Поиск путей в искуственном интеллекте

Наумов Д.А., доц. каф. КТ

Экспертные системы и искусственный интеллект, 2019

Содержание лекции

Поиск пути

Чтобы решить задачу, когда нет ясного вычисления допустимых решений, используетмся поиск пути.

- с помощью деревьев игр для игр с двумя игроками;
- с помощью деревьев поиска для игр для одного игрока.

Эти подходы опираются на дерево состояния:

- корневой узел - начальное состояние
- ребра - потенциальные ходы, которые преобразуют состояние в новое состояние.

Дерево игры

Поиск - сложная проблема, базовая структура не вычисляется полностью из-за взрывного роста количества состояний. Шашки $5\dot{1}0^{2}0$ различных позиций на доске. Деревья, в которых выполняется поиск, строятся по требованию по мере необходимости.

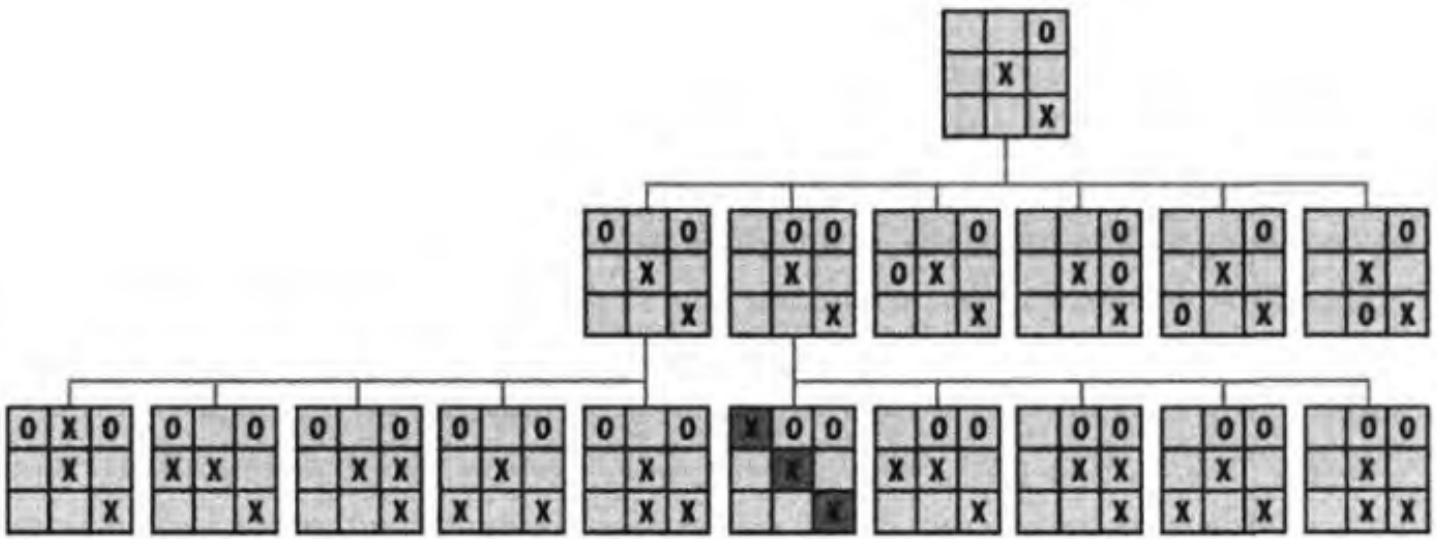
Дерево игры

- Два игрока по очереди выполняют ходы, которые изменяют состояние игры из ее первоначального состояния.
- Есть много состояний, в которых любой игрок может выиграть в игре.
- Могут быть некоторые состояния «ничьей», в которых не выигрывает никто.
- Алгоритмы поиска пути увеличивают шанс, что игрок выиграет или обеспечит ничью.

