Поиск на игровых деревьях

Деревья игры. Поиск выигрышной стратегии

Будем рассматривать класс ***игр двух лиц с полной информацией***. В таких играх участвуют два игрока, которые поочередно делают свои ходы. В любой момент игры каждому игроку известно все, что произошло в игре к этому моменту и что может быть сделано в настоящий момент. Игра заканчивается либо выигрышем одного игрока (и проигрышем другого), либо ничьей.

Таким образом, в рассматриваемый класс не попадают игры, исход которых зависит хотя бы частично от случая – большинство карточных игр, игральные кости, «морской бой» и проч. Тем не менее класс достаточно широк: в него входят такие игры, как шахматы, шашки, реверси, калах, крестики-нолики и другие игры.

Для формализации и изучения игровых стратегий в классе игр с полной информацией может быть использован подход, основанный на редукции задач. При этом подходе должны быть определены следующие составляющие: форма описания задач и подзадач; операторы, сводящие задачи к подзадачам; элементарные задачи; а также задано описание исходной задачи.

Наиболее интересна задача поиска выигрышной стратегии для одного из игроков, отправляясь от некоторой конкретной конфигурации (позиции) игры (не обязательно начальной). При использовании подхода, основанного на редукции задач, выигрышная стратегия ищется в процессе доказательства того, что игра может быть выиграна. Аналогично, поиск ничейной стратегии, исходя из некоторой конкретной позиции, ведется в процессе доказательства того, что игра может быть сведена к ничьей.

Ясно, что описание решаемой задачи должно содержать описание конфигурации игры, для которой ищется нужная стратегия. Например, в шашках игровая позиция включает задание положений на доске всех шашек, в том числе дамок. Обычно описание конфигурации содержит также указание, кому принадлежит следующий ход.

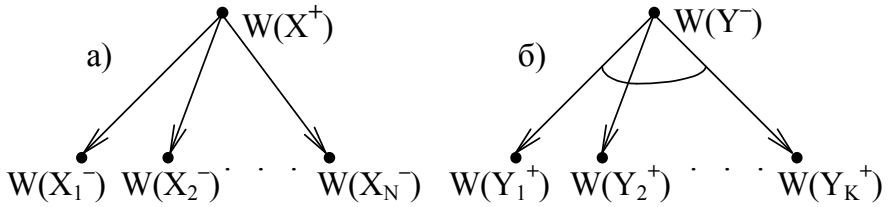
Пусть именами игроков будут ПЛЮС и МИНУС. Будем использовать следующие обозначения:
X^S или Y^S – некоторая конфигурация игры, причем индекс S принимает значения + или –, указывая тем самым, кому принадлежит следующий ход (т.е. в конфигурации X^+ следующий ход должен делать игрок ПЛЮС, а в X^- – игрок МИНУС);
$W(X^S)$ – задача доказательства того, что игрок ПЛЮС может выиграть, исходя из позиции X^S;
$V(X^S)$ – задача доказательства того, что игрок МИНУС может выиграть, отправляясь от позиции X^S.

Рассмотрим сначала игровую задачу $W(X^S)$. Операторы сведения (редукции) этой задачи к подзадачам определяются исходя из ходов, допустимых в проводимой игре:

* Если в некоторой конфигурации X^+ очередной ход делает игрок ПЛЮС, и имеется N допустимых ходов, приводящих соответственно к конфигурациям X_1^-, X_2^-, . . . X_N^- , то для решения задачи $W(X^+)$ необходимо решить по крайней мере одну из подзадач $W(X_i^-)$. Действительно, так как ход выбирает игрок ПЛЮС, то он выиграет игру, если хотя бы один из ходов ведет к выигрышу – см. рис. 1(а).

* Если же в некоторой конфигурации Y^- ход должен сделать игрок МИНУС, и имеется K допустимых ходов, приводящих к конфигурациям Y_1^+ , Y_2^+ , . . . Y_K^+, то для решения задачи $W(Y^-)$ требуется решить каждую из возникающих подзадач $W(Y_i^+)$. Действительно, поскольку ход выбирает МИНУС, то ПЛЮС выиграет игру, если выигрыш гарантирован ему после любого хода противника – см. рис. 1(б).

Рис.1



Последовательное применение данной схемы редукции к исходной конфигурации игры порождает И/ИЛИ-дерево (И/ИЛИ-граф), которое называют ***деревом*** (***графом***) ***игры***. Дуги игрового дерева соответствуют ходам игроков, вершины – конфигурациям игры, причем листья дерева – это позиции, в которых игра завершается выигрышем, проигрышем или ничьей. Часть листьев являются заключительными вершинами, соответствующими элементарным задачам – позициям, выигрышным для игрока ПЛЮС. Заметим, что для конфигураций, где ход принадлежит ПЛЮСу, в игровом дереве получается ИЛИ-вершина, а для позиций, в которых ходит МИНУС, получается И-вершина.

Цель построения игрового дерева или графа – получение решающего поддерева для задачи $W(X^S)$, показывающего, как игрок ПЛЮС может выиграть игру из позиции X^S независимо от ответов противника. Для этого могут быть применены разные алгоритмы поиска на И/ИЛИ-графах. Решающее дерево заканчивается на позициях, выигрышных для ПЛЮСа, и содержит полную стратегию достижения им выигрыша: для каждого возможного продолжения игры, выбранного противником, в дереве есть ответный ход, приводящий к победе.

Для задачи $V(X^S)$ схема сведения игровых задач к подзадачам аналогична: ходам игрока ПЛЮС будут соответствовать И-вершины, а ходам МИНУСа – ИЛИ-вершины, заключительные же вершины будут соответствовать позициям, выигрышным для игрока МИНУС.

