Chapter 5

Моделиране на игри

How Intelligent is Deep Blue?
Drew McDermott
http://www.cs.yale.edu/HTML/YALE/CS/homepage/op-ed/deep_blue.html

5.1 Лекции

5.1.1 Връзка с ИИ

- Добре дефинирани задачи, изискващи интелигентност.
- Трудни задачи (1): предполагат неопределеност (не може предварително да се определи ходът на противика).
- Трудни задачи (2): пространството на състоянията в повечето случаи е огромно (шах: разклонения = 35, дълбочина = 50, възли = 35^{100}) .
- Въпреки това хората играят добре без да се притесняват от огромното пространство на търсене.
- Подходяща област за изследване на методите за търсене.

5.1.2 Формализация на задачата

- Частен случай на задачата за търсене в пространството на състоянията.
- Състоянията, в които играта свършва се наричат *терминални*.
- Функция на полезност (utility) – присвоява стойност на терминалните състояния (например: победа = 1, загуба = −1, равен = 0).
- Дървото на търсене е разделено на нива, на всяко от които са ходовета на един играч. Играчите условно могат да се нарекат MIN (този, който се стреми на минизира фунцията на полезност) и MAX (стараещ се да максимизира функцията на полезност).

5.1.3 Минимаксен алгоритъм

- Генериране на пълното дърво на играта до терминалните възли (виж фигура 5.1).
- Изчисляване на функцията на полезност за всеки възел отдолу нагоре, започвайки от терминалите (виж фигура 5.2).
- За всеки възел MAX се избира максималната стойност на неговите наследници (т.е. ходът, който максимизира полезността).
- За всеки възел MIN се избира минималната стойност на неговите наследници (т.е. ходът, който минимизира полезността).
- По този начин се определя оптималния ход в корена на дървото (изборът на текущ ход).

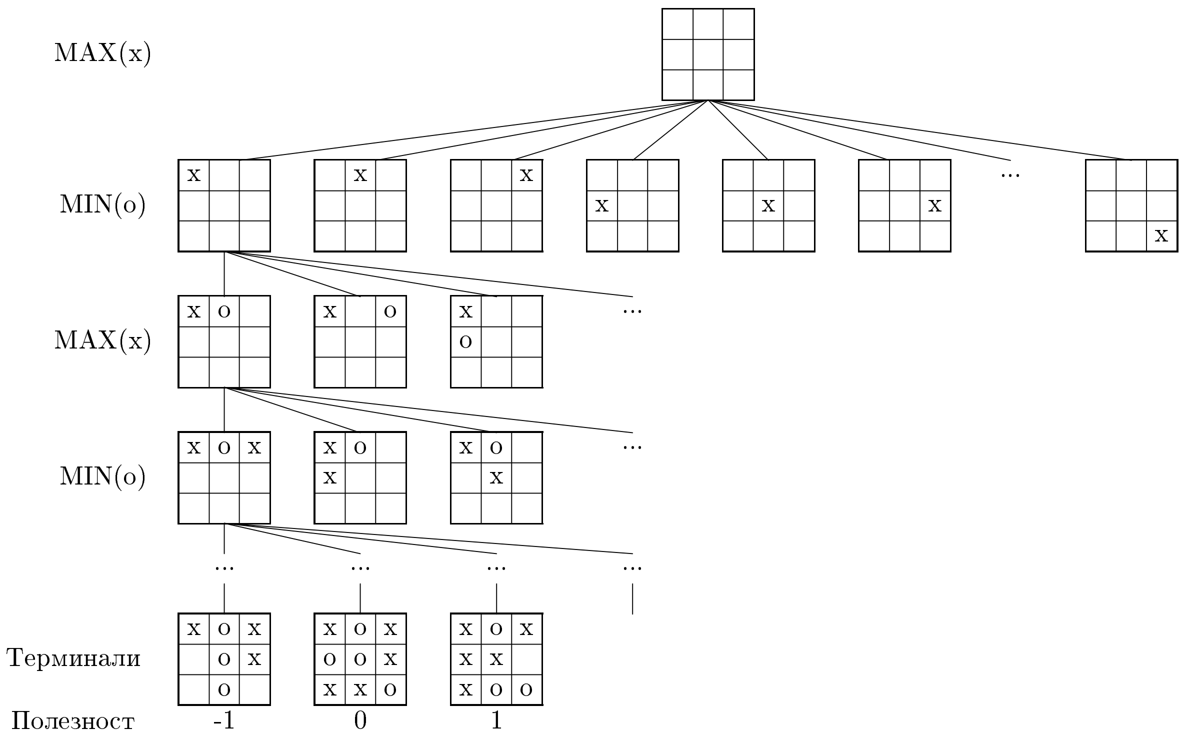

Figure 5.1: Дърво на играта "кръстчета и нули"

