Изкуствен интелект

**Лекция 1, част 2: Интелигентни агенти**

**Агент -** нещо, което възприема средата (чрез рецептори или сензори) и въздейства върху нея (чрез ефектори). В най- широк смисъл този термин обхваща хора, роботи и програми.

**Интелигентен** (разумен, рационален) агент: агент с целесъобразно (според определени критерии) поведение, изграждано въз основа на възприетите до текущия момент сведения, вградените знания на агента и възможностите му да действа.

**Изображение** (mapping): съответствие между възприятията и очакваните в отговор действия, чрез което се определя поведението на агента. Може да е реализирано чрез таблица (която в много случаи би трябвало да е безкрайна), функция или стратегия за самообучение.

**Основни характеристики на интелигентните агенти**

* **автономност**, свойството на агента да определя поведението си въз основа на натрупания опит (за сметка на вградените знания) и да действа по своя инициатива
* **адаптируемост**: способност на агента
  + да се самообучава в действие
  + да реагира на изменения в средата и да се приспособява към тези от тях, които са трайни
  + да общува с други агенти чрез някаква знакова система
* **сътрудничество**, работа в многоагентни системи, включваща разделяне на задачите, обединение на резултатите и обмен на знания и опит
* **мобилност**: самонасочено придвижване на агент от един носител към друг в мрежова среда

**Оценка на ефективността на поведението** **(performance measure):** формулировката й е съществен елемент на началната спецификация при проектирането на даден интелигентен агент.

**PEAS**: Performance measure, Environment, Actuators, Sensors (елементи на спецификацията на всеки конкретен интелигентен агент)

**Строеж на интелигентните агенти (агент = архитектура + програма)**

* **програма на агента**: функция, реализираща изображението от възприятия към действия. В общия случай програмата разполага с памет, в която съхранява част от историята на получените възприятия и предприетите действия, за да може отговорът на всяко въздействие да не бъде функция само на това въздействие. Тялото на програмата представлява цикъл, всяка стъпка на който включва възприятие (отразявано в паметта), избиране и осъществяване на действие (също отразявано в паметта). Задачата на изкуствения интелект е именно създаването на такива програми.
* **архитектура**: това, върху което работи програмата - обикновен компютър или специализирано устройство. Може да включва платформено независимо програмно осигуряване, с помощта на което агентът да бъде програмиран на високо ниво.  
  За реализацията на програмата на агента могат да се използват различни, често нетрадиционни техники: разсъждения, основани на знания (най-често правила), статистически анализ, размита логика, невронни мрежи, еволюционно (генетично) програмиране и др.

**Типове агентни програми**

Могат да бъдат обособени четири основни типа агентни програми, подредени по-долу по степента им на общност:

* **прости рефлекторни агенти**: действията се определят от правила, от които агентът избира това, което съответства най- точно на ситуацията. Програмите могат да бъдат много ефективни, но приложимостта им е малка.
* **рефлекторни агенти, следящи света** (основани на модел на света): в паметта се поддържа информация за състоянието на света, независимото му развитие и ефектите от въздействието на агента върху него.
* **агенти, основани на цели**: действията се избират въз основа на това, което се знае за очакваните им резултати, и целите на агента. Методите са търсене и планиране. Поведението им е по- малко ефективно, отколкото на рефлекторните агенти, но е по- гъвкаво.
* **агенти, основани на полезност**: състоянията на света могат да се сравняват по произволно множество критерии. Полезността е функция на състоянието, отразяваща степента на задоволеност на агента. Включва разумно разрешаване на ситуации, които са проблематични за целевите агенти: конфликтни цели, всяка от които пречи на постигането на останалите, и несигурни цели, различни по важност и вероятност.

Важна характеристика на широк клас интелигентни агенти е способността им да се самообучават:

**Среди**

**Характеристики на средите, в които действат агентите:**

* **(не)достъпност**: агентът (не) може да възприема цялостно състоянието на средата по всяко време;
* **(не)детерминираност**: агентът (не) може точно да определи следващото състояние на средата въз основа на текущото й състояние и своите действия;
* **(не)епизодичност**: историята (не) е разделена на епизоди, във всеки от които агентът получава възприятие и след това предприема действие, като ставащото във всеки епизод не зависи от станалото в предишните;
* **статичност/динамичност**: средата (не) може да се промени, докато агентът планира действията си. Ако средата не се променя с времето, но се променя оценката за ефективността на агента, средата е полудинамична;
* **дискретност/непрекъснатост**: има/няма ограничен брой ясно определени и разграничени възприятия и действия.

**Моделиране на средата**: средата може да се моделира чрез симулатор, който предава на агентите техните възприятия, получава от тях действията им, обновява средата и евентуално пресмята оценки за ефективността на агентите. Желателно е симулаторът да може да работи с различни среди от определена фамилия.

**Лекция 2: Търсене в пространството на състоянията**

**Пространство на състоянията – основни понятия и задачи**

**Обща постановка**. Решаването на много задачи, традиционно смятани за интелектуални, може да бъде сведено до последователно преминаване от едно описание (формулировка) на задачата към друго, еквивалентно на първото или по-просто от него, докато се стигне до това, което се смята за решение на задачата.

Примери: задачи от областта на т. нар. интелектуални игри, аналитични преобразования на алгебрични и тригонометрични изрази, решаване на уравнения и т.н.

**Основни дефиниции**

**Състояние**: едно описание (формулировка) на задачата в процеса на нейното решаване.

**Видове състояния**: начално, междинни, крайни (целеви).

**Оператор**: начин (правило, алгоритъм), по който от дадено състояние се получава друго.

**Пространство на състоянията (ПС)**: съвкупността от всички възможни състояния, които могат да се получат от дадено начално състояние.

**Представяне на ПС**: чрез ориентиран граф (граф на състоянията, ГС) с възли – състоянията и дъги – операторите. Когато ПС може да се представи във вид на дърво, се говори за т. нар. дърво на състоянията (ДС).

**Действия, свързани с ПС**

**• Генериране на състояния**

- генериране на следващ наследник  
- генериране на всички наследници

**• Оценяване на състояние**

- двоични оценки (true/false)  
- числови оценки в определен интервал (оценката на целта съвпада с единия край на интервала)

**Основни типове задачи, свързани с ПС**

• генериране на ПС  
• решаване на задачи (търсене) върху генерирано ПС  
• комбинирано генериране и търсене в ПС

Стратегиите за решаване на тези задачи са сходни и затова обикновено се говори само за търсене (а се подразбира и/или генериране).

**Основни типове задачи за търсене в ПС**

**1. Търсене на път до (определена) цел** – търси се път от дадено начално състояние до определено целево състояние (целевото състояние е описано явно или може да бъде разпознато). Пътят може да се търси под формата на списък от възли или списък от дъги в ГС.

Варианти: търсене на минимален (най-къс или най-икономичен) път до цел и др.  
Примери: търсене на път върху географска карта, задача за търговския пътник и др.

**2. Търсене на печеливша стратегия** (при игри за двама играчи).

Примери: шахмат, шашки, “кръстчета и нулички” и др.

**3. Търсене на цел при спазване на ограничителни условия** (задачи за удовлетворяване на ограничения).  
Примери: задача за осемте царици, решаване на криптограми, съставяне на разписания и др.

**Задачи за удовлетворяване на ограничения** (Constraint Satisfaction Problems, CSP)

**Формулировка**

Дадени са:

* **Множество от променливи** v1, v2, … , vn със съответната за всяка променлива **област от допустими стойности Dvi  (домейн).**

Променливите са:

* + - дискретни
      * крайни
      * изброими безкрайни
    - непрекъснати
* **Множество от ограничения** (допустими/недопустими комбинации от стойности на променливите)

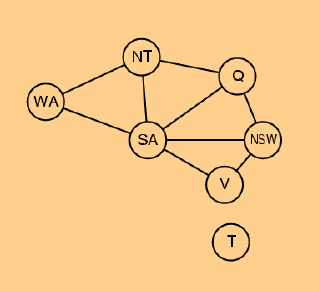
- **унарни** – когато засягат 1 променлива  
 - **бинарни** - когато засягат 2 променливи  
 - **от по-висок ред** – когато засягат повече от 2 променливи  
 - **преференции/предпочитания** (soft constraints) – с препоръчващ характер, определят критерии за избор между няколко решения (дефинират цената на някои свързвания на променливи)

**Целево състояние** (състояния): множество от свързвания със стойности на променливите {v1=c1, v2=c2, … , vn=cn}, които удовлетворяват всички ограничения.

