Галкина Екатерина, Шрамов Георгий

Lecture 6: Search: Games, Minimax, and Alpha-Beta

Введение

Примерно в 1963 году известный в МТИ философ по имени Хьюберт Дрейфус написал статью, в которой был заголовок «Компьютеры не умеют играть в шахматы». Его после этого пригласили в лабораторию искусственного интеллекта, чтобы сыграть в шахматы с машиной Гринблатта. И, конечно, он проиграл.

После этого Саймур Павитт написал опровержение известной статьи Дрейфуса, в котором был заголовок «Дрейфус тоже не умеет играть в шахматы». Как ни странно, Дрейфус мог бы быть прав и был бы прав, если бы написал, что компьютеры не умеют играть в шахматы таким же образом, как и люди.

Как бы то ни было, примерно в 1968 году шахматист по имени Дэвид Леви поспорил с выдающимся основателем искусственного интеллекта Джоном Маккарти, что ни один компьютер не сможет обыграть чемпиона мира в течение следующих 10 лет. И спустя 5 лет Маккарти сдался, потому что уже стало понятно, что ни один компьютер не выиграет так, как хочет Маккарти, то есть играя в шахматы тем же образом, как это делает человек. Но спустя 30 лет, в 1997 году, Deep Blue обыграл чемпиона мира, и интерес к шахматам внезапно упал.

Но мы сегодня будем говорить об играх, поскольку в ходе игры возникают элементы, которые моделируют некоторый вид интеллекта. И если мы хотим понять что такое интеллект в общем случае, мы хотим понять и этот вид интеллекта в частности.

Возможные подходы к созданию искусственного интеллекта для игр

Рассмотрим несколько возможных подходов к созданию искусственного интеллекта, способного играть в игры, например в шахматы.

Первый подход состоит в том, чтобы компьютер описывал игровую доску так же, как и человек, с учётом пешечной структуры, безопасности короля, правильного момента для рокировки и т. п. Всё это анализируется с учётом тактики и стратегии, и в итоге делается определённый ход. В случае настольной игры с игровым полем, каждый следующий шаг будет определяться подобным процессом. Проблема в том, что никто не знает, как это сделать. В этом смысле Дрейфус был прав. Никакие игровые программы на сегодняшний день не включают в себя такие процессы. И поскольку никто не знает, как такое создать, мы не будем обсуждать дальше этот подход.

Но мы можем рассмотреть другие способы, например, **использование правил с условием**. Как они работают? Вы смотрите на конфигурацию доски, представленную узлом в дереве игры, и если можно сделать ход пешкой перед ферзём, вы делаете этот ход. Этот алгоритм не даёт числовую оценку ситуации, а просто рассматривает текущую конфигурацию доски и возможные действия и выбирает ход на основании одного из правил. Таким способом трудно получить хорошего игрока в шахматы, но любопытно, что кому-то удавалось написать хорошую программу для игры в шашки. Но в целом это не самый результативный метод.

Третий способ состоит в том, чтобы **оценить возможные последствия каждого хода**. Вы рассматриваете все возможные ситуации и выбираете лучшую для себя. Для этого необходимо оценить каждую игровую ситуацию. Рассмотрим самый распространённый способ такой оценки. У шахматной доски есть множество характеристик. Обозначим их $f_1, f_2, .., f_n$. Применим к этим характеристикам функцию и получим статическую оценку $s = g(f_1, .., f_n)$. Она называется статической, поскольку не исследуются все возможные случаи. Вы просто смотрите на доску, оценивая безопасность короля, пешечную структуру и т. д. Каждое из этих свойств характеризуется числом, которое является аргументом оценочной функции, которая также возвращает число. Это число и является оценкой конфигурации игровой доски с вашей точки зрения. Обычно в качестве функции g используется линейный полином, называемый линейной оценочной функцией: $g = c_1 f_1 + c_2 f_2 + ... + c_n f_n$, где c_i — константы.

Таким образом, мы можем использовать эту функцию, чтобы получить числа, характеризующие каждую из возможных игровых ситуаций, и выбрать наибольшее из этих чисел. На самом деле, нам не нужна сама оценочная функция, потому что в итоге нам не нужно знать оценку каждого хода, а нужно выбрать лучший из них, но это можно сделать в том числе и с помощью оценочной функции.

