Algoritmselektion genom Support Vector Machines för General Video Game Artificial Intelligence Competition

Susanna Dahlgren
susda272@student.liu.se
Linköpings Universitet

Viktor Holmgren
vikho394@student.liu.se
Linköpings Universitet

Elin Larsson
elila927@student.liu.se
Linköpings Universitet

Yousif Touma
youto814@student.liu.se
Linköpings Universitet

Fredrik Wallström
frewa814@student.liu.se
Linköpings Universitet

George Yildiz
geoyi478@student.liu.se
Linköpings Universitet

Sammanfattning

I den här rapporten undersöker vi hur en algoritmselekterande agent står sig jämfört med en icke selekterande agent i tävlingen General Video Game Artificial Intelligence Competition (GVG-AI). I undersökningen har en portfölj använts med fyra algoritmer som tidigare presterat bra i GVG-AI. 80 spel har använts för att träna en klassificerare. Med hjälp av klassificeraren har vi lyckats skapa en algoritmselekterande agent som presterar bättre än varje enskild algoritm. Agenten har tre fler vinster än den näst bästa agenten. Vi tror att ett bättre resultat skulle kunna uppnås genom att bland annat ha mer träningsdata tillhands.

Inledning

GVG-AI är en tävling inom AI och beslutsfattande som används som en plattform för att jämföra och utvärdera olika algoritmer för just beslutsfattning och planering. Det som går att se är att de tävlandes algoritmer ofta är bra på olika spel vilket leder en till att tro att underliggande struktur i spelen avgör huruvida en viss algoritm är bäst lämpad. I denna rapport har vi undersökt möjligheten att applicera algoritmselektion för att få ett bättre resultat än den enskilt bästa algoritmen.

Vi ser också hur vi rör oss allt mer mot generell AI. Mot agenter som inte bara klarar av en stor mängd olika typer av uppgifter, utan som gör det utan att skaparen har designat dem med specifik domänkunskap för de områden som de agerar i.

En metod för att konstruera dessa generella agenter är att kombinera ett antal olika typer av agenter in i en slags superagent som sedan väljer bland dessa delagenter baserat på probleminstansen. Denna metod kallas algoritmselektion och bygger på att använda algoritmer som kompletterar varandra och sedan välja bland dem på ett sådant sätt att styrkorna används medan svagheterna undviks.

Mål

Målet med det här projektet är att implementera en generell spelagent som utnyttjar algoritmselektion för att spela flera olika okända 2D arkadspel med realtidskrav på beslutsfattandet. Vidare är målet att vår algoritmselekterande agent, hädan efter även kallad Copybot, på ett antal för agenten okända ospelade spel, skall prestera bättre än den enskilt bästa algoritmen.

Syfte

Syftet är att undersöka huruvida en portfölj av planeringsalgoritmer tillsammans med algoritmselektion leder till ett genomsnittligt förbättrat resultat i jämförelse med en agent som inte använder sig av selektion.

Bakgrund

Algoritmselektion

Algoritmselektion handlar om att välja en algoritm ur en portfölj av olika algoritmer med varierande prestanda. Definitionen lyder som följer:
Givet

- en mängd I av probleminstanser från en distribution D,
- en algoritmrymd A, och
- ett prestandamått m; $I \times A \rightarrow R$,

så är syftet att hitta en avbildning s: $I \rightarrow A$ som optimerar $\mathbf{E}_{i \sim D}(i, s(i))$, det vill säga det förväntade prestandamåttet för instanser i distribuerade enligt D, genom att köra den valda algoritmen $s(i)$ för instansen i. I praktiken så implementeras avbildningen s vanligtvis med hjälp av karaktäriseringar av instanserna $i \in I$, även kallat instansfunktioner. Dessa instansfunktioner avbildas sedan till en algoritm med hjälp utav maskininlärningstekniker. Detta medför dock ökade beräkningskostnader som bör tas i beaktning i prestandamåttet m. Det finns många olika sätt att hantera algoritmselektion och i dagsläget baseras nästan alla metoder på någon form utav maskininlärning. Antingen kan en enkel modell användas eller en mer komplex kombination av flera modeller som, givet en probleminstans att lösa, bestämmer vilken algoritm eller vilken kombination av algoritmer som ska användas.