**Представяне на ограниченията**

Бинарни ограничения: за представянето им се използва граф

• възли – променливи (X, Y, … )  
• дъги – ограничения  
На всяко ограничение p(X,Y) съответстват по две насочени дъги в графа: <X,Y> и <Y,X>  
• Дъгата <X,Y> е съвместима, ако за всяка стойност на X от DX съществува стойност на Y от DY, удовлетворяваща ограничението p(X,Y)  
• Редуциране: ако дъгата <X,Y> не е съвместима, тогава всички стойности от DX, за които няма съответна стойност от DY, могат да бъдат изтрити от DX и това би направило дъгата <X,Y> съвместима

Примерен граф на ограниченията (constraint graph):  


**Примерни задачи от разглеждания тип**

• Задача за осемте (n-те) царици, криптоаритметика, оцветяване на географска карта, пъзели, кръстословици, судоку и др.  
• планиране, проектиране, съставяне на разписания и др.

Пример: оцветяване на географска карта (Австралия). Целта е всяка област (държава) върху картата да бъде оцветена с подходящ цвят по такъв начин, че да няма две съседни (граничещи) области, оцветени с един и същ цвят.

• Променливи: WA, NT, Q, NSW, V, SA, T  
• Области на допустимите стойности: Di = {red,green,blue}  
• Ограничения: съседните области трябва да бъдат оцветени с различни цветове, например WA ≠ NT или (WA,NT) ∈ {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

**Алгоритми за решаване на задачи за търсене на цел при спазване на ограничителни условия**

* **генериране и тестване (generate and test)**
  + Генерира се множеството D = DV1 X DV2 X … X DVn
  + Тества се всяко свързване на променливите с елемент на D за това, дали удовлетворява ограниченията
* **търсене с възврат (backtracking)**
  + Свързване на променливите със стойности в определен ред
  + След свързването на всяка поредна променлива със стойност се извършва проверка дали свързаните до момента променливи удовлетворяват ограниченията:
    - ако ограниченията се удовлетворяват, се продължава с избора на стойност за следващата несвързана променлива
    - ако ограниченията не се удовлетворяват, се избира нова стойност за последната свързана променлива

**Подобряване на ефективността на търсенето с възврат (backtracking)**

Използването на общи методи (какъвто е текущо разглежданият) поставя редица въпроси, свързани с ефективността на търсенето:

* коя променлива трябва да бъде свързана най-напред?
* в какъв ред трябва да бъдат пробвани различните допустими стойности на избраната променлива?
* може ли да се прецени предварително дали дадено свързване ще доведе до неуспех?
  + Евристика: избор на най-ограничената променлива - Най-напред се избира за свързване онази променлива, която има най-малък брой допустими стойности
* разпространяване на ограниченията (forward checking)
  + Следи се за оставащите допустими стойности за несвързаните променливи. Търсенето се прекратява, ако на дадена стъпка се окаже, че няма допустими стойности за някоя променлива.
* разпространяване на ограниченията (constraint propagation)
  + Forward checking разпространява информация (ограничения) от свързаните към несвързаните променливи, но не открива всички случаи, които ще доведат до неуспех:
  + NT и SA не могат едновременно да имат стойност blue.
  + Constraint propagation е общото понятие за разпространяване на ограничения към от една променлива към други.
  + Constraint propagation изисква многократно засилване на ограниченията на локално равнище.

**Съвместимост на дъгите**

Проверката за съвместимост и осигуряването на съвместимост на дъгите чрез редукция е бърз метод за разпространяване на ограниченията (при задачи с бинарни ограничения), който е по-силен от forward checking.

**Лекция 3: Търсене на път до определена цел**

**Примерни задачи от реалния свят:** намиране на път (при routing в компютърни мрежи, при планиране на пътувания с автомобилен/самолетен транспорт и др.), задача за търговския пътник, навигация на роботи, задачи за планиране и конструиране (асемблиране) на сложни обекти и др.

**Характеристики на алгоритмите за търсене:**

* пълнота
* оптималност
* сложност
  + по време « брой изследвани възли
  + по памет « максимален размер на фронта

Необходимост от избягване на повторения и зацикляне.

**Параметри на търсенето:**

* дълбочина на “най-плитката” цел – d (или максимална “дълбочина” на пространството/графа на състоянията - m)
* коефициент на разклонение (разклоненост) на ГС - b

**Методи за неинформирано ("сляпо") търсене**

**Обща характеристика**: пълно изчерпване по твърда (фиксирана отнапред) стратегия. Прилагат се, когато липсва специфична информация за предметната област и оценяващата функция може само да провери дали дадено състояние е целево или не е.

**Програмна реализация**: чрез използване на работна памет (списък Open/frontier/fringe на т. нар. открити възли или списък от натрупани/изминати пътища, започващи от началния възел - нарича се още фронт на търсенето).

**Общ алгоритъм за неинформирано търсене**

Тръгва се от началния възел, като на всяка стъпка фронтът на търсенето се разширява в посока към неизследваните възли, докато се достигне до целеви възел.

Начинът, по който се разширява фронтът, както и правилата за избор на конкретен елемент на фронта, от който ще продължи неговото разширяване, определят стратегията на търсене (search strategy).

**Неинформирано търсене в дърво**

**Реализация** - представяне на възлите от графа (дървото) на състоянията

В следващите дефиниции на функции за търсене (написани на псевдокод) ще предполагаме, че всеки възел от ГС е структура от данни с 5 компонента:

STATE: състоянието от пространството, на което съответства разглежданият възел

PARENT: възелът от ГС, като наследник на който е генериран разглежданият възел

ACTION: операторът, с помощта на който е бил генериран разглежданият възел

PATH-COST: цената (традиционно означавана с g(n)) на пътя от началното състояние до състоянието, съответно на разглеждания възел (в съответствие с указателите към родителските възли)

DEPTH: броят на стъпките, от които е съставен пътят от началното състояние до състоянието, съответно на разглеждания възел

**Неинформирано търсене в граф**

**Стратегии за неинформирано търсене**

1. **Изчерпване (търсене) в дълбочина (depth first search - DFS) -** добавяне на новите възли (новия възел) в началото на списъка Open/fringe.

fringe = стек (LIFO, put successors at front)

Търсенето е евтино (линейно по отношение на необходимата памет), но не е нито пълно, нито оптимално (пълно е, когато графът на състоянията е краен или поне не съществуват безкрайно дълги ациклични пътища в него).

**Построяване на пътя като списък от последователни състояния**

• Фронтът на търсенето е списък от пътища (отначало е списък от един път, който включва само началното състояние).

• При търсенето в дълбочина фронтът се обработва като стек.

• Ако фронтът има вида [р1, р2, ... , рп], то:

- Избира се р1. Ако р1 е довел до целта, край.  
- Пътищата р11, р12, … , р1к, които разширяват р1, се добавят в началото на фронта (преди р2), т.е. фронтът придобива вида [р11 р12, … , р1к. р2, ... , рn]. Преминава се към следващата стъпка от цикъла.

Вариант 1: **търсене в дълбочина до определено ниво** (depth bound search, depth limited search).

Вариант 2: **итеративно търсене по нива** (iterative deepening search).

1. **Изчерпване (търсене) в широчина (breadth-first search) -** добавяне на новите възли в края на списъка Open/fringe.

fringe = опашка (FIFO, put successors at end)

Търсенето е пълно (когато всеки възел от ГС има краен брой наследници) и оптимално, но е скъпо (експоненциално по отношение на необходимата памет).

**Построяване на пътя като списък от последователни състояния**

* Фронтът на търсенето е списък от пътища (отначало е списък от един път, който включва само началното състояние).
* При търсенето в широчина фронтът се обработва като опашка.
* Ако фронтът има вида [р1, р2, ... , рп], то:
  + Избира се р1. Ако р1 е довел до целта, край. о Пътищата р11, р12, … , р1 к, които разширяват р1, се добавят в края на фронта (след рп), т.е. фронтът придобива вида [р2, ... , рn, р 11, р12, … , р1к]- Преминава се към следващата стъпка от цикъла.