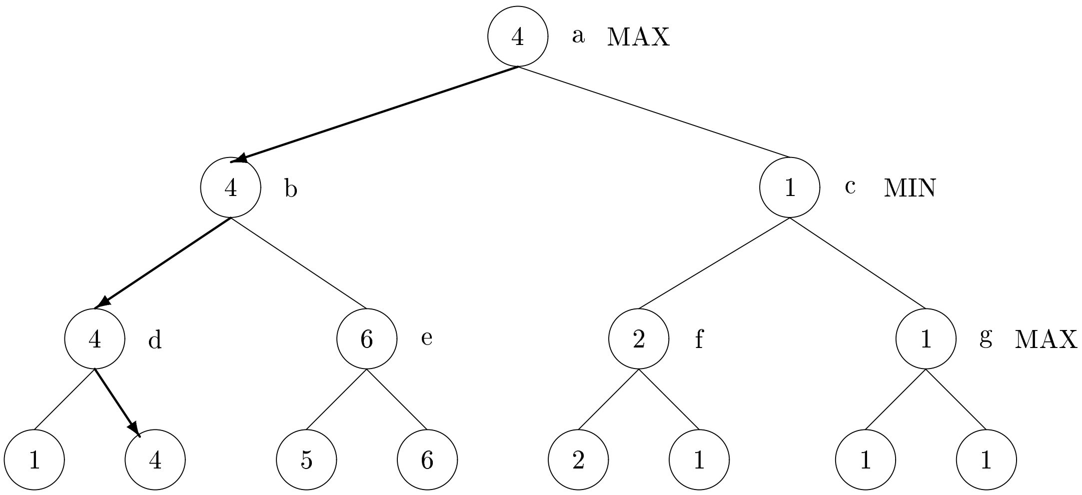

Figure 5.2: Прилагане на минимаксния алгоритъм към примерно игрово дърво (ходовете, избрани по този алгоритъм с означени със стрелки с плътна линия)

