fast

Algoritmer og teknikker for sjakkprogrammer - teori og praksis

Rune Djurhuus, stormester i sjakk

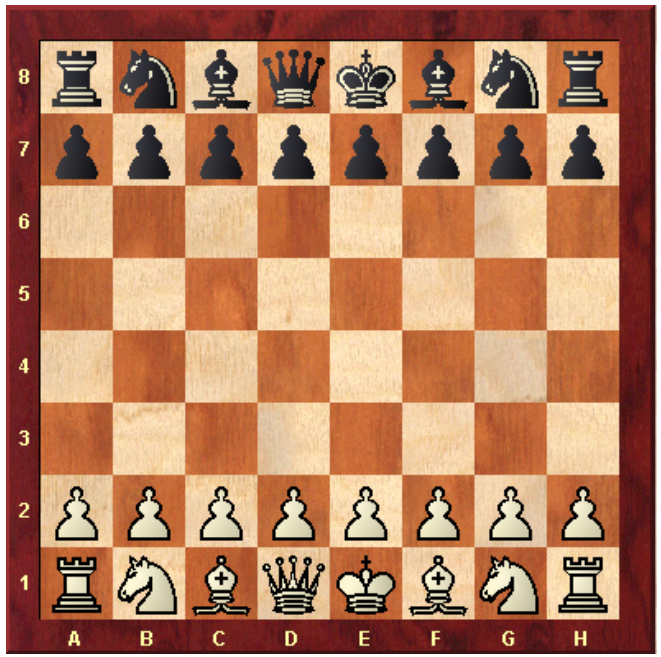
25 October 2007

Innhold

- Sjakkspillets kompleksitet
- Historien til computersjakk
- Søketre og stillingsvurdering
- Minimax: Den grunnleggende søkealgoritmen
- Avskjæringsteknikker:
  - Alpha-beta avskjæring
  - Det beste trekket først
  - Killer-move heuristikk
  - Null-trekk heuristikk
- Iterativt dypere dybde-først-søk (IDDFS)
- Søketreutvidelser
- Transposition table
- Andre utfordringer
- Sluttspillsdatabaser: Tablebases
- Demo

Sjakkspillets kompleksitet

- 20 mulige starttrekk, 20 mulige svartrekk, osv
- 400 mulige stillinger etter 2 ply (halvtrekk)
- 197 281 stillinger etter 4 ply
- 7^13 stillinger etter 10 ply (5 hvite og 5 svarte trekk)
- Omtrent 40 lovlige trekk i en typisk stilling
- Finnes omtrent 10^120 mulige sjakkpartier

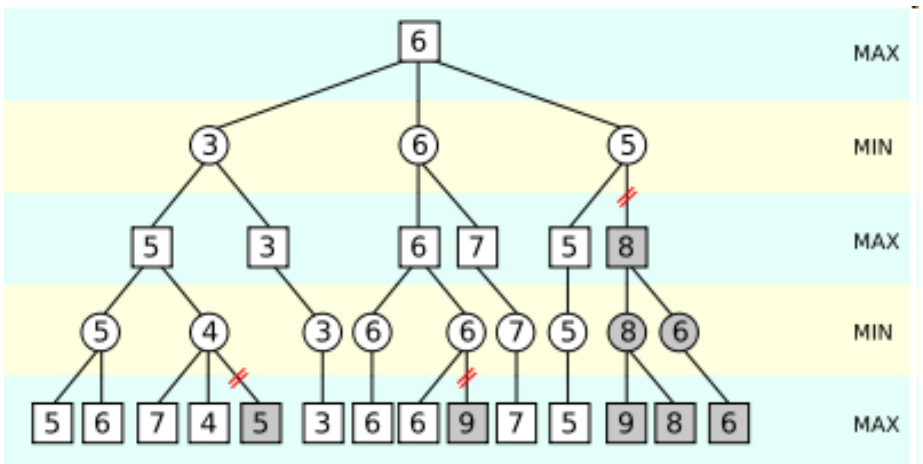


Historien til computersjakk

- Sjakk passer bra for computere
  - Klart definerte regler
  - Forholdsvis enkelt å bedømme stillinger
  - “Passe” stort søketre
- 1950: Programming a Computer for Playing Chess (Claude Shannon)
- 1951: Første program (på papir) som spiller sjakk (Alan Turing)
- 1958: Første dataprogram som kan spille et helt parti
- 1981: Cray Blitz vinner turnering mot mennesker i Mississippi og får mesterrating
- 1989: Deep Though taper 0-2 mot Garry Kasparov
- 1996: Deep Blue vinner et parti mot Kasparov, men taper kampen 2-4.
- 1997: Oppgradert Deep Blue beseirer Kasparov 3,5-2,5
- 2005: Hydra knuser Michael Adams 5,5-0,5
- 2006: Vladimir Kramnik taper 2-4 mot Deep Fritz (PC-program)

