Tetris-AI med genetiska algoritmer

Fördjupningsuppgift i Artificiell Intelligens, HKGBB0, ht 2005 vid Linköpings Universitet

I denna rapport beskriver jag mitt försök att skapa ett tetrisspelande program som optimeras med hjälp av genetiska algoritmer. Resultatet är ett fungerande program, dock utan några toppresultat. Jag ger också en redogörelse för tidigare forskning om tetris och ett tidigare försök att optimera ett tetrisspelande program med hjälp av genetiska algoritmer.

Christer Byström
790304-7830
chrby001@student.liu.se

Inledning

Tetris är ett välkänt och svårt pusselspel. Spelet har ett flertal gånger varit föremål för vetenskapliga rapporter och ett antal tetrisspelande algoritmer har skrivits. Jag har satt mig in i en del av dessa rapporter och försökt replikera ett försök med en tetrisspelande algoritm som optimerades med hjälp av genetiska algoritmer. Jag kommer här att redogöra för en del intressanta aspekter av tetris samt presentera resultatet av mitt försök.

Bakgrund

Kort om tetris

Tetris är ett pusselspel som först konstruerades av Alexey Pajitnov 1985 och som sedan implementerats på en mängd olika system och figurerat i många olika varianter, typiskt med en spelplan bestående av ett 10 x 20 rutor stort rutnät, där olika spelpjäser faller mot botten. (Wikipedia1) Spelpjäserna består av fyra rutor i olika konfigurationer, så kallade tetraminor. När en hel rad är fylld rensas den bort och alla rutor ovanför raden flyttas ned ett steg. När spelplanen är så fylld att inte fler pjäser längre kan släppas ner i den är spelet slut. Målet med spelet är oftast att överleva så länge som möjligt.

Det finns sju olika tetraminor. De brukar benämnas på lite olika sätt, men jag använder mig av följande namn på de olika pjäserna, enligt Colin Faheys taxonomi (Fahey1):

O **I** **S** **Z** **L** **J** **T**

I många varianter av tetris finns ett läge som visar nästkommande pjäs, (hädanefter kallat för pjäsvarsel). Ett sådant läge antas minska svårighetsgraden i spelet. Huruvida det stämmer för mänskliga spelare vet jag inte, men för tetrisspelande program förbättras prestationen väsentligt. Fahey har något slags tävling för tetrisalgoritmer, där rekordet med pjäsvarsel är 7 miljoner rensade rader medan rekordet utan pjäsvarsel är 2 miljoner. (Fahey2)

Det mått som oftast används för att mäta prestation hos tetrisspelande program är antal rensade rader. Antal rensade rader med en standardspelplan är i längden approximativt antal släppte pjäser dividerat med 2,5 (varje pjäs består av fyra rutor, en rad består av tio, 10/4=2,5).

Tetris är ändligt

Heidi Burgiel (1997) har visat en viss sekvens av pjäser alltid kommer att avsluta ett spel efter mindre än 127200 pjäser. Hon utgår från en standardspelplan på 10 x 20 rutor. Den sekvens hon pratar om är en sekvens där varannan pjäs är en S-tetramina och varannan en Z-tetraminor.

Burgiel börjar med att visa att ett spel med bara denna sekvens av pjäser slutar efter

senaste 69600 placerade pjäser. Vi börjar med en tom spelplan. För eller senare kommer en spelare som stöter på en sådan sekvens tvingas stapla pjäserna på högkant i kolumner, S-tetraminor för sig och Z-tetraminor för sig, för att klara sig. Detta kommer att ge en spelplan med två staplar med den ena typen av tetraminor och tre av den andra. Vi antar att S-tetraminor staplas i två kolumner och Z-tetraminor i den andra. De två staplarna med S-tetraminor kommer så småningom att vara såg höga att inga fler pjäser kan placeras ovanpå dem. Nästa S-tetramina måste då placeras ovanpå någon av staplarna som består av Z-tetraminor. Detta kommer att ge upphov till ett hål som inte kan fyllas. Det är inte möjligt att ha fler än 50 sådana hål (10 per stapel) utan att spelet avslutas.

Burgiel visar hur högst 120 pjäser kan placeras 'fel' (i det här fallet en pjäs som inte placeras ovanpå en pjäs av samma typ i samma orientering) utan att spelet avslutas.

Eftersom varje felplacerad pjäs kan fylla som mest två hål, och antal möjliga hål innan spelet avslutas är 50, kan spelet hålla på tills 120 * 2 + 50 = 290 hål har bildats.

Genom att stapla denna sekvens av varannan S- eller Z-tetramina i två respektive tre kan som högst 240 pjäser placeras utan att spelaren tvingas placera en tetramina i 'fel' stapel och skapa ett hål.

Antal hål som kan bildas innan spelet avslutas är 290. Efter senast var 240:e pjäs kommer ett hål att skapas. Detta ger att ett spel måste ta slut senast efter 240 * 290 = 69600 placerade pjäser.

Burgiel generaliserar detta bevis till ett standardtetrisspel, där startförhållandena innan en sekvens av S- och Z-tetraminor kommer är godtyckliga. Hon visar att antal pjäser som kan placeras 'fel' blir som mest 240 istället för 120. Detta ger att en sekvens på (120 * 2 + 50) * 240 = 127200 pjäser bestående av varannan S- och Z-tetraminor avslutar ett godtyckligt spel. I ett oändligt spel kommer denna sekvens att uppträda, därför är tetris ändligt.

