Записки по „Изкуствен интелект (ИИ)“, ФМИ, 3-ти курс, 2-ри семестър, 2023-2024

# Въведение – опит за дефиниция на ИИ

Опит за **дефиниция** на изучаваната област наречена изкуствен интелект:

* Моделиране на „интелигентно поведение”.
* Наука за това как да накараме компютрите да вършат неща, които засега ние хората вършим по-добре.
* Създаване на машини, които изпълняват функции, които изискват „интелигентност“, когато са изпълнявани от човек.
* Наука за концепциите и методите, даващи възможност на компютрите да извършват дейности, които се считат от хората за интелектуални (тази точка може би е най-доброто описание в едно изречение).

**Рационално поведение** – извършване на правилното действие, което предполагаме, че ще ни докара най-близо до целта, въз основа на наличната информация, с която разполагаме.

Актуални направления на развитие през последните години:

* Извличане на информация и откриване на знания от данни и текст:
  + разпознаване на двойно значение, преувеличения и невярна информация;
  + анализ на мнения и разпознаване на емоции;
  + резюмиране на съдържание.
* Разбиране на естествен език и генериране на текстове на естествен език.
* Генериране на изображения, музика, видео.
* Генериране програмен код.
* Семантични технологии:
  + семантично анотиране на текстове;
  + семантично обогатяване на колекции от данни;
  + машини за интелигентно (семантично) търсене.
* Компютърно зрение.
* Вградени интелигентни системи и комплекси от вградени интелигентни системи:
  + интелигентни роботи – приложения в хирургията, промишлеността и мн. др.;
  + системи „интелигентен дом“ – роботизирана битова техника и комплекси от роботизирани битови устройства с възможност за управление чрез смартфон, таблет или уеб браузър посредством Интернет или домашна мрежа.
* Машинно самообучение, включително т. нар. дълбоко самообучение (Deep Learning).
* Автономни интелигентни системи – интелигентни системи, които функционират без потребителски контрол; притежават способност да определят поведението си въз основа на натрупания опит (на базата на вградените си знания) и да действат по своя инициатива
  + роботи-хуманоиди;
  + промишлени манипулатори;
  + автономни превозни средства;
  + автономни дронове.

# 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. Например една прахосмукачка робот е агент – може да смуче прах (ефектор) и има сензор за допир с предмет (рецептор), като я пуснем, тя ще въздейства върху средата си (ще я направи по-чиста).

**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. Например шофьор на кола е агент – има цел да стигне до дестинацията си и може да въздейства на средата чрез карането си.

**Изображение (mapping)** – съответствието между възприятията и очакваните в отговор действия, чрез които се определя поведението на агента. Ако е достатъчно просто т.е. имаме краен брой възможности може да е зададено с таблица като на всяко възприятие ще съответства определено действие от таблицата. В общия случай го представяме като функция, която приема възприятие и връща действието на агента.

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. Т.е. агент, който действа само въз основа на вградените си знания изобщо не е автономен; агент, които въз основа на минали възприятия може да „трупа опит“ и да променя поведението си е по-автономен от първия (разбира се, промяната на поведението искаме да ни води по-близо до зададената цел).
* **адаптируемост** – способността 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 по-някакъв начин.
* **колаборативност** – способността за сътрудничество при 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 (това не знам за какво се отнася).

**Оценка на ефективността на поведението (performance measure)** – формулировката ѝ е изключително важна част от началната спецификация при проектирането на един интелигентен агент. Тя е в основата на това дали агентът ще прави, това което искаме или, това което сме му казали[[1]](#footnote-1).

**PEAS** – Performance measure, Environment, Actuators, Sensors (оценка на ефективността, среда, ефектори, рецептори) – това са основните елементи на спецификацията на интелигентен агент.

Пример:

* Agent: Medical diagnosis system.
* Performance measure: Healthy patient, minimize costs, lawsuits.
* Environment: Patient, hospital, staff.
* Actuators: Screen display (questions, tests, diagnoses, treatments, referrals).
* Sensors: Keyboard (entry of symptoms, findings, patient's answers).

Строеж на интелигентните агенти

**агент = архитектура + програма**

**Програма на агента** – конкретна реализация на изображението от възприятия към действия (някаква програмна функция). 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 е функция само на това въздействие. Тялото на програмата представлява цикъл, всяка стъпка на който включва възприятие (получаване информация от рецепторите), избиране и осъществяване на действие, като тези неща евентуално ги съхраняваме в паметта за „натрупване на опит“. Задачата на изкуствения интелект е именно създаването на такива програми. За конкретни реализации ще научим в идните теми.

