Lekce 1: Úvod do Umělé Inteligence a Její Historie

Tento studijní průvodce rozšiřuje a kombinuje poznámky z přednášky s kontextem z učebnice, aby poskytl ucelený úvod do oblasti umělé inteligence.

1. Co je Umělá Inteligence?

Pojem "umělá inteligence" (UI) se skládá ze dvou slov: "umělá" a "inteligence". Pochopení obou je klíčové pro definování celého oboru.

1.1 Umělý Artefakt

Slovo **umělý** odkazuje na člověkem vytvořený artefakt. Proces jeho vzniku lze popsat ve třech krocích:

1. **Existence přirozeného vzoru:** Existuje nějaká přirozená věc nebo jev, který je možno duplikovat (např. sníh, kloub, inteligence).
2. **Lidský záměr:** Existuje vědomý záměr člověka vytvořit duplikát této přirozené věci.
3. **Provedení záměru:** Dojde k samotnému aktu vytvoření, jehož výsledkem je umělý artefakt.

V kontextu UI je onou "přirozenou věcí" lidská inteligence a záměrem je vytvořit stroje, které ji dokáží napodobit nebo simulovat.

1.2 Definice Inteligence

Inteligence je komplexní pojem, který byl definován mnoha způsoby. Zde jsou tři pohledy:

- **W. Stern:** "Všeobecná schopnost individua vědomě orientovat vlastní myšlení na nové požadavky; je to všeobecná duchovní schopnost přizpůsobit se novým životním úkolům a podmínkám."
- **D. Wechsler:** "Vnitřně členitá a zároveň globální schopnost individua účelně jednat, rozumně myslet a efektivně se vyrovnávat se svým okolím."
- **J. P. Guilford:** "Schopnost zpracovávat informace, přičemž informacemi je třeba chápat všechny dojmy, které člověk vnímá."

Společným jmenovatelem těchto definic je schopnost adaptace, účelného jednání, rozumného myšlení a zpracování informací.

1.3 Umělá Inteligence (AI)

Když spojíme obě části, dostaneme definici oboru. **Umělá inteligence (AI)** je vědní obor, který se pokouší nejen porozumět inteligentním entitám, ale také je **vytvářet**. Je to snaha postavit stroje, které vnímají, chápou, předpovídají a manipulují se světem, který je mnohem větší a složitější než ony samy.

2. Přístupy k Umělé Inteligenci

Existují čtyři hlavní přístupy k UI, které se liší ve dvou dimenzích: zda se zaměřují na **myšlení** nebo na **chování** a zda měří úspěch porovnáním s **lidským výkonem** nebo s ideálním standardem **racionality**.

	Lidské (Humanly)	**Racionální (Rationally)**

Myšlení	1. Myšlení podobné lidskému	2. Myšlení racionálně
Chování	3. Jednání podobné lidskému	4. Jednání racionálně

1. **Myšlení podobné lidskému (Thinking Humanly): Kognitivní modelování**
    - Cílem je vytvořit programy, které myslí jako lidé. Abychom to mohli udělat, musíme pochopit, jak lidé myslí (pomocí introspekce, psychologických experimentů). Tento přístup je úzce spjat s **kognitivní vědou**. Příkladem byl program GPS (General Problem Solver) od Newella a Simona.
2. **Myšlení racionálně (Thinking Rationally): Zákony myšlení**
    - Tento přístup je založen na myšlence "správného myšlení" a irrefutabilních procesů odvozování. Jeho kořeny sahají až k Aristotelovi a jeho sylogismům. Moderním nástrojem je **logika**. Cílem je formalizovat znalosti a odvozovat z nich logicky platné závěry.
3. **Jednání podobné lidskému (Acting Humanly): Turingův test**
    - Tento přístup definoval Alan Turing (1950). Cílem je vytvořit stroj, jehož chování je nerozeznatelné od lidského. **Turingův test** spočívá v tom, že lidský tazatel pokládá psané otázky a na základě odpovědí se snaží určit, zda komunikuje s člověkem nebo se strojem. Aby stroj testem prošel, musel by zvládat zpracování přirozeného jazyka, reprezentaci znalostí, automatizované uvažování a strojové učení.
4. **Jednání racionálně (Acting Rationally): Racionální agent**
    - Tento přístup je v současné UI dominantní. **Agent** je cokoli, co jedná (vnímá své prostředí a provádí akce). **Racionální agent** je takový, který jedná tak, aby dosáhl nejlepšího možného výsledku (nebo nejlepšího očekávaného výsledku v případě nejistoty). Je to obecnější přístup než "zákony myšlení", protože správné odvozování je jen jedním z mechanismů pro dosažení racionality. Tento přístup je také lépe uchopitelný pro vědecký vývoj.

3. Druhy Inteligence

Kromě přístupů k UI můžeme rozlišovat i různé druhy inteligence, které se snažíme modelovat:

- **Abstraktní inteligence:** Schopnost řešit dobře definované, akademické problémy, které mají jednoznačnou odpověď. Příkladem je řešení matematických úloh nebo hraní šachů.
- **Praktická inteligence:** Schopnost řešit problémy každodenního života, které mají často nejednoznačné zadání i řešení. Například plánování cesty nebo oprava spotřebiče.
- **Sociální inteligence:** Schopnost efektivně se pohybovat a jednat v sociálním prostředí, chápat sociální signály a interagovat s ostatními.
- **Emoční inteligence:** Schopnost rozpoznávat, chápat a ovládat vlastní emoce i emoce ostatních.

4. Silná vs. Slabá UI

Toto je klíčové filozofické dělení v rámci UI.

- **Slabá UI (Weak AI):** Hypotéza, že stroje mohou **jednat, jako by byly** inteligentní. Většina současného výzkumu v UI spadá do této kategorie. Cílem je vytvářet užitečné nástroje, které řeší konkrétní problémy, bez ohledu na to, zda skutečně "myslí".