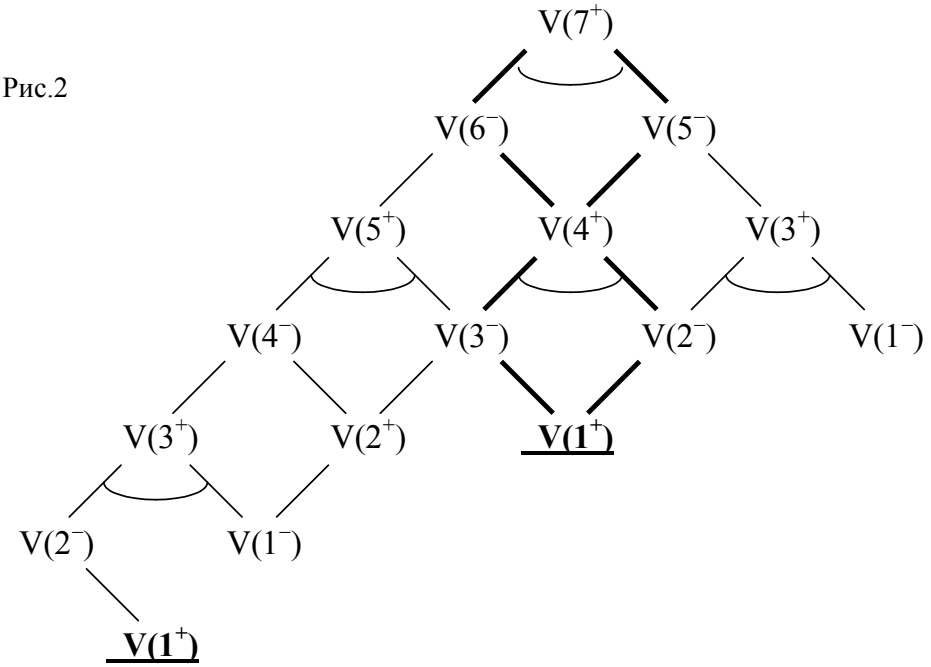
Конечно, подобная редукция задач применима и в случае, когда нужно доказать существование ничейной стратегии в игре. При этом определение элементарной задачи должно быть соответствующим образом изменено.

Рассмотрим в качестве иллюстрации простую игру, называемую «последний проигрывает». Два игрока поочередно берут по одной или две монеты из кучки, первоначально содержащей семь монет. Игрок, забирающий последнюю монету, проигрывает.

На рис.2 показан полный граф игры для задачи $V(7^+)$, жирными дугами на нем выделен решающий И/ИЛИ-граф, который доказывает, что второй игрок (т.е. игрок МИНУС, начинающий вторым), всегда может выиграть. Конфигурация игры описана как число оставшихся в текущий момент монет, также указание, кому принадлежит следующий ход. Заключительные вершины, соответствующие элементарной задаче $V(1^+)$, т.е. выигрышу игрока МИНУС, в графе подчеркнуты.

Представленная в решающем графе выигрышная стратегия может быть сформулирована словесно так: если в очередном ходе игрок ПЛЮС берет одну монету, то в следующем ходе МИНУС должен взять две, а если ПЛЮС берет две монеты, то МИНУС должен забрать одну. Отметим, что для аналогичной задачи $W(7^+)$ решающий граф построить не удастся (начальная вершина неразрешима); таким образом, у игрока ПЛЮС нет выигрышной стратегии в этой игре.

Рис.2



В большинстве игр, представляющих интерес, таких как шашки и шахматы, построить полные решающие деревья и найти полные игровые стратегии не представляется возможным. Например, для шашек число вершин в полном игровом дереве оценивается величиной порядка 10^{40}, и просмотреть такое дерево практически нереально. Алгоритмы же упорядоченного перебора с применением эвристик не настолько уменьшают просматриваемую часть дерева игры, чтобы дать существенное, на несколько порядков, сокращение времени поиска.

Тем не менее для неполных игр в шашки и шахматы (например, для эндшпилей), так же как и для всех несложных игр, таких как «крестики-нолики» на фиксированном квадрате небольшого размера, можно успешно применять алгоритмы поиска на И/ИЛИ-графах, позволяющие обнаруживать выигрышные и ничейные игровые стратегии.

Рассмотрим, к примеру, игру «крестики-нолики» на квадрате 3×3. Игрок ПЛЮС ходит первым и ставит крестики, а МИНУС – нолики. Игра заканчивается, когда составлена либо строка, либо столбец, либо диагональ из крестиков (в этом случае выигрывает ПЛЮС) или ноликов (выигрывает МИНУС).

Оценим размер полного дерева игры: начальная вершина имеет 9 дочерних вершин, каждая из которых в свою очередь – 8 дочерних; каждая вершина глубины 2 имеет 7 дочерних и т.д. Таким образом, число концевых вершин в дереве игры равно 9!=362880, но многие пути в этом дереве обрываются раньше на заключительных вершинах. Значит, в этой игре возможен полный просмотр дерева и нахождение выигрышной стратегии. Однако ситуация изменится при существенном увеличении размеров квадрата или же в случае неограниченного поля игры.

В таких случаях, как и во всех сложных играх вместо нереальной задачи поиска полной игровой стратегии решается, как правило, более простая задача – поиск для заданной позиции игры достаточно хорошего первого хода.

Минимаксная процедура

С целью поиска достаточно хорошего первого хода просматривается обычно часть игрового дерева, построенного от заданной конфигурации. Для этого применяется один из переборных алгоритмов (в глубину, в ширину или эвристический) и некоторое искусственное окончание перебора вершин в игровом дереве: например, ограничивается время перебора или же глубина поиска.

В построенном таким образом частичном дереве игры оцениваются вершины, и по полученным оценкам определяется наилучший ход от заданной игровой конфигурации. При этом для оценивания концевых вершин (листьев) полученного дерева используется так называемая ***статическая оценочная функция***, а для оценивания остальных вершин – корневой (начальной) и промежуточных между корневой и концевыми вершинами – используется так называемый ***минимаксный принцип***.

Статическая оценочная функция, будучи применена к некоторой вершине-конфигурации игры, дает числовое значение, оценивающее различные достоинства этой игровой позиции. Например, для шашек могут учитываться такие (статические) элементы конфигурации игры, как продвинутость и подвижность шашек, количество дамок, контроль ими центра и проч. По сути, статическая функция вычисляет эвристическую оценку шансов на выигрыш одного из игроков. Для определенности будем рассматривать задачу выигрыша игрока ПЛЮС и соответственно поиска достаточно хорошего его первого хода от заданной конфигурации.

Будем придерживаться общепринятого соглашения, по которому значение статической оценочной функции тем больше, чем больше преимуществ имеет игрок ПЛЮС (над игроком МИНУС) в оцениваемой позиции. Очень часто оценочная функция выбирается следующим образом:

- статическая оценочная функция положительна в позициях, где игрок ПЛЮС имеет преимущества;
- статическая оценочная функция отрицательна в конфигурациях, где МИНУС имеет преимущества;
- статическая оценочная функция близка к нулю в позициях, не дающих преимущества ни одному из игроков.