Вспомним алгоритмы поиска о которых мы говорили, в частности тот, с которым сравниваются все остальные: **полный перебор**. Этот алгоритм не требует вообще никакого интеллекта. Мы можем оценить ходы во всём дереве игры. Прежде чем понять, насколько хорош или плох такой подход, введём несколько основных понятий.

Рассмотрим дерево игры. На каждом уровне существует несколько вариантов выбора хода. Самих уровней тоже определённое количество. Стандартными обозначениями для этого являются коэффициент ветвления b и глубина дерева d (например, см. рис. 1). Эти числа однозначно определяют количество листьев. При условии, что коэффициент ветвления постоянный, количество терминальных узлов будет равно b^d.

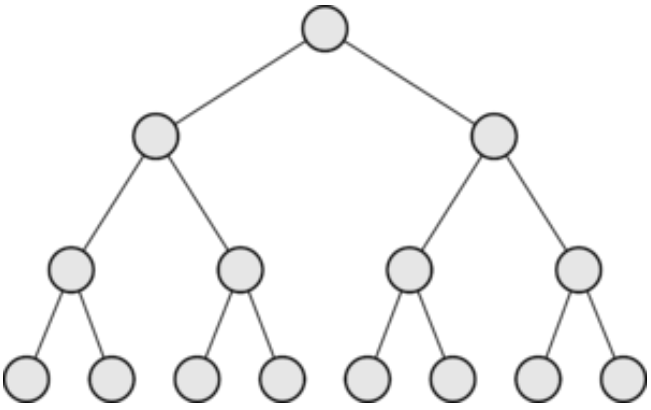

Рис. 1: $b = 2, d = 3, b^d = 8$

Попробуем оценить количество вычислений в алгоритме полного перебора для игры в шахматы. Каждый игрок в среднем делает 50 ходов, поэтому глубина дерева игры равна примерно 100. Коэффициент ветвления зависит от стадии игры и многого другого, но в среднем равен 14-15. Если бы он был равен 10, то число листьев было бы 10^{100}. Для удобства вычисления оценим число листьев как 10^{120}, поскольку на самом деле коэффициент ветвления больше 10. Именно столько вычислений статического параметра пришлось бы сделать, если использовать алгоритм полного перебора. Насколько это возможно сделать сегодня, с использованием облачных вычислений и т. п.?

Считается, что во вселенной порядка 10^{80} атомов. Количество секунд в году примерно равно $\pi 10^7$, а в каждой секунде 10^9 наносекунд. С начала истории вселенной прошло

приблизительно 10^{10} лет. В итоге получаем $10^{80+7+9+10} = 10^{106}$ наносекунд в истории вселенной.

Итак, если бы все атомы во вселенной вычисляли статические оценки каждую наносекунду с момента Большого взрыва, они бы до сих пор отставали на 14 порядков. Очевидно, что переборный алгоритм в данном случае бесполезен.

Далее мы будем рассматривать ещё один подход. Он состоит в том, чтобы **оценивать не один уровень, а попытаться заглянуть настолько далеко, насколько это возможно**. Эта идея была предложена несколько раз, в том числе Клодом Шенноном и Аланом Тьюрингом.

Алгоритм минимакс

Рассмотрим простое дерево игры с коэффициентом ветвления и глубиной равными двум (рис. 2). Числа в листьях дерева означают оценку игрового поля с точки зрения игрока, который делает первый ход. Будем считать, что этот игрок хочет добиться результата с максимальной оценкой (в нашем случае 8). Будем называть этого игрока **максимизирующий игрок**. Но у него есть соперник, **минимизирующий игрок**, который хочет свести игру к результату с минимальной оценкой. Поэтому предложенный метод получил название **алгоритм минимакс**.

Сложно понять, по какой ветви пойдёт игра в этом случае, но если мы окажемся на среднем уровне в роли минимизирующего игрока, то мы можем сказать, что он выберет ход с минимальным значением 2, и никогда не пойдёт в сторону 7. Аналогично для соседней ветви: минимизирующий игрок выберет ход с оценкой 1, а не 8. Итак, мы взяли значения с последнего уровня и получили оценки ходов на среднем уровне. Мы можем продолжить делать то же самое.

