Системы искусственного интеллекта

02.03.03 - Математическое обеспечение и администрирование информационных систем, направленность (профиль) - разработка и администрирование информационных систем
09.03.03 - Прикладная информатика, направленность (профиль) - прикладная информатика в экономике

http//vikchas.ru

Тема 1.Введение в искусственный интеллект и машинное обучение

Лекция 4 «Решение задач посредством поиска»

Часовских Виктор Петрович
д-р техн. наук, профессор кафедры ШИиКМ

ФГБОУ ВО «Уральский государственный экономический университет»

Екатеринбург 2023

Рассмотрим, каким образом агент может найти последовательность действий, позволяющую достичь его целей.

Когда правильное действие, которое следует предпринять, не вполне очевидно, агенту может потребоваться разработать план наперед: определить такую последовательность действий, которая обеспечит достижение целевого состояния.

Такой агент называется **агентом**, решающим задачи, а вычислительный процесс, который он при этом выполняет, называется **поиском.**

Будем использовать понятия об асимптотической сложности, т.е. о системе обозначений O(n) и NP-полноте из области анализа алгоритмов.

Анализ алгоритмов и нотация O( ) позволяют рассуждать об эффективности конкретного алгоритма.

Однако эти методы не позволяют определить, может ли существовать лучший алгоритм для рассматриваемой задачи.

В области **анализа** сложности исследуются задачи, а не алгоритмы.

Первая широкая градация в этой области проводится между задачами, которые могут быть решены за время, измеряемое полиномиальным (*время работы не слишком сильно зависит от размера входных данных*) соотношением, и задачами, которые не могут быть решены за время, измеряемое полиномиальным соотношением, независимо от того, какой алгоритм для этого используется.

Класс полиномиальных задач - т.е. задач, которые могут быть решены за время $O(n^k)$ для некоторого k, - обозначается как P.

Эти задачи иногда называют "простыми", поскольку данный класс содержит задачи, имеющие такую продолжительность выполнения, как O(log n) и задачи с затратами времени O(n). Но он содержит и задачи с затратами времени $O(n^{1000})$, поэтому определение "простая" не следует понимать слишком буквально.

Другим важным классом является **NP** (Nondetenninistic Polynomial) – класс недетерминированных (*неопределённый или вероятностный детерминированный алгоритм*) полиномиальных задач.

Задача относится к этому классу, если существует алгоритм, позволяющий выдвинуть гипотезу о возможном решении, а затем проверить правильность этой гипотезы с помощью полиномиальных затрат времени.

Идея такого подхода состоит в том, что если бы можно было воспользоваться сколь угодно большим количеством процессоров, чтобы проверить одновременно все гипотезы, или оказаться крайне удачливым и всегда с первого раза находить правильную гипотезу, то NP-трудные задачи стали бы P-трудными задачами.

Одним из самых важных нерешенных вопросов в компьютерных науках является то, будет ли класс NP эквивалентным классу P, если нельзя воспользоваться бесконечным количеством процессоров или способностью находить правильную гипотезу с первого раза.

Большинство специалистов в области компьютерных наук согласны с тем, что $P \neq NP$, иными словами, что NP-являются изначально трудными и для них не существует алгоритмов с полиномиальными затратами времени. Но это утверждение так и не было доказано.

Агенты, решающие задачи

Наиболее известные примеры решения задач подразделяются на два типа - ***стандартизированные*** и ***реальные*** задачи.

Стандартизированная задача предназначена для иллюстрации или проверки различных методов решения задач.

Ей может быть дано краткое, точное описание, а это означает, что такая задача может использоваться разными исследователями для сравнения производительности алгоритмов.

Реальной задачей, такой как проблема ориентации робота, называется задача, решение которой действительно требуется людям.

Формулировка таких задач, как правило, строго индивидуальна и не стандартизирована, например потому, каждый робот имеет свой набор датчиков, которые предоставляют ему уникальный набор данных.

Стандартизированные задачи

Среда задач **клеточного мира** (grid world) представляет собой двухмерный прямоугольный массив из квадратных ячеек, в котором агенты могут перемещаться из клетки в клетку.