Например, для шашек в качестве простейшей статической функции может быть взят перевес в количестве шашек (и дамок) у игрока ПЛЮС. Для игры «крестики-нолики» на фиксированном квадрате возможна такая статическая оценочная функция:

$$E(P) = \begin{cases} +\infty & \text{если P есть позиция выигрыша игрока ПЛЮС} \\ -\infty & \text{если P есть позиция выигрыша МИНУСа} \\ (N_L^+ + N_C^+ + N_D^+) - (N_L^- + N_C^- + N_D^-) & \text{в остальных случаях} \end{cases}$$

где $+\infty$ – очень большое положительное число;

$-\infty$ – очень маленькое отрицательное число;

N_L^+, N_C^+, N_D^+ – соответственно число строк, столбцов и диагоналей, «открытых» для игрока ПЛЮС (т.е. где он еще может поставить три выигрышных крестика подряд),

а N_L^-, N_C^-, N_D^- – аналогичные числа для игрока МИНУС.

На рис.3 приведены две игровые позиции (на квадрате 4×4) и соответствующие значения статической оценочной функции.

Рис.3

а)

	O		
	X	X	

б)

		O	
		X	
		X	
	O	X	

E (P) = 8 - 5 = 3 E (P) = 6 - 4 = 2

Подчеркнем, что с помощью статической оценочной функции оцениваются только концевые вершины дерева игры, для оценок же промежуточных вершин, как и начальной вершины, используется минимаксный принцип, основанный на следующей простой идее. Если бы игроку ПЛЮС пришлось бы выбирать один из нескольких возможных ходов, то он выбрал бы наиболее сильный ход, т.е. ход, приводящий к позиции с наибольшей оценкой. Аналогично, если бы игроку МИНУС пришлось бы выбирать ход, то он выбрал бы ход, приводящий к позиции с наименьшей оценкой.

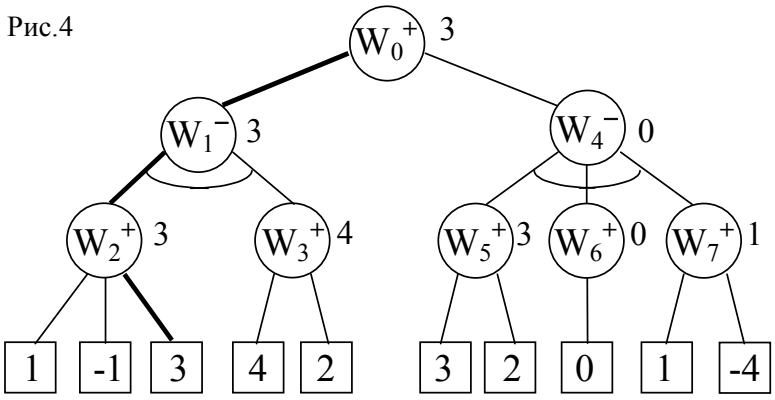
Сформулируем теперь сам минимаксный принцип:

- ИЛИ-вершине дерева игры приписывается оценка, равная максимуму оценок ее дочерних вершин;
- И-вершине игрового дерева приписывается оценка, равная минимуму оценок ее дочерних вершин.

Минимаксный принцип положен в основу ***минимаксной процедуры***, предназначенной для определения наилучшего (более точно, достаточно хорошего) хода игрока исходя из заданной конфигурации игры S при фиксированной глубине поиска N в игровом дереве. Предполагается, что игрок ПЛЮС ходит первым (т.е. начальная вершина есть ИЛИ-вершина). Основные этапы этой процедуры таковы:

1. Дерево игры строится (просматривается) одним из известных алгоритмов перебора (как правило, алгоритмом поиска вглубь) от исходной позиции S до глубины N;
2. Все концевые вершины полученного дерева, т.е. вершины, находящиеся на глубине N, оцениваются с помощью статической оценочной функции;
3. В соответствии с минимаксным принципом вычисляются оценки всех остальных вершин: сначала вычисляются оценки вершин, родительских для концевых, затем родительских для этих родительских вершин и так далее; таким образом оценивание вершин происходит при движении снизу вверх по дереву поиска – до тех пор, пока не будут оценены вершины, дочерние для начальной вершины, т.е. для исходной конфигурации S;
4. Среди вершин, дочерних к начальной, выбирается вершина с наибольшей оценкой: ход, который к ней ведет, и есть искомый наилучший ход в игровой конфигурации S.

Рис.4



На рис.4 показано применение минимаксной процедуры для дерева игры, построенного до глубины N=3. Концевые вершины не имеют имен, они обозначены своими оценками – значениями статической оценочной функции. Числовые индексы имен остальных вершин показывают порядок, в котором эти вершины строились алгоритмом перебора вглубь. Рядом с этими вершинами находятся их минимаксные оценки, полученные при движении в обратном (по отношению к построению дерева) направлении. Таким образом, наилучший ход – первый из двух возможных.

На рассматриваемом игровом дереве выделена ветвь (последовательность ходов игроков), представляющая так называемую *минимаксно-оптимальную* игру, при которой каждый из игроков всегда выбирает наилучший для себя ход. Заметим, что оценки всех вершин этой ветви дерева совпадают, и оценка начальной вершины равна оценке концевой вершины этой ветви.

В принципе статическую оценочную функцию можно было бы применить и к промежуточным вершинам, и на основе этих оценок осуществить выбор наилучшего первого хода, например, сразу выбрать ход, максимизирующий значение статической оценочной функции среди вершин, дочерних к исходной. Однако считается, что оценки, полученные с помощью минимаксной процедуры, есть более надежные меры относительного достоинства промежуточных вершин, чем оценки, полученные прямым применением статической оценочной функции. Действительно, минимаксные оценки основаны на просмотре игры вперед и учитывают разные особенности, которые могут возникнуть в последующем, в то время как простое применение оценочной функции учитывает лишь свойства позиции как таковой. Это отличие статических и минимаксных оценок существенно для «активных», динамичных позиций игры (например, в шашках и шахматах к ним относятся конфигурации, в которых возникает угроза

взятия одной или нескольких фигур). В случае же так называемых «пассивных», спокойных позиций статическая оценка может мало отличаться от оценки по минимаксному принципу.