- **Silná UI (Strong AI):** Hypotéza, že stroje, které jednají inteligentně, skutečně **myslí** a mají vědomí, nikoli jen simulaci myšlení. Tato myšlenka tvrdí, že povaha mysli je algoritmická a nezáleží na tom, zda je algoritmus implementován v mozku nebo v počítači.

Slavným argumentem proti Silné UI je myšlenkový experiment **Čínský pokoj (Chinese Room)** od Johna Searla, který tvrdí, že pouhá manipulace se symboly podle pravidel (což dělají počítače) neznamená skutečné porozumění.

5. Vývojové Paradigmy UI

- **Klasická UI:** Chápe inteligenci jako atribut jedné mysli. Reprezentuje ji přístup GOFAI (Good Old-Fashioned AI), který je založen na symbolické reprezentaci znalostí a logickém odvozování.
- **Distribuovaná UI:** Chápe inteligenci jako produkt sociálních interakcí více myslí. Tento přístup vedl ke vzniku oblasti multi-agentních systémů, kde spolupracuje nebo soutěží více autonomních agentů.
- **Nová UI:** Chápe inteligenci jako emergentní výsledek činnosti velkého množství jednoduchých, primitivních entit. Nejlepším příkladem je **konekcionismus** (neuronové sítě), kde jednoduché jednotky (neurony) propojené do sítě dávají vzniknout komplexnímu chování.

6. Stručná Historie UI

- **Gestační období (1943–1955):** První práce, které jsou dnes považovány za UI. Warren McCulloch a Walter Pitts navrhli model umělého neuronu. Donald Hebb představil pravidlo pro učení (Hebbian learning).
- **Zrození UI (1956):** Na Dartmouthském workshopu byl poprvé použit termín "umělá inteligence". Setkali se zde klíčoví zakladatelé oboru, včetně McCartyho, Minskyho, Newella a Simona.
- **Raný entuziasmus (1952–1969):** Období velkých očekávání a úspěchů v omezených oblastech (tzv. mikrosvětech). Vznikl program General Problem Solver (GPS), jazyk Lisp a programy na řešení geometrických a algebraických úloh.
- **Dávka reality (1966–1973):** První "zima UI". Ukázalo se, že metody, které fungovaly v mikrosvětech, selhávají u reálných problémů kvůli **kombinatorické explozi**. Kritika vedla k seškrtání financování, např. po Lighthillově zprávě ve Velké Británii.
- **Znalostní systémy (1969–1979):** Posun od univerzálních metod k systémům se specifickými znalostmi pro úzkou doménu. Vznikly **expertní systémy** jako DENDRAL (chemie) a MYCIN (medicína), které byly založeny na velkém množství pravidel typu "pokud-pak".
- **UI se stává průmyslem (1980–současnost):** První komerční úspěchy expertních systémů (např. R1 od DEC). Vznikají specializované firmy a UI se stává miliardovým průmyslem.
- **Návrat neuronových sítí (1986–současnost):** Znovuobjevení algoritmu zpětného šíření (back-propagation) vedlo k renesanci konekcionismu a neuronových sítí.
- **Vznik inteligentních agentů (1995–současnost):** Výzkum se opět zaměřuje na "celého agenta", který integruje různé schopnosti (vnímání, plánování, učení, jednání). Velký vliv má internet a potřeba autonomních softwarových agentů.
- **Dostupnost velkých dat (2001–současnost):** Důraz se přesouvá z algoritmů na data. Ukazuje se, že i jednodušší algoritmy dosahují skvělých výsledků, pokud mají k dispozici obrovské množství dat pro učení (např. miliardy webových stránek nebo obrázků).

Lekce 2: Vyhodnocování Inteligence Umělých Systémů

Tento studijní průvodce se zabývá fundamentální otázkou: "Mohou stroje myslet?" a zkoumá různé historické i moderní přístupy k hodnocení inteligence umělých systémů.

1. Filozofické Dělení Umělé Inteligence

Než se ponoříme do testování, je důležité rozumět různým cílům a hypotézám v rámci UI.

- **Slabá UI (Weak AI):** Tato hypotéza tvrdí, že stroje mohou **jednat, jako by byly** inteligentní. Cílem je vytvářet užitečné nástroje, které řeší konkrétní problémy, a simulovat lidské mentální schopnosti, abychom lépe porozuměli lidské mysli. Většina současného výzkumu v UI spadá do této kategorie a nezajímá se o to, zda stroj skutečně "myslí".
- **Silná UI (Strong AI):** Tato hypotéza jde dál a tvrdí, že stroje, které jednají inteligentně, skutečně **myslí** a mají kognitivní stavy, jako je rozumění a vědomí. Nejde jen o simulaci, ale o skutečnou inteligenci.
- **Specifická UI (Specific AI):** Zaměřuje se na tvorbu programů pro řešení úzce vymezených, specifických úloh (např. hraní šachů, diagnostika nemocí).
- **Obecná UI (Artificial General Intelligence - AGI):** Cílem je tvorba programů pro obecné řešení úloh a inteligentní jednání srovnatelné s lidskou univerzálností. Takový systém by se dokázal adaptovat na širokou škálu úkolů, podobně jako člověk.

2. Rané Úvahy: René Descartes

Francouzský filozof René Descartes (17. století) položil základy mnoha otázek, které jsou relevantní i dnes.

- **Metodologický skepticizmus:** Descartes prosazoval myšlenku, že skrze systematické pochybování o všem lze dospět k pevným, nepochybným principům vědění.
- **Racionalizmus vs. Empirizmus:** Byl zastáncem racionalizmu, který považuje rozum za primární a rozhodující zdroj poznání, na rozdíl od empirizmu, který zdůrazňuje smyslovou zkušenost. Tvrdil, že smysly nás mohou klamat.
- **Dualizmus:** Zavedl myšlenku dualismu těla a mysli. Tělo je materiální, funguje jako stroj, zatímco mysl je nemateriální a je sídlem myšlení a vědomí.

