Handelsresandeproblemet

Daniel Lindblad 1901805
Kandidatavhandling I Datateknik
Handledare: Jan Westerholm
Fakulteten för naturvetenskap och teknik
Åbo Akademi
2022

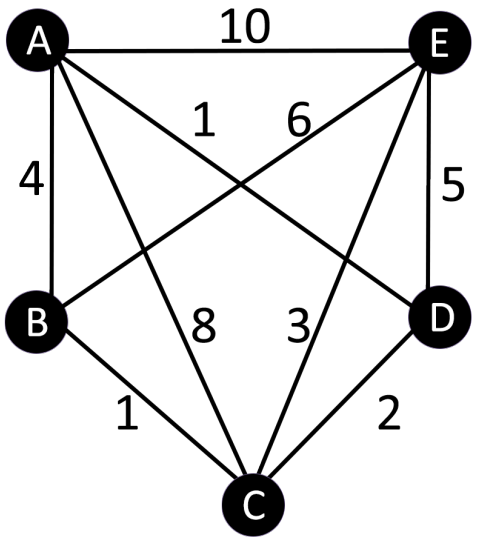
Abstrakt

Innehåll

1 Introduktion

Handelsresandeproblemet (*Travelling Salesman Problem*) är ett av de mest kända och mest undersökta problemen inom kombinatorisk optimering. Problemet kan definieras på följande sätt, Givet en lista av städer och avstånden mellan varje stadpar, vilken är den kortaste möjliga rutten som besöker varje stad exakt en gång och återvänder till ursprungsstaden? En sådan rutt brukar kallas en turné. Den kortaste möjliga turnén sägs vara optimal.

Handelsresandeproblemet kan uttryckas som en oriktad viktad graf: där städerna representeras av grafens noder, rutterna mellan städerna representeras av bågarna i grafen och distansen mellan städerna av vikten av bågarna. I fall där det inte finns en båge mellan två noder i grafen så kan man tillsätta ett stort värde för att göra grafen komplett, d.v.s. varje par av distinkta noder har en båge mellan sig.



Figur 1: En illustration av en viktad graf

Handelresandeproblemet studerades på 1800-talet av irländska matematikern Sir William Rowan Hamilton. Hamilton uppfann ett spel som blev känt som ”The Traveller’s Dodecahedron” *(den resandes dodekaeder)* eller ”A Voyage Around the World” *(En resa världen runt)*. Spelet innehöll en dodekaeder där varje nod representerade en plats (Bryssel, Canton, Delhi o.s.v) och var markerad med en stav. Till spelet hörde till en tråd som kunde trädas runt en stav och på så sätt kunde man bilda en rutt. Målet i spelet var att bilda en Hamiltoncykel, d.v.s. en rutt som besöker varje nod en gång och anländer tillbaka till startnoden. I spelet kallades detta för en *voyage round the world*[3].

Det egentliga optimeringsproblemet innebär att hitta den turné som har minsta möjliga

totala längden. För att uppnå detta ändamål så är det fördelaktig att ange avstånden mellan alla möjliga par av städer i en matris. Förutom de exakta geometriska avståndsvärdena kan andra värden användas beroende på den exakta formuleringen av problemet, t.ex. restid eller kostnaden för den nödvändiga mängden bränsle. Målet är att planera resan på ett sådant sätt att det totala avståndet, restiden eller den totala kostnaden minimeras. För ett n-antal städer och en given ursprungsstad blir det totala mängden möjliga rutter som uppfyller kraven för handelresandeproblemet $\frac{(n-1)!}{2}$ Detta innebär att en naiv algoritm som försöker hitta den optimala turnén genom att räkna alla giltiga turnéer har tidskomplexitet $\mathcal{O}(n!)$.

Handelresandeproblemet tillhör en klass av utmanande problem: de så kallade NP-hårda problemen[2, s.61-63]. Det finns inga effektiva algoritmer för denna klass av problem. Det antas att varje algoritm som försöker hitta en exakt lösning kräver exponentiellt många steg. Det finns dock inget matematiskt bevis för detta antagande. Effektiva algoritmer finns för speciella varianter av handelresandeproblemet och för beräkning av ungefärliga lösningar, som tillåter turnéer vars totala kostnad kan vara större än den optimala turnén. Handelresandeproblemet förekommer som delproblem i en mångfald praktiska optimeringsproblem, t.ex. optimering av framställning av kretskort och optimering av paketleveransvägar.[9]
Handelresandeproblemet kan vidare delas in två distinkta varianter: det symmetriska handelresandeproblemet *(symmetric travelling salesman problem)* (sTSP) och det asymmetriska handelresandeproblemet *(asymmetric travelling salesman problem)* (aTSP).

För sTSP gäller det att för varje rutt mellan två städer är distansen den samma oberoende av vilken riktning man använder. Detta kan uttryckas på följande sätt. Låt $V = \{s_1, s_2, s_3 ...\}$ vara mängden av städer, $A = \{(r,s) : r,s \in V\}$ vara mängden av bågar i V, och där $d_{rs} = d_{sr}$ vara distansen för bågen $(r,s) \in A$.

En TSP är aTSP ifall det finns en rutt mellan två städer där distansen är olika beroende på vilken riktning man använder, d.v.s. om $d_{rs} \neq d_{sr}$ gäller för en (r,s).[8]

Denna avhandling kommer jämföra körtiden och resultaten av några vanliga algoritmer för beräkning av optimala och approximativt optimala rutter för handelresandeproblemet. Dessa algoritmer är brute-force-algoritmen, giriga algoritmen, 2-opt, 3-opt. De ovannämnda algoritmerna är relativt välkända och har varit i bruk en längre tid. För att utreda prestandan för dessa algoritmer har jag implementerat dem i programmeringsspråket Java och kört dem mot gemensamma stokastiska handelresandeproblem.

2 Tillämpningar

Handelresandeproblemet kan tillämpas inom många olika områden och branscher och kan förekomma i både symmetrisk och asymmetrisk form. Ytterligare begränsningar kan tilläggas på problemet, t.ex. finns det ett problem med flera agenter som har den extra komplexiteten att flera resande säljare ingår i problemet. Dessa begränsningar och krav kan orsaka antingen att själva problemet blir svårare eller gör det svårare att konstruera en algoritm för att hitta en lösning. I vissa fall, t.ex. när det gäller ruttplanering av lager, kan egenskaper hos det aktuella problemet göra det underliggande delproblemet med resande handelsresenärer lättare att lösa, åtminstone då det gäller beräkning.

2.1 Logistik

2.1.1 Orderplockning i lager

Detta problem handlar om att hitta den optimala rutten för att hämta artiklar från ett lager. Anta att en beställning kommer in till ett lager för en viss delmängd av de artiklar som finns lagrade. Någon agent måste samla in alla artiklar som ingår i beställningen för att skicka dem till kunden. Detta är uppenbarligen ett symmetriskt TSP problem. Varornas lagringsplatser motsvarar noderna i grafen. Avståndet mellan två noder ges av tiden som behövs för att flytta fordonet från den ena platsen till den andra och givetvis är avståndet detsamma oberoende av vilket håll agenten går. Problemet att hitta den kortaste vägen för fordonet kan nu lösas som ett TSP. Oftast har dock lager liknande fotavtryck, det vill säga att de innehåller hyllor som är sorterade i långa kolumner med gångar som löper vinkelrätt genom dem. Denna detalj gör det möjligt att lösa denna variant av TSP linjärt. Denna algoritms prestanda minskar dock då flera vinkelräta vägar läggs till i lagret.[10], [8]

