INF 4130
6. oktober 2011

Stein Krogdahl

- Dagens program:
  - Første time:

    Kap 23.5: Spilltrær og strategier for spill med to spillere
  - Andre time, gjesteforelesning:

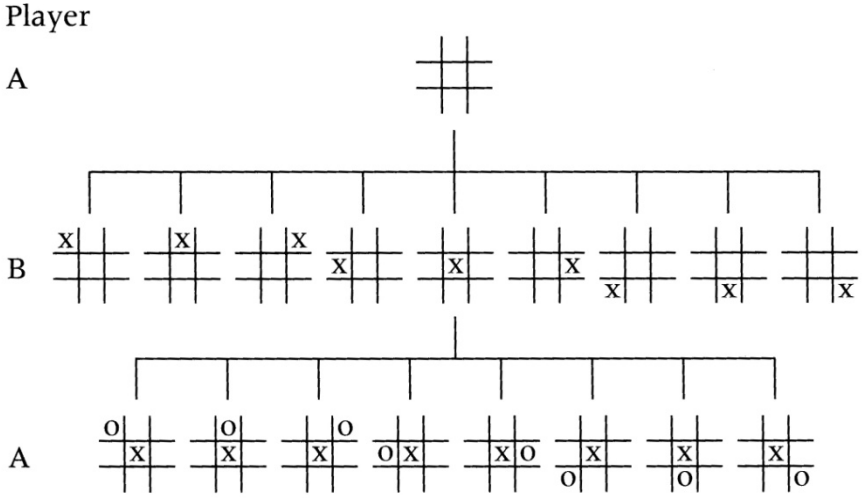
    Rune Djurhuus: Om sjakkspillende programmer

    (Ikke pensum, egne foiler legges ut)
- Neste uke:
  - Uavgjørbarhet

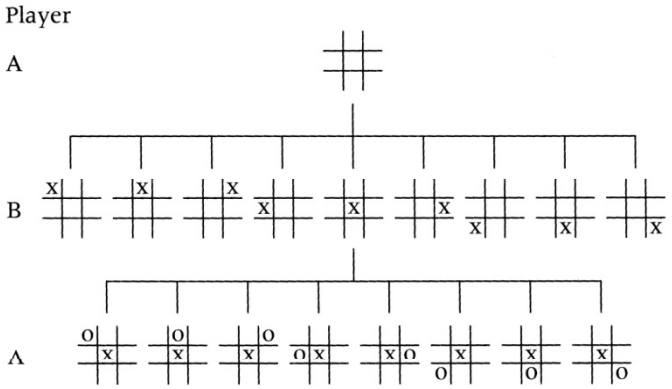
    Dino Karabeg

Kapittel 23.5: Spill, spilltrær og strategier

- Vi kunne lage et tre av avgjørelser for f.eks. ”15-spillet” eller leting etter en beste rundtur for en ”travelling salesman”, og prøve å finne veien til en løsningsnode eller en beste vei.
  - Men da er det bare én ”spiller”, og man er ute etter en ”nøytral” løsning
- Om det er to spillere som spiller mot hverandre blir saken annerledes. Det den ene ser som en bra situasjon ser den andre som dårlig.
  - Trærne av mulige spill blir ofte kjempestore. For sjakk estimert til 10^{100} noder.
  - Men selv for et enkelt spill som ”tick-tack-toe” kan ved rett fram utlegging for 9! = 362 880 noder.
- Starten av et tre for tic-tac-toe blir slik:
- Spilleren som starter er A
  - Vi skal også gjøre alle vurderinger ut fra A synspunkt (inntil videre)