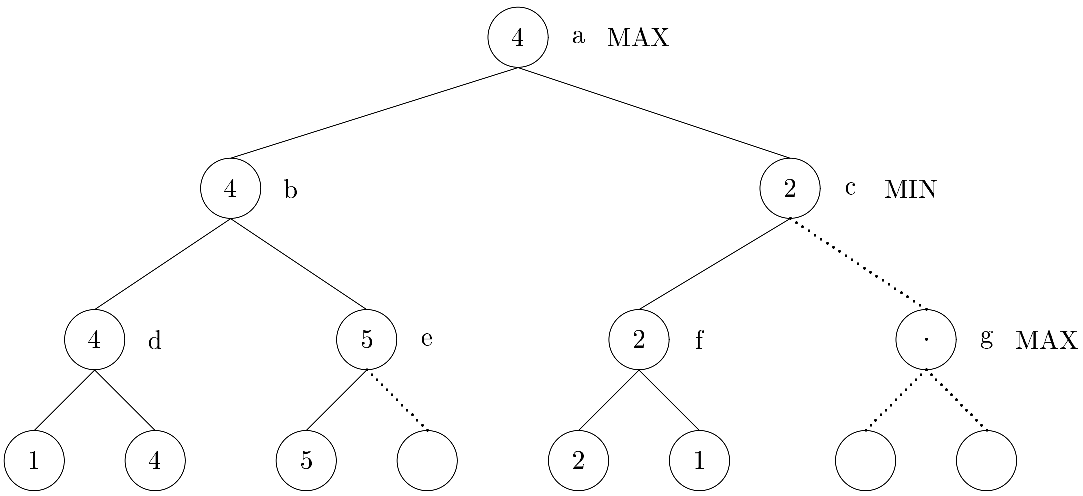

Figure 5.3: Прилагане на алфа-бета процедура към примерно игрово дърво

5.1.4 Усъвършенствуване на минимаксния алгоритъм

- Освен за съвсем прости игри (например 3X3 "кръстчета и нули") алгоритъмът е неприложим поради огромния брой възли в дървото.
- Отсичане на дървото от определен възел надолу и намиране на евристична оценка на полезността.
- Идеалната евристика би трябвало да отразява вероятността, че MAX ще победи (MIN ще загуби) при зададента позиция на играта.
- Пример за проста евристика (шах): тегловна сума от броя на притежаваните фигури (недостатък: не отчита позицията).
- Определяне на нивото на отсичане (предполага се, че евристиките за по-долните нива на дървото се намират по лесно и са по точни):
  - Предварително задаване на граница на дълбочината d.
  - Използуване на итеративно търсене по нива при предварително зададено време за изчисление.
  - Прилагане на евристична оценка само в "спокойните" позиици на играта (позиции, които не предполагат рязка промяна на оценката при по-нататъшната игра).
  - Проблемът с хоризонта: някои ситуации изискват продължаване на търсенето още по-дълбоко (не могат да се оценят, защото са "зад хоризонта"). Например превръщането на пешка в царица при шаха.

5.1.5 Алфа-Бета отсичане (ефективна реализация на минимаксния алгоритъм)

- Приближено изчисляване на стойностите на полезността (отбележете разликите между фигури 5.2 и 5.3), което обаче е достатъчно за определяне на верния ход.
- Търсене в дълбочина.
- Два текущи параметъра по време на търсене в игровото дърво:
  - α = максималната стойност за MAX, получена до момента.
  - β = минималната стойност за MIN, получена до момента.
- Ако MIN получи стойност по-голяма от β, то неговите съседи на същото ниво могат да не се оценяват (отсичане на дървото). Например в позиция левия наследник на e (фигура 5.3) MIN получава 5, което е по-голямо от $\beta = 4$, следователно десния наследник на e не се оценява.

- Ако MAX получи стойност по-малка от α, то неговите съседи на същото ниво могат да се пропуснат. Например в позиция f (фигура 5.3) MAX получава 2, което е по-малко от $\alpha = 4$, следователно позиция g може да се пропусне.
- Недостатък: зависи от реда на обхождане на наследниците на позициите в дървото. Ако най-перспективните ходове се разглеждат първи, то се получава намаляване на сложността от $O(b^d)$ до $O(b^{\frac{d}{2}})$.

5.1.6 Игри с елемент на случайност

- Например игри със зарове (табла).
- Освен MIN и MAX дървото на играта се разширява с още един вид нива – нива със случайни възли.
- Случайните нива съдържат толкова възела, колкото са възможните изходи при хвърлянето на заровете, с различна роля за играта (например за таблата има 21 различни изхода).
- Вариант на минимаксния алгоритъм, включващ случайност.
- Нарастване на сложността: $O(b^d n^d)$ (n – случайни изхода). В реални игри рядко се използува голяма дълбочина.
- Аналогът на алфа-бета отсичането е по-сложен. Необходимо е на се намери граница на полезността, която обаче е средна стойност и зависи от броя на изследваните възли. Това може да стане като се поставят граници на стойностите на терминалите (напр. $+1$ и -1) и използувайки броя на случайните възли да се изчисли граница и за случайните възли.

5.2 Семинарни упражнения

- Представяне на игровото дърво в общия случай (позиции, ходове, оценки на терминалите).
- Реализация на минимаксния алгоритъм.
- Реализация на алфа-бета алгоритъма.
- Реализация на играта "кръстчета и нули" с размерност 3X3.
- Намиране на подходяща оценъчна функция за нетерминални позиции на играта "кръстчета и нули". Реализация на алгоритъм за итеративно търсене по нива (минимакс и алфа-бета).

5.3 Лабораторни упражнения

- Експериментални сравнения (брой на изследвани възли) на минимакс и алфа-бета върху симулационно игрово дърво (фигура 5.2, файл games.pl).
- Програмиране на играта "кръстчета и нули" и експерименти с минимакс и алфа-бета търсене (брой на изследвани ходове и бързодействие).