Решение задач в ИИ

Подходы

- Поиск решения в пространстве состояний
- Редукция к подзадачам
- Доказательство теорем

State space search (SS-problem)

Базовая постановка задачи:

- S – множество состояний системы;
- S_0 – начальное состояние (или множество состояний);
- F – множество операторов, отображающих S в S;
- T – множество целевых состояний;

Найти: последовательность [f1,f2,…], которая обеспечивает переход из начального состояния в некоторое целевое. Иногда в качестве целей рассматривают последовательность состояний от начального до целевого (что практически эквивалентно).

Вариации задач

- Множество состояний может отсутствовать – операторы определяют результат применения к некоторому состоянию и порождают дочерние состояния;
- Может вводится функция стоимости, определенная на множестве операторов – цена применения данного оператора (соответствует весам дуг в графах);
- Оценочная функция для последовательности операторов может отсутствовать – нужно любым способом получить решение или установить, что задача не имеет решения.

Операторы

Т.к. множество состояний может отсутствовать или иметь слишком много элементов, то операторы могут описываться в виде некоторых правил, например, правил (продукций) формальных грамматик, и применяться для вывода новых состояний. В этом случае оператор – не функция, связывающая два состояния из множества состояний, а некоторое средство для построения нового состояния.

Начальное состояние (S_0): *abaabab* – строка;

Множество состояний (S): множество строк, получаемых из исходной применением правил переписывания (операторов);

Операторы (F): правила переписывания:

$$1: \$_1 ab \$_2 \rightarrow \$_1 P \$_2$$

$$2: \$_1 aP \$_2 \rightarrow \$_1 P \$_2$$

$$3: \$_1 Pb \$_2 \rightarrow \$_1 P \$_2$$

$$4: \$_1 PP \$_2 \rightarrow \$_1 P \$_2$$

где P – предложение, a,b – символы

Целевое состояние (T): P

Способы описания операторов

Наиболее простой и очевидный способ – это неявное описание, интегрированное в алгоритм поиска. В более сложных случаях:

- Продукции (правила переписывания), имеющие вид:

- Графовое представление – граф, вершинами которого являются состояния либо множества состояний, дуги соответствуют операторам;
- Описание в виде конечных (или абстрактных) автоматов, с соответствующими способами описания переходов – например, диаграммами состояний.

Представление пространства состояний

Аналогичны способам представления графов:

- Списки смежных вершин – для разреженных графов (sparse graphs);
- Матрицы смежности – для плотных (dense graphs);
- Другие – например, неявное представление при рекурсии.

Оценка алгоритмов поиска

Свойства:

- Полнота – будет ли найдено решение, если оно существует;
- Оптимальность – будет ли найдено оптимальное решение;
- Временная сложность – время, которое требуется для нахождения решения;
- Пространственная сложность – требования к ресурсам (памяти).

Базовая стратегия поиска

1. Поместить s в <ОТКРЫТ>;
2. Пока <ОТКРЫТ> не пуст:
    1. Взять очередную вершину (n) из <ОТКРЫТ>;
    2. Если n – целевая, то закончить поиск (успех);
    3. Поместить ее в <ЗАКРЫТ>;
    4. Раскрыть n, образовав все дочерние вершины и занести их в <ОТКРЫТ>, указав для каждой родителя – n;
3. Анализ результата поиска.

Вариации

Поиск в глубину

- Если используются стеки – поиск в глубину (Depth-first search/process);
- С ограничением глубины;
- Итеративный поиск в глубину (Iterative-depth search, с постепенным углублением);
- Поиск с возвратами;

Поиск в ширину

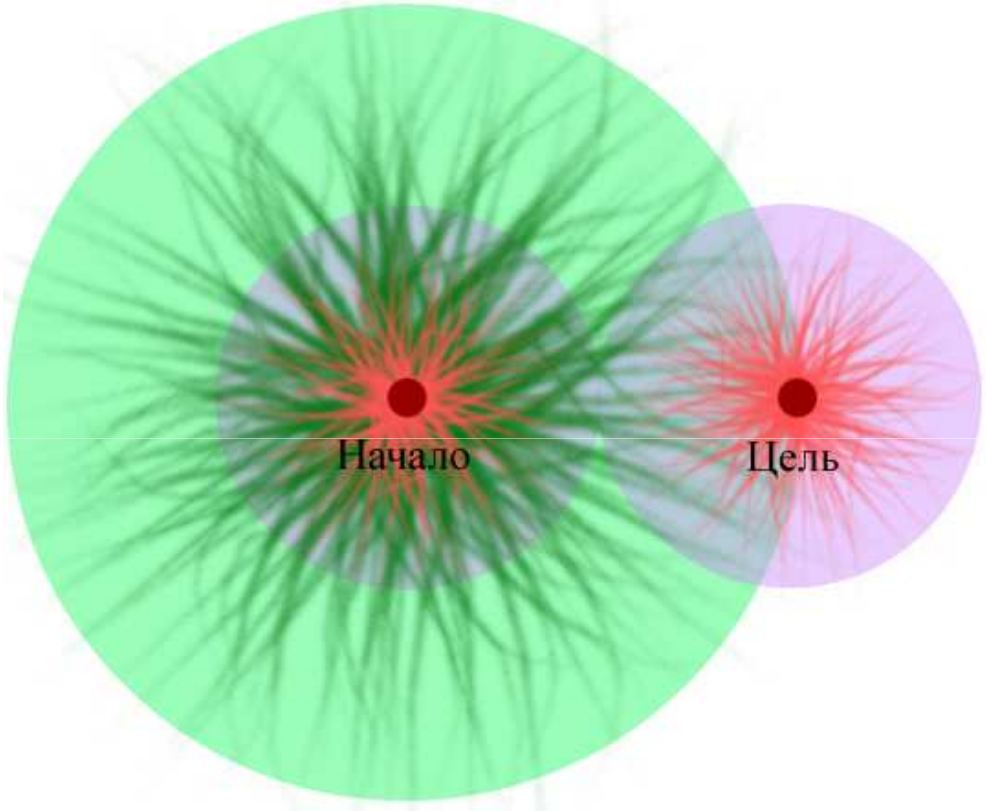
- Очереди – поиск в ширину (Breadth-first search/process);
- Двунаправленный поиск (Bi-directional search);