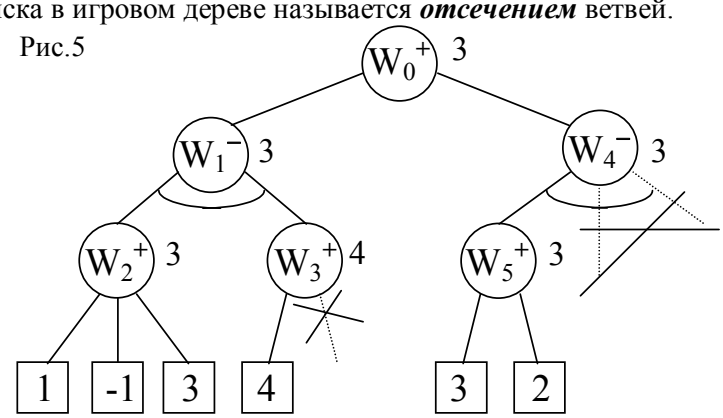
Альфа-бета процедура

Минимаксная процедура организована таким образом, что процесс построения частичного дерева игры отделен от процесса оценивания вершин. Такое разделение приводит к тому, что в целом минимаксная процедура – неэффективная стратегия поиска хорошего первого хода. Чтобы сделать процедуру более экономной, необходимо вычислять статические оценки концевых вершин и минимаксные оценки промежуточных вершин одновременно с построением игрового дерева. Этот путь приводит к так называемой ***альфа-бета процедуре*** поиска наилучшего первого хода от заданной позиции, при этом можно добиться существенного сокращения вычислительных затрат, прежде всего, времени вычисления статических оценок. В основе сокращения поиска лежит достаточно очевидное соображение: если есть два варианта хода одного игрока, то худший в ряде случаев можно сразу отбросить, не выясняя, насколько в точности он хуже.

Рассмотрим сначала идею работы альфа-бета процедуры на примере игрового дерева, приведенного на рис.19. Дерево игры строится до глубины N=3 алгоритмом перебора вглубь. Причем сразу, как это становится возможным, вычисляются не только статические оценки концевых вершин, но и *предварительные* минимаксные оценки промежуточных вершин. Предварительная оценка определяется соответственно как минимум или максимум уже известных к настоящему моменту оценок дочерних вершин. В общем случае, эта оценка может быть получена при наличии оценки хотя бы одной дочерней вершины. В ходе дальнейшего построения дерева игры и получения новых оценок вершин предварительные оценки постепенно уточняются, опять же по минимаксному принципу.

Пусть таким образом построены вершины W_1^-, W_2^+ и первые три конечные вершины (листья) – см. рис.5. Эти листья оценены статической функцией, и вершина W_2^+ получила точную минимаксную оценку 3, а вершина W_1^- – предварительную оценку 3. Далее при построении и раскрытии вершины W_3^+ статическая оценка первой ее дочерней вершины дает величину 4, которая становится предварительной оценкой самой вершины W_3^+. Эта предварительная оценка будет потом, после построения второй ее дочерней вершины, пересчитана. Причем согласно минимаксному принципу оценка может только увеличиться (поскольку подсчитывается как максимум оценок дочерних вершин), но даже если она увеличится, это не повлияет на оценку вершины W_1^-, поскольку последняя при уточнении по минимаксному принципу может только уменьшаться (так как равна минимуму оценок дочерних вершин). Следовательно, можно пренебречь второй дочерней вершиной для W_3^+, не строить и не оценивать ее, поскольку уточнение оценки вершины W_3^+ не повлияет на оценку вершины W_1^-. Такое сокращение поиска в игровом дереве называется ***отсечением*** ветвей.

Рис.5



Продолжим для нашего примера процесс поиска в глубину с одновременным вычислением предварительных (и точных, где это возможно) оценок вершин вплоть до момента, когда построены уже вершины W_4^- , W_5^+ и две дочерних последней, которые оцениваются статической функцией. Исходя из оценки первой дочерней вершины начальная вершина W_0^+, соответствующая исходной позиции игры, к этому моменту уже предварительно оценена величиной 3. Вершина W_5^+ получила точную минимаксную оценку 3, а ее родительская W_4^- получила пока только предварительную оценку 3. Эта предварительная оценка вершины W_4^-может быть уточнена в дальнейшем, и в соответствии с минимаксным принципом возможно только ее уменьшение, но это уменьшение не повлияет на оценку вершины W_0^+, поскольку последняя, опять же согласно минимаксному принципу, может только увеличиваться. Таким образом, построение дерева можно прервать ниже вершины W_4^-, отсекая целиком выходящие из нее вторую и третью ветви (и оставляя ее оценку предварительной).

На этом построение игрового дерева заканчивается, полученный результат – лучший первый ход – тот же самый, что и при минимаксной процедуре. У некоторых вершин дерева осталась неуточненная, предварительная оценка, однако этих приближенных оценок оказалось достаточно для того, чтобы определить точную минимаксную оценку начальной вершины и наилучший первый ход. В то же время произошло существенное сокращение поиска: вместо 17 вершин построено только 11, и вместо 10 обращений к статической оценочной функции понадобилось всего 6.

Обобщим рассмотренную идею сокращения перебора. Сформулируем сначала правила вычисления оценок вершин дерева игры, в том числе предварительных оценок промежуточных вершин, которые для удобства будем называть ***альфа-*** и ***бета-величинами***:

- концевая вершина игрового дерева оценивается статической оценочной функцией сразу, как только она построена;
- промежуточная вершина предварительно оценивается по минимаксному принципу, как только стала известна оценка хотя бы одной из ее дочерних вершин; каждая предварительная оценка пересчитывается (уточняется) всякий раз, когда получена оценка еще одной дочерней вершины;
- предварительная оценка ИЛИ-вершины (альфа-величина) полагается равной наибольшей из вычисленных к текущему моменту оценок ее дочерних вершин;
- предварительная оценка И-вершины (бета-величина) полагается равной наименьшей из вычисленных к текущему моменту оценок ее дочерних вершин.

Укажем очевидное следствие этих правил вычисления: альфа-величины не могут уменьшаться, а бета-величины не могут увеличиваться.