Burgiel noterar att ett tetrisspel vars pjäsuppsättning inte innehåller tetraminorna S och Z inte nödvändigtvis är ändligt. Till exempel kan ett spel med enbart O-tetraminan (en fyrhörning) i pjäsuppsättningen kan pågå oändligt lång tid.

En svaghet jag ser med detta resonemang är att datorer inte använder sig av rena slumptal, bara normalfördelade funktioner som är svåra för en människa att förutsäga. Därför kommer inte nödvändigtvis den avslutande sekvensen av S- och Z-tetraminor att uppträda.

Tetris är troligtvis NP-komplett

Demaine, Hohenberger och Liben-Nowell (2003) har studerat komplexiteten hos en så kallad ”offlineversion” av tetris där sekvensen av tetraminor är helt känd. De har visat att svårigheten hos tetris ligger i spelets natur och inte bara i den begränsade reaktionstiden för en spelare, med andra ord att tetris är svårt även om man får tänka hur länge som helst före varje drag. Ett ”offlinespel” är naturligtvis mycket lättare än ”onlineversionen” av tetris, men fångar något av svårigheterna med ett normalt tetrisspel.

I deras ”offlineversion”, där de från en spelplan som initialt har ett antal placerade pjäser, har de bevisat att ett antal antal funktioner som är vitala för ett tetrisspel är np-kompletta:

- maximera antal bortrensade rader med en given sekvens av pjäser
- maximera antal pjäser innan en förlust inträffar
- maximerar antal tetrisar – att samtidigt rensa bort fyra rader
- minimera den maximala pjäshöjden på spelplanen under den spelade sekvensen

Deras arbete är digert och innehåller ett flertal formella och informella bevis. Den ”offlineversion” av tetris de utgår ifrån börjar med ett antal pjäser på spelplanen och man kan därför formellt inte generalisera deras bevis för att gälla spel där spelplanen till en början är tom. Det ger dock en bra fingervisning om komplexiteten och beräkningsbarheten hos spelet i allmänhet.

Tidigare arbeten

Även om Demaines et al. (2003) bevis för att vissa varianter av tetris NP-kompletta inte formellt kan generaliserats till alla varianter av tetris (såsom tetris som startar med tom spelplan) är dock tetris erkänt svårt att beräkna.

Eftersom det verkar vara omöjligt att beräkna den optimala draget för varje pjäs och spelplan, brukar tetrisspelande algoritmer istället använda sig av någon sorts heuristik för att förutsäga hur bra ett drag är. Alla tetrisspelande algoritmer jag känner till arbetar efter principen att generera alla möjliga spelplaner givet den nuvarande spelplanen och nuvarande pjäs och ranka den resulterande spelplanen med hjälp av ett antal mått som sedan viktas (Fahey3).

En tetrispjäs kan vara på som mest 10 olika positioner (stående 'I') på brädet i standardvarianten av tetris, och ha som mest fyra olika orienteringar ('I' har två och 'O' en). Det finns som mest 2 * 9 + 2 * 8 = 34 spelplaner (mina beräkningar). Av de sju pjäserna gäller denna beräkning för fem av dem, alla utom 'I' och 'O'. Om vi tar 'S' som exempel så har pjäsen 2 orienteringar där den är 2 rutor bred, vilket ger 9 möjliga positioner, och 2 orienteringar där den är 3 rutor bred vilket ger 8 möjliga positioner.

Två av de tetris-AI jag stött på på har optimerat sina vikter med hjälp av genetiska algoritmer.

Ett evolutionärt försök

Böhm, Kókai och Mandl (2005) har använt sig av genetiska algoritmer för att optimera vikterna hos ett tetrisspelande program. Det framgår inte om de använt sig av pjäsvarsel eller inte. De använde sig av tre olika sorters viktningsfunktioner.

1. *linjär:* $R_l(b)=\sum_{i=1}^{n} w_i r_i(b)$
2. *exponentiell:* $R_e(b)=\sum_{i=1}^{n} w_i r_i(b)^{e_i}$
3. *exponentiell med förskjutningsvärde (displacement value):* $R_d(b)=\sum_{i=1}^{n} w_i |r_i(b)-d_i|^{e_i}$

w_i och e_i är vikter och r_i är måtten från de olika mått/viktfunktionerna. Idén bakom den tredje viktningsfunktion var att det optimala värdet för vissa mått kanske inte är noll. Om målet är att rensa fyra rader så ofta som möjligt så måste den djupaste brunnen (ett vertikalt sammanhängande hål, en ruta brett utan fylld ruta ovanför sig) vara minst fyra. De rapporterar dock att användningen av ett förskjutningsvärde inte gav någon prestandaförbättring.

Som fitnessvärde användes antal släppta pjäser. Fitnessvärdet beräknades på mellan fem och tolv spelade omgångar per individ, där det bästa och sämsta resultatet räknades bort vid minst sju spelade rundor. De använde sig oftast av en population på 100 individer mer

rapporterar inte om storleken på elitselektionen. De använde sig av en korsningselektionsmekanism som var proportionell mot fitnessvärdet och mutation med hjälp av multiplikation och slumpvärden med ett medelvärde runt 1. Vidare skalades alla individer med en konstant faktor.