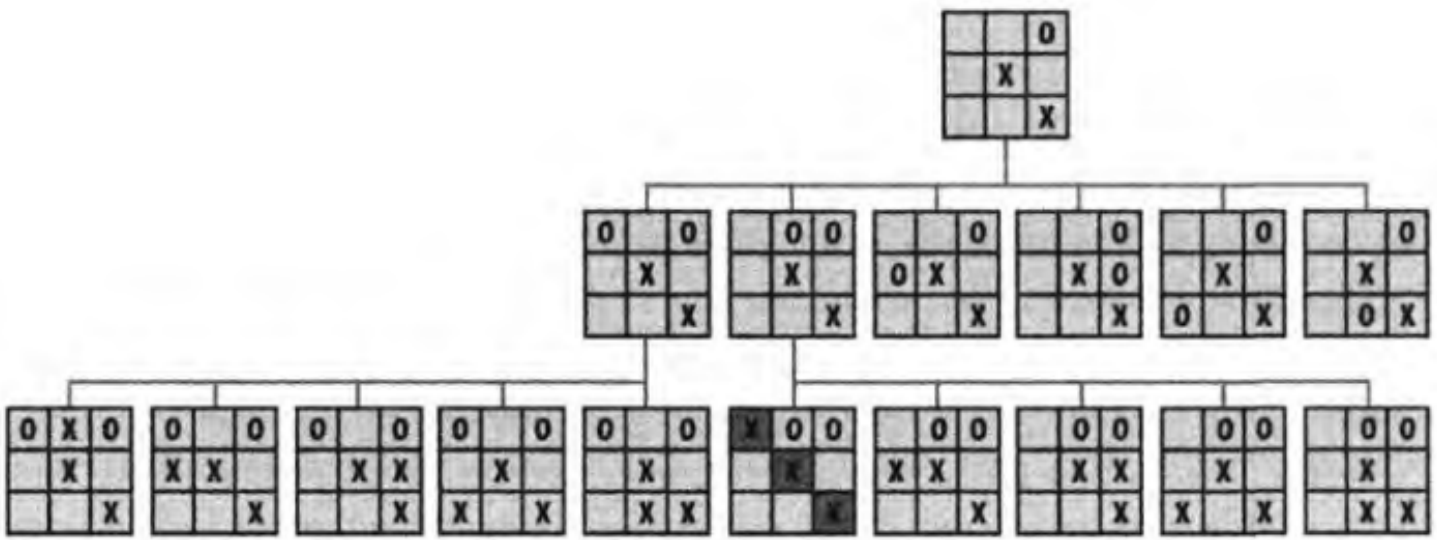
Дерево поиска

- Один игрок начинает игру с некоторого начального состояния и делает допустимые ходы до достижения желаемого целевого состояния.
- Алгоритм поиска пути определяет точную последовательность ходов, которые преобразуют исходное состояние в целевое.

Крестики-нолики: 765 уникальных позиций (без учета отражений и поворотов), 26830 возможнных игр.



Дерево игры известно также как дерево И/ИЛИ, поскольку оно образуется из двух различных типов узлов.



- Верхний узел является узлом ИЛИ, так как целью игрока О является выбор только одного из шести доступных ходов на среднем уровне.
- Узлы среднего уровня являются узлами И, потому что цель (с точки зрения игрока 0) состоит в том, чтобы убедиться, что все ходы противника (показанные как дочерние узлы на нижнем уровне) по-прежнему приведут к победе 0 или ничьей.

- В сложной игре дерево игры никогда не может быть вычислено полностью изза его размера.
- Целью алгоритма поиска пути является определение на основе состояния игры хода игрока, который максимизирует его шансы на победу в игре (или даже гарантирует ее).
- Таким образом, мы преобразуем множество решений игрока в задачу поиска пути в дереве игры.

Пример: игра в шашки

- доска размером 8x8 с начальным набором из 24 шашек (12 белых и 12 черных).
- может ли игрок, делающий первый ход, обеспечить ничью или победу?
- Schaeffer, J., N. Burch, Y. Bjornsson, A. Kishimoto, M. Muller, R. Lake, P Lu, and S. Sutphen, «Checkers is solved», Science Magazine, September 14, 2007, 317(5844): 1518-1522, http://www.sciencemag.org/cgi/content/abstract/317/5844/1518.

Два типа подходов для поиска задач поиска:

Тип A

- Рассмотрим различные разрешенные ходы для обоих игроков для фиксированного количества будущих ходов и определим наиболее благоприятное положение, получающееся в результате для исходного игрока.
- Затем выберем начальный ход, который движет игру в этом направлении.

Shannon, C., «Programming a computer for playing chess», Philosophical Magazine, 41(314): 1950, http: //tinyurl.com/ChessShannon-pdf.

Тип Б

- Добавим некоторые адаптивные решения, основанные на знании игры, а не на статических оценках.
- Говоря более точно, а) оценим перспективные позиции на столько ходов вперед, сколько необходимо для выявления устойчивой позиции, в которой вычисления действительно отражают силу полученной позиции, и б) выберем подходящие доступные ходы.
- Этот подход пытается предотвратить возможность бессмысленной траты драгоценного времени

Рассмотрим семейство алгоритмов типа А, который предоставляет подход общего назначения для поиска в дереве игры наилучшего хода для игрока в игре с двумя игроками.

- Minimax;
- AlphaBeta;
- NegMax.

Существует несколько способов сделать поиск более интеллектуальным:

- Выбор порядка и количества применяемых разрешенных ходов
  - При рассмотрении доступных ходов для данного состояния игры сначала следует вычислить ходы, которые вероятнее других приведут к успешным результатам.
  - Кроме того, можно отказаться от определенных ходов, которые заведомо не приводят к успешным результатам.
- Выбор состояния игры для «обрезки» дерева поиска
  - По мере выполнения поиска может быть обнаружена новая информация, которую можно использовать для устранения состояний игры, которые (в некоторый момент времени) были выбраны в качестве части поиска.

Функция статических оценок

оценивает состояния игры в промежуточных точках в процессе вычислений с последующим упорядочением множества доступных ходов так, чтобы сначала испытывались ходы, с более высокой вероятностью ведущие к выигрышу

Функция статической оценки должна:

- учитывать различные особенности позиции в дереве игры
- возвращать целочисленный балл, который отражает относительную силу позиции с точки зрения игрока.

Samuel, A., «Some studies in machine learning using the game of checkers», IBM Journal 3(3): 210-229, 1967, http://dx.doi.org/10.1147/rd.116.0601.

- оценка позиции на доске путем рассмотрения двух десятков параметров, таких как сравнение количества шашек у игрока и его противника или возможностей размена шашек.

Крестики-нолики

Функция BoardEvaluation, определенная Нилом Нильссоном. Nilsson, N., Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, 1971
Пусть nc(gs,p) — количество строк, столбцов или диагоналей в состоянии игры gs, в которых игрок p все еще может выстроить в ряд три своих значка.
Затем мы определим score(gs, p) следующим образом:

- $+\infty$ - если игрок p победил в игре в состоянии gs;
- $-\infty$ - если противник игрока p победил в игре в состоянии gs;
- $ncgs, p - nc(gs, opponent)$ - если ни один игрок в состоянии игры gs не победил.