Поиск по критерию стоимости

- Очереди с приоритетами – метод равных цен (алгоритм Дейкстры). В этом случае добавляется оценка операторов (ребер графа), и оценка состояний как сумма оценок (весов) ребер от начального состояния к текущему.

Вторая структура – <ЗАКРЫТ> – только в том случае, если множество операторов порождает не дерево состояний, а граф – возможны возвраты (*алгоритмы, которые забывают свою историю, обречены на то, чтобы ее повторять*).

Двунаправленный поиск



Плотность «кругов» будет расти при удалении от центра, поэтому двунаправленный поиск может быть более эффективен (особенно при большом коэффициенте ветвления), нежели однонаправленный, но потребует дополнительных затрат для хранения информации о противоположных фронтах состояний и проверки принадлежности.

Стоимость поиска

Глубина	Кол-во узлов	Время	Память
2	1100	0,11 с	1 Мб
4	111100	11 с	106 Мб
6	10e7	19 мин	10 Гб
8	10e9	31 час	1 Тб
10	10e11	129 суток	101 Тб
12	10e13	35 лет	10 Петабайт
14	10e15	3523 года	1 Эксабайт

Коэффициент ветвления b = 10 (из каждого состояния выводится 10 производных), скорость формирования – 10000 узлов/с, память – 1000 байт/узел.

Информированный поиск

Эвристика (от греч. *heurésko* — отыскиваю, открываю), в данном случае некоторое эмпирическое правило, содержащее дополнительную информацию о задаче, состояниях и проч., и помогающее найти решение.

Используется функция *h(n)*, позволяющая оценить «перспективность» решения.

$$f(n) = g(n) + h(n)$$

Смысл *h(n)* – оценка количества ходов (в общем случае – «расстояния», или «близости») в лучшем пути из текущего состояния до целевого.

Как правило, «*качество*» *h* – то, насколько хорошо она оценивает *g,* оценка снизу или сверху – это существенно!

Мера «*информированности*» алгоритмов – сравнение их эвристических функций. Если для любого возможного состояния функция $h_1(n)$ дает не менее точную оценку (а для некоторых состояний более точную), чем функция $h_2(n)$, то такая функция считается более «качественной», или «информированной».

Вариации эвристического поиска

Жадный поиск по первому наилучшему совпадению (*Best-first search*) – использование только эвристической информации. В данной вариации нам не существенно, сколько ходов мы уже сделали, важно то, насколько «близко» мы подобрались к решению. Для раскрытия выбираются наиболее «перспективные» в этом плане состояния из рассматриваемых.

Алгоритм A* - оценка вершины как сумма стоимости и эвристической оценки. Как правило, при некоторых условиях – полный, оптимальный и оптимально эффективный.

Замечание: даже для A* потребности в памяти все еще растут экспоненциально по отношению к длине пути.

Допустимость и оптимальность A*

Эвристическая функция *допустима*, если она не «*переоценивает*» минимальную стоимость решения. Пусть C* – стоимость оптимального решения, n – некоторая вершина на оптимальном пути, на котором достигается оптимальное решение, m – неоптимальная целевая.

$$f(m) = g(m) + h(m) = g(m) > C^*$$

$$f(n) = g(n) + h(n) \le g(n) + H(n) = C^*$$

$$f(n) < f(m),$$

но тогда мы не должны были рассматривать m раньше n.

Доказательство оптимальной эффективности проистекает из разбиения множества вершин на подмножества с одинаковой оценкой. Алгоритм рассматривает эти подмножества последовательно, что гарантирует минимум раскрытых вершин.

При использовании преемственной (монотонной) эвристической функции нет необходимости в пересчете оценок для уже открытых вершин, и оптимальность сохраняется для графов:

$$h(n_1) \le c(n_1,n_2) + h(n_2)$$ – требование преемственности

Смысл: реальное положение дел хуже, чем казалось.

Формальное доказательство приведено в [1]

Методы ветвей и границ

Используются две оценки – сверху и снизу:

- Строится ветвление – раскрываются порождаемые варианты текущего этапа.
- Вычисляется минимум оценки сверху – C**
- Отбрасываются те варианты (строятся границы), оценка снизу для которых больше оценки C**

Данные методы позволяют модифицировать поиск в глубину, отсекая заведомо «плохие» варианты.