Как правило, агент может перейти в любую соседнюю клетку, если на его пути нет препятствий, - по горизонтали, по вертикали, а в некоторых задачах и по диагонали.

Клетки могут содержать объекты, которые агент может подбирать, толкать или воздействовать на них иным образом.

Стена или другое непреодолимое препятствие в клетке препятствует проникновению в нее агента. Мир пылесоса может быть определен как задача клеточного мира следующим образом.

Состоянии. Состояние мира определяет, какие объекты в нем присутствуют и в каких клетках они находятся.

Для мира пылесоса объектами являются агент и любой мусор.

В самом простом варианте с двумя клетками агент может находиться в любой из них, при этом каждая из клеток может содержать мусор или нет.

Исходя из этого можно определить, что в этом мире возможны всего 2 x 2 x 2 = 8 состояний.

В общем случае мир пылесоса с n клетками будет иметь $n * 2^n$ состояний.

Граф пространства состояний для мира пылесоса с двумя клетками. В нем имеется 8 состояний и по три действия для каждого состояния: L = Left, R = Right, S = Suck (*уборка*)

Начальное состояние. В качестве начального состояния может быть назначено любое состояние мира.

Действие. В мире пылесоса с двумя клетками были определены три возможных действия: убрать мусор *(Suck)*, переместиться влево *(Left)* и переместиться вправо *(Right)*.

В произвольном двухмерном многоклеточном мире агенту потребуется больше движений.

Можно было бы добавить действия переместиться вверх (*Upward)* и переместиться вниз *(Downward)*, что дало бы агенту четыре абсолютных действия перемещения, или же можно было бы перейти к эгоцентричным действиям, определяемым относительно направления взгляда агента, например двигаться вперед *(Forward)*, назад *(Backward)*, повернуть направо *(TurnRight)* и повернуть налево *(TurnLeft)*.

Функция определении преемника. Действие *Suck* удаляет мусор в клетке, где находится агент. Действие *Forward* вызывает перемещение агента на одну клетку вперед, т.е. в том направлении, куда он направлен, если только он не упрется в стену - в этом случае действие не имеет никакого эффекта.

Действие *Backward* вызывает перемещение агента в противоположном направлении, в то время как действия *TurnRight* и *TurnLeft* изменяют направление на 90° вправо и влево соответственно.

Целевое состояние. Состояние, в котором мусор отсутствует во всех клетках мира.

Функция стоимости. Каждое действие стоит 1 балл.

Головоломка Сокобан

Другим типом клеточного мира является Головоломка ***Сокобан,*** в которой цель агента состоит в том, чтобы поместить фишки, изначально произвольно размещенные в клетках мира, на отведенные им места. В каждой клетке может быть не более одной фишки. Когда агент перемещается вперед в клетку с фишкой, сразу за которой находится пустое место, то вперед перемещаются и агент, и фишка.

Агент не может переместить фишку в клетку, где уже находится другая фишка, или же вытолкнуть ее за стену. В таком мире, имеющем n свободных клеток и b фишек, будет n * n!/(b!(n - b)!) состояний; например, в мире с 8 x8 клетками и 12 фишками будет более 200 000 000 000 000 состояний!