Представление состояния

Каждый узел дерева игры или дерева поиска содержит всю информацию о состоянии, известную как позиция в игре.

Пример: шахматы, рокировка возможно, если:

1. ни одна из этих фигур еще не делала хода,
2. промежуточные клетки доски пустые и в настоящее время не находятся под боем фигур противника
3. король в настоящее время не находится под шахом.

Состояние игры:

- должно храниться наиболее компактно;
- может быть значительно уменьшено путем удаления эквивалентных состояний, которые можно получить простым поворотом или отражением

Сложные представления для шахмат или шашек: Pepicelli, G., «Bitwise optimization in Java: Bitfields, bitboards, and beyond,» O'Reilly on Java.com, February 2, 2005, http://www.onjava.com/pub/a/onjava/2005/02/02/bitsets.html.

Вычисление доступных ходов

Чтобы найти наилучший ход, в каждом состоянии должно быть возможно вычисление доступных ходов игрока.

Коэффициент ветвления

среднее количество ходов, которые разрешены из отдельного состояния.

- кубик Рубика - 13,50
- четыре в ряд - 7,00
- шашки - 6,14 (1,20 для позиций со взятием, 7,94 без взятия)
- Го - 361,00

Если коэффициент ветвления у игры оказывается высоким, а ходы не упорядочены должным образом на основе некоторой вычисляемой меры успеха, слепой поиск в дерева оказывается неэффективным.

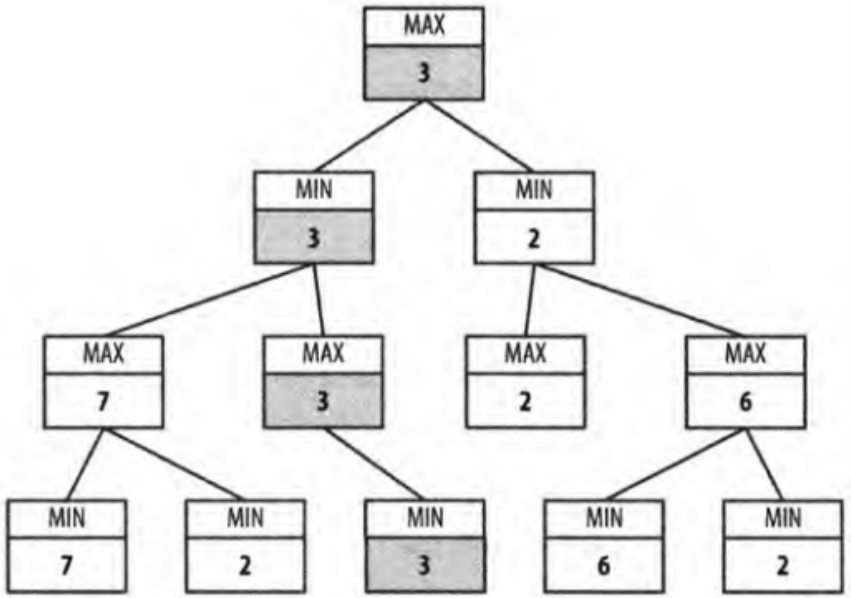
Максимальная глубина расширения

- Из-за ограниченных ресурсов памяти некоторые алгоритмы поиска ограничивает степень расширения деревьев поиска и игр.
- Этот подход проявляет свою слабость, в первую очередь, в играх, в которых продуманная стратегия формируется с помощью последовательности ходов.
- Фиксированная глубина расширения формирует «горизонт», за который поиск не может заглянуть, и это часто препятствует успешному поиску.
- Для игр с одним игроком фиксация максимальной глубины означает, что алгоритм не в состоянии найти решение, которое лежит сразу за горизонтом.

Например, в шахматах нередка жертва фигуры для получения потенциальных преимуществ. Если эта жертва происходит на краю максимального расширения, выгодное состояние игры может не быть обнаружено.

Minimax

- Для заданной конкретной позиции в дереве игры с точки зрения начального игрока программа поиска должна найти ход, который привел бы к наибольшим шансам на победу (или хотя бы на ничью).
- Но вместо только текущего состояния игры и доступных для этого состояния ходов программа должна рассматривать любые ходы противника, которые он делает после хода нашего игрока.



Minimax

- функция оценки score (state, player), которая возвращает целое число, представляющее оценку состояния игры с точки зрения игрока player;
- дерево игры расширяется путем рассмотрения будущих состояний игры после последовательности из n ходов.
- Каждый уровень дерева по очереди представляет собой уровень MAX (на котором цель заключается в обеспечении выгоды для исходного игрока путем максимизации вычисленного состояния игры) и MIN (на котором цель заключается в обеспечении выгоды для противника путем минимизации вычисленного состояния игры).
- На чередующихся уровнях программа выбирает ход, который максимизирует score (state, initial), но на следующем уровне она предполагает, что противник будет выбирать ход, который минимизирует значение score(state, initial).

Minimax

- функция оценки score (state, player), которая возвращает целое число, представляющее оценку состояния игры с точки зрения игрока player;
- дерево игры расширяется путем рассмотрения будущих состояний игры после последовательности из n ходов.
- Каждый уровень дерева по очереди представляет собой уровень MAX (на котором цель заключается в обеспечении выгоды для исходного игрока путем максимизации вычисленного состояния игры) и MIN (на котором цель заключается в обеспечении выгоды для противника путем минимизации вычисленного состояния игры).
- На чередующихся уровнях программа выбирает ход, который максимизирует score (state, initial), но на следующем уровне она предполагает, что противник будет выбирать ход, который минимизирует значение score(state, initial).