Сравнение стратегий поиска

	Стоимость поиска			Эффективный коэффициент ветвления		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	–	539	113	–	1,44	1,23
16	–	1301	211	–	1,45	1,25
18	–	3056	363	–	1,46	1,26
20	–	7276	676	–	1,47	1,27
22	–	18094	1219	–	1,48	1,28
24	–	39135	1641	–	1,48	1,26

Алгоритмы локального поиска и задачи оптимизации

При некоторых особенностях задачи получаем другой класс задач:

- Систематическое исследование пространства состояний не требуется, или невозможно;
- Может отсутствовать информация о множестве целевых состояний – например, как в задаче поиска максимума функции;
- Нет необходимости в нахождении пути к решению, требуется лишь найти целевое состояние – например, задача о расстановке ферзей на шахматной доске.

Особенности алгоритмов локального поиска

- Действуют с учетом только текущего состояния
- Информация о путях не сохраняется
- Низкие требования к памяти
- Работа в больших или бесконечных пространствах состояний

Ландшафт пространства состояний

Проблемы поиска

- Локальные экстремумы – оценка состояния как целевого ввиду отсутствия приемлемых производных состояний
- Хребты – тяжелы для прохождения
- Плато или уступы – нет очевидных следующих ходов

Алгоритмы продвижения к вершине

- Жадный алгоритм – выбирается наилучший следующий ход;
- Стохастический поиск с восхождением к вершине – следующий ход выбирается случайно из допустимых;
- Поиск с восхождением к вершине с выбором первого варианта;
- Поиск с восхождением к вершине и перезапуском случайным образом – последовательно случайно генерируются начальные состояния, из которых ведется поиск;
- Локальный лучевой поиск – параллельное отслеживание нескольких наиболее перспективных преемников;

Алгоритм отжига (эмуляция отжига) или симуляция восстановления (simulated annealing)

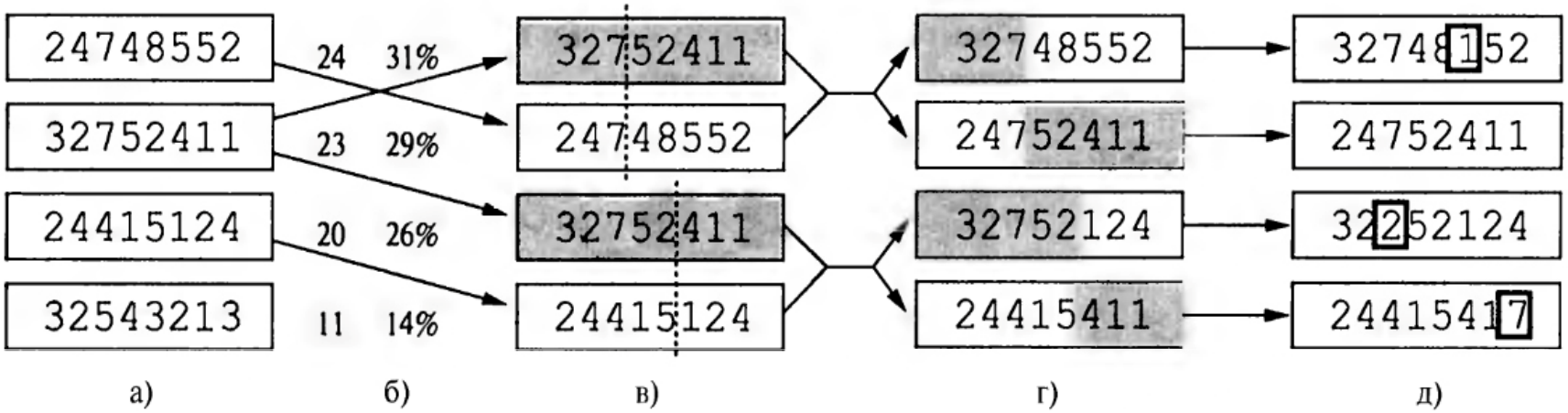
1. Создать начальное решение
2. Повторять:
    1. Установить температуру T как функцию от времени;
    2. Создать случайное решение на основе текущего;
    3. Оценить новое решение:
        1. Если новое лучше – принять в качестве текущего
        2. Если новое хуже – принять в качестве текущего с вероятностью $P(dE)= \exp(-dE/T)$ для поиска минимума.

С течением времени плохие шаги станут недоступными, и получим аналог стохастического поиска. Наиболее часто температуру просто понижают в зависимости от количества сделанных шагов

Генетические алгоритмы (Genetic algorithms)

- Вариант стохастического лучевого поиска;
- Состояния обираются для дальнейшего исследования на основе некоторой функции «отсеивания»;
- Функция пригодности оценивает перспективность состояния для скрещивания – стремление к максимуму;
- Преемники создаются комбинированием на основе двух родительских состояний;
- Вводится понятие мутации – некоторой вариации индивидов, что обеспечивает случайное исследование.

Вариант генетического алгоритма (задача о ферзях)



а) функция пригодности (количество неатакующих друг друга пар ферзей)

б) отбор

в) скрещивание (случайное перемешивание информации)

д) мутация – перемещение одного ферзя – случайность

Можно использовать ***схемы*** – некоторые характерные «блоки». Пример схемы – строка с пустыми ячейками

Локальный поиск в непрерывных пространствах

- Задача о размещении аэропортов;
- Дискретизация множества состояний;
- Градиентные методы (возможно, эмпирический градиент);
- Сведение к линейному поиску, хотя и является дорогостоящим в многомерных пространствах – переход к приближенным алгоритмам;
- Задачи с ограничениями – линейное программирование.