2.1.2 Ruttplanering av fordon

Anta n paket måste levereras till varsin unika mottagare i en stad. För att maximera logistikföretagets vinst måste rutten planeras så att chauffören både kör en så kort rutt som möjligt och samtidigt en rutt som är så snabb som möjligt, eftersom han får timlön. Detta bildar uppenbarligen ett asymmetriskt TSP problem. Orsaken till att detta blir asymmetriskt är att $d_{rs} = d_{sr}$ inte nödvändigtvis gäller för par av noder. Vid vissa ställen i staden finns det eventuellt enkelriktade gator. I ett fall där det finns två noder som är anslutna till en enkelriktad gata kan chauffören inte köra i fel riktning och då gäller att $d_{rs} \neq dsr$. För att kunna lätt räkna vikten, d.v.s. kostnaden för en båge mellan två punkter a och b så kan man beräkna ett nytt värde som på något sätt tar i beaktande både distansen och tiden det

tar att köra.
Problemet kan göras ytterligare utmanande ifall man antar att det också finns ett godtyckligt antal chaufförer *m*. I detta fall skulle det handla om att hitta rutter för alla chaufförer så att paketen levereras så snabbt och effektivt som möjligt. Dessutom så kan man ytterligare lägga tidskrav på när paketen skall levereras till kunden. I verkligheten måste ett logistikföretag dessutom ta i beaktande kapaciteten av chaufförens fordon och andra variabler, t.ex. hur länge en chaufför får köra under dagens lopp.[8]

2.2 Mönsterkort och kreskort

Ett mönsterkort är en laminerad struktur som består av ledande och isolerande ytor. Ibland används ordet kretskort för att beskriva dessa, men ett kretskort är i verkligheten ett mönsterkort med påmonterade elektroniska komponenter, t.ex. processorer o.s.v. Då ett mönsterkort tillverkas måste en maskin borra olika diameters hål i kretskortet för att ansluta ledare som finns på olika nivåer i kortet.

För att kunna variera storleken på hålen måste maskinen som borrar hålen byta verktygshuvud från en verktygslåda. Här uppstår uppenbarligen handelsresandeproblemet. I detta fall kan man tänka sig att en de olika hålen som skall borras bildar noderna i en graf. Bågarna för vissa av dessa kan innebära att maskinen endast flyttar verktygshuvudet till en viss koordinat och borrar ett hål där. Bågen mellan två hål av olika diameter utgörs av tiden det tar för maskinen att flytta verktygshuvudet till verktygslådan, byta verktygshuvudet och sedan flytta det nya verktygshuvudet till koordinaten av andra hålet.

För en fabrik som producerar mönsterkort är optimering av rutten som maskinen använder viktigt eftersom en minskning av den tid det tar att tillverka mönsterkortet kommer i sin tur att öka produktiviteten, vilket i sin tur kan leda till ökad vinst för företaget.[8].

handelsresandeproblemet uppstår också vid tillverkning av kretskort, där det gäller att optimera hur en maskin placerar olika elektriska komponenter på ett mönsterkort. [4] har visat att genom somTSP *(self-organizing map for travelling salesman problem)* optimering så kan man öka produktiviteten av fabrik som producerar kretskort.

Figur 2 visar en illustration av TSPLIB-RL5915 problemet. Detta är ett exempel problem som berör borrning av hål i mönsterkort. TSPLIB är en samling av handelresandeproblem av varierande storlekar och kontext.

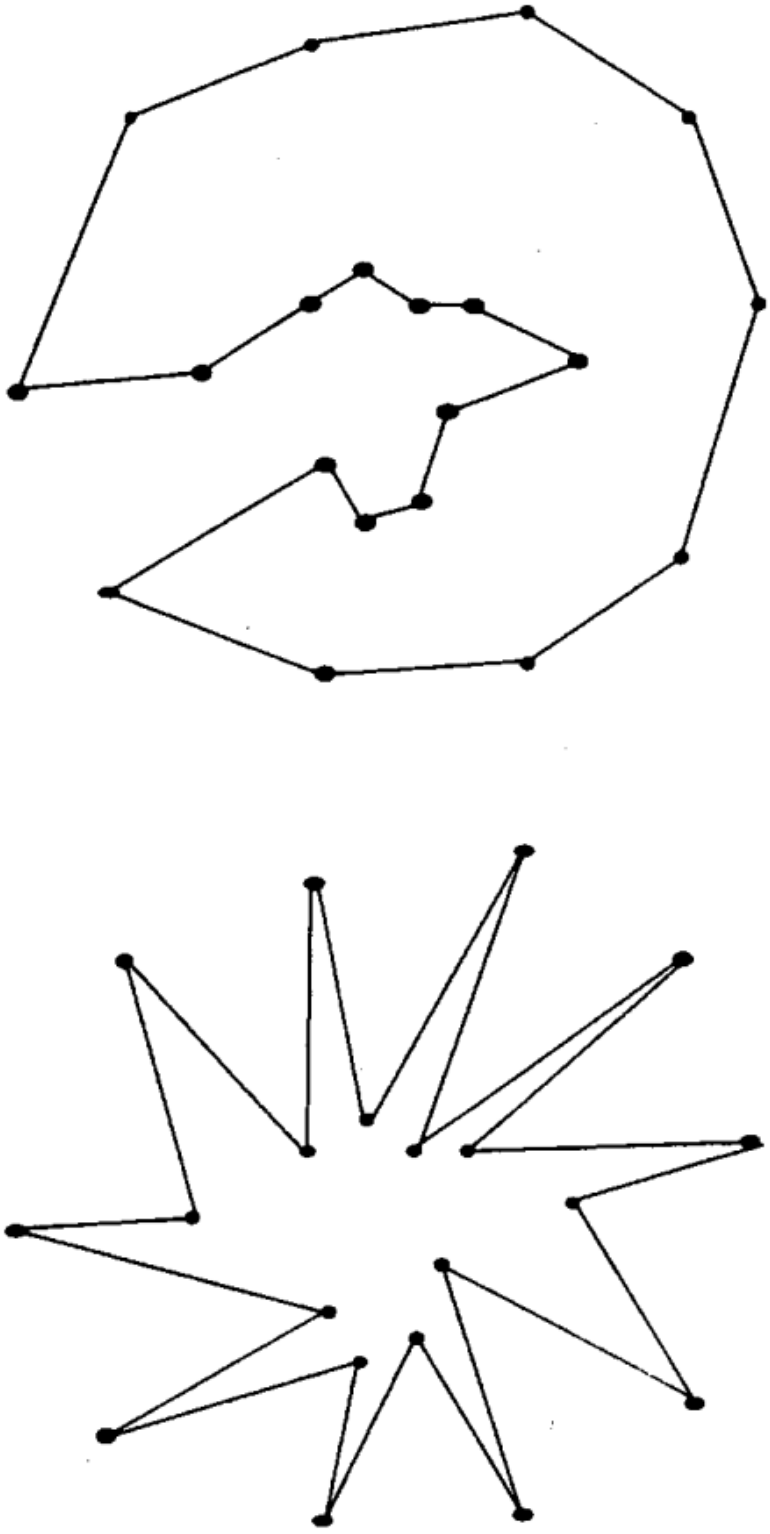
Figur 2: Illustration av RL5915

3 Mänsklig prestation

Eftersom handelsresandeproblem ofta kan representeras i ett tvådimensionellt euklidiskt rum är handelsresandeproblem med ett ganska litet antal punkter lätta för människor att både uppfatta visuellt och att lösa på ett ungefär optimalt sätt. De mentala mekanismer som ligger till grund för denna prestation verkar vara av låg komplexitet. I allmänhet konstateras att människans prestationer försämras linjärt när problemstorleken ökar exponentiellt. Människors förmåga att rita en optimal turné utan träning och på mycket kort tid, vanligtvis några sekunder, då antalet alternativa lösningar kan uppgå till miljarder eller biljoner står i stark kontrast till andra områden för resonemang och problemlösning. När det gäller handelresandeproblem finns det visserligen skillnader mellan de teoretiska beskrivningar av människans prestationer som finns i litteraturen, men en gemensam nämnare är att mekanismer som finns till för att upfylla andra perceptuella behov tas i bruk när människan ställs inför ett problem.[6]