Вариант: търсене с равномерна цена на пътя (uniform-cost search) - сортиране на списъка Open/fringe според цената на изминатия път.

**Оценка на сложността**. Пространствената сложност (сложността по памет) на търсенето в дълбочина е 0{Ьт), докато пространствената сложност на търсенето в широчина е 0{bd+1). Времевата сложност на търсенето в дълбочина е 0(bm), а времевата сложност на търсенето в широчина е 0{bd+1). При задачи, които имат много решения, търсенето в дълбочина може да бъде по-бързо от търсенето в широчина, тъй като има добър шанс да се намери решение след изследване на малка част от цялото ПС.

Търсенето в ограничена дълбочина (търсенето в дълбочина до определено ниво) има сложност, подобна на тази на търсенето в дълбочина (изисква 0{b) време и 0{Ь1) памет, където / е максималната дълбочина на изследване). Итеративното търсене по нива има времева сложност 0{bd) и пространствена сложност 0{bd), където където de максималната дълбочина на изследване. Това е най-предпочитаният метод за търсене, когато ПС е с голям размер и няма никаква информация за дълбочината на решението.

**Лекция 4: Методи за евристично търсене   
на път до определена цел**

**Обща характеристика:** реализират пълно изчерпване по гъвкава стратегия или търсене с отсичане на част от графа на състоянията. Приложими са при наличие на специфична информация за предметната област, позволяваща да се конструира оценяваща функция (**евристика**), която връща небулева оценка (числова оценка в предварително определен интервал). Тази оценка може да служи например за мярка на близостта на оценяваното състояние до целта или на необходимия ресурс за достигане от оценяваното състояние до целта.

Най-често се използва евристична оценяваща функция h, която връща като резултат приблизителната стойност на определен ресурс, необходим за достигане от оценяваното състояние до целта.

**Програмна реализация**: чрез използване на работна памет (списък Open/frontier/fringe на т. нар. открити възли или списък от натрупани/изминати пътища, започващи от началния възел - фронт на търсенето).

Примерна задача: търсене на път между две селища. Пътна карта на Румъния.

Ще използваме данните от тази карта с цел илюстриране на работата на някои алгоритми за евристично търсене при намиране на път от Arad до Bucharest.

Евристичната оценяваща функция ще връща като резултат разстоянието по права линия (straight-line distance) между текущо достигнатия град и целта (Bucharest).

**1. Метод на най-доброто спускане** (търсене на най-добър път, **best first search**) - сортиране на списъка Open/fringe в съответствие с евристичната функция (например h(node) = <оценка на цената на пътя от node до целта>).

**Построяване на пътя като списък от последователни състояния**

* Фронтът на търсенето е сортиран отношение на стойностите на евристичната функция списък от пътища (отначало е списък от един път, който включва само началното състояние).
* Ако фронтът има вида [р1; р2, ... , рп], то:
  + Избира се р-|. Ако pi е довел до целта, край.
  + Пътищата рц, pi2, ... , pik, които разширяват рь се добавят към фронта и новополученият фронт се сортира в нарастващ ред на оценките на пътищата, в резултат на което придобива вида [qi, q2, ... , qn+k-i].
  + На следващата стъпка се обработва пътят от фронта, който има най-добра евристична оценка, т.е. q-i.

**2. Търсене с ограничена широчина (търсене в лъч, beam search)** - ограничаване на списъка Open/fringe до първите п най-добри възела от него (в съответствие с евристиката). При п=1 се доближава до метода на най-бързото изкачване.

**3. Метод на най-бързото изкачване (hill climbing)** - списъкът Open/fringe се ограничава до най-добрия му елемент (в съответствие с евристиката), и то само ако той е по-добър от своя родител. Търсенето е еднопосочно, без възможност за възврат.

**Възможни проблеми:**

* **локален екстремум** - състоянието е по-добро от съседните (наследниците си), но не е най-доброто в цялото ПС;
* **плато** - съседните състояния (наследниците на текущото състояние) изглеждат също толкова добри, колкото и текущото;
* **хребет** - никой от възможните оператори не води до по- добро състояние от текущото, макар че два или повече последователни оператори биха могли да доведат до такова състояние.

**4. Търсене с минимизиране на общата цена на пътя (А\*)** - комбинира търсене с равномерна цена на пътя с търсене на най-добър път. Списъкът Open/fringe се сортира в съответствие с функцията f(node) = g(node) + h(node). Тук функцията g връща като резултат цената на изминатия път от началния възел до node, а евристичната функция h връща като резултат приблизителна стойност на цената на оставащата част от пътя от node до целта.

**Построяване на пътя като списък от последователни състояния**

* Фронтът на търсенето е сортиран по отношение на стойностите на функцията f списък от пътища (отначало е списък от един път, който включва само началното състояние).
* Ако фронтът има вида [р1, р2, ... , рn], то:
  + Избира се р,. Ако pi е довел до целта, край.
  + Пътищата рц, р12, ... , р1к, които разширяват pi, се добавят към фронта и новополученият фронт се сортира в нарастващ ред на оценките на пътищата, в резултат на което придобива вида [q1, q2, ... , qn+k-i].
  + На следващата стъпка се обработва пътят от фронта, който има най-добра оценка (най-добра стойност на функцията f), т.е. q1.

**Оценка на метода А\*:**

Стратегията е пълна, ако разклоненията (наследниците) на всеки възел от ГС са краен брой и цената на преходите е положителна, и оптимална, ако евристиката е приемлива (оптимистична), т.е. никога не надценява стойността (цената) /7\*на оставащия път (ако h\*(node) >h(node) за всеки възел node).

Времевата и пространствената сложност на метода са експоненциални.  
Примери за приемливи (оптимистични) евристики

При задачата за 8-те плъзгащи се плочки (the 8-puzzle problem):

* h1(S) = броя на плочките, чиято текуща позиция е различна от позицията им в целевото състояние
* h2(S) = тоталното (сумарното) Манхатънско разстояние

**Манхатънско разстояние** (Manhattan Distance) между точките (Xi.Yi) и (Xj.Yj): D = |Xi-Xj| + |Yi-Yj|.

**Лекция 5, част 1:  
Анализ на целите и средствата**

**General Problem Solver (GPS)**

GPS (General Problem Solver, Обща програма за решаване на задачи) е ранна програма за търсене, която има изключително влияние върху изследванията по ИИ. Тя реализира алгоритъм за търсене, който използва| функция за оценка на състоянията и функция за нареждане на операторите, за да направлява търсенето. Следователно, тя не прилага всички възможни оператори към дадено състояние, а само тези, които изглеждат перспективни.

GPS е била предназначена да моделира човешкото поведение в задачите за търсене като игрови задачи, символно интегриране и др. Целта й е била прекалено амбициозна (не всички проблеми могат да се разглеждат като задачи за търсене), но при някои предположения GPS може да се използва за генериране на планове на действия, за решаване на задачи и самообучение по аналогия и др.

GPS прилага една техника, наречена анализ на целите и средствата (Means-Ends Analysis, МЕА). Най-общо МЕА е методология за търсене в пространството на състоянията, при която на всяка стъпка се прави опит за редуциране на различията между текущото състояние и целта.

**Means-Ends Analysis (MEA)**

Идеята на MEA се състои в следното. При условие, че е дадено състоянието, което трябва да се преобразува, то се съпоставя с описанието на целевото състояние. В резултат на съпоставянето или се установява, че няма разлика между текущото и целевото състояние (и следователно текущото състояние съвпада с целевото състояние), или се намират съществуващите разлики между текущото и целевото състояние. Във втория случай GPS се опитва да намали разликите между текущото и целевото състояние.

GPS използва разликите между състоянията по два начина. Първо, броят на разликите служи като груба мярка за общия напредък към целта. Второ, разликите могат да предложат какви действия да се предприемат по-нататък.

Всъщност преброяването на разликите е само груба мярка за разстоянието до целта, защото то пренебрегва факта, че някои разлики са по-важни от други. Затова обикновено се въвеждат тегла (приоритети) на разликите, които намират отражение във формулировката на оценяващата функция.

Втората употреба на разликите в МЕА е за направляване на избора на операторите. В GPS операторите имат по 3 компонента:

* предусловия;
* трансформираща функция;
* списък на намаляваните разлики.

Първият компонент представлява описание (в същите термини, както и описанието на целевото състояние), което включва необходимите (а често и достатъчни) условия за приложимост на оператора.