Treet blir fort stort



9 noder

9*8 = 72 noder

9*8*7 = 504 noder

9*8*7*6 = 3024 noder

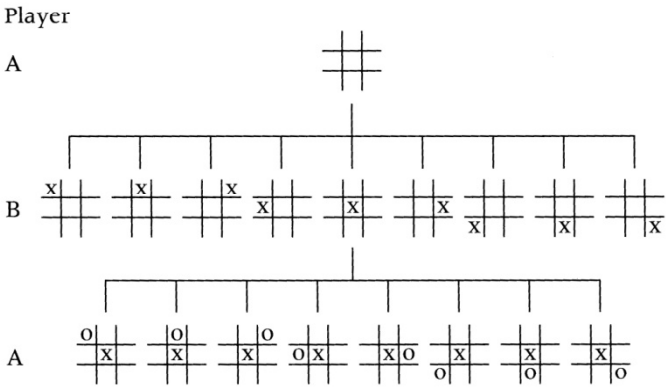
9*8*7*6*5 = 15120 noder

. . . . . .

9*8*7*6*5*4*3*2*1 = 362 880 noder

Ved å gå dybde først gjennom dette treet vil man aldri behøve å lagre mer enn 9 noder samtidig, men det tar tid å gå gjennom 362 880 noder (og oftest mange fler!).

Men om vi slår sammen like noder …



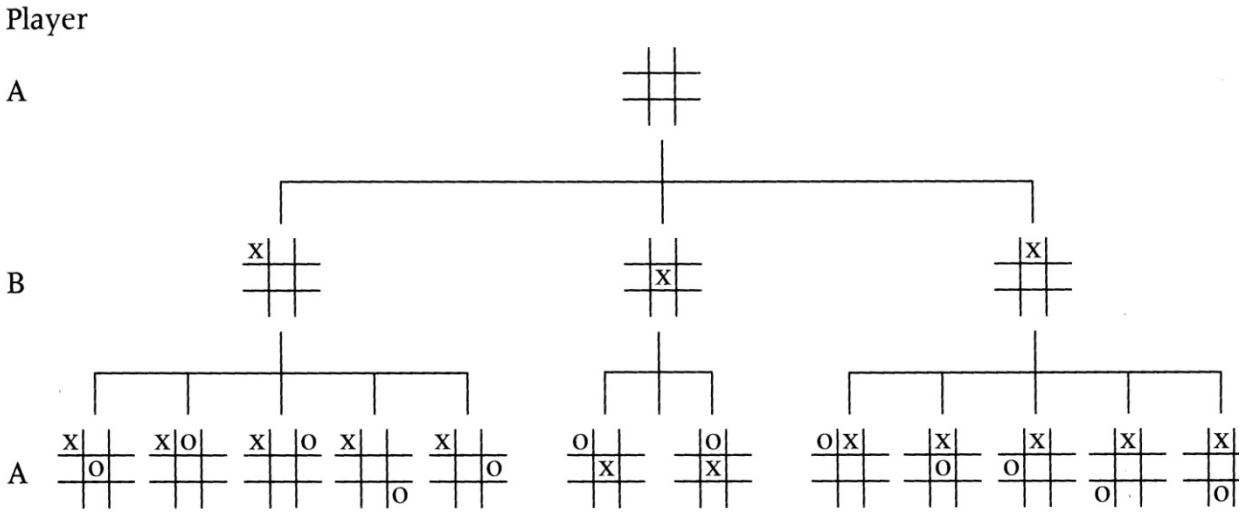
9 noder	1 node
9*8 = 72 noder	9 noder
9*8*7 = 504 noder	72 **forskjellige** noder
9*8*7*6 = 3024 noder	252 **forskjellige** noder
9*8*7*6*5 = 15120 noder	756 **forskjellige** noder
9*8*7*6*5* 4 = 60480 noder	1260 **forskjellige** noder
……	1680 **forskjellige** noder
9*8*7*6*5*4*3*2*1 = 362 880 noder	……
	126 **forskjellige** noder

(Og enda har vi *ikke* tatt bort symmetrier)

Men man kan i noen spill tjene masse på å kjenne igjen like noder, og ikke gjøre undersøkelsen om igjen (minner litt om dynamisk programmering).