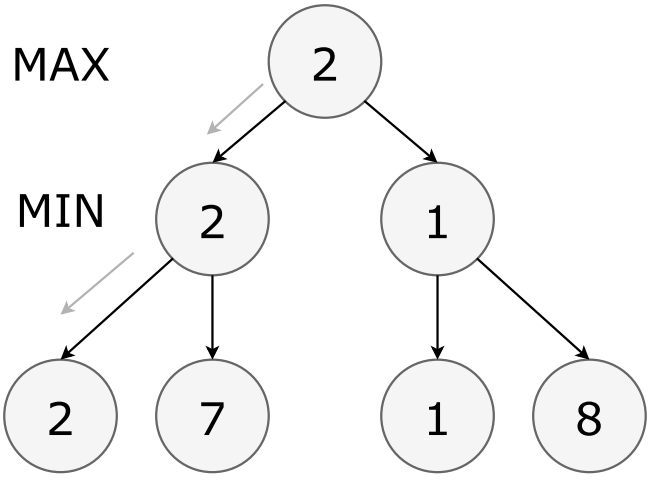

Рис. 2: Пример работы алгоритма минимакс

Теперь очередь максимизирующего игрока делать ход. Если он пойдёт по левой ветви, то оценка хода равна 2, по правой ветви — 1. Таким образом, игрок пойдёт по левой ветви, и в итоге игра закончится с конфигурацией, оценённой значением 2. Это не максимальное и не минимальное значение, но игроки соревнуются друг с другом, играя в состязательную игру, поэтому и не рассчитывают получить эти значения.

В общем случае алгоритм минимакс работает так: вычисляются статические оценки, сравниваются друг с другом и сохраняются на следующем уровне, пока не будет получено значение в корне дерева.

Если запустить программу для дерева игры глубиной 4 и коэффициентом ветвления 2, то будет подсчитано 16 статических значений. Подсчёт происходит достаточно быстро, но в

реальных играх глубина дерева достигает 10, а коэффициент ветвления — 14-15 и больше. Мы знаем, что количество статических параметров растёт экспоненциально в зависимости от глубины дерева. Как показывает опыт, 7-8 уровней недостаточно, чтобы создать сильного игрока. Но если спуститься до 15-16 уровня, то такой искусственный интеллект способен обыграть чемпиона мира. Поэтому важно уметь доходить до максимально возможной глубины дерева игры, вычисляя как можно меньше статических оценок.

Альфа-бета отсечение

При рассмотрении метода ветвей и границ мы старались оптимизировать его таким образом, чтобы не просматривать некоторые поддеревья в дереве поиска. Мы можем поступать подобным образом и с деревом игры, не вычисляя лишние статические оценки.

Рассмотрим то же самое дерево игры, в котором первый ход делает максимизирующий игрок, только в этот раз будем вычислять статические оценки по очереди (рис. 3). Первая из них равна двум, поэтому минимизирующий игрок, находясь на среднем уровне дерева, знает, что итоговое значение оценки в узле не может быть больше 2. Следующее значение равно 7, это больше 2, поэтому теперь минимизирующий игрок может быть уверен, что значение в левой ветви на среднем уровне в точности равно 2. Как только получено точное значение, максимизирующий игрок в корне дерева может считать, что оценка итогового хода будет больше либо равна 2.