Алгоритм Minimax

Наилучший, средний и наихудший случаи: $O(b^{ply})$

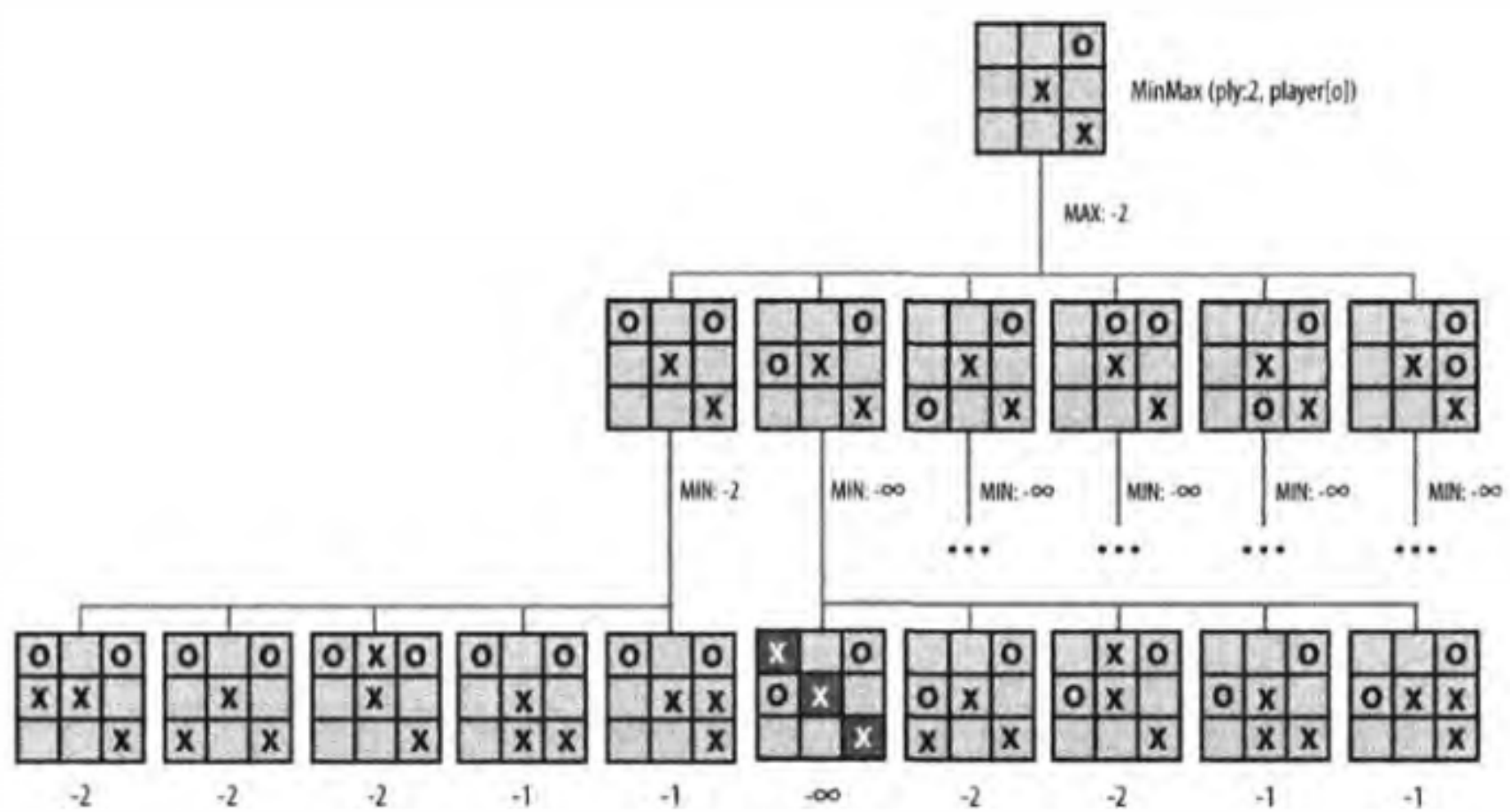
```
bestmove (s, player, opponent)
    original = player                                              ❶
    [move, score] = minimax (s, ply, player, opponent)
    return move
end

minimax (s, ply, player, opponent)
    best = [null, null]
    if ply равен 0 или допустимых ходов нет then                   ❷
        score = оценка состояния s для исходного игрока
        return [null, score]
    foreach допустимый ход m игрока player в состоянии s do
        Выполнение хода m в состоянии s
        [move, score] = minimax(s, ply-1, opponent, player)        ❸
        Отмена хода m в состоянии s
        if player == original then                                 ❹
            if score > best.score then best = [m, score]
        else
            if score < best.score then best = [m, score]
    return best
end
```

❶ Запоминаем исходного игрока, так как оценка состояния всегда выполняется с его точки зрения.

❷ Если больше ходов не остается, игрок побеждает (или проигрывает), что эквивалентно достижению целевой глубины предпросмотра.

❸ При каждом рекурсивном вызове выполняется обмен игрока и противника, отражающий чередование ходов.

❹ Последовательное чередование уровней между MAX и MIN.