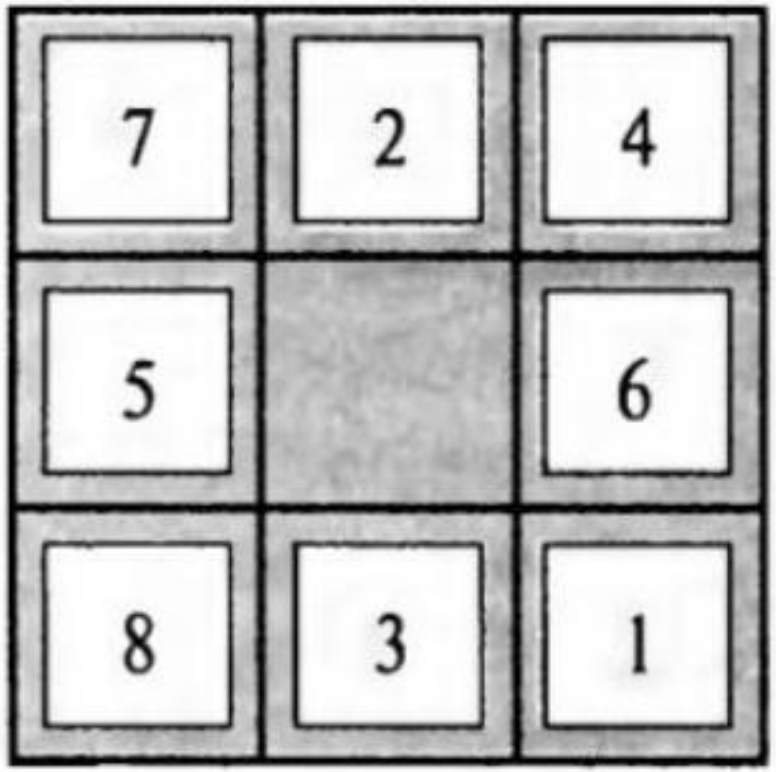
Головоломках с перемещением плиток, количество плиток в коробке может быть разным, но всегда есть одно или более пустых мест, что позволяет перемещать плитки на пустое пространство.

Одним из известных вариантов является головоломка "Час пик", в которой автомобили и грузовики перемещаются по доске размером 6 x 6 ячеек с целью освободить заданный автомобиль из пробки.

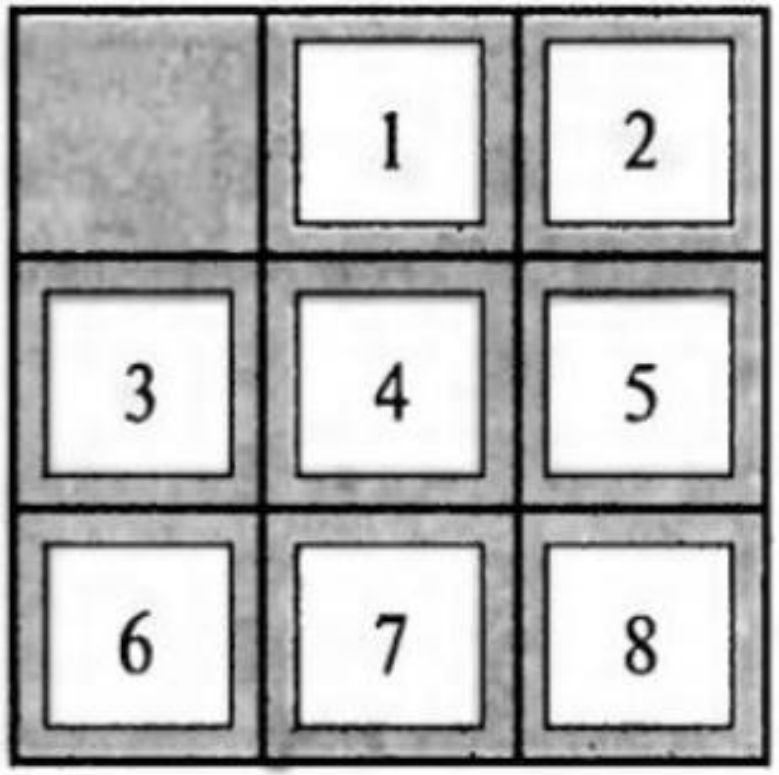
Возможно, наиболее известным вариантом такого мира является головоломка ***Восемь***, состоящая из коробки 3 x 3 квадрата с восемью пронумерованными плитками и одним пустым пространством, а также подобная ей головоломка Пятнадцать с коробкой 4 x4 квадрата и с 15-тью плитками.

Цель в каждом случае состоит в том, чтобы достичь определенного целевого состояния, например такого, как показано на рисунке, справа.

Задача игры в восемь



Начальное состояние



Целевое состояние

Стандартная формулировка этой задачи следующая:

Состояния. Описание состояния определяет местонахождение каждой из восьми плиток и пустого участка на игральной доске из девяти квадратов.