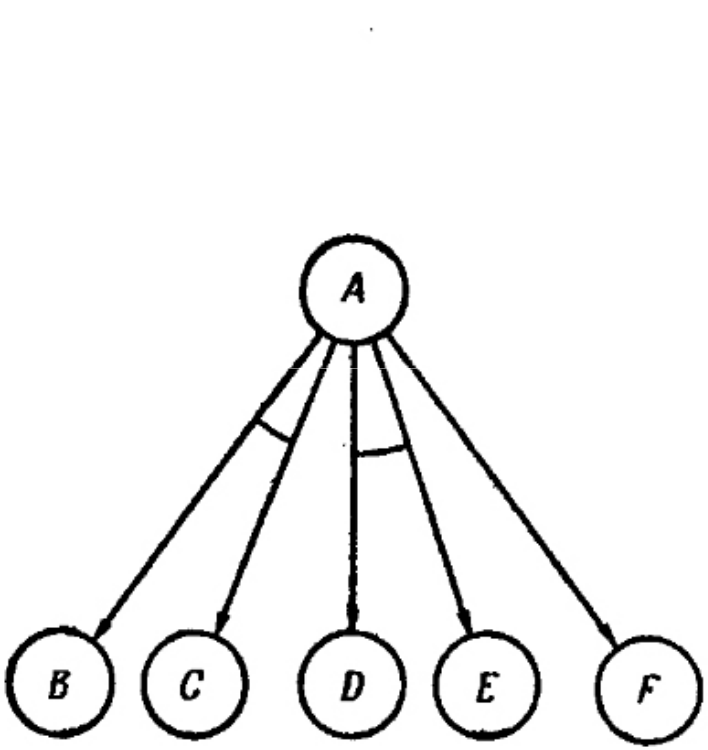
Поисковые агенты

- Соответствует задачам функционирования роботов в неизвестной среде;
- Множества состояний и возможных действий неизвестно, но детерминированы;
- Есть наблюдаемое состояние, и на его основе – множества допустимых действий с некоторой заданной стоимостью и, как следствие, производные состояния;
- Нужны алгоритмы с минимальным *коэффициентом конкурентоспособности* (>1);
- Даже если состояние безопасно исследуемо, КК может быть бесконечным;
- Пример – Learning Real Time A* (LRTA*).

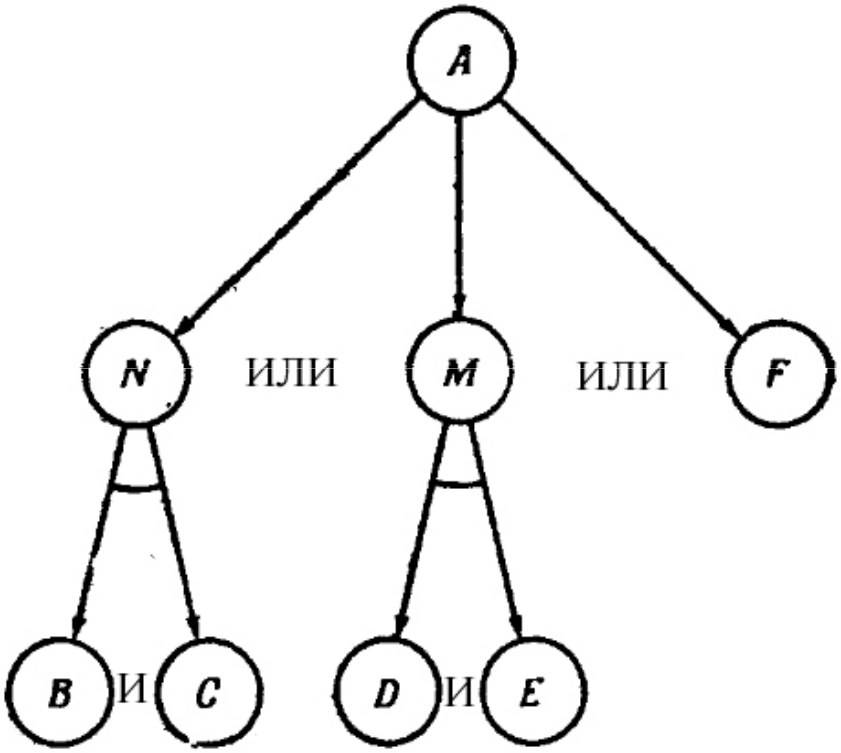
Метаалгоритмы

- Варианты определения: «*Выполняют поиск в пространстве алгоритмов*» либо «*Генерируют новые алгоритмы на основе изученных*»;
- Требуется возможность строить комбинации алгоритмов – например, суперпозиции, линейные комбинации и проч.;
- Возможна работа как с простыми, так и с составными алгоритмами;
- Проблема замыкания множества алгоритмов;
- Проблема обратной связи – оценки результатов;
- Проблема устойчивости/изменчивости;
- Пример: проект Robocode