Ve své publikaci "Rozprava o metodě" (1637) Descartes přímo zpochybnil, že by stroje mohly dosáhnout lidské úrovně myšlení. Argumentoval, že strojům chybí **univerzálnost myšlení**. Mohou sice některé úkoly dělat lépe než člověk, ale selžou v mnoha jiných, protože jejich schopnosti jsou úzce vymezené, zatímco lidský rozum je univerzální. Tím předjal mnoho moderních debat o specifické vs. obecné UI.

3. Turingův Test: Imitační Hra

Alan Turing (1950) navrhl nahradit vágní otázku "Mohou stroje myslet?" konkrétním behaviorálním testem.

3.1 Princip Imitační Hry

Tzv. **Turingův test** (původně "imitační hra") probíhá následovně:

1. Lidský **tazatel (C)** vede textovou konverzaci se dvěma neviditelnými protějšky.
2. Jeden protějšek je **člověk (B)**, druhý je **stroj (A)**.
3. Úkolem tazatele je na základě odpovědí určit, který z protějšků je stroj.
4. Cílem stroje je **oklamat tazatele**, aby si myslel, že je člověk.

Pokud stroj dokáže tazatele klamat s dostatečnou úspěšností (Turing navrhoval 30 % po pěti minutách konverzace), pak testem prošel.

3.2 Definice "Stroje"

Turing se ve svých úvahách omezil na **digitální počítače**, které popsal jako **stroje s diskrétním stavem**. Takový stroj se skládá ze tří hlavních částí:

- **Paměť (Storage):** Teoreticky nekonečný prostor pro ukládání informací a instrukcí.
- **Výkonná jednotka (Executive Unit):** Provádí jednotlivé operace.
- **Řídící jednotka (Control):** Zajišťuje správné provedení instrukcí v daném pořadí.

3.3 Námitky proti Možnosti Myslících Strojů

Turing ve svém článku předjímal a vyvracel řadu námitek:

- **Teologická námitka:** "Myšlení je funkce nesmrtelné duše, kterou Bůh dal člověku, ale ne strojům." Turing argumentuje, že nevidí důvod, proč by Bůh nemohl dát duši i stroji.
- **Námitka "strkání hlavy do písku":** "Představa myslících strojů je hrozná, doufejme tedy, že nemohou existovat." Toto je emocionální argument, nikoli logický.
- **Matematická námitka:** Odkazuje na Gödelovy věty o neúplnosti, které ukazují, že v každém dostatečně silném formálním systému existují pravdivá tvrzení, která nelze dokázat. Argument zní, že stroje jsou takovými systémy, ale lidé ne. Turing oponuje, že neexistuje důkaz, že by lidé nebyli omezeni stejným způsobem.
- **Argument z vědomí:** Tvrdí, že stroj skutečně nemyslí, dokud necítí emoce a není si vědom svých činů. Turing toto elegantně obchází tím, že pokud bychom tento argument brali vážně, nemohli bychom vědět, zda myslí i kdokoli jiný kromě nás samých.

4. Myšlenkový Experiment s Čínským Pokojem

Filozof John Searle (1980) představil vlivný argument proti Silné UI.

Představte si člověka, který **neumí čínsky**, zavřeného v místnosti. Tento člověk má k dispozici obrovskou knihu pravidel v jeho rodném jazyce (např. v angličtině). Pravidla mu říkají, jak manipulovat s čínskými znaky. Zvenku mu někdo podává lístky s otázkami v čínštině. Člověk uvnitř nerozumí ani otázkám, ani odpovědím, ale pečlivě sleduje pravidla v knize, která mu říkají, jaké znaky má na základě vstupních znaků napsat na výstupní lístek.

- **Z vnějšího pohledu:** Systém (člověk + kniha) se chová, jako by rozuměl čínsky. Odpovídá smysluplně na otázky a mohl by projít Turingovým testem.
- **Z vnitřního pohledu:** Člověk uvnitř místnosti se čínsky nenaučil. Stále jen manipuluje se symboly, kterým nerozumí.

Searlův závěr je, že **syntaxe (manipulace se symboly) není totéž co sémantika (skutečné rozumění)**. A protože počítačové programy nedělají nic jiného než manipulaci se symboly podle pravidel, nemohou samy o sobě nikdy dosáhnout skutečného rozumění.

5. Moderní Přístup: Racionální Agenti

Současná UI se z velké části odklonila od filozofických debat a zaměřila se na praktický koncept **racionálních agentů**.

- **Agent:** Cokoli, co vnímá své prostředí pomocí **senzorů** (sensors) a jedná v něm pomocí **aktuátorů** (actuators).
- **Prostředí (Environment):** Svět, ve kterém agent existuje a jedná.

- **Vjem (Percept):** Vstupní data, která agent v daném okamžiku získá ze senzorů.
- **Sekvence vjemů (Percept Sequence):** Kompletní historie všeho, co agent kdy vnímal.

Základní cyklus interakce je: agent vnímá svět, na základě vjemů provede akci, za kterou může (ale nemusí) dostat odměnu nebo trest, a cyklus se opakuje.

Funkce agenta (Agent Function) je matematická abstrakce, která mapuje jakoukoli sekvenci vjemů na akci. Program agenta (Agent Program) je konkrétní implementace této funkce.

Racionální agent je takový, který pro každou možnou sekvenci vjemů vybere akci, která **maximalizuje jeho očekávanou míru výkonnosti** (performance measure), na základě dosavadních vjemů a vestavěného vědění.