**Архитектура** – това върху което работи програмата – обикновено компютър или специализирано устройство.

Типове агентни програми

Има 4 основни типа, подредени по това колко общо приложими са:

* **прости рефлекторни агенти (simple reflex agents)** – действията се определят от правила, които взимат в предвид само текущото възприятие, т.е. агентът не пази история на действията и възприятията си (затова е reflex agent; както мигането при хората – когато нещо доближи окото ни, не мислим, а просто мигаме). Тези програми 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, т.е. вече под някаква форма пазим миналите си възприятия и действия и последиците от тях. След всяко възприятие и действие обновяваме състоянието на света в паметта. За да може представата за средата да е горе-долу в съответствие с истинската среда, програмата на агента е нужно да знае:
  + Какви последици имат действията на агента и как средата се изменя от само себе си;
  + Как да интерпретира информацията, която получава от рецепторите (на пример при кола с автопилот по време на дъжд стъклото на камерата се мокри и това променя образа).
* **агенти, основани на цели (goal-based agents)** – имаме рефлекторен агент, на който възлагаме цел. Така действията се избират въз основа на това, което се знае за очакваните им резултати, и целите на агента. В общия случай целта не се достига едностъпково, затова се ползват методите **търсене** и планиране (агентът планира няколко стъпки напред и евентуално намира нужните действия за постигане на целта си). Целта в случая е бинарна величина зависеща от състоянието: ако средата е в състояние А, то целта е постигната; ако средата е в състояние Б, то целта не е постигната.
* **агенти, основани на полезност (utility-based agents)** – целите не винаги са достатъчни за полезно поведение на агента – може да има конфликтни цели, които не могат да бъдат изпълнени едновременно, може различните цели да са с различна важност, може начина на постигането да е недобър. Критерии, за това колко близо сме до целите си, ни дава **функцията на състоянието (utility function)**, която отразява степента на задоволеност на агента според текущото състояние на средата. Вече дефинирахме **оценка на ефективността на поведението (performance measure)** като ф-я на състоянието на света. Функция на състоянието е вътрешната репрезентация на външната оценка на ефективността. Ако **utility function** и **performance measure** съвпадат, и агентът максимизира своята **utility function**, то този агент максимизира оценката на ефективността си (т.е. прави това, което искаме 😊).

Друга важна характеристика е способността на агентите да се самообучават.

Характеристика на средите, в които действат агентите

1. **(не)достъпност** – агентът (не) може да възприеме цялостното състояние на средата по всяко време.
2. **(не)детерминираност** – агентът (не) може точно да предвиди следващото състояние на средата въз основа на текущото ѝ състояние и своите действия.
3. **(не)епизодичност** – историята (не) е разделена на епизоди. При епизодичност във всеки епизод агентът получава възприятия и прави съответни действия, като всеки епизод е независим от другите.
4. **статичност/динамичност** – средата (не) може да се промени, докато агентът планира действията си. Ако средата не се променя докато агентът „мисли“, но се променя оценката за ефективността на агента, то средата е полудинамична.
5. **дискретност/непрекъснатост** – дали има краен брой (или изброимо много ясно разграничени) или неизброимо много състояния. Това всъщност е описание на средата, времето, възприятията и действията на агента. В шаха например имаме краен брой среди (позиции на фигури на дъската) и краен брой действия; може да считаме времето за дискретно – ходовете се изпълняват един по един. Самоуправляващ се автомобил има на практика непрекъснати газ, спирачка и ъгъл на завъртане на кормилото; образът от камера също считаме за непрекъснат; средата е непрекъсната времето е непрекъснато.

# Търсене в пространство на състоянията

## Общи характеристики

**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.

**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 състояния (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). В случая на дърво е по-лесно търсенето, защото няма цикли и не можем да зациклим.

**Действия, свързани с ПС:**

* Генериране на състояния:
  + генериране на следващ наследник.
  + генериране на всички наследници.
* Оценяване на състояние:
  + двоични оценки (true/false). Дадено състояние или считаме за целево, или не.
  + числови оценки в определен интервал (оценката на целта съвпада с единия край на интервала).

**Основни типове задачи, свързани с ПС:**

* Генериране на ПС;
* 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.

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.

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

Примери: шахмат, шашки, морски шах и др.

1. **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.

