Examensarbete

HÖGSKOLAN I SKÖVDE

EN SCHACK AI BASERAD PÅ CASE-BASED REASONING

A CASE-BASED REASONING APPROACH TO A CHESS AI

Examensarbete inom huvudområdet Datavetenskap
Grundnivå 30 högskolepoäng
Vårtermin 2015

Johannes Qvarford

Handledare: Peter Sjöberg
Examinator: Anders Dahlbom

Innehållsförteckning

1 Introduktion

(Nytt intro)

Schack är ett spel som fått mycket uppmärksamhet inom forskningsområdet artificiell intelligens (AI). Redan på 50-talet presenterade Shannon (1950) ett förslag på hur en schackspelande AI-agent kunde fungera och sedan dess har många forskningsarbeten dedikerats till att förbättra hans ursprungliga design. Trots alla möjliga förbättringar som presenterats genom åren har den grundläggande tekniken som Shannon föreslog dock förblivit den samma och få utförliga arbeten om alternativa tillvägagångsätt har presenterats (Schaeffer 1991). AI-agenten undersöker vilket drag som är bäst genom att internt utföra alla möjliga kombinationer av nästa x antal drag, gradera de resulterade lägena efter en heuristik och utvärdera vilket drag som leder till det bästa läget, givet att motståndaren spelar optimalt.

I det här arbetet appliceras en alternativ teknik för att utveckla en schackspelande AI-agent i hopp om att gynna forskningen om schack-AI. Tekniken heter *Case-based Reasoning* (CBR) och är en problemlösningsteknik som går ut på att basera lösningar på nya problem, på lösningar från tidigare, liknande problem. Ett problem tillsammans med sin lösning kallas för ett fall. För att AI-agenten ska kunna ha några tidigare problem med lösningar att hänvisa till, samlas information in t.ex. genom att låta en mänsklig eller artificiell expert lösa ett antal exempelproblem. Problemen med expertens lösningar lagras i en fallbas som AI-agenten kan hänvisa till när den behöver lösa ett problem. Kopplat till schack kan ett problem vara ett läge och en lösning det drag som ska utföras i läget.

I det här arbetet ska lämpligheten att använda CBR för att utveckla schackspelande AI-agenter utvärderas genom att skapa ett program av en CBR-baserad schackspelande AI-agent. AI-agenten kommer kunna basera sin fallbas på partier spelade av olika schackspelare. Det ska undersökas om AI-agenten spelar bättre med fallbaser baserade på partier spelade av bättre experter för att avgöra om en experts rankning kan överföras till skicklighet för AI-agenten. Undersökningen kommer utföras genom att låta AI-agenten spela mot sig själv flera gånger med olika fallbaser och notera hur många partier som AI-agenten vinner med respektive fallbaser.

2 Bakgrund

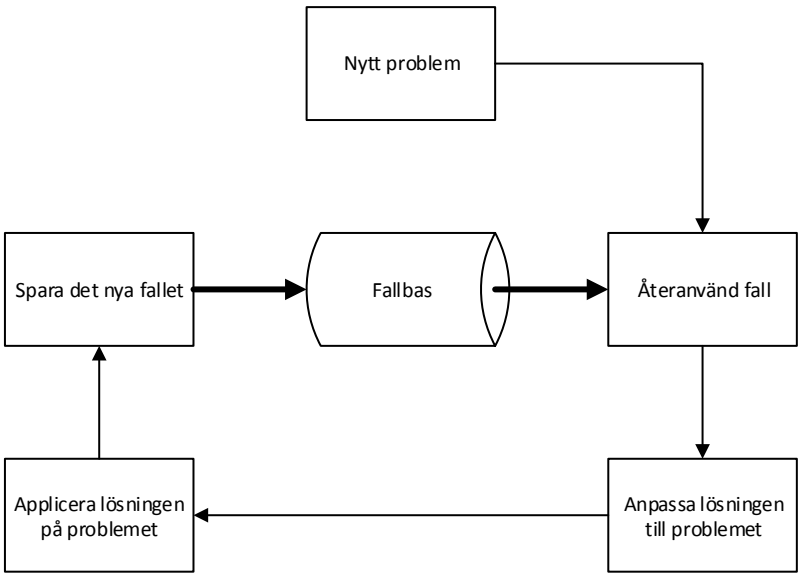
(Nytt)

I denna sektion presenteras bakgrundsinformation om ämnen och termer som nämns i arbetet. I sektion 2.1 presenteras CBR och hur arbetet bygger på tidigare forskning kring CBR. Sektion 2.2 innehåller reglerna till schack, termer som ofta används inom schack och tidigare arbeten om alternativa tillvägagångssätt till schack-AI.

