kan virke som det først er mot slutten av evolusjonen at distribuering og store populasjoner er nødvendig. Dette kan være en problemstilling for en senere analyse. Øking av populasjonen i slutten av evolusjonen kan for så vidt delvis simuleres ved å øke mutasjonen når standardavviket synker.

Merk: dette er resultatet av enkle kjøringer, og må tas kun som et overslag.



Figur 10: Distribuerte noder i forhold til en samlet node. Den øverste grafen viser utviklingen med totalt 500 individer. Figuren under viser utviklingen ved 1000 individer.

Popsize	Total(T)	Distribuert(D)	Kontroll(C)	D - T	D - C
50	-38.6063	-39.0969	-53.8323	-0.4906	14.7354
100	-30.6137	-31.5658	-44.4888	-0.9521	12.9230
250	-29.3875	-28.7882	-35.0474	0.5993	6.2592
500	-27.2598	-27.2445	-31.5984	0.0153	4.3539
750	-27.105	-26.5685	-28.1621	0.5365	1.5936
1000	-26.4803	-26.2569	-28.9619	0.2234	2.7050
1500	-26.2018	-25.9398	-27.9568	0.2620	2.0170
3000	-27.0553	-26.243	-26.5686	0.8123	0.3256

Tabell 1: Resultater med hensyn til ulik populasjonstørrelse

6 Eksporteringsmekanismer

I dette forsøket varieres graden av eksportering for å se hvor stor effekt dette gir. Utgangspunktet for disse forsøkene er tre populasjoner a 750 individer som måles mot en enkelt populasjon på 2250 individer. Det kjøres forsøk med eksport av 1, 5, 10, 50, 100, 150 og 300 individer hver tiende generasjon. I dette forsøket brukes en elitism, tournament og random eksportering. Det brukes her som tidligere tre noder i en sirkelstruktur. Hvert forsøk gjøres ti ganger og resultatene som oppgis er gjennomsnittet av disse.

De ulike fasene som brukes i disse forsøkene er

1. Eksport
   Eksport av individer. Eksporteringsmekanismen og antallet varierer.

2. Import

3. Elitism (1)
   Det beste individet kopieres videre til neste populasjon. Merk at dette gjøres før mutasjonen.

4. Mutate (20%)
   For hvert individ er det 20% sannsynlig at det forekommer en mutasjon.

5. Tournament Combine (50%)

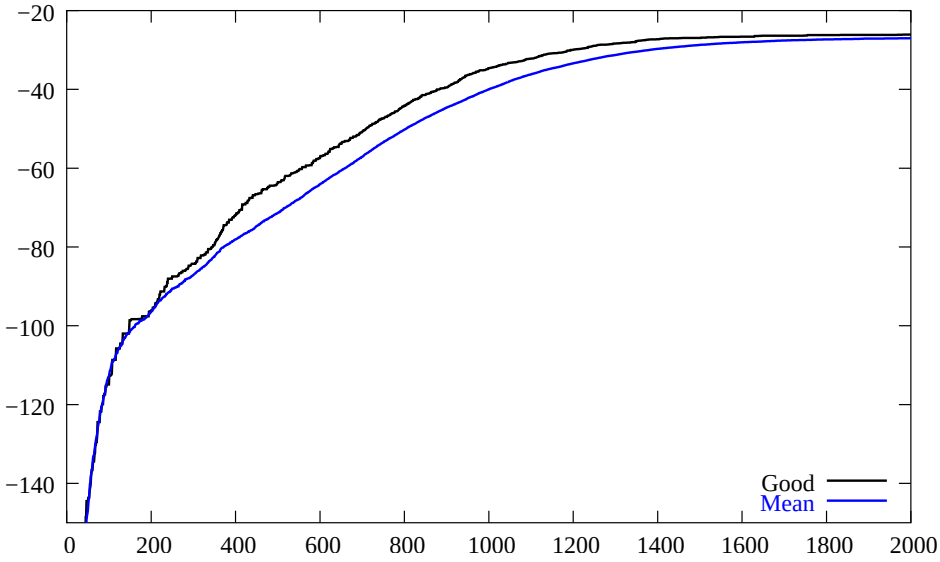
50% av neste generasjon fylles opp med individer som er laget av kombinasjoner.

6. Tournament Copy (50%)
   Mest 50% av neste generasjon fylles av individer kopiert fra nåværende populasjon. Fordi neste generasjon allerede er fylt opp av mer enn 50%, på grunn av importen, blir det ikke fullt 50%, men litt under.

Importraten er satt til eksportrate/eksportfrekvens per generasjon rundet opp. Dette vil i praksis si at alle eksporterte individer blir tatt i mot i løpet av eksportfrekvensen, som her er ti generasjoner. En eksport av 50 individer hver 10. generasjon vil føre til en import på 5 individer hver generasjon. Importerte individer begrenser den siste fasen som utføres i hver generasjon. 5 importerte individer vil føre til 5 individer mindre produsert i den siste fasen, med andre ord vil unnlate å kopiere fullt 50% av individene. Kort fortalt er importraten lik eksportraten, bare at den fordeles ut over hele det tilgjengelige tidsrommet.

Grafene plottes som beste fitness hos én av de tre koblede nodene subtrahert beste fitness hos den store populasjonen. De viser med andre ord forskjellen mellom en samlet og en distribuert modell. Videre er disse jevnet ut for å redusere støyen. Grafene viser hvor godt nodene gjør det i forhold til en enkelt stor populasjon med like mange individer som alle nodene til sammen. Altså, vi ser på resultatene fra de distribuerte nodene subtrahert med resultatet fra en samlet populasjon. En positiv verdi sier med andre ord at det distribuerte systemet finner en bedre løsning. Førsteaksen er antall generasjoner og andreaksen er en absolutt fitnessdifferanse.

Vi må her velge noen evolusjoner av en samlet populasjon som referanse. Vi velger da to stykker, en god (GOOD), og en gjennomsnittlig (MEAN) - konstruert av gjennomsnittet av ti kjøringer. Disse to slutter på henholdsvis -26.1 og -27.0. Se figur 11.



Figur 11: Referansepopulasjoner

6.1 Elitism som eksport

Elitism eksport fungerer ved å finne de n beste individene ved hvert eksporteringstrinn og kopierer disse individene videre til en annen subpopulasjon.

Figur 12 gir ikke så mye informasjon, Den viser at distribusjonen til tider gjør det bedre enn en gjennomsnittlig samlet populasjon, særlig rundt midten av evolusjonen. Dette kan for så vidt tyde på en noe for rask konvergering, siden den samlede populasjonen stort sett tar innpå senere.

Det er interessant å observere den vanlige formen som gjelder for distribuerte sammenlignet med gjennomsnittlige samlede populasjoner. De samlede gjør det mye sterkere de første 400 generasjonene, men de distribuerte tar forspranget igjen i generasjonene 400-1000. Etter dette trinnet konvergerer de begge sakte mot en rimelig god løsning. En mulig forklaring på dette fenomenet er at en samlet populasjon har tilgang på langt flere gener i utgangspunktet. Alle er tilgjengelig for kombinasjon. De distribuerte vil bruke mange generasjoner på å finne gener som sammen gir et godt resultat. Hvorfor de distribuerte gjør det bedre i området 400-1000 står fremdeles litt åpent, selv om det mest trolig skyldes at elitism får hele populasjonen til å spesialisere

seg rimelig fort.

I grafen for sammenligning med et godt resultat i figur 12 ser man noen rimelig koherente svingninger. Dette skyldes først og fremst at det gode resultatet har noen ujevnheter. Disse vises dårlig på figur 11 fordi svingningene bare er på et par fitnesspoeng. Grafene som vises for elitism, har mye mindre målestokk, dermed vil utslagene komme tydligere fram.

Det er også tydelig at de distribuerte sjelden gjør det like godt som den beste referansepopulasjonen. Dette har sammenheng med at evolusjoner sjelden finner et like godt resultatet som vi gjorde i denne referansepopulasjonen, samt at de distribuerte her er et gjennomsnitt. Mot et gjennomsnittstilfelle går det noe bedre. Eksporteringsgraden ser ut til å ha forholdsvis lite innvirkning. Det er godt mulig at man heller burde måle et utvalg av de beste populasjonene mot den beste referansepopulasjonen.

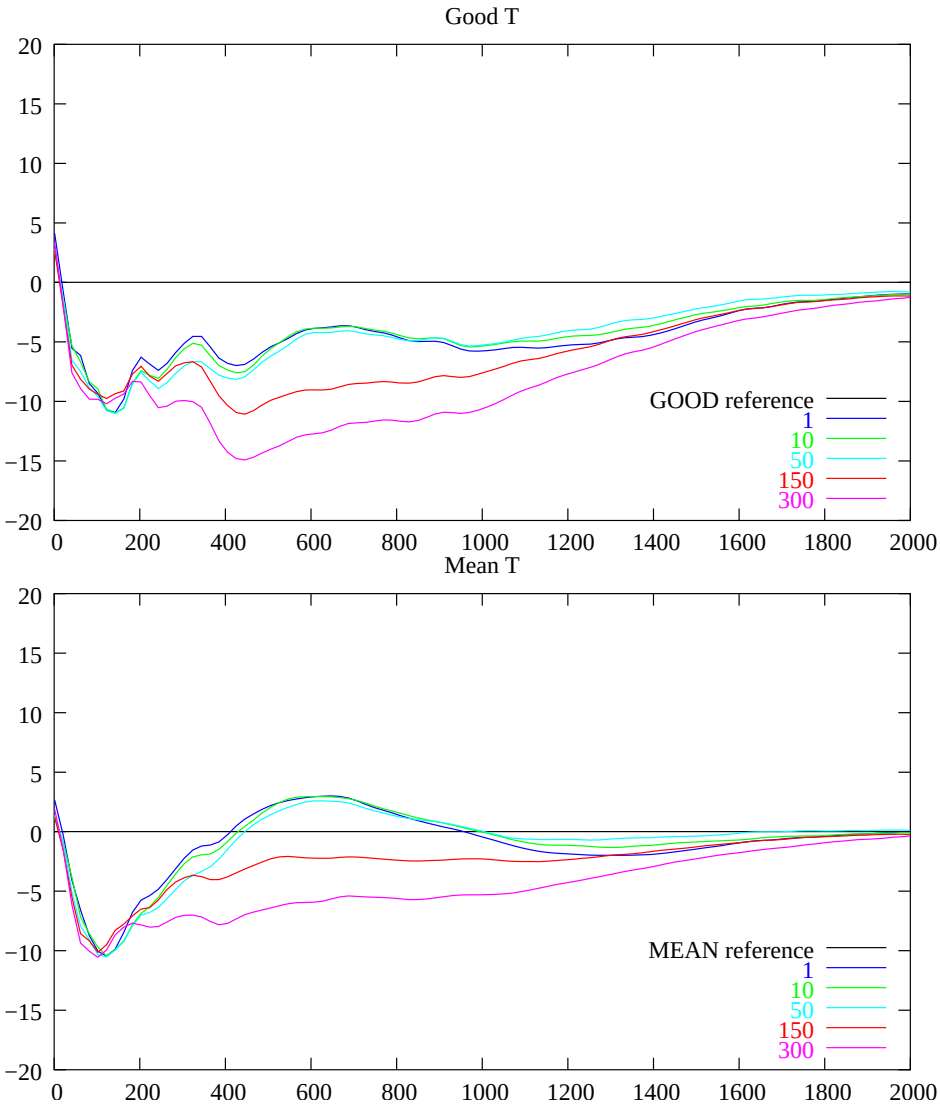
Resultatene ender opp omtrent som resultatet fra gjennomsnittsreferansepopulasjonen. Et resultat som gjennomsnittet er egentlig det beste vi kan regne med å finne, så dette er et rimelig godt resultat. Et resultat som gjør det like godt som gjennomsnittet av en samlet modell, sier at distribusjonen ikke har noen negativ innvirkning. Andre [12] har rapportert om noe bedre (super lineære) resultater fra distribuerte modeller.

6.2 Tournament eksport

Tournament eksport går ut på å velge to tilfeldige individer, for så igjen å velge den beste av disse med 60% sannsynlighet. Lave eksporteringsgrader vil da være svært nær et tilfeldig utvalg. Større eksporteringsgrader vil resultere i et stokastisk godt utvalg.

Figur 13 viser resultater som ligner noe på elitism eksport, bare at der hvor elitism får dårligere resultater ved høye eksporteringsgrader, får tournament60% bedre. En elitism-eksport på 1-50 gjør det rimelig likt som tournament60% eksport med 50-100 individer.

En større sannsynlighet for å velge det beste individet vil mest trolig få en heldigere utvikling. Så lav sannsynlighet for å velge det gode individet som



Figur 12: Elitism eksport i forhold til en samlet populasjon.
Øverst mot den gode (GOOD) referanse-evolusjonen og til under mot den gjennomsnittlige (MEAN)
Førsteaksen er antall generasjoner og andreaksen beskriver den absolutte fitnessforskjellen mellom referansepopulasjonen og en distribuert modell. Ved en positiv andreakse har den distribuerte modellen et bedre resultat enn referansen.
De ulike grafene har ulik eksporteringsgrad som beskrevet tidligere.

benyttes her vil ikke gi mye ekstra til en distribuert evolusjon. Det vil igjen si at de andre nodene ikke får tilstrekkelig tilgang på gode gener fra andre eksporterende noder.