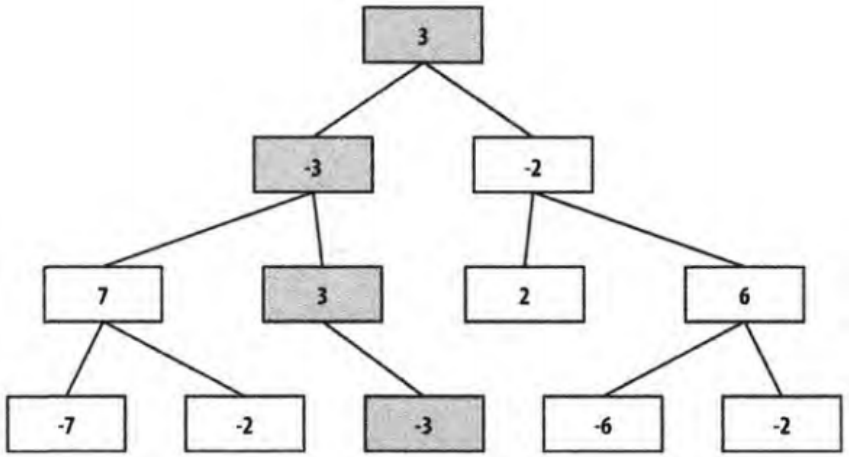

MinMax (ply:2, player[o])
MAX: -2
MIN: -2
MIN: -∞
MIN: -∞
MIN: -∞
MIN: -∞
MIN: -∞
-2
-2
-2
-1
-1
-∞
-2
-2
-1
-1

Анализ алгоритма Minimax

- Если для каждого состояния игры имеется фиксированное количество ходов b (или даже когда количество доступных ходов уменьшается на единицу с каждым уровнем), общее количество состояний игры, в которых выполняется поиск при предпросмотре глубиной d, представляет собой $O(b^n)$ демонстрируя экспоненциальный рост.
- Ограничения глубины предпросмотра могут быть устранены, если дерево игры достаточно мало для полной оценки за приемлемый промежуток времени.
- Поскольку мы предполагаем, что и игрок, и противник играют без ошибок, мы должны найти способ остановить расширение дерева игры после того, как алгоритм определяет, что дальнейшее изучение данного поддерева бессмысленно.

NegMax

- Алгоритм NegMax заменяет чередование уровней MAX и MIN алгоритма Minimax единым подходом, используемым на каждом уровне дерева игры.
- Состояние игры всегда оценивается с точки зрения игрока, делающего начальный ход (что требует от функции оценки хранения этой информации).
- Алгоритм NegMax последовательно ищет ход, который дает максимум из значений дочерних узлов состояния с обратным знаком.