Сформулируем теперь правила прерывания перебора, или отсечения ветвей игрового дерева:.

A) Перебор можно прервать ниже любой И-вершины, бета-величина которой не больше, чем альфа-величина одной из предшествующих ей ИЛИ-вершин (включая корневую вершину дерева);

B) Перебор можно прервать ниже любой ИЛИ-вершины, альфа-величина которой не меньше, чем бета-величина одной из предшествующих ей И-вершин.

При этом в случае А говорят, что имеет место ***альфа-отсечение***, поскольку отсекаются ветви дерева, начиная с ИЛИ-вершин, которым приписана альфа-величина, а в случае В – ***бета-отсечение***, поскольку отсекаются ветви, начинающиеся с бета-величин).

Важно, что рассмотренные правила работают в ходе построения игрового дерева вглубь; это означает, что предварительные оценки промежуточных вершин появляются лишь по мере продвижения от концевых вершин дерева вверх к корню и реально отсечения могут начаться только после того, как получена хотя бы одна точная минимаксная оценка промежуточной вершины.

В рассмотренном примере на рис. 20 первое прерывание перебора было бета-отсечением, а второе – альфа-отсечением. Причем в обоих случаях отсечение было неглубоким, поскольку необходимая для соблюдения соответствующего правила отсечения альфа- или бета-величина находилась в непосредственно предшествующей к точке отсечения вершине. В общем же случае она может находиться существенно выше отсекаемой ветви – где-то на пути от вершины, ниже которой производится отсечение, к начальной вершине дерева, включая последнюю.

После прерывания перебора предварительные оценки вершин в точках отсечения остаются неуточненными, но, как уже отмечалось, это не препятствует правильному нахождению предварительных оценок всех предшествующих вершин, как и точной оценки корневой вершины и ее дочерних вершин, а значит, и искомого наилучшего первого хода.

На приведенных выше правилах вычисления оценок вершин и выполнения отсечений (всюду, где это возможно) основана альфа-бета процедура, являющаяся более эффективной реализацией минимаксного принципа. Справедливо следующее

Утверждение:

Альфа-бета процедура всегда приводит к тому же результату (наилучшему первому ходу), что и простая минимаксная процедура той же глубины.

Важным является вопрос, насколько в среднем альфа-бета процедура эффективнее обычной минимаксной процедуры. Нетрудно заметить, что количество отсечений в альфа-бета процедуре зависит от степени, в которой предварительные оценки (альфа- и бета-величины), полученные первыми, аппроксимируют окончательные минимаксные оценки: чем ближе эти оценки, тем больше отсечений и меньше перебор.

Таким образом, эффективность альфа-бета процедуры зависит от порядка построения и раскрытия вершин в дереве игры. Возможен и неудачный порядок просмотра, при котором придется пройти без отсечений через все вершины дерева, и в этом, худшем случае, альфа-бета процедура не будет иметь никаких преимуществ против минимаксной процедуры.

Наилучший случай, т.е. наибольшее число отсечений, достигается, когда при переборе в глубину первой обнаруживается конечная вершина, статическая оценка которой станет в последствии минимаксной оценкой начальной вершины. При максимальном числе отсечений строится и оценивается минимальное число концевых вершин. Показано, что в случае, когда самые сильные ходы всегда рассматриваются первыми, количество концевых вершин глубины N, которые будут построены и оценены альфа-бета процедурой, примерно равно числу концевых вершин, которые были бы построены и оценены на глубине N/2 обычной минимаксной процедурой. Таким образом, при фиксированном времени и памяти альфа-бета процедура сможет пройти при поиске вдвое глубже по сравнению с обычной минимаксной процедурой.

В заключение отметим, что статическая оценочная функция и альфа-бета процедура – две непременные составляющие почти всех компьютерных игровых программ (в том числе коммерческих). Часто используются также дополнительные эвристические приемы в самой альфа-бета процедуре:

- направленное (эвристическое) отсечение неперспективных ветвей: например, построение дерева игры обрывается на «пассивных» позициях, к которым применяется оценочная функция, для «активных» же позиций поиск продолжается дальше, до нужной глубины или даже глубже, поскольку на это можно использовать время, сэкономленное вследствие отсечения ветвей;
- последовательное углубление, при котором альфа-бета процедура применяется неоднократно: сначала до некоторой небольшой глубины (например, 2), а затем глубина увеличивается с каждой итерацией, причем после каждой итерации выполняется переупорядочение всех дочерних вершин – с тем, чтобы увеличить число отсечений в последующих итерациях;
- фиксированное упорядочение вершин при спуске-построении дерева вглубь, при котором в первую очередь строится и раскрывается дочерняя вершина, оцениваемая как более перспективная (эта оценка может быть проведена как с помощью статической оценочной функции, так и более простой, хотя и менее надежной эвристической функции);
- динамическое упорядочение вершин, при котором каждый раз после уточнения минимаксных оценок и проведения отсечений производится переупорядочивание вершин во всем построенном к текущему моменту дереве (с помощью некоторой эвристической функции) и для дальнейшего раскрытия выбирается наиболее перспективная вершина (заметим, что по существу это означает переход от классического перебора вглубь к алгоритму упорядоченного перебора на И/ИЛИ-графах).

Для усиления игры могут быть также использованы библиотеки типовых игровых ситуаций.

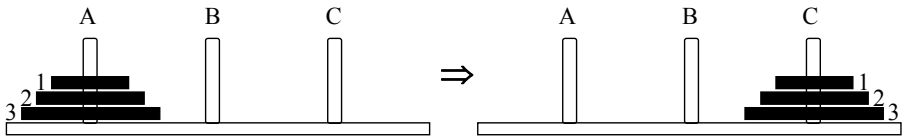
Редукция задач

Кроме уже рассмотренного подхода – представления задач в пространстве состояний – для решения ряда задач возможен и другой, более сложный подход. При этом подходе производится анализ исходной задачи с целью выделения такого набора подзадач, решение которых означает решение исходной задачи. Каждая из выделенных подзадач в общем случае является более простой, чем исходная, и может быть решена каким-либо методом, в том числе – с использованием пространства состояний. Но можно продолжить процесс, выделяя последовательно из возникающих задач подзадачи – до тех пор, пока не получим ***элементарные задачи***, решение которых уже известно. Такой путь называется подходом, основанным на ***сведении задач к подзадачам***, или на ***редукции задач.***

Для иллюстрации этого подхода рассмотрим один из вариантов известной головоломки – задачи о пирамидке, или ханойской башне. В ней используются 3 колышка (обозначим их буквами A, B, C) и 3 диска разного диаметра, которые можно нанизывать на колышки через отверстия в центре. В начале все диски расположены на колышке A, причем диски меньшего диаметра лежат на дисках большего диаметра – см. рис. 6 (диски занумерованы в порядке возрастания диаметра). Требуется переместить все диски на колышек C, двигая их по очереди и соблюдая следующие правила. Перемещать можно только самый верхний диск, и нельзя никакой диск класть на диск меньшего размера. На рис.6 показаны начальное и целевое состояния задачи о пирамидке.