Men dette krever som regel mye plass! (Selv om det her krever bare max 1680 noder)

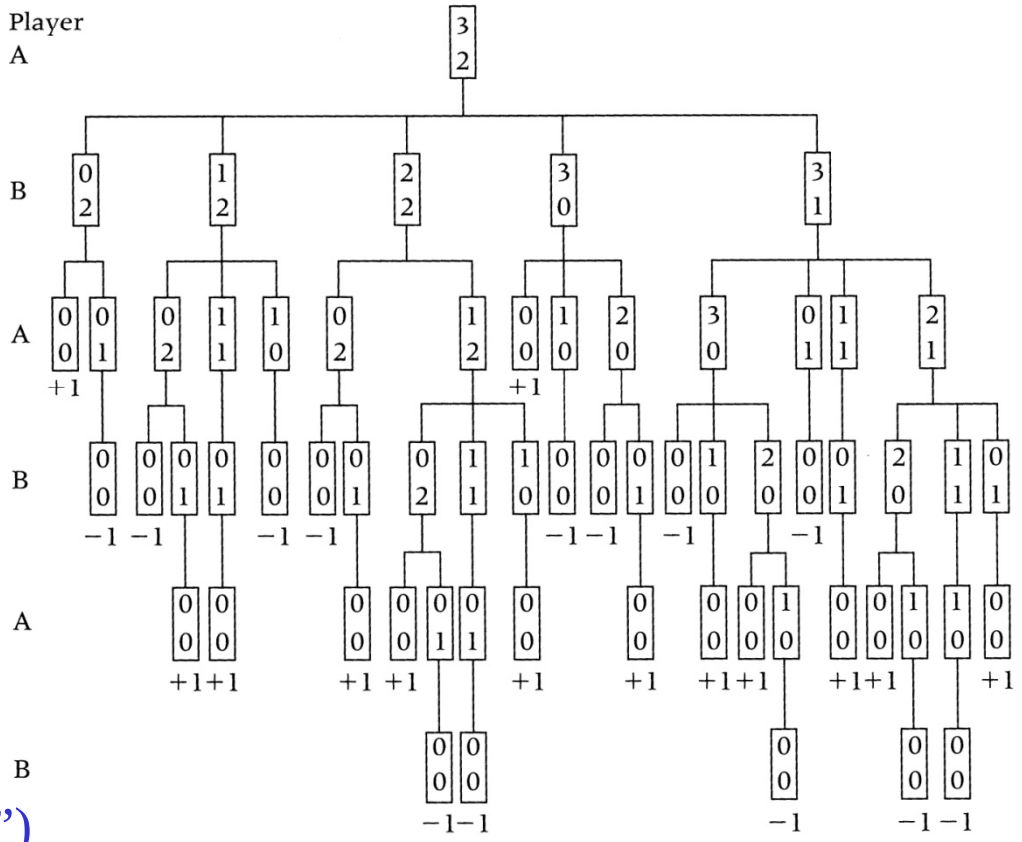
Ta bort symmetrier etc.



