Slutrapport i TDDD92:
Automatisk uppgiftstilldelning i StarCraft II

Felix Lagnöhed
fella149@student.liu.se

Sammanfattning

Följande rapport utreder två olika lösningsmetoder för tilldelningsproblemet och utvärderar deras lämplighet att implementeras i *StarCraft II*. De utvärderade lösningsmetoderna är en girig tilldelningsalgoritm och ungerska metoden. Efter utvärderingen kan det konstateras att ungerska metoden är bättre på att lösa tilldelningsproblemet men är svårare att implementera än den giriga algoritmen. Svaret på vilken algoritm som är att föredra blir därför subjektivt och beror på om enkelheten eller exaktheten är viktigast.

1 Problemställning

I följande avsnitt kommer problemet som teknikerna analyseras och utvärderas kring presenteras.

1.1 Motivering

Ett viktigt moment i *StarCraft II* är att så snabbt som möjligt samla tillgångar i form av gas och mineraler för att samtidigt kunna förbättra sin bas genom att exempelvis konstruera olika byggnader. För att få detta gjort krävs att arbetare tilldelas dessa uppgifter kontinuerligt, samt att rätt arbetare tilldelas rätt uppgift. Eftersom utförandet av viktiga uppgifter ska ta så lite tid som möjligt, kan det vara fördelaktigt att ha en algoritm som kan bestämma vilken arbetare som ska tilldelas en särskild uppgift. Ett exempel kan vara när både gas och mineraler ska samlas, samtidigt som två arbetare finns tillgängliga. Om raffineriet och mineralfältet står på motsatta sidor om varandra i basen är det mest tidseffektivt att tilldela den arbetare som står närmast mineralgruvan att hämta mineraler, samt att låta den andra arbetaren hämta gas. Den här situationen blir mer tidseffektiv än om arbetarna måste ödsla tid på att springa genom hela basen.

1.2 Konkretisering av problemet

För att lättare kunna illustrera problemet och dess applicerbarhet i *StarCraft II* krävs det att in- och utdata konkretiseras. Indatan kommer att vara de agenter och uppgifter som finns tillgängliga. Utdatan kommer att ge en tilldelning i form av en *mängd* över alla agent/uppgift-par.
Indata:

1. Tillgängliga agenter
2. Tillgängliga uppgifter

Utdata:

1. Tilldelningar mellan agenter och uppgifter

Något som är viktigt att poängtera är att en uppgift inte kan vara att *hämta mineraler*. En uppgift skulle istället kunna vara *hämta mineraler i mineralfält med id* X, det vill säga en specifik enhet eller plats istället för en handling. Detta är nödvändigt för att kunna konkretisera *vart* agenten ska gå för att kunna utföra sin uppgift, annars blir det svårt att räkna ut distansen mellan agenten och uppgiften.

Notera att i denna rapport kommer begreppet *agent* att användas flitigt. I tilldelningsproblemet pratas det ofta om tilldelning av agenter till uppgifter men agent kan även vara den agent som spelar *StarCraft*. Anledningen till varför agent är ett bra begrepp för det framlagda problemet i *StarCraft* är att det finns flera typer av agenter: arbetare, marinsoldater, *siege tanks* och *marauders*, för att nämna några. Att bara nämna arbetare alternativt marinsoldater i rapporten blir således för avgränsande. Detta tillsammans med det faktum att agenter är ett vedertaget begrepp inom tilldelningsproblemet gör att det kommer användas även i denna rapport.

2 Utvärderingskriterier

De tekniker som presenteras i rapporten kommer utvärderas enligt följande kriterier:

1. Hur väl verkar det gå att förstå den tänkta lösningsmetoden och att applicera den under implementationsfasen?
2. Hur väl verkar lösningsmetoden kunna lösa exakt det problem som ställdes upp i avsnitt 1?
3. Hur väl verkar lösningsmetoden kunna bidra till att agenten blir bättre på att spela *StarCraft*?
4. Hur *säkert* verkar det vara att välja den tänkta lösningsmetoden?

3 Tekniker och algoritmer

Här presenteras de valda lösningsmetoderna som utredningen görs kring.

3.1 Ungerska metoden

En gammal och välkänd lösning på tilldelningsproblemet är ungerska metoden. Den formulerades 1955 av Harald Kuhn, som baserade sin lösning på 2 ungerska matematikers verk,

vilket lade grund till algoritmens namn [1]. Ungerska metoden är komplett, vilket betyder att den är garanterad att hitta en lösning om en sådan finns. Vidare är den även optimal i den mening att den lösning som ges till problemet kommer vara den bästa.