Sequential Convex-hull Model är en modell som ursprungligen föreslogs av MacGregor och medarbetare[7]. De hävdar att tidiga perceptuella processer gör det möjligt att skapa en skiss av ett konvext skrov, varefter de inre punkterna i TSP:n kopplas samman sekventiellt. Denna modell stöds av det faktum att en ökning av antalet punkter inom det konvexa skrovet, dvs. de inre punkterna, minskar lösningens kvalitet, medan en ökning av antalet punkter på det konvexa skrovet inte påverkar lösningens kvalitet. Även införandet av manipulationer som förhindrar att det konvexa skrovet för det problem som används minskar lösningens kvalitet avsevärt[7]. Figur 3 illustrerar konceptet av Sequential Convex-hull modellen. Den övre figuren visar en kombination av noder som en människa naturligt föredrar, medan den nedre figuren innehåller ett stort antal indragningar som inte är optimalt.

Det har observerats att största delen av de lösningar till handelsresandeproblem som skapats av människor innehåller inte korsande bågar[6], detta illustreras i figurerna 4 och 5. Det är känt att optimala turer inte kan innehålla dessa korsande bågar. Denna heuristik att undvika korsande bågar har implementerats i en del algoritmer på så sätt att algoritmen genom en iterativ process avlägsnar korsande bågar. Heuristik är en metod eller tumregel baserad på någon observation som i fallet av handelresandeproblem optimerar rutten.



Figur 3: En illustration av Sequential Convex-hull modellen

4 Algoritmer och Heuristik

4.1 Brute-force algoritm

Brute-force algoritmen är den enklaste algoritmen för att hitta en optimal rutt i ett TSP problem. Algoritmen går genom varje möjliga turné och beräknar för var och en dess totala kostnad varefter den kan hitta den optimala rutten genom att hitta rutten med lägsta kostnaden. Denna algoritm är dock inte effektiv då antalet noder ökar. För n noder kommer algoritmen först välja en stad, sedan har algoritmen $n-1$ alternativ för nästa stad. På detta sätt kommer antalet möjliga rutter vara $(n-1)!$. Ifall det är fråga om en symmetrisk TSP så kommer det totala antalet rutter vara $\frac{(n-1)!}{2}$ eftersom riktningen inte har betydelse.

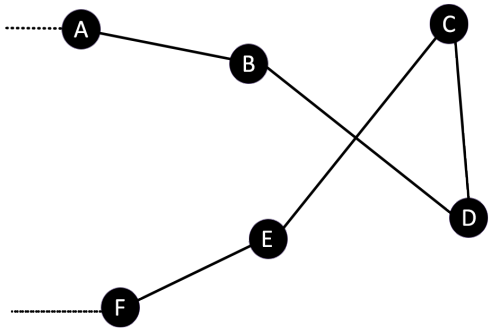
4.2 Girig algoritm

En girig algorim är en sådan algoritm som använder heuristiken av att alltid välja det lokalt optimala alternativet för att försöka hitta ett globalt optimum. Giriga algoritmer kommer sällan att hitta den optimala lösningen till ett problem men kan i vissa fall ge lösningar som approximerar optimala lösningen. Den största fördelen som giriga algoritmer har är deras genomförandetid. I handelsresandeproblemet blir denna heuristik följande: “Vid varje iteration, besök den närmaste, obesökta staden“. Giriga algoritmen kan altså endast generera en rutt för en given start nod i en graf och kan därmed endast genererar totalt n antal rutter för n antal noder. Den giriga algoritment brukas också kallas till närmaste granne heuristik (*Closest neighbor heuristic*).

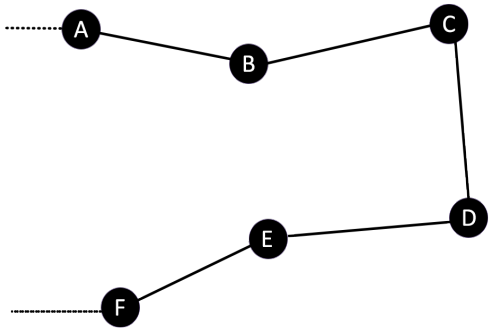
4.3 2-opt

2-opt är en lokal sökalgoritm som kan användas för att hitta en approximativt bra lösning till TSP. 2-opt-algoritmen genererar inte själv en turné utan försöker istället minska den totala längden på en befintlig turné. Algoritmen fungerar genom att radera två bågar från grafen och sedan återkoppla noderna så att en ny turné bildas. En av anledningarna till att algoritmen är effektiv är att den på sätt och vis tillämpar den heuristiska metoden att undvika korsningar i grafen.

En implementation av 2-opt algoritmen kommer att iterera över alla kombinationer av sekventiella noder. För varje kombination kommer nodernas ordning svängas om och distansen räknas. Ifall det händer sig att den nya turnén som genererats är kortare än den nuvarande turnén så uppdateras den nuvarande turnén med de ändringar som just gjordes. Algoritmen kör så länge tills en hel iteration av alla möjliga kombinationer av sekventiella noder har prövats utan att hitta en bättre turné.



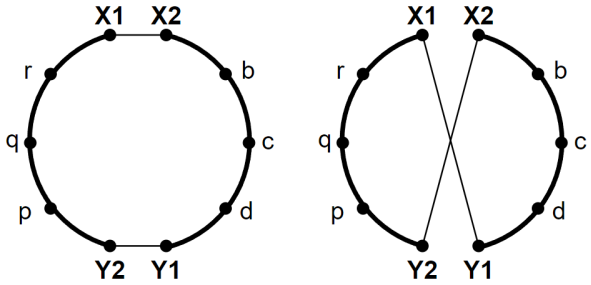
Figur 4: En illustration av en del av en graf där kanterna d_{BD} och d_{CE} korsar



Figur 5: En illustration av samma graf som figur 2 där korsningen blivit otrasslad

Exempel av 2-opt (från figur 2)
Nuvarande bästa turnén är [a,b,d,c,e,f]. 2-opt använder två heltalsvariabler i och k för att bestämma vilken delgraf skall svängas om. När $i = 2$, $k=3$ så byter elementet vid index 2 (d) och index 3 (c) plats. Grafen updateras till [a,b,c,d,e,f]

Förrutom med exakta euclidiska representation kan man också representera 2-opt algoritmen genom en skematisk graf. I figur 6 ser man en skematisk representatoin av 2-opt. Det är inte meningen att uppfatta grafen som euklidisk utan endast som en representation av möjliga kombinationer, utan att ta i beaktande de resulterande längderna av turnéerna.



Figur 6: En skematisk representation av de möjliga 2-opt byten

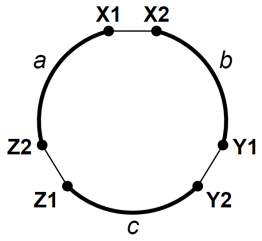
Den vänstra grafen i figur 6 visar en turné:
[X1-X2-b-c-d-Y1-Y2-p-q-r].
Den högra representerar grafen efter att 2-opt har gjort en omkombination genom att radera bågarna *X1-X2* och *Y1-Y2* och omkombinerat dem för att bilda turnén:
[X1-Y1-d-c-b-X2-Y2-p-q-r]

2-opt är en mycket bra algoritm för att generera lösningar till TSP-problem där antalet noder helt enkelt är för stort för att kunna beräknas med en metod som skulle ge den optimala turnén. I vissa fall kan den ge en optimal turné för TSP-problem där antalet noder är litet. Algoritmen är populär eftersom den är ganska lätt att implementera snabbt, är enkel att förstå och ger relativt goda resultat. Algoritmens tidskomplexitet är $\mathcal{O}(n^2)$. En turné som optimerats med 2-opt kallas 2-opt komplett.