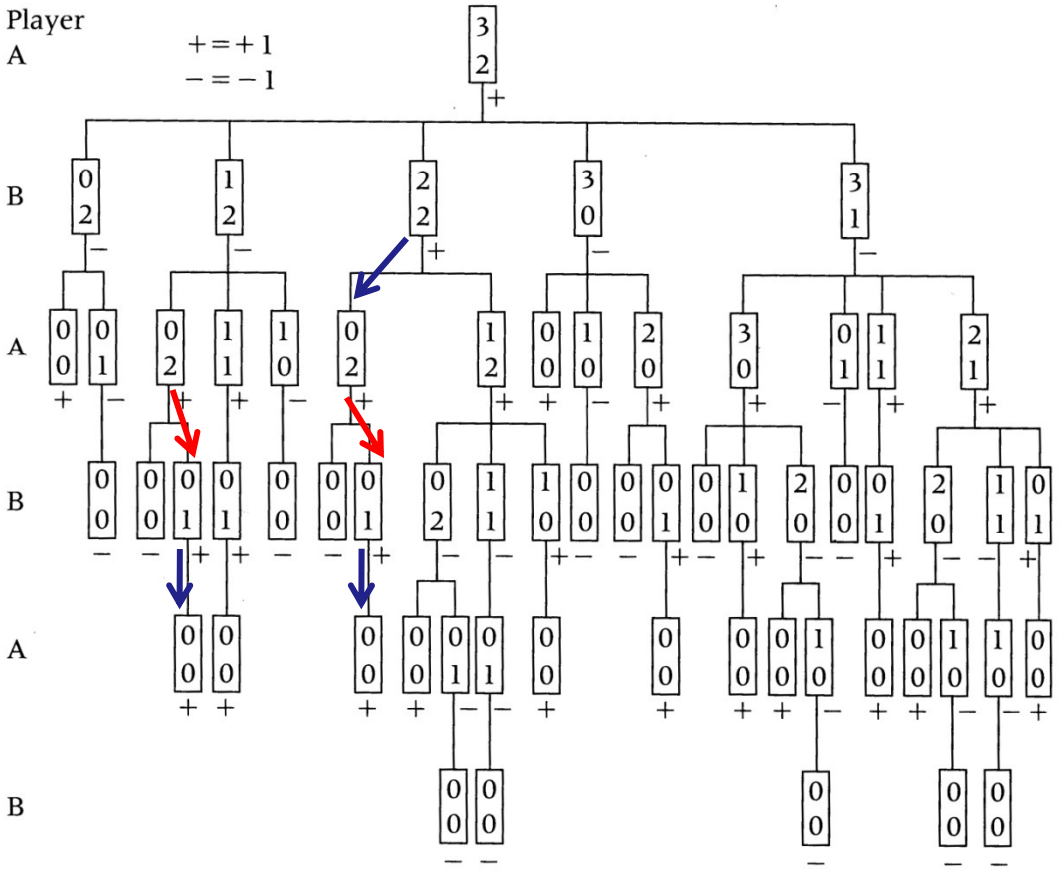
- Man kan også spare mye på symmetri-betrakninger:
  - Ta bare med de barna som er essensielt forskjellige
  - Altså her, slå sammen løsninger som er symmetriske av hverandre
  - Kan også legges inn i letingen etter *samme situasjon*, på forrige foil.
  - Reduserer plassbehovet betraktelig i noen spill

Fystikk-spillet Nim

- Vi ser bare på ”null-sum-spill”. Altså, når den ene spilleren vinner noe så taper den andre tilsvarende.
- Enkleste variant: Det er to utfall, A vinner (+1) og B taper (-1), eller omvendt
- En del spill har også ”uavgjort”, for eksempel tic-tac-toe. Gies verdien 0.
- Spillet Nim:
  - Start med to (eller flere) bunker med fyrstikker.
  - Antall: m og n
  - En spiller kan ta så mange man vil fra én bunke, men minst 1
  - Den som tar siste fyrstikken har tapt.
- Her blir det aldri uavgjort
- Om vi starter med m=3 og n=2 er hele spilltreet tegnet ut her:
- Verdien sett fra A er satt på hver av slutt-nodene (”teminal-nodene”)