6.2.1 Tournament eksport med 80%

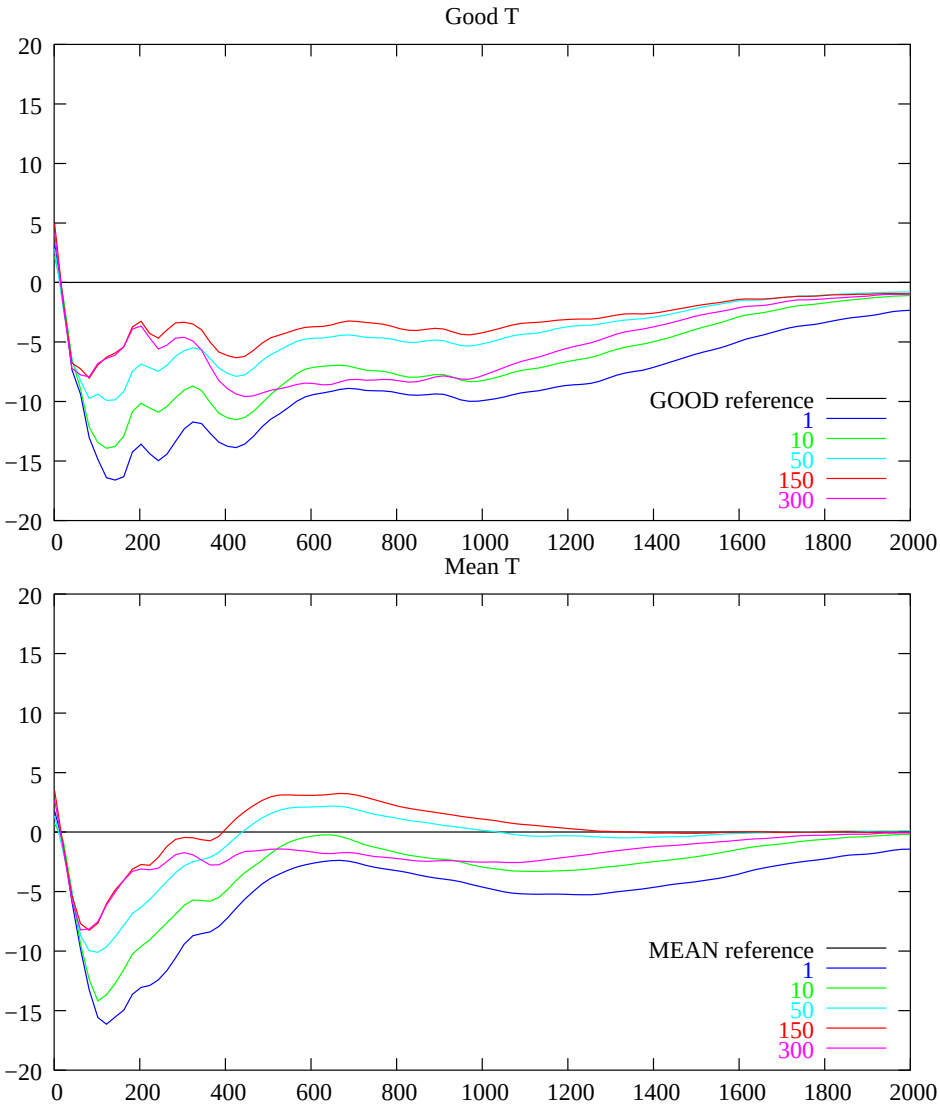
Her skal jeg la sjansen for å velge det beste individet være 80% og ikke 60% som i forsøket over. Dette kan bidra til at systemet får en utvikling noe nærmere elitism, siden en større andel gode individer kan eksporteres.

Figur 14 viser at dette gir noe bedre resultater enn antatt. Forsøkene gjør det jevnt over såvidt bedre enn den gjennomsnittlige referanseevolusjonen, selv om forskjellene er så små at de egentlig har liten betydning. Det er imidlertid interessant å se at denne eksporteringsmetoden gir resultater som ligger godt foran samlede populasjoner ved midten av evolusjonen.

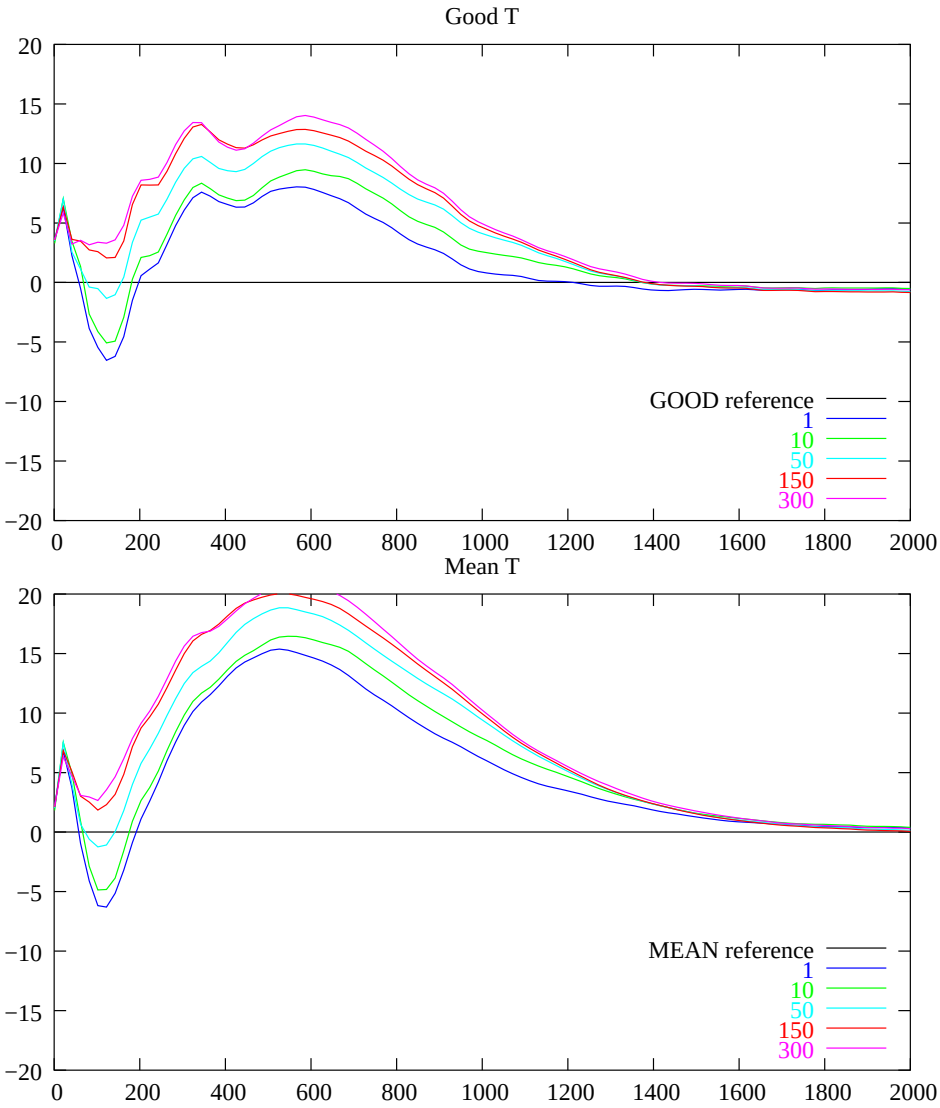
Ved tournament velger man ut n individer som har en gitt sannsynlighet for å være det beste av to tilfeldige individer. Ved større eksporteringsgrader vil man da ha flere gode, men ulike individer å eksportere. Man kan også velge samme individet flere ganger. Disse to faktorene fører til at eksporten både er variert og har høy fitness. Dette kan forklare en raskere utvikling.

Evolusjonene følger hverandre rimelig tett. Det ser ikke ut som eksporteringsgraden har noen ekstrem påvirkning, bortsett fra noe korrelasjon mellom høy eksportrate og hurtig utvikling. 300 eksporterte individer hver 10. generasjon har den raskeste utviklingen. En så kraftig start på utviklingen kan muligens gi problemer senere, hvis man lar evolusjonen gå lenger enn 2000 generasjoner. Dette blir omtalt nærmere senere i dette kapittelet.

Imidlertid er det eksportraten 10 som finner det beste resultatet, samt har gjennomsnittlig beste sluttresultat. På den annen side er differansen mellom de ulike gjennomsnittlige sluttresultatene mindre enn standardavvikene, så resultatet må tolkes med forsiktighet.



Figur 13: Tournament eksport 60% i forhold til en samlet populasjon. Øverst målt mot den gode (GOOD) referanseevolusjonen. Den under mot den gjennomsnittlige (MEAN).
Førsteaksen er antall generasjoner og andreaksen beskriver den absolutte fitnessforskjellen mellom referansepopulasjonen og en distribuert modell. Ved en positiv andreakse har den distribuerte modellen et bedre resultat enn referansen.



Figur 14: Tournament eksport med 80% målt mot en samlet populasjon. Øverst mot en enkelt god utvikling og under mot en gjennomsnittlig utvikling.
Førsteaksen er antall generasjoner og andreaksen beskriver den absolutte fitnessforskjellen mellom referansepopulasjonen og en distribuert modell. Ved en positiv andreakse har den distribuerte modellen et bedre resultat enn referansen.

6.3 Random eksport

Random eksport innebærer å velge ut tilfeldige individer og eksportere dem. På grunnlag av hva som skjedde med tournament eksport forventes det en rimelig dårlig grad av utvikling her. Dette tilsvarer egentlig tournament eksport med 50% sannsynlighet for å velge det beste individet. Etter å ha sett forskjellen mellom 60% og 80% får vi neppe noe veldig overraskende resultat av dette forsøket.

Figur 15 viser omtrent den samme tendens som for tournament eksport. Svært få eksporterte individer (i dette tilfellet ett enkelt individ) har en litt dårligere utvikling. Likevel kommer resultatet på omtrent samme nivået som en gjennomsnittlig populasjon, med et unntak for den laveste eksporteringsgraden.

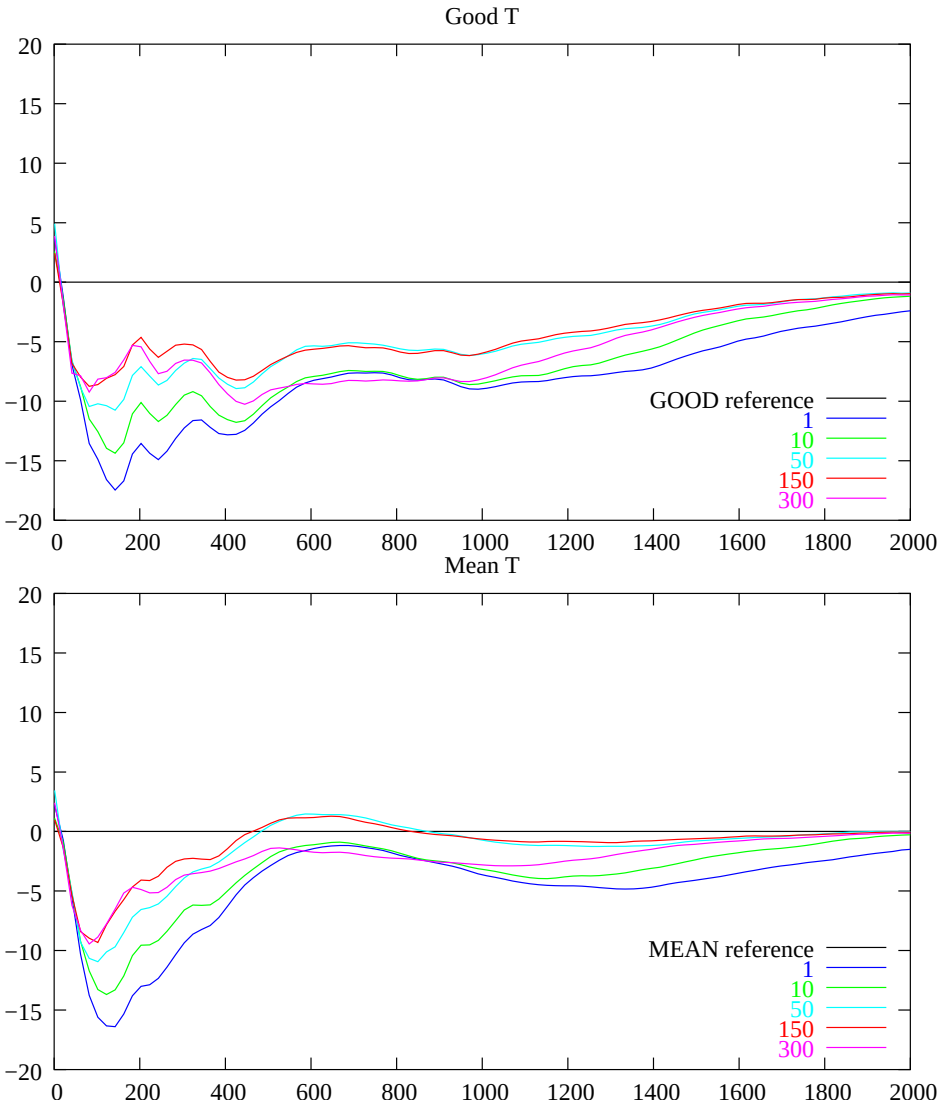
Ved tilfeldig eksportering kan man tilfeldigvis finne noen bedre resultater, fordi evolusjonen ikke like enkelt henger seg opp. Hver node mottar ikke mange gode individer fra noe annet sted som kan presse ut den lokale populasjonen.

6.4 Sammenligning

Jeg skal nå vurdere hvordan de ulike eksporteringsmekanismene gjør det i forhold til hverandre. Grafen viser en sammenligning mellom de ulike eksporteringsmekanismene, men hvor bare de beste eksporteringsgradene er med. For elitism, random og tournament60% er 50 den beste eksporteringsgraden av de utprøvde. Tournament80% finner et litt bedre resultat på 10 som eksportrate, men for å gjøre det mest mulig likt brukes 50 for denne også. Denne eksportraten hadde fremdeles et bedre sluttresultat enn de andre. Resultatet er vist i figur 16.

Resultatene kan raskt grupperes, stort sett alle har rimelig lik utvikling, bortsett fra tournament80% og den gode referansepopulasjonen som har en noe heldigere utvikling.

For alle forsøkene i dette kapittelet var det beste resultatet etter 2000 generasjoner -25.820 og det dårligste -29.228. Gjennomsnittlig var resultatet på



Figur 15: Elitism eksport målt mot en samlet populasjon. Førsteaksen er antall generasjoner og andreaksen beskriver den absolutte fitnessforskjellen mellom referansepopulasjonen og en distribuert modell. Ved en positiv andreakse har den distribuerte modellen et bedre resultat enn referansen.

	Elitism	Tournament 60%	Tournament 80%	Random
1	**-27.06581**	-28.43795	-26.65572	-28.50154
10	-27.08711	-27.20058	**-26.6015**	-27.27193
50	-26.89679	-26.8938	-26.85389	**-27.00569**
150	-27.24302	**-27.0412**	-26.9428	-27.06318
300	-27.36076	-27.08342	-26.71384	-27.16306
Beste	-25.8201	-25.9516	-25.8751	-26.2266

Tabell 2: Beste sluttresultater for ulike eksporteringsmekanismer (gjennomsnitt av 10 kjøringer)
Raden "Beste" viser det beste resultatet som ble funnet i en enkelt kjøring, kontra gjennomsnittet av ti kjøringer som står ellers i tabellen.

-27.090 med et sample standardavvik på 0.659.