4.4 3-opt

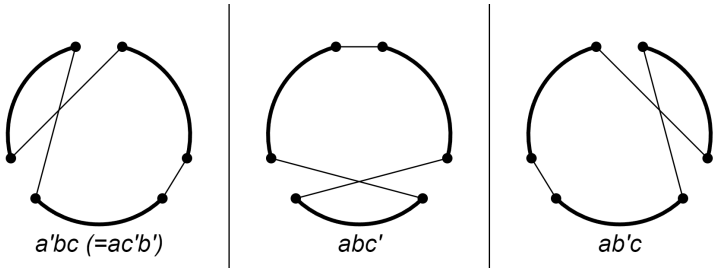
3-opt är en algoritm som bygger på de metoder som används i 2-opt, men där 2-opt tar hänsyn till de olika sätten att kombinera två borttagna bågar, tar 3-opt hänsyn till de olika sätten att kombinera tre borttagna bågar. Algoritmen 3-opt kombinerar dessa bågar för att bilda åtta nya turkombinationer och beräknar sedan det totala avståndet för varje tur, varefter den kortaste nya turen väljs, och om ingen är kortare görs inga ändringar.

Det totala antalet olika omkonfigurationer som kan produceras av 3-opt är åtta. Den första av dessa omkonfigurationer är helt enkelt den ursprungliga konfigurationen.



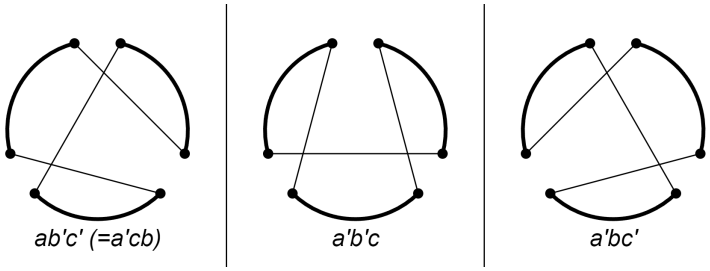
Figur 7: En skematisk representation av ursprungskonfigurationen av 3-opt [1]

Därefter finns det tre flyttningar som i sig själva utgör en enda 2-opt-flyttning.



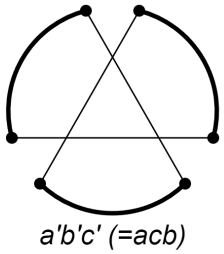
Figur 8: En skematisk representation av enskilda 2-opt drag som evalueras av 3-opt [1]

De följande tre utgör två på varandra följande 2-opt drag.



Figur 9: En skematisk representation av dubbel 2-opt drag som evalueras av 3-opt [1]

Den sista konfigurationen består av tre på varandra följande 2-opt drag.



Figur 10: En skematisk representation det trippel 2-opt draget som evalueras av 3-opt [1]

Av detta kan vi se att eftersom en 3-opt-algoritm kommer att utföra minst alla drag som en 2-opt-algoritm skulle göra så är en 3-opt-optimal tur också 2-opt-optimal. Denna idé kan generaliseras ytterligare till att säga att varje K-opt-optimerad tur också är (K-1)-opt-optimerad. Eftersom 3-opt-algoritmen kan beakta ett större antal möjliga omkombinationer av den ursprungliga rundturen kan den också hitta lösningar som närmar sig den optimala lösningen på ett TSP-problem mer exakt. Denna fördel har dock en viktig nackdel, nämligen tidskomplexitet. Tidskomplexiteten för 3-opt-algoritmen är $\mathcal{O}(n^3)$.

Utifrån detta kan vi redan se ett mönster bland k-opt-algoritmerna, nämligen att tidskomplexiteten för alla k-opt-algoritmer är $\mathcal{O}(n^k)$. På grund av denna ökning av tidskomplexiteten för ökande värden på k är det egentligen bara möjligt att använda en enkel version av en k-opt-algoritm för små k-värden, som 2 och 3.

Genomförandet av 3-opt-algoritmen är relativt okomplicerat, eftersom den använder sig av samma metod för omvändning av delräckor som 2-opt. Medan 2-opt-algoritmen tar hänsyn till en av de två möjliga bytena för två kanter, varav den ena är den ursprungliga kombinationen, tar 3-opt hänsyn till sju omkombinationer. Det är därför ganska lätt att implementera 3-opt, förutsatt att man redan har en 2-opt-implementering. Implementationen behöver bara en till for-slinga och ett heltalsvärde k som gör det möjligt för algoritmen att byta ut tre kanter.

Efter att for-slingan har lagts till valde jag att hårdkoda de metoder som används för att kontrollera turavståndet, eftersom det bara är sju möjliga kombinationer som behöver kontrolleras. Om man skulle genomföra ytterligare k-opt-algoritmer, t.ex. 4-opt, så skulle det finnas totalt 48 möjliga omkombinationer som var och en skulle behöva utvärderas. I så fall kan det vara meningsfullt att skapa en metod som kan skapa alla möjliga permutationer av byten för att minska kodens komplexitet och underlätta läsbarheten. K-opt-algoritmer med k-värden högre än 3 kommer dock att drabbas hårt på grund av den ökningen av tidskomplexiteten. I de fall då dessa k-opt-algoritmer med högre värde används, används de med ytterligare heuristik för att minska deras sökutrymme.

4.5 Simulerad Glödgning

Glödgning är en process där ett material värms upp och tillåts sedan svalna långsamt. Det finns flera användningsområden för denna process, t.ex. glödhett järn som tillåts långsamt svalna har annorlunda egenskaper än om det snabbt kyls ned genom avkylning.

När ett material har värmts upp tillåts atomerna som normalt befinner sig i en kristallin struktur att röra på sig. Då man låter materialet svalna långsamt har atomerna en möjlighet att hitta en ny struktur. När det gäller en järnstång kommer denna process i lyckade fall ge en järnstång som är mer flexibel, mjukare och innehåller färre ojämnheter.

Simulerad glödgning innebär att man simulerar denna process. Denna metod gör det möjligt att undvika lokala optimum för att möjligtvis hitta ett globalt optimum. Detta beror på att algoritmen tillåts att ibland välja nya turnéer till handelesresandeproblemet som har en längre total längd än den tidigare turnéen.

Processen av svalnande simuleras genom att långsamt minska sannolikheten för att algoritmen ska acceptera en längre turné för handelsresandeproblemet. På implementeringsnivå kan detta åstadkommas genom att använda ett siffervärde som under på varandra följande iterationer av algoritmen minskar. Så länge som talvärdet är högt är det också relativt sett troligt att algoritmen väljer en längre väg än den bästa vägen. Den sannolikhetsfunktion som används för att avgöra om en ny turné som är längre än den nuvarande optimala vägen skall accepteras är följande:

$$p(x) = e^{\frac{Kostnad}{t}}$$

p(x) är sannolikheten att den nya turnéen accepteras. *Kostnad* är skillnaden mellan längden av den gamla turnéen och den nya, d.v.s. differensen mellan dessa. Värdet t är temperaturen vid en given iteration av algoritmen.