Eftersom man använde sig av multiplikation för att mutera generna (vikterna) skulle det visa sig vilka mått som var mest betydelsefulla, då deras vikter skulle uppvisa de högsta värdena. Det visade sig vara höjd, antal hål, vertikalt sammanhängande hål och högsta brunnsdjup.

De använde sig både av en standardspelplan på 10 x 20 rutor och en mindre på 6 x 12 för att göra spelet svårare och på så vis minska tiden för varje generation. De använde sig initialt av sex vikter och lyckades få runt 850000 rader på den stora spelplanen (med den linjära funktionen), men utökade sedan antal vikter till till tolv. Det bästa resultatet på den stora spelplanen var med 12 vikter nästan 5,5 miljoner rader för den linjära vilket var bättre än den exponentiella.

Man prövade också att använde vikterna fromevolverade på den lilla spelplanen på den stora. Resultatet var 34 miljoner rader för den exponentiella viktningsfunktionen och mer än 480 miljoner rader för den linjära. För 6 x 12 spelplanen lyckades den exponentiella bättre än den linjära.

Att evolvera 30 generationer med den exponentiella funktionen på 1 0 x 20 tog 3 månader.

Xtbot

Roger Espel Llima har tidigare gjort ett försök med en tetrisspelande algoritm optimerad med genetiska algoritmer. Hans algoritm finns i en tetrisspelande AI kallad Xtbot (Fahey2) gjord för en flerspelarvariant av tetris kallad Xtris. Algoritmen gjorde ett snitt på 42000 klarade rader över 150 spel, utan att använda sig av pjäsvarsel. Originalkoden uppges uppskattningsvis vara från 1996.

Implementation

De specifikationerna av tetris som jag utgått ifrån är Colin Faheys Standard Tetris (Fahey1), som i sin tur utgår från tidiga versioner av originalspelet. Vidare har jag i någon mån försökt reproducera Böhms et al (2005). försöka med evolutionära algoritmer. Om inte annat anges så har jag efterliknat deras försök så gott som det gått med den information som fanns tillgänglig.

Heuristiken

För varje ny pjäs som släpps genererar AI:n alla möjliga spelplaner utifrån nuvarande spelplan och pjäs, med standardspelplanen som mest 34 stycken. AI:n använder sedan ett antal viktade mått för att ranka brädet. De mått jag har använt mig av är följande:

1. Högsta rad med fyllda rutor
2. Antal hål (tomma rutor med minst en fylld ruta ovanför)
3. Antal sammanhängande hål (samma som antal hål med horisontellt intilliggande tomma rutor räknas som ett hål)
4. Antal brunnar (tomma rutor som går att nå direkt uppifrån, med en fylld ruta till

höger och vänster, spelplanens kanter räknas som fyllda rutor)

5. Djupaste brunn (summan av max antal sammanhängande tomma rutor, varav den översta har en fylld ruta till höger eller vänster)
6. Höjdskillnad (skillnad mellan högsta fyllda ruta och lägsta fyllda ruta som är nåbar direkt uppifrån, dvs inte har någon ruta ovanför sig)
7. Antal rensade rader i föregående drag

Alla mått utom nummer 4 användes av [evoltionary] i deras första försök med sin tetris-AI. Nummer 4 använde de senare då de använde sig av 12 mått.

Jag har använt mig av två av [evolutionarys] rakningsfunktioner, den linjära, $R_l(b)=\sum_{i=1}^{n} w_i r_i(b)$ och den exponentiella, $R_e(b)=\sum_{i=1}^{n} w_i r_i(b)^{e_i}$. Den linjära använder sig av en vikt per mått, den exponentiella två. Eftersom [evolutionary] rapporterade att användning av ett förskjutningsvärde inte gav någon prestandaförbättring använde jag mig inte av någon sådan funktion. En implementationsmässig skillnad är att jag använder mig av flyttal för både w_i och e_i medan de använder sig av heltal för w_i. Jag tror inte att det gör någon större skillnad i praktiken, möjligtvis att mina beräkningar blir lite tyngre.

Evolution

Hos en individ representerades varje vikt som ett flyttal (en double i C).

Jag använde mig av en populationsstorlek på 100 individer och en elitselektion på 20 individer. Varje individ testkördes 12 gånger, det bästa och det sämsta resultatet räknades bort och ett medelvärde drogs som fick vara fitnessvärdet. De 20 individerna i elitselektionen kopierades rakt av till nästa generation, och övriga 80 platser fylldes med korsningar av individer som valdes slumpmässigt med en sannolikhet proportionellt mot deras fitnessvärde. Jag experimenterade även med att explicit öka elitselektionens reproduktionssannolikhet jämfört med övriga populationen, men i slutändan gjorde det inte någon större skillnad på slutresultatet.