2.1 Case-based Reasoning

(Nytt utökat material nedan. Många av mina påstående är grundade i Richter och Webers bok, och jag vet inte om jag konstant ska referera till dem, eller på nått sätt indikera att hela sektionen i stort sätt indirekt är hämtat därifrån.)

CBR är en teknik för problemlösning inom AI som är baserad på idén att använda lösningar på tidigare, liknande problem (Richter & Weber 2013). För att använda CBR måste en samling testfall med problem och deras respektive lösningar samlas in. Hur testfallen samlas in varierar och kan ske genom observation av artificiella eller mänskliga experter. Lösningarna på problemen från experten behöver inte nödvändigtvis uppfylla några korrekthetskrav. Ett problem tillsammans med sin lösning beskrivs som ett fall, och en grupp fall kallas för en fallbas. När ett nytt problem ska lösas kan AI-agentens lösning baseras på hur en expert löste problemet eller ett liknande problem genom att konsultera en fallbas. När det fall har hittats vars problem är mest likt det nya problemet, kan fallets lösning anpassas till det nya problemet. Lösningen kan sedan appliceras. Tillsammans bildar det nya problemet med dess anpassade lösning ett nytt fall, som kan inkluderas i fallbasen och återanvändas. Denna process illustreras i Figur 1.



Figur 1 Figur över processen för att applicera CBR.

2.1.1 Representation

Problem kan representeras på olika sätt inom CBR, och de passar olika bra beroende på vad för sorts problem som ska lösas. En vanlig representation är en tupel av värden, där varje

element överensstämmer med ett attribut (Richter & Weber 2013, s. 93). Ett värde kan vara sammansatt av flera värden, och kan nästlas till ett arbiträrt djup. Till exempel kan ett värde vara en tupel, vars första element i sin tur kan vara en tupel av ett antal mängder osv.. Som exempel kan det finnas en AI-agent utvecklat för att estimera priset på begagnade bilar baserat på priset av tidigare sålda begagnade bilar. En bil kan ha flera attribut men vissa attribut kan vara mer relevanta för att identifiera liknande, tidigare sålda bilar. T.ex. kan en bil representeras med attributen modell, tillverkningsår och mätarställning men sakna attributet färg, för att en bils färg inte har haft en märkbar påverkan på en bils pris tidigare. Representationen illustreras i Figur 2. Andra möjliga representationer av problem är bilder, text och ljud (Richter & Weber 2013, ss. 89-90). Om det visar sig att bilar som ser liknande ut, har liknande beskrivningar eller låter liknande har en större chans att säljas för liknande pris, så kan det vara en lämplig problemrepresentation.

$$(Modell, Tillverkningsår, mätarställning)$$

$$(Audi\ A7, 2012, 17203\ mil)$$

$$(Bmw\ 320d, 2013, 707\ mil)$$

Figur 2 Ett exempel på hur en begagnad bil kan representeras som ett problem i CBR och två exempel på begagnade bilar.

2.1.2 Liknelse

Richter och Weber definierar problem som lika om de har liknande lösningar (2013, s. 114). Det är därför viktigt att identifiera vad som gör problem lika för att hitta liknande och passande lösningar till nya problem. Likhet är i denna benämning inte exakt, utan en grad på en skala. Ett problem kan vara sammansatt av flera värden och det finns olika sätt att beräkna likhet mellan sammansatta värden och icke-sammansatta värden.

Två vanliga sätt att beräkna likhet mellan icke-sammansatta värden är luddig matchning och avstånd (2013, ss. 115-116). Luddig matchning associerar ett tal till ett par av värden, som är proportionerligt till graden av deras likhet. Avstånd associerar också ett tal till ett par värden, men avståndet är proportionerligt till inversen av graden av deras likhet. Ingen av metoderna är bättre, ibland kan det dock vara enklare att uttrycka likhet på ett eller annat sätt.