Эта задача легко может быть формализована в пространстве состояний: состояние задачи задается списком из трех элементов, каждый из которых указывает местоположение соответствующего диска (первый элемент – первого диска, второй – второго, третий – третьего). Начальное состояние описывается списком (AAA), а целевое – (CCC). При этом предполагается, что если на одном колышке находится более одного диска, то любой больший диск расположен ниже любого меньшего.

Рис.6



Полное пространство состояний задачи о пирамидке включает 27 вершин.

Посмотрим, как можно решить эту задачу методом редукции задач. Ключевая идея состоит в том, что для перемещения всей пирамидки необходимо переложить самый нижний диск 3, а это возможно, только если располагающаяся над ним пирамидка из двух меньших дисков 1 и 2 перенесена на колышек B – см. рис. 7(а). Таким образом, исходную задачу можно свести к следующим подзадачам:

1) переместить диски 1 и 2 с колышка A на колышек B (1, 2 : A ✹ B);

2) переместить диск 3 с колышка A на колышек C (3 : A ✹ C);

3) переместить диски 1 и 2 с колышка B на колышек C (1, 2 : B ✹ C).

На рис. 7(б) показано исходное состояние для третьей задачи.

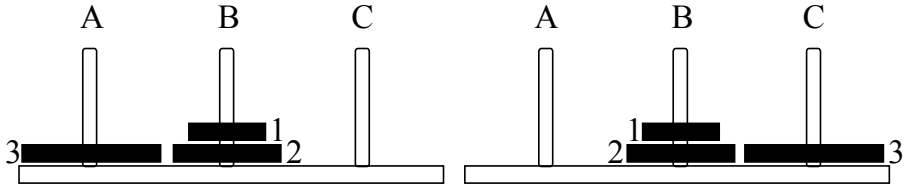
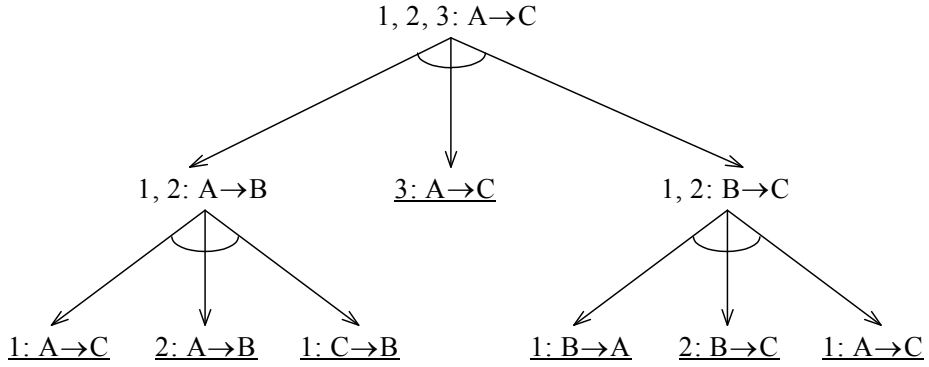


Рис.7 а) б)

Каждая из трех указанных задач проще исходной: действительно, в первой и в третьей задачах требуется переместить всего два диска, вторая же задача может рассматриваться как элементарная, так как ее решение состоит ровно из одного хода – перемещения диска. В первой и третьей задачах можно

Рис.8



вновь применить метод редукции задач, и свести их к элементарным задачам. Весь процесс редукции можно схематически представить в виде дерева (рис. 8). Вершины дерева соответствуют решаемым задачам/подзадачам, причем листья дерева – элементарным задачам перемещения дисков, а дуги связывают редуцируемую задачу с ее подзадачами.

Заметим, что рассмотренная стратегия сведения задачи к совокупности подзадач может быть применена и в случае, когда начальная конфигурация задачи о пирамидке содержит не три, а большее число дисков. В любом случае, существенным является порядок, в котором решаются результирующие задачи: например, вторая задача не может быть решена ранее первой. Таким образом, в случае подхода, основанного на редукции задач, мы получаем также пространство, но состоящее не из состояний, а из задач/подзадач (точнее, их описаний). При этом роль, аналогичную операторам в пространстве состояний, играют операторы, сводящие задачи в подзадачи. Точнее, каждый ***оператор редукции*** преобразует описание задачи в описание множества результирующих подзадач, причем это множество таково, что решение подзадач обеспечивает решение редуцированной задачи.

При решении задач методом редукции, как и при решении в пространстве состояний, может возникнуть необходимость перебора. Действительно, на каждом этапе редукции может оказаться несколько применимых операторов (т.е. способов сведения задачи к подзадачам) и, соответственно,

несколько альтернативных множеств подзадач. Некоторые способы, возможно, не приведут к решению исходной задачи, поскольку обнаружатся неразрешимые подзадачи, другие же способы могут дать окончательное решение. В общем случае для полной редукции исходной задачи необходимо перепробовать несколько операторов. Процесс редукции продолжается, пока исходная задача не будет сведена к набору элементарных задач, решение которых известно.

Аналогично представлению в пространстве состояний, формализация задачи в рамках подхода, основанного на редукции задач, включает определение следующих составляющих:

- формы описания задач/подзадач и описание исходной задачи;
- множества операторов и их воздействий на описания задач;
- множества элементарных задач.

Эти составляющие задают неявно ***пространство задач***, в котором требуется провести поиск решения задачи.