Tento přístup je pragmatický. Neptá se, zda agent "myslí", ale zda jedná optimálně vzhledem ke svému cíli.

Lekce 3: Řešení Úloh ve Stavovém Prostoru

Tento studijní průvodce se zaměřuje na formalizaci problémů a algoritmy prohledávání stavového prostoru, které tvoří základ mnoha systémů umělé inteligence.

1. Úvod do Řešení Úloh

Mnoho problémů v UI lze formalizovat jako hledání cesty od počátečního stavu k cílovému stavu. K tomu potřebujeme tři hlavní komponenty:

1. **Definice problému:** Co je počáteční situace.
2. **Definice cíle:** Jak vypadá požadovaný koncový stav.
3. **Akce:** Jaké operace můžeme provádět, abychom se přesouvali mezi stavy.

Příkladem mohou být Hanojské věže, kde cílem je přesunout disky z jedné tyčky na druhou za dodržení určitých pravidel (přesun jen jednoho disku, větší nelze na menší).

1.1 Abstraktní Reprezentace

Pro řešení problémů používáme dva klíčové principy:

- **Generalizace:** Zobecnění skupiny podobných problémů nalezením jejich společných rysů. Místo řešení "cesty z Aradu do Bukurešti" řešíme obecný problém "nalezení cesty mezi dvěma městy".
- **Abstrakce:** Zjednodušení problému zanedbáním nedůležitých detailů. Například při plánování trasy autem zanedbáváme detaily jako rádio, počasí nebo přesné pohyby volantem.

1.2 Stavový Prostor

Stavový prostor je abstraktní model úlohy, který se skládá z:

- **Stavů:** Situace, které mohou v problému nastat.
- **Akcí (přechodů):** Operace, které nás přesouvají z jednoho stavu do druhého.

Stavový prostor může být definován:

- **Explicitně:** Všechny stavy a přechody jsou přímo dány (např. mapa pro navigaci).
- **Implicitně:** Stavy a přechody jsou generovány podle pravidel (např. šachy, kde stavy vznikají platnými tahy).

2. Formální Definice Problému

Problém můžeme formálně definovat pomocí pěti komponent:

1. **Počáteční stav (Initial State):** Stav, ve kterém se agent na začátku nachází (např. `In(Arad)`).
2. **Akce (Actions):** Popis možných akcí, které jsou agentovi k dispozici. Funkce `ACTIONS(s)` vrací množinu akcí proveditelných ve stavu `s`.
3. **Přechodový model (Transition Model):** Popis toho, co jednotlivé akce dělají. Funkce `RESULT(s, a)` vrací stav, který je výsledkem provedení akce `a` ve stavu `s`. Společně s počátečním stavem a akcemi definuje **stavový prostor**.
4. **Cílový test (Goal Test):** Určuje, zda daný stav je cílovým stavem.
5. **Cena cesty (Path Cost):** Funkce, která přiřazuje číselnou cenu každé cestě. Obvykle je to součet cen jednotlivých kroků.

Řešením problému je sekvence akcí, která vede z počátečního stavu do cílového stavu. **Optimální řešení** je řešení s nejnižší cenou cesty.

3. Algoritmy Prohledávání

Algoritmy prohledávání systematicky zkoumají stavový prostor, aby nalezly řešení. Pracují s **prohledávacím stromem**, kde:

- **Kořen** je počáteční stav.
- **Větve** jsou akce.
- **Uzly** odpovídají stavům.

Algoritmy udržují:

- **Frontu (Fringe / Open List):** Seznam dosud nenavštívených (nerozvinutých) uzlů.
- **Prozkoumanou množinu (Explored Set / Closed List):** Seznam již navštívených (rozvinutých) uzlů, aby se předešlo cyklům a redundantním cestám.

Základní operací je **rozvinutí (expanze)** uzlu, což znamená aplikaci všech možných akcí na daný stav a přidání výsledných uzlů (následníků) do fronty.

3.1 Slepé (Neinformované) Algoritmy

Tyto algoritmy nemají žádné další informace o problému kromě jeho definice. Všechny následníky považují za rovnocenné.

- **Prohledávání do šířky (Breadth-First Search - BFS):**
  - Používá frontu typu **FIFO (First-In, First-Out)**.
  - Vždy rozvíjí nejméně hluboký uzel.
  - **Vlastnosti:**
    - **Úplnost:** Ano, vždy najde řešení, pokud existuje.
    - **Optimalita:** Ano, pokud mají všechny kroky stejnou cenu.
    - **Složitost (časová i paměťová):** O(b^d), kde `b` je faktor větvení a `d` je hloubka řešení. Je velmi náročný na paměť.
- **Prohledávání s uniformní cenou (Uniform-Cost Search - UCS):**
  - Rozšíření BFS, které řeší ohodnocené přechody.
  - Fronta je **prioritní fronta** uspořádaná podle ceny cesty `g(n)`.

    - Vždy rozvíjí uzel s nejnižší cenou cesty.
    - **Vlastnosti:**
        - **Úplnost a Optimalita:** Ano, pokud jsou ceny kroků nezáporné.
        - Složitost závisí na ceně optimálního řešení C*.

- **Prohledávání do hloubky (Depth-First Search - DFS):**
    - Používá frontu typu **LIFO (Last-In, First-Out)**, často implementováno rekurzivně.
    - Vždy rozvíjí nejhlubší uzel.
    - **Vlastnosti:**
        - **Úplnost:** Ne, může uvíznout v nekonečné větvi.
        - **Optimalita:** Ne.
        - **Paměťová složitost:** O(bm), kde `m` je maximální hloubka. Je velmi úsporný na paměť.

- **Prohledávání s omezenou hloubkou (Depth-Limited Search - DLS):**
    - Varianta DFS, která prohledává jen do předem dané hloubky `l`. Řeší problém nekonečných větví, ale je neúplný, pokud `l < d`.