Likhet mellan sammansatta värden kan beräknas genom att aggregera deras elements likheter. Två samlingar är då lika om deras element är lika. Hammingsmätning (2013, ss. 127-128) går ut på att mäta likhet mellan två samlingar baserat på antalet element som är exakt lika. Med viktad hammingsmätning värdesätt olika element i samlingen olika, så att två samlingars likhet är mer beroende av vissa elements likhet än andra. Med metrisk likhet (2013, ss. 129-130) summeras istället elementens likhetsgrad. Hammingsmätning kan ge mindre precisa resultat än metrisk likhet, men är mer effektiv att implementera ur en prestandasynpunkt.

2.1.3 Hämtning

Syftet med hämtningsprocessen är att hitta en lösning vars problem är mest likt problemet som ska lösas. För att hämta lösningen används en hämtningsmetod, vars krav kan variera från system till system. Det kan vara viktigt att den hämtade lösningen tillhör det mest lika problemet, men detta kan påverka systemets effektivitet. För att hitta det mest lika problemet och dess tillhörande lösning används ofta sekventiell sökning (2013, ss. 171-172). Eftersom

problem kan likna varandra på olika sätt kan de ordnas efter olika kriterier och saknar en definitiv ordning. Av denna anledning är binärsökning mindre användbar för att söka efter fall. Om likhetsmetoden är prestandatung, kan problemen först gallras med en lättare likhetsmetod och sedan kan den tyngre likhetsmetoden användas på de kvarvarande problemen.

2.1.4 Anpassning

När det mest lika problemet har upptäckts är det inte säkert att dess lösning är direkt applicerbar på det nya problemet. Lösningen kan därför behöva anpassas. Anpassning sker genom att applicera ett antal regler på lösningen för att får fram en ny lösning. En regel består av ett förvillkor och en handling som ska utföras om förvillkoret är sant. Handlingar delas in i två kategorier: transformationsbaserade och genererande handlingar(2013, ss. 198-199). Transformationsbaserade handlingar utgår från lösningen och byter ut delar av den för att anpassa den till det nya problemet. Genererande handlingar används ibland när lösningen har beräknats från dess tillhörande problem. Om lösningen inte är giltigt för det nuvarande problemet, så används samma teknik som användes för att beräkna lösningen från problemet, med den nya lösningen som grund. Detta är användbart när det är mycket dyrare prestandamässigt att beräkna hela lösningen från början, än att utgå från en nästan giltig lösning. Ett exempel är en färdbeskrivning från en plats till en annan. Det kan då vara enklare att utgå från en färdbeskrivning mellan två städer nära ursprungsplatsen och destinationen än att beräkna en helt ny färdbeskrivning.

2.1.5 Tidigare arbeten

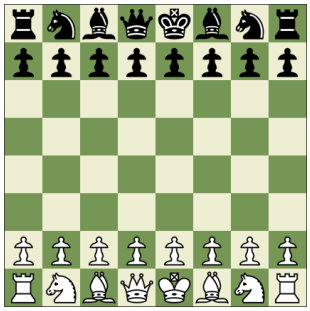
(V Känns tomt här, men jag vet inte vad jag säga hur mitt arbete relaterar till de andra arbetena. Jag har inte sett något jag kan återanvända/bygga på i mitt arbete.)

CBR har tidigare applicerats på spel. Bellamy-McIntyre (2008) har undersökt hur CBR kan appliceras för att lära en AI-agent att spela bridge. I Rubins arbete (2013) undersöks lämpligheten av att använda CBR för att utveckla en pokerspelande AI-agent. Wender och Watson (2014) presenterar hur CBR kan användas för att mikrohantera enheter i spelet Starcraft. Schack utmärker sig från dessa spel i den aspekten att det saknar slump och dold information. Det finns även en väldigt stor mängd expertdata tillgängligt att utgå ifrån, vilket minskar risken för att AI-agenten misslyckas på grund av bristande information.

2.2 Schack

2.2.1 Regler

Schack är ett turbaserat brädspel för två spelare där målet är att besegra sin motståndare (FIDE 2014c). Spelet utspelar sig på en 8x8-rutors spelplan, där varje spelare kontrollerar varsin armé av spelpjäser – vit och svart. I Figur 3 visas en bild av spelplanen i början av spelet.



Figur 3 Bild av spelplanen i början av spelet.