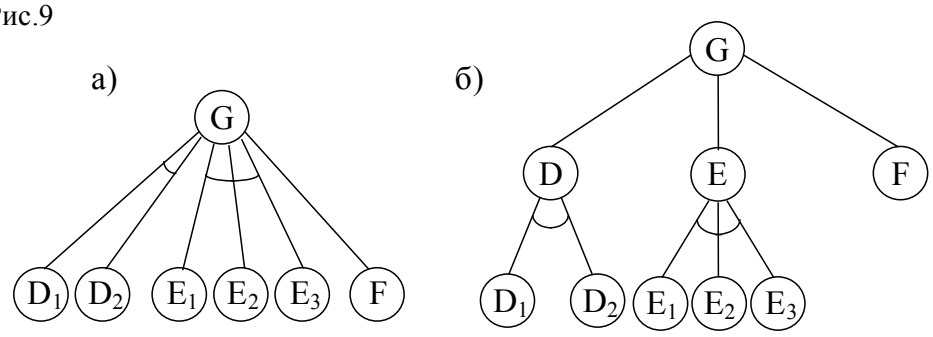
Что касается формы описания задач/подзадач, то часто их удобно описывать в терминах пространства состояний, т.е. задавая начальное состояние и множество операторов, а также целевое состояние или его свойства. В этом случае элементарными задачами могут быть, к примеру, задачи, решающиеся за один шаг перебора в пространстве состояний.

В дополнение отметим, что подход с использованием пространства состояний можно рассматривать как вырожденный случай подхода, основанного на редукции задач, так как применение оператора в пространстве состояний сводит обычно исходную задачу к несколько более простой задаче, т.е. редуцирует ее. При этом результирующее множество подзадач состоит только из одного элемента, т.е. имеем простейший случай замены редуцируемой задачи на ей эквивалентную.

И/ИЛИ графы. Решающий граф

Для изображения процесса редукции задач и получающихся при этом альтернативных множеств подзадач используются обычно графоподобные структуры, вершины которых представляют описания задач/подзадач, а дуги связывают любую пару вершин, соответствующих редуцируемой задаче и одной из результирующих подзадач, причем стрелки на дугах указывают направление редукции. Пример такой структуры приведен на рис. 10 (а): задача G может быть решена путем решения либо задач D_1 и D_2, либо E_1, E_2 и E_3 либо задачи F. При этом ребра, относящиеся к одному и тому же множеству подзадач, связываются специальной дугой. Чтобы сделать такую структуру более наглядной, вводятся дополнительные промежуточные вершины, и каждое множество результирующих задач группируется под своей родительской вершиной. При этом структура на рис.10(а) преобразуется в структуру, изображенную на рис.10(б): для двух из трех альтернативных множеств подзадач добавлены соответственно вершины D и E.

Рис.9



Если считать, что вершины D и E соответствуют описаниям альтернативных путей решения исходной задачи, то вершину G можно назвать ***ИЛИ-вершиной***, так как задача G разрешима или способом D, или способом E, или способом F. Аналогично вершины D и E можно назвать ***И-вершинами***, поскольку каждый из соответствующих им способов требует решения всех подчиненных задач, что и обозначается специальной дугой. По этой причине структуры, подобные структурам, изображенным на рис.9(б) и рис.10, называются ***И/ИЛИ-графами***.

Если некоторая вершина такого графа имеет непосредственно следующие за ней (дочерние) вершины, то либо все они являются И-вершинами, либо все они – ИЛИ-вершины. Заметим, что если у некоторой вершины И/ИЛИ-графа имеется ровно одна дочерняя вершина, то ее можно считать как И-вершиной, так и ИЛИ-вершиной – как, например, вершину F на рис.9(б) .

На языке И/ИЛИ-графов применение некоторого оператора редукции задачи будет означать, что сначала будет построена промежуточная И-вершина, а затем непосредственно следующие за ней ИЛИ-вершины подзадач. Исключение составляет случай, когда множество задач состоит только из одного

элемента, в этом случае будет образована ровно одна вершина, будем для определенности считать ее ИЛИ-вершиной.

Вершину И/ИЛИ-графа, соответствующую описанию исходной задачи, будем называть ***начальной*** вершиной. Вершины же, которые соответствуют описаниям элементарных задач, будем называть ***заключительными*** вершинами. В графе, показанном на рис.10, начальной является вершина P_0, а заключительными – вершины P_1, P_4, P_5, P_7 и P_8 (они изображены жирными кружками).

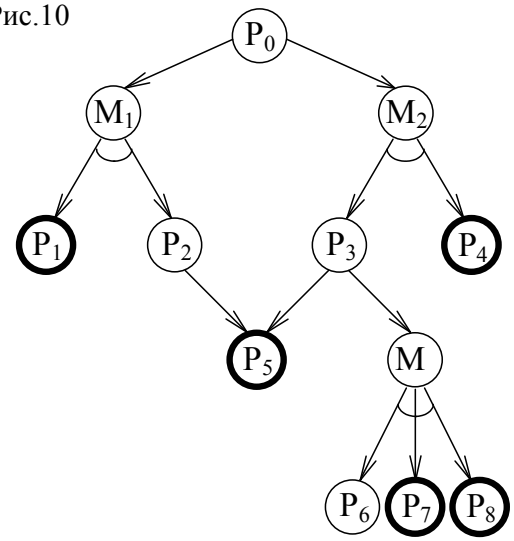
Поиск, осуществляемый путем перебора вершин графа, применим и в подходе, основанном на редукции задач. Цель поиска на И/ИЛИ-графе – показать, что ***разрешима*** исходная задача, т.е. начальная вершина. Разрешимость этой вершины зависит от разрешимости других вершин графа.

Сформулируем общее рекурсивное определение разрешимости вершины в И/ИЛИ-графе:

- заключительные вершины разрешимы, так как они соответствуют элементарным задачам;
- вершина, не являющаяся заключительной и имеющая дочерние И-вершины, разрешима тогда и только тогда, когда разрешима по крайней мере одна из ее дочерних вершин;
- вершина, не являющаяся заключительной и имеющая дочерние ИЛИ-вершины, разрешима тогда и только тогда, когда разрешима каждая из ее дочерних вершин.

Если в процессе поиска удалось показать, что начальная вершина разрешима, то это значит, что обнаружено решение исходной задачи, которое заключено в так называемом решающем графе. ***Решающий граф*** – это подграф И/ИЛИ-графа, состоящий только из разрешимых вершин и доказывающий разрешимость начальной вершины.