Вторият компонент генерира наследниците на разглежданото състояние.

Третият компонент свързва оператора с разликите, които той редуцира (за чието намаляване той е подходящ). Обикновено този компонент се съхранява в специална структура, наречена **таблица на връзките** или **таблица оператор-разлика**. Тази таблица се предоставя от потребителя (от поставящия задачата) и позволява на GPS при дадена разлика да намери оператор за нейното намаляване.

**Алгоритъм MEA(Current,Goal)**

1. Сравняват се (съпоставят се) Current и Goal. Ако между тях няма разлики, тогава алгоритъмът прекратява работата си.

2. В противен случай се избира най-важната разлика между Current и Goal. Избраната разлика се редуцира чрез изпълнение на следните действия (докато се стигне до успех или неуспех):

(а) Избира се неизползван (неизпробван) до момента оператор О, който е приложим към избраната разлика. Ако не съществуват такива оператори, алгоритъмът завършва с неуспех.

(б) Прави се опит да се приложи О към Current. Генерират се описанията на две състояния:

O-Start (състояние, в което са изпълнени предусловията на О) и

O-Result (състояние, което се получава след прилагането на О върху O-Start).

(в) Ако

First-Part := MEA(Current,O-Start) и Last-Part := MEA(0-Result,Goal) завършат c успех, то алгоритъмът завършва с успех и връща като резултат конкатенацията на First-Part, О и Last-Part.

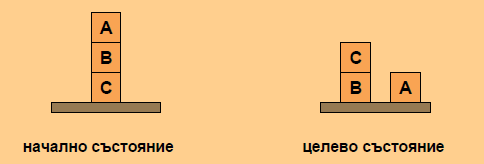
**Лекция 5, част 2:   
Планиране на действията**

**Дефиниция на задачата за планиране**

Декомпозиране на задачата на подзадачи и построяване на поредица от елементарни (в определен смисъл) стъпки, чието последователно изпълнение може да доведе до решаване на първоначално поставената задача.

Частен случай на търсене в пространството на състоянията, при което се използва логическо представяне на състоянията (в частност целите) и операторите.

Примерна предметна област: планиране на действията на робот в “света на кубчетата”.



**Ситуационно смятане**

Ситуационното смятане (Situation Calculus, McCarthy & Hayes, 1969) е подмножество на предикатното смятане от първи ред, създадено с цел формализиране на описанието на ситуациите и действията при решаване на задачи от областта на планирането.

**Основни елементи и характеристики на ситуационното смятане:**

* **Действията** и **фактите** (т.е. елементите на плана и елементите на описанието на състоянието на сцената) се представят чрез термове, например: puton(A,B), on(A,B);
* **Верните** **факти** **в** **дадена** **ситуация** се описват с помощта на предиката holds. Например holds(on(A,B),S) означава, че А се намира върху В в ситуацията S;
* **Ситуациите** са термове, описващи състоянията на света (предметната област, сцената). При зададена начална ситуация, например s0, всяка следваща ситуация се получава чрез функцията result. Например result(puton(A,B),s0) представлява ситуацията, получена в резултат на прилагане на действие puton(A,B) в ситуация s0. По този начин могат да се записват и по-сложни ситуации: result(puton(C,A),result(puton(A,B),s0));
* Удобно представяне на връзката между ситуацията и плана за достигането й на Пролог: началната ситуация се представя чрез празния списък [], действието puton(A,B) в ситуация S води до ситуация [puton(A,B)|S]. Така всяка ситуация директно описва плана (списъка от действията, натрупани в обратен ред) за своето достигане;
* **Аксиомите** описват **действията** (с техните предусловия и ефекти). Те се представят чрез импликации, свързващи предусловията и ефектите (т.е. промените, които предизвикват), изразени с помощта на предиката holds. Например:

holds(clear(A),S)∧holds(clear(B),S) ->

holds(on(A,B),result(puton(A,B),S))

* В резултат от изпълнението на дадено действие ситуацията се променя, следователно е необходимо да се постулира също кои факти остават верни и в новата ситуация (т.е. какво от състоянието на сцената не се променя при извършването на действието). Това става чрез т. нар. фреймови аксиоми. Например:

holds(clear(X),S)∧¬eq(X,B) ->

holds(clear(X),result(puton(A,B),S))

* За всеки факт, който се използва при представянето (т.е. в описанието на състоянието на сцената), трябва да се зададе съответна фреймова аксиома. Следователно, възниква необходимост от голям брой фреймови аксиоми (това е същността на т. нар. рамков или **фреймов** **проблем**, **frame** **problem**). По такъв начин представянето се лишава от елегантност, а търсенето става неефективно.

Системата STRIPS може да бъде разглеждана в частност като сполучлив опит за решаване на фреймовия проблем. Тук се предполага, че всяка формула от описанието на сцената, която е била истина преди изпълнението на действието и не принадлежи на списъка на изтриванията на това действие, е истина и след неговото изпълнение.

Алгоритъмът за планиране в системата STRIPS представлява конкретизация на алгоритъма MEA.

**Подход, използван в системата STRIPS**

**Описание на състоянията и целта -** Състоянията (начални, междинни, целеви) се представят чрез конюнкции от термове, в които се използват предикатите on, ontable, clear, holding, handempty.

Примери:  
on(A,B) ∧ ontable(B) ∧ clear(A) ∧ holding(C)  
ontable(A) ∧ on(B,A) ∧ handempty  
on(x,A) ∧ on(y,x)



начално състояние целево състояние

on(C,A)∧ontable(A)∧ on(C,B)∧on(A,C)  
ontable(B)∧clear(C)∧  
clear(B)∧handempty

**Моделиране на действията на робота**

Елементарните действия на ръката на робота се представят с помощта на правила (оператори), всяко от които има три компонента:

* формула на предварителното условие (предусловие, Precondition);
* списък на изтриванията (Delete List);
* формула на добавянията (Add Formula).
* **Precondition**: ППФ от предикатното смятане от първи ред, която трябва да бъде логическо следствие от фактите в описанието на текущото състояние, за да бъде съответното правило приложимо към това състояние на сцената.
* **Delete** **List**: списък от литерали, който задава елементите от описанието на състоянието на сцената, които ще престанат да бъдат верни след изпълнението на правилото (оператора).
* **Add** **Formula**: конюнкция от литерали, които трябва да се добавят към описанието на текущото състояние, за да се получи новото състояние, което е резултат от изпълнението на даденото правило.

**Алгоритъм за планиране**

Поддържа се стек на целите и усилията по решаването на задачата се съсредоточават върху целта във върха на стека. Работата на алгоритъма за планиране се прекратява при получаване на празен стек на целите.

* Първоначално състояние на стека на целите: съдържа само главната цел.
* Действие при намиране на удовлетворена цел на върха на стека на целите: такава цел се изтрива от върха на стека и евентуално използваната субституция на съответствието се прилага към елементите на стека, разположени под разглежданата цел.
* Действие при намиране на неудовлетворена цел на върха на стека на целите: зависи от типа на неудовлетворената цел (еднолитерална или съставна).
  + Ако целта е съставна, в стека се добавят всички нейни компоненти.
  + Ако целта е еднолитерална, се търси правило, което съдържа във формулата на добавянията си литерал, съпоставим с тази цел. Разглежданата еднолитерална цел се заменя със съответния частен случай на избраното правило и над това правило се поставя съответният частен случай на предусловието на същото правило.
* Действие при намиране на правило на върха на стека на целите: правилото се изпълнява върху текущото състояние на сцената и в резултат състоянието се променя. Правилото се изтрива от върха на стека на целите и се добавя към плана.
* Преодоляване на проблемите, свързани с т. нар. взаимодействие на целите.

**В системата STRIPS се използват различни управляващи стратегии:**

* За избор при наличие на алтернативни възможности (например: при наличие на конюнкция от литерали и възможност за избор на реда на поставянето им във върха на стека на целите; при наличие на повече от един приложим оператор и др.);
* За откриване и премахване на поредици от излишни действия.

**Йерархично планиране**

Планът се изгражда на няколко равнища – като последователност от по-сложни действия, всяко от които може също да бъде обект на планиране.

Въвежда се отлагане на изпълнението (и проверката) на някои от предварителните условия. За целта се определя йерархия на условията, свързана обикновено с трудността на постигането на съответната цел.