Som beskrivet i avsnitt 1.2 består indata till algoritmen av två mängder; A, som innehåller alla arbetare, och T, som innehåller alla uppgifter. För varje arbetare $a \in A$ och varje uppgift $t \in T$ finns en kostnadsfunktion $v(a, t)$ som ger kostnaden att tilldela agent a till uppgift t. Kostnaden för varje kombination av agent/uppgift-par placeras sedan i en $n \times m$-matris där $n = |A|$ och $m = |T|$. Notera att ett krav i tilldelningsproblemet säger att en arbetare endast kan anta en uppgift, och en uppgift kan endast bli tilldelad till en arbetare. Det innebär att flera arbetare inte kan gå till samma uppgift, och att en arbetare inte kan ta fler än en uppgift.

Nedan följer en beskrivning av utförandet av algoritmen, där den är uppdelad i fem olika steg. I beskrivningen är kostnadsmatrisen en kvadratisk matris av ordning n, där alla agenter är kvalificerade att anta alla jobb.

3.1.1 Steg 1 - Radreducering

När kostnadsmatrisen har skapats är det dags för steg 1: att reducera rader. Det går ut på att hitta det minsta värdet för varje horisontell rad i matrisen. När det minsta värdet för varje rad har konstaterats, subtraheras det minsta värdet från elementet på den aktuella raden.

3.1.2 Steg 2 - Kolonnreducering

Steg två liknar steg 1, med enda skillnaden att operationerna ska utföras på kolonner, istället för rader. Indata till steg 2 blir utdatan från steg 1, närmare bestämt den radreducerade matrisen. Det minsta värdet hämtas från varje kolonn, och varje element i varje kolonn subtraheras med det minimivärdet. Sedan går algoritmen vidare till steg 3.

3.1.3 Steg 3 - Hitta nolltermsranken

Nu har både matrisens rader och kolonner reducerats, och det är dags att undersöka möjligheten för en optimal tilldelning. I den nuvarande matrisen, hitta det minsta antal linjer L som behövs dras i matrisen så att alla nollor är överstrukna. Metoden kallas nolltermsranken (en. *zero-term rank*) av en matris och beskrivs mer utförligt i en rapport skriven av LeRoy Beasley m.fl. [2]. En linje täcker antingen en hel rad eller en hel kolonn. Om matrisens dimensioner är $n \times n$ kan det minsta antalet linjer L vara $1 \leq L \leq n$. Nu finns det två alternativ; 1) $L = n$ och 2) $L < n$. Om $L = n$, gå till steg 5. Då finns det underlag för en optimal tilldelning. Om $L < n$ kan inte en optimal tilldelning göras, vilket gör att steg 4 måste genomföras.

3.1.4 Steg 4 - Skifta nollor

Steg 4 går ut på att skifta nollorna i matrisen för att kunna få till fallet $L = n$ i steg 3. Steg 4 börjar med att hitta det minsta värdet som inte är överstruket av en linje. Det värdet subtraheras från alla icke överstruktna värdena samtidigt som det adderas till alla värdena där två linjer korsar varandra. Steg 4 är nu klart och användaren hänvisas tillbaka till steg 3 med den nya matrisen.

3.1.5 Steg 5 - Slutgiltig tilldelning

Det sista steget är att göra den slutgiltiga tilldelningen. I den nuvarande matrisen ska n stycken nollor väljas så att inga valda nollor delar rad eller kolonn. Detta görs enklast genom att märka de valda nollorna med en asterisk. Om flera godtyckliga alternativ finns att tillgå spelar det ingen roll vilken sekvens av positioner som har valts, eftersom den totala kostnaden för båda kommer vara optimal. När nollorna har markerats jämförs positionerna med den ursprungliga kostnadsmatrisen, där summan av de markerade positionerna blir en totala optimala kostnaden för problemet.

3.1.6 Exempel

Nedan följer ett konkret räkneexempel som följer steg 1–5. Låt säga att kostnadsmatrisen ser ut enligt följande:

$$\begin{bmatrix} 30 & 25 & [\mathbf{10}] \\ 15 & [\mathbf{10}] & 20 \\ 25 & 20 & [\mathbf{15}] \end{bmatrix}$$

Raderna i matrisen motsvarar agenter och kolonnerna motsvarar uppgifter. Rad 1, med värdena [30, 25, 10], innehåller exempelvis de kostnader som fås när agent a_1 tilldelas uppgift t_i, där $1 \leq i \leq n = 3$. På samma sätt är kolumn 1 de kostnader som fås när agent a_i tilldelas uppgift t_1. De markerade cellerna i matrisen är de minsta värdena för respektive rad. Efter radreduceringen i steg 1 markeras de minsta värdena för respektive kolonn i den nya matrisen enligt följande:

$$\begin{bmatrix} 20 & 15 & [\mathbf{0}] \\ [\mathbf{5}] & [\mathbf{0}] & 10 \\ 10 & 5 & 0 \end{bmatrix}$$