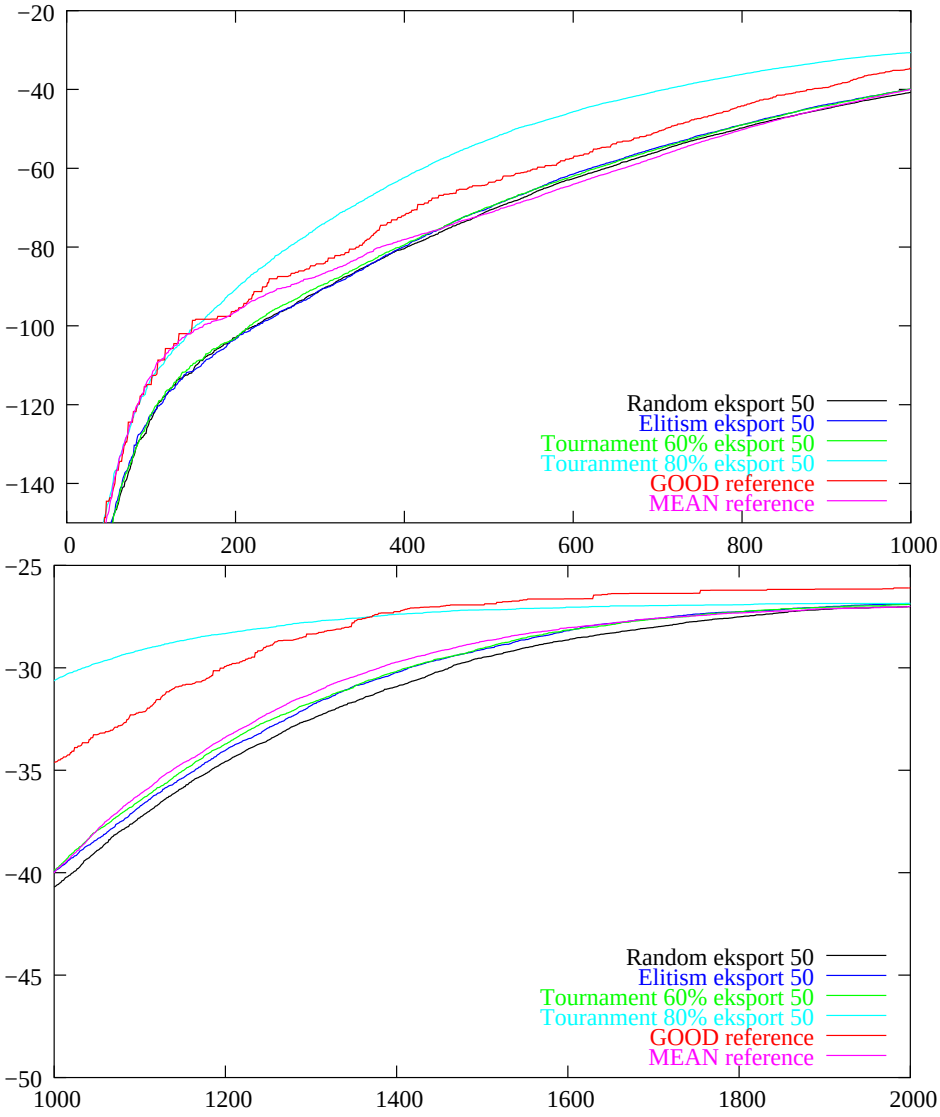
Om man heller ser på den gjennomsnittlige beste innstillingen (tournament 80% med 10 eksport) er det beste resultatet -25.875 og det dårligste -27.304. Gjennomsnittet ligger på -26.601 med et standardavvik på 0.421, som er rimelig konsistent gode løsninger. Disse innstillingene hadde også det absolutt beste resultatet.

6.4.1 Utviklingen videre

Det kan være interessant å se hva som skjer videre i utviklingen, altså etter 2000 generasjoner. I figur 17 vises fire eksempler på dette, med ulike mekanismer. Figuren gjengir både det enkelte beste resultatet som ble funnet ved en enkelt evolusjon, og et gjennomsnitt av alle evolusjonene med en gitt eksportmekanisme. Det er samtidig tatt med en enkelt node med 750 individer.

Alle modellene bruker 50/10 som eksport, siden disse tallene hadde det beste sluttresultatet i testen over. Noe overraskende får random-eksporteringen fram det beste resultatet. Legg merke til at det ikke blir store forbedringer her. Fra den 2000. generasjonen til 5000. skjer det stort sett ingen merkbare forandringer. Elitism har f.eks. en forbedring på 0.7 fitnesspoeng.

Hvis man tar de gjennomsnittlige sluttresultatene i betraktning, ser det



Figur 16: Sammenligning av ulike ekporteringsmekanismer
Grafene viser gjennomsnittet av alle eksporteringsgradene, bortsett fra referanseutviklingene.

fremdeles ut som tournament80% er den beste løsningen. Imidlertid overrasker random eksport stort ved å finne den desidert beste løsningen. Dette skyldes nok at tournament80% nesten har en premature convergence, hvor random fremdeles klarer å holde på mange ulike gener, siden hver subpopulasjon ikke får noe målrettet tilstrømning av gener.

Elitism og tournament80% taper imidlertid forspranget de hadde etter rundt 1000 generasjoner. Etter dette ligger de rimelig likt igjen. At random eksport til slutt får det beste resultatet er noe overraskende, men kan mest trolig forklares av en større utforskningstendens enn de andres mer utnyttelsesrettede strategier. Random eksportering kan til dels sies å totalt mangle utnyttingsaspektet fordi det per definisjon ikke er noen tanke bak hvilke individer som eksporteres. På den andre siden går evolusjonen sin gang i den enkelte subpopulasjon.

Videre kan man sammenligne forskjellene på gjennomsnittsgrafen og enkeltforsøket med den beste løsningen. Forskjellen mellom de beste løsningene funnet og gjennomsnittet er noe under ett enkelt fitnesspoeng (0.7 - 0.4). Standardavviket[10] fra gjennomsnittet er 0.2-0.45 fitnesspoeng.

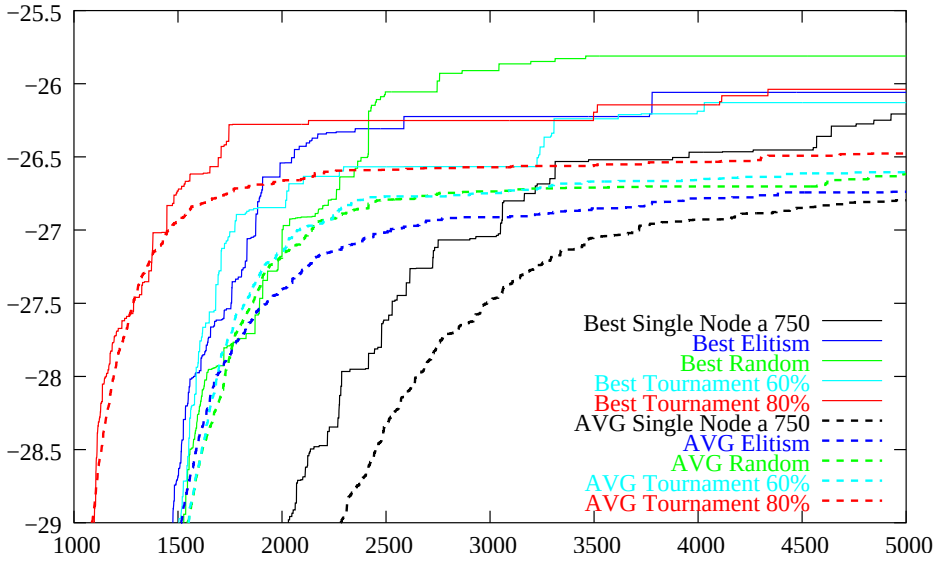
Forskjellen mellom de ulike mekanismene blir lite tydelige ved at det bare gjennomføres ti forsøk. Standardavviket er omtrent lik forskjellen mellom de ulike gjennomsnittlige sluttresultatene.

6.5 Konklusjon

Forskjellen på eksportmekanismer er rimelig liten, men en tournament-variant, gjerne med forholdsvis høy sannsynlighet for å velge det gode individet, har en noe raskere utvikling enn andre metoder. Derfor skal denne metoden benyttes videre.

En annen måte å tolke resultatene på, er at det ikke betyr så mye hvilken mekanisme man bruker som eksportseleksjon. Dette kan igjen kanskje innebære at distribueringen ikke gjør så mye, eller det kan si at så lenge en

[10]Sampling standard deviation



Figur 17: Ulike eksporteringsmekanismer i lengre perioder
Både de absolutt beste resultatene og gjennomsnittet for en eksporteringsmekanisme vises.

har en viss utveksling av gener, så vil distribueringen gi et positivt bidrag til utviklingen.

6.5.1 Problemer

Det kan godt hende at resultatene fra denne enkle testen ikke lar seg generalisere til mange noder. Ved mange flere noder kan det hende at forskjellene mellom mekanismene slår mye mer ut, eller det kan hende at de skifter karakter.

Tournament-seleksjon kan også ha en ulempe ved at gode gener kan dominere mye av den totale populasjonen i løpet av forholdsvis kort tid, noe som kan redusere muligheten til at ulike subpopulasjoner følger litt forskjellige spor.

7 Variasjon av antall noder

Jeg skal se på hvordan systemet reagerer når det deles opp. Systemet deles opp i 1, 2, 3, 5, 10, 15, 25 og 50 noder. Hvordan hver node oppfører seg skal varieres for hvert forsøk.

En tournament-export med 80% brukes som koblingsmekanisme mellom nodene. Jeg antar at dette generaliseres fra forsøket tidligere hvor systemet kun hadde tre noder. Videre antar jeg at eksporteringstrinnet ikke bruker noen nevneverdig CPU-tid i forhold til de andre skrittene.

7.1 Konstant total CPU-tid

Her skal vi se hvordan systemet fungerer hvis vi lar den totale CPU-tiden være konstant. For dette forsøket skaleres populasjonstørrelsen i forhold til antall noder. Den totale populasjonstørrelsen er 2250 i dette tilfellet, samtidig går evolusjonen alltid 2000 generasjoner. Dette vil i prinsippet si at en populasjon oppdelt i 50 noder blir ferdig 50 ganger så fort. Det forventes at en samlet populasjon gjør det best.

7.1.1 Konstant total eksport

I dette forsøker skal den totale eksporten holdes konstant samtidig som systemet deles opp i ulike antall noder. Hensikten er å se hvordan systemet reagerer hvis det deles opp med hensyn til kommunikasjonen. Det forventes at ytelsen svekkes etterhvert som antallet noder øker, siden en rundtur i systemet vil ta lengre og lengre tid, og systemet dermed har mindre utveksling av gener.

Populasjon per node 2250 / antall noder

Antall noder 1, 2, 5, 10, 15, 25, 50

Mutation 20%

Antall noder	subpopulasjon per node	beste resultat	gjennomsnitt av beste
1	2250	-25.7058	-26.75311
2	1125	-26.3135	-26.92613
5	450	-26.155	-26.58139
10	225	-26.1308	-26.68414
15	150	-26.3734	-27.1347
25	90	-27.6798	-28.8698
50	50	-33.1873	-34.46312

Tabell 3: Variasjon av antall noder
Det beste resultatet er beste resultatet som ble funnet. Gjennomsnitt av beste er gjennomsnittet av det beste resultatet for ti separate kjøringer med like parametre.

Struktur RING single

Eksport 100 Individer / antall noder hver tiende generasjon. Dette fører til at det totalt eksporteres 100 individer hver tiende generasjon. Tournament 80% brukes som eksportseleksjon.

Import Varierende. Alle individer blir tatt inn i løpet av ti generasjoner.

Resultatet står i tabell 3.

Evolusjoner med samme subpopulasjonstørrelse vises i figur 18. Den største forskjellen er noe dårligere resultater og litt raskere degradering ettersom populasjonen deles opp. Hovedinntrykket er at en oppdeling i flere enn 15 noder degraderes vesentlig. Dette er forventet, for en populasjon på 45 individer ikke har så mye å stille opp med.

I forhold til enkle populasjoner gjør imidlertid de distribuerte det svært bra. Dette er heller ikke særlig overraskende.

I gjennomsnitt finner man en løsning på -28.20 i fitnessverdi med et standardavvik 2.73. Den beste løsningen ligger på -25.706 fitnesspoeng og den dårligste har -35.539. Den beste løsningen er her akseptabel, men ikke helt optimal.

Det fremgår tydelig av figur 18 at en oppdeling i 50 noder gir et mye dårligere resultat enn de andre.

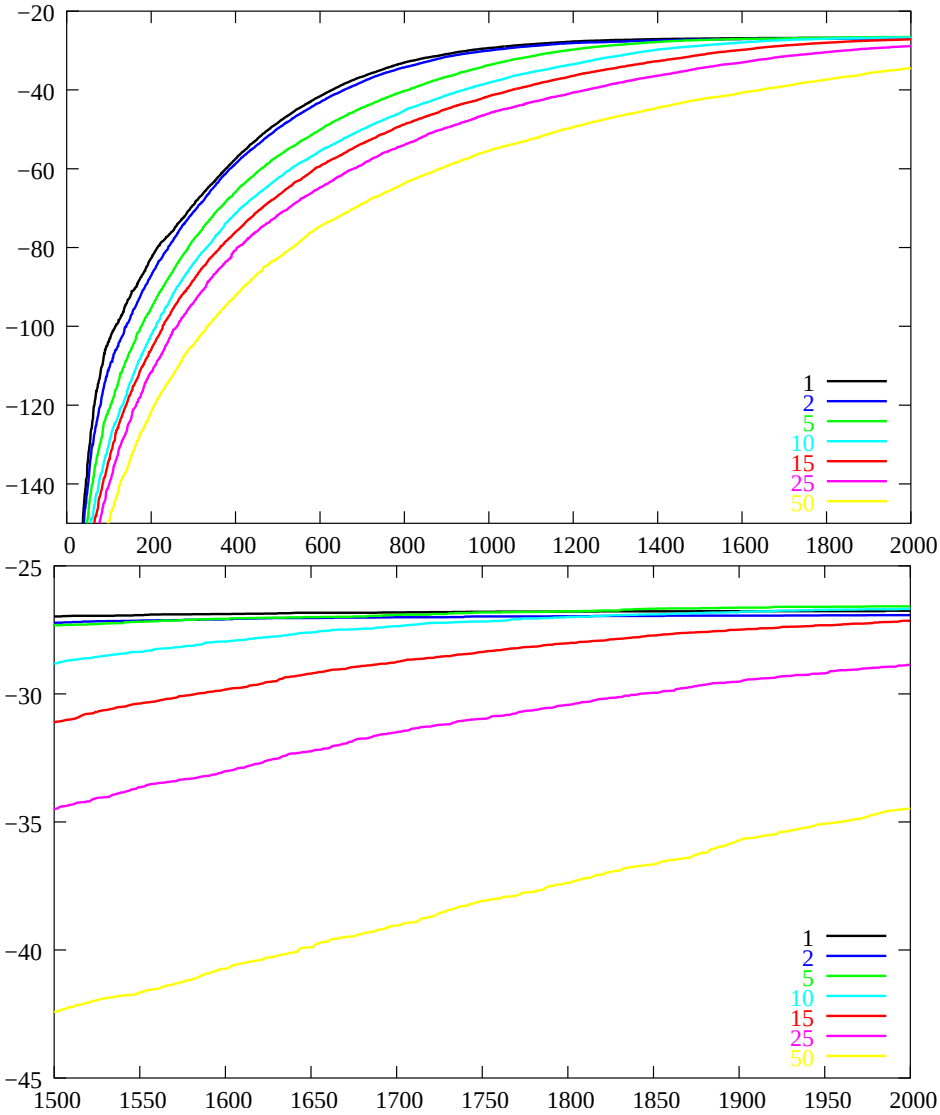
Oppdelte populasjoner gjør det svært bra. Man kan med rimelig grad av trygghet dele opp populasjonen i ti biter uten at det gir negativ effekt.