Det mest naturliga skulle vara att välja enbart en algoritm för att lösa en given probleminstans, men den stora nackdelen med den metoden är att det inte finns något sätt att åtgärda ett dåligt val. Systemet kan inte återhämta sig om den valda algoritmen utför ett dåligt jobb på det givna problemet. Det alternativa valet skulle istället vara att, utifrån portföljen av algoritmer, skapa ett schema som bestämmer

ordning och tidsåtgång för alla eller en del av algoritmerna. Vanligtvis baseras detta schema på den förväntade prestationen för varje algoritm.

En fråga som kan uppstå är huruvida man ska starta om en algoritm och i så fall vid vilket tillfälle. Istället för att utföra algoritmselektion enbart en gång innan ett problem ska lösas så kan man utföra det upprepade gånger under den tid det tar att komma fram till lösningen för problemet. På detta sätt kan man dra nytta av den information som uppdagar sig medan algoritmen exekveras. Sådana metoder övervakar exekveringen av den eller de valda algoritmerna och agerar om utförandet avviker från det förväntade, eller utför upprepade selektioner för delproblem av den givna instansen (Bischl et al. 2016).

Support Vector Machine

En stödvektormaskin (SVM) är en typ av klassificerare, som vi har valt för att klassificera olika typer av spel. SVM anses vara en lämplig metod att använda när man inte har stor kunskap om domänen. Klassificeringen går ut på att dela in indatat i olika klasser genom att skapa en linjär separerare mellan klasserna. I vårt fall kommer den linjära separeraren att klassificera de olika spelen, beroende på vad spelen har för egenskaper. Separeraren genereras genom att en SVM maximerar marginalen till respektive klass. Detta gör den genom att välja de mest extrema punkterna för varje klass och skapa en marginal mellan dem för att sedan skapa separeraren i mitten utav marginalen. Det är dock inte alltid möjligt att separera data linjärt. Det man gör då är att utöka indatavektorn till en högre dimension. Detta leder till att det finns en linjär separerare för indatat i någon sådan högre dimension. Den kan sedan översättas tillbaka till en olinjär separerare i ursprungsdimensionen (Russell et al. 2003).

LibSVM - SVM Bibliotek

LibSVM är ett bibliotek som är tillgängligt i Java och som vi har använt oss av i studien. LibSVM hjälper oss att enkelt applicera funktionaliteten hos en SVM som beskrivits ovan. LibSVM har en hög popularitet inom maskininlärning och ger stöd åt flera SVM formuleringar för klassificering, regression (skapa en funktion som passar data bäst) och fördelningsuppskattning (uppskatta den optimala utlovade lösningen utifrån urvalen).

LibSVM använder sig av filer för träning, validering samt klassificering. För att träna en klassificerare skickar man in en textfil med dels en etikett, vilket motsvarar klass, och dels en egenskapsvektor där man etiketterar egenskaper samt ger dem ett reellt värde. Som svar får man en modellfil som sedan kan användas till att validera modellen med en valideringsfil på samma format som träningsfilen. På så vis kan man få reda på hur bra modellen är. Den kan förstås även användas till att klassificera en egenskapsvektor. I båda dessa fall kan man få reda på dels vilken klass klassificeraren väljer men även sannolikhetsfördelningen över sina klasser för den egenskapsvektorn (Chang and Lin 2011).

General Video Game AI Competition

General Video Game AI (GVG-AI) Competition handlar om att implementera en agent som ska klara av att spela en mängd olika spel utan någon förkunskap om själva spelet i sig. Det finns tre olika kategorier att delta i: single-player planning track, 2-player planning track och level generation track. I denna rapport har vi utgått ifrån single-player planning track.

GVG-AI använder sig utav Video Game Description Language (VGDL) för att beskriva spelen på ett enkelt och koncist sätt. Varje spel kräver enbart några dussin rader text som är uppdelat i fyra sektioner; spritemängd som beskriver alla tillgängliga sprites i spelet och deras parametrar, nivåavbildningen som beskriver relationer mellan olika karaktärer, interaktionsmängd som beskriver vad som händer om två sprites i ett givet spel kolliderar samt avslutningsmängd som beskriver sluttillståndet för spelet och även om agenten vinner eller förlorar. Ramverket som används under tävlingen är Java-vgdl och det klarar av att ladda spel och nivåer skrivna i VGDL. VGDL-beskrivingen av spelen är inte tillgängliga för agenten utan det är upp till agenten själv att lista ut spelmiljön och vad som behöver göras för att vinna. Till hjälp finns ett tillståndsobjekt som dels kan användas för att få information om spelets nuvarande tillstånd men också genom att simulera handlingar för att få ut ett förväntat framtida tillstånd. Utöver detta finns det två restriktioner som måste följas varje spelomgång för att agenten inte ska diskvalificeras. Konstruktorn, som kallas en gång per spel, har endast 1 sekund på sig och metoden *act*, som kallas i varje spelsekvens, har endast 40 millisekunder på sig att bestämma agentens nästa handling.