Dessa markerade värden subtraheras från alla celler i matrisen. I steg 3 ska nolltermsranken för den nuvarande matrisen undersökas:

$$\begin{bmatrix} 15 & 15 & 0 \\ 0 & 0 & 10 \\ 5 & 5 & 0 \end{bmatrix}$$

Det minsta antalet linjer som behövs för att täcka alla nollor i matrisen är 2. Eftersom $L = 2 < n = 3$ går algoritmen vidare till steg 4. Av de element som inte är överstrukna är 5 det minsta värdet. De icke överstrukna värdena subtraheras med 5. Värdet 10 befinner sig på den enda positionen i matrisen där två linjer korsar varandra, så det värdet adderas med 5 enligt instruktionerna för steg 4. Dessa operationer resulterar i följande matris:

$$\begin{bmatrix} 10 & 10 & 0 \\ 0 & 0 & 15 \\ 0 & 0 & 0 \end{bmatrix}$$

Nu satisfierar matrisen kriteriet $L = n = 3$ och steg 5 kan påbörjas. Nu ska n st nollor väljas enligt villkoret att de valda nollorna inte delar varken rad eller kolonn. Detta kan göras på två sätt:

$$\begin{bmatrix} 10 & 10 & 0* \\ 0 & 0* & 15 \\ 0* & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 10 & 10 & 0* \\ 0* & 0 & 15 \\ 0 & 0* & 0 \end{bmatrix}$$

Asteriskerna indikerar att elementet har blivit valt. Byt ut den nuvarande matrisen mot den ursprungliga kostnadsmatrisen men behåll markeringarna:

$$\begin{bmatrix} 30 & 25 & 10* \\ 15 & 10* & 20 \\ 25* & 20 & 15 \end{bmatrix} \quad \begin{bmatrix} 30 & 25 & 10* \\ 15* & 10 & 20 \\ 25 & 20* & 15 \end{bmatrix}$$

Summan av de markerade värdena blir den totala kostnaden för tilldelningen. I detta fall blir summan $25 + 10 + 10 = 15 + 20 + 10 = 45$. Alltså är kostnaden för den optimala tilldelningen för det givna problemet 45. För tydlighetens skull kan det även konstateras att tilldelningen resulterade i följande uppsättningar av agent/uppgift-par: $\{(a_3, t_1), (a_2, t_2), (a_1, t_3)\}$ och $\{(a_2, t_1), (a_3, t_2), (a_1, t_3)\}$.

3.2 Agentbaserad girig algoritm

Den andra algoritmen som undersökningen innefattar är en agentbaserad girig algoritm som presenteras i Heintz och Präntares undersökning [3]. I detta avsnitt kommer det kortfattat förtydligas vad som menas med en girig algoritm och varför den presenterade algoritmen är just en girig sådan. Slutligen kommer pseudokod till algoritmen presenteras.

3.2.1 Giriga algoritmer

En algoritm klassas som girig när den ger lokalt optimala lösningar istället för globalt. En bra tumregel är att varje delproblem kan klassas som ett eget problem i en girig algoritm, vilket innebär att algoritmen kommer välja det mest uppenbara alternativet för stunden utan att tänka om det ger den bästa lösningen sett till helheten. Giriga algoritmer är ofta bra att använda när *lokalt* optimala lösningar alltid leder till *globalt* optimala lösningar. Det innebär med andra ord att om det för varje delproblem görs ett val där lönsamheten är störst, kommer det leda till en optimal slutgiltig lösning. Giriga algoritmer är däremot inte optimala i problem där det tidigare nämnda kriteriet att lokalt optimala lösningar leder till globalt optimala lösningar uppfylls. Ett bra exempel är när objekt med olika vikt och värde ska placeras i en ryggsäck med en viss viktkapacitet. Syftet med problemet är att maximera värdet i ryggsäcken. En girig algoritm fungerar sämre i sådana problem eftersom det måste tas hänsyn till både vikt och värde. Att välja det mest värdefulla alternativt lättaste föremålet må vara en lokalt optimal lösning, men garanterar inte en globalt optimal lösning.

3.2.2 Girig algoritm för tilldelningsproblemet

Som Präntare och Heintz skriver i sin rapport garanterar deras giriga algoritm inte en optimal lösning på tilldelningsproblemet [3]. I deras undersökning används den mer som en måttstock till en annan algoritm som presenteras i texten. Vidare skriver de att meningen med deras agentbaserade giriga algoritm är att ge en godtycklig lösning där lokala optimum prioriteras utan hänsyn till det globala värdet. Nedan ges pseudokoden för algoritmen. Den har en uppsättning agenter A och en uppsättning uppgifter T som indata, och ger ett mängd S som utdata, som är själva tilldelningen.