Dette er første kjøring hvor mange noder er brukt, og det kan da være interessant å se hvordan utviklingen går i forhold til de ulike nodene. Hvilke noder som for tiden har det beste individet tegnes opp, se figur 19. Grafen leses ved å se følge generasjonene langs førsteaksen, og observere hvilken node som har det beste individet på andreaksen. Hvor høye impulsene er sier med andre ord ikke noe om den absolutte fitnessen, men er et mål på hvilken av de n nodene som har det beste individet. Gitt at flere noder har like godt resultat, noe som forekommer ganske ofte, benyttes verdien til den noden som sist hadde det beste resultatet.

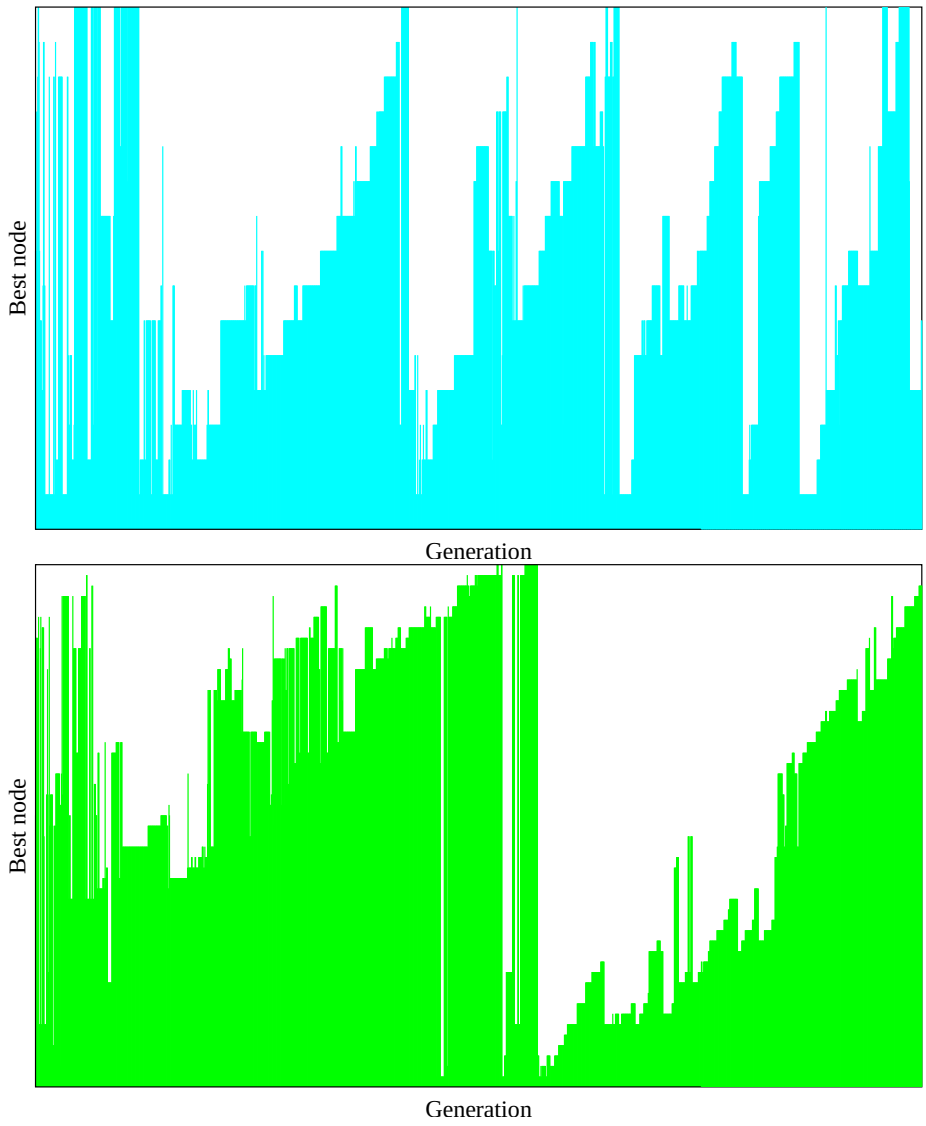
Et interessant fenomen er at det beste individet sakte flyter bortover nodene. Når vi eksporterer et individ, virker det som den nye populasjonen tilføres noe nytt som eksporteres videre til neste node. Subpopulasjonene utvikler seg dermed i litt ulike retninger, mens en beste løsning så langt passeres som en stafettpinne bortover lenken. Vi utnytter muligens en god divergerende populasjon til det beste, eller vi har problem med at det tar lang tid før de beste genene kommer fram. Følgelig kan retningen og hvordan noder er lenket sammen har mye å si.

Hvis man har flere noder tar denne mekanismen lenger tid. Figur 19 viser at for 15 noder ser det ut til at den beste noden går rundt hele systemet omtrent fire ganger. Dette ser vi på rampene oppover. Fenomenet vises tydeligere på varianten med 50 noder, hvor det er to tydeligere ramper. Vi opererer med en sirkulær struktur, så alle rampene henger sammen, og det beste individet beveger seg bare rundt.

Hvilken struktur man bruker kan da ha konsekvenser for hvor oppdelt man kan ha en evolusjon, og hvor mye eksport som trengs for å utnytte den valgte strukturen. Problemet om strukturforandringer er omtalt senere i oppgaven.



Figur 18: Ulike antall noder - gjennomsnitt av ti kjøringer med konstant total CPU-tid og en konstant total eksport.
Tallet for hver graf viser hvor mange noder evolusjonen er delt opp i. (1-50)
Den øverste grafen viser alle generasjonene, mens den andre viser et utsnitt av slutten.



Figur 19: Hvilken node har det beste individet. 15 noder (oppe) og 50 noder (nede)
I disse grafene er førsteaksen antall generasjoner som har gått (0-2000) og andreaksen sier hvilken node (1-15 for grafen til venstre og 1-50 for den høyre) som i det generasjonstrinnet har det beste individet.

7.1.2 Med variabel total eksport

Ved å la alle nodene eksportere like mye kan det hende resultatene forandres noe. Mye av den trege utviklingen i forsøket over kan mest trolig tilskrives en noe mangelfull kommunikasjon. Rundturstiden vil for så vidt ikke bli endret i dette forsøket i forhold til det over, siden man fremdeles eksporterer individer hver tiende generasjon. For 50 noder vil det da ta 500 generasjoner for at et individ hypotetisk skal kunne flytte seg fra en node og rundt hele runden. Siden vi har sett av figur 19 og senere vil se i figur 25 at de beste resultatene finnes i bølger som følger eksporteringsretningen, er det rimelig å anta at "rundetiden" har innvirkning på hvor rask utviklingen går.

Populasjon per node 2250 / antall noder

Antall noder 1, 2, 5, 10, 15, 25, 50

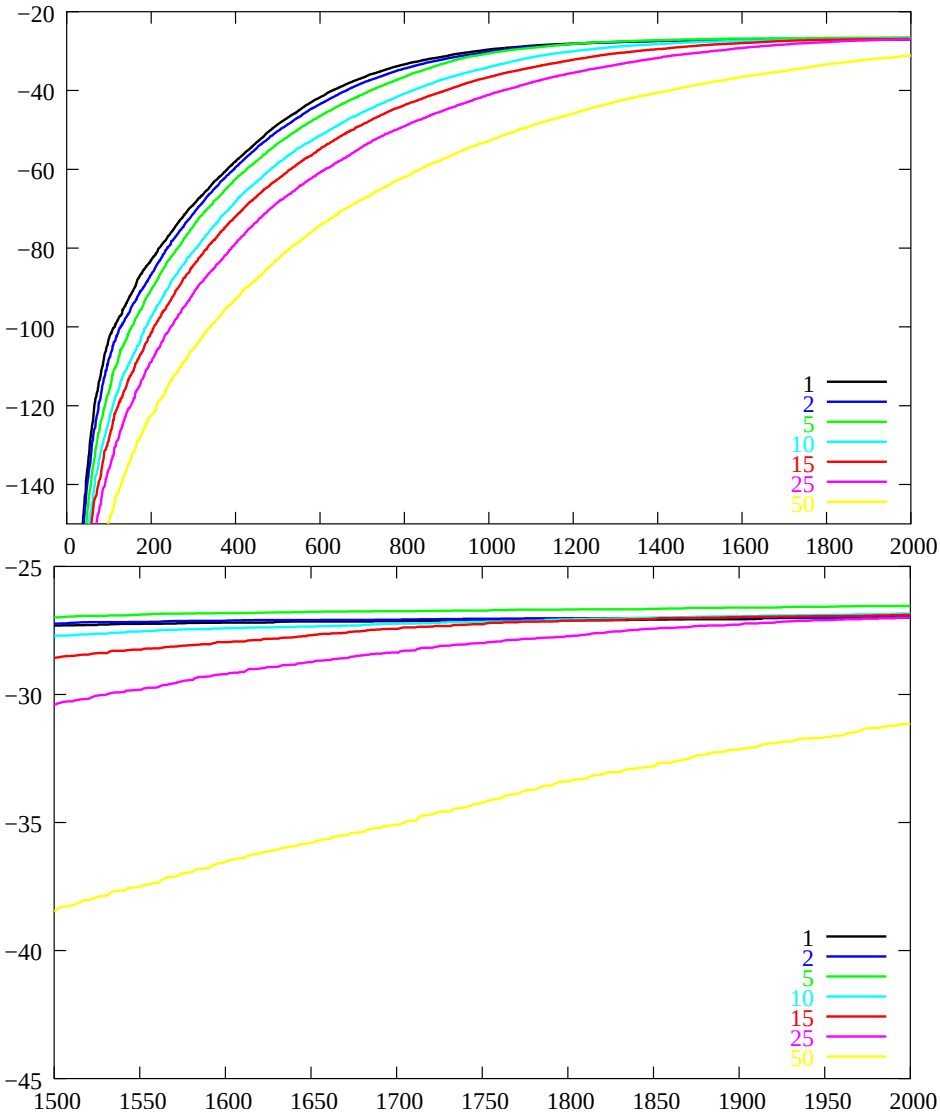
Mutation 20%

Struktur RING single

Eksport 100 Individer hver tiende generasjon. Dette fører til at 50 noder har en eksport på over antallet individer i hver populasjon. Hver node vil dermed eksportere en del individer minst to ganger.

Import Varierende. Alle individer blir tatt inn i løpet av ti generasjoner. For 50 noder tas det da inn ti individer per generasjon, som mest trolig er litt for mye, men likevel bør fungere.

Etter 70 kjøringer, ti for hvert antall noder man kan dele systemet opp i, hadde den beste løsningen man fant en fitnessverdi på -25.71. Gjennomsnittlig fitness var på -27.49 med et standardavvik på 1.61 og den dårligste var på -32.29, som er over tre fitnesspoeng bedre enn tidligere. En økt overføring av individer mellom nodene har tydeligvis en god innvirkning, først og fremst på systemer med mange noder.



Figur 20: Ulike antall noder - gjennomsnitt av ti kjøringer med konstant total CPU-tid, men hvor alle nodene eksporterer 100 individer hver generasjon. Tallet for hver graf viser hvor mange noder evolusjonen er delt opp i. (1-50)

Figur 20 viser grafer over utviklingen. Det mest slående i forhold til forrige test er at det kun er systemet med 50 noder som ligger vesentlig etter de andre ved slutten av evolusjonen.

Spørsmålet jeg stilte meg selv etter dette forsøket var hvor mye eksport trenger man for at 50 noder skal kunne finne et resultat i samme klasse som de andre variantene.

7.1.3 50 noder

Jeg skal her forsøke å se litt på hvordan eksportratene virker inn på en evolusjon med ganske mange noder, her 50 noder. Hensikten er å finne ut hvordan man kan få resultatet best mulig i løpet av 2000 generasjoner. Vi vet fra tidligere at man skal kunne finne resultatet som tilsvarer noe under -25, men varianter med 50 noder har ikke kommet tilstrekkelig nære dette.

Systemet skal forsøke å doble eksportfrekvensen for å se om det kan ha noen positiv innvirkning. For dette tilfellet vil det si at individer eksporteres hver 5. generasjon. Med dette reduseres tiden det tar før gode gener har mulighet til å eksportere seg tilbake til utgangspunktet. Man skal ikke se bort fra at det skal mer til enn dette for å oppnå tilstrekkelige gode resultater. Det kan godt hende en annen struktur vil ha bedre muligheter for å finne et godt resultat. Strukturforandringer testes i et senere kapittel. I alle tilfeller kan resultatet av noen ulike eksporteringsmuligheter ses i figur 21.