Deltagarna i tävlingen har tillgång till en mängd spel att träna sin agent på och varje spel består av fem olika nivåer. Därtill finns en tävlingsserver som under tävlingens gång tar emot bidrag och utvärderar dom på ytterligare 10 okända spel som kallas för valideringsmängd. Deltagarna kan skicka in sina bidrag flertalet gånger till denna server och en topplista produceras på webbsidan baserat på resultaten. Detta gör att dom tävlande kan köra och utvärdera sina agenter på okända spel, och dessa spel offentliggörs sedan efter tävlingens slut. Den tredje och sista mängden innehåller även den tio okända spel och används för att fastställa den slutgiltiga rankingen av deltagarna. Varje agent körs på alla tio spel, tio gånger per nivå (fem nivåer per spel) och för varje spel räknas antal vinster, totala summan av poäng samt den totala tiden det tog att spela. Agenterna rankas baserat på dessa kriterier där antalet vinster först tas i beaktning. Om två agenter har lika antal vinster så vägs totala summan av poäng in och därefter den totala tiden för att spela om agenterna fortfarande måste särskiljas. När alla agenter är rankade för varje spel så tilldelas poäng baserat på deras position. Summan av alla dessa poäng fastställer sedan vem som vinner tävlingen (Perez et al. 2015) (Perez-Liebana et al. 2016).

Metod

Implementationen av selektionen delades upp i tre steg. I steg ett togs en portfölj fram med algoritmer som kompletterar varandra. I steg två identifierades egenskaper i spelen för att ta fram en klassificerare för olika typer av spel. I sista steget implementerades algoritmselektionen med hjälp av den framtagna portföljen och klassificeraren. Här valde vi

att utöka tidsbegränsningen för konstruktorn till sammanlagt sju sekunder. Sex sekunder till insamling av egenskaper plus en sekund till att initiera den valda algoritmen.

Portfölj

Projektet fokuserar på optimering av en beslutsfattande agent med hjälp av selektion. Vi valde därför att använda oss av färdiga algoritmer i portföljen och istället fördjupa oss i själva selektionen.

Vid framtagandet av portföljen började vi med att gå igenom tidigare års vinnare och vad de hade använt för algoritmer. Det var vissa svårigheter i att få tag på inte bara rapporter kring deras arbeten utan även källkoden för deras agenter. Detta resulterade i det relativt begränsade antalet algoritmer som vi sedan analyserade i vår portfölj. De algoritmer som analyserades var MaastCTS2, NovTea, Return42, YBCriber, YOLOBOT, adrienctx samt thorbjrn.

Efter insamling av algoritmer påbörjades arbetet att analysera vilka algoritmer som kompletterade varandra för att ge oss störst räckvidd men samtidigt med så minimalt antal algoritmer som möjligt. Algoritmerna lades in i vårt program och vi skrev en algoritm som automatiserat för alla algoritmerna körde igenom våra träningsspel, på fem nivåer, och fem gånger för varje nivå. Antalet vinster för varje spel återfinns i Tabell 5.

Utifrån resultatet kan man se att vissa algoritmer är bra på vissa spel medan andra är bra på andra spel. Olika algoritmer verkar till synes vara bra på lite olika typer av spel. Genom att utnyttja vetskapen om detta valde vi ut en delmängd som kompletterar varandra genom att prestera bra på olika spel. Dessa algoritmer utgjorde vår slutliga portfölj som selektionen använder sig av.

Selektion

Syftet med selektionen är att för ett givet spel välja den algoritm som är bäst lämpad. Vi valde att använda *supervised learning*, vilket betyder att inlärningen sker genom att träna systemet med färdiga indata-utdata par. För varje spel tog vi ut en mängd av egenskaper, vilka sedan matchades mot den bästa algoritmen för det spelet. Detta bildade de indata-utdata par som användes för att träna modellen. För att kunna utvärdera selektionen delade vi upp spelen som fanns att tillgå i en träningsmängd och valideringsmängd. Träningsmängden bestod av 80 spel och valideringsmängden av 12 spel.