Algoritmen i avsnitt 3.2.2 är som sagt agentbaserad och itererar därför först över alla agenter. Variabeln k som initieras till 0 i varje iteration över uppsättningen agenter betecknar det index där den lokalt optimala lösningen finns. Variablerna

Algoritm 1 Agentbaserad girig algoritm

```
1:  S ← ∅
2:  for i = 1,...,|A| do
3:      k ← 0
4:      u ← −∞
5:      for j = 1,...,|T| do
6:          u′ ← v(S[j], T[j])
7:          S[j] ← S[j] ∪ {A[i]}
8:          u″ ← v(S[j], T[j])
9:          if u″ − u′ > u then
10:             k ← j
11:             u ← u″ − u′
12:         end if
13:         S[j] ← S[j]\{A[i]}
14:     end for
15:     S[k] ← S[k] ∪ {A[i]}
16: end for
17: return S
```

u, u' och u'' är beteckningar för värdet vid olika tillfällen; u betecknar det nuvarande bästa värdet för den nuvarande agenten, u' och u'' används för att undersöka om den temporära tilldelningen ger ett bättre värde än tidigare. Som pseudokoden visar går algoritmen ut på att för varje agent hitta den uppgift som ger bäst värde för just den agenten. När den uppgiften har hittats läggs agenten till i S på den plats där det bästa värdet hittades.

Något som är viktigt att notera är att pseudokoden för algoritmen som nyss presenterats utgår från att värdet mellan agent och uppgift ska maximeras. För lösningar till tilldelningsproblemet brukar det istället vara av intresse att minimera värdet, eller rättare sagt kostnaden. Det innebär att antingen algoritmen eller kostnadsfunktionen behöver en lättare modifikation. Ett förslag på förändring är att ändra kostnadsfunktionen från $u = v(a, t)$ till $u = 1/v(a, t)$. Modifikationen kommer resultera i att värdet u minskar när avståndet $v(a, t)$ ökar, vilket passar algoritmens uppbyggnad mer. Ett annat sätt att anpassa algoritmen till det nämnda problemet är att modifiera *if-satserna* i pseudokoden. Istället för att, på rad 9, kontrollera att differensen mellan de temporära värdena är större än det hittills hösta värdet kan det vara bättre att kolla om differensen är mindre. Fördelen att göra på det här sättet kan vara att funktionen $u = v(a, t)$ kan fortsätta tolkas som kostnaden mellan ett agent/uppgift-par istället för att introducera det nya, något mer svårtolkade begreppet *värde* till problemet.

Som tidigare nämnts är den presenterade algoritmen inte optimal för tilldelningsproblemet. Det går så klart att pricka rätt enstaka gånger med problemet, men generellt är den inte optimal. Exempelvis kan den lokalt optimala tilldelningen för agent 1 vara uppgift 1. Om agent 2 också har uppgift 1 som värdemässigt optimal uppgift, med ett större värde än det värdet mellan agent 1 och uppgift 1, blir lösningen inte optimal. Det händer eftersom tilldelningen är explicit, alltså att samma agent inte kan tilldelas till olika uppgifter och samma uppgift kan inte tilldelas fler agenter.

3.3 Kostnadsfunktion: euklidiska avståndet

I båda de presenterade teknikerna krävs en kostnadsfunktion för att avgöra hur bra en viss tilldelning är. Den kostnaden

som används i denna utredning är det euklidiska avståndet mellan en agent och en uppgift. Enligt artikeln är det en av de vanligaste metoderna att mäta distans mellan två objekt [4]. För att räkna ut det euklidiska avståndet appliceras Pythagoras sats. Avståndet c räknas således ut med ekvationen $c = \sqrt{(a_y - t_y)^2 + (a_x - t_x)^2}$, där a_y och a_x är benämningar på *agenternas* y- och x-koordinater och t_y samt t_x är benämningar på *uppgifternas* diton.

3.4 Sammanfattning av tekniker att undersöka

Här kommer en sammanfattning av teknikerna att undersöka:

1. Ungerska metoden, ensam.
2. Agentbaserad girig algoritm, ensam.

4 Användning i StarCraft II

I följande sektion beskrivs hur de tidigare nämnda teknikerna kan implementeras i *StarCraft II* för att lösa tilldelningsproblemet.