Populasjon per node 45

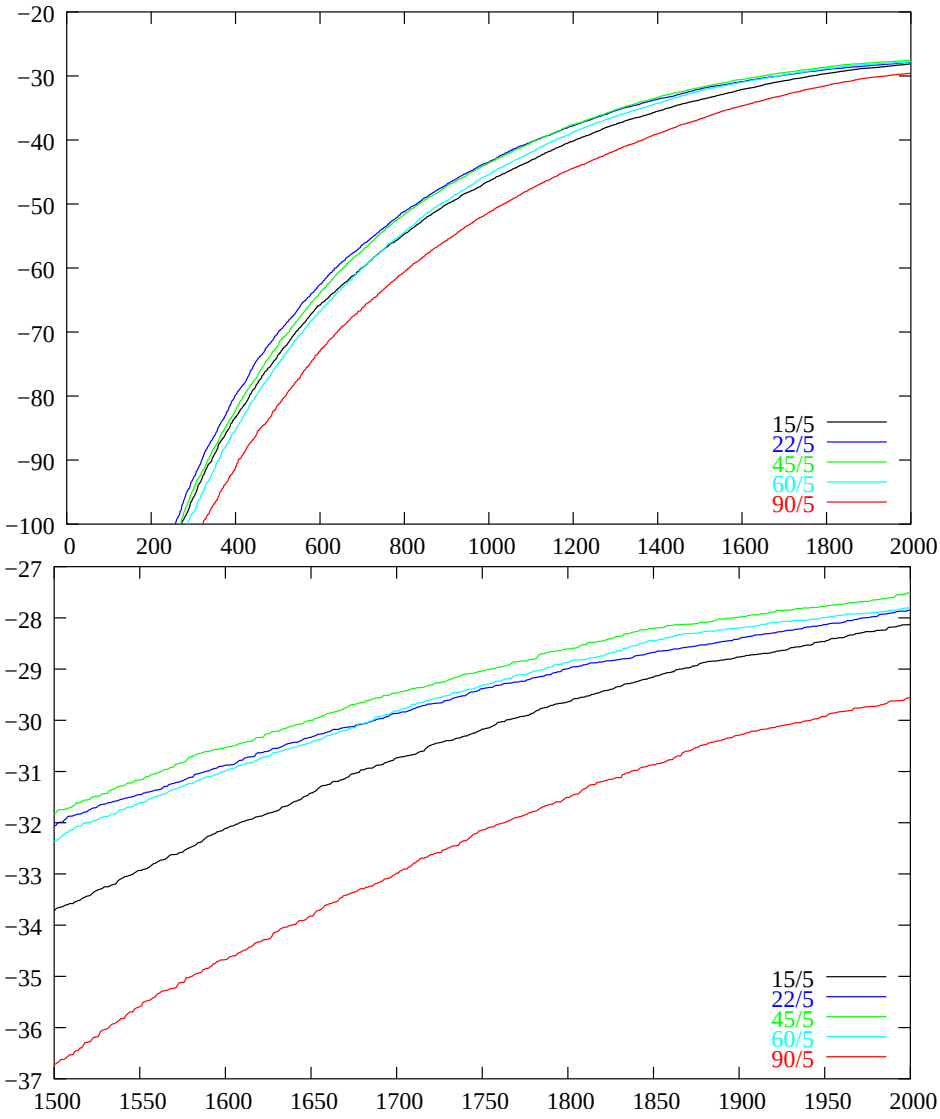
Antall noder 50

Mutation 20%

Struktur RING single

Eksport Varierende per 5 generasjoner.

Import Varierende. Alle individer blir tatt opp jevnt i løpet av eksportperioden.



Figur 21: Ulike eksporteringsmuligheter for 50 noder, gjennomsnittet av ti kjøringer.
Tallene for hver graf er antall individer eksportert per antall generasjoner. 15/5 er altså 15 individer eksportert hver 5. generasjon.
Den øverste grafen visere hele evolusjonen, mens den nederste viser et utvalg av slutten.

Resultatene er ikke veldig oppløftende, men heller ikke veldig overraskende. Den beste resultatet vi finner ligger -26.73. Dette var for 45/5 som eksport, som gir både det beste enkelte resultatet og det gjennomsnittlig beste. Sluttresultatene for denne eksporteringen varierte fra -26.73 til -28.55. Gjennomsnittet var -27.51 med et standardavvik på 0.5.

Mest trolig trengs en større grad av kommunikasjon for å forbedre resultatene, eller det kan hende at en subpopulasjon på 45 rett og slett er for liten til å kunne holde en god utvikling.

7.2 Konstant lokal prosessortid

Vi skal nå se hva som skjer når det benyttes like mye CPU-tid på hver node i forhold til at den totale CPU-tiden er konstant som i en samlet populasjon. Dette er samme forsøk som over, men vi lar populasjonen fortsette lenger. Dette vil da føre til omtrent like mange evalueringer per node som det gjøres i en samlet populasjon. Hensikten er her at dette tar like lang tid - fra brukerens synspunkt - som en enkelt populasjon. I og med at vi har en asymptotisk tilnærming mot ca -25 er det først og fremst de tidlige utviklingstrinnene som er interessante.

Merk at for kapittelet med konstant lokal prosessortid benyttes elitism som eksport.

7.2.1 Konstant total populasjonstørrelse og varierende antall generasjoner

Vi vil se hva som skjer om hver node få like mye prosessortid, også antall noder varierer. Femti noder vil da bruke totalt femti ganger mer prosessortid enn en enkelt node. Siden vi også skalerer subpopulasjonene slik at den totale populasjonstørrelsen er konstant, utvider vi antall generasjoner. Dette gjennomføres etter oppsettet i tabell 4, hvor også de beste resultatene er gjengitt.

Antall noder	subpopulsjon	generasjoner	2000. generasjon	sluttresultat
1	2250	2k	-27.0308	-27.0308
2	1125	4k	-27.6298	-26.6196
3	750	6k	-27.8822	-27.189
5	450	10k	-26.3309	-26.0541
10	225	20k	-27.421	-26.2466
15	150	30k	-27.3649	-26.2873
25	90	50k	-29.3113	-25.6773
50	45	100k	-32.0257	**-25.3918**

Tabell 4: Varierende antall generasjon med konstant total populasjon

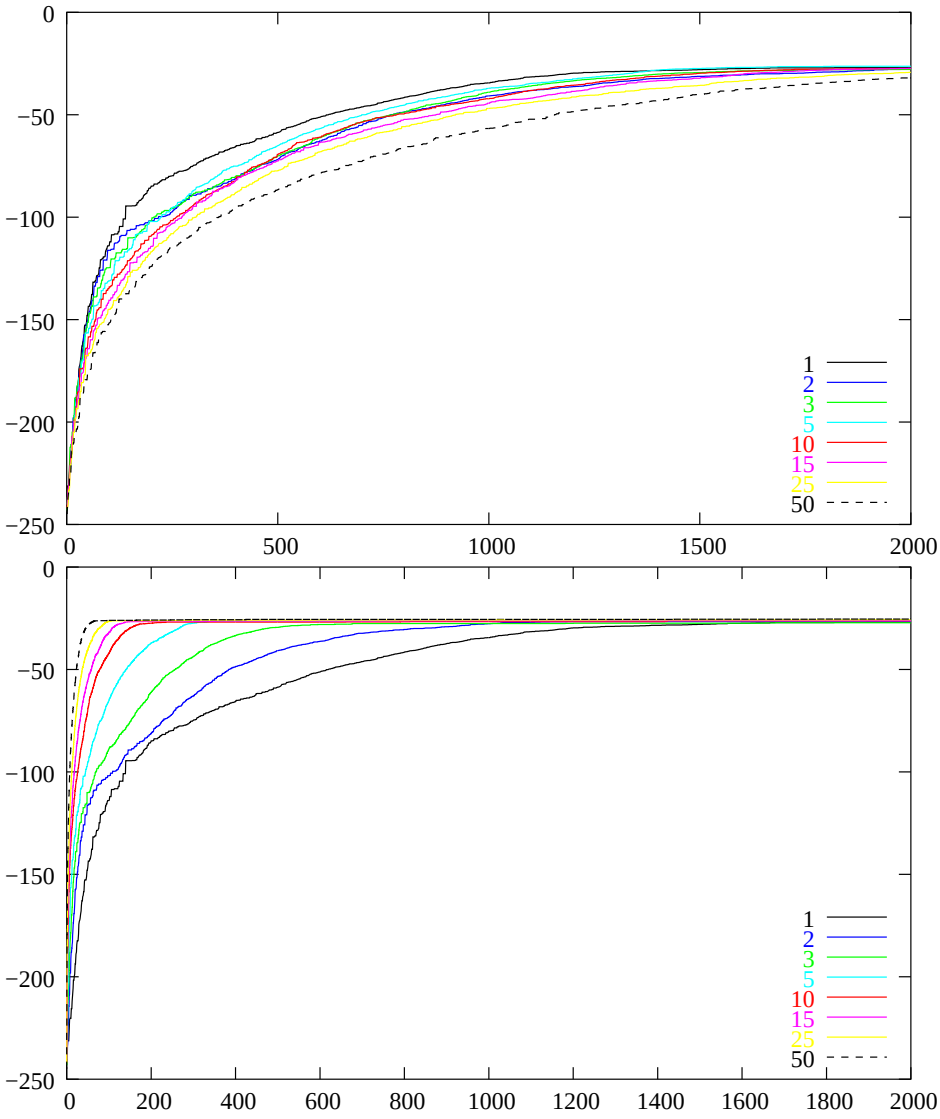
Resultatene viser at man får en bedre løsning ved flere generasjoner. Dette er ikke oppsiktsvekkende, siden det også brukes femti ganger mer prosessortid. Utviklingen er rimelig jevn hvis man ser i forhold til generasjoner, selv om en samlet populasjon har her en noe raskere utvikling. Hele området er rimelig jevnt. Sett fra et reelt tidsperspektiv utvikler en samlet populasjon seg mye tregere,og finner heller ikke de beste løsningene.

Figur 22 viser de ulike evolusjonene.

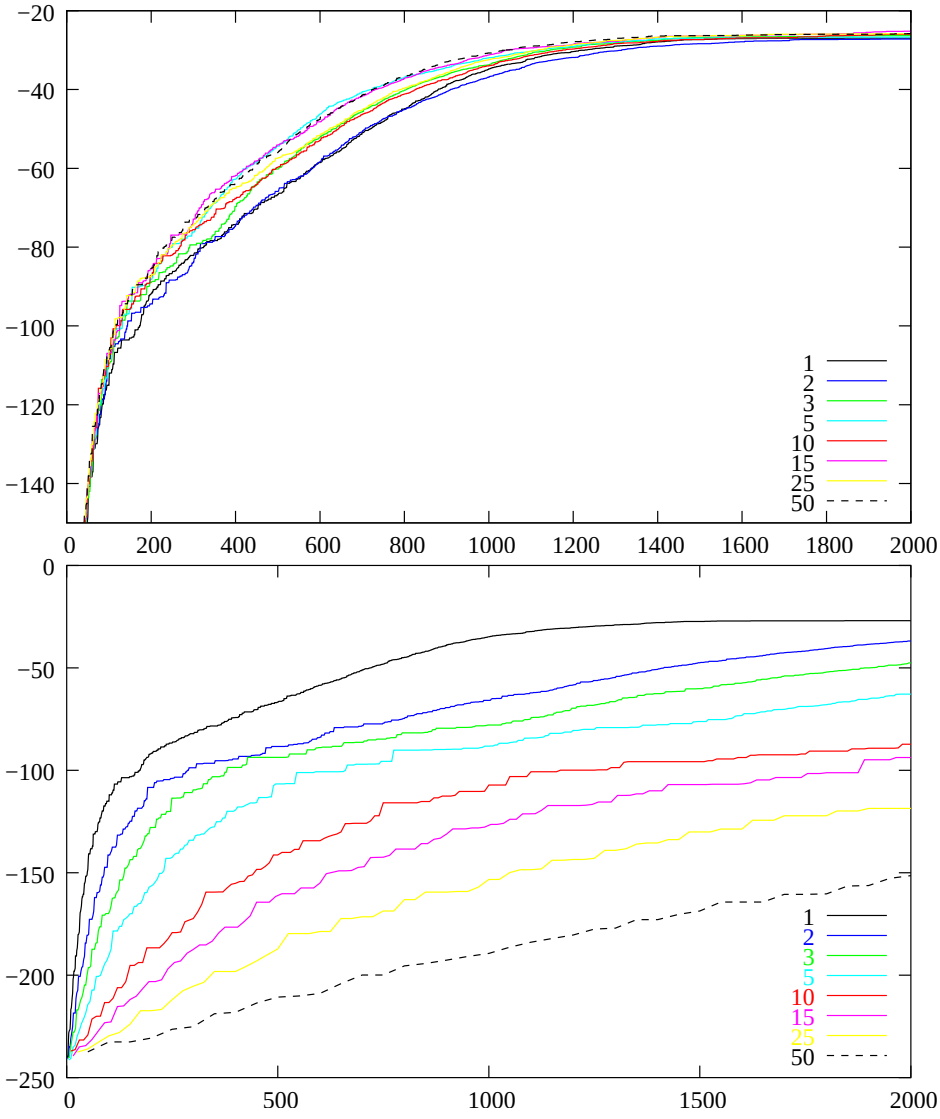
7.2.2 Konstant lokal populasjonstørrelse med konstant tidsskritt

I dette forsøket skal hver subpopulsjon ha 2250 individer og gå gjennom 2000 generasjoner. Hver node vil da jobbe like mye, uavhengig av hvor mange noder man har. Her vil altså også den totale prosessortiden være proporsjonal med antall noder. Oppsettet og resultatet er gjengitt i tabell 5.

Det viktigste resultatet fra dette forsøket er at en øking av maskiner på en forholdsvis optimal populasjonstørrelse ikke har noen større innvirkning. Dette sees greit i den øverste grafen i figur 23. Det er opplagt at systemet finner bedre resultater ved flere maskiner, men dette skyldes mest trolig økingen i prosessortid - og i søkerom - og ikke til så stor grad samspillet. Grafen øverst i figur 23 viser at forsøkene hvor det brukes mange noder i den distribuerte evolusjonen, fortsetter å finne gode løsninger i raskere grad etter å ha passert -100 i fitness enn forsøk hvor det kun brukes få noder.



Figur 22: Konstant lokal prosessortid. Ulike populasjonstørrelser og varierende antall generasjoner.
Øverst sees kun de første 2000 generasjonene.
Under beregnes førsteaksen tilsvarende antall generasjoner ved én node. Dette gir et reelt tidsperspektiv. Ved 50 noder går gjennom 100,000 generasjoner, men for sammenligningens del settes det opp at dette tilsvarer 2000 generasjoner, da 50-nodersvarianten bare har en populasjon på 45.



Figur 23: Konstant prosessortid på alle nodene. Like populasjonstørrelser. Øverst brukes generasjoner langs førsteaksen. Dette tilsvarer reel tid. Under brukes prosessortid langs førsteaksen. Det vil si at generasjonen ganges med antall noder i forsøket. Dette blir omvendt av figur 22. Her brytes det ned til CPU-tid, hvor den andre figuren brytes ned til reel tid.

Antall noder	Populasjon per node	Antall generasjoner	Beste sluttresultat
1	2250	2k	-26.951
2	2250	2k	-27.1833
3	2250	2k	-26.3475
5	2250	2k	-26.7117
10	2250	2k	-26.0321
15	2250	2k	**-25.1976**
25	2250	2k	-25.8035
50	2250	2k	-25.8258

Tabell 5: Varierende total populasjon som følge av varierende antall noder

7.3 Konklusjon

Det vil lønne seg å ha rimelig mange individer i hver node. Ved å dele populasjonen i veldig små biter har systemet større problemer med å finne gode løsninger. Grunnen til dette er at hver enkelt node ikke får noen fornuftig utvikling, samt at hver node tar i mot store deler av sin populasjon fra andre noder. Hver enkelt subpopulasjon får dermed liten mulighet til å differensiere seg fra de andre og effekten av å sende et godt individ inn i et nytt miljø går tapt. Evolusjonen drives i stor grad av utviklingen i hver enkelt node, hvor gode gener sakte distribueres ut til alle.