Spelarna turas om att flytta spelpjäser i sina arméer. En spelare får bara flytta en spelpjäs per drag. Två pjäser av samma färg får inte ockupera samma ruta. Om en spelare flyttar en av sina spelpjäser på en ruta ockuperad av en motståndarpjäs, så fångas motståndarpjäsen och lämnar spelplanen för resten av matchen. Högst en pjäs i taget får ockupera en ruta, och en pjäs får generellt inte flytta till en ruta om andra pjäser står i vägen till rutan. Om en pjäs kan flytta till specifik ruta betraktas det som att pjäsen hotar rutan, eller pjäsen som står på rutan.

Bonden (♟) kan flytta sig ett steg rakt framåt (sett från den ägande spelarens håll), eller ett steg diagonalt framåt om draget är ett fångande drag. Springaren (♞) kan flytta sig två steg horisontellt eller vertikalt, och ett steg på den resterande axeln. Springaren kan flytta till en ruta även om det finns pjäser som blockerar vägen. Löparen (♝) kan röra sig diagonalt. Tornet (♜) kan röra sig horisontellt eller vertikalt. Drottningen (♛) kan antingen röra sig horisontellt, vertikalt eller diagonalt. Kungen (♚) kan röra sig ett steg horisontellt, vertikalt eller diagonalt.

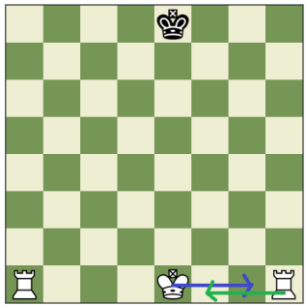
Om en bonde når den sista raden sedd ur ägarens perspektiv, så kan den omvandlas till vilken annan pjäs som helst.

Bonden kan flytta två steg rakt framåt på sitt första drag. Om en spelare flyttar en bonde två steg, så kan bonden betraktas som om den bara tog ett steg, om den fångas av en motståndarbonde nästa drag. Detta kallas *en passant* och illustreras i Figur 4.

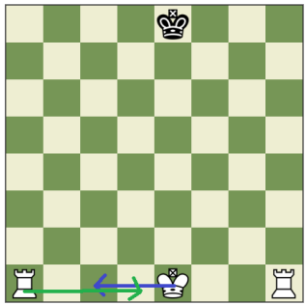
Figur 4 Bild av *an passant*. Om vit flyttar sin bonde två rutor framåt kan den svarta bonden fånga den genom att flytta till rutan som den röda pilen indikerar.

En spelare kan göra så kallad rockad med sin kung och ett torn, om det inte finns några pjäser mellan tornet och kungen, och varken tornet eller kungen har flyttats förut. Utöver det får kungen inte vara hotad, färdas över några rutor som hotas och dess slutgiltiga ruta får inte heller vara hotad. Rockaden går till så att kungen flyttas två steg i tornets riktning, och tornet flyttas i kungens riktning så att den hamnar en ruta på andra sidan av kungens nya position.

Figur 5 illustrerar hur detta kan se ut om den vita kungen gör rockad med det närmaste tornet, och Figur 6 illustrerar rockad med tornet längst bort. Detta kallas kort respektive lång rocked.

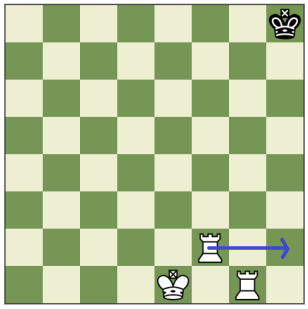


Figur 5 Bild som visar hur pjäserna flyttas när vit gör kort rockad.



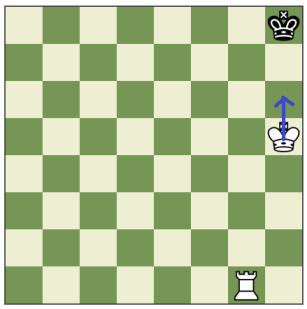
Figur 6 Bild som visar hur pjäserna flyttas när vit gör lång rockad.

En spelare får aldrig göra ett drag som leder till att motståndaren hotar spelarens kung. Om en spelare gör ett drag så att motståndarens kung hotas kallas det för schack. Detta gäller även om kungen hotas av pjäser som skulle lämna sin kung hotad om de flyttade från sin ruta. Om motståndaren inte kan följa upp med ett drag som försätter kungen ur schack vinner spelaren, vilket kallas för schack matt. Ett exempel av schack matt visas i Figur 7.