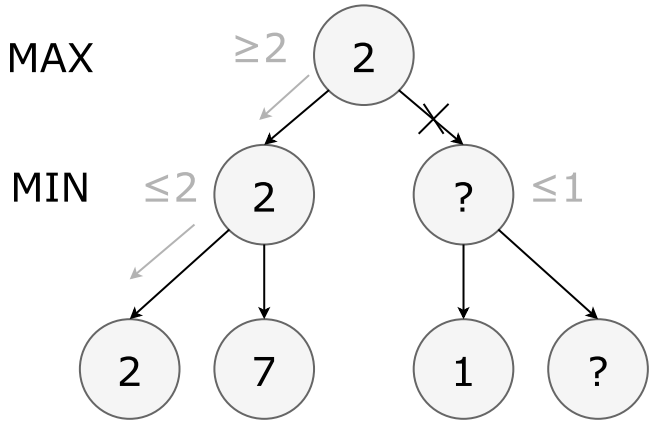

Рис. 3: Пример работы алгоритма минимакс с альфа-бета отсечением

Вычислим оценку в следующем листе. Она равна 1, поэтому в правой ветви минимизирующий игрок получит значение не больше, чем 1. Нужно ли производить вычисления дальше? С точки зрения максимизирующего игрока становится видно, что если он пойдёт по левой ветви, он может получить итоговую оценку, равную 2. Если же он пойдёт по правой, то она будет не больше 1. Таким образом, для принятия решения не нужно вычислять значение оценки в последнем листе. Совершенно не важно, чему на самом деле равно это значение, 8, 1000, бесконечность или минус бесконечность.

Итак, **альфа-бета алгоритм является «надстройкой» над алгоритмом минимакс и позволяет отсекать ветви в дереве игры**. Важно понять, что это не альтернатива алгоритму минимакс, а его оптимизация.

Рассмотрим более сложный пример, чтобы увидеть те особенности альфа-бета отсечения, которые проявляются на деревьях глубины 4 и больше. Дерево игры приведено на рисунке 4. Красным цветом выделены те листья, в которых происходило вычисление статической оценки. В левой части дерева алгоритм работает аналогично предыдущему примеру, при этом 3 из 8 значений не были вычислены из-за отсечения. В результате получена оценка для корневой вершины: значение в ней не может быть меньше 7.

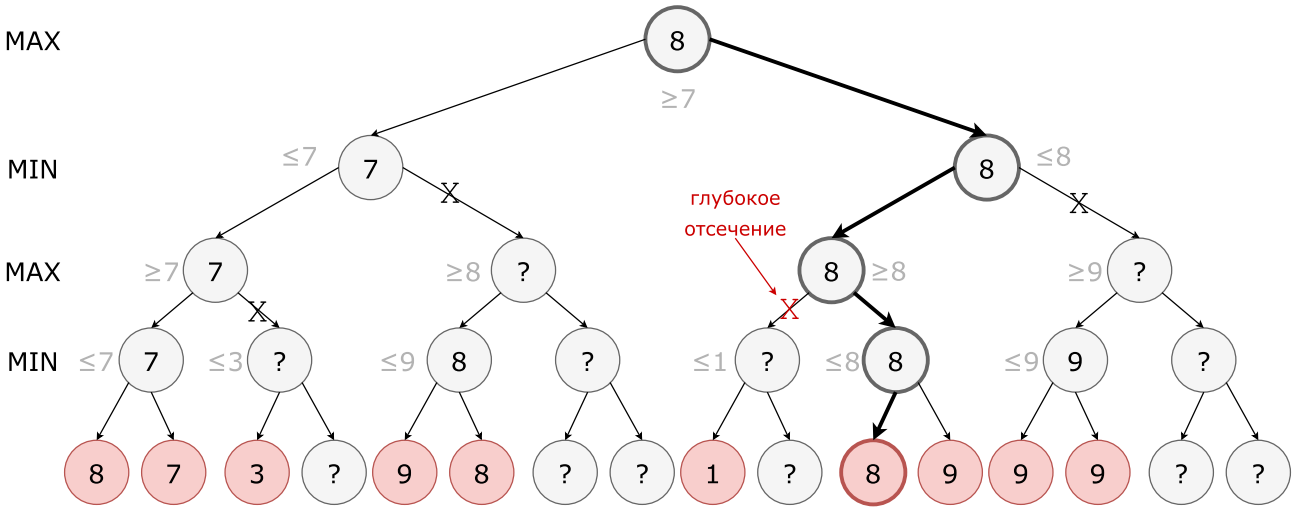

Рис. 4: Пример работы алгоритма минимакс с альфа-бета отсечением на дереве глубины 4

В правой части поддерева первой будет вычислена оценка, равная 1. Поэтому минимизирующий игрок может гарантированно получить значение, не превышающее 1, если игра пойдёт по этой ветви. Но в корневой вершине уже есть оценка, т. е. максимизирующий игрок знает, что может гарантированно сделать ход с оценкой, равной 7. До этого момента в ходе работы алгоритма сравнивались значения оценок только на соседних уровнях дерева. В данной ситуации мы сравниваем оценки, разделённые несколькими уровнями, но всё равно можем сделать вывод, что дальнейшее рассмотрение узла бессмысленно. В такой ситуации говорят, что происходит **глубокое отсечение**. Дальнейших ход работы алгоритма и его результат также представлены на рисунке 4.