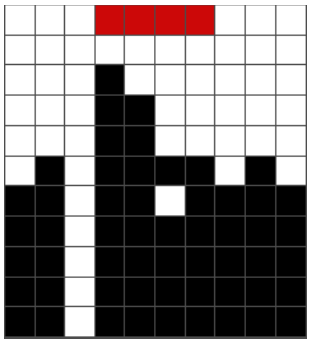
Efter att korsningen skett muterades varje gen i varje individ, bortsett från elitselektionen med en konstant sannolikhet. Mutationer skedde med hjälp av multiplikation på samma som som i Böhms et al. (2005) försök. Vidare skalades hela populationen, inklusive elitselektionen med en konstant faktor.

Att beräkna godkända drag

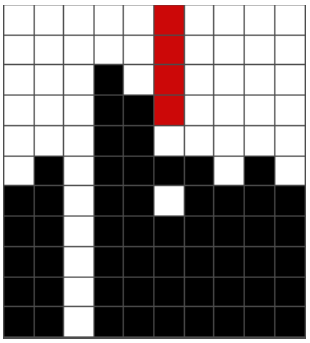
När en mänsklig spelare spelar tetris sker det i realtid. Alla förflyttningar som görs sker på den aktuella spelplanen. Med en tetrisspelande AI måste alla möjliga förflyttningar systematiskt prövas för att kunna ranka alla möjliga drag. Här föreligger en svårighet, då höjden börjar krypa uppåt så att det bara finns två rader högst upp som är fria från pjäser. En förflyttning eller rotation är godkänd så länge pjäsen inte tar i någon fylld ruta. Problemet är att om man bara använder sig av strategin förflytta från mitten till sidled sedan rotera, så utesluter man möjliga godkända drag, då det kan vara möjligt att släppa en pjäs från den höjden, men vägen dit inte kan beskrivas med en rotation till önskad orientering och sedan en eventuell förflyttning i sidled. Att först förflytta och sedan rotera löser inte heller problemet, eftersom nästföljande rotation kanske möjliggör ytterligare förflyttningar i sidled. Man riskerar nu att hamna i ett situation där man måste söka en väg till målet, om inte sekvensen rotation *n* gånger följt av sidledsförflyttning *n* gånger

fungerar.

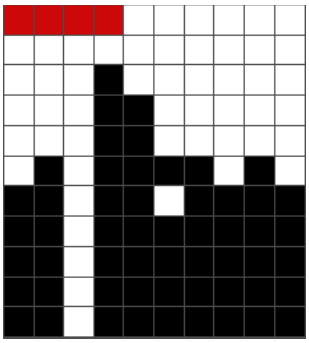
Nu verkar det ju som sagt inte som om det går att beräkna det optimala draget i tetris, men om man åtminstone vill ta med alla möjliga drag i rankningen så måste man göra en sökning. Ett tydligt exempel där det teoretiskt sätt kan vara betydelsefullt är då man har en djup brunn på vänstra sidan av spelplanen, men mellan mitten och brunnen är rutorna fyllda så högt att bara två rader är fria. Antag att vi får en I-tetramina. Det går att flytta den till vänster, om man väljer att förflytta innan man roterar. Jag illustrerar med några figurer föreställande toppen på en spelplan:



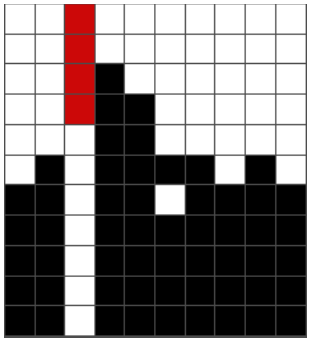
1. Utångsposition



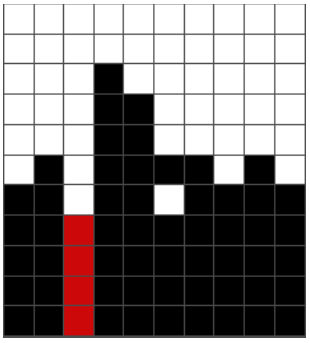
2. Rotering från utgångsposition. Pjäsen går inte att flytta åt vänster.



3. Förflyttning från utgångsposition (1) utan att rotera.



4. En rotering efter (3)



5. Pjäsen släppt efter (4)

Det finns mycket mer komplexa problem där man exempelvis måste flytta, sedan rotera och till sist kanske rotera igen för att komma åt önskad position. Antag att brunnen i figuren låg längst till vänster, då hade vi behövt flytta två steg åt vänster efter rotation.

Att ta sådana här hänsyn skulle kunna resultera i en kraftig ökning av komplexiteten i beräkningarna för att ranka ett drag. Jag har dock inte stött på någon diskussion om detta problem, kanske för att det i sammanhanget är försumbart.

Om man jobbar med en evolutionär algoritm för att att optimera vikter är det mycket möjligt att man kan förenkla reglerna för förflyttningar för att få upp farten på evolutionen, men alla noteringar om toppresultat som skett under en evolution måste då ses som ogiltiga.

Min lösning

Jag har valt en enkel, icke-optimal men korrekt lösning. Jag roterar en pjäs till dess önskade orientering, och gör därefter eventuella förflyttningar till önskad x-position. Om pjäsen tar i någonstans under orienterings- eller förflyttningsfasen förklaras förflyttningen

ogiltig och nästa position/orientering prövas. Den enda optimering jag använder mig av är att jag låter de pjäser och orienteringar som behöver det falla ett steg först innan jag roterar.