Рис.10



Для И/ИЛИ-графа, изображенного на рис.10, разрешимыми являются (кроме заключительных) вершины M_1, M_2, P_2, P_3. Этот граф содержит два решающих графа: первый состоит из вершин P_0, M_1, P_1, P_2 и P_5 ; а второй – из вершин P_0, M_2, P_3, P_4 и P_5. Заметим, что вершины M_3 и P_6 не являются разрешимыми (M_3 неразрешима в силу неразрешимости вершины P_6).

Пример: задача символьного интегрирования

В качестве примера применения метода редукции рассмотрим решение задачи символьного интегрирования, т.е. нахождения неопределенного интеграла $\int F(x)dx$. Обычно эта задача решается путем последовательного преобразования интеграла к выражению, содержащему известные табличные интегралы. Для этого используется несколько правил интегрирования, в том числе: правило интегрирования суммы функций, правило интегрирования по частям, правило вынесения постоянного множителя за знак интегрирования, а также применение алгебраических и тригонометрических подстановок и использование различных алгебраических и тригонометрических тождеств.

Для формализации этой задачи в рамках подхода, основанного на редукции задач, необходимо определить форму описания задач/подзадач, операторы редукции и элементарные задачи. В качестве возможной формы описания задач может быть взята символьная строка, содержащая запись подынтегральной функции и переменной интегрирования (если последняя не фиксирована заранее). Операторы редукции будут основаны, очевидно, на упомянутых правилах интегрирования. Например, правило интегрирования по частям $\int udv = u\int dv - \int vdu$ сводит исходную задачу (неопределенный интеграл в левой части равенства) к двум подзадачам интегрирования (два соответствующих интеграла в правой части равенства).

Заметим, что часть получаемых таким образом операторов редукции (как, например, операторы, соответствующие правилу интегрирования по частям и правилу интегрирования суммы функций), действительно редуцируют исходную задачу и порождают И-вершину в И/ИЛИ-графе, в то время как алгебраические и тригонометрические подстановки и тождества (как, например, деление числителя на знаменатель или дополнение до полного квадрата) лишь заменяют одно подынтегральное выражение на другое, порождая, таким образом, ИЛИ-вершины.

Элементарные задачи интегрирования соответствуют табличным интегралам, к примеру, ⌠sin(x)dx = - cos(x) + C. Отметим, что поскольку каждая из таких табличных формул содержит переменные, на самом деле она является схемой, задающей бесконечное множество элементарных задач.

Особенностью задачи интегрирования является то, что, например, при интегрировании по частям может оказаться несколько способов разбиения исходного подынтегрального выражения на части и соответственно несколько способов применения этого правила интегрирования. Это означает, что в общем случае для одного правила возможно несколько вариантов редукции задачи, т.е. несколько способов применения одного и того же оператора редукции.

Другая особенность рассматриваемой задачи состоит в том, что на каждом шаге процесса редукции применимо обычно большое количество операторов (включая несколько применений одного и того же оператора), и получающийся И/ИЛИ-граф задачи слишком велик даже для несложных задач интегрирования. Поэтому, чтобы сделать поиск на таком графе достаточно эффективным, необходимо как-то ограничивать и/или упорядочивать множество порождаемых при поиске вершин. К примеру, можно упорядочить операторы редукции по степени их полезности, и приписать больший приоритет операторам, соответствующим правилам интегрирования суммы и интегрирования по частям.

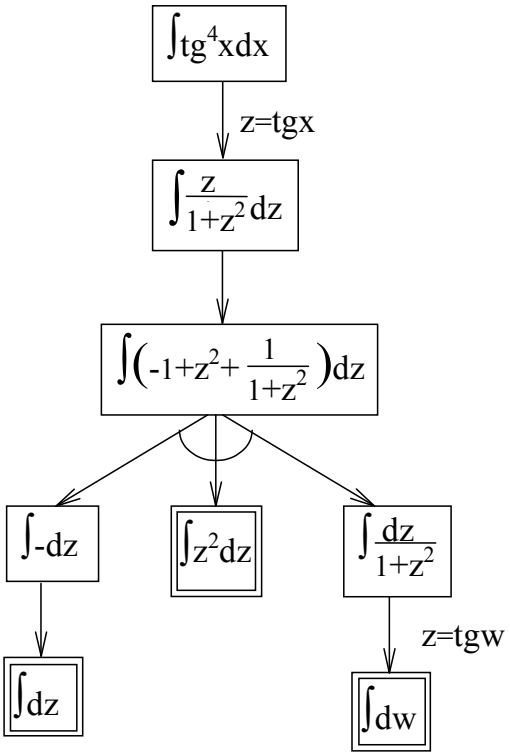
На рис.11 показан решающий граф для одной задачи интегрирования. В вершинах графа указаны соответствующие задачи/подзадачи, заключительные вершины заключены в двойные рамки. Решение задачи может быть собрано из содержащейся в графе информации. Оно состоит из следующих шагов:

1. применения подстановки z=tg(x);
2. эквивалентного преобразования подынтегрального выражения;
3. применения правила интегрирования суммы функций.

При этом одна из трех результирующих подзадач оказывается элементарной, а две остальные решаются за один шаг (первая – путем вынесения постоянного множителя за знак интеграла, вторая – применением подстановки z=tg(w)).

Подход, основанный на редукции задач, применим и имеет преимущества по сравнению с подходом, использующем представление в пространстве состояний, когда получающиеся при редукции подзадачи можно решать независимо друг от друга, как в примере с интегрированием. Впрочем, это условие взаимной независимости результирующих задач можно несколько ослабить. Метод редукции применим также, если решения получающих подзадач зависят друг от друга, но при этом существует такой порядок их редукции, при котором найденные решения одних, более ранних подзадач не разрушаются при решении других, более поздних – как в головоломке о ханойской башне, в которой важен лишь порядок решения выделяемых подзадач.

Рис.11


$\int tg^4 x dx$
z=tgx
$\int \frac{z}{1+z^2} dz$
$\int \left(-1+z^2+\frac{1}{1+z^2}\right) dz$
$\int -dz$
$\int z^2 dz$
$\int \frac{dz}{1+z^2}$
z=tgw
$\int dz$
$\int dw$