**Характеристики на алгоритмите за търсене:**

* **Пълнота (completeness)** – алгоритъм е пълен (complete), ако е гарантирано, че ще намери решение, ако такова съществува[[2]](#footnote-2).
* **Оптималност (cost optimality)** – алгоритъм е оптимален т.с.т.к., ако намереното решение е винаги с най-малката възможна стойност на пътя. Може един алгоритъм да е оптимален, но да не е пълен (например, ако зацикля на някой от входовете си).
* **Сложност по време и памет.**

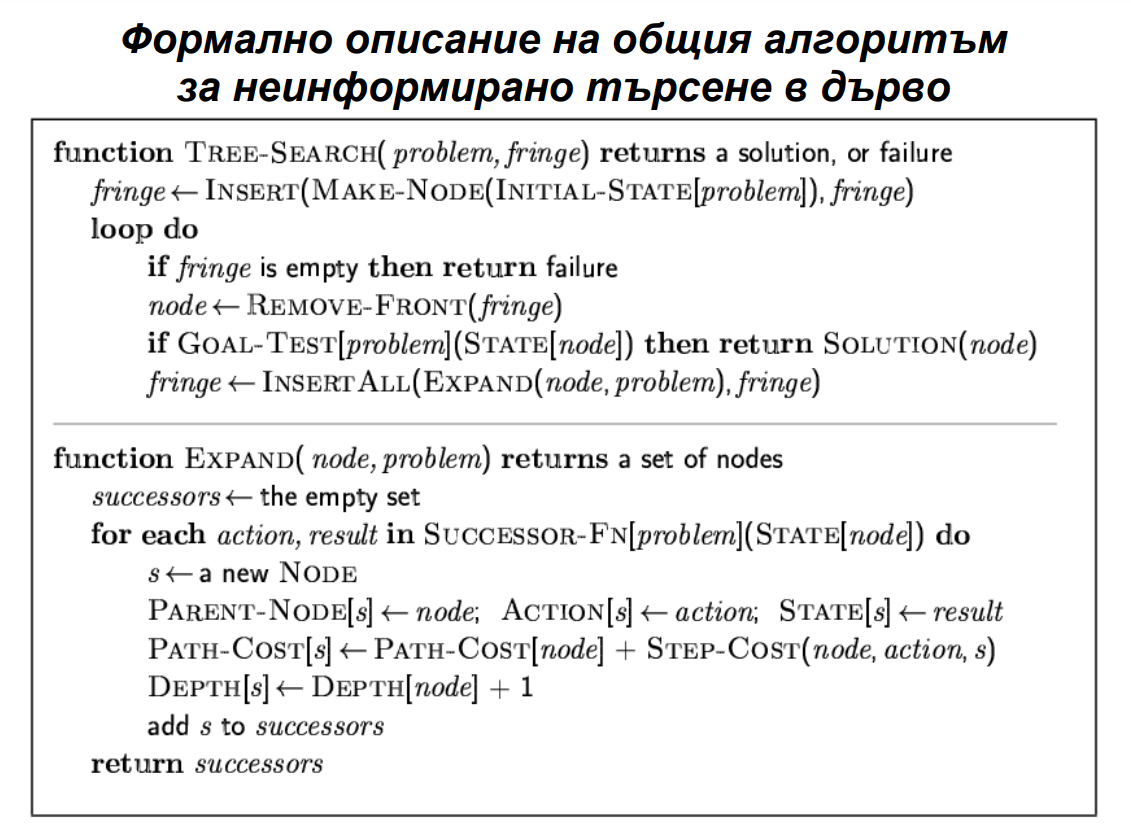
**Параметри на търсенето:**

* **d** – дълбочина на „най-плитката“ цел (ако такава съществува[[3]](#footnote-3)). През колко ребра от графа трябва да минем, за да достигнем до най-плитката цел.
* **m** – максималната дълбочина на графа на състоянията (може да е ∞).
* **b** – коефициент на разклоненост на графа. Средно колко наследника има всеки връх. Възможно е да е ∞, но в този курс, ако не е изрично споменато, ще считаме за крайно число.