Xtbot

Jag tittade på källkoden[1] till xtbot och den verkar inte pröva olika vägar utan antar helt enkelt att en pjäs börjar existera på den position varifrån den ska släppas. Det funkar i de flesta lägen, utom då man kommit för högt.

Ett klipp ur källkoden (Rad 237-259 i decide.c):

```
for (x=xmin; x<=xmax; x++) {
  for (rot=0; rot<rotations[piece]; rot++) {
    y = y0;
    if (!fits(piece, rot, x, y))
    continue;
    while (fits(piece, rot, x, ++y));
    if (sticksout(piece, rot, x, --y))
    continue;
    put (piece, rot, x, y, CURRENT);

    values[x-xmin][rot] = v = eval(x, y, rot);
    ys[x-xmin][rot] = y;

    remove(piece, rot, x, y);

    if (v == maxval)
    ties++;
    else if (v > maxval) {
    maxval = v;
    ties = 1;
    }
  }
}
```

Vi ser här två slinger som löper igenom alla x-positioner och alla orienteringar. Det är möjligt att det är något jag missförstått med källkoden (själva AI-delen var helt okommenterad), men vad jag kan förstå kontrollerar koden inte för om förflyttningar från mitten och utgångsorientering till önskad position är möjlig. Denna lösning använder sig av något slags relaxerade regler för att fatta beslut. Det fungerar tills man beslutar sig för ett drag som är ogenomförbart, men då det inträffar är det å andra sidan redan stor risk att spelet ska avslutas.

Standard tetris

Fahey har använt sig av en lösning som går ut på att låta en pjäs falla ett steg, för att sedan göra rotationer och till sist förflyttningar. Som jag förstår det tillåter den inte att AI:n gör ogenomförbara förflyttningar. Det linkar min lösning, men är något mindre optimal då den inte tar hänsyn till om pjäsen måste falla ett steg eller inte.

En kommentar ur källkoden[2] till Faheys Standard Tetris-applikation (standard_tetris_strategy.cpp, rad 120-129):

1 Hämtad 2005-10-20 klockan 15:30 från http://www.iagora.com/~espel/xtris/xtris-current.tar.gz

2 Hämtad 2005-10-20 klockan 14:23 från http://www.colinfahey.com/2003jan_tetris/standard_tetris_v2003june11.zip

```
// *** WARNING: When you get the "best" rotation and translation
// from the following function, you must wait until the piece has
// its origin at least as low as row 0 (zero) instead of its initial
// row -1 (negative one) if any rotations (1,2,3) are required.
// Perform all rotations, and then perform translations.  This
// avoids the problem of getting the piece jammed on the sides
// of the board where rotation is impossible. ***
// Also, the following strategy does not take advantage of the
// possibility of using free-fall and future movements to
// slide under overhangs and fill them in.
```

Resultat

Det mått jag använde mig av var antal släppta pjäser. På en 10 x 20 spelplan motsvarar en rensad rad approximativt 2,5 släppta pjäser, och på 6 x 12 1,5 släppta pjäser. Varje pjäs består av fyra rutor, på en 10 x 20 spelplan är varje rad 10 rutor därav 10 / 4 = 2,5 pjäser / rad. För 6 x 12 gäller 6 / 4 = 1,5 pjäser / rad.

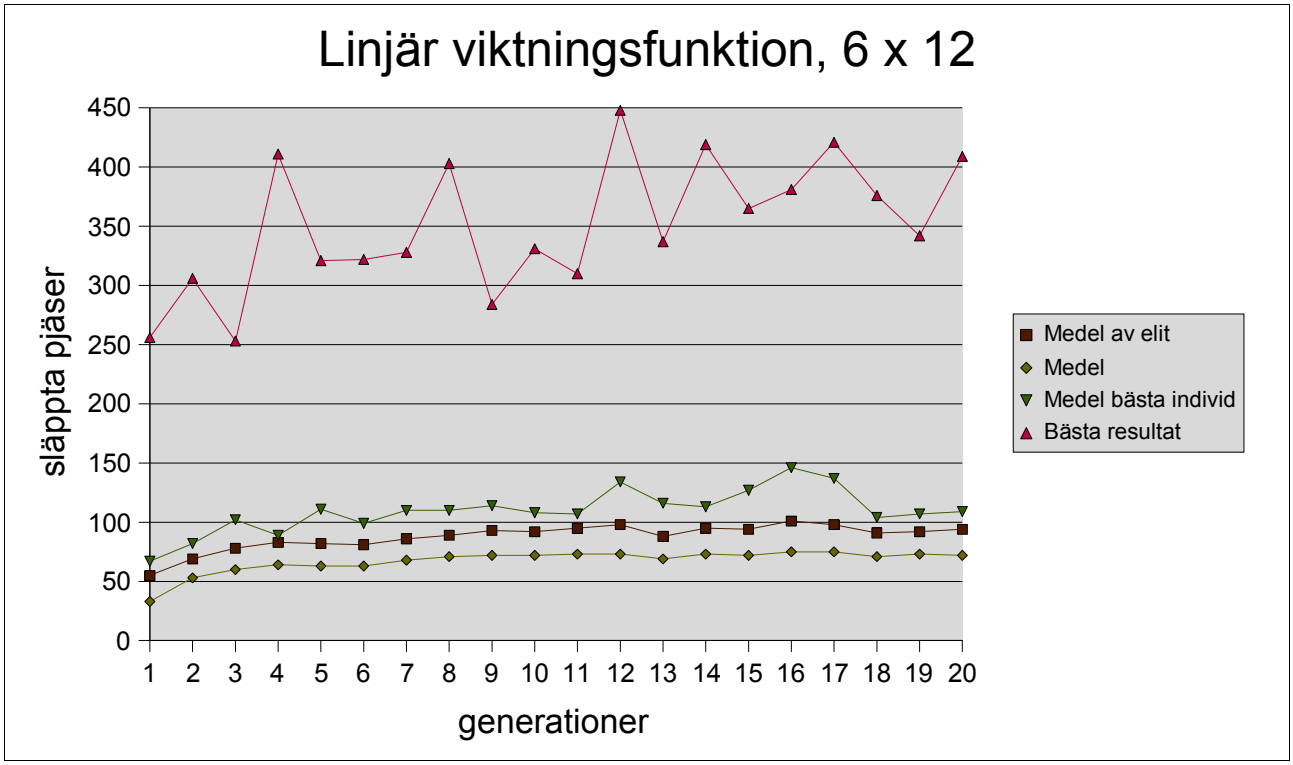
Vilken funktion som lyckades bäst av den linjära och den exponentiella beror på vilket mått man använder sig av. Det presterade i princip likvärdiga resultat.