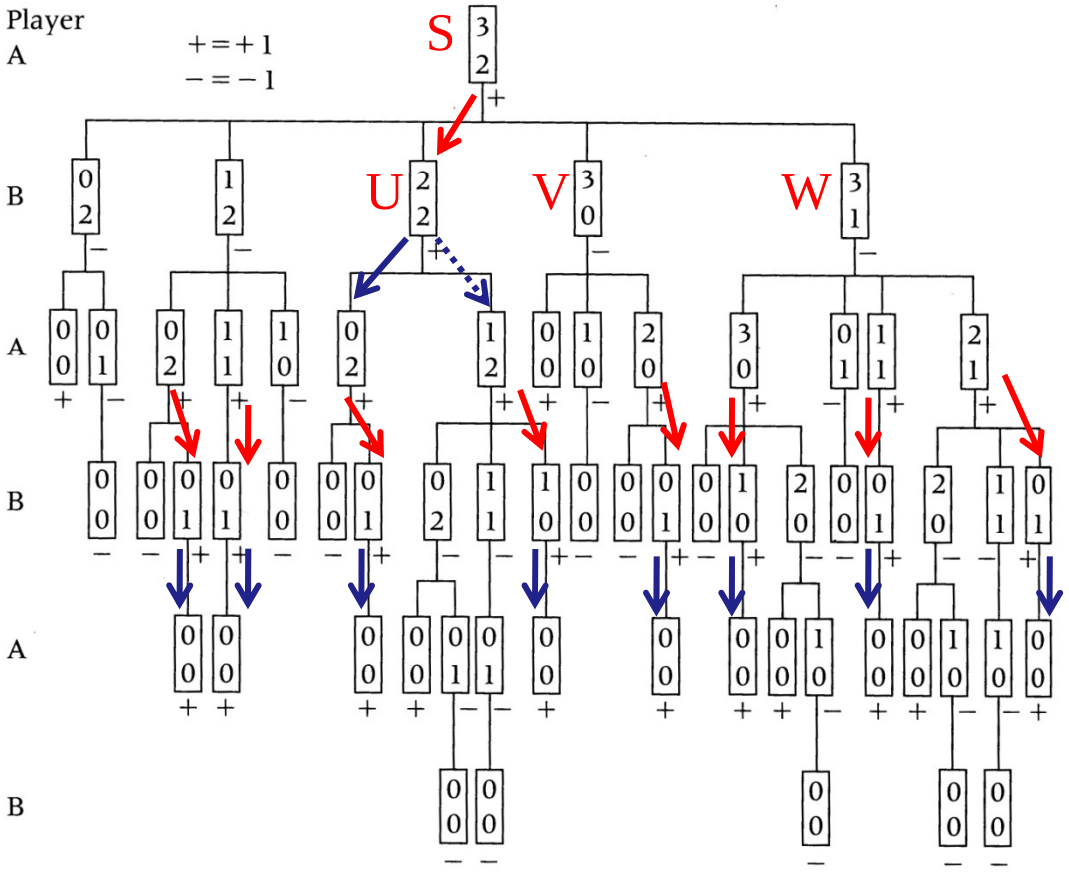


Hvordan finne strategi for å vinne, sett fra A?



- Vi angir hele tiden verdien av en node ut fra A's synspunkt
- A vil gjøre optimale trekk sett fra seg: Vil velge å gå til node med størst verdi
- A må regne med at B gjør optimale trekk sett fra seg.
- Siden verdiene noteres sett fra A, vil da B trekke til den *minste* subnoden
- For å få vite verdien av en node må vi derfor vite verdiene i alle subnodene
- Dette kan vi gjøre ved *dybde først* gjennomgang, og å beregne verdiene postfiks (på tilbakevei).

Hvordan finne strategi for å vinne, sett fra A?