**Реактивно планиране**

Осъществява се при системи, в които състоянието на сцената се променя не само в резултат на действията на агента, чийто действия се планират. Например, в света на кубовете, на сцената освен робота може да действа и малко дете, което разбърква по произволен начин предметите върху масата.  
Пример за реактивна планираща система с реална сложност: симулатор на военни действия.  
Методите за планиране в такива системи са съвсем други (класическите методи за планиране не вършат работа в тези системи).

**Лекция 6:  
Моделиране на игри**

**Класическа постановка на задачата:**

Разглеждат се т. нар. интелектуални игри с пълна информация, които се играят от двама играчи и върху хода на които не оказват влияние случайни фактори. Двамата играчи играят последователно и всеки от тях има пълна информация за хода на играта.

Най-често се решава задачата за намиране на най-добър първи ход на играча, който трябва да направи текущия ход.

ДС при тези задачи е дърво на възможните позиции в резултат на възможните ходове на двамата играчи. Общият брой възли в ДС е от порядъка на bd (Ь - коефициент на разклонение на игровото дърво, d- височина (дълбочина) на игровото дърво).

Пример. В шахмата е установено, че b има средна стойност около 35, а всеки играч играе около 50 хода, т.е. d има средна стойност около 100.

**Минимаксна процедура (Minimax Algorithm)**

**Обща характеристика**Метод за намиране на най-добрия ход на първия играч при предположение, че другият играч също играе оптимално.Изисква построяване на цялото ДС и намиране на оценките на листата (наричат се статични оценки).

Предполага се, че двамата играчи имат противоположни интереси, изразени в това, че единият търси стратегия, която води до получаване на позиция с максимална оценка (максимизиращ играч, МАХ), а другият търси стратегия, която води до получаване на позиция с минимална оценка (минимизиращ играч, MIN).

Минимаксната процедура е метод за получаване на оценките (наричат се придобити или породени оценки) на възлите от по- горните нива на ДС, които позволяват на първия играч да избере най-добрия си ход.

**Получаване на оценките на възлите от ДС**

Оценките се разпространяват отдолу нагоре, като всеки от възлите, съответни на ход на максимизиращия играч, получава оценка, равна на максималната от оценките на преките му наследници, а всеки от възлите, съответни на ход на минимизиращия играч, получава оценка, равна на минималната от оценките на преките му наследници.

**Оценка на минимаксната (minimax) процедура**

Изисква построяване на цялото ДС, което може да се окаже твърде голямо в общия случай на реална игра. Точен, но крайно неефективен и често нереализуем на практика метод.

**Алфа-бета процедура**

**Идея на метода -** Преодоляват се проблемите при минимаксната процедура, породени от обстоятелството, че там процесът на генерирането на ДС е напълно отделен от процеса на оценяване на позициите, което води до силна неефективност.

Тук листата се оценяват веднага след генерирането им и при първа възможност се пресмятат и съответните придобити (породени) оценки на възлите от по-горните нива или поне се намират подходящи горни граници на оценките на възлите от минимизиращите нива (съответни на ходове на минимизиращия играч) и/или долни граници на оценките на възлите от максимизиращите нива (съответни на ходове на максимизиращия играч). Така броят на операциите за намиране на същия резултат, както при минимаксната процедура, може да се намали значително.

Алфа-бета процедурата изисква генериране на ДС в дълбочина, при което се преценява безполезността от генерирането и оценяването на някои клонове, неоказващи влияние върху резултата.

**Дефиниции**

**Алфа-стойност** на даден максимизиращ възел се нарича текущо установената долна граница на породената му оценка.   
**Бета-стойност** на даден минимизиращ възел се нарича текущо установената горна граница на породената му оценка.

Първоначално определените алфа- и бета- стойности на възлите могат да се уточняват в процеса на работа, като алфа- стойностите могат само да растат, а бета-стойностите - само да намаляват.

**Описание на метода**

Правила за прекратяване на генерирането и търсенето:

* **Алфа-отсичане**. Не е необходимо да се извършва генериране и търсене върху всяко поддърво, което произлиза от минимизиращ възел, бета-стойността на който е по-малка или равна на алфа-стойността на съответния максимизиращ родител.
* **Бета-отсичане**. Не е необходимо да се извършва генериране и търсене върху всяко поддърво, което произлиза от максимизиращ възел, алфа-стойността на който е по-голяма или равна на бета-стойността на съответния минимизиращ родител.

Забележка. В много литературни източници понятията алфа- стойност и бета-стойност се дефинират не локално (по отношение на даден възел от съответния тип), а “глобално” (по отношение на текущо изследвания път в игровото дърво). Под алфа-стойност се разбира установената до момента най-добра стойност, която може да достигне максимизиращият играч по текущо изследвания път. Аналогично под бета-стойност се разбира установената до момента най-добра стойност, която може да достигне минимизиращият играч по текущо изследвания път. И при този подход към дефиницията на понятията алфа-стойност и бета- стойност правилата за алфа-отсичането и бета-отсичането могат да бъдат формулирани по начин, аналогичен на разгледания по- горе.

**Оценка на алфа-бета процедурата**

При използване на алфа-бета процедурата се получава напълно точен резултат. Броят на генерираните и оценени възли при алфа-бета процедурата е най-малко от порядъка на bd'2. Тази стойност се получава в идеалния случай, когато оценките на листата са наредени максимално добре. Следователно алфа-бета процедурата намалява скоростта на развитие на комбинаторния взрив, но не го предотвратява.

**Практически реализации на минимаксната (minimax) процедура**

* Отсичане на игровото дърво от дадено ниво надолу и намиране на евристични оценки на получените терминални възли (които в общия случай не са листа в ДС).
* Идеалната евристика би трябвало да отразява вероятността, че съответният играч ще победи в дадената (оценяваната) позиция. При добре познатите игри се използват опитът и интуицията на силните играчи. Евристиките могат да бъдат усъвършенствани експериментално.
* Пример за елементарна евристика при шахмата: разлика на тегловните суми от броя на белите и черните фигури (при което например 1 пешка е равна на 1/3 кон или офицер, 1/5 топ или 1/9 дама). Разположението на фигурите може да бъде отчетено чрез специално подбрани коефициенти.
* Определяне на нивото на отсичане (предполага се, че евристичните оценки на възлите от по-долните нива се намират по-лесно и са по-точни):
  + Предварително задаване на горна граница на дълбочината на генерираната част от ДС;
  + Използване на итеративно търсене по нива при предварително зададено максимално време за работа;
  + Търсене на затихване: генериране на дървото до достигане на “спокойна” позиция (позиция, която не предполага рязка промяна на оценката в непосредствено бъдеще).

**Лекция 7:  
Генетични алгоритми**

**Дефиниция на генетичен алгоритъм**

Вариант на **стохастично търсене в лъч**, при което новите състояния се генерират чрез комбиниране на двойки родителски състояния вместо чрез модифициране на текущото състояние.

**Основни принципи**

* **Състоянията** се представят като низове над дадена крайна азбука (често като низове от нули и единици).
* **Оценяваща функция (fitness function)**: оценява пригодността (близостта до целта) на съответното състояние. Има по-големи стойности за по-добрите състояния.
* Алгоритъмът започва работа с **множество** (**популация**) от k случайно генерирани състояния (поколение 0).
* Принципи на получаването на състоянията от следващите поколения: **селекция, кръстосване, мутация.**

**Селекция**

В генерирането на състоянията от следващото поколение участват някои от най-добрите представители на текущото поколение (съгласно оценяващата функция), избрани на случаен принцип.

**Кръстосване**

Избират се двойка “родителски” състояния и се определя т. нар. точка на кръстосването им (позиция в двата низа). Състоянието-наследник се получава чрез конкатенация на началната част на първия и крайната част на втория родител. Възможно е да се получи и друг наследник, в конструирането на който участват неизползваните части на двамата родители.

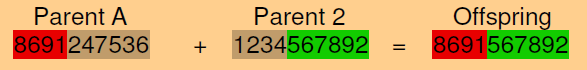
**Мутация**

Извършване на случайни промени в случайно избрана малка част от новата популация с цел да се осигури възможност за достигане на всяка точка от пространството на състоянията и да се избегне опасността от попадане в локален екстремум.