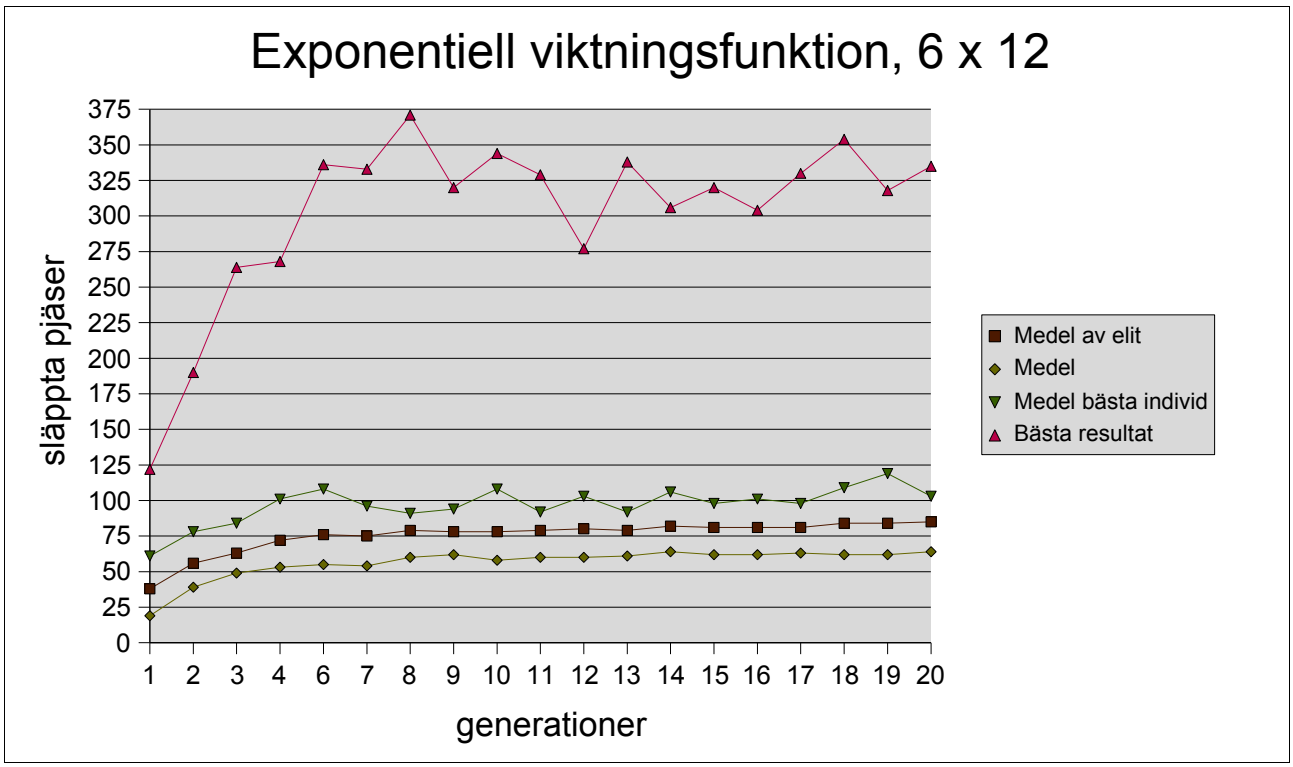
Jag gjorde en körning med den exponentiella viktningsfunktion på 10 x 20 planen och evolverade 20 generationer. Det bästa resultatet inträffade under generation 19 med 14611 släppta pjäser vilket motsvarar ungefär 5800 rader. Mot slutet låg medelvärdet för mellan 1200 och 1300 pjäser och medelvärdet för elitselektionen låg runt 2000.