Det ser ikke ut til at systemet skalerer lineært, og man bør med andre ord bruke noe flere individer enn $\frac{totalt\ antall\ individer}{antall\ noder}$ per node for å for å finne løsninger tilsvarende det en samlet populasjon ville funnet.

8 Strukturforandringer

I dette kapittelet undersøkes hvordan ulike strukturer påvirker utviklingen. Jeg har i all hovedsak lagt vekt på tre ulike strukturer; RING layout, HUB layout og GRID layout. For RING og GRID er det aktuelt å se på både unidirektional og bidirektional lenking. Videre har RING også den ekstra muligheten av en dobbel foroverlenking. Forklaring på disse strukturene står i 1.4.3. Det brukes en tournament eksport med 80% sannsynlighet for å velge

det beste individet for alle kjøringer i dette kapittelet.

Jeg skal her først se hvordan en dobbel lenking virker inn på strukturene, så se de ulike strukturene opp mot hverandre.

Ved dobbel lenking blir eksporten av individer høyere enn ved enkel, siden det er eksporteringsparametene bestemmer hvor mange individer som skal eksporteres per vektor. Noder med mer enn en lenke vil da eksportere like mye som eksporteringsraten ganger antall ut-vektorer. Her er dette forsøkt stabilisert ved at modeller med enkel lenking eksporterer hver 5. generasjon og hvor modellene med dobbel lenking eksporterer hver tiende generasjon. Dette er ikke en perfekt måte å gjøre det på, det kan hende at antallet eksporterte individer bør forandres kontra å forandre frekvensen. Argumentet for å forandre frekvensen er at modeller med enkel lenking får en veldig lang rundetid.

8.1 RING

Her skal jeg se hva slags innvirkning en dobbel lenking har på en sirkelstruktur. Dobbel lenking vil si at individer migrerer både med og mot klokka i en sirkel. Enkel lenking eksporterer bare en vei. De to største forskjellene ved dobbel lenking er for det første at en node fort kan få tilbakemelding fra en annen node den akkurat har eksportert til, og for det andre at nodene i snitt ligger nærmere hverandre. I korte trekk fører dette til en større sammenkobling ved dobbel lenking, og vi kan forvente at denne har en noe raskere utvikling, på den andre siden er det en større mulighet for at evolusjonen stopper opp i et lokalt maksima.

I dette forsøket skal det også brukes enda en struktur, nær beslektet til en ring, nemlig en dobbel foroverlenkende ring. Det vil rett og slett si at en node på posisjon 'k' eksporterer til nodene på posisjon k+1 og k+2. Dette gjøres for gjøre "rundetiden", altså tiden det tar for gode resultater å komme seg rundt hele ringen, blir mye kortere.

Populasjon per node 500

Antall noder 16

Mutation 20%

Struktur RING single // double // double forward

Eksport 5, 10 og 20 individer hver femte og tiende generasjon.

Import Alle individer importeres jevnt fordelt utover hver eksportrunde.

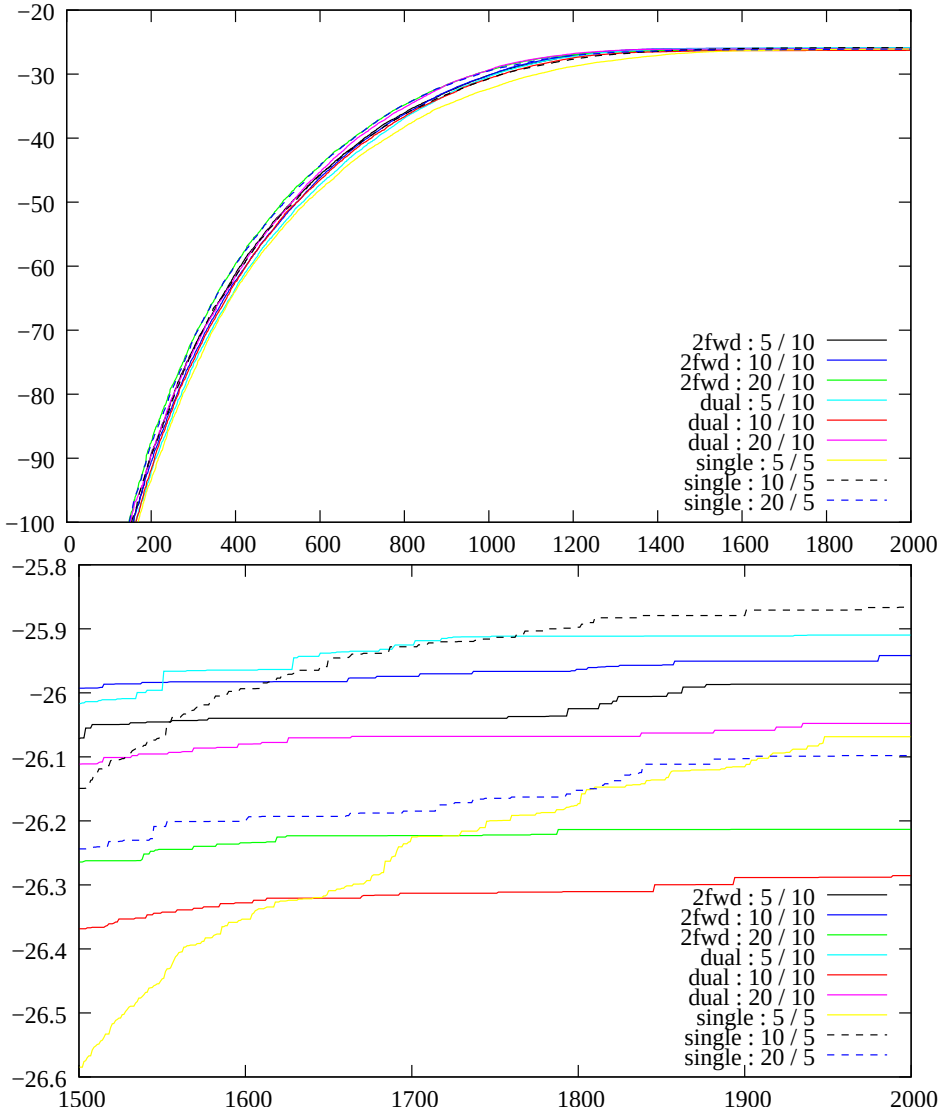
Vi har her med andre ord ni ulike kjøringer. Det framgår av figur 24 at alle mulighetene gjør det rimelig likt. Man finner konsistent løsninger med fitness rundt -26 $\pm$0.42. På den annen side finner vi det beste resultatet hittil (-25.023) i en dual ring med 5/10 i eksport. Mer informasjon om resultatene er gitt i tabell 6.

Figur 25 viser at informasjonen propagerer begge veier. Dette viser også at det er mest sannsynlig at en node som får informasjon fra den forrige beste noden, kommer til å gjøre det bra. Sagt på en annen måte, så er gode gener fra en annen node viktig for å kunne utvikle en god løsning. Dette gir en liten indikasjon på hvordan distribuering av evolusjon fungerer, altså at en migrasjon av gode gener gjør noe fornuftig - og at ikke nodene operer i et vakuum.

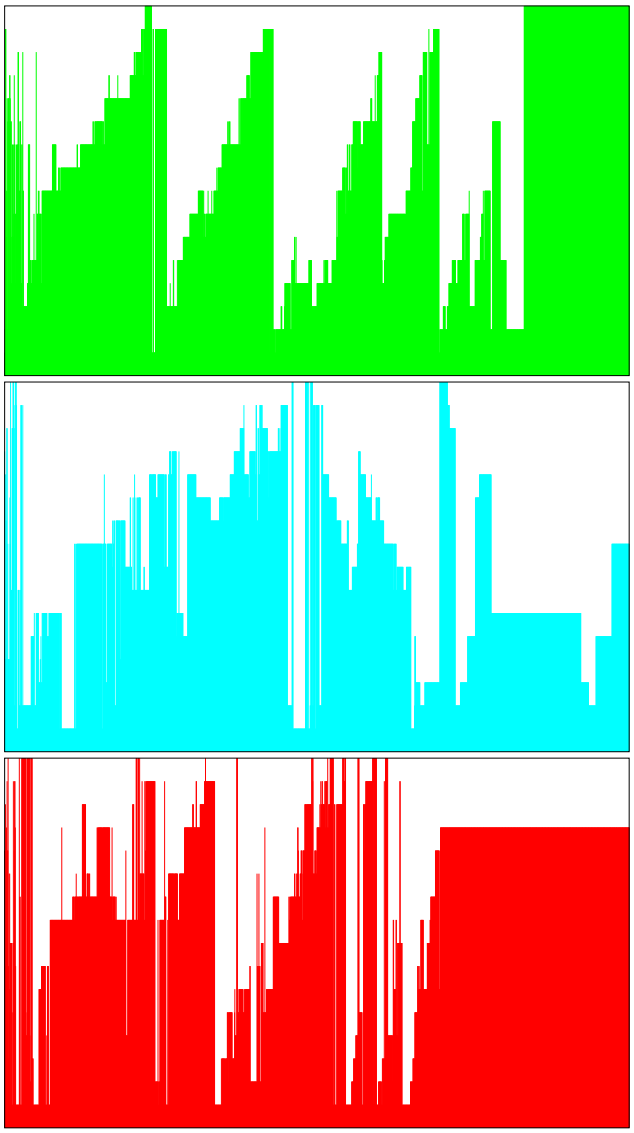
Vi ser av figur 25 at en god node som regel gir opphav til at en nabonode finner et enda bedre resultat. Dette er enklest å lese av den første grafen i figur 25, der man her ser markante diagonale linjer. Graf nummer to, med dobbel lenking, litt annen struktur grunnet lenkingen; Her flyttes det beste resultatet i begge retninger som man kan se at grafen flyter både opp og ned. Den siste grafen viser et eksempel på hva som skjer med en dobbel foroverlenking, her ser man en noe raskere (altså brattere diagonale linjer) forflytting av det beste resultatet.

8.2 GRID

Ved å lage en matrise av nodene får vi også en raskere distribusjon av nye gener. Det er svært kort mellom nodene i snitt. Ved enkel lenking er det her



Figur 24: Single mot dobbel lenking for RING-konfigurasjon



Figur 25: single link (Øverst), double link (Midten) og en dobbel foroverlink (Bunn)
I disse grafene er førsteaksen antall generasjoner som har gått (0-2000) og andreaksen sier hvilken node (1-16) som i det generasjonstrinnet har det beste individet. Det vi til dels kan se er at i single link (øverste graf) går det en bølge med det beste individet rundt sirkelen. Derimot ser vi ved double link (midterste graf) at denne effekten til dels kan gå begge veier. Til slutt ser vi ved en dobbel foroverlenke (nederste graf) at vi her til tider har en litt brattere stigning enn ved en enkel lenking.

bare fire noder før informasjon har gått rundt hele systemet. For en grafisk framstilling kan man se på figur 3.

Populasjon per node 500

Antall noder 16

Antall rader 4

Antall kolonner 4

Mutation 20%

Struktur GRID single // double

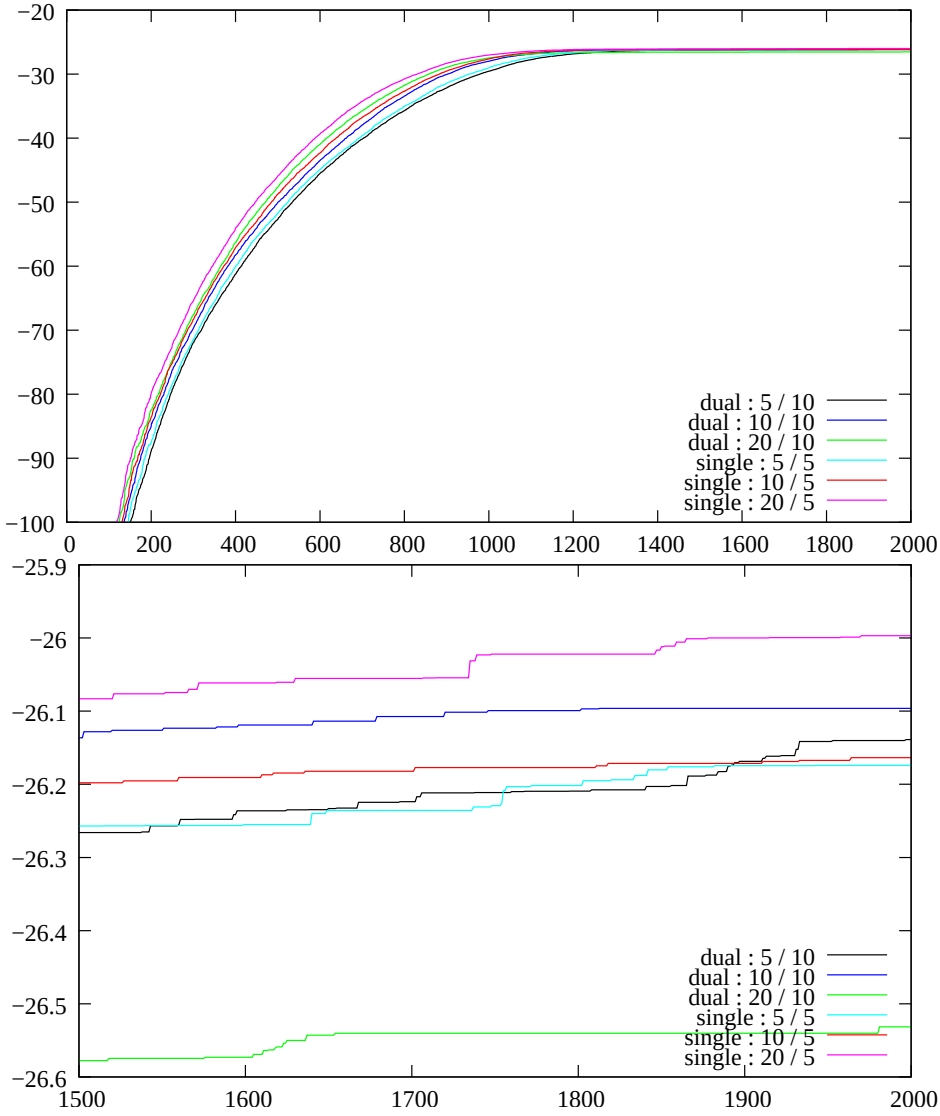
Eksport 5, 10 og 20 individer hver femte og tiende generasjon.