Søketre og stillingsvurdering

- Søketre (noder er stillinger, kanter er sjakktrekk)
- Bladnodene er sluttstillinger, som må vurderes
- Enkel vurdering: Sjakk matt? Hvis ikke, tell opp brikkene…
- Nodene blir merket med tallvurdering av stillingen

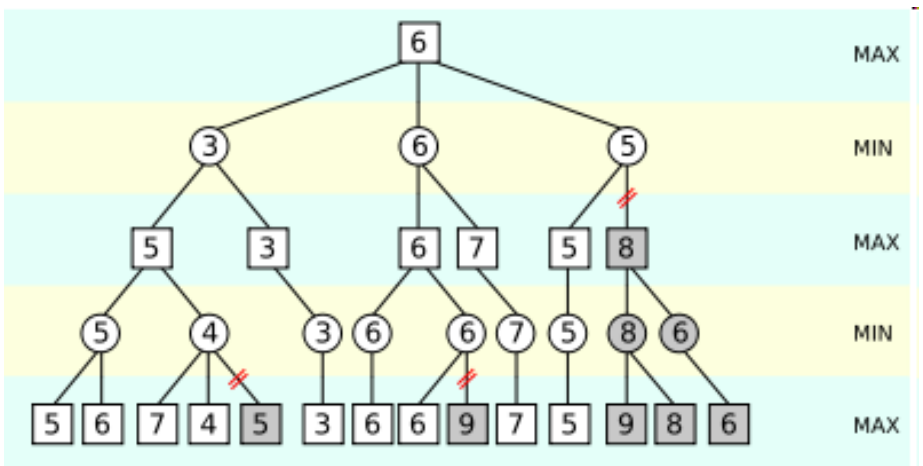


fast

Minimax: Den grunnleggende søkealgoritmen

- Minimax: Anta at både hvit og svart spiller de beste trekkene. Skal maksimere hvits vurdering (score)
- Utfør dybde-først søk og vurder bladnodene
- Velg den maksimale barnenoden med hvit i trekket
- Velg den minimale barnenoden med svart i trekket
- Branching factor er 40 i en typisk sjakkstilling!

Hvit
Svart
Hvit
Svart
Hvit



ply = 0
ply = 1
ply = 2
ply = 3
ply = 4

fast

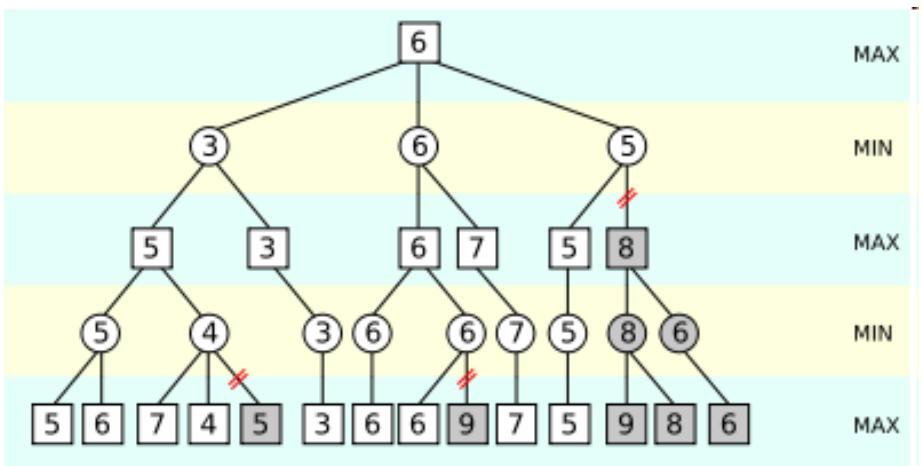
Avskjæringsteknikker:

- Kompleksiteten for å søke d ply fremover er: $O(b*b*...*b) = O(b^d)$
- Med branching factor (b) på 40 er det avgjørende å avskjære ("prune") søketreet

Alpha-beta avskjæring

- Bruk tidligere kjente max- og min-verdier til å begrense søketreet
- Alpha-verdi: Hvit er garantert minst så bra vurdering (start: -∞)
- Beta-verdi: Svart kan ikke "score" bedre enn dette (start: +∞)
- Når beta blir mindre enn alpha, kan ikke stillingen oppstå ved beste spill
- Hvis søketreet under blir evaluert fra venstre til høyre, så kan vi hoppe over de grå subtreeene.
- Samme hva slags verdier vi får der, så kan ikke de påvirke evalueringen til rotnoden. Prøv!
- "Stillingen er så bra for hvit (eller svart) at motstanderen ved beste spill ikke vil tillate varianten som gir denne stillingen"