Начальное состояние. В качестве начального может быть определено любое состояние. Обратите внимание, что в этом мире свойство четности делит пространство состояний на две части: любая заданная цель может быть достигнута точно из половины возможных начальных состояний.

Действия. В то время как в физическом мире головоломка состоит из плиток, перемещаемых в пределах границ головоломки, простейший способ из описания действий в соответствующем клеточном мире - представить, что перемещается пустое поле. Соответствующие этому случаю действия вполне типичны: *Left* (влево), *Right* (вправо), *Up* (вверх) и *Down* (вниз).

Если пустое поле окажется на краю сетки или в углу, не все действия будут применимы.

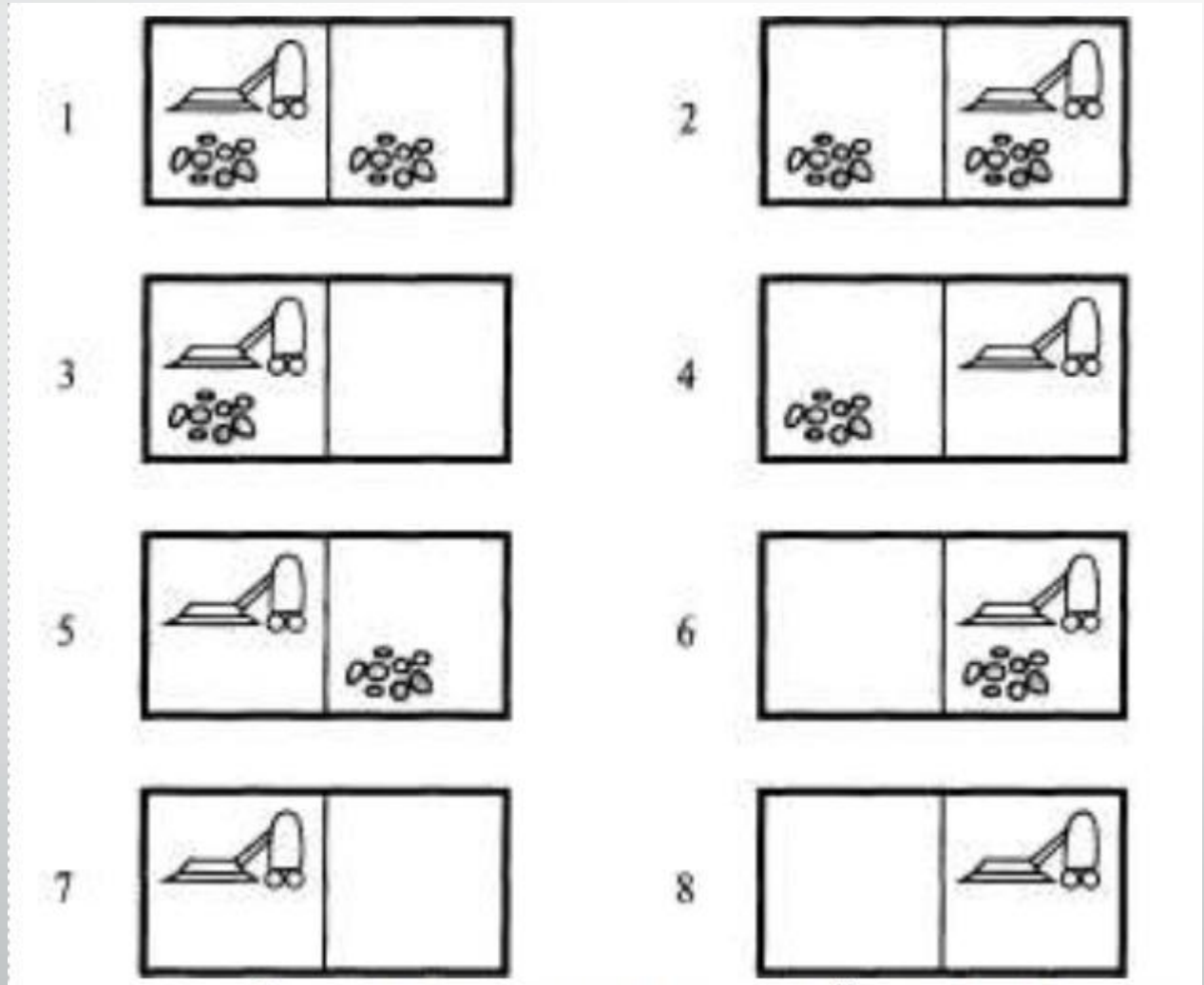
Функция определения преемника. Отображает состояния и действия в результирующие состояния. Например, если выполнить действие *Left* в начальном состоянии мира, представленном на рисунке, пустое место и плитка 5 поменяются местами.