Редукция к подзадачам
PR-problem



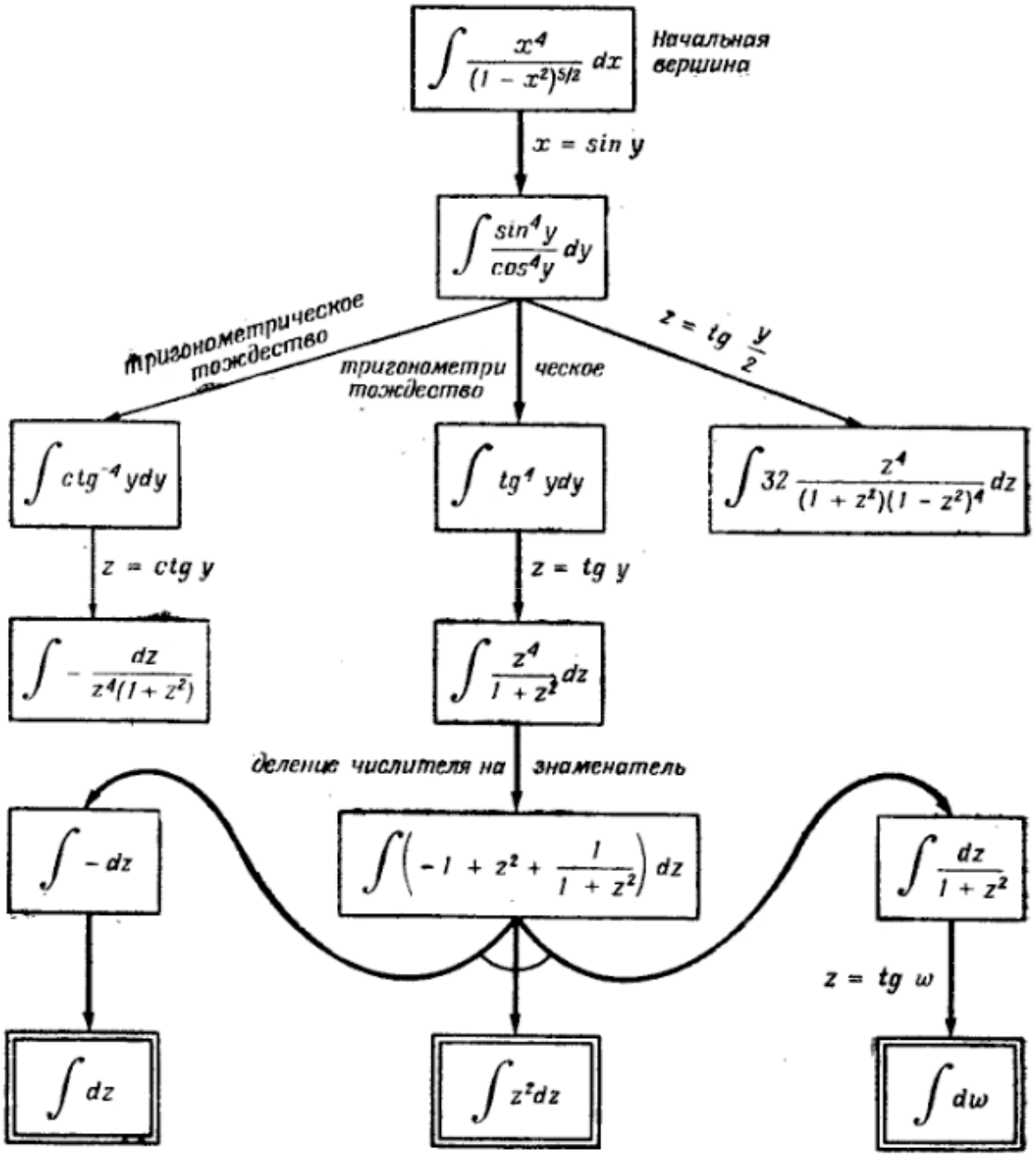
Множества подзадач для A



“И/ИЛИ” граф

Описание

- Начальная вершина – описание исходной задачи;
- Заключительные вершины – соответствуют описанию элементарных задач, которые могут быть решены;
- Оператор Г – порождает дочерние вершины;
- Задача – определить, разрешима ли начальная вершина. Заключительные вершины разрешимы, для остальных – очевидно;
- Требуется найти решающий граф – подграф из разрешимых вершин;
- Процесс решения – построение части «И/ИЛИ» графа, содержащей решающий граф.


$\int \frac{x^4}{(1-x^2)^{5/2}}\,dx$
Начальная вершина
$x = \sin y$
$\int \frac{\sin^4 y}{\cos^4 y}\,dy$
тригонометрическое тождество
тригонометрическое тождество
$z = \operatorname{tg} \frac{y}{2}$
$\int \operatorname{ctg}^4 y\,dy$
$\int \operatorname{tg}^4 y\,dy$
$\int 32 \frac{z^4}{(1+z^2)(1-z^2)^4}\,dz$
$z = \operatorname{ctg} y$
$z = \operatorname{tg} y$
$\int -\frac{dz}{z^4(1+z^2)}$
$\int \frac{z^4}{1+z^2}\,dz$
деление числителя на знаменатель
$\int -dz$
$\int \left(-1 + z^2 + \frac{1}{1+z^2}\right) dz$
$\int \frac{dz}{1+z^2}$
$z = \operatorname{tg} \omega$
$\int dz$
$\int z^2 dz$
$\int d\omega$

Поиск – алгоритмы разметки

- Разрешимые вершины – заключительные вершины, либо имеют дочерние вершины типа «И» и они все разрешимы, либо аналогично для дочерних вершин типа «ИЛИ»;
- Неразрешимые – аналогично;
- Процедуры разметки устанавливают разрешимость исходной вершины, работая с графом перебора – подграфом неявно заданного «И/ИЛИ» графа.