- **Iterativní prohlubování (Iterative Deepening Search - IDS):**
    - Kombinuje výhody BFS a DFS.
    - Opakovaně spouští DLS s postupně se zvyšující hloubkou (0, 1, 2, ...).
    - **Vlastnosti:** Stejné jako BFS (úplný, optimální), ale s paměťovou náročností DFS.

- **Obousměrné prohledávání (Bidirectional Search):**
    - Prohledává současně od počátečního stavu dopředu a od cílového stavu dozadu.
    - Řešení je nalezeno, když se obě fronty protnou.
    - Výrazně snižuje složitost na O(b^(d/2)), ale je náročné na implementaci (vyžaduje schopnost prohledávat "pozpátku") a paměť.

3.2 Informované (Heuristické) Algoritmy

Tyto algoritmy používají **heuristickou funkci** `h(n)`, která odhaduje cenu cesty z uzlu `n` do nejbližšího cíle. Heuristika je klíčová pro efektivní řešení složitých problémů.

- **Uspořádané prohledávání (Greedy Best-First Search):**
    - Rozvíjí uzel, který se zdá být nejblíže k cíli. Používá `f(n) = h(n)`.
    - Je "chamtivý", protože se snaží co nejrychleji přiblížit k cíli, i když to může vést do slepé uličky.
    - Není úplný ani optimální.

- **Algoritmus A***:
    - Nejznámější a nejrozšířenější informovaný algoritmus.
    - Hodnotí uzly kombinací ceny cesty od počátku `g(n)` a heuristického odhadu do cíle `h(n)`.
    - **Vyhodnocovací funkce:** `f(n) = g(n) + h(n)`
    - Používá prioritní frontu seřazenou podle `f(n)`.
    - **Vlastnosti:**
        - **Úplnost a Optimalita:** Ano, pokud je heuristika **přípustná (admissible)**, tzn. nikdy nepřecení skutečnou cenu do cíle (`h(n) <= h*(n)`).

- **Gradientní prohledávání (Hill-Climbing):**
    - Lokální prohledávací metoda, která neudržuje celý prohledávací strom.
    - Začíná v nějakém stavu a iterativně se posouvá do nejlepšího sousedního stavu, dokud nedosáhne vrcholu ("peak"), ze kterého již nevede cesta vzhůru.
    - Může uvíznout v **lokálním maximu**.

3.3 Náhodné Algoritmy

Tyto algoritmy zavádějí do prohledávání prvek náhody, často proto, aby se vymanily z lokálních extrémů.

- **Simulované žíhání (Simulated Annealing):**
    - Inspirováno procesem žíhání kovů.
    - Podobné gradientnímu prohledávání, ale s určitou pravděpodobností přijímá i horší řešení.
    - Tato pravděpodobnost závisí na "teplotě" `T`, která se v čase postupně snižuje. Na začátku (vysoká teplota) je pravděpodobnost přijetí horšího řešení vyšší, což umožňuje "přeskočit" lokální minima. Ke konci (nízká teplota) se algoritmus chová spíše jako gradientní prohledávání.
- **Genetické algoritmy (Genetic Algorithms - GA):**
    - Inspirováno biologickou evolucí.
    - Pracuje s **populací** stavů (jedinců).
    - V každém kroku vybírá jedince na základě jejich "zdatnosti" (fitness) a vytváří novou generaci pomocí operací **křížení (crossover)** a **mutace (mutation)**.

4. General Problem Solver (GPS)

Historicky významný algoritmus (Newell & Simon, 1961), který řešil úlohy metodou postupného rozkladu na podúlohy. Pracoval na základě výpočtu **diferencí** mezi aktuálním a cílovým stavem a snažil se tuto diferenci zmenšit aplikací vhodných operátorů.

Lekce 4: Splňování Podmínek (Constraint Satisfaction Problems - CSP)

Tento studijní průvodce se zabývá speciální třídou problémů, které lze efektivněji řešit využitím jejich vnitřní struktury – proměnných a omezujících podmínek.

1. Co je Úloha na Splňování Podmínek?

Na rozdíl od předchozích metod, které považovaly stavy za nedělitelné "černé skříňky", CSP přistupuje ke stavům strukturovaně. **Úloha na splňování podmínek (Constraint Satisfaction Problem - CSP)** je tvořena třemi hlavními komponentami:

1. **X: Množina proměnných (Variables)**
    - Konečná množina proměnných $\{X_1, \ldots, X_n\}$, které popisují stav světa. V problému barvení mapy jsou proměnnými jednotlivé regiony (státy).
2. **D: Množina domén (Domains)**

- Pro každou proměnnou X_i je definován její **definiční obor** D_i, což je množina povolených hodnot $\{v_1, ..., v_k\}$. V problému barvení mapy je doménou pro každý region množina $\{\text{červená, zelená, modrá}\}$.

3. **C: Množina podmínek (Constraints)**
   - Podmínky (omezení), které specifikují povolené kombinace hodnot pro proměnné. Podmínka je tvořena párem $\langle scope, rel \rangle$, kde $scope$ je n-tice proměnných, kterých se podmínka týká, a rel je relace definující přípustné hodnoty. Například podmínka, že dva sousední regiony nesmí mít stejnou barvu, je $SA \neq WA$.

Řešením CSP je takové přiřazení hodnot všem proměnným, aby byly splněny všechny podmínky. Formálněji řečeno:

- **Přiřazení (Assignment):** Přiřazení hodnot některým nebo všem proměnným.
- **Konzistentní (nebo legální) přiřazení:** Přiřazení, které neporušuje žádnou podmínku.
- **Úplné přiřazení:** Přiřazení, ve kterém mají hodnotu všechny proměnné.
- **Řešení (Solution):** Konzistentní a zároveň úplné přiřazení.