**Примери за различни типове кръстосване**

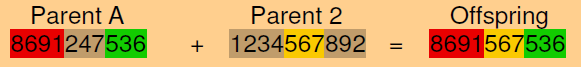
* **Кръстосване в единична точка**

Избира се една точка на кръстосване. Низът - резултат от началото си до точката на кръстосване е копие на началната част на единия родител, останалата му част е копие на съответната част на втория родител.



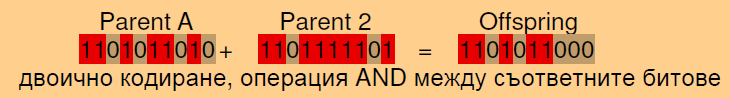
* **Кръстосване в две точки**

Избират се две точки на кръстосване. Низът - резултат от началото си до първата точка на кръстосване е копие на съответната част от първия родител, частта на резултата от първата точка на кръстосване до втората точка на кръстосване е копие на съответната част на втория родител и останалото е копие на оставащата след втората точка на кръстосване част на първия родител.



* **Аритметично кръстосване**

Извършва се определена операция (аритметична, логическа и т.н.) между двамата родители и в резултат се получава новото потомство.



**Характерни приложения на генетичните алгоритми**

* Решаване на оптимизационни задачи (задача за търговския пътник, задача за раницата и др.)
* Решаване на задачи за удовлетворяване на ограничения
* Избор на стратегия при игри за двама (по-общо, N) играчи
* Самообучение на невронни мрежи (уточняване на теглата на връзките между елементите в невронна мрежа с определена архитектура)
* Генетично програмиране

Пример: задача за търговския пътник. Дадени са група градове и разстоянията между тях. Търговският пътник трябва да посети всички градове, като минимизира общата дължина на пътя. Да се намери такъв маршрут на търговския пътник, който започва от даден град, завършва в същия град, включва всички останали градове точно по веднъж и образува път с възможно най-малка обща дължина.

**Кодиране (представяне на състоянията)**

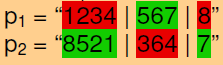
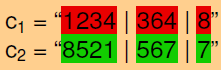
Състоянията се представят като поредици от означенията на съответните градове в реда, в който търговският пътник ще ги посети.

Пример: състоянието, представено чрез низа “153264798”, означава маршрут на търговския пътник, съставен от следната поредица от градове (предполага се, че градовете са означени с цифрите от 1 до 9): 1-5-3-2-6-4-7-9-8-1.

**Кръстосване**

Може да бъде както в единична точка, така и в две точки. И в двата случая са необходими допълнителни действия, които да осигурят получаването на коректно състояние като резултат от кръстосването.

Пример: кръстосване в две точки с получаване на поколение от двама наследници.

Ако  са две родителски състояния, то при кръстосването им ще се получат 

Очевидно с1 и с2 представят некоректни състояния, затова се прибягва до следната допълнителна операция: размените 3 <-> 5, 6 <-> 6 и 4 <->7, които са извършени при кръстосването на родителските състояния в областта между двете точки на кръстосване, се повтарят в областите извън точките на кръстосване.

Така окончателно се получава потомство 

**Мутация**

Размяна на два случайно избрани знака от съответния низ.

**Прекратяване на изпълнението на даден генетичен алгоритъм**

В зависимост от конкретната задача могат да бъдат използвани и други критерии (освен посочените в общата дефиниция) за проверка на “сходимостта” и вземане на решение за прекратяване на изпълнението на един генетичен алгоритъм. Такива критерии, които често се използват на практика, са:

* съвпадение/близост на оценките на най-добрите състояния от последните няколко популации;
* наличие на висок процент състояния от текущата популация, които са идентични или подобни едно на друго.

**Лекция 8:  
Представяне и използване на знания**

**Работа със знания в системите с изкуствен интелект. Данни и знания. Архитектура на системите, основани на знания**

**Обща постановка**

Работата със знания е една от съществените отличителни черти на програмните системи с ИИ. Според едно от популярните определения интелектът е способност за формулиране, натрупване и използване на знания (Intelligence is applied knowledge). Терминът “знания” (“знание”) в системите с ИИ се използва за означаване на кодирания опит на агентите. Опитът е източникът на знания (информация) за решаването на задачи. Чрез определението “кодиран” се означава обстоятелството, че знанията са формулирани, записани и готови за използване.

Съществена за системите с ИИ е възможността за преход от работа с данни към работа със знания. Традиционните програмни системи работят с информация, организирана във вид на бази от данни (БД), докато за системите с ИИ и по-точно за т. нар. системи, основани на знания (СОЗ; Knowledge-based Systems, KBS) е характерна работата с бази от знания (БЗ). Има фундаментална разлика между възможностите, които предоставят тези два типа системи за съхранение и предоставяне на информация.

При БД е възможно извличане само на такава информация, която е представена в явен вид в базата. При БЗ е възможно извършването на разсъждения (извод), в резултат на което може да се генерира нова информация, която не присъства в явен вид в базата. В този смисъл по принцип БД не могат да представят и СУБД не могат да обработват непълна информация, докато за БЗ такова ограничение не съществува.

**Същност на знанието**

В общ енциклопедичен план знанието обикновено се определя като система от съждения с принципна и единна организация, основана на обективни закономерности.  
В обективен смисъл (т.е. като резултат от определен познавателен процес) знанието се противопоставя на заблуждението.  
В субективен смисъл знанието представлява мнение, вяра, убеждение (по реални причини) в истинността на наблюдението.  
В СОЗ знанията представляват съвкупност от твърдения, представящи мнението, вярата и убежденията на т. нар. когнитивен агент.

Различия между данни и знания в СИИ:

* В степента на общност (и постоянност/изменяемост);
* По начина на използване;
* Наличие на класификационно-йерархична структура на знанията (основана на родово-видови отношения от тип обект – клас, клас – суперклас, тип – подтип, ситуация – подситуация и т.н.).

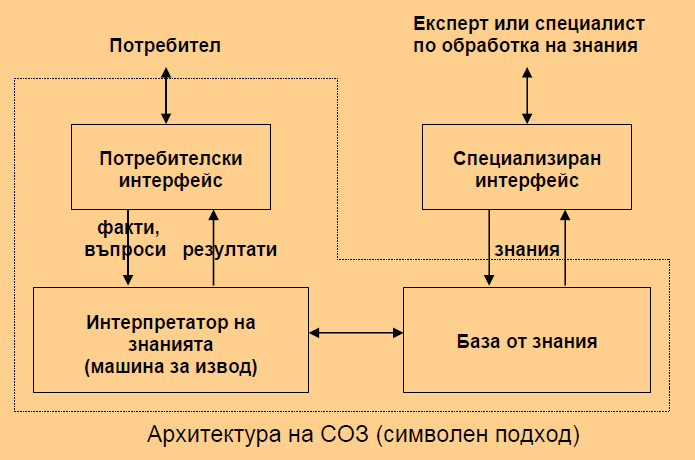
Няма рязка граница между данни и знания в СИИ.

**Архитектура на СОЗ**

Основава се на приемане (символен подход) или отхвърляне (конекционистки подход) на т. нар. хипотеза за представянето на знанията (knowledge representation hypothesis).

При символния подход:

* база от знания (knowledge base)
* машина за извод (inference engine) или интерпретатор на знанията (knowledge interpreter)



**Типове знания в СОЗ:**

(Знания за предметната област):

* знания за обектите и фактите в предметната област (фактологически знания)
* знания за връзките (релациите) между обектите и фактите
* метазнания  
  Видове метазнания: стратегически знания, поддържащи знания и др.

**Изисквания към формализмите за представяне и използване на знания (ПИЗ):**

* ясна семантика (значението на правилно построените изрази да бъде добре определено, т.е. всеки правилно построен израз да може да се интерпретира еднозначно);
* коректност на правилата за извод (правилата за извод да бъдат такива, че ако явно зададените знания в БЗ са верни, то всички извлечени от тях знания също да са верни);
* естественост на представянето (по отношение на терминологията и структурата на знанията за съответната предметна област);
* модулност на представянето;
* ефективност (по отношение на изискванията за памет и време).