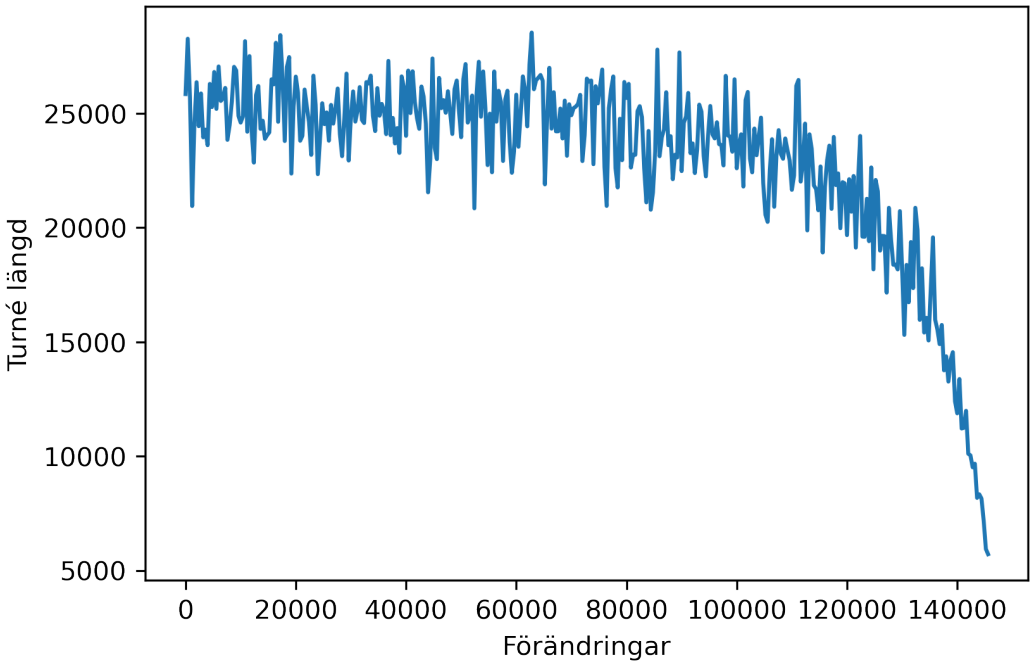
Om värdet av kostnadsskillnaden är negativt kommer värdet av exponentialfunktionen att vara mindre än ett. För allt mindre värden på kostnaden kommer också uttryckets totala värde minska. På samma sätt, när temperaturvärdet minskar tenderar uttryckets totala värde mot noll.

För att avgöra om en ny väg skall accepteras utvärderar algoritmen uttrycksvärdet *p(x)* i förhållande till ett slumpmässigt jämnt fördelat värde mellan 0 och 1. Om uttrycksvärdet är större än det slumpmässiga värdet kommer algoritmen att acceptera den nya vägen. Om värdet av uttrycket är mindre än det slumpmässiga värdet hoppar algoritmen över den nya vägen. De slumpmässiga talen som genereras skall vara av en uniform distribution mellan 0 och 1[5].

Vid startläget när temperaturvärdet är som störst, kommer algortimen att acceptera nya, längre turnéer med hög sannolikhet. Under algoritmens körtid kommer temperaturen

att multipliceras med ett avklyningsvärde *(cooling rate)* som kommer att minska värdet av temperaturen vid varje iteration. Då temperaturen sjunker, sjunker också sannolikeheten att nya, längre turnéer accepteras. Algoritmen fortsätter att köra tills temperaturvärdet har sjunkit under en på förhand vald gräns, varefter den slutar att köra.

Det drag som simulerad glödgning använder för att ändra turordningen är detsamma som 2-opt använder. Genom att ta bort två kanter och återkoppla de ingående noderna kan vi i praktiken generera alla möjliga kombinationer av turer för ett givet TSP-problem, och därför är 2-opt-bytesmetoden tillräcklig för att genomföra simulerad glödgning.



Figur 11: En graf som illusterar den totala längden av en rutt som optimeras av simulerad glödgning

5 Jämförelse av prestanda

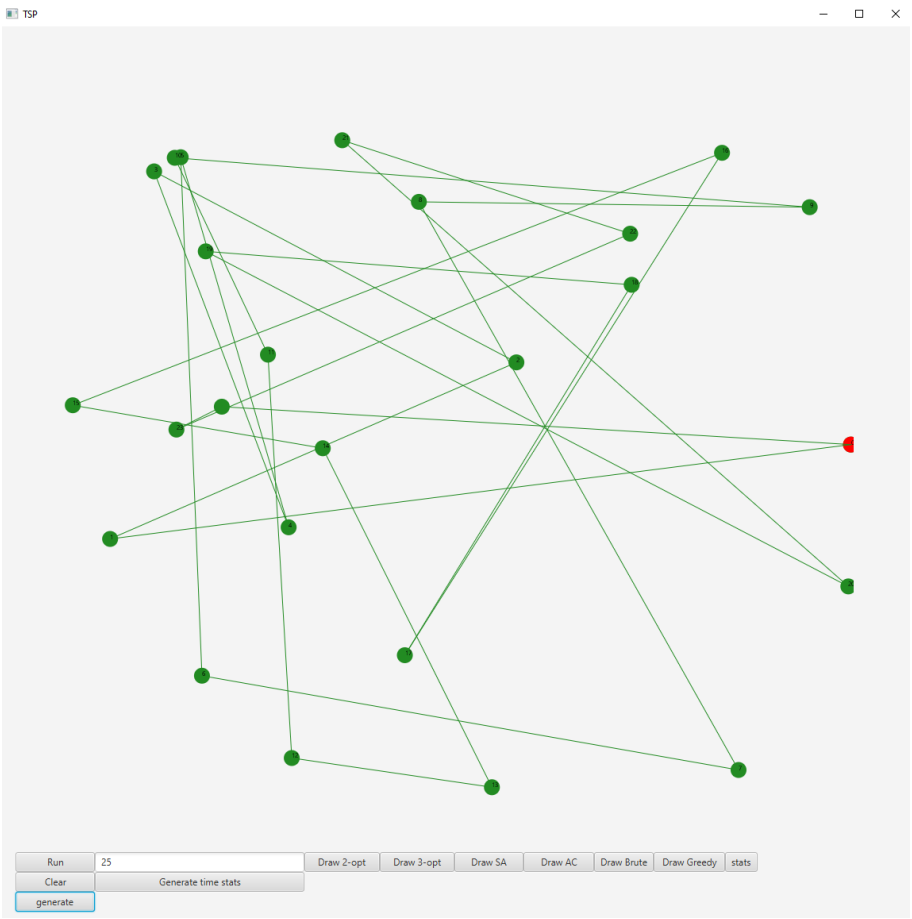
Eftersom det finns ett stort antal olika tillämpningar av TSP finns det också olika krav på att hitta en möjlig väg. I vissa tillämpningar kan det finnas krav på att hitta en lösning snabbt, t.ex. att hitta en logistisk rutt för en leveransbil. I andra tillämpningar kan tiden för att hitta en lösning vara mindre pressande, medan kravet är att hitta en lösning som närmar sig den optimala rutten så bra som möjligt, t.ex. optimering av en verktygsbana för en maskin. Dessa krav påverkas också utifrån det totala antalet städer, eller noder, som kan förväntas i problemet. I fall med mycket få städer fungerar naturligtvis brute-force-algoritmen, men på grund av dess dåliga tidskomplexitet $\mathcal{O}(n!)$ försämras dess prestanda mycket snabbt. När antalet städer växer över den punkt där brute force-algoritmen inte kan beräkna den optimala vägen i tid, används heuristik. Även dessa heuristiska algoritmer har dock sin egen tidskomplexitet som påverkar deras prestanda. Jag har därför för avsikt att testa prestandan hos de algoritmer som jag hittills har granskat i den här artikeln.