Vi började med att undersöka vilka egenskaper vi har åtkomst till. Det vill säga, vilka metoder vi kunde använda oss av i GVG-AI ramverket för att plocka ut egenskaper. Här menar vi inte nödvändigtvis bara de egenskaper som kan fås genom att observera starttillståndet av spelet, utan även de egenskaper som kan upptäckas under spelets gång. En *featureCollectionAgent* skapades för att samla in egenskaper hos spelet. Agenten bygger på *sampleOneStepLookAhead* som simulerar framåt i spelet och observerar speltillståndet i varje steg. Utifrån dessa data kan sedan fler egenskaper eventuellt upptäckas. Vi tog även hjälp av (Mendes, Nealen, and Togelius 2016) forskning för att se vilka egenskaper vi eventuellt har åtkomst till. De resulterande egenskaperna kan ses i Tabell 3.

Det fanns föraningar om att vissa egenskaper var redundanta. Vi valde då att undersöka alla delmängder av egenskaperna för att hitta de delmängder med bäst resultat. Antalet delmängder i detta fall var endast 2^{16} så valet gjordes att använda bruteforce för att generera och testa alla möjliga delmängder. Från detta valdes en av de framtagna delmängderna som den slutgiltiga egenskapsuppsättningen vilken återfinns i Tabell 4.

Utvärdering

Som beskrivet ovan så delades de tillgängliga spelen in i två disjunkta delmängder; en träningsmängd och en valideringsmängd. För att utvärdera hur väl algoritmselektionen fungerar så jämför vi Copybot mot den enskilt bästa algoritmen i portföljen. Vi jämför även dessa två agenter mot den *optimala* algoritmselektionensagenten, vilket vi definierar som den agent som alltid väljer den bästa agenten ur portföljen för ett givet spel. På detta sätt så kunde selektionen isoleras som den enda faktorn vid utvärderingen. Jämförelsen mellan agenterna gjordes på valideringsmängden. Spelen spelades fem gånger på varje nivå.

Resultat

Klassificeraren

Ett delresultat är hur klassificeraren väljer algoritm utifrån en given egenskapsvektor. I och med att vi har spelat spelen i valideringsmängden med samtliga algoritmer i portföljen vet vi vilken som är det optimala valet. Med hjälp av modellen samt egenskapsvektorerna för spelen i valideringsmängden kunde vi få ut vilka algoritmer som väljs. Vi kan även se klassificerarens uppskattade sannolikhet för att respektive algoritm skulle vara rätt val i varje fall. Resultatet redovisas i Tabell 1.

Copybot

Från resultaten på träningsmängden som återfås i Tabell 5 fås att YOLOBOT är den enskilt bästa algoritmen och därmed den som ingår i jämförelsen. I Tabell 2 ses resultaten på valideringsmängden för alla de algoritmer som ingår i jämförelsen. Här ser vi att Copybot vinner 11 gånger fler än vad YOLOBOT gör, vilket motsvarar 6% fler vinster. Vidare ser vi att en optimal algoritmselekterande agent vinner 239 gånger av 300 möjliga, vilket motsvarar 34% fler vinster än YOLOBOT. Fulltständiga resultat över samtliga algoritmer samt individuella spel återfinns i Tabell 6

Diskussion

Det slutgiltiga resultatet av Copybot uppnådde vårt mål, det vill säga, att Copybot presterade bättre än alla enskilda algoritmer ingående i dess portfölj. Resultatet är dock inte fullt lika bra som vi hade hoppats på när det kommer till jämförelsen mot den optimala algoritmselekterande agenten. Den optimala selektionsagenten vinner 50 gånger fler än vad Copybot gör vilket kan bero på flertalet faktorer.

En av dessa faktorer som vi tror kan ha påverkat är mängden träningsdata. Ett alternativ är att vi skulle behandla

varje nivå i varje spel som ett individuellt spel. Detta leder till 400 träningsexempel, 80 spel med vardera fem nivåer istället för ett exempel per spel. Vi märkte dock att egenskapsvektorerna för olika nivåer i samma spel gav snarlika resultat vilket innebär att träningsexemplena inte skulle tillföra klassificeraren något.