1.1 Příklady CSP

- **Barvení mapy:** Proměnné jsou regiony, domény jsou barvy, podmínky zakazují stejnou barvu pro sousední regiony.
- **8 královen (8 Queens):** Proměnné jsou sloupce, domény jsou řádky (1-8), podmínky zakazují, aby se dvě královny ohrožovaly.
- **Sudoku:** Proměnné jsou políčka (81), domény jsou číslice (1-9), podmínky vyžadují unikátnost číslic v každém řádku, sloupci a 3x3 boxu.
- **Kryptoaritemetické hádanky (např. SEND + MORE = MONEY):** Proměnné jsou písmena, domény číslice (0-9), podmínky jsou algebraické rovnice a unikátnost přiřazení.
- **Reálné úlohy:** Tvorba univerzitních rozvrhů, plánování výroby, alokace zdrojů.

2. Způsoby Řešení CSP

2.1 Matematické Metody (Operační výzkum)

Operační výzkum se snaží najít optimální řešení problému při daných omezeních. Problém je často modelován pomocí matematických rovnic a nerovnic. Příkladem je **lineární programování**, kde jsou podmínky i cílová funkce lineární.

2.2 Booleovská Splnitelnost (SAT)

Problém CSP lze převést na problém **Booleovské splnitelnosti (SAT)**.

1. Podmínky se převedou na logické formule v konjunktivní normální formě (CNF).
2. Hledá se takové ohodnocení (přiřazení $true$ / $false$) proměnných, aby byla výsledná formule pravdivá. Tento problém je NP-úplný, ale existují pro něj vysoce optimalizované řešiče (např. založené na **DPLL algoritmu**).

2.3 Využití Stavového Prostoru

Nejpřirozenějším přístupem v rámci UI je formulovat CSP jako problém prohledávání stavového prostoru.

- **Stav:** Částečné přiřazení hodnot proměnným.
- **Počáteční stav:** Prázdné přiřazení (žádná proměnná nemá hodnotu).
- **Akce:** Přiřazení hodnoty jedné z dosud nepřiřazených proměnných.

- **Cílový test:** Zjišťuje, zda je přiřazení úplné a konzistentní.

Standardní prohledávací algoritmy (jako DFS) by byly neefektivní kvůli obrovskému faktoru větvení. Proto se pro CSP používá specializovaný algoritmus **zpětného prohledávání (Backtracking Search)**.

Algoritmus Backtrackingu

Backtracking je forma prohledávání do hloubky, která postupně přiřazuje hodnoty jednotlivým proměnným.

1. Vybere se dosud nepřiřazená proměnná.
2. Postupně se zkouší přiřadit jí hodnoty z její domény.
3. Pro každou hodnotu se zkontroluje, zda je **konzistentní** s dosud přiřazenými hodnotami.
    - Pokud **ano**, pokračuje se rekurzivně s další proměnnou.
    - Pokud **ne**, zkusí se další hodnota.
4. Pokud pro danou proměnnou dojdou hodnoty, algoritmus se **vrátí zpět (backtrack)** k předchozí proměnné a zkusí pro ni jinou hodnotu.

Tento základní algoritmus lze výrazně zefektivnit pomocí **šíření podmínek (constraint propagation)**, což je forma inference, a heuristik pro výběr proměnných a hodnot.

Lekce 5: Teorie Her a Prohledávání v Konfliktních Situacích

Tento studijní průvodce se věnuje matematické teorii her, která analyzuje konfliktní rozhodovací situace, a algoritmům, které umožňují umělé inteligenci hrát hry proti protivníkovi.

1. Matematická Teorie Her

Teorie her je disciplína aplikované matematiky, která analyzuje situace, kde dochází ke střetu zájmů. Jejím cílem je odpovědět na dvě základní otázky:

1. Která strategie (rozhodnutí) je optimální?
2. Jak tuto optimální strategii nalézt?

Základním předpokladem je **racionalita hráčů** – každý hráč se snaží maximalizovat svůj vlastní zisk (nebo minimalizovat ztrátu).

1.1 Formální Popis Hry

Hra je formálně popsána pomocí:

- **Množiny hráčů**.
- **Množin strategií** pro každého hráče (jaké akce může provést).
- **Výher (užitku)** pro každého hráče při dané kombinaci strategií.

1.2 Typy Her

Hry lze klasifikovat podle několika kritérií:

- **Podle počtu hráčů:** Dvouhráčové, vícehráčové.
- **Podle typu výhry:**

    - **Hry s konstantním součtem (Constant-Sum):** Součet výher všech hráčů je vždy stejný. Speciálním případem jsou **hry s nulovým součtem (Zero-Sum)**, kde co jeden hráč získá, druhý ztratí (např. šachy, go).
    - **Hry s nekonstantním součtem:** Hráči mohou spolupracovat, aby dosáhli oboustranně výhodného výsledku.
- **Podle míry informace:**
    - **S úplnou informací (Perfect Information):** Všichni hráči znají kompletní stav hry (např. šachy).
    - **S neúplnou informací (Imperfect Information):** Hráči neznají všechny informace (např. karetní hry jako poker, kde neznáte karty protihráče).
- **Podle prvku náhody:**
    - **Deterministické:** Výsledek tahu je plně určen rozhodnutím hráče (šachy).
    - **Stochastické (s prvkem náhody):** Výsledek závisí i na náhodě (např. hod kostkou v backgammonu).

V kontextu AI se nejčastěji zabýváme **deterministickými, dvouhráčovými hrami s úplnou informací a nulovým součtem**.