Целевое состояние. Хотя в этом мире любое состояние может быть выбрано как целевое, в данной головоломке ее решение определено как состояние, когда все плитки расположены по возрастанию их номеров, как показано рисунке, справа.

Функция стоимости. Каждое перемещение плитки стоит 1 балл.

Решение задач -Непредсказуемый мир пылесоса

Мир пылесоса имеет восемь возможных состояний как показано на рисунке



Восемь возможных состояний мира пылесоса

В нем возможны три действия : *Right* (вправо), *Left* (влево) и *Suck* (убрать), - а целью является уборка всего мусора (состояния 7 и 8).

Если среда полностью наблюдаема, детерминирована и известна, то задача легко решается с помощью любого из алгоритмов, и решение
представляет собой четкую последовательность действий.

Например, если исходным является состояние 1, то выполнение последовательности действий *Suck*, *Right*, *Suck* приведет мир в целевое состояние 8.

Теперь предположим, что пылесос в этом мире стал более мощным, но при этом иногда может вести себя неожиданным образом, создавая непредвиденные ситуации, а значит, этот мир стал ***недетерминированным***

В непредсказуемом мире пылесоса действие *Suck* работает следующим образом.

- При выполнении в квадрате с мусором это действие убирает мусор в данном квадрате, а иногда еще и в соседнем.
- При выполнении в чистом квадрате после этого действия на ковре иногда остается немного мусора.

Для того чтобы предоставить более точную формулировку этой задачи, необходимо обобщить понятие функции определении преемника. Вместо определения этого понятия как функции RESULT, всегда возвращающей одно результирующее состояние, мы теперь определим его как функцию RESULTS, возвращающую множество возможных результирующих состояний.

Например, в мире непредсказуемого пылесоса действие Suck в состоянии 1 может убрать мусор либо только в текущем местоположении, либо сразу в обоих местоположениях:

$$RESULTS(l, Suck) = \{5, 7\}$$

Таким образом, если исходным является состояние 1, нет единственной фиксированной последовательности действий, позволяющей решить задачу, но следующий условный план позволяет это сделать

[Suck, **if** State = 5 **then** [Right, Suck] **else** [ ]]

Отсюда видно, что условный план может содержать этапы **if-then-else**; а это означает, что решения являются деревьями, а не последовательностями.

В этом случае условие в операторе **if** предназначено для проверки, каким является текущее состояние, оно должно представлять собой что-то, что агент может наблюдать во время выполнения плана, но еще не знал при его составлении.

В качестве альтернативы можно предложить формулировку, использующую проверку восприятия агента, а не состояния.

Многие задачи в реальном, физическом мире являются задачами, предполагающими непредвиденные ситуации, поскольку точное предсказание будущего просто невозможно. Именно поэтому многие люди даже во время прогулки внимательно следят за происходящим вокруг.

Деревья поиска И-ИЛИ

Для поиска условного решения недетерминированных задач создадим деревьев поиска, но в этом случае, в отличие от детерминированных задач, деревья имеют иной характер.

В детерминированной среде ветвление происходит только в результате собственного выбора агента в каждом состоянии: "Я могу выполнить это ***или*** то действие".

Назовем такие узлы **ИЛИ-узлами**.

Например, в мире пылесоса агент в ИЛИ-узле выбирает между Left или Right, или Suck.