4.1 Ungerska metoden

Den ungerska metoden skulle kunna implementeras i *StarCraft II* för att ge en optimal lösning på tilldelningsproblemet. I *StarCraft II* finns arbetare och soldater för att kunna förbättra basen och strida mot motståndaren. Tilldelningsproblemet kan appliceras när dessa ska tilldelas uppgifter, och den ungerska metoden skulle då kunna användas här. Kostnadsfunktionen skulle kunna vara det euklidiska avståndet mellan en enhet och en uppgift. Detta kan dock bli ett problem i de fall där hinder står mellan enhet och uppgift, vilket gör att den egentliga vägen för enheten att förflytta sig är mycket längre än fågelvägen. Oftast kommer sådana situationer sannolikt inte att uppstå under spelets gång, men det kan vara värt att notera risken. Kostnadsmatrisen skulle således kunna byggas upp där varje cell i matrisen är kostnaden för att tilldela en arbetare en uppgift. Notera nu att en arbetare inte kan ingå i en armé och soldater inte kan exempelvis samla mineraler. Det finns en utökning på den ungerska metoden som inte tagits med i denna rapport som tar hänsyn till om en enhet är kvalificerad att anta en uppgift eller inte. Anledningen till varför den funktionaliteten inte riktigt behövs när Ungerska algoritmen implementeras i *StarCraft II* är för att soldater och arbetare hanteras separerat; de har alltså inget med varandra att göra. Att då blanda dessa i samma kostnadsmatris skulle således kunna vara förvirrande och kontraintuituvt.

När kostnadsfunktionen har konstruerats är det bara att följa de olika stegen i algoritmen för att till slut få en optimal uppgiftstilldelning. Beroende på behov kan den ungerska algoritmen antingen användas till att tilldela soldater uppgifter, eller att tilldela arbetare uppgifter. Eftersom de har olika färdigheter krävs det att algoritmen genomförs två gånger; en för arbetare och en för soldater. Det kommer inte bara vara uppsättningen agenter som då blir annorlunda; de tillgängliga uppgifterna ändras naturligtvis också. Vanliga uppgifter för en soldat kan vara att försvara basen, attackera motståndare eller att utforska oupptäckta områden på kartan.

4.2 Agentbaserad girig algoritm

Den giriga algoritmen skulle också kunna implementeras för att lösa tilldelningsproblemet i *StarCraft II*. Som det tidigare diskuterades i avsnitt 3.2.2 är definitionen av kostnadsfunktionen viktig. I *StarCraft II* vill spelaren att agenterna spenderar så lite tid som möjligt att förflytta sig på kartan för att kunna arbeta så effektivt som möjligt. Rimligen är kostnadsfunktionen även här det euklidiska avståndet mellan en agent och en uppgift. Det går säkerligen, exempelvis med hjälp av en sökalgoritm, att få fram längden på den snabbaste vägen för en agent att gå för att komma till uppgiften. Således skulle det måttet vara bättre för kostnadsfunktionen, men det euklidiska avståndet ger en godtycklig uppskattning som i de flesta fall är god nog för tilldelningsproblemet. Eftersom indata och utdata för den agentbaserade giriga algoritmen och den ungerska metoden är identiska behövs ingen ytterliga information tilläggas för att även den giriga algoritmen ska kunna implementeras i *StarCraft II*.

5 Utvärdering

Här utvärderas de föreslagna lösningsmetoderna baserat på kriterierna som presenterades i avsnitt 2.

5.1 Kriterium 1: Förstå och applicera

Ungerska metoden. Den ungerska metoden kan uppfattas som svår att förstå och applicera. Orsaken till varför det är nödvändigt att leta efter emphnolltermsranken och skifta nollor i matrisen är knappast intuitivt. Algoritmen innehåller som bekant 5 steg, vilket inte är allt för många. Dessvärre är några av stegen, som exempelvis steg 3 och 4, relativt omfattande vilket gör att algoritmen kan uppfattas som krånglig. Något som är bra med algoritmen är att stegen är så pass tydliga att det är lätt att dela upp koden i hjälpfunktioner. Varje steg skulle kunna arbeta självständigt för att lösa en deluppgift, vilket underlättar i strukturen vid eventuell applicering av algoritmen i implementationsfasen. Algoritmens struktur må vara lätt att förstå, men när det gäller specifika detaljer lämnar den en hel del att önska.

Agentbaserad girig algoritm. Den giriga algoritmen är mycket enkel (nåja, allt är relativt) att förstå. Den är intuitiv och innehåller inslag som kännetecknar en typisk girig algoritm. Eftersom den itererar över alla agenter, för att för varje agent välja den billigaste tilldelningen av uppgifter, är den rättfram och gör att den är lätt att förstå. Applicering av algoritmen ska inte heller vara några större problem. Instruktionen som presenterades i [3] är, som pseudokod ju ska vara, anpassad för programmering. Jämförs denna pseudokod med instruktionerna för den ungerska algoritmen, som mer är anpassad för matematiska problem, är det mycket enklare att följa under en eventuell implementationsfas.