Import Alle individer importeres jevnt fordelt utover hver eksportrunde.

Resultatet av kjøringen kan ses i figur 26. Resultatet er nesten overraskende likt resultatet for det samme forsøket med RING konfigurasjon, selv om det virker som GRID finner et lokalt optima noe tidligere. Det ser ut som den bremser opp ved rundt 1200. generasjoner. Begge forsøkene finner tilnærmet like godt beste resultat.

Det er svært små forskjeller på resultatene som følge av ulike parametre når det gjelder GRID-strukturen, resultatene er gjengitt i tabell 6.

Å tegne et diagram for hvordan den beste noden flytter seg er noe vanskeligere å gjøre for en GRID-struktur enn for en RING. Dette har sammenheng med at koblingen fra hver node kan flytte seg i to dimensjoner og ikke bare en. Følgelig blir det vrient å visualisere hvordan den beste noden flytter seg. Det blir derfor ikke laget noen figur som viser hvordan den beste flytter seg for dette forsøket.



Figur 26: Single mot dobbel lenking for GRID-konfigurasjon

8.3 HUB

Det ble også gjort forsøk med en nav-struktur, med en enkelt senternode som tar seg av kommunikasjonen. Vi har tidligere sett at det beste individet ofte følger en bølge bortover en sirkelstruktur. Det kan virke som de andre nodene er noe overflødige. Dette fremstår likevel ikke som sannsynlig siden den beste løsningen fortsetter å flytte på seg (se figur 19 og figur 25). Hver node har utvikling i sin retning, og får i tillegg innsprøyting av gode noder når turen har gått rundt hele sirkelen.

For HUBstruktur blir det meningsløst å variere med enkel og dobbel lenking da en enkel lenking ikke gir videre mening. Kun eksport eller kun import fra senternoden vil gi lite mening. For at nodene skal kunne kommunisere må man ha en dobbel - eller bidirektional lenking. Derimot kan det finnes andre variasjoner.

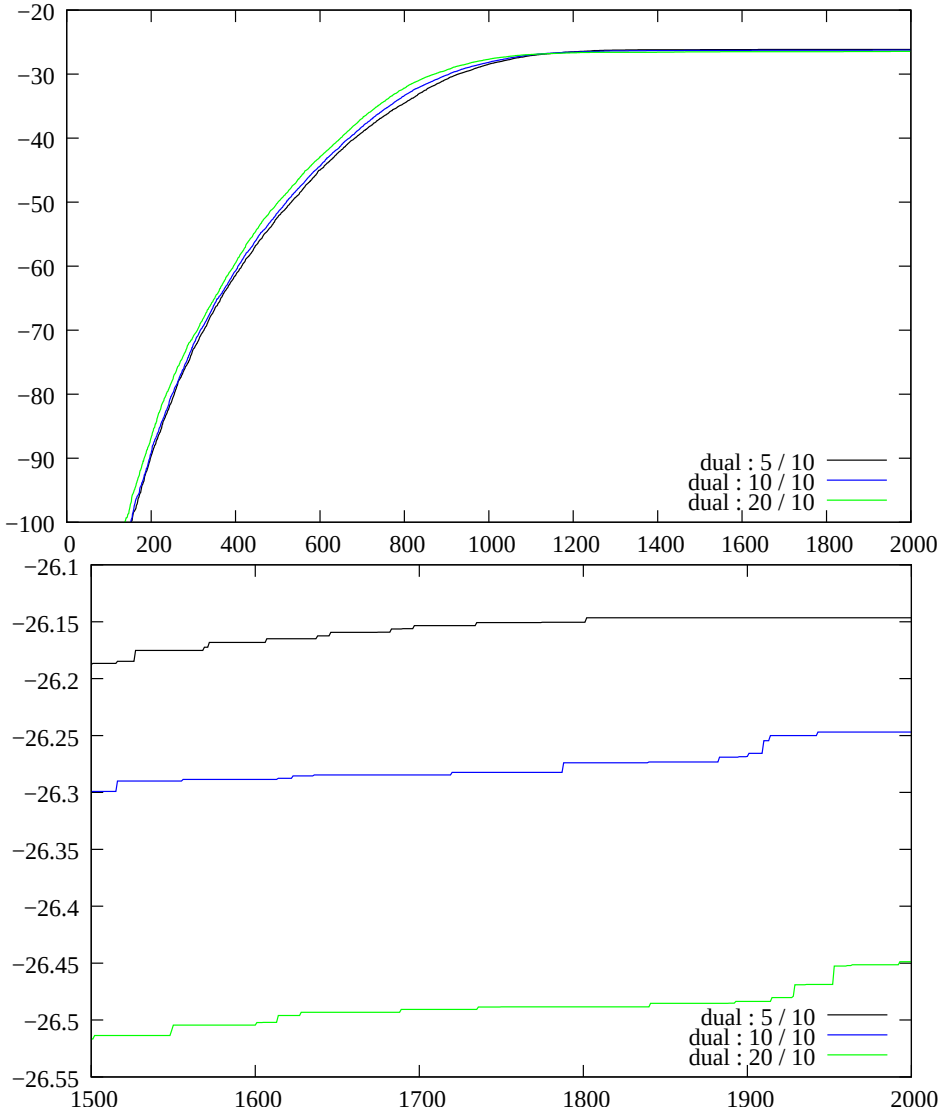
Fra figur 27 virker det som en høyere eksporteringsrate virker positivt inn, men man kan ikke se at en høyere mutasjon bidro slik en kunne håpe, nemlig ved å avverge en stagnering.

8.4 Sammenligning

For å gjøre en litt fornuftig sammenligning av de ulike strukturene har jeg kjørt forsøk med subpopulasjonstørrelsen 500. I alle tilfellene har jeg brukt 16 noder. Eksportraten er her satt noe høyere enn vanlig - hver 5. generasjon, og eksportraten varierer mellom 5, 10 og 20 individer per denne frekvensen.

Det er også gjort ti kjøringer for hvert oppsett og det beste resultatet er plukket ut. Resultatene er gjengitt i tabell 6. Det viktigste resultatet av sammenligningen er at forskjellene er svært små. Det dårligste beste resultatet (altså, den kjøringingen som til slutt fant det dårligste resultatet) er bare noe over to fitnesspoeng dårligere enn det beste som ble funnet. Standardavviket er under et halvt poeng.

Å si noe særlig om forskjellene på de ulike strukturene er litt meningsløst, siden forskjellene er så små. Det eneste fornuftige vil være å si at det tenderer



Figur 27: dobbel lenking for HUB-konfigurasjon

mot at få eksporterte individer lønner seg i snitt. Dette kan ses i sammenheng med at de ulike subpopulasjonene får mest mulighet til å utvikle seg i sin egen retning.

8.5 Andre mulige strukturer

I dette kapittelet har jeg sett på de enkleste strukturene; GRID, RING og HUB. Disse tre bør ha et rimelig godt spenn for de ulike egenskapene en struktur kan ha, men det finnes mange andre mulige måter å la individer vandre fra en node til en annen.

8.5.1 Kombinasjonstrukturer

De ulike strukturene kan kombineres med hverandre. Det mest nærliggende er en blandet HUB og RING struktur. Tanken vil her være å utnytte et høyt mutasjonssenter med en vanlig ring-struktur for overføring av vanlig data. Ring-strukturen har vist seg mest effektiv, men også den kan henge seg opp. Høymutasjonstesten gav i seg selv ikke så godt resultat, men dette kan skyldes at de originale genene ikke fikk så stor mulighet til å utvikle seg. Her benytter man seg av strukturen til å utføre noe som ikke enkelt kan gjøres i en samlet populasjon.

8.5.2 Meta-random mutasjon i gridmode

Dette vil være en variant av høymutasjonsideen. Hver node i en GRID kan ha litt ulik mutasjonsgrad, slik at man både får en fornuftig overføring av data mellom utviklende noder og en tilførsel av ny informasjon grunnet mutasjon. Denne vil ha store likhetstrekk til en kombinasjonstruktur med RING+HUB.

8.5.3 Random linkage.

En annen mulighet er å basere lenkingen på en gjennomsnittlig kardinalitet av nodene, og muligens en gitt avstand mellom nodene, for deretter å koble

	max	min	mean	stddev
All	-25.023	-27.782	-26.131	0.446
All rings	-25.023	-27.181	-26.046	0.420
Single ring	-25.433	-26.843	**-26.011**	0.382
Dual ring	**-25.023**	-26.963	-26.081	0.469
2fwd ring	-25.344	-27.181	-26.047	0.415
5 ring	**-25.023**	-26.963	**-25.988**	0.470
10 ring	-25.439	-26.777	-26.031	0.322
20 ring	-25.309	-27.181	-26.120	0.456
All grids	-25.214	-27.782	-26.184	0.468
single grid	**-25.214**	-27.000	**-26.111**	0.433
dual grid	-25.369	-27.782	-26.256	0.498
5 grid	-25.435	-27.000	-26.156	0.400
10 grid	-25.369	-26.895	**-26.130**	0.438
20 grid	**-25.214**	-27.782	-26.264	0.565
All hubs	-25.447	-27.311	-26.281	0.433
5 hub	-25.813	-26.803	**-26.147**	0.316
10 hub	**-25.447**	-27.311	-26.247	0.558
20 hub	-25.951	-26.955	-26.449	0.375

Tabell 6: Litt statistikk om de ulike strukturene.
Alt er målt i forhold til de beste resultatene som ble funnet for hver kjøring. De beste resultatene står med uthevet skrift, men forskjellene på de ulike metodene er for små til å være signifikante.
Denne oversikten viser de ulike funnene for ulike strukturer. Den er delt opp først og fremst i de tre ulike mulighetene RING, GRID og HUB og en samling ALL.
All viser statistikk for alle muligheter.
All rings viser statistikk for alle ringene. Så fordeles ringer i ulike grupper. Enkel, dobbel og to-forover lenking og samtidig 5, 10 og 20 eksport hver tiende generasjoner. Disse ulike gruppene overlappes. Den doble lenkingen vil inneholde 5, 10 og 20 i eksportverdier.
Ringer deles opp på samme måte, altså med enkel og dobbel lenking, samt 5, 10 og 20 eksport hver tiende generasjon.
Hub har ingen muligheter for lenkevariasjon, men eksportvariasjonene skjer.

sammen nodene på en stort sett tilfeldig måte, men som likevel tilfredsstiller disse kravene. Det enkleste ville være å gi hver node muligheten til å koble seg til [m,n] andre noder og velge tilfeldig blant dem. Det bør da være en test på at alle nodene er koblet sammen - og at alle har både en inn og en ut vektor. Dette bør ikke være noe videre vanskelig å implementere.

8.5.4 Fully linked

Det er også et alternativ å lage en modell hvor alle nodene er forbundet med hverandre. Etter forsøkene som er omtalt i denne oppgaven, virker ikke dette som en særlig god idé. Det ser ikke ut som en ekstrem kommunikasjon gir noe videre fortrinn. Det kan tilnærme seg en samlet populasjon, men man får ikke muligheten til å utnytte at man opererer med et distribuert system.

9 Konklusjon

Selv uten en super-lineær skalering, er det fornuftig å distribuere en evolusjon. En evolusjon kan deles opp slik at den kan kjøres på mange maskiner og på denne måten spare mye reell tid. Man kan ikke nødvendigvis forvente en lineær skalering i forhold til en enkelt populasjon. Ofte kan distribueringen føre til en del duplisert arbeid, gjerne ved at enkelte subpopulasjoner ikke får mulighet til å utvikle seg i sin egen retning på grunn av for stor innførsel av gener. Subpopulasjonstørrelsen ser ut til å ha en betydelig innvirkning. Det vil si at man bør ha store nok subpopulasjoner til å kunne utføre en fornuftig lokal evolusjon. For eksporteringsgrader finnes det to ekstremer. En høy migrasjonsrate nærmer seg en enkelt populasjon, men mister mulighetene en lav eksporteringsgrad kan ha til å utnytte den algoritmiske fordelen ved distribueringen.

Det er flere ganger blitt rapportert at distribuering av en evolusjon kan ha en super-skalar effekt, men dette kom ikke tydelig fram i denne oppgaven. Dette kan skyldes flere faktorer, en av dem er at eksporteringsfrekvensen kan

ha vært for høy. Ved en for høy eksporteringsfrekvens vil ikke de ulike subpopulasjonene ha muligheten til å gå ulike veier, og effekten av *punctuated equilibria* forekommer ikke i så stor grad som ønskelig. Når denne effekten inntrer, ser man at den beste løsningen vil forflytte seg bortover subpopulasjonsrekkene, (se f.eks figur 19). At denne effekten forekommer ved stort sett alle forsøkene, er trolig det mest interessante resultatet i denne oppgaven.

Det kan også hende at TSP ikke er et problem hvor en super-skalering kan finne sted. TSP kan også ha problemer med at det ikke lar seg spre ut til et genetisk mangfold. En god kant i TSP vil alltid være en god kant, derfor kan man fort få problemer med å danne ulike "raser". Uten ulike retninger som problemet kan utvikle seg i, vil ikke distribueringen kunne tilføre en algoritmisk fordel, selv om den reelle tiden drastisk reduseres av en parallellisering. Større figurer uttrykt av genotypen kan imidlertid lage ulike typer løsninger som kan hjelpe evolusjonen.

Til slutt er det sannsynlig at størrelsen på subpopulasjonene har tydelig innvirkning på hvor gode resultater evolusjonen produserer. Hvis subpopulasjonstørrelsen kommer under en viss grense hvor den enkelte node ikke lenger kan utføre en fornuftige selvstendig evolusjon, synker ytelsen betraktelig. Det er gjort forsøkt med små subpopulasjoner tidligere[14], men der med distribuerte seleksjonsmekanismer som motvirket de små subpopulasjonene. Dette er gjort på en slik måte at de små subpopulasjonene i realiteten overlapper hverandre og framstår som en større subpopulasjon med tilstrekkelig mange individer til å kunne utføre en evolusjon.

Videre har det vist seg at selve typen eksporteringsmekanisme ikke nødvendigvis har så stor innvirkning. Som tidligere nevnt er dette et litt usikkert resultat, fordi det ikke er testet med større antall subpopulasjoner, men det støttes indirekte av annen litteratur. Svært ulike mekanismer har alle gitt den samme effekten. Siden det er brukt et større utvalg eksporteringsmekanismer og helt andre arkitekturer for utveksling av genetisk informasjon, som også har gitt gode resultater, ser det ut til at det er selve oppdelingen i flere grupper som er det virksomme elementet i en distribuert evolusjon. Skal det først velges en eksporteringsseleksjon, ser det ut til at tournament virker fornuftig,

siden den er et generelt tilfelle av både elitism og random, samtidig som den er veldig effektiv.