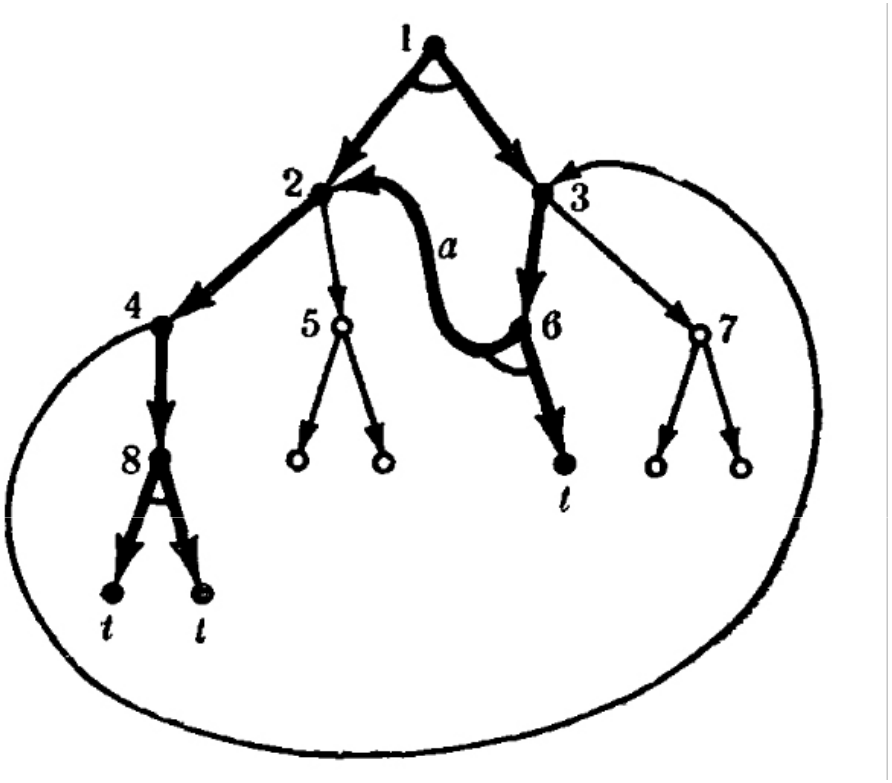
Алгоритмы поиска

- Практически аналогичны уже рассмотренным – раскрываем вершины и анализируем;
- Алгоритм разметки разрешимых вершин – очевиден, позволяет установить как разрешимые, так и неразрешимые вершины;
- Можно удалять состояния из рассмотрения, если установлено, что они неразрешимы, но аккуратно отслеживать циклы!

Проблемы поиска

- Эквивалентные подзадачи – проблема «сравнения» двух подзадач;
- Проблема циклов – уже развертывали вершину;
- У подзадачи может существовать несколько родителей – проблема с указателями на родителей;
- Проблема определения глубины – как считать глубину в графе?
- Зачастую требуется найти дерево решения минимальной стоимости;
- Суммарная стоимость решения – сумма стоимости всех дуг, максимальная стоимость дерева решения – стоимость максимального пути (например, некий аналог задач о расписаниях).

Проблема цикла



Если вершины раскрываются в соответствии с номерами, то необходимо включить в рассмотрение 2 как дочернюю для 6, хотя 6 в то же время уже является дочерней для 2. Тут неясно, где можно разорвать цикл.

Вычисление оценок

Концевая вершина

Заключительная вершина:

0

Не является заключительной:

не определено

Неконцевая «ИЛИ»-вершина

$$\hat{h}(n) = \min_i [c(n, n_i) + \hat{h}(n_i)]$$

Неконцевая «И»-вершина

Суммарные стоимости:

$$\hat{h}(n) = \sum_{i=1}^{k} [c(n, n_i) + \hat{h}(n_i)]$$

Максимальные стоимости:

$$\hat{h}(n) = \max_i [c(n, n_i) + \hat{h}(n_i)]$$

Принцип выбора потенциального дерева решения

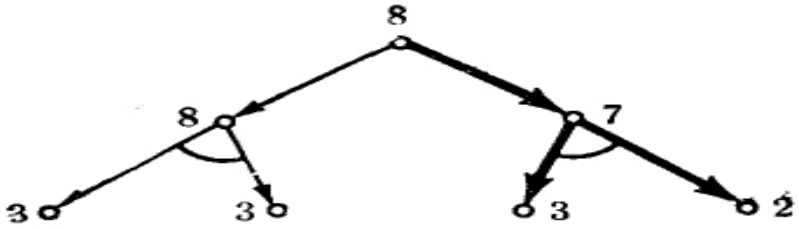
- Начальная вершина входит;
- Из вершин типа «ИЛИ» берем вершину с минимальной оценкой (в случае равных оценок выбор произволен);
- Из вершин типа «И» - входят все дочерние.