5.2 Kriterium 2: Lösa problemet

Ungerska metoden. Problemet som ställdes upp i avsnitt 1 var som bekant tilldelningsproblemet; tilldela varje agent till exakt en uppgift så att den totala kostnaden blir minimal. Den ungerska metoden ger en optimal lösning på tilldelningsproblemet, vilket innebär att den alltid kommer ge den globalt billigaste lösningen. Kostnaden i *StarCraft II* är som bekant det euklidiska avståndet mellan agent och uppgift; något som ska vara minimalt för att i sin tur minimera tiden som agenter rör sig mellan uppgifter.

Agentbaserad girig algoritm. Till skillnad från den ungerska metoden ger den agentbaserade giriga algoritmen inte en garanti på optimal lösning. Präntare och Heintz påpekar i sin rapport att algoritmen bara letar lokalt optimala lösningar och bryr sig således inte om den globala kostnaden [3]. Vidare förklarar de att algoritmen kan ses som en bra lägre gräns och måttstock för optimala lösningar på tilldelningsproblemet. Det innebär alltså att den absolut bästa lösningen inte kommer väljas alla gånger, men lösningen som ges kan ändå ses som tillräcklig. Skillnaden i tid mellan en tillräcklig och en optimal lösning får ses som försumbar i de flesta fall, särskilt när körtiden är kort.

5.3 Kriterium 3: Spela bättre

Ungerska metoden. Den här lösningsmetoden kan absolut hjälpa agenten att bli bättre på *StarCraft*. Utan en specifik algoritm att följa kommer tilldelningen ske slumpmässigt, vilket kan orsaka i värsta fall att exempelvis en arbetare måste gå en lång sträcka för att ta sig till sin nya uppgift. Med den här algoritmen finns inte den risken; algoritmen ger ju till och med en optimal tilldelning. Eftersom spelet kan ses som en strategisk kapplöpning mot motståndaren är alla tidsmässiga övertag fördelaktiga, vilket ännu mer motiverar en applicering av ungerska algoritmen för att hjälpa agenten.

Agentbaserad girig algoritm. Även den här lösningsmetoden verkar förbättra agentens möjlighet till att prestera bättre i *StarCraft*. Nackdelen med att tillämpa en girig algoritm är som tidigare nämnts att den inte är optimal. Däremot ger en girig algoritm nästan alltid en bättre lösning än en slumpmässig tilldelning, så trots att den inte ger den bästa lösningen kommer den ändå kunna göra att agenten kan spela *StarCraft* bättre.

5.4 Kriterium 4: Säkerhet

Ungerska metoden. Något som är viktigt att komma ihåg är att den ungerska metoden ger en optimal tilldelning baserat på kostnadsfunktionen som används för att skapa kostnadsmatrisen. Med andra ord ger den en optimal tilldelning givet en viss kostnadsmatris. Det innebär att en olämplig prissättning på ett agent/uppgift-par kanske inte ger den bästa lösningen i spelet. Den valda kostnadsfunktionen för den här algoritmen var som sagt det euklidiska avståndet, vilket kan vara skadligt för agenten. Tänk om exempelvis det står ett stort hinder, som en lång mur, mellan agent och uppgift. Det euklidiska avståndet är givetvis litet, det är ju bara att gå över väggen. I *StarCraft* kan agenter inte gå över väggar, så den faktiska resevägen blir sannolikt mycket längre än det euklidiska avståndet. I sådana fall är den valda kostnadsfunktionen mindre bra vilket leder till att algoritmen är mindre säker att använda.

Agentbaserad girig algoritm. Den här algoritmen delar som bekant kostnadsfunktion med den ungerska metoden i den här utredningen, vilket ger den agentbaserade giriga algoritmen samma potential till misslyckande i de situationer där differensen mellan det euklidiska och det faktiska avståndet är stor. Det är inte alltid som det euklidiska ger en bra uppskattning på hur nära ett objekt är ett annat, åtminstone inte i miljöer där hinder förekommer. Således är inte heller den här metoden särskilt säker att implementera. För att öka säkerheten kan kostnadsfunktionen förbättras, bland annat genom att använda en annan sökalgoritm för att hitta den kortaste resvägen för en agent att ta sig till en uppgift i StarCraft.