Figur 7 Bild som visar hur vit kan göra schack matt. Kungen hotas av tornet på andra raden, samtidigt som den inte kan flytta sig utan att fortfarande hotas.

Om motståndarens kung inte hotas, men samtidigt inte kan göra något drag utan att kungen hotas så blir det lika, vilket även kallas för remi. Ett exempel av remi visas i Figur 8.



Figur 8 Bild som visas hur vit kan göra remi. Den svarta kungen hotas inte, men samtidigt kan den inte flytta sig någonstans utan att hotas av tornet eller den vita kungen.

2.2.2 Elo-rankning

(Utökad och mer detaljerad beskrivning.)

Elo-rankning är ett sätt att ranka schackspelare relativt till varandra (FIDE 2014a). Enligt Elo-rankningssystemet rankas spelare i form av poäng. FIDE:s rankningar av spelare uppdateras kontinuerligt allt eftersom spelare spelar matcher mot varandra i schacktävlingar. Hur mycket en spelares rankning påverkas av vinster och förluster beror på hur hög dess rankning är proportionerligt till dess motspelare. FIDE har arbiträrt bestämt troligheten att en spelare kommer besegra en annan spelare baserat på skillnaden i deras rankning. Troligheten är ett spann från 0 % till 100 % (uttryckt som 0 till 1), och skillnaden i rankning är från -800 till +800. Om skillnaden i rankning är större eller lägre än +-800 är troligheten 100 % att spelaren med högre rankning vinner.

Hur mycket en spelares ranking påverkas av en match beräknas på följande sätt:

- För varje match får en spelare 1, 1/2 eller 0 poäng om den vann, gjorde remi eller förlorade.
- Troligheten att en spelare skulle vinna (mellan 0 och 1) subtraheras från antalet poäng den fick. Detta tal kan bli 1 som mest och -1 som minst.
- Detta tal multipliceras med en koefficient mellan 10 till 40, baserat på hur många matcher spelaren tidigare spelat, dess nuvarande rankning och ålder.
- Det slutgiltiga (möjligen negativa) talet adderas med spelarens nuvarande rankning, för att få dess nya rankning.

Detta system ser till att spelare inte går ner mycket i rankning om de inte förväntas vinna; en spelare kan inte ens minska i rankning om den förlorar mot en spelare med mer än 800 i rankning. Likaså kan en spelare inte gå upp i rankning om den bara besegrar spelare med mycket lägre rankning.

2.2.3 Portable Game Notation

(Jag nämner inte PGN någonstans i texten längre, utan tänker skjuta på det till genomförandet. Ha kvar/ta bort sektionen?)

I artikeln *Standard: Portable Game Notation Specification and Implementation Guide* (1994) beskrivs PGN som ett format för att spara och beskriva schackmatcher. Formatet blev snabbt populärt och idag finns det tusentals allmänt tillgängliga sparade partier på t.ex. FIDE:s hemsida. Ett PGN-dokument kan innehålla ett antal matcher och varje match

innehåller metainformation om matchen och de drag som utfördes i matchen. Informationen kan gälla när eller var matchen spelades och av vilka. Dragen skrivs med algebraisk notation (AN).

AN är en notation som beskriver drag kortfattat till den grad att de inte är tvetydiga. Raderna numreras från vits synvinkel med bokstäver från a till h, och kolumnerna med siffrorna 1 till 8. En ruta på spelplanen kan då beskrivas med dess tillhörande rad och kolumn t.ex. e4 eller a2. Drag har ett prefix med stor bokstav som beskriver vilken sorts pjäs som flyttades. N för springare, R för torn, B för löpare, Q för drottning, K för kung, medan bonde saknar prefix. Detta följs av positionen som pjäsen flyttades till. Exempel: e4, Nf3, Bb5. Om draget är ett fångande drag så sätts ett x framför rutan som pjäsen flyttades till. De drag som leder till schack har ett plustecken som suffix. Kort rockad representeras med ”O-O” och lång rockad representeras med ”O-O-O”.

I de fall då ett drag är tvetydigt, t.ex. om två springare på e4 respektive e6 kan flytta till c5, så följs pjäsbokstaven av radkoordinaten eller kolumnkoordinaten beroende på vilken som kan uttrycka draget unikt (Nee5 är inte unikt i detta fall, medan N4e5 är det). Efter det sista draget i matchen visas resultatet 1-0, 0-1, eller ½-½ om vit vann, förlorade, respektive gjorde remi med svart. I Figur 9 visas ett exempel av en match beskriven i PGN.