Hvit
Svart
Hvit
Svart
Hvit



ply = 0
ply = 1
ply = 2
ply = 3
ply = 4

Det beste trekket først

- Med alpha-beta avskjæring, men der vi undersøker hvits dårligste trekk først, får vi fortsatt O(b*b*...*b) = $O(b^d)$
- Undersøker vi alltid beste trekk først (også rekursivt), kan man vise at kompleksiteten blir O(b*1*b*1*b*1…) = $0(b^{d/2}) = 0(\sqrt{b^d})$
- Vi kan dermed søke dobbelt så dypt uten å bruke mer ressurser.
- Konklusjon: Det er veldig viktig å prøve å undersøke det beste trekket først.

Killer-move heuristikk

- Killer-move heuristikk baserer seg på antagelsen om at et trekk som ga en stor avskjæring i et subtre, også er et godt trekk i andre noder i søketreet.
- Prøv derfor drepertrekkene først!

Null-trekk heuristikk

- Alpha-Beta cutoff: “Stillingen er så bra for hvit (eller svart) at motstanderen ved beste spill ikke vil tillate varianten som gir denne stillingen”
- Null-trekk heuristikk er basert på at i de fleste stillinger er det en fordel å være i trekket
- La spilleren (eks. hvit) som nettopp har gjort et trekk, gjøre et trekk til (to trekk på rad), og utfør et grunnere (gjerne 2-3 ply mindre) og dermed raskere søk fra denne stililngen.
- Hvis det grunne søket gir en cutoff-verdi (eks. dårlig stillingsverdi for hvit), så betyr det med stor sannsynlighet at søketreeet kan avskjæres i denne stillingen uten å gjøre et dypt søk: Et dypt søk vil trolig ikke gi bedre verdi for hvit enn det grunne søket antydet siden to hvite trekk på rad ikke hjalp.
- Veldig effektivt avskjæringsteknikk!
- Men pass på sjakk og sluttspill (hvor det ofte forekommer trekktvang - der alle trekk forverrer stillingen).

Iterativt dypere dybde-først-søk (IDDFS)

- Siden det er så viktig å undersøke det beste trekket først, så kan det lønne seg å gjøre et grunnere søk først, og så bruke cutoff-verdier (alphfa/beta) der som startverdier i et dypere søk
- Siden halvparten av nodene er på nederste nivå i et balansert tre, er det relativt billig å gjøre ekstra, grunnere søk

Søketreutvidelser

- PC-sjakkprogram i dag regner gjerne 10-12 ply fremover (Deep Blue regnet 12 ply mot Kasparov i 1997, Hydra (64 noder med FPGAs) regner 18 ply)
- I tillegg er det viktig å utvide søket i bladnodestillinger som er “ustabile”.
- Gode søkeutvidelser er å undersøke alle mulige sjakker og slag av brikker helt til stillingen har roet seg ned.

Transposition table

- Samme stilling kan ofte forekomme via flere trekkrekkefølger
- Alle sjakkprogrammer har derfor en transposition table (stillings-cache).
- Implementeres gjerne som en hash-tabell med sjakkstillingen som nøkkel.
- Kan slippe å undersøke store subtrær om igjen.
- Sjakkprogrammer bruker opptil halvparten av tilgjengelig minne til hash-tabellen, det viser hvor viktig den er.

Andre utfordringer

- Trekkgenerator
- Effektiv lagring av sjakkbrettet
- Åpningsbibliotek som passer computeren
- Stillingsbedømmelsen:
  - Tradisjonelt har sjakkcomputere basert seg på dype søk med enkle vurderinger
  - Men det beste PC-programmet i dag, Rybka, bruker veldig mye tid på stillingsvurderingen

Sluttspillsdatabaser: Tablebases

- Sjakkcomputere spiller sluttspill med 3-6 brikker igjen på brettet perfekt ved å slå opp i en database (tabell).
- Disse sluttspillsdatabasene kalles Tablebases
- Retrograde-analyse: Tabellene genereres ved å ta utgangspunkt i mattstillinger/remisstillinger og arbeide seg bakover i søketreet inntil alle stillinger er markert som tapt, vunnet eller remis.
- Heftige komprimeringsalgoritmer (Eugene Nalimov)
- Alle 3-5 brikker sluttspill og noen 6 brikker sluttspill får plass på 21 GB.

Demo

– Demo: ChessBase med sjakkmotor (chess engine).

Tusen takk

Presenter: Rune Djurhuus
Contact: Rune.Djurhuus@fast.no
runed@ifi.uio.no

Version 1.0