NegMax

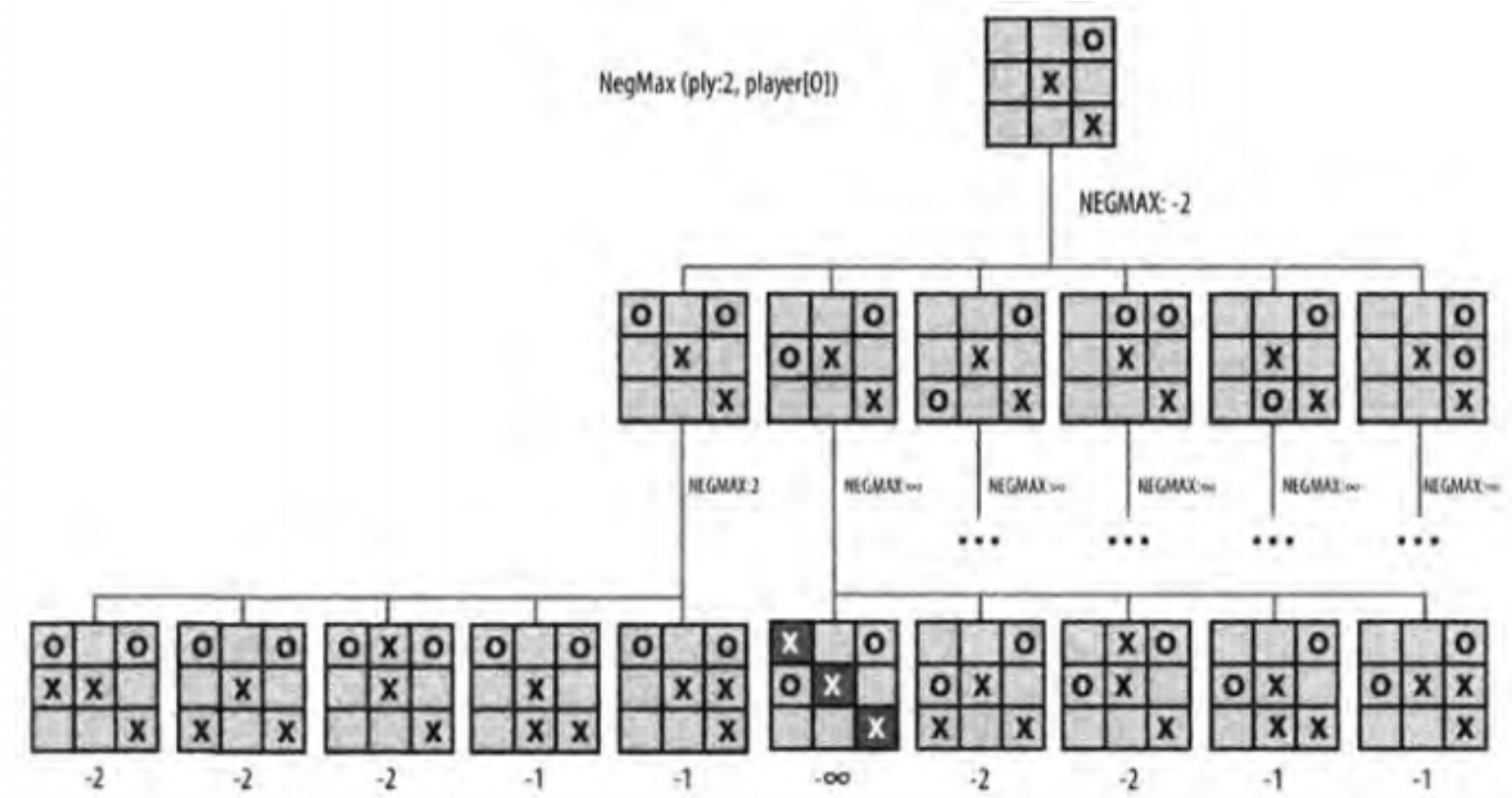
Алгоритм NegMax

Наилучший, средний и наихудший случаи: $O(b^{ply})$

```
bestmove (s, player, opponent)
    [move, score] = negmax (s, ply, player, opponent)
    return move
end

negmax (s, ply, player, opponent)
    best = [null, null]
    if ply == 0 или больше нет допустимых ходов then
        score = Вычисление s для игрока
        return [null, score]

    foreach Допустимый ход m для игрока в состоянии s do
        Выполнение хода m для s
        [move, score] = negmax (s, ply-1, opponent, player)      ❶
        Отмена хода m для s
        if -score > best.score then best = [m, -score]           ❷
    return best
end
```

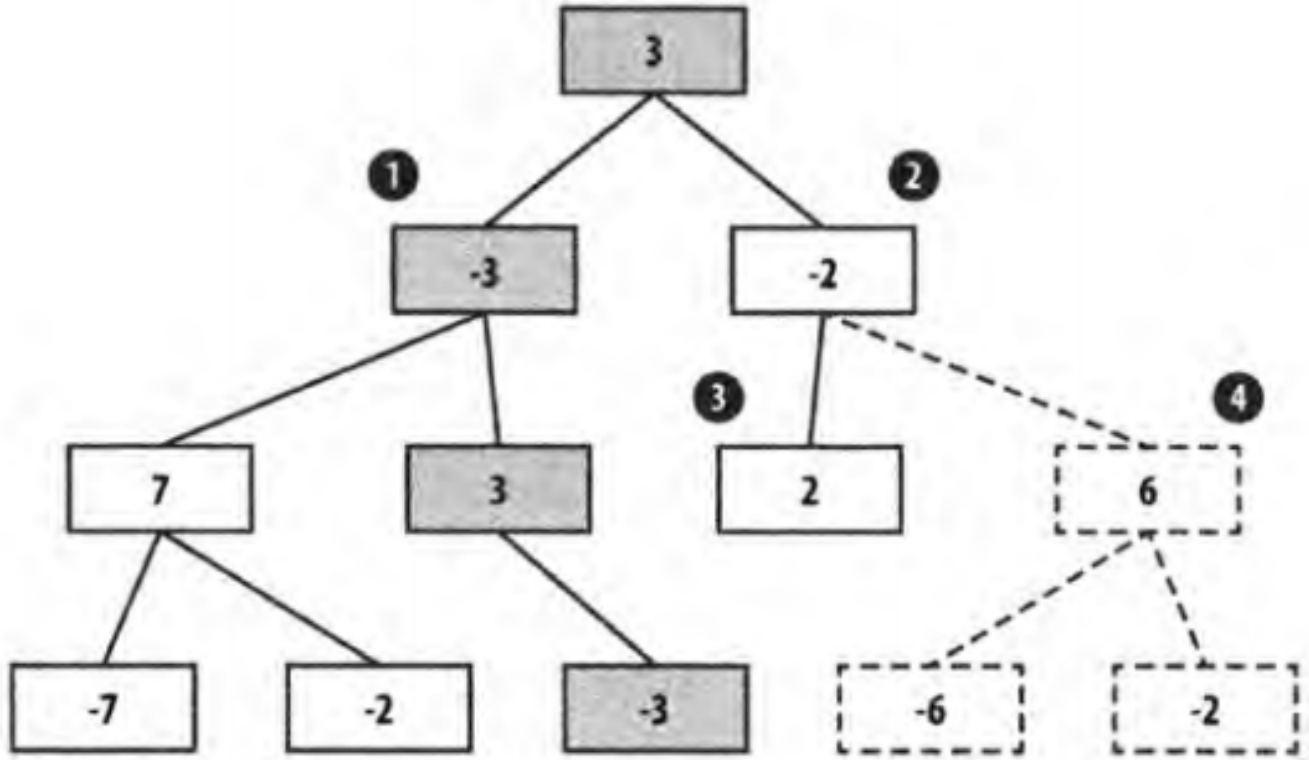
❶ Алгоритм NegMax обменивает игроков на каждом следующем уровне.
❷ Выбор наибольшей среди оценок дочерних узлов с обратным знаком.