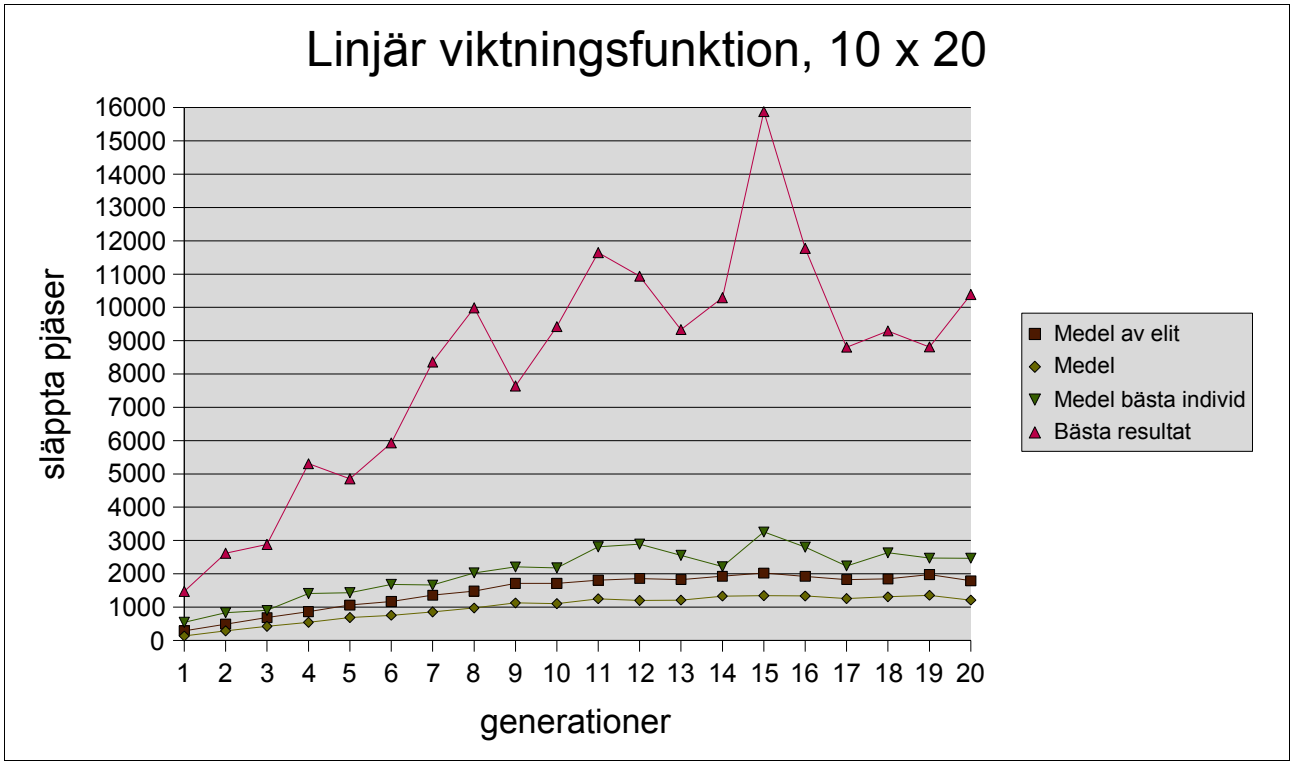
För den linjära viktningsfunktionen på den stora spelplanen var efter 20 generationer det bästa resultatet 15880 pjäser (6400 rader), medelvärde mot slutet mellan 1200 och 1300 pjäser och för elitselektionen runt 1900.