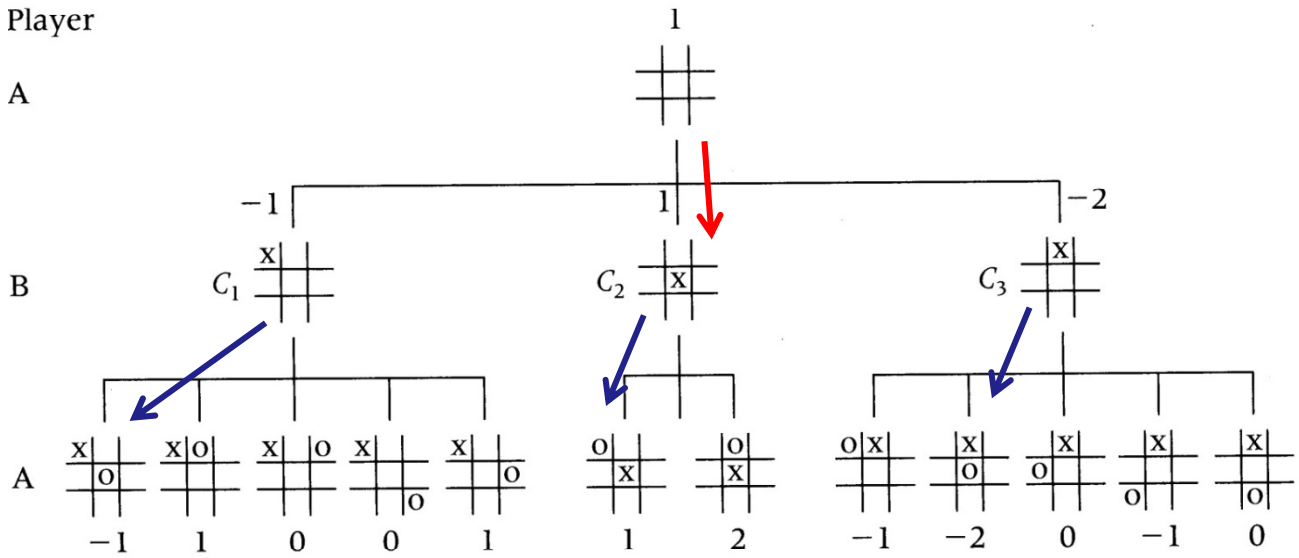


- Dette kan vi gjøre ved *dybde først* gjennomgang, og å beregne verdiene postfiks (på tilbakevei).
- Samtidig kan vi i hver *indre* pluss-node angi hvilken vei vi kan gå (det kan være flere) som vil lede A til seier.
- Dette er angitt på figuren, slik det vil være etter at dybde-først-søket er ferdig.

Optimalisering:

- I startsituasjonen S kan A allerede etter å ha sett på tre av sine subtrær (fra venstre) se at S må bli en vinst-node siden A kan trekke til vinst-noden U. Dermed behøver den ikke se på subtrærne under V og W.
- Slik optimalisering skal vi se mer på om et øyeblikk.