[Event "F/S Return Match"]
[Site "Belgrade, Serbia Yugoslavia|JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 {This opening is called the Ruy Lopez.}
4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6 16. Bh4 c5 17. dxe5
Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6 21. Nc4 Nxc4 22. Bxc4 Nb6
23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+ 26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5
hxg5 29. b3 Ke6 30. a3 Kd6 31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5
35. Ra7 g6 36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5 41. Ra6
Nf2 42. g4 Bd3 43. Re6 1/2-1/2

Figur 9 En schackmatch i PGN-formatet. Notera att numreringen inte ökar för varje drag, utan varje par av drag.

2.2.4 Tidigare arbeten inom alternativ AI (Ny sektion)

Gould & Levison (1991) presenterar i sitt arbete en schackspelande AI-agent vid namn *Morph*. Morph utmärks av att den under ett antal spelade partier lär sig associera mönster av schackformationer med drag som bör utföras. Likt AI-agenten i detta arbete är den baserad på tanken att liknande problem har liknande lösningar. Morph skiljer sig dock från AI-agenten i detta arbete eftersom den börjar med väldigt lite kunskap, och lär sig genom att spela matcher.

3 Problemformulering

3.1 Problembeskrivning

Syftet med det här arbetet är att undersöka hur lämpligt CBR kan vara för att utveckla schackspelande AI-agenter. Ingen har ännu upptäckt en perfekt strategi som garanterar vinst i schack, och användningen av CBR för utveckling av schackspelande AI-agenter kan ge mer insikt i hur strategier implicit utförs av spelare. Detta kan leda till förbättringar inom forskningen av schackspelande AI-agenter.

Det kommer inte undersökas om den utvecklade AI-agenten kan spela på samma nivå som de bästa mänskliga eller artificiella spelarna i världen. I arbetet läggs istället mer fokus på att undersöka hur bra AI-agentens resultat av spelade matcher stämmer överens med rankningen av experten som dess fallbas är baserad på. Här benämns hur AI-agenten spelar med en fallbas som ett beteende. Frågan som ska försöka besvaras i det här arbetet är då följande: hur väl stämmer en schackspelande, CBR-baserad AI-agents beteendes skicklighet överens med skickligheten av experten som de expertdata den är baserat på, i relation till andra beteendes skicklighet. Om det visar sig finnas en stark koppling, ger det möjlighet i framtida arbeten att undersöka om en CBR-baserad schackspelande AI-agent kan spela i världsklass givet en tillräckligt bra fallbas baserat på en tillräckligt bra expert.

(^ Frågan är stor, klumpig och kanske tvetydig. Jag vill helst minska ner den och flytta en del till metoden. Problemet är att en kortare fråga skulle vara för vag och svår att besvara i arbetet. Jag skulle kunna säga ”hur lämpligt är CBR för att skapa en bra schackspelande AI-agent”, men hur kan fastställa om den är ”bra” när jag inte jämför med existerande schack-AI/spelare? Spelar det någon roll att frågan är så lång om den fortfarande går att förstå och besvara?)

(^Sista meningen känns dålig. Jag vill säga att ”om man bara hittar tillräckligt många bra fall och/eller en tillräckligt bra expert så kan den bli jättebra” men jag vet inte hur jag ska uttrycka det bra.)

För att tekniken ska vara av användning för utvecklandet av schack-AI måste AI-agenten kunna implementeras på konsumenthårdvara och utföra drag under tidskrav. I FIDE-tävlingar får en schackspelare 90 minuter på sig att utföra sina första 40 drag (FIDE 2014b), vilket är det tidskrav som AI-agenten förväntas följa.

3.2 Metodbeskrivning

Ett program ska skapas av en schackspelande AI-agent baserad på CBR, som kan använda olika fallbaser. Anledningen att ett program används, är att det skulle vara svårt att för hand analysera den stora mängden expertdata som AI-agenten kommer använda och dra slutsatser om AI-agentens prestation med en given fallbas.

För att undersöka om AI-agentens skicklighet är relativ till skickligheten av experten som dess fallbas är baserad på, ska den tävla mot sig själv med olika beteenden. Om ett beteenden vinner över ett annat ska det vinnande beteendet få ett poäng. Om det blir lika ska båda få ett halvt poäng. När varje beteende har spelat mot varje annat beteende ett antal gånger ska beteendena graderas i relation till varandra baserat på hur många poäng de fick. Om de högre

rankade spelarnas beteenden generellt fick högre ranking, skulle det visa att AI-agenten presterar bättre med bättre expertdata, vilket är det som ska undersökas i arbetet.