**Основни типове формализми за представянето и използването на данни (ПИЗ)**

Съществуват два основни типа формализми за ПИЗ: формализми от декларативен тип (декларативни формализми – какво?) и формализми от процедурен тип (процедурни формализми – как?). При декларативните формализми основната тежест пада върху представянето на знанията; при процедурните формализми съществен е начинът на използване на знанията. При декларативните формализми знанията се представят явно, в декларативен формат, а при процедурните формализми знанията се съдържат в процедурите на някаква програма. Процедурните формализми дават възможност за по-голяма ефективност при използването на знанията, но при тях измененията в БЗ стават по-трудно, отколкото при декларативните.

**Представяне и използване на знания чрез процедури**

**Обща характеристика**

Типичен процедурен формализъм. Процедурите са обособени като формализъм за ПИЗ, защото всяка програма “носи” в себе си своите знания (част от знанията на своя автор) и в този смисъл те са представени чрез нея.

**Идея на представянето**

Най-често знанията се представят чрез системи от малки по обем процедури – демони, в които са кодирани знания за определени характерни ситуации и всяка от които се активира само при настъпване на съответната ситуация. За разлика от стандартните програмни системи, за които е характерна йерархична структура, за базите от процедури (демони) е типична хетерархична структура.

**Характерни области на приложение**

При управление на роботи, управление на процеси, протичащи в реално време, и др. (в области, в които преобладават алгоритмични знания).

**Представяне и използване на знания чрез средства на математическата логика**

**Обща характеристика**

Математическата логика може да се разглежда като един от най-рано създадените формализми за ПИЗ, който има ясно изразен декларативен характер. Всяка логическа система е формален език, с помощта на който могат да се изразяват различни твърдения (знания) и да се извеждат нови твърдения с помощта на съответни правила за извод.

По-точно, представянето на знания в термините на дадена формална логическа система се извършва с помощта на правилно построени формули в тази система, а използването на знанията се осъществява посредством валидните за логическата система правила и методи за извод. Най-често използваната формална логическа система за целите на ПИЗ е предикатното смятане от първи ред.

**Характерни особености на предикатното смятане от първи ред като формализъм за ПИЗ:**

* полуразрешимост
* монотонност (невъзможност за работа със знания, верни по подразбиране)

Некласически логики, които се използват като формализми за ПИЗ: немонотонни логики, модални логики, размита логика и др.

**Оценка на логическите системи като средства за ПИЗ**

В сила са всички общи предимства и недостатъци на декларативните формализми за ПИЗ. Най-съществени специфични предимства на логическите системи: ясна семантика, голяма изразителна сила.

**Представяне и използване на знания чрез системи от продукционни правила**

**Обща характеристика**

Декларативен в основата си формализъм с елементи на процедурност на по-ниско ниво. Негова основна характеристика е декомпозирането на знанията на малки части (правила) от типа условие – следствие (ситуация – действие).

**Архитектура на системите, основани на правила:**

* работна памет (контекст)
* база от правила
* интерпретатор на правилата

**Работна памет**

Съдържа данните за конкретно решаваната задача, които са установени (известни) към текущия момент. Произход на данните в работната памет:

* от потребителя (зададени по условие при дефинирането на задачата или получени като отговор на уточняващ въпрос, зададен от интерпретатора на правилата в процеса на обратен извод);
* от интерпретатора на правилата (получени в процеса на логическия извод).

**База от правила**

Съхранява знанията за предметната област, представени под формата на импликации от вида

(<име\_на\_правило> <лява\_страна> <дясна\_страна>)

Лявата страна на правилото се нарича негово условие (предпоставка), а дясната страна на правилото – негово следствие (действие). Най-често лявата страна на правилото представлява конюнкция от т. нар. елементи на условието. Дясната страна обикновено представлява поредица от елементи, описващи действия, които следва да се изпълнят (най-често това са твърдения - следствия, които се записват в работната памет в резултат на изпълнението на съответното правило). Структурата на елементите на условията и елементите на десните страни на правилата е сходна с тази на елементите на работната памет.

**Интерпретатор на правилата**

Програмна система, чието основно предназначение е да приложи описаните чрез правилата знания върху данните за конкретната задача.  
Процесът на работа на интерпретатора на правилата е цикличен, като всяка стъпка от работния цикъл на интерпретатора се разделя на две фази: избор на правило и изпълнение на избраното правило.  
При избора на правило, което да се изпълни на текущата стъпка, се извършват множество операции на съпоставяне по образец.

**Стратегии на работа на интерпретатора на правилата:**

* **Прав извод** (forward chaining) или извод, управляван от данните (data-driven inference). Пример: OPS5;
  + Предимства
    - Forward Chaining работи чудесно, когато наличната информация може да се използва за достигане на целевото състояние
    - Forward Chaining има способността да предоставя много данни от ограничените първоначални данни
    - Forward Chaining е най-подходящ за приложението на експертната система, изискващо повече контрол, планиране и наблюдение
    - Forward Chaining трябва да се прилага, когато има ограничен брой първоначални състояния или факти
  + Недостатъци
    - Интерпретаторът на правилата ще генерира нова информация, без да знае коя информация ще бъде от значение за достигане на целевото състояние.
    - Потребителят може да се наложи да въведе много информация първоначално, без да знае коя информация ще бъде използвана за достигане на целевото състояние.
    - Интерпретаторът на правилата може да задейства много правила, които не допринасят за достигане на целевото състояние.
    - Това може да даде различно заключение, което може да доведе до високата цена на процеса на веригиране
* **Обратен извод** (backward chaining) или извод, управляван от целите (goal-driven inference). Пример: Пролог.
  + Предимства
    - Backward chaining е насочено, така че обработката да приключва, когато фактът е проверен
    - Backward chaining разглежда само подходящи части от базата знания, така че никога да не се правят излишни изводи
    - За разлика от Forward Chaining, тук са необходими само няколко точки данни (data points), но правилата се търсят изчерпателно.
    - Backward chaining е много ефективен при проблеми като диагностициране и дебъгване
  + Недостатъци
    - Backward chaining е управляван от целите, следователно целта трябва да се знае предварително
    - Трудно е за имплементация

Конфликтно множество: множеството от правила, чиито леви страни се удовлетворяват от съдържанието на работната памет на текущата стъпка от работния цикъл на интерпретатора. Стратегии за разрешаване на конфликтите: стратегии за избор на правило от конфликтното множество (избор на първото възможно правило, което не предизвиква зацикляне; избор на най-актуалното правило; избор на правилото с най-сложното/най-простото условие и др.).

**Обща оценка на правилата като формализъм за ПИЗ**

**Специфични преимущества:**

* естественост на представянето на експертни знания;
* модулност на базата от знания;
* възможност за обяснение на резултатите от извода.

**Недостатъци и проблеми:**

* проблеми при генериране на съдържателни обяснения на взетите решения (необходимост от използване на добре структурирана база от знания, която освен знанията за предметната област, представени чрез правила, включва и различни типове метазнания, подпомагащи генерирането на обяснения);
* недостатъчна изразителна сила;
* потенциална неефективност на работата на интерпретатора на правилата (решения: алгоритъм RETE, архитектури от тип “черна дъска”).

**Експертни системи (експертна система)**

**Основни характеристики на експертните системи**

**Експертната система** (ЕС) е компютърна програма, която съдържа знания и извършва логически извод относно специализирана предметна област с цел решаване на определени задачи или даване на подходящи съвети. По обхвата, обема, съдържанието, структурата и организацията си знанията на една ЕС са сравними с тези на хората – експерти в съответната предметна област.

Обикновено ЕС се използват за решаване на следните типове задачи:

* Интерпретация на данни (например звукови сигнали);
* Диагностика на повреди или заболявания;
* Структурен анализ на сложни обекти (например химични съединения);
* Конфигурация на сложни обекти (например компютърни системи);
* Планиране на последователности от действия.