Постепенное углубление

Ранее мы предполагали, что коэффициент ветвления не изменяется. Но на самом деле он может изменяться в ходе игры в зависимости от состояния игрового поля. Вы можете посчитать, сколько вычислений придётся сделать за 2 минуты или сколько времени тратится в среднем на один ход. Но вы не можете точно сказать, насколько глубоко вы можете пройти по дереву игры, потому что это зависит от хода самой игры.

Когда программы для игр только начали появляться, они пренебрегали многими вычислениями и вычисляли ход за пару секунд. Но они могли бы пойти совершенно по-другому, если бы использовали все вычислительные возможности. С другой стороны, они могли бы усердно вычислять ходы, но через несколько минут у них всё ещё не было бы ответа, поэтому они делали случайный ход. Это не очень хорошая ситуация, но это происходило, потому что программа не знала, насколько глубоко может зайти в дереве игры.

Рассмотрим дерево игры глубины d и коэффициентом ветвления b. На нижнем уровне b^d листьев. Если мы пока забудем про альфа-бета отсечение, то нам придётся вычислять все b^d статические оценки. На уровне выше их будет b^{d-1}, то есть в b раз меньше. Если нам не получится продвинуться в вычислениях дальше предпоследнего уровня, нам нужна гарантия, что в результате всё равно получится верный ход. Если коэффициент ветвления равен 10, то можно вычислить оценки ходов на предпоследнем уровне, и это составляет 10% от вычислений, которые необходимо было бы произвести на нижнем уровне. А если окажется, что коэффициент ветвления настолько велик, что мы не сможем вычислить значения и на предпоследнем уровне? Тогда у нас должен быть готов ответ ещё на уровень раньше.

Таким образом, нашей целью является наличие хода в любой момент времени. Для этого необходимо производить вычисления в вершинах на каждом уровне. Всего получим $S = 1 + b + ... + b^d$ оценок, которые нужно вычислить. Насколько большое это число? Его можно представить в виде $\frac{b^d-1}{b-1}$, что приближенно равно b^{d-1}. С учётом этого приближения можно сказать, что количество вычислений, необходимых для гарантированного наличия ответа на любом уровне, примерно равно количеству вычислений на предпоследнем уровне. Эта идея получила название «постепенное углубление».

Заключение

Повторим основные идеи и принципы, которые мы обсудили ранее:

1. Альфа-бета отсечение позволяет не вычислять значительное количество статических оценок и даже не генерировать лишние ходы. Они всё равно не привели бы к ответу. Это является не альтернативой алгоритму минимакс, а его оптимизацией, дающей от же ответ.

2. Последовательное углубление позволяет получить ответ, на каком бы уровне дерева игры не происходили вычисления.

3. Эта идея может быть рассмотрена как простой частный случай **алгоритма с отсечением по времени**. Такой алгоритм может дать ответ как только он понадобится, в любой момент времени.

Итак, программы, играющие в игры, работают на основе алгоритма минимакс с альфа-бета отсечением и постепенным углублением. Следующим шагом к увеличению производительности является использование альфа-бета отсечения при вычислении промежуточных частей дерева игры при постепенном углублении.

То, как работал Deep Blue, не сильно отличалось от рассмотренных нами алгоритмов. В 1997 году он вычислял примерно 200 миллионов статических оценок в секунду и опускался по дереву игры на 14-16 уровней. В нём применялись алгоритм минимакс с альфа-бета отсечением и последовательное углубление, отдельно учитывались ситуации начала и конца игры, использовались параллельные вычисления, а также неравномерное дерево игры. Это очень важное отличие, поскольку мы предполагали без каких-либо оснований, что дерево игры всегда доходит до одного и того же уровня во всех ветвях. Но если не делать это предположение, то можно построить дерево игры, которое даёт более точное значение оценки для наилучшего хода.

Теперь мы знаем, как компьютеры играют в игры, но вопрос о том, происходит ли то же самое в голове у человека, пока можно считать открытым.