## Методи за неинформирано (сляпо) търсене

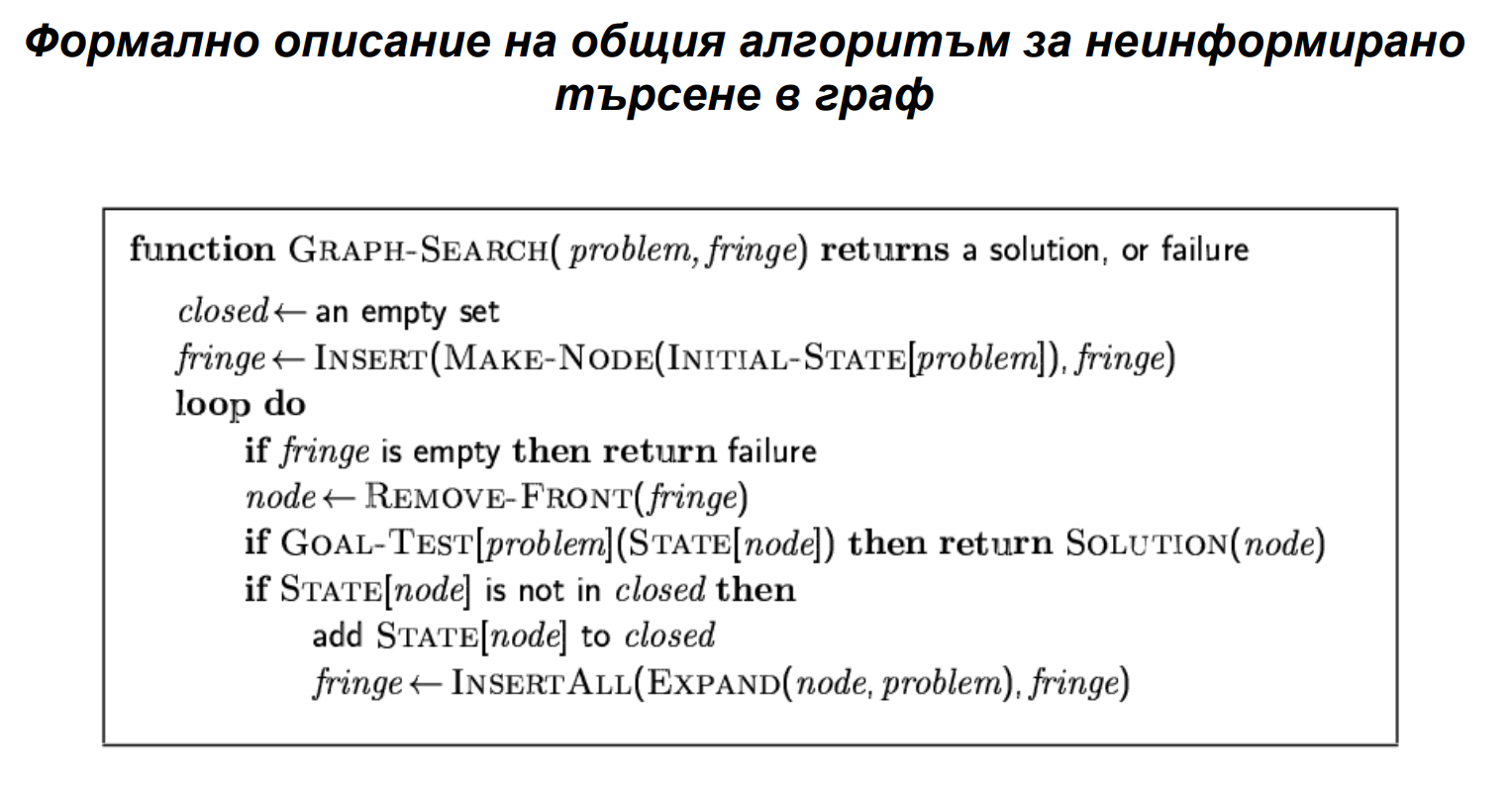
**Обща характеристика** – пълно изчерпване по твърда (фиксирана отнапред) стратегия. Прилагат се, когато липсва специфична информация за предметната област и оценяващата функция може само да провери дали дадено състояние е целево или не е (мисли например BFS и DFS в произволен граф).

**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)** (например стратегията при BFS е систематично да се обхожда ниво по ниво).



**Описание на ползваните функции и структури от данни:**

* problem – описанието на задачата.
* fringe – означава „покрайнини“, „ръб на нещо“ – това е структура от данни, която съхранява фронтът от тези върхове, които сме достигнали, но все още не сме обработили напълно (напр. при BFS е опашка, при DFS е стек). Ф-ята Tree-Search се извиква с празен fringe (и на първия ред добавяме началния връх).
* node – запис, който има член-данни:
  + State – съответното състояние на задачата;
  + Parent – родителския връх;
  + Action – използвания оператор, чрез който е получено състоянието от родителя;
  + Depth – дълбочината (броят на стъпките от началото);
  + Path-Cost – цената на пътя от началото (традиционно означаваме с g(n)).
* Successor-Fn() – връща всички възможни състояния, които могат да се получат от даденото.
* Expand() – връща изхода на Successor-Fn(), само че обвит в записа node, с попълнена съответната информация за върховете.
* Значението на останалите неща е очевидно.



Виждаме, че единствената разлика с търсенето в дърво е, че пазим кои върхове сме обходили в структурата closed, и не ги обхождаме повече от веднъж.

## Конкретни алгоритми за неинформирано търсене

Прескочи на съответната секция, за да видиш само обобщените резултати за пълнота, оптималност и сложност.

### Изчерпване (търсене) в дълбочина (depth-first search)

За да стане общият алгоритъм за неинформирано търсене в граф по-горе алгоритъм за търсене в дълбочина, единствено fringe го правим на стек.