2. Hra jako Úloha Prohledávání

Hru můžeme formalizovat jako problém prohledávání, který je definován:

1. **Počátečním stavem (S_0):** Jak je hra na začátku postavena.
2. **Funkcí následníků (ACTIONS):** Vrací množinu legálních tahů v daném stavu.
3. **Přechodovým modelem (RESULT):** Definuje výsledek tahu.
4. **Testem konce (TERMINAL-TEST):** Zjišťuje, zda hra skončila.
5. **Užitkovou funkcí (UTILITY):** Přiřazuje konečnému stavu číselnou hodnotu z pohledu jednoho hráče (výhra, prohra, remíza).

Cílem agenta (hráče) je najít **optimální strategii** – takovou, která vede k výsledkům, jež jsou nejlepší možné proti bezchybnému protivníkovi.

3. Algoritmus Minimax

Minimax je základní algoritmus pro nalezení optimálního tahu. Vychází z předpokladu, že oba hráči hrají optimálně. Hráče označujeme jako **MAX** (snaží se maximalizovat užitek) a **MIN** (snaží se minimalizovat užitek z pohledu MAXe).

Princip:

- Pro každý uzel v herním stromu se vypočítá **minimaxová hodnota**.
- **Pro MAXe** je hodnota uzlu maximem hodnot jeho následníků.
- **Pro MINa** je hodnota uzlu minimem hodnot jeho následníků.
- **Pro koncové uzly** je hodnota dána užitkovou funkcí.

Algoritmus rekurzivně prohledává herní strom do hloubky, spočítá hodnoty koncových stavů a poté tyto hodnoty "propaguje" nahoru stromem (tzv. "backing up"). Kořenový uzel (aktuální stav) pak získá hodnotu, která představuje nejlepší dosažitelný výsledek pro MAXe, a tah, který k němu vede, je optimální.

Složitost: Minimax prohledává celý strom do hloubky `m` s faktorem větvení `b`, takže časová složitost je **O(b^m)**, což je pro většinu reálných her (jako šachy) nepraktické.

4. Alfa-Beta Prořezávání (Alpha-Beta Pruning)

Alfa-beta prořezávání je optimalizace algoritmu Minimax, která dosahuje stejného výsledku, ale prohledává výrazně menší část herního stromu.

Princip: Algoritmus si během prohledávání udržuje dvě hodnoty:

- **α (alfa):** Nejlepší (nejvyšší) hodnota, kterou může **MAX** zaručeně dosáhnout na cestě od kořene k aktuálnímu uzlu.
- **β (beta):** Nejlepší (nejnižší) hodnota, kterou může **MIN** zaručeně dosáhnout na cestě od kořene k aktuálnímu uzlu.

Prořezávání nastane, když algoritmus zjistí, že daná větev nemůže ovlivnit konečné rozhodnutí:

1. **U MIN uzlu:** Pokud hodnota některého z jeho následníků je **menší nebo rovna α**, můžeme tuto větev prořídnout. Důvod: MAX by nikdy nedovolil, aby se hra do této větve dostala, protože už má k dispozici jinou cestu s garantovanou hodnotou α.
2. **U MAX uzlu:** Pokud hodnota některého z jeho následníků je **větší nebo rovna β**, můžeme tuto větev prořídnout. Důvod: MIN by nikdy nedovolil, aby se hra do této větve dostala, protože už má k dispozici jinou cestu s garantovanou hodnotou β.

Efektivita:

- Při **optimálním seřazení tahů** (kdy jsou nejdříve zkoumány nejlepší tahy) se časová složitost snižuje na **O(b^(m/2))**. To v praxi znamená, že alfa-beta dokáže prohledat zhruba dvakrát hlubší strom než Minimax ve stejném čase.
- Při náhodném pořadí je složitost zhruba O(b^(3m/4)).

Alfa-beta prořezávání neovlivňuje výsledek – vždy najde stejný optimální tah jako Minimax.

Lekce 6: Plánování a Rozvrhování

Tento studijní průvodce se zaměřuje na rozdíly a spojitosti mezi plánováním a rozvrhováním a představuje klíčové přístupy k řešení plánovacích problémů v umělé inteligenci.

1. Plánování vs. Rozvrhování

Ačkoliv jsou tyto dva pojmy úzce propojené, je důležité chápat jejich odlišné role v rozhodovacím procesu.

1.1 Úloha Plánování (Planning)

- **Co řeší? Jaké akce** jsou potřeba pro dosažení cílů.
- **Vstup:** Počáteční stav světa, popis dostupných akcí a jejich efektů, a požadovaný cílový stav.
- **Výstup:** Sekvence (nebo částečně uspořádaná množina) akcí – **plán**.
- **Zaměření:** Plánování se primárně soustředí na **kauzální vztahy** mezi akcemi. Řeší, které akce vybrat a v jakém logickém pořadí je provést, aby byly splněny předpoklady dalších akcí a nakonec i samotný cíl. **Nezabývá se konkrétním časem a zdroji.**

1.2 Úloha Rozvrhování (Scheduling)

- **Co řeší?** Jak naplánované aktivity **alokovat na zdroje v čase**.
- **Vstup:** Skupina aktivit (často výstup z plánovače) a dostupné zdroje s jejich omezeními (kapacita, časová dostupnost).

- **Výstup: Rozvrh**, který specifikuje, kdy a kde se každá aktivita provede.
- **Zaměření:** Cílem je optimalizovat využití zdrojů, minimalizovat celkový čas, vyhnout se konfliktům a dodržet časové termíny.

Vztah: Typicky **plánování předchází rozvrhování**. Někdy je však výhodné řešit obě úlohy současně, například když existuje mnoho možných plánů, ale jen málo z nich má přípustný rozvrh.