Алгоритм упорядоченного перебора для деревьев «И/ИЛИ»

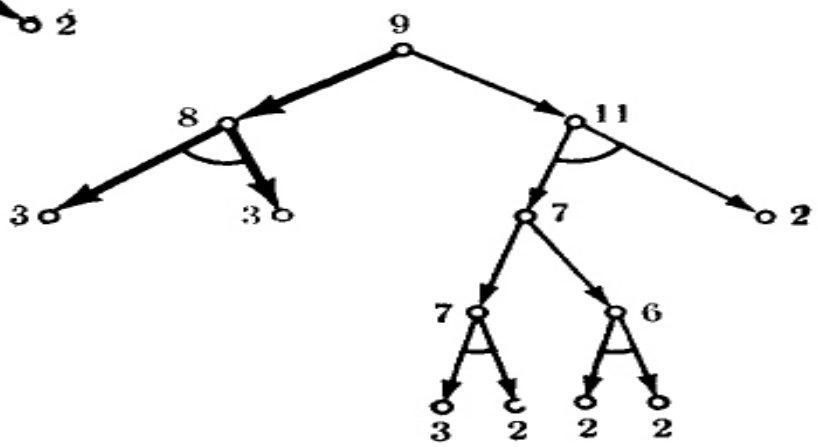
1. Поместить начальную вершину s в список ОТКРЫТ, вычислить для нее эвристическую оценку;
2. Получить потенциальное дерево решения T согласно описанному алгоритму;
3. Выбрать некоторую концевую вершину n дерева T из списка ОТКРЫТ, перенести ее в список ЗАКРЫТ;
4. Если n – заключительная, то пометить как разрешимую и продолжать, иначе перейти к (9);
5. Применить к T процедуру разметки вершин;
6. Проверить начальную вершину на разрешимость;
7. Изъять из ОТКРЫТ все вершины, имеющие предшествующие разрешимые вершины;

продолжение

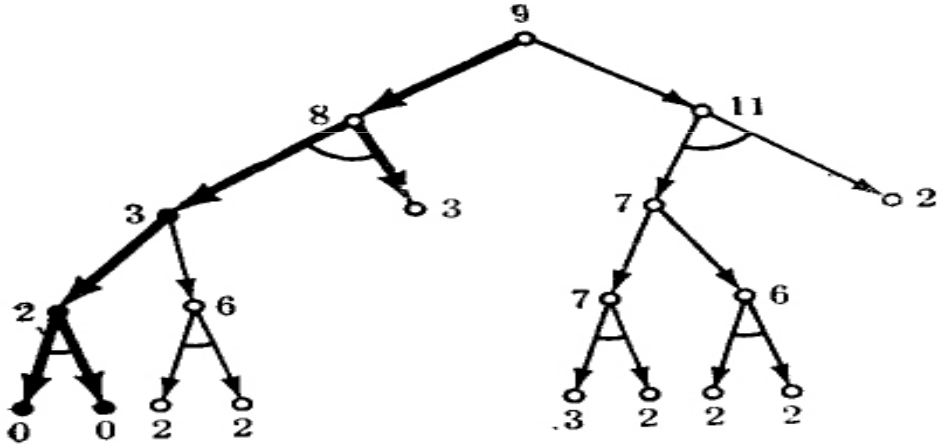
8. Перейти к (2);
9. Применить к n оператор Г. Если дочерних нет, то пометить n как неразрешимую, иначе перейти к (14);
10. Разметить неразрешимые вершины;
11. Проверить начальную вершину;
12. Изъять из ОТКРЫТ все вершины, имеющие неразрешимые предшествующие вершины;
13. Перейти к (2);
14. Поместить дочерние в ОТКРЫТ, вычислить оценки для них и для предшествующих вершин;
15. Перейти к (2).


8
8
7
3
3
3
2

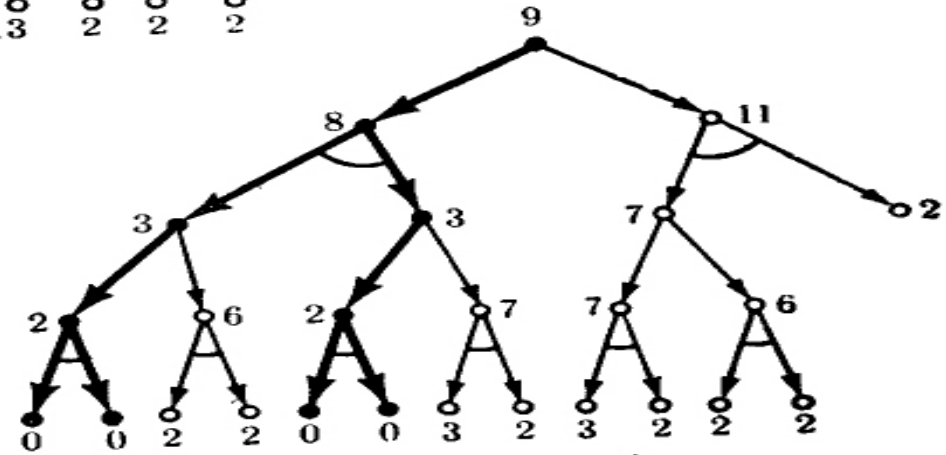
а


9
8
11
3
3
7
2
7
6
3
2
2
2

б


9
8
11
3
3
7
2
2
6
7
6
0
0
2
2
3
2
2
2

в


9
8
11
3
3
7
2
2
6
2
7
7
6
0
0
2
2
0
0
3
2
3
2
2
2

Стратегия выбора вершин

Пусть есть дерево – кандидат на решение, есть набор концевых вершин.

Выбираем ту, которая с наибольшей вероятностью ухудшит оценку текущего дерева. Если кандидат – часть оптимального дерева, то порядок раскрытия не важен. Если же ошибочен – тогда лучше как можно раньше переключиться на другое дерево-кандидат. (Размышления «интуитивно правдоподобны» - Нильсон).

Задачи удовлетворения ограничений

(Constraint Satisfaction Problem – CSP):

- Множество переменных (x1,x2,…,xN) с областями определений (D1,D2,…,DN);
- Множество ограничений {C1,C2,…,Cm}, каждое из которых задает на некотором подмножестве X допустимые комбинации;
- Задача – назначить каждой переменной значение так, чтобы не нарушить ограничения. Возможно, с использованием целевой функции.