5.1 Metod

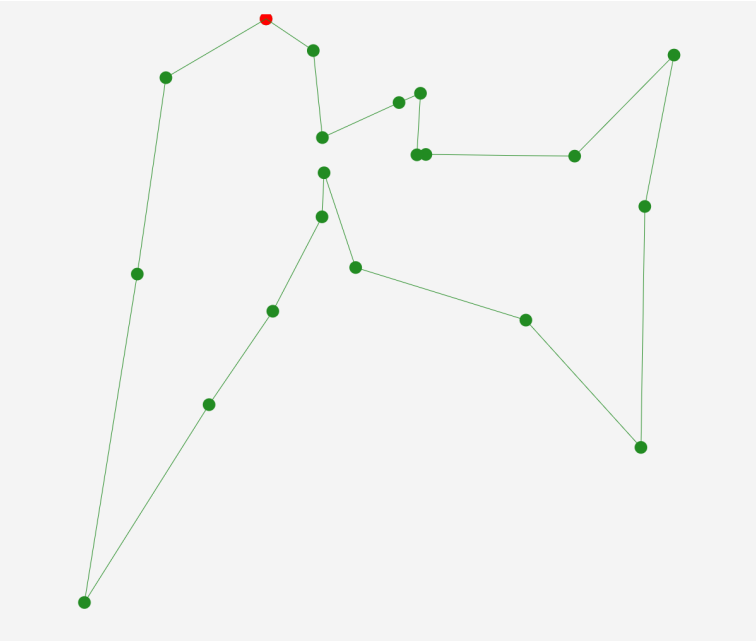
För att lätt visualisera handelsresandeproblem och för att ge en intuitiv förståelse för hur de olika algoritmerna beter sig har jag skapat ett program som genererar slumpmässiga TSP-instanser av en given storlek. Detta program innehåller funktionalitet för att rita den väg som förbinder punkterna i handelsresandeproblem med hjälp av de olika algoritmerna som behandlas i denna uppsats. Figur 12 visar en skärmdump av det grafiska användargränssnittet av *TravellingSalesmanProblemDemo* programmet.

Figurerna 13,14,15 och 16 visar lösningar som genereras av giriga algoritmen, 2-opt, 3-opt och simulerad glödgning respektive. Alla figurer använder sig en gemensam instans av handelsresandeproblemet. Programmet kommer automatiskt att rita upp den turnéen som genererats av algoritmen. Ursprungsstaden är den röda punkten medan de övriga gröna punkterna är de städerna som skall besökas.

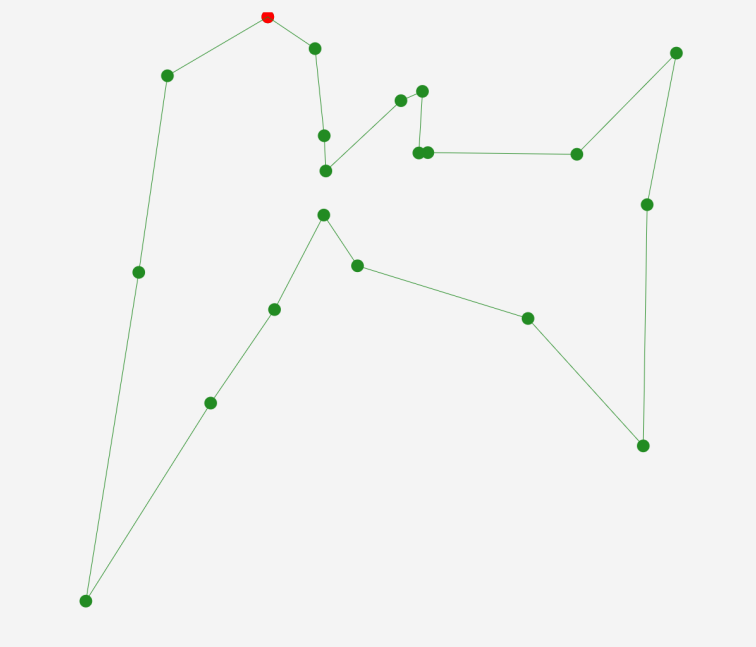
Alla algoritmer har implementerats i programmeringsspråket Java. Det TSP som används är av symmetrisk typ, där städerna representeras med klassen Point. En Point innehåller en X- och Y-koordinat. Ett gemensamt gränssnitt som innehåller en metod för att hitta och återge en array av typen punkt används. Implementeringarna av själva algoritmerna använder sig av en matris av matriser av primitiva dubbeltyper för att mäta avståndet mellan rutter. Där det är möjligt har jag använt de mest primitiva datatyperna för att algoritmerna ska kunna köras med minsta möjliga overhead. Ytterligare optimeringar kan vara möjliga, men på grund av varje algoritms enskilda tidskomplexitet kan dessa optimeringar inte ge några betydande tidsförbättringar i förhållande till de andra algoritmerna, särskilt när de körs på större handelsresandeproblem instanser.



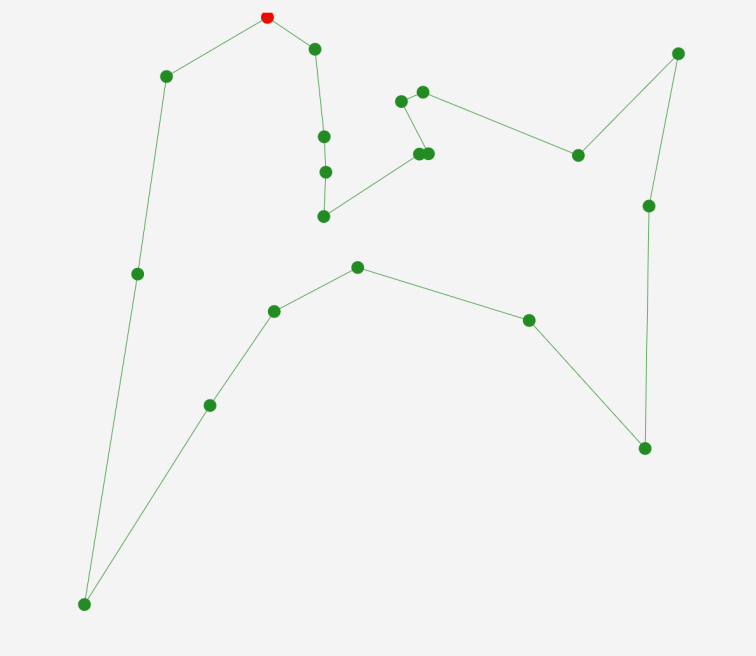
Figur 12: En skärmdump av TravellingSalesmanProblemDemo programmets användargränssnitt



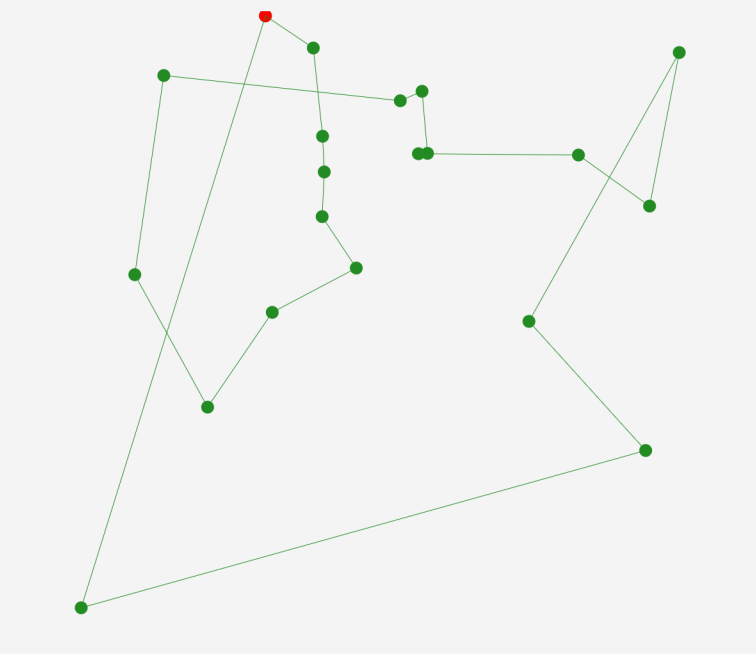
Figur 13: En skärmdump av den tur som genereras av 2-opt, ritad av *TravellingSalesman-ProblemDemo* programmet