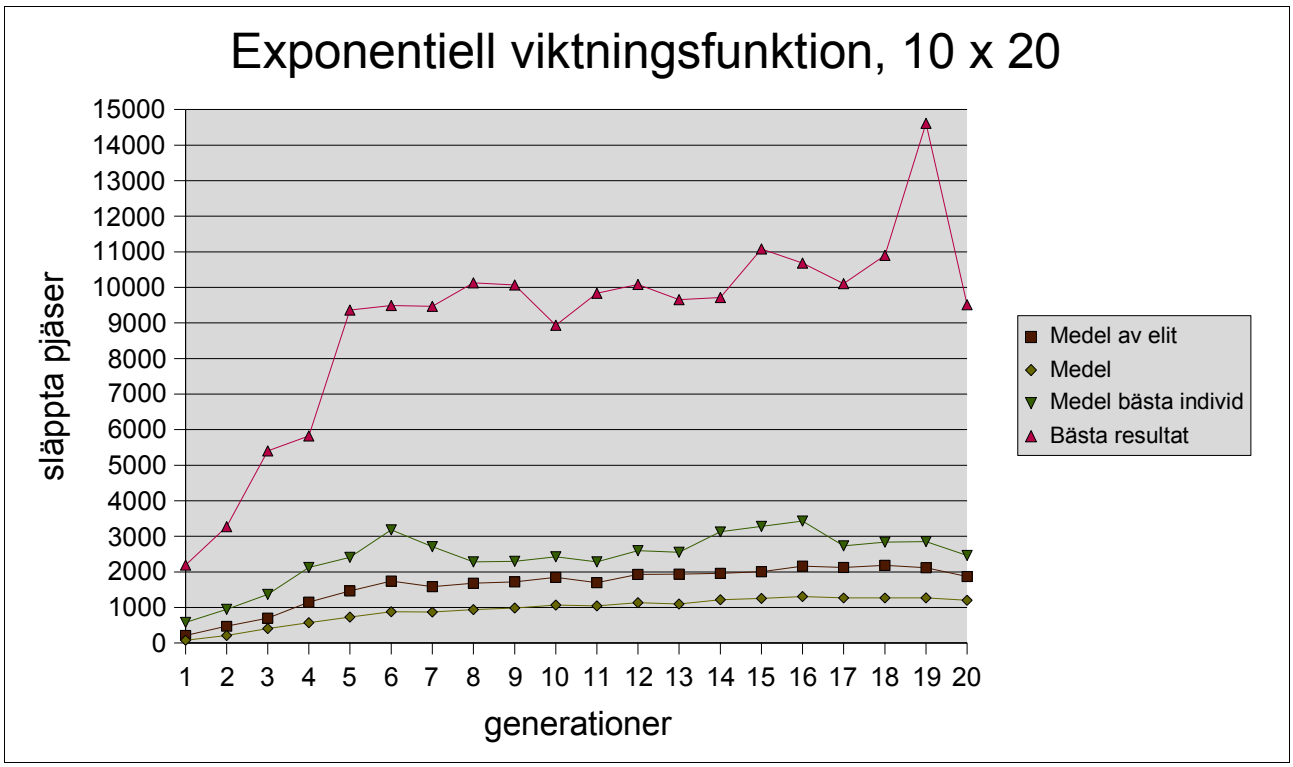
För den mindre spelplanen (6 x 12) lyckades den linjära funktionen lite bättre än den exponentiella funktionen som hade ett högsta värde på 354 pjäser och mot slutet ett medelvärde mellan 62 och 62. Den linjära hade ett högsta värde på 448 pjäser och medelvärdet på slutet var mellan 72 och 75.

Att evolvera 20 generationer med den exponentiella funktionen på 20 x 10 spelplanen tar ungefär två timmar på en Pentium Celeron D med en klockfrekvens 2,5 Ghz. Det kan jämföras med Böhm et al. (2005), som behövde 3 månader för att evolvera 30 generationer, dock med 12 mått istället för 7. Då nådde å andra sidan resultat på några miljoner rader, medan jag lyckades som bäst runt 15000. Även med 6 vikter klarade de över 5 miljoner rader.


Linjär viktningsfunktion, 6 x 12
släppta pjäser
0
50
100
150
200
250
300
350
400
450
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
generationer
Medel av elit
Medel
Medel bästa individ
Bästa resultat


Exponentiell viktningsfunktion, 6 x 12
släppta pjäser
0
25
50
75
100
125
150
175
200
225
250
275
300
325
350
375
1 2 3 4 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
generationer
Medel av elit
Medel
Medel bästa individ
Bästa resultat





Diskussion

Det resultat som jag har lyckats få fram står sig slätt jämfört med de två exempel på evolutionärt optimerad tetris som jag givit. Då Böhm et. al (2005) använde sex av de sju heuristiker jag använt lyckades de mycket bättre än vad jag gjorde. Gällande Xtbot vet jag inte hur många heuristiker den använder, men den lyckas i alla fall mycket bättre än vad

jag gjorde. Jag kan inte utesluta att de låga resultaten kan bero på ett programmeringsfel, även om jag inte tror att så är fallet.

Troligtvis har de låga resultaten att göra med den evolutionsalgoritm som jag använt, inte den tetrisspelande algoritmen. Faktorer som kan ha påverkat resultatet kan vara initiala värden på vikterna, storlek på den konstanta skalningen av vikterna, sannolikheten och storleken på mutation och elitselektionens storlek. Även selektionsalgoritmen för korsning kan ha spelat in. Eftersom Böhm et al. (2005) inte angav några detaljer för dessa parametrar kunde jag bara pröva mig fram. Jag har ingen större erfarenhet genetiska algoritmer och ett djupare studium av dessa och vidare experimenterade hade troligtvis lett till högre resultat. Å andra sidan är jag glad att mina körningar av evolutionsalgoritmen inte tog tre månader.

Referenser

Heidi Burgiel, How to lose at Tetris. Mathematical Gazette:p. 194 (July 1997).

Niko Böhm, Gabriella Kókai and Stefan Mandl, An Evolutionary Approach To Tetris. In Proceedings of The Sixth Metaheuristics International Conference (MIC2005) (2005).

Erik D. Demaine, Susan Hohenberger and David Liben-Nowell, Tetris is Hard, Even to Approximate. In Proceedings of the 9th International Computing and Combinatorics Conference (COCOON 2003) (2003).

Internet

Fahey1: http://www.colinfahey.com/2003jan_tetris/tetris_standard_specifications.htm, ”Tetris AI: Standard Tetris Specifications”

Fahey2: http://www.colinfahey.com/2003jan_tetris/tetris_world_records.htm, ”Tetris AI: World Records”

Fahey3: http://www.colinfahey.com/2003jan_tetris/tetris_strategy.htm, ”Tetris AI: Strategy Algorithms”