Som tidigare nämnt tilldelas sex sekunder till att hitta egenskaper för ett spel samt välja vilken algoritm som ska spela spelet. Anledningen till att sex sekunder valdes beror på att det bör vara tillräckligt med tid för att hitta representativa värden på egenskaperna. Majoriteten av egenskaperna kan fås redan i första tillståndet innan spelet avanceras. Detta gäller dock inte alla egenskaper. Till exempel kan information angående antalet portaler variera mellan startläget och senare tillstånd. Av den anledningen måste vi simulera drag för att kunna få ett representativt svar angående hur många portaler det finns. Huruvida vi hade fått andra egenskapsvektorer om vi hade haft en annan tidsgräns för att plocka ut egenskaper är osäkert. Vi använde sampleOneStepLookAhead för att simulera spelet vilket är en algoritm som under sex sekunder hinner med många iterationer. Vi tror därmed inte att en längre tid hade gjort skillnad, däremot skulle en kortare tid varit intressant att utforska.

En annan möjlighet är att man förutom att samla egenskaper innan man börjar spela även samlar egenskaper under spelets gång för att dynamiskt kunna välja algoritm. Man skulle kunna tänka sig att det är lämpligt i spel där miljön eller förutsättningarna förändras vartefter spelet fortgår. Vi tror dock att det finns begränsningar i detta då vi skulle behöva ta tid från den algoritm som senare ska användas till att hitta egenskaper och be klassificeraren om ett svar. Många bra algoritmer använder sig även av en kunskapsbank som byggs upp under spelets gång vilket är en anledning till att de presterar så bra som de gör. Denna kunskapsbank kommer dels kunna vara rent felaktig då information går förlorad när en algoritm byts ut men även göra att en inbytt algoritm inte alls presterar som tänkt då dess kunskapsbank är tom trots att spelet potentiellt fortgått under en längre tid.

Ett annat potentiellt problem är överlappande algoritmer. Syftet med en supervised learning algoritm som SVM är att lära sig en funktion som avbildar indata på utdata. I vårt fall, egenskapsvektorer på rätt algoritmklass. Om det dock visar sig att två spel med samma indata har olika utdata får SVM:en konflikterande exempel. Om samma indata ger olika utdata finns det ingen funktion som avbildar detta, utan det behövs alltså någon form av slump inblandat. Detta scenario kan uppstå om vi har algoritmer som är väldigt lika i vad de är bra på men med små variationer.

Indelning av de tillgängliga spelen till en träningsmängd och valideringsmängd gjordes genom att ordna spelen alfabetiskt och låta de 80 första utgöra träning och de tolv sista utgöra validering. I resultatet visades att YOLOBOT var bäst under träningsspelen medan NovTea var bäst under valideringsspelen. Eftersom vi inte har haft mycket träningsdata kommer YOLOBOT att vara den dominerande algoritm som väljs, även under valideringen. Detta är olyckligt då det inte är den algoritm som bör väljas.

För att få ett bättre mått på hur bra klassificeraren är skulle vi kunna ha använt korsvalidering. Då partitionerar man tränings- och valideringsdata på olika sätt och beräknar för varje sådan partitionering hur bra klassificeraren är. Ett genomsnitt används sedan för att få ett mer korrekt värde på korrektheten hos klassificeraren (Arlot, Celisse, and others 2010).

Bortsett från förbättringsmöjligheterna är vi nöjda med arbetet som bevisar att vi med en klassificerare kan nå ett bättre resultat än de enskilt bästa algoritmerna.

Referenser

[Arlot, Celisse, and others 2010] Arlot, S.; Celisse, A.; et al. 2010. A survey of cross-validation procedures for model selection. *Statistics surveys* 4:40–79.

[Bischl et al. 2016] Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Fréchette, A.; Hoos, H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; et al. 2016. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence* 237:41–58.

[Chang and Lin 2011] Chang, C.-C., and Lin, C.-J. 2011. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2(3):27.

[Mendes, Nealen, and Togelius 2016] Mendes, A.; Nealen, A.; and Togelius, J. 2016. Hyperheuristic general video game playing. *Proceedings of Computational Intelligence and Games (CIG). IEEE*.

[Perez et al. 2015] Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015. The 2014 general video game playing competition.

[Perez-Liebana et al. 2016] Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Lucas, S. M.; and Schaul, T. 2016. General video game ai: Competition, challenges and opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[Russell et al. 2003] Russell, S. J.; Norvig, P.; Canny, J. F.; Malik, J. M.; and Edwards, D. D. 2003. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River.