2. Jazyk pro Definici Plánování (PDDL)

Pro formalizaci plánovacích problémů se používá **Planning Domain Definition Language (PDDL)**. Umožňuje definovat:

- **Počáteční stav:** Konjunkce základních, bezfunkčních a pozitivních atomů (fluentů), např. `At(C1, SFO) ∧ At(P1, SFO)`. Platí zde předpoklad uzavřeného světa (co není uvedeno, je nepravdivé).
- **Cíl:** Konjunkce literálů (mohou být i negativní), které popisují požadovaný stav.
- **Akce (Action Schemas):** Šablony pro akce, které obsahují:
  - **Název a parametry:** Např. `Fly(p, from, to)`.
  - **Předpoklady (PRECOND):** Konjunkce literálů, které musí platit, aby akce mohla být provedena.
  - **Efekty (EFFECT):** Konjunkce literálů, které popisují, jak se stav světa po provedení akce změní. Literály, které se mají stát pravdivými, jsou v **add listu**, a ty, které se mají stát nepravdivými, v **delete listu**.

3. Přístupy k Plánování

3.1 Plánování jako Prohledávání Stavového Prostoru

Toto je nejběžnější přístup, kde:

- **Uzly** odpovídají stavům světa.
- **Hrany** odpovídají akcím.
- **Cílem** je nalézt cestu z počátečního stavu do cílového.

a) Dopředné prohledávání (Progression Planning)

- **Princip:** Začíná se v počátečním stavu a aplikují se akce, dokud se nedosáhne cílového stavu.
- **Výhody:** Intuitivní, umožňuje použití velmi silných, doménově nezávislých heuristik.
- **Nevýhody:** Může prohledávat velké množství nerelevantních akcí (např. v problému nákupu knihy existují miliony akcí `Buy(isbn)`).
- **Heuristiky:** Úspěch této metody závisí na kvalitních heuristikách, které odhadují vzdálenost do cíle. Tyto heuristiky se často získávají z **relaxovaných problémů** (např. ignorováním negativních efektů akcí).

b) Zpětné prohledávání (Regression Planning)

- **Princip:** Začíná se u cíle a postupuje se "dozadu" aplikací inverzních akcí. Stav zde není jeden konkrétní stav, ale **množina stavů** popsaná konjunkcí literálů.
- **Výhody:** Prohledává pouze **relevantní akce**, tedy takové, které přispívají k dosažení cíle. To výrazně snižuje faktor větvení.
- **Nevýhody:** Je složitější a obtížněji se pro něj definují dobré heuristiky.

3.2 Plánování jako Splňování Podmínek (CSP) nebo Booleovská Splnitelnost (SAT)

Plánovací problém pro pevně danou délku plánu `k` lze převést na:

- **CSP:** Proměnné mohou reprezentovat akce v každém časovém kroku nebo fluenty v každém stavu. Podmínky zajišťují logickou konzistenci (efekty akcí, splnění předpokladů).
- **SAT (SATPLAN):** Problém se převede na velkou booleovskou formuli. Pokud je formule splnitelná, model (přiřazení pravdivostních hodnot) kóduje platný plán.
  - **Postup:**
    1. Vytvoří se propozice pro každý fluent a každou akci v každém časovém kroku až do `k`.
    2. Přidají se axiomy: počáteční stav, cíl v čase `k`, a tzv. **successor-state axioms**, které definují, jak se stav mění mezi kroky.
    3. Výsledná formule se předá SAT-solveru.
  - Tento přístup je překvapivě efektivní díky moderním a vysoce optimalizovaným SAT-solverům.

3.3 Plánování v Prostoru Plánů (Partial-Order Planning)

- **Princip:** Místo prohledávání stavů se prohledává prostor **částečně uspořádaných plánů**.
- **Začátek:** Prázdný plán, který obsahuje jen počáteční stav a cíl.
- **Kroky:** Algoritmus iterativně identifikuje **nedostatky (flaws)** v plánu (např. nesplněný předpoklad) a opravuje je přidáním nové akce nebo uspořádávacího omezení (`A musí být před B`).
- **Výhody:** Flexibilní, řídí se principem **nejmenšího závazku (least commitment)** – odkládá rozhodnutí o přesném pořadí akcí, dokud to není nutné. Dobře řeší problémy s nezávislými podproblémy.
- **Nevýhody:** Složitější na implementaci a v současnosti méně výkonný než nejlepší dopředné plánovače.

3.4 Plánovací Graf (Planning Graph)

Plánovací graf je datová struktura používaná pro extrakci plánu nebo pro odhad heuristik.

- **Struktura:** Orientovaný graf organizovaný do střídajících se vrstev **stavů (literálů)** S_i a **akcí** A_i.
- **Expanze grafu:** Graf se postupně staví od S_0 (počáteční stav). A_i obsahuje všechny akce, jejichž předpoklady jsou v S_i. S_{i+1} obsahuje všechny efekty akcí z A_i.
- **Mutexy:** Graf zaznamenává páry akcí nebo literálů, které se vzájemně vylučují (nemohou nastat současně).
- **Využití:**
  1. **Heuristika:** Počet vrstev potřebných k dosažení cílových literálů (bez mutexů) je velmi dobrou (a přípustnou) heuristikou pro dopředné prohledávání. Toto je základ úspěchu plánovače **FF (Fast Forward)**.
  2. **Algoritmus GRAPHPLAN:** Přímo hledá řešení v plánovacím grafu pomocí zpětného prohledávání. Pokud řešení nenajde, rozšíří graf o další vrstvu a zkusí to znovu.

Lekce 7: Strojové Učení

Tento studijní průvodce se věnuje základním konceptům strojového učení, což je proces, při kterém agent zlepšuje své chování na základě zkušeností.

1. Co je Učení?

Učení je proces, při kterém agent zlepšuje svůj výkon v budoucích úlohách na základě pozorování světa. Je to klíčová vlastnost inteligentních systémů.

1.1 Metody Učení

Existuje mnoho způsobů, jak se agent může učit:

- **Učení se zapamatováním (Rote Learning):** Agent si ukládá nové poznatky (např. telefonní číslo) bez jejich dalšího zpracování.
- **