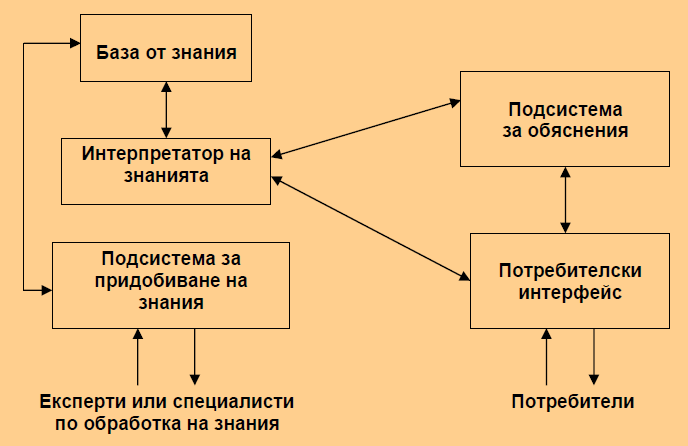
ЕС имат някои особености, които ги отличават от традиционните програмни системи:

* ЕС моделират начина, по който хората правят изводи в определена предметна област, а не самата област;
* ЕС извършват извод на базата на някакво представяне на човешки знания;
* ЕС обикновено решават задачи с помощта на евристични или приблизителни методи.

**Евристиките** са правила, извлечени от опита, които кодират определени знания за начина за решаване на даден проблем от определена област. Евристичните методи са приблизителни в смисъл, че не изискват точни данни и решенията могат да бъдат изведени от системата с определена степен на сигурност.

**Архитектура на ЕС - компоненти**

ЕС съдържат традиционните за всяка СОЗ **компоненти**, към които са добавени още т. нар. подсистема за обяснения и подсистема за придобиване на знания.



**Инструментални средства за създаване на ЕС:**

* редактори на знания и среди за придобиване на знания;
* празни ЕС (ядра на ЕС, ES shells) и среди за представяне на знания.

**Сравнителни таблици**

**Алгоритми за неинформирано търсене**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Depth First | Breadth First | Uniform Cost | Depth Limited | Iterative Deepening |
| Пълнота | не | да  (ако b – крайно) | да (за тегла >= ε) | да  (за l >= d) | да |
| Време | bm | bd+1 |  | bl | bd |
| Памет | b.m | bd+1 |  | b.l | b.d |
| Оптималност | не | да  (за нетегловни) | да | не | да  (за нетегловни) |
| Структура | стек | опашка | приор. опашка | стек | стек |

**Алгоритми за информирано търсене**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Критерий | Greedy | Beam Search | Hill Climbing | A\* | IDA\* |
| Пълнота | не,  защото може да влезе в безкраен цикъл  да,  за крайни графи и ако се добави проверка за повтарящи се състояния | не | не | да  освен ако няма безкрайно мн върхове с f <= f(G) | да |
| Време | bm | b.l.m | b.m | bd | bd |
| Памет | bm | b.l | b | bd | b.d |
| Оптималност | не | не | не | да | да |
| Структура | приор. опашка | приор. опашка с лимит | приор. опашка с лимит | приор. опашка | приор. опашка |

**Легенда:**

b – максимална разклоненост на дървото (колко най-мн деца може да има един връх)

d – дълбочината до най-късото решение

m – максималната дълбочина, която можем да имаме (може да е безкрайност)

C\* - цената на пътя до оптималното решение

**От асистента**

Видове агенти

Дефиниции и задачи в ИИ,

Какво е експертна система,

Видове и характеристики на техниките за представяне на знания,

Видове и характеристики на планирането

**Въпроси, които трябва да се намерят**

1. Какви са основните типове дъги в семантична мрежа? Дайте по един пример за всеки от изброените типове.

2. От какво зависи ефективността при извличане на информация?

3. Кои са основните етапи на процеса на разпознаване на образи?

4. Как се извършва обобщение при номинални атрибути в машинното самообучение?

5. Колко типа неврони има в една невронна мрежа (в общия случай)?

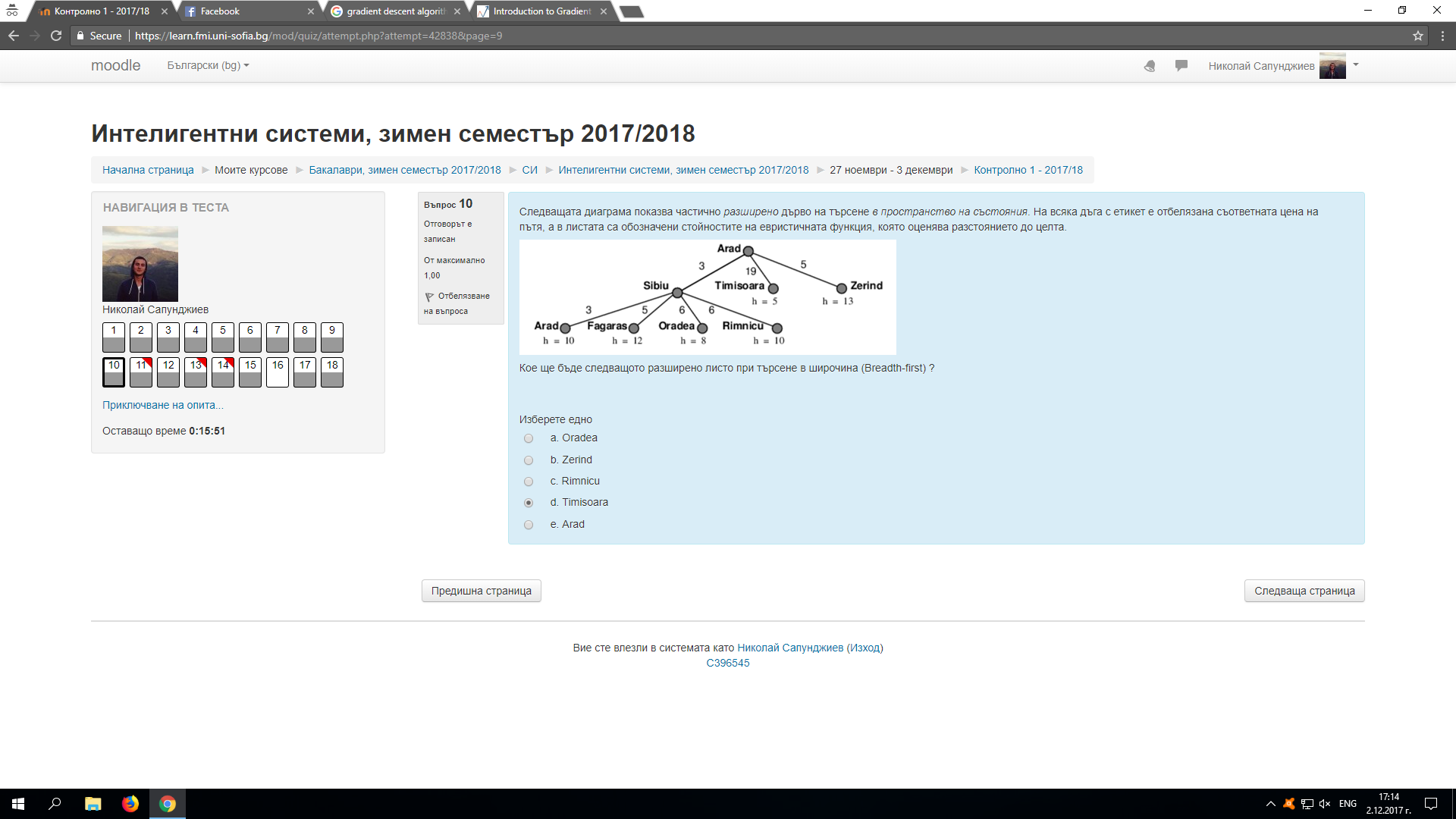
6. Какви типове извод се използват при бейсови мрежи?

7. Как оценяваме дали даден алгоритъм се представя адекватно върху конкретен проблем (Метрики за оценяване)?

8. Кои от следните алгоритми използва случайност за избягване на локален максимум?

1. Hill Climbing
2. Best First Search
3. Local Beam Search
4. Simulated Annealing
5. Gradient descent

9. 2017-2018 -> 1 -> 10



10. Какъв основен формализъм за представяне на знания използват традиционните Експертни системи?

1. Пропозиционални логики
2. Семантични мрежи
3. Фреймово представяне
4. Продукционни (правила) системи ?
5. Понятийни графи

11. Сравнете генетичните алгоритми с алгоритъма Local Beam Search (локално търсене в лъч).

Прилики:

* и двата са локално търсещи
* използват изнформирано търсене

Разлики:

* генетичните алгоритми подобряват най-добрите си състояния и ги комбинират, докато local beam search избира от текущия връх този, който е потенциално най-добър.
* генетичните алгоритми използват идеята за естествен подбор, докато local beam search използва определена евристика

12.