Примеры задач

- Судоку – задача о расстановке цифр 1..9 в квадратной области с учетом ограничений;
- Раскраска графа с заданным количеством цветов – раскрасить вершины графа цветами из заданного фиксированного набора (например, заданы 4 цвета) так, чтобы ни одно из ребер графа не соединяло вершины одинакового цвета.

Криптоарифметические задачи

Ребус 1

```
  один
+ один
-------
  идва
```

Ребус 2

```
  один
+ один
  один
-------
  итри
```

Ребус 3

```
  один
  один
+ один
  один
-------
  четыре
  ч = 0
```

Ребус 4

```
  один
  один
+ один
  один
  один
-------
  пять
```

Ребус 4

```
  дома
  дома
+ дома
  дома
  дома
-------
  улицы
```

Ребус 5

```
  улицы
  улицы
+ улицы
  улицы
-------
  город
```

Ребус 6

```
  город
  город
+ город
  город
  город
  город
  город
-------
  страна
  д = а
```

Crypt arithmetic puzzle: SEND + MORE = MONEY, OLD + OLD + OLD = GOOD, EAT + THAT = APPLE

Задачи из книги Ю.А. Аленкова «650 головоломок и задач на сообразительность»

Присваивания

- Состояние определяется присваиванием значений набору переменных (не обязательно всех);
- Совместное присваивание – не нарушаются ограничения;
- Полное присваивание – используются все переменные;
- Решение – полное совместное присваивание.

Особенности

- Возможен высокий коэффициент ветвления, что делает задачи неудобными для стандартных методов поиска – криптоарифметические задачи;
- Зачастую допускают дополнительные ограничения, упрощающие поиск решения;
- Обычно путь к решению не важен;
- Поиск с возвратами – по сути, не создается дерево, а работаем с текущим состоянием.

Эвристики

- Minimum Remaining Values – наименьшее число оставшихся значений – выбор вершины наиболее «опасной». *Например, в судоку выбираем для следующего «шага» клетку, у которой при текущей конфигурации наименьшее число возможных цифр*;
- Степенная эвристика – выбор вершины, участвующей в наибольшем числе ограничений (степень вершины). *Для задачи о раскраске – выбираем сначала вершины, имеющие наибольшее количество дуг*;
- Наименее ограничительное значение – выбираем значение, не накладывающее ограничений на другие переменные. *В задаче о раскраске если все вершины, связанные с данной, то раскрашиваем и эту*;

Дополнительно вводим предварительную проверку (forward checking) – после присваивания удалить из областей определения переменных недопустимые значения. *В судоку после установки значения в некоторую клетку для клеток, находящихся в том же стобце, строке и квадрате, вычеркиваем из множеств допустимых значений это значение.*

Распространение ограничения

Constraint propagation – распространение на другие переменные результата применения некоторого ограничения к одной переменной. После того, как назначили значение некоторой переменной, исследуем все множество ограничений (не только те, в которых участвует эта переменная), и выполняем анализ. По сути, проверка на то, что множество оставшихся переменных (с усеченными на данный момент областями ограничений) и множество ограничений все еще составляют совместную систему.

Более сильная техника, чем предварительная проверка, весьма эффективна при условии, что проверка не требует больших затрат. Можно использовать графовое представление. Подробнее см. [3], стр. 209-239.

Поиск в условиях противодействия

Из теории игр – игры двух игроков с нулевой суммой и полной информацией.

«Ходы» делаются поочередно, есть полная информация о текущем состоянии, можно предсказать результат хода как одного, так и другого игрока. Для каждого заключительного состояния можно вычислить функцию результата. Набор ходов «прозрачен».

Игровые деревья

Можно анализ выполнять на деревьях для игр, однако велико количество вариантов:

- Крестики-нолики (tic-tac-toe) – менее 362 000 позиций (посчитать) – «ничейная смерть»;
- Шашки – то же самое (для английских шашек Chinook). Позиций – 2^40;
- Шахматы – 2^120 позиций.

Метод минимакса

- Выстраиваем дерево игры на некоторую глубину (насколько позволяют ресурсы);
- Выполняем оценку концевых вершин – «листьев» – на основе некоторых эвристик;
- Размечаем дерево снизу вверх исходя из предположения, что противник выбирает наилучшие ходы

Альфа-бета-отсечение

При рассмотрении дерева вариантов используем:

- Альфа – значение наилучшего варианта, который был найден для Игрока 1. «*Если у нас есть вариант лучше, то этот отбрасываем*»
- Бета – значение наилучшего варианта для Игрока 2. «*Если у противника есть вариант лучше, то этот отбрасываем – противник не сделает этот ход*».
- Если оценка узла ниже текущей α для нас, либо ниже текущей β для противника, то узел далее не раскрывается. Иначе пересчитываем значения α и β.

Литература

1. Люгер Дж. Ф. Искусственный интеллект. Стратегии и методы решений сложных проблем, 4-е изд., – М.: «Вильямс», 2003.
2. Нильсон Н. Искусственный интеллект. Методы поиска решений – М.: «Мир», 1973;
3. Рассел С., Норвиг П. Искусственный интеллект. Современный подход, 2-е изд., – М.: «Вильямс», 2006.
4. Хант Э. Искусственный интеллект. – М.: «Мир», 1978;