NegMax



AlhaBeta

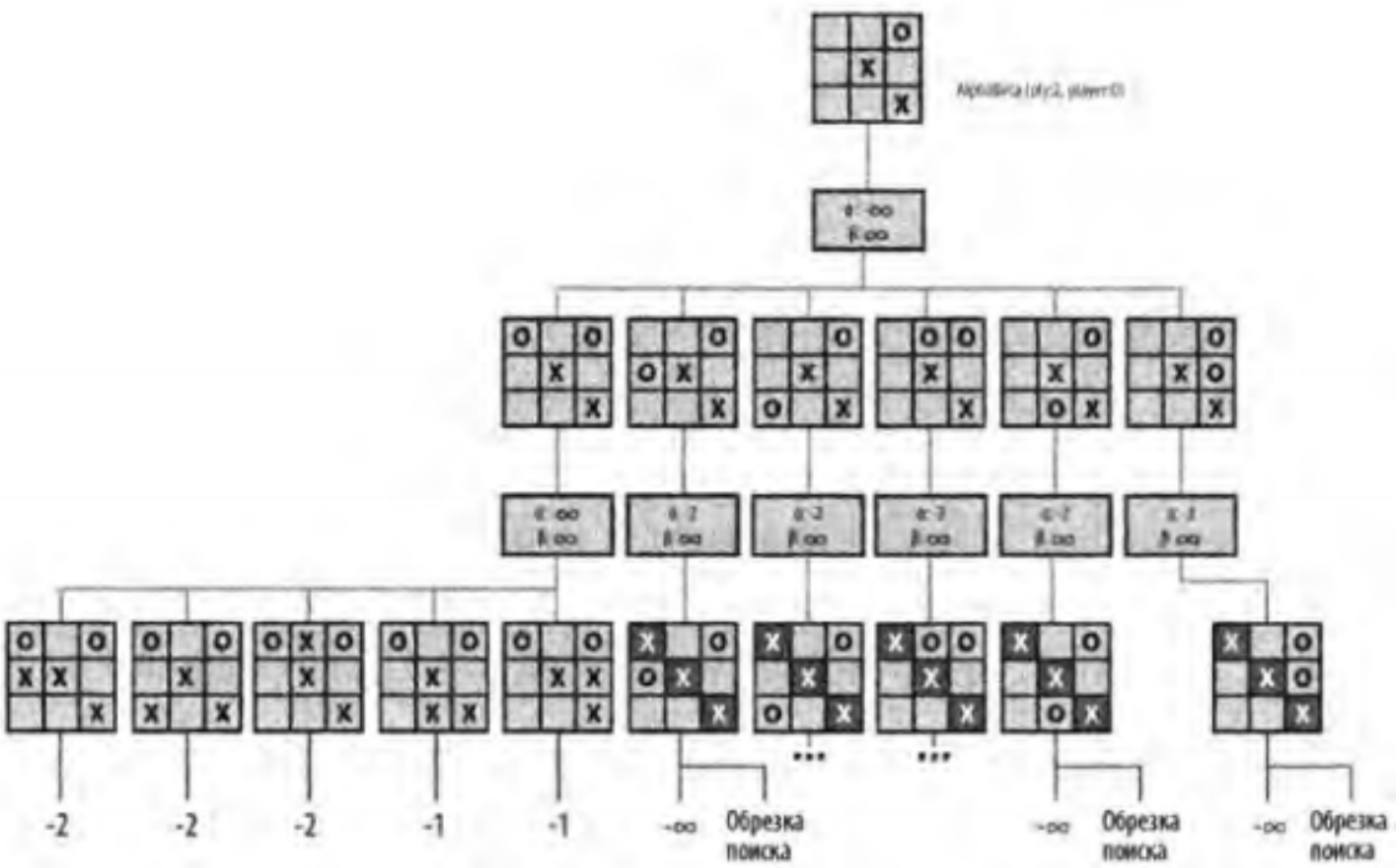
Алгоритм AlphaBeta определяет стратегию последовательного обрезания всех непродуктивных поисков в дереве



AlhaBeta

- После оценки поддерева игры с корнем в (1) алгоритм AlphaBeta знает, что если этот ход сделан, то противник не может сделать позицию хуже, чем -3.
- Это означает, что лучшее, что может сделать игрок, — добиться состояния с оценкой 3.
- Когда AlphaBeta переходит в состояние игры (2), то его первый дочерний узел — состояние (3) — имеет оценку, равную 2.
- Это означает, что если выбрать ход, приводящий в (2), то противник может заставить игрока перейти в состояние игры с оценкой, меньшей, чем лучшая найденная к этому моменту (т.е. 3).
- Таким образом, не имеет смысла проверять поддерево с корнем в (4), и оно полностью отбрасывается.

Предпросмотр AlhaBeta глубиной 2



Предпросмотр AlhaBeta глубиной 2

- Когда алгоритм AlphaBeta ищет лучший ход, он помнит, что X может достичь оценки не выше 2, если O делает ход в левый верхний угол.
- Для каждого другого хода 0 AlphaBeta определяет, что X имеет хотя бы один ход, который превосходит первый ход 0 (на самом деле для всех прочих ходов 0 игрок X может победить).
- Таким образом, дерево игры расширяется только до 16 узлов, что представляет собой экономию более 50% по сравнению с алгоритмом Minimax.