1. Пълнота[[4]](#footnote-4) – НЕ Е пълно в общия случай. При краен граф е пълно (и тогава винаги завършва). При безкраен граф може да забие по някой безкраен клон, на който няма решение.
2. Оптималност – НЕ Е оптимално. Може „втория“ наследник на върха да е решение, но алгоритъмът първо ще изследва поддървото на „първия“ и може там да намери някое по-дълбоко решение.
3. Сложност – да допиша

Въпреки че DFS не е пълен, в случая когато ПС има много решения и са на дълбоки, може да се окаже добър избор за алгоритъм за търсене.

### Търсене в дълбочина до определено ниво (depth-bound / depth-limited search)

Вариант на DFS. Просто ограничаваме дълбочината, до която може да стигнем.

1. Пълнота – НЕ (може да има решение на по-долно ниво).
2. Оптималност – НЕ (същите съображения като DFS).
3. Сложност – да допиша

### Итеративно търсене по нива (iterative deepening search)

Вариант на DFS. За всяка дълбочина 0,1,2,… пускаме depth-limited search.

1. Пълнота – ДА. Систематично обхождаме нивата и не може да забием по безкраен клон на графа без решение. Всъщност е пълен само когато **b** < ∞, но го считаме за пълен.
2. Оптималност – ДА. Отново аргументът е, че систематично изследваме по нива и няма как да пропуснем първото решение.
3. Сложност – да допиша

Това е най-предпочитаният алгоритъм за търсене, когато ПС е с голям размер и не знаем нищо за дълбочината на решението.

### Изчерпване (търсене) в широчина (breadth-first search)

За да стане общият алгоритъм за неинформирано търсене в граф по-горе алгоритъм за търсене в дълбочина, единствено fringe го правим на опашка.

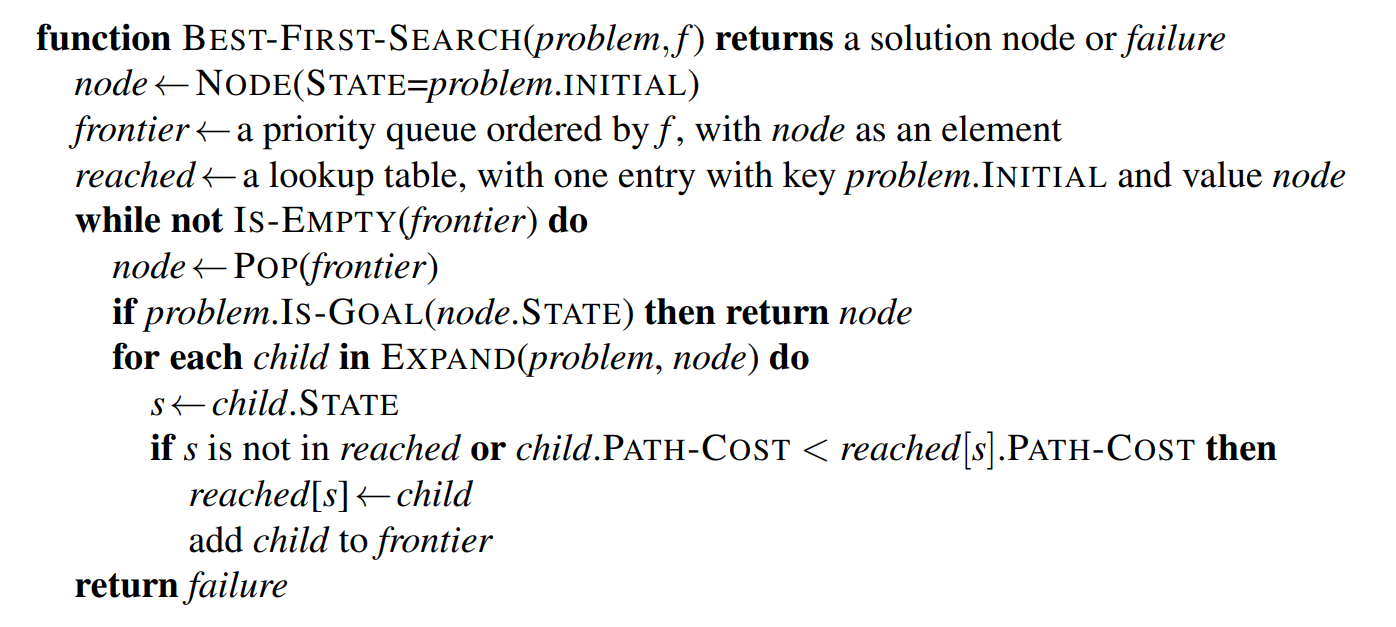
1. Пълнота – ДА. Всъщност е пълен само когато **b** < ∞, но го считаме за пълен.
2. Оптималност – ДА.
3. Сложност – да допиша

Всъщност BFS e частен случай на следващия алгоритъм – uniform-cost search.