6 Slutsats

Beroende på prioriteringar kommer svaret på vilken lösningsmetod som är att föredra att variera. Är den högsta prioriteringen att tilldelningen ska bli så bra som möjligt är den ungerska metoden helt klart att föredra. Detta nästan enbart grundat i dess garanti på optimal tilldelning. Är prioriteringen istället på att metoden ska vara lätt att förstå och applicera är den giriga algoritmen att föredra. I slutändan borde kriterium 3 vara det viktigaste vilket gör att den ungerska metoden borde ses som överlägsen då den garanterar en optimal tilldelning. En övrig slutsats som kan dras är att kostnadsfunktionen är otroligt viktig, oavsett vald lösningsmetod. Den ungerska algoritmen i kombination med en mer precis kostnadsfunktion än den som presenterats i denna rapport verkar alltså vara att föredra.

Något som är viktigt att erinra sig är att både ungerska metoden och den giriga algoritmen är kompletta. Något som dock skiljer dem åt är att ungerska metoden även är optimal, till skillnad från den giriga algoritmen. Frågan är dock hur bra den godtyckliga lösningen som en girig algoritm skulle generera faktiskt är. Hur stort blir övertaget för den agent som använder ungerska metoden gentemot en motståndare som tilldelar efter lokala optimum?

Del 2: Projektrapport

7 Faktisk användning i *StarCraft II*

Valet av tilldelningsalgoritm att implementera i Starcraft II var inte svårt alls. I basagenten fanns redan en implementation av en girig algoritm, vilket gjorde att jag valde att implementera ungerska algoritmen. Kostnadsfunktionen blev, precis som planerat, det euklidiska avståndet mellan uppgift och enhet. Det är som tidigare nämnts både vanligt förekommande inom andra projekt och utredningar samt en godtycklig och inte allt för krånglig uppskattning av hur långt en enhet måste gå för att anlända till sin tilltänkta uppgift. För att tackla problemet när listan på tillgängliga agenter inte är lika stor som listan på uppgifter valde jag två åtgärder. Om antalet agenter var större än antalet uppgifter lades *nollrader* till i kostnadsmatrisen, detta för att verkligen få med alla agenter och ge den optimala lösningen. Om det istället är antalet uppgifter som är större än antalet agenter minskar listan på uppgifter så att båda listorna blir lika stora. Detta är logiskt då det bara kan göras så många uppgifter som det finns agenter till. För att bestämma vilka uppgifter som inte följer med in i algoritmen sorteras dessa med avseende på uppgifternas respektive prioritet. I implementationen användes python-biblioteket *numpy* för att underlätta matrisoperationer som annars skulle krävt en hel del onödig komputationskraft.

8 Kopplingar till övrig implementation

Till skillnad från många andra gruppmedlemmars implementationer är min implementation relativt självständig. Indatan till algoritmen är byggnadsordningen, det vill säga den modul som bestämmer när och vad som ska byggas. Utdatan skickas inte till någon annan, av gruppen skriven, modul då uppgiftstilldelning inte riktigt skickas vidare. Istället görs tilldelningen baserat på sorteringen som den ungerska algoritmen gett direkt efteråt, i samma modul. Det skulle kunna diskuteras att möjligen påverkar algoritmens utdata exempelvis scoutingen, då denna modul ger en signal till scoutingmodulen att det är dags att sätta igång den sekvensen. I övrigt är det dock ingen annan modul som beror på tilldelningsalgoritmen.

Den ungerska metoden körs varje gång det finns uppgifter att göra och enheter att tilldelas dessa uppgifter. När det itereras över listan på uppgifter samlas alla enheter som har kompetens att utföra dessa. På så sätt väljs automatiskt endast exempelvis arbetare till att bygga, utan att blanda in marinsoldater eller *siege tanks*.

9 Resultat

Som tidigare nämnts har jag genomfört två implementationer av den ungerska algoritmen - både i och utanför StarCraft II. Detta gjordes för att lättare kunna testa den faktiska AI-teknikens prestanda, för att till slut kunna tillämpa en välfungerande algoritm i själva spelagenten. Angående implementationens resultat har jag kommit fram till två slutsatser. Den första är att själva algoritmen fick ett mycket satisfierbart resultat. Bland annat klarar den utan problem av att räkna ut 200 matriser med dimension 50x50 med slumpvalda värden. Den ger dessutom rätt svar på relativt kort tid, vilket jag ser som lyckat. Den andra slutsatsen må vara något pessimistisk. Integrationen av algoritmen i agenten blev inte riktigt lika bra som jag hade hoppats på. Det ursprungliga målet jag hade när projektet startade var att tilldela uppgifter till enheter snabbare för att förskaffa vår agent ett *tidsmässigt* övertag - ett övertag som verkar ha uteblivit. Mina tester i StarCraft II visade att den totala distansen alla enheter gick under 5 minuter i spelet minskade efter implementeringen av den ungerska metoden. I genomsnitt verkade det löst räknat som att det totalt minskade med cirka 200 distansenheter på de 5 minuter som testet pågick. Det som tyvärr gör att resultatet inte är lika satisfierbart som man kunde hoppats på är att dessa uträkningar tar tid - mer tid än för den giriga algoritmen. Nu blir en ytterligare fråga relevant: Vad är viktigast? Att det går snabbare eller att enheterna går kortare sträcka? Naturligtvis är svaret att tidsaspekten är den viktigaste. En annan fråga värd att ställa är om det är på grund av min tolkning av algoritmen som gör att effektiviteten minskar eller om det kan vara en annan faktor som påverkar, som exempelvis brist på komputationskraft i min dator. Oavsett vad anledningen till detta fenomen är kan det tyvärr konstateras att huvudmålet - att minska tiden det tar att tilldela uppgifter på - inte kunde nås trots implementering av en optimal algoritm. Kanske har man stirrat sig blind på funktionell optimalitet men missat tidsmässig optimalitet?