Oftest kan man ikke gå gjennom hele spilltreet

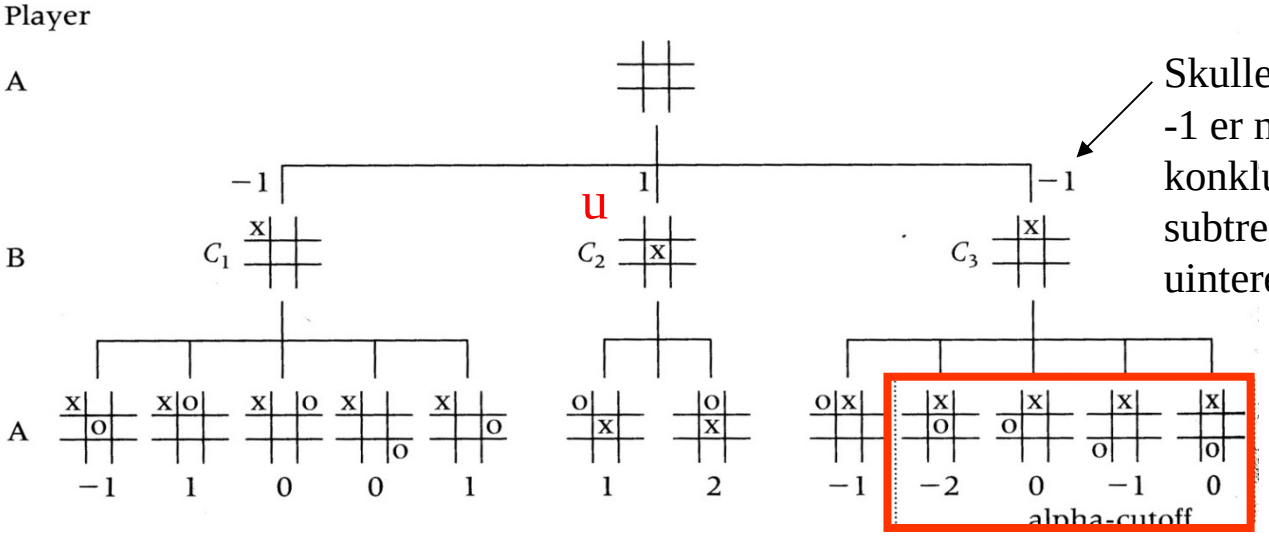


- Man kan da for eksempel bare gå til en viss dybde, og her beregne et estimat (ut fra en gitt heuristikk-funksjon) for hvor god situasjonen er (sett fra A).
- I figuren over går vi til dybde 3 (eller 2, avhengig av hvordan man definerer dybde)
- Heuristikk her: Antall ”vinnende” linjer for A minus det tilsvarende for B.
- A vil derfor her trekke til C_2, som ser bra ut
- Men denne heuristikken er ikke lur når vi har gjort noen trekk. Tar ikke hensyn til at å vinne er bedre enn alle heuristikker. Gir derfor vinst-node verdi $+\infty = 9$
- Da får vi en god strategi for A, et stykke på vei.
- For: Om begge spiller perfekt, så blir det uavgjort i tic-tac-toe.
- F.eks. må slutt-situasjonen til høyre (uten vinner, uavgjort) få verdien 0:
- NB: Vanskelig avveiing mellom dybde og tidkrevende heuristikk!