### Търсене с равномерна цена на пътя (uniform-cost search)

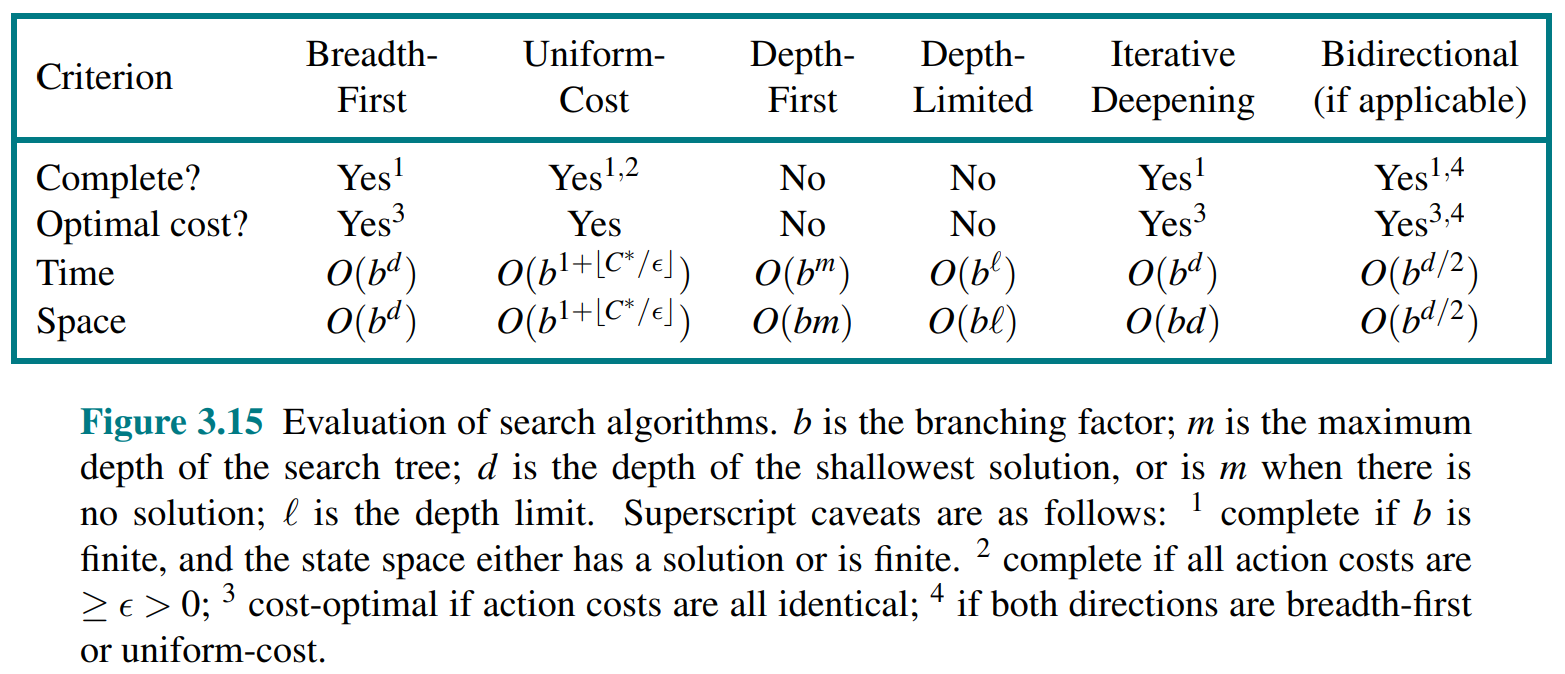
Тук вече графът на състоянията е тегловен, всяко ребро си има цена c ≥ ε > 0, където ε е фиксирано число. Това всъщност е точно алгоритъмът на Dijkstra за намиране на най-кратък път между два върха в (ориентиран) тегловен граф, просто в областта на ИИ е прието да се казва uniform-cost search. Името uniform-cost идва от това, че фронтът се състои от тези върхове, които имат горе-долу еднаква цена на пътя от корена (докато фронтът на BFS е от върховете на еднаква дълбочина от корена). fringe е приоритетна опашка подредена по нарастващ ред на дължините на пътищата.

Uniform-cost search може да се имплементира по следния начин, като извикаме Best-First-Search с параметър f – дължината на пътя. За повече инфо виж книгата на курса.