Алгоритм AlhaBeta

Алгоритм AlphaBeta

Наилучший и средний случаи: $O(b^{ply/2})$; наихудший случай: $O(b^{ply})$

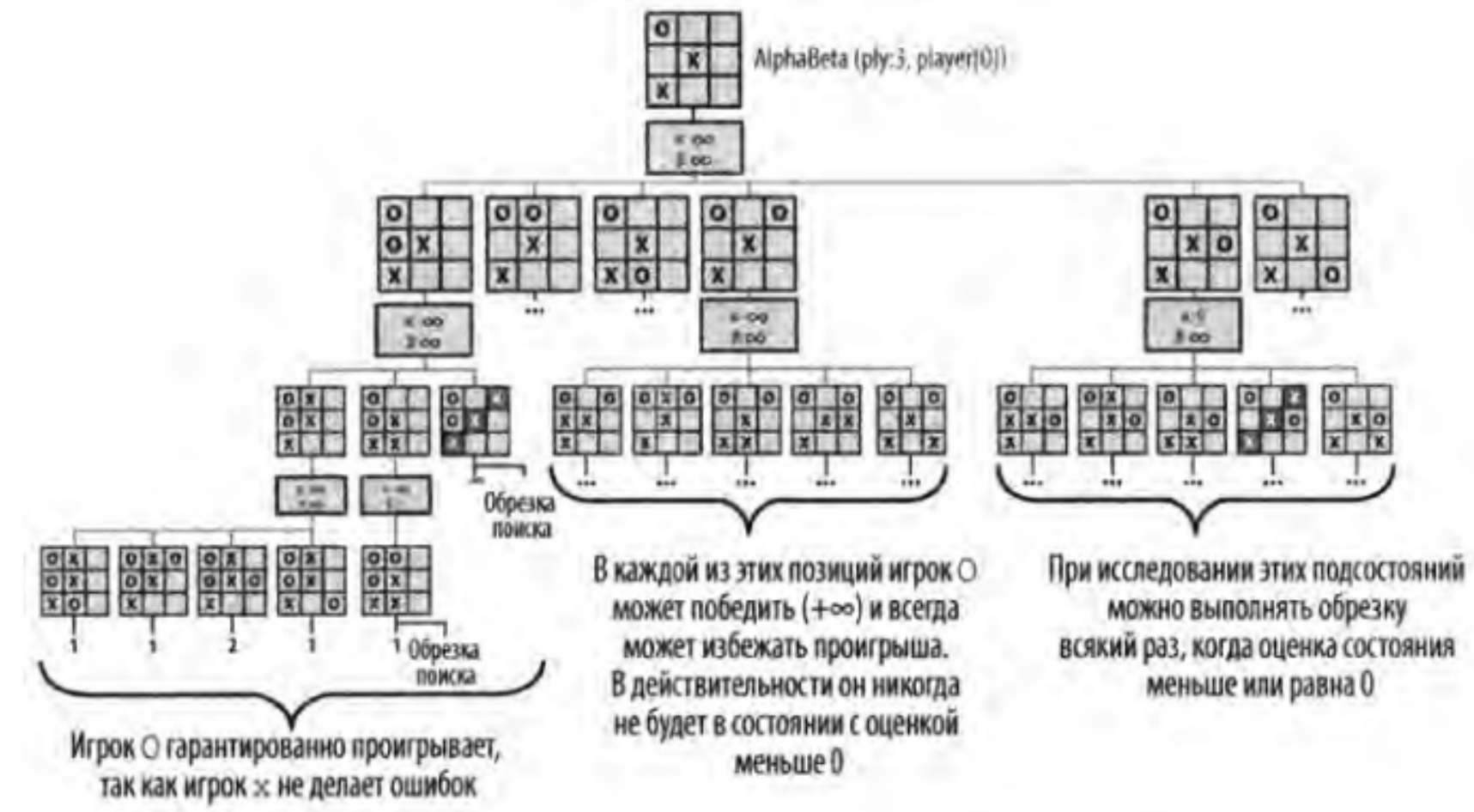
```
bestmove (s, player, opponent)
    [move, score] = alphaBeta (s, ply, player, opponent, -∞, ∞)    ❶
    return move
end

alphaBeta (s, ply, player, opponent, low, high)
    best = [null, null]
    if ply == 0 или нет допустимых ходов then                      ❷
        score = вычисление s для игрока player
        return [null, score]

    foreach Корректный ход m для игрока player в состоянии s do
        execute move m on s
        [move,score] = alphaBeta(s,ply-1,opponent,player,-high,-low)
        Отмена хода m для s

        if -score > best.score then
            low = -score
            best = [m, -low]
        if low ≥ high then return best                             ❸
    return best
end
```

❶ В начале худшее, что может сделать игрок, – проиграть (low = -∞).
Лучшее, что может сделать игрок, – выиграть (high = +∞).
❷ AlphaBeta оценивает листья, как и алгоритм NegMax.
❸ Прекращение исследования "братьев", когда наихудшая возможная

Алгоритм AlhaBeta

- Алгоритм рекурсивно выполняет поиск в дереве игры и поддерживает два значения, α и β, которые определяют «окно возможностей» для игрока до тех пор, пока $\alpha < beta$.
- Значение α представляет нижнюю границу состояний игры, найденную для игрока до настоящего времени (или $-\infty$, если ничего не было найдено) и объявляет, что игрок нашел ход, гарантирующий, что он может достичь как минимум указанного значения. Более высокие значения α означают, что игрок поступает правильно; при $\alpha = +\infty$ игрок выиграл, и поиск можно завершать.
- Значение β представляет верхнюю границу состояний игры, найденную для игрока до настоящего времени (или $+\infty$, если ничего не было найдено) и объявляет максимальное значение, которого игрок может достичь. Если значение β уменьшается все сильнее и сильнее, значит, противник поступает верно и делает наилучшие ходы, ограничивающие доступные для игрока

Алгоритм AlhaBeta



Анализ алгоритмов

Пусть каждый раз игроки могут сделать b ходов, а алгоритм просматривает на d ходов вперед.

- число вершин MinMax, NegMax - b^d;
- AlphaBeta оценивает b состояний игры для начинающего игрока на каждом уровне, но только одно состояние игры — для его противника.
- число вершин для AlphaBeta - $b^(d/2)$.

Алгоритм AlphaBeta может также исследовать то же общее количество состояний игры, что и алгоритм Minimax, расширяя глубину дерева игры до 2d.

Глубина	Состояния Minimax	Состояния AlphaBeta	Совокупное уменьшение
6	549 864	112 086	80%
7	549 936	47 508	91%
8	549 945	27 565	95%

Глубина предпросмотра 9^-k, которая гарантирует, что просмотрены все возможные ходы