Optimalt val	Sannolikhet (optimal agent)	Vald agent	Sannolikhet (vald agent)
YBCriber	14.4%	YOLOBOT	76.1%
YBCriber	50.1%	YBCriber	50.1%
YBCriber	44.2%	YBCriber	44.2%
thrbjrn	7.8%	YOLOBOT	58.0%
YBCriber	22.4%	YOLOBOT	58.0%
thrbjrn	7.8%	YOLOBOT	58.0%
thrbjrn	7.8%	YOLOBOT	58.0%
YOLOBOT	76.8%	YOLOBOT	76.8%
thrbjrn	13.9%	YOLOBOT	58.6%
NovTea	6.3%	YOLOBOT	76.1%
YBCriber	48.3%	YBCriber	48.3%
YBCriber	31.7%	YBCriber	31.7%

Tabell 1: Valideringsomgång med klassificeraren

Agent	Totala antal vinster
Optimal selektion	239
Copybot	189
NovTea	186
YOLOBOT	178
thorbjrn	178
YBCriber	155

Tabell 2: Slutresultat

Egenskapsgrupp	Egenskaper
Resurser	Om det finns resurser Om spelaren har resurser Antal olika typer av resurser Antal olika typer av resurser för spelare
NPC	Antal olika typer av NPCs Antal NPCs
Dimension	Area Blockstorlek (antal pixlar i ett block)
Sprites	Antal olika typer av orörliga sprites Antal olika typer av rörliga sprites Antal olika typer av sprites skapade av spelare
Portaler	Antalet portaler
Handlingar	Om vi kan röra oss upp eller ner (vertikalt) Om vi har tillgång till USE handlingen Om vi kan skjuta

Tabell 3: Alla identifierade egenskaper

Egenskapsgrupp	Egenskaper
Resurser	Antal olika typer av resurser
NPC	Antal olika typer av NPCs
Dimension	Blockstorlek (antal pixlar i ett block)
Portaler	Antalet portaler
Handlingar	Om vi kan röra oss upp eller ner (vertikalt) Om vi har tillgång till USE handlingen

Tabell 4: Slutgiltiga egenskaper

	MaastCTS2	NovTea	Return42	YBCriber	YOLOBOT	adrienctx	thorbjrn
Spel 1	25	25	24	25	25	25	25
Spel 2	0	0	2	0	0	1	1
Spel 3	0	0	0	0	0	0	0
Spel 4	17	8	9	0	12	19	1
Spel 5	15	15	9	14	20	2	10
Spel 6	1	1	1	0	0	1	0
Spel 7	1	0	0	0	2	2	4
Spel 8	0	0	0	0	0	0	0
Spel 9	12	19	8	1	7	4	2
Spel 10	2	2	4	0	4	1	1
Spel 11	14	4	13	1	10	18	15
Spel 12	11	9	16	1	12	19	21
Spel 13	0	9	7	5	8	0	5
Spel 14	25	25	25	25	25	25	25
Spel 15	0	0	2	0	5	7	0
Spel 16	25	13	15	25	25	5	12
Spel 17	9	1	2	12	25	3	0
Spel 18	0	0	0	0	0	0	0
Spel 19	12	11	10	6	14	14	9
Spel 20	12	5	13	13	16	7	0
Spel 21	4	5	1	6	3	0	0
Spel 22	10	20	7	11	0	8	0
Spel 23	25	0	18	0	13	0	10
Spel 24	5	7	4	5	0	0	0
Spel 25	9	0	4	25	25	0	0
Spel 26	21	0	6	8	25	25	9
Spel 27	0	0	1	0	0	0	0
Spel 28	24	20	23	22	24	10	24
Spel 29	0	0	0	0	1	0	0
Spel 30	0	1	10	0	8	1	7
Spel 31	2	1	2	0	4	7	0
Spel 32	2	19	13	21	12	0	21
Spel 33	0	0	1	13	0	1	2
Spel 34	25	25	19	25	15	15	20
Spel 35	23	24	22	25	19	22	23
Spel 36	2	3	0	5	2	0	0
Spel 37	0	0	0	3	0	0	0
Spel 38	8	14	1	22	25	5	9
Spel 39	0	0	0	0	0	0	0
Spel 40	17	16	14	11	21	3	16
Spel 41	0	0	0	0	7	0	0
Spel 42	4	1	9	0	6	0	2
Spel 43	25	16	24	25	20	25	18
Spel 44	25	25	25	23	25	17	25
Spel 45	21	21	0	16	18	19	15
Spel 46	25	17	25	25	25	25	24
Spel 47	25	25	25	25	25	7	25
Spel 48	10	0	12	11	13	0	0
Spel 49	12	12	1	9	6	4	25