1. Пълнота – ДА. Ако **b** < ∞, и всички тегла са ≥ ε > 0, ε – фиксирано.
2. Оптималност – ДА.
3. Сложност – да допиша

### Обобщени резултати за алгоритмите за неинформирано търсене



**Означения:**

* **b** – коефициент на разклонение на ГС
* **d** – дълбочина на най-плитката цел
* **m** – максимална дълбочина на ГС (може да бъде ∞)
* **l** – максимална дълбочина на изследване
* **C\*** – цена на оптималния път (предполага се, че всички стъпки имат цена ≥ ε, ε > 0)

## Методи за евристично (информирано) 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 (наречена евристика), която връща числова оценка в предварително определен интервал. Тази оценка може да служи например за приблизителна мярка на близостта на оценяваното състояние до целта или на необходимия ресурс за достигане от оценяваното състояние до целта.

### Евристика

**Евристична функция (евристика) (heuristic function)** – приема състояние (връх от графа) и ни връща приблизителната цена на пътя от това състояние до целевото състояние. Обикновено бележим с h(n).

Например, ако имаме карта на градове и пътищата между тях, евристична функция, която връща разстоянието по права линия между текущия и целевия град, обикновено е нелош метод за приблизителното оценяване на пътя между двата града. Общо взето, колкото по-близко е една евристики до реалната дължина на пътя, толкова по-добре ни вървят алгоритмите, т.е. дават по-близко до оптималното решение и са по-бързи като разглеждат само състояния, които са на пътя до целта. В екстремалния случай, когато евристиката съвпада с реалното разстояние до целта, буквално ще „вървим“ единствено по оптималния път директно до целта.

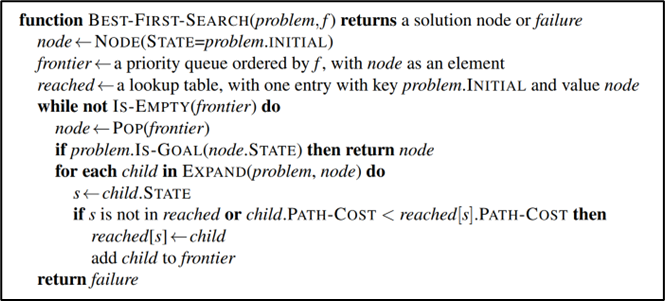
h(n) връща стойности в някакъв предварително определен интервал [a, b] от числа, като h(goal) = a (или по-често a = 0 и h(goal) = 0)

**Приемлива (оптимистична, admissible) евристика** – такава, която никога не надценява цената на пътя до целта h(node) ≤ actual-cost.

**Консистентна (consistent) евристика** – такава, за която е изпълнено h(n) ≤ c(n, n’) + h(n’), където n‘ е дете на n и c(n, n’) е цената на реброто между тях. Това наподобява неравенството на триъгълника – евристичното разстояние между два върха винаги ще е по-малко от това първо да отидем до друг връх и след това да добавим евристичното разстояние. Консистентността е по-силно понятие от приемливостта. Ако една евристика е консистентна, то тя е и приемлива, но обратното не е вярно.

## Конкретни алгоритми за информирано търсене

Алгоритмите в тази секция са базирани на Best-First-Search, просто с различни аргументи f.



### Метод на най-доброто спускане (greedy best-first search)[[5]](#footnote-5)

Викаме Best-First-Search с f(n) = h(n). Понеже в общия случай h(n) ни дава само приблизителна оценка на цената на пътя, то този алгоритъм нито е пълен, нито оптимален.

1. Пълнота – НЕ. Пълен само за крайно ПС.
2. Оптималност – НЕ.
3. Сложност в най-лошия случай – време O(bm), памет O(bm).