10 Egna slutsatser kring AI-tekniker

Att hitta, förstå och djupdyka i en ur ett personligt perspektiv okänd AI-teknik var väldigt spännande. Det gick snabbt att förstå hur den ungerska metoden löser matriser *för hand* - det som istället var krångligt var att översätta det tankesättet till programmeringskod. Människan kan ju exempelvis ganska snabbt se hur många linjer som behövs för att täcka alla nollor i steg 3 i den ungerska metoden men desto svårare är det att faktiskt ge konkreta och logiska instruktioner till en dator för att kunna göra likadant. En annan aspekt av projektet som försvårade arbetet något var integreringen i StarCraft II. Agenten uppvisade vid flertalet gånger märkliga beteenden - trots att implementeringen utanför StarCraft fungerade felfritt. Det ledde till att frågan om det uppkomna problemet ligger innanför kursens ramar, det vill säga om problemet faktiskt rör AI-tekniker överhuvudtaget eller om det är inom ett annat område det krånglar. Dessutom ger implementeringen i StarCraft inte något särskilt bra mått på hur bra implementeringen av den valda AI-tekniken faktiskt blev - åtminstone inte inom uppgiftstilldelning. När tekniken integreras med spelet adderas ytterligare faktorer som kan påverka teknikens prestanda. Ett bra exempel på det är naturligtvis alla uträkningar som måste göras i varje kallelse på *on_step*, som tar tid och prestanda att genomföra.

11 Insatser i gemensamt arbete

Då min implementation var relativt självständig har jag inte kunnat hjälpa till i den gemensamma implementationen i någon större utsträckning. Utöver konkret hjälp med den gemensamma implementationen har jag varit med i diskussioner kring andras implementationer och satt mig in i hur övriga moduler arbetar tillsammans.

12 Övrigt

Då dokumentationen gällande den ungerska metodens användning i programmering är något bristfällig har arbetet med att få fram en fullt fungerande algoritm blivit lite krångligt. Jag började med att lära mig algoritmen på papper, det vill säga att lösa tilldelning för hand. Olika källor gav olika direktiv på hur algoritmen fungerar, främst i de fall då det förekom mer komplexa kostnadsmatriser. Många onlinekalkylatorer lämpade för den ungerska metoden kunde inte lösa dessa matriser och det fanns begränsat med information om hur dessa skulle lösas. Det ledde till att jag fick implementera en egen tolkning av algoritmen för att lösa vissa fall där algoritmen fastnade i en evighetsloop. Exempelvis var det tillfällen då vissa nollor i matrisen inte blev täckta av linjer, vilket gjorde att det minsta icke-täckta värdet sattes till 0. Det i sin tur leder till att skiftningen av matrisen i steg 4 blir oförändrad. Ett annat exempel på när algoritmen fastnade var när en temporär linje drogs över den rad eller kolumn där den icke-täckta nollan var placerad. Det gjorde att algoritmen fastnade mellan 2–3 stadier som den inte kunde bryta sig loss från. Det löste jag genom att istället för att stryka ett temporärt streck över den raden eller kolumnen, så adderade jag det minsta icke-täckta värdet större än noll till det icke-täckta värdet. På så sätt modifierar man resterande värden på den kolumn eller rad när problemet hade uppstått, vilket metoden med temporärt streck inte klarade av.

Referenser

[1] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

[2] LeRoy Beasley, Young-Bae Jun, and Seok-Zun Song. Zero-term ranks of real matrices and their preservers. *Czechoslovak Mathematical Journal - CZECH MATH J*, 54:183–188, 03 2004.

[3] Fredrik Präntare and Fredrik Heintz. An anytime algorithm for optimal simultaneous coalition structure generation and assignment. 2020.

[4] Marcello D'Agostino. What's so special about euclidean distance? a characterization with applications to mobility and spatial voting. *Social Choice and Welfare*, 33:211–233, 08 2009.