Figur 14: En skärmdump av den tur som genereras av 3-opt, ritad av *TravellingSalesman-ProblemDemo* programmet



Figur 15: En skärmdump av den tur som genereras av simulerad glödgning, ritad av *TravellingSalesmanProblemDemo* programmet



Figur 16: En skärmdump av den tur som genereras av algoritmen greedy, ritad av *TravellingSalesmanProblemDemo* programmet

Jag har valt att köra totalt tre olika tester med olika stora handelsresandeproblem instanser. Först ett test med 10 slumpmässigt genererade TSP-instanser som var och en innehåller 50 slumpmässigt placerade städer. För det andra ett test med 10 instanser med 100 städer. För det tredje ett test med 10 instanser med 200 städer. Dessa städer genereras inom ett koordinatsystem av 1000 pixlars höjd och bredd. Städerna kan anta koordinater mellan 0 och 1000.

För att mäta prestandan har jag lagt till funktionalitet till programmet som gör det möjligt att köra alla algoritmer på en samling lika stora, slumpmässigt genererade handelsresandeproblem. Det totala avståndet för varje algoritms väg lagras för varje problem varefter dessa uppgifter kan användas för att mäta den genomsnittliga prestandan för varje respektive algoritm. Det är också möjligt att registrera körtiden för varje genomförande.

Dessutom kommer jag att jämföra de olika algoritmernas körtider för allt större handelsresandeproblem.

Testerna kommer att köras på en maskin utrustad med en AMD Ryzen 5 3600x sexkärnig processor med en boostklockhastighet på 4,2 GHz, 384KB L1 cache, 3.0MB L2 cache och 32.0MB L3 cache.

5.2 Resultat

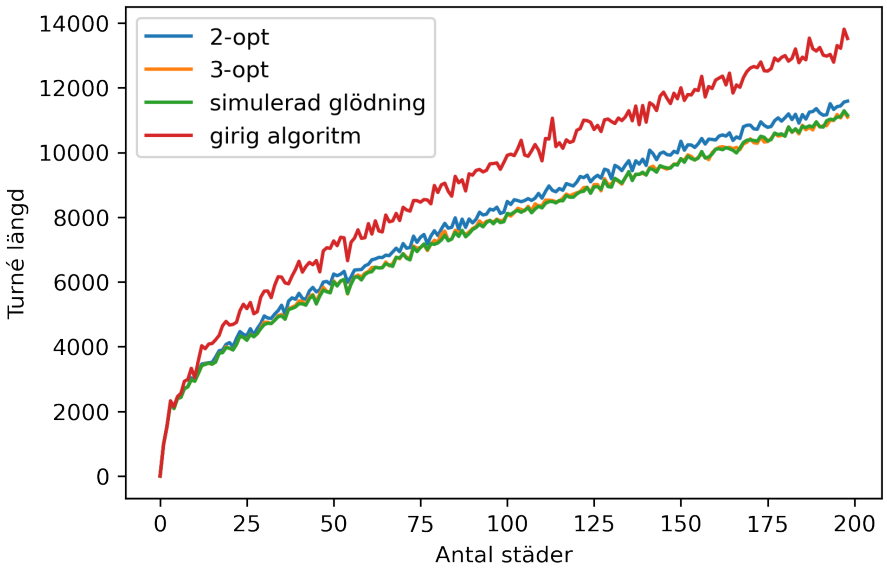
Algoritm:	Test 1 (n = 50)	Test 2 (n = 100)	Test 3 (n = 250)
Girig algoritm	7182.70	9714.98	13 334 .97
2-OPT	5986.06	8370.62	11 583 .40
3-OPT	5781.16	8049.22	11 012 .65
Simulerad glödning	5742.53	8018.16	11 114 .73

Figur 17: En tabell som visar medelvärdet av de genererade turnéerna för olika algoritmer

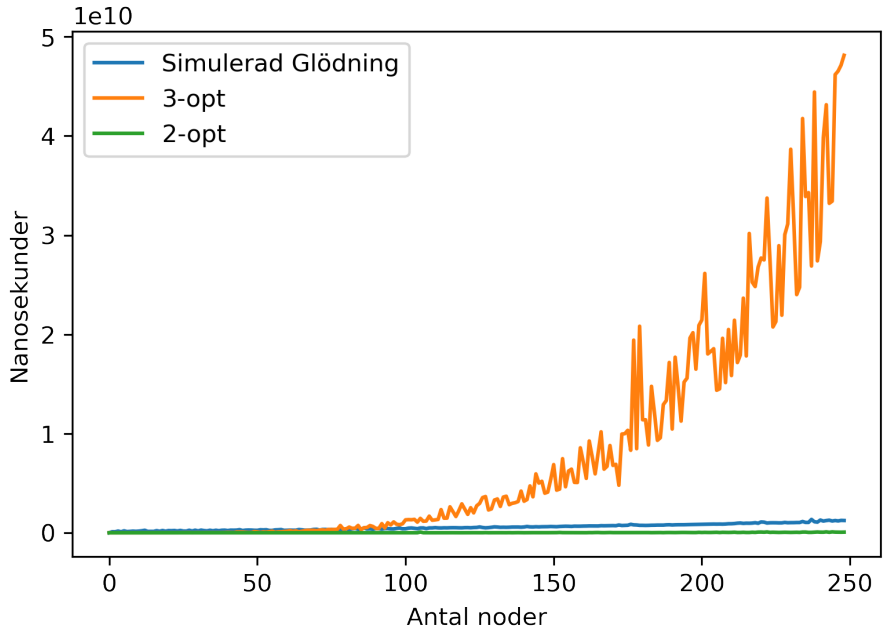
Det är värt att notera att eftersom dessa resultat genereras från helt slumpmässiga TSP-instanser är deras optimala väg inte känd, så det är inte möjligt att avgöra hur nära någon av metoderna ligger den optimala vägen, vi kan bara se de relativa skillnaderna mellan algoritmerna. En sak är omedelbart uppenbar från de resultat som visas i figur 12, nämligen att de giriga algoritmernas prestanda ligger långt efter de heuristiska metoderna. Detta är inte förvånande eftersom vi redan vet att den giriga algoritmen kan innehålla korsningar av bågar, vilket är känt att inte vara optimalt.

Resultaten visar också tydligt att 3-opt-algoritmen ger en märkbar förbättring jämfört med 2-opt-algoritmen.

Slutligen överträffar simulerad glödgning 3-opt i både test 1 och 2, men inte i test 3. Dessutom är den absoluta förbättring som simulerad glödgning uppnår i förhållande till



Figur 18: En graf som visar de genomsnittliga resultaten för varje algoritm för växande antal noder



Figur 19: En graf av exekveringstiderna för de olika algortimerna enligt antal noder

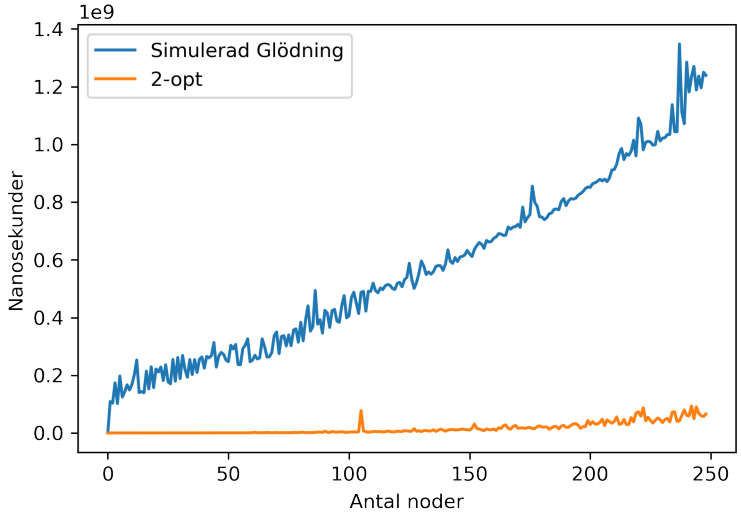
3-opt mindre än den som 3-opt uppnår i förhållande till 2-opt.