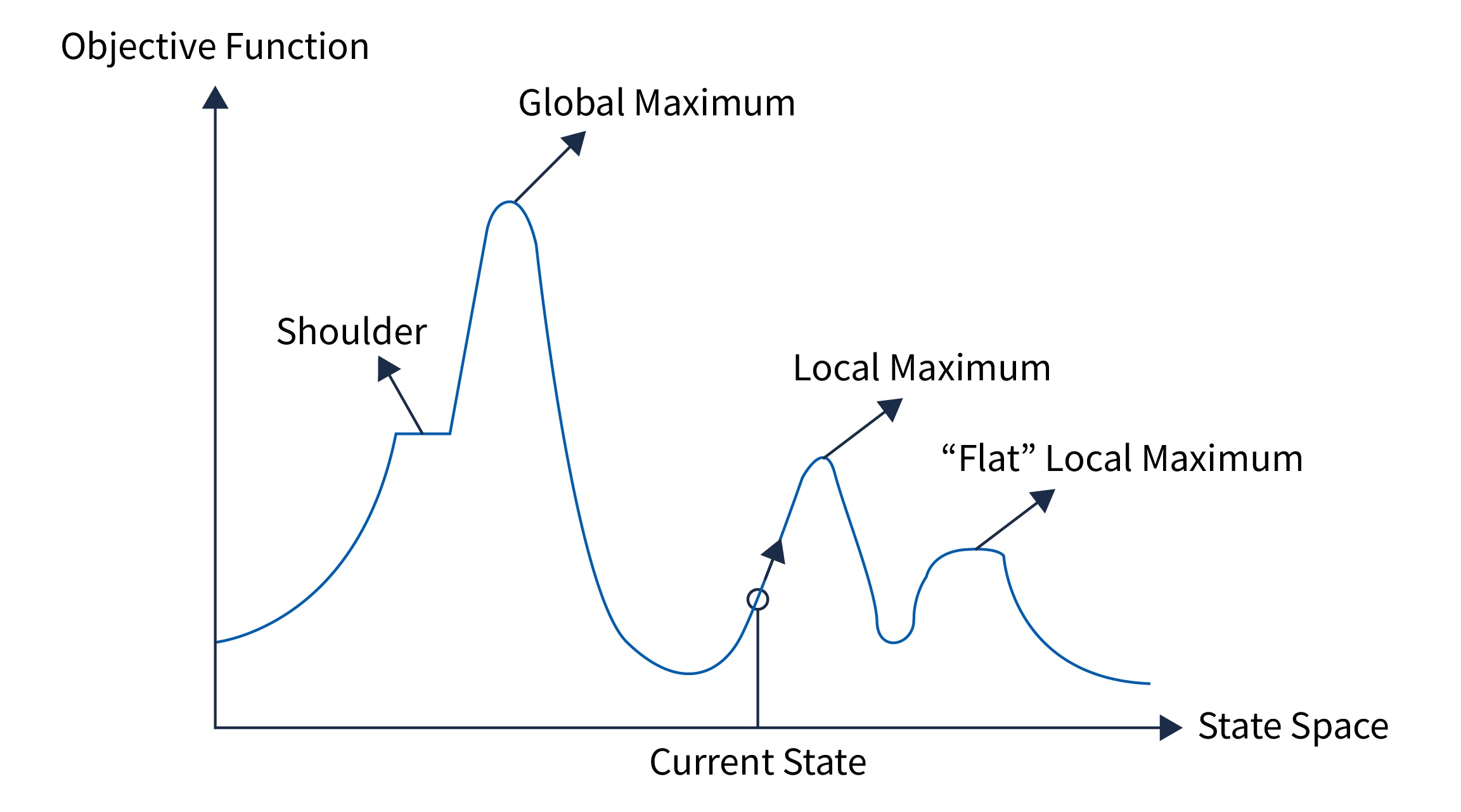
Въпреки че най-лошият случай е с много голяма сложност, с подходяща евристика може да работи със сложност достигаща O(bm).

Варианти:[[6]](#footnote-6)

1. **Търсене с ограничена широчина (търсене в лъч, beamsearch)** – ограничава списъка Open/fringe до първите n най-добри възела от него (в съответствие с евристиката). При n=1 е методът на най-доброто изкачване.
2. **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 („плосък“ локален екстремум)** – съседните състояния (наследниците на текущото състояние) изглеждат също толкова добри, колкото и текущото;
* **стъпало** – 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.



### Търсене с минимизиране не общата цена на пътя (A\* search)

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.

1. Пълнота – ДА, ако **b** < ∞ и цената на преходите е ≥ ε > 0, ε – фиксирано.
2. Оптималност – ДА, ако евристиката е приемлива.
3. Сложност – време O(bm), памет O(bm).

Това е най-често използваният алгоритъм за информирано търсене.

1. Тук се сещам за известния цитат: “Computers always do what we tell them, but rarely do what we want.” [↑](#footnote-ref-1)
2. Тук е възможно да поискаме и по строго условие, а именно: алгоритъм е пълен, ако е гарантирано, че ще намери решение, ако такова съществува и ще изведе съобщение, ако такова не съществува. Според тази дефиниция, ако алгоритъм е пълен, то той винаги завършва, което е много строго условие, затова ще се придържаме към първата дефиниция. [↑](#footnote-ref-2)
3. Очевидно, ако крайна цел съществува, то задължително **d** < ∞. [↑](#footnote-ref-3)
4. Припомням, че алгоритъм за търсене е пълен т.с.т.к. намира решение, ако такова съществува. [↑](#footnote-ref-4)
5. В книгата на курса се го наричат **greedy best-first search**, но на лекции го наричаме само **best-first search**. [↑](#footnote-ref-5)
6. Всъщност тези варианти могат да се приложат кажи-речи за всеки алгоритъм за търсене. [↑](#footnote-ref-6)