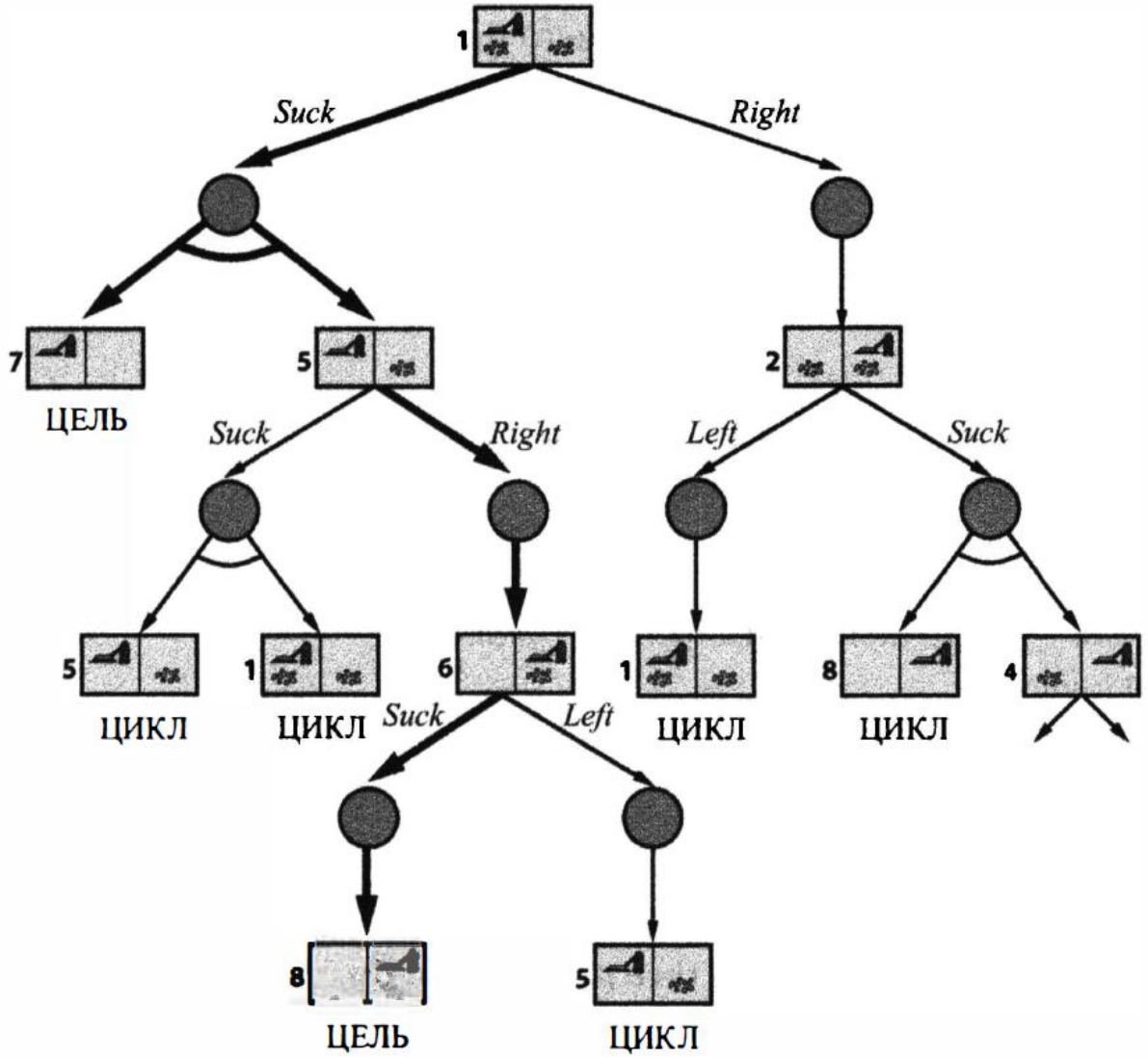
В недетерминированной среде ветвление также возникает в результате выбора, но теперь уже выбора результата каждого действия ***окружающей средой***.