Spel 50	1	0	8	0	0	1	9
Spel 51	25	25	25	25	25	25	25
Spel 52	25	20	20	25	25	20	25
Spel 53	0	0	0	0	0	0	0
Spel 54	0	0	0	0	0	0	0
Spel 55	0	0	0	0	0	0	0
Spel 56	25	22	24	20	22	10	23
Spel 57	10	20	8	24	20	5	15
Spel 58	25	25	24	22	9	19	21
Spel 59	0	0	0	0	0	0	0
Spel 60	25	25	24	25	5	25	25
Spel 61	18	18	14	12	17	13	15
Spel 62	0	1	1	0	6	0	0
Spel 63	25	10	23	19	25	12	24
Spel 64	14	16	16	22	3	24	10
Spel 65	5	5	2	5	1	5	5
Spel 66	20	16	3	18	13	6	14
Spel 67	0	0	1	0	0	0	0
Spel 68	1	1	0	0	12	1	10
Spel 69	0	0	0	0	0	0	0
Spel 70	16	1	0	0	2	0	0
Spel 71	6	0	15	0	10	14	10
Spel 72	7	0	1	13	6	1	7
Spel 73	16	6	18	14	11	13	21
Spel 74	25	10	18	24	25	25	25
Spel 75	25	25	25	25	25	25	25
Spel 76	13	20	15	23	20	7	15
Spel 77	0	2	6	0	3	0	0
Spel 78	0	12	20	23	8	0	2
Spel 79	25	25	23	25	25	18	25
Spel 80	14	10	13	14	11	10	9
TOTALT	**871**	**734**	**774**	**842**	**881**	**626**	**766**

Tabell 5: Resultat av algoritmer över träningsmängden

	Copybot	NovTea	YOLOBOT	thorbjrn
Spel 1	5	5	5	3
Spel 2	4	2	3	2
Spel 3	0	1	0	0
Spel 4	4	5	4	5
Spel 5	2	2	3	3
Spel 6	5	4	5	5
Spel 7	3	5	4	5
Spel 8	3	1	3	4
Spel 9	5	3	3	4
Spel 10	2	3	2	1
Spel 11	5	5	4	5
Spel 12	1	2	1	0
Spel 13	5	5	5	3
Spel 14	3	3	3	2
Spel 15	0	1	0	0
Spel 16	4	5	2	5
Spel 17	2	1	3	3
Spel 18	5	3	4	4
Spel 19	4	5	4	5
Spel 20	3	1	5	2
Spel 21	4	5	4	4
Spel 22	2	3	2	1
Spel 23	5	5	5	5
Spel 24	1	1	1	0
Spel 25	5	5	5	2
Spel 26	3	1	3	2
Spel 27	0	1	0	0
Spel 28	4	5	3	5
Spel 29	2	1	3	3
Spel 30	5	5	3	4
Spel 31	4	5	4	5
Spel 32	4	1	4	4
Spel 33	4	2	4	3
Spel 34	2	3	2	1
Spel 35	4	5	5	5
Spel 36	1	1	1	0
Spel 37	5	4	5	3
Spel 38	4	2	3	2
Spel 39	0	1	0	0
Spel 40	4	5	2	5
Spel 41	2	2	3	3
Spel 42	3	3	2	5
Spel 43	3	5	4	5
Spel 44	2	2	3	5
Spel 45	2	4	4	5
Spel 46	2	3	2	1
Spel 47	5	5	5	5
Spel 48	1	2	1	0
Spel 49	5	5	5	3
Spel 50	4	3	3	2
Spel 51	0	1	0	0
Spel 52	4	5	3	5

Spel 53	2	1	3	3
Spel 54	5	2	4	4
Spel 55	5	5	3	5
Spel 56	4	3	2	3
Spel 57	5	3	2	3
Spel 58	2	2	2	1
Spel 59	4	5	4	5
Spel 60	1	2	1	0
TOTALT	189	186	178	178

Tabell 6: Resultat av algoritmer över valideringsmängden