Expertdata från olika experter ska komma från drag som experten utfört i olika lägen i tidigare matcher. Experter kommer rankas efter deras ELO-rankning, så att bättre experter har högre rankning. Expertdata ska komma från en allmänt tillgänglig databas av sparade schackmatcher. Ett möjligt problem är att det inte är säkert att en delmängd av en spelares historik av spelade schackmatcher visar hur det fick sin rankning. En spelare kan ha haft en låg rankning ett långt tag, men nyligen blivit mycket bättre; ska spelarens äldre partier då representera hur spelaren spelar, lika mycket som de nya? I detta arbete ignoreras problemet för att det anses oväsentligt.

En nackdel med metoden är att hur bra AI-agenten skulle kunna prestera mot andra spelare inte undersöks. Hur bra AI-agenten kan spela är självrefererande, och det är fullt möjligt att olika beteenden spelar olika bra i relation till varandra men att ingen av dem kan besegra en nybörjare, trots att flera av experterna är högt rankade. Det är även möjligt att fallbasernas interna rankning mot varandra inte skulle stämma överens med deras rankning i relation till andra spelare. De internt bästa beteendena kan utföra strategier som de sämre beteendena faller för, men inte andra spelare. Likaså kan andra spelare utföra strategier som de internt bättre beteendena faller för, men inte de sämre. Att lämna dessa frågor obesvarade skulle göra lämpligheten för CBR-baserade schackspelande AI-agenter oklar. För att besvara dessa frågor måste AI-agenten spela mot ett antal andra rankade spelare, för att beräkna varje beteendes ELO-rankning. Det är dock väldigt tid- och resurskrävande att hitta ett större antal rankade spelare och låta dem spela ett stort antal matcher mot alla beteenden. Av denna anledning lämnas frågorna obesvarade i detta arbete.

(^ I de två ovanstående paragraferna tog jag upp alla problem med metoden jag kunde komma på. Jag kan ta bort en del om det känns överflödigt. Speciellt problemet att beteendenas skicklighet är självrefererande känns jobbigt. Det känns som arbetet kan bära eller brista på den punkten.)

Referenser

Bellamy-McIntyre, J. (2008). *Applying Case-Based Reasoning to the Game of Bridge*. Diss. University of Auckland, New Zealand.

Gould, J. & Levinson, R., A. (1991). *Method integration for experience-based learning*. USA: University of California.

Huber, R. (2006). *Description of the universal chess interface (UCI).* http://download.shredderchess.com/div/uci.zip [2015-02-08]

Mann, T. & Muller, H. G. (2009). *Chess Engine Communication Protocol.* http://www.gnu.org/software/xboard/engine-intf.html [2015-02-09]

Rubin, J. (2013) *On the Construction, Maintenance and Analysis of Case-Based Strategies in Computer Poker*. Diss. University of Auckland, New Zealand.

Richter, M. M., & Weber, R. O. (2013). *Case-Based Reasoning: A Textbook*. Berlin: Springer Verlag.

Schaeffer, J. (1991). Computer Chess: Science of Engineering? I *IJCAI-91, Proceedings of the Twelfth International Conference on Artificial Intelligence*. Sydney, Australia.

Shannon, C. E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, 41(314).

***Standard: Portable Game Notation Specification and Implementation Guide* (1994). http://www6.chessclub.com/help/PGN-spec [2015-02-16]**

Wender, S., & Watson, I. (2014). *Integrating Case-Based Reasoning with Reinforcement Learning for Real-Time Strategy Game Micromanagement*. Diss. University of Auckland, New Zealand.

World Chess Federation (2014a). *FIDE Rating Regulations Effective From 1 July 2014.* http://www.fide.com/fide/handbook.html?id=172&view=article [2015-02-09]

World Chess Federation (2014b). *General Rules and Recommendations for Tournaments: Time Control.* www.fide.com/component/handbook/?id=39&view=category [2015-02-09]

World Chess Federation (2014c). *Laws of Chess: For competitions Starting On or After 1 July 2014*. http://www.fide.com/fide/handbook.html?id=171&view=article [2015-02-09