I tillegg til at en distribuering kan gi et algoritmisk fortrinn, gir også fordelingen av subpopulasjoner ut til mange prosessorer et stort fortrinn i reell tid. En mengde "mindre" maskiner gir en vesentlig ytelse-/prisfordel i forhold til sentrale systemer. I tillegg er det mulig å bygge opp samlede systemer (*cluster*) som går langt over det som eksisterer av større *single-processor* systemer. Det bør også nevnes at det er fullt mulig å la hver node være en vanlig arbeidstasjon, hvor evolusjonen kun trenger å gå som en bakgrunnsprosess. Det er med andre ord fullt mulig å utnytte ressurser som allerede eksisterer, framfor å sette opp et dedikert system.

9.1 Parallellisering

Hvis resultatene av en evolusjon er det viktige, vil en distribuering raskere gi resultater. Den evolusjonære prosessen lar seg forholdsvis greit distribuere, og er rimelig parallell i sin natur. Selv en generell lettvektsdistribuering, som gjennomført i denne oppgaven, vil føre til en betraktelig raskere evolusjon enn om den skulle utføres på noe tilsvarende en enkelt node. Ved et distribuert forsøk kan det totale antallet evalueringer, kombinasjoner og mutasjoner være det samme som ved en normal ikke-distribuert genetisk algoritme, gitt tilsvarende antall generasjoner. Samtidig kan denne prosessen utføres mange ganger så raskt fordi man har flere prosessorer som opererer i parallell.

En distribuert evolusjon vil ikke kunne gå saktere enn en ikke-distribuert metode. Den nedre grensen er med andre orddet samme som ytelsen til en av de enkelte nodene. Evolusjonen kan utføres på n ulike noder, hvor ingen har kommunikasjon med noen av de andre. Dette kan, i det minste som et tankeeksperiment, tilføres tilstrekkelig kommunikasjon til å utføre nøyaktig den samme evolusjonen som et ikke-distribuert system. Her må det forutsettes en fully-linked struktur, og noe ekstra overhead knyttet til å plukke individer fra ulike subpopulasjoner for kombinering. En øvre grense for hvor

raskt en distribuering kan utføre en evolusjon vil altså være n ganger raskere enn en enkel evolusjon, hvor n er antallet noder. Her ser vi bort fra effekten av *punctuated equilibria*, som er nevnt tidligere. Et distribuert system bør altså operere et sted mellom like fort og n ganger fortere, hvis distribueringen i seg selv ikke utnyttes til noe.

På den annen side ser det ut til at det finnes grenser for hvor mange noder evolusjonen kan fordeles ut på. Kapittel 7.1 viser at å dele opp denne oppgavens eksempelproblem (256-punkters TSP - 2250 individer - 2000 generasjoner) i 50 biter fungerte vesentlig dårligere enn mindre oppdelinger. Her har vi altså truffet en grense for akkurat dette problemet. Grensen har antageligvis oppstått enten på grunn av kommunikasjonen, ved en problematisk struktur, ved for lite kommunikasjon eller ved at importerte individer dominerer populasjonen. Domineringen har sammenheng med den andre trolige grunnen til degradert ytelse, nemlig at subpopulasjonen på $\frac{2250}{50} = 45$ individer mest trolig er for liten til selv å utføre en fornuftig evolusjon. Et svært oppdelt evolusjonært system vil etter all sannsynlighet fremdeles spare inn mye reell tid, men faller godt under det lineært skalerbare systemet.

En parallellisering er ikke nødvendigvis en måte å løse svært store problemer på, men en distribuering vil føre til en raskere utføring. En distribuering er også rimelig enkel å implementere og bør forholdsvis enkelt kunne tilpasses en mengde problemer. Med andre ord er en distribuering en praktisk og forholdsvis pålitelig metode for å øke ytelsen til en evolusjon.

9.2 Problemer / videre arbeid

Siden det alt er kjent at en (samlet) populasjonstørrelse på 1600-3200 er optimalt for å løse TSP med 256 punkter[1], kan det hende at en distribuering ikke kan tilføre noe ekstra. Etter 2000 generasjoner finner en samlet populasjon konsistent gode resultater, noe den distribuerte modellen også gjør. Mest trolig burde systemet ha spesialisert seg etter tilfeller hvor man ikke har tilstrekkelige ressurser til rådighet. Den enkleste måten å gjøre dette på vil være å f.eks. halvere antallet totale evalueringer som kan utføres. Altså ved

å forsøke en distribuert modell for et problem som en seriell modell allerede løser tilfredsstillende, vil det være vanskelig å gjøre store forbedringer.

På grunn av at jeg ikke i utgangspunktet fant noen superskalar effekt, kan jeg ha mistet litt av fokuset på dette målet. Mye av oppgaven går heller ut på å tilnærme seg en lineær skalering. Min tanke er at den linære skaleringen kanskje kan skyldes noe for høy kommunikasjonsrate. Imidlertid virker det tidvis som om en høyere kommunikasjon tilnærmer seg en samlet populasjon. Hele systemet oppfører seg da som et *deceptive problem*. Framfor å øke kommunikasjonen, burde noder isoleres mer. Da vil *punctuated equilibria*-effekten kunne bli bedre utnyttet. Lengre isolasjonstider kan gjøre effekten av denne større.

En måte å implementere et mer isolert system på, er å vente med import av individer til populasjonen har nådd et gitt platå, for eksempel ved å sette som kriterium at ingen individer importeres før subpopulasjonen har hatt det samme beste individet i n generasjoner. Først da kan individer importeres. Her kan importkøen også ha variasjoner. Den kan hindre import av individer, enten ved å holde individer fram til import, eller ved å skulle ha plass til et fast antall individer, slik at individer som står for lenge i kø, forsvinner ut i det store intet.

Som følge av eksporteringstrategien som brukes, kan ofte det samme individet eksporteres mange ganger. Dette bør det antageligvis demmes opp for, enten på eksport- eller importsiden, slik at det ikke innføres for mye av den samme genetiske informasjonen. Ved innførsel av mye likt materiale kan det hende man synkroniserer populasjonene for mye til at *punctuated equilibria*-effekten kommer tilstrekkelig i gang. De mest åpenbare alternativene er å fjerne alle like individer i importkøen eller ikke importere individer som allerede eksisterer i populasjonen. Den siste løsningen er mer kompleks og tar mest kjøretid.

Et problem som ikke tas opp i denne oppgaven er serialisering av individer. En serialisering er nødvendig for å overføre et individ fra en maskin til en annen. For TSP er dette problemet trivielt, hvert individ kan uttrykkes som en streng

med tall.[11] For mer komplekse problemer, som f.eks et genetisk programmeringsproblem med automatisk definerte funksjoner, kan serialiseringen by på mange problemer. Først og fremst fordi det eksisterer informasjon utenfor det enkelte individet som er nødvendig for at individet skal fungere. Med andre ord trengs det enda en synkroniseringsmekanisme for disse utenfor-individet egenskapene. Faktorer som disse er vanskeligere å gjøre til generelle systemer, og må eksplisitt programmeres inn i program for at slike problemer skal kunne løses.

[11] For 256 noder holder det med en char for hvert punkt, og network og host byte-order er uvesentlig.

Referanser

[1] Jarmo T. Alander. On optimal populationsize of genetic algorithms. In Patrick Dewilde and Joos Vandewalle, editors, CompEuro 1992 Proceedings, Computer Systems and SoftwareEngineering, 6th Annual European Computer Conference, pages 65-70, The Hague, 4.-8. May 1992. IEEE Computer Society, IEEE Computer Society Press.5, 11, 12

[2] Whitley, D., Starkweather, T., Shaner, D.: The Travelling Salesman and Sequence Scheduling: Quality Solutions using Genetic Edge Recombination. In Davis, L., (Ed.): Handbook of Genetic Algorithms. 350-372, Van Nostrand Reinhold, 1991.

[3] S. Forrest and M. Mitchell (1993). Relative building-block fitness and the buildingblock hypothesis. In D. Whitley (ed.), Foundations of Genetic Algorithms 2, 109– 126. San Mateo, CA: Morgan Kaufmann.

[4] Holland, J. H. 1975. Adaption in Natural and Artificial Systems. University of Michigan Press. (Second edition: MIT Press, 1992)

[5] Mitchell, M. (1996) An introduction to genetic algorithms. MIT Press.

[6] http://www.math.princeton.edu/tsp/history.html

[7] V. S. Gordon and D. Whitley (1993), "Serial and Paralell Genetic Algorithms as Function Optimizers", Proc. 5th ICGA, Morgan Kaufmann, pp. 177-183

[8] S. C. Lin, W. Punch and E. Goodman (1994), "Coarse-Grain Paralell Genetic Algorrithms: Categorization and New Approach", Proc 6th IEEE Symposium on Parallel and Distributed Processing, pp. 28-37

[9] E. Cantu-Paz (1998), "A survey of Parallell Genetic Algorithms", Calcuateurs Paralleles, Reseaux et Systems Repartis, Vol.10, No.2, pp.141-171

[10] Garey, M. R. and Johnson, D. S. (1979). Computers and Intractability: a Guide to the Theory of NP-Completeness. W.H. Freeman, New York.

[11] Wright, Sewall. (1943). Genetics 28. p 114.

[12] D. Andre and J. R. Koza (1997) “A parallel implementation of genetic programming that achieves super-linear performance”.

[13] Heinz M¨uhlenbein (1991) “Evolution in Time and Space - The Parallel Genetic Algortihm”

[14] Shumeet Baluja (1992) “A Massively Distributed Parallel Genetic Algorithm".

[15] Darrell Whitley, Soraya Rana, Robert B. Heckendorn "The island model genetic algorithm: on separability, population size and convergence".

[16] Enrique Alba, Carlos Cotta og Jose M. Troya (1999) “Entropic and real-time analysis of the search with panmictic, structured and parall el distribueted genetic algorithms”.

[17] Enrique Alba, Carlos Cotta og Jose M. Troya (2000?) “Numerical and real time analysis of parallel distributed GAs with structured and panmictic populations".

[18] David Andre og John R. Koza (97) “A parallel impelemetation of genetic programming that achieves super linear performance”.

[19] Heinz Muhlenbein (1992) "Parallell genetic algorithm in combinatorial optimization"

[20] Heinz Muhlenbein (1996) "Asynchronous parallell search by the PGA"

A Ordliste

cluster En samling maskiner som gjerne jobber med samme oppgave. I dette tilfellet vil det være en samling maskiner som alle jobber med samme evolusjon.

coarse-grained Parallelle genetiske algoritmer kan grovt deles opp etter hvor mye kommunikasjon som eksisterer mellom de ulike subpopulasjonene. Hvis det er lite kommunikasjon i forhold til hvor mye annet som utføres i en subpopulasjon, sies det at systemet som helhet er *coarse-grained*. Det vil være typisk for *coarse-grained* systemer å ha store subpopulasjoner.

combine Kombinasjonen mellom to individer som skaper ett nytt individ. Kalles tradisjonelt *crossover*, men i mange tilfeller er *crossover* misvisende.

crossover Et spesialtilfelle av *combine*, hvor nye individer konstrueres ved å "krysse" mellom to opphavsindivider.

deceptive problem Et misvisende problem. Gjerne hvor den beste løsningen ligger helt motsatt i forhold til en gradient. Hvis et problem er å finne fram til et tall 0-10, kan tallene 1-9 ha seg selv som fitness, men tallet 0 kan ha en fitness på 10. Det vil her være vanskelig å bruke *hillclimbing*metoder for å finne det beste resultatet.

directed graph Rettet graf. En graf hvor retningen på kantene har betydning.

elitism Seleksjonsmekanisme for evolusjonære systemer. Betegner mekanismen som alltid lar de beste n individene går videre til neste generasjon.

evaluation Evaluering. Steget som tas for å finne *fitness* for et individ.

fine-grained Parallelle genetiske algoritmer kan grovt deles opp etter hvor mye kommunikasjon som eksisterer mellom de ulike subpopulasjonene. Hvis det er mye kommunikasjon i forhold til hvor mye annet som utføres

i en subpopulasjon, sies det at systemet som helhet er *fine-grained*. Det vil være typisk for *fine-grained* systemer å ha små subpopulasjoner, gjerne 10 individer eller mindre.

fitness Et tall som uttrykker hvor godt en individ klarer seg i sitt miljø. Dette er gjerne en refleksjon av hvor godt individet løser problemet som evolusjonen angriper.

fully linked En graf hvor alle nodene deler en kant med alle de andre nodene i grafen.

GA Genetisk algoritme. Brukes både som samlebegrep for evolusjonære systemer og en subgruppe av dette.

genotype Genmatrialet til et individ. I denne oppgaven er det ingen forskjell på *genotype* og *phenotype*.

hillclimbing En lokal problemløsningstrategi. Denne foregår ved å betrakte den nærmeste punktene rundt løsningen og velge det som gir best forbedring. Løsningen vil da klatre sakte høyere i *fitness*-landskapet.

hub Nav.

mutation probability Sjansen for at det forgår en mutasjon.

phenotype En genotype uttrykt som et individ. I denne oppgaven er det ingen forskjell på *genotype* og *phenotype*.

population Populasjon. En samling individer.

premature convergence Fenomenet som oppstår hvis en evolusjon har spesialisert seg på et lokalt maksimum.

punctuated equilibria En effekt som skjer ved når et individ treffer et nytt miljø. Effekten har fått navnet fordi en subpopulasjon (miljø) her en tendens til å stagnere på et plåta, men dette platået kan brytes ved tilførsel av nye gode gener.

random access Tilfeldig tilgang. Dette vil si at man kan få tilgang til hvilket som helst element i en samling.

roulette wheel Seleksjonsteknikk for evolusjonære systemer. Beskriver hvordan hvert individ har muligheten til å bli valgt proporsjonalt med sin fitness i forhold til alle de andre individene. Implementeres ved at man lager et lykkehjul hvor hvert individ har en viss buelengde skalert etter sin fitness.

selection Seleksjon. Utvelgingsmekanisme.

stochatic universal sampling En spesialutgave av *roulette wheel*, hvor lykkehjulet bare spinnes en gang. Konseptuelt kan man tenke på dette som at det ikke bare er en nål som peker på hjulet, men mange.

stream En strøm. Man har her bare tilgang til det første elementet.

tournament En seleksjonsteknikk for evolusjonære systemer. Det velges ut to individer og det beste individet av disse har en gitt sannsynlighet for å bli valgt videre.

transfer Overgangen fra en generasjon til en annen. Individer må på en eller annen måte forflytte seg fra en generasjon til en annen.