Figur 18 visar en graf medeltalet för längden av turnéer genererad av de olika algoritmerna för olika antal noder. För varje antal noder har programmet genererat 10 olika handelseresandeproblem som varje algortim har löst, sedan har programmet räknat medeltalet av dessa. Grafen ger vidare bevis att giriga algoritmens lösningar är betyligt längre än de övriga algoritmernas lösningar. Från grafen framgår att den absoluta förbättringen mellan giriga algoritmen och 2-opt är störst, och att de efterföljande algoritmerna endast ger små förbättringar. Dessutom märks det att 3-opt och simulerad glödgning ger liknande resultat.

Genom att göra olika test fann jag att även om 3-opt i genomsnitt ger bättre resultat än simulerad glödgning på större problem, ger simulerad glödgning vanligtvis en tur med kortare total distans om man får chansen att köra några gånger på ett visst problem. Med tanke på detta verkar det som om simulerad glödgning är en bättre algoritm när man försöker hitta en optimal tur, men man måste ställa detta i kontrast till algoritmernas totala körtider. Ett diagram över algoritmernas körtider enligt den metod som beskrivs i föregående avsnitt kan ses i figur 19.

I figur 19 kan vi se att algoritmernas totala körtider förblir hanterbara för n-värden under 100. Efter 100 noder börjar körtiden för 3-opt att växa snabbt, så snabbt att det är svårt att tydligt se skillnaden mellan körtiderna för 2-opt och simulerad glödgning. Mot slutet av grafen, när n närmar sig 250, kan vi se att körtiden för en instans av 3-opt kan vara nästan 50 sekunder. Det är också värt att notera att körtiden för 3-opt inte alltid ökar när antalet noder ökar. Detta beror på att 3-opt alltid kör en ny iteration av sig själv om en ändring görs i den aktuella iterationen. Eftersom TSP-instanserna genereras slumpmässigt kan turerna i vissa instanser mycket snabbt optimeras till ett lokalt minimum, medan andra kräver många iterationer av 3-opt-algoritmen.

I figur 20 kan vi se en graf av samma körtidsdata utan 3-opt-körtiderna. Här kan vi tydligare se att körtiden för simulerad glödgning växer snabbare än för 2-opt-algoritmen.



Figur 20: En graf av exekveringstiderna för 2-opt och simulerad glödgning enligt antal noder

6 Sammanfattning

Handelsresandeproblemet är ett gammalt problem och har en stor mängd vetenskaplig litteratur bakom sig. Nya metoder och förbättringar av gamla metoder för att hitta optimala eller ungefär optimala turer publiceras ständigt. Problemet fortsätter att vara ett populärt forskningsområde på grund av att det är vanligt förekommande inom många olika områden och discipliner. På grund av denna förekomst finns det ett stort incitament för ytterligare forskning t.ex. inom ekonomiska strävanden som optimering av logistik.

Eftersom Brute force-algoritmen är så dålig när det gäller tidskomplexitet och blir helt otymplig förbi TSP-fall med mer än 15 noder, kommer heuristiker troligen att vara den mest tillgängliga och enkla vägen för programvaruingenjörer att implementera.

Den giriga algoritmen ger en visuellt någorlunda acceptabel tur för TSP, men den innehåller fortfarande vissa egenskaper som gör det lätt att se att den inte är optimal, nämligen korsningar av bågar. 2-opt ger en stor förbättring jämfört med den grådiga algoritmens turlängd. Den kan också uppnå detta med en acceptabel tidskomplexitet $\mathcal{O}(n^2)$ som inte leder till okontrollerbara körtider. Dessa egenskaper gör 2-opt till en bra algoritm för instanser av handelsresandeproblemet där en lösning måste beräknas ganska snabbt och där kompromissen att ha en något längre tur än vad som kan uppnås med till exempel 3-opt är acceptabel.

3-opt ger en liten förbättring av turlängden jämfört med 2-opt. Denna förbättring är dock inte lika stor som den mellan greedy-algoritmen och 2-opt. Förbättringen sker dessutom till priset av ökad tidskomplexitet, vilket gör att 3-opt-algoritmernas körtid är mycket längre än 2-opt för stora TSP-instanser (n=250). I vissa fall där algoritmens totala körtid är mindre viktig och där minskningen av turlängden är viktigast, är 3-opt en ganska bra algoritm.

Simulerad glödgning kan med tillräckligt många försök generera turer med lägre total längd än 3-opt. Den kan också göra detta med en mycket kortare körtid. Även om 2-opt har en kortare körtid är de förbättringar som simulerad glödgning ger i fråga om turlängd ganska betydande. Av alla de algoritmer som behandlas i detta dokument verkar simulerad glödgning vara den mest lämpliga algoritmen. Den är enkel att genomföra, ger goda förbättringar av den totala turlängden jämfört med de andra algoritmerna och gör det i en jämförelsevis snabb körtid.

Källförteckning

[1] *3-opt move*. 2017. URL: https://tsp-basics.blogspot.com/2017/03/3-opt-move.html.

[2] David L. Applegate. *The traveling salesman problem a computational study*. Utg. av David L. Applegate. Course Book. Princeton series in applied mathematics. Ä Princeton University Press e-book.--Cover. Princeton: Princeton University Press, 2006, s. 606.

[3] N. Biggs, E.K. Lloyd och R.J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, 1986. ISBN: 9780198539162. URL: https://books.google.fi/books?id=XqYTk0sXmpoC.

[4] Kikuo Fujimura, Kwaw Obu-Cann och Heizo Tokutaka. ”Optimization of surface component mounting on the printed circuit board using SOM-TSP method”. I: (1999).

[5] Scott Kirkpatrick, C Daniel Gelatt Jr och Mario P Vecchi. ”Optimization by simulated annealing”. I: *Science* 220.4598 (1983), s. 671–680.

[6] James N MacGregor och Yun Chu. ”Human performance on the traveling salesman and related problems: A review”. I: *The Journal of Problem Solving* 3.2 (2011), s. 2.

[7] James N MacGregor och Tom Ormerod. ”Human performance on the traveling salesman problem”. I: *Perception & psychophysics* 58.4 (1996), s. 527–539.

[8] Rajesh Matai, Surya Singh och Murari Lal Mittal. ”Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches”. I: *Traveling Salesman Problem*. Utg. av Donald Davendra. Rijeka: IntechOpen, 2010. Kap. 1. DOI: 10.5772/12909. URL: https://doi.org/10.5772/12909.

[9] Stefan Näher. ”The Travelling Salesman Problem”. I: *Algorithms Unplugged*. Utg. av Berthold Vöcking m. fl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, s. 383–391. ISBN: 978-3-642-15328-0. DOI: 10.1007/978-3-642-15328-0_40. URL: https://doi.org/10.1007/978-3-642-15328-0_40.

[10] H. Ratliff och Arnon Rosenthal. ”Order-Picking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem”. I: *Operations Research* 31 (juni 1983), s. 507–521. DOI: 10.1287/opre.31.3.507.