Назовем такие узлы **И-узлами**.

Например, выполнение действия *Suck* в состоянии 1 дает результат в виде доверительного состояния {5, 7 } , поэтому агент должен сформировать условный план для состояний *5 и 1*.

Эти два вида узлов чередуются, приводя к образованию **И-ИЛИ-дерева**, как показано на рисунке.

Первые два уровня дерева поиска для непредсказуемого мира пылесоса.



Узлы состояния являются ИЛИ-узлами, в которых агент должен выбрать какое-то действие.

В И-узлах, показанных на рисунке в виде кружков, каждый возможный результат действия должен быть обработан, что подчеркивается дугами, соединяющими исходящие их этих узлов ветви.

Пути к решению показаны **утолщенными** линиями.

Решением задачи поиска в дереве И-ИЛИ является такое поддерево в полном дереве поиска, в котором

1) каждый лист является ***целевым*** узлом;

2) Выбирается ***одно*** действие в каждом из его ИЛИ-узлов; и

3) включена каждая исходящая ветвь во всех его И-узлах.

Решение, представленное на рисунке **утолщенными** линиями, Соответствует, ранее показанному плану

[Suck, **if** State = 5 **then** [Right, Suck] **else** [ ]]

Рассмотрим рекурсивный алгоритм поиска в глубину, реализующий поиск по графу И-ИЛИ, генерируемых в недетерминированных средах.

Решение представляет собой условный план, учитывающий каждый недетерминированный результат и включающий план для каждого из них.

function AND-OR-SEARCH(proЬ/em) **returns** условный план *или failure*
return *OR-SEARCH (proЬlem, proЬlem.INITIAL,* [])

function OR-SEARCH(proЬlem, *state, path)* **returns** условный план *или failure*
If *proЬlem.Is.GoAL(state)* **then return** пустой план
if Is-CYCLE(path) **then return** *failure*
for each *action* ***in*** *proЬlem.ACTION S(state)* **do**
plan ← And- SEARCH(proЬ/em, *RESVLTS (state, action), [state] + path])*
If *plan* $\neq$ *failure* **then return** *[action]* **+** *plan]*
return *failure*

function AND- SEARCH(proЬ/em, *states, path)* **returns** условный план *или failure*
for each *s;* **in** *states* **do**
plan; **+-** OR- SEARCH(proЬlem**,** *S;, path)*
If *plan;* **=** *failure* **then return failure**
return [if s_1 **then** $plan_1$ **else if** s_2 **then** $plan_2$ **else . . . if** s_{n-1} **then** $plan_{n-1}$ **else** $plan_n$*]*

Ключевым аспектом этого алгоритма является способ, используемый для работы с циклами, часто возникающими в недетерминированных задачах (например, если действие иногда не дает никакого результата или если непреднамеренно полученный эффект может быть исправлен).

Если текущее состояние идентично некоторому состоянию на пути к нему от корня, то алгоритм возвращает ошибку. Это не означает, что для текущего состояния ***не существует*** решения; это лишь означает, что если нециклическое решение существует, оно должно быть достижимо из предыдущего воплощения текущего состояния, а значит, его новое воплощение может быть отброшено.

Благодаря этой проверке можно гарантировать, что алгоритм обязательно завершит свою работу в любом конечном пространстве состояний, поскольку каждый путь должен либо достичь цели, либо привести в тупик, либо вернуться к уже пройденному состоянию.

Обратите внимание, что алгоритм не проверяет, является ли текущее состояние повторением состояния на каком-то ином пути от корня, что очень важно для его эффективности.

Рассмотренный алгоритм, это поиск в графах И-ИЛИ, генерируемых в недетерминированных средах.

Решение представляет собой условный план, учитывающий каждый недетерминированный результат и включающий план для каждого из них.

Для поиска в графах И-ИЛИ могут также использоваться алгоритмы поиска в ширину и поиска по первому наилучшему совпадению.

Отметим, что существует аналог приведенного алгоритма поиска, позволяющий находить **оптимальные** решения.

Благодарю за внимание!