O	O	X
X	X	O
O	X	X

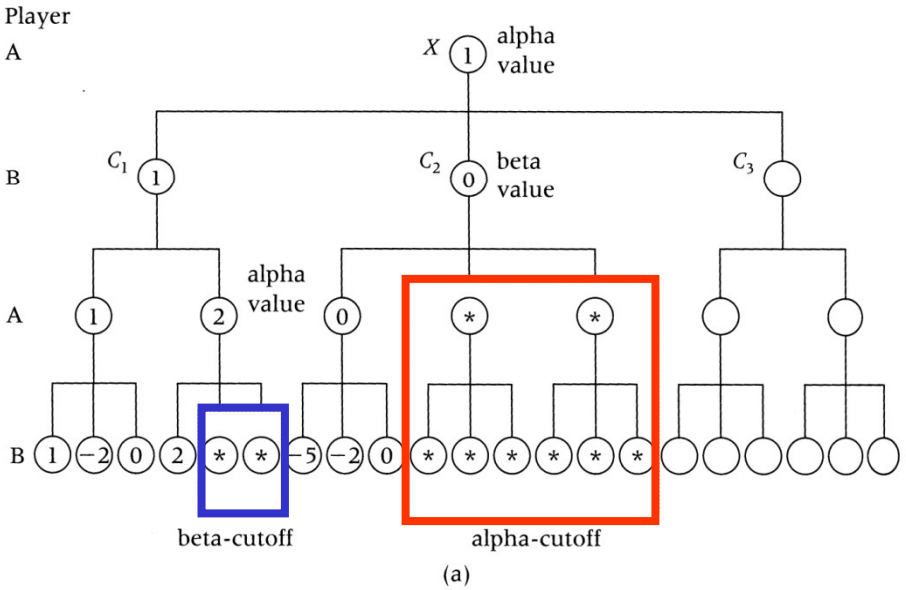
Alfa-beta-avskjæring

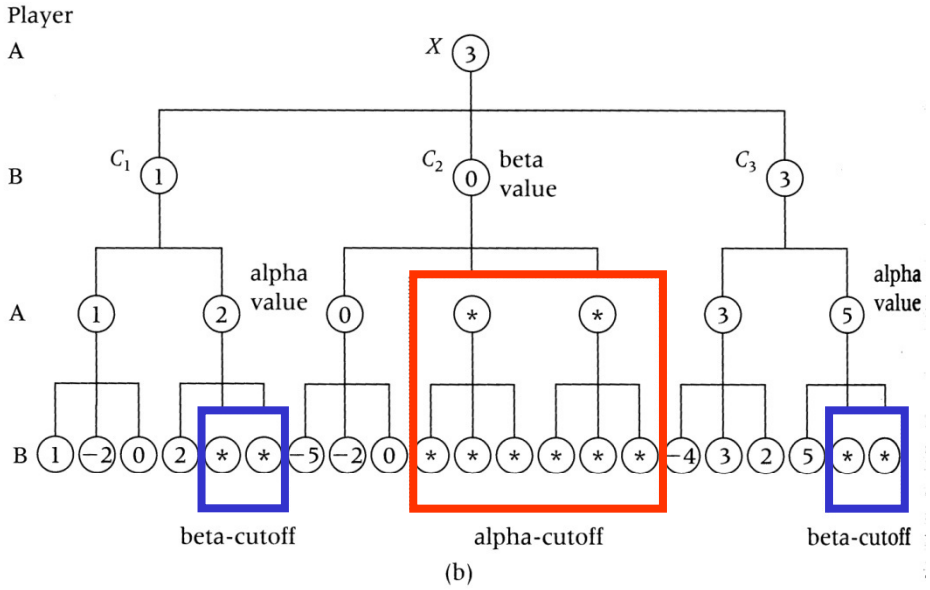
- Ved siden av å forsøke å redusere treet ved å se på symmetrier og/eller kjenne igjen noder som allerede er analysert, kan man gjøre ”alfa-beta-avskjæring”
- Dette er som følger:
  - Jeg, A, vurderer trekk ett og ett trekk fra en gitt A-situasjon (øverst)
  - Jeg har allerede sett så gode trekk at jeg kan oppnå verdien u (av C_1 og C_2, u = 1)
  - Jeg ser så på neste potensielle trekk som fører til situasjonen C_3, og ser at B herfra kan gjøre et trekk som fører til en god situasjon for B (men dårlig for A), og som sett fra A har verdien v = -1. Da kan ikke verdien i C_3 bli større en v (husk at B minimaliserer!)
  - Om da v < u så ”gidder” ikke A gå videre å studere om B skulle ha et enda bedre trekk. Det kan ikke gjøre A mer lysten på trekket til C_3. C_2 er helt sikkert bedre.



Skulle egentlig blitt -2, men -1 er nok for A til å konkludere med at resten av subtreet under C_3 er uinteressant.

Eksempler på alfa-beta-avskjæring





- Avskjæring som gjøres ut fra A's betrakning kalles alfa-avskjæring
- Samme betrakning som gjøres fra B-tilstander, men med mimimalisering. Disse kalles beta-avskjæring
- Ved siden av ser vi eksempler på både alfa- og beta-avskjæring

- Når vi skal implementere alfa-beta-avskjæring kan det være greit å skifte synspunkt for hvert nivå.
  - Da skal det *alltid* maksimaliseres
  - Men man må skifte tegn på verdiene når man skifter nivå.
- En slik implementasjon står på neste foil

Alfa-beta-søk (snur verdiene hver gang)

```
real function ABNodeValue(
        X,   // Noden vi vil ha alfa/beta-verdien for. Barn: C[1], C[2], … , C[k]
        numLev,   // Antall nivåer som står igjen
        parentVal) // Alfa/beta-verdien fra forelder-noden (-LB fra foreldren)
// returverdi: Den endelige alfa/beta-verdien for noden X
{       real LB;  // Løpende LowerBound for alfa/beta-verdi for denne noden

        if <X er terminal-situasjon> then {return <verdien av denne sett fra X>;}
        else if numLev = 0 then {return <estimat av kvaliteten av denne, sett fra X>;}
        else {
           LB := - ABNodeValue(C[1], NumLev-1, ∞);
           for i :=  2 to k do {
               if  LB >= parentValue  then {return LB;}
               else { LB := max(LB, - ABNodeVal(C[i], Numlev-1, -LB) ); }
           }
         }
         return LB;
}
Startkall:      situasjonsKvalitet := ABNodeValue(rotnoden, 10, -∞)
```

Trykkfeil i algoritmen

- Det er noen enkle trykkfeil i programmet på side 741 i boka (*ikke* rettet i ny utgave):
  - ”AB” manglet i navnet på prosedyren, der den blir kalt rekursivt
  - Det mangler en sluttparentes på enden av linja der **max** kalles
- De *er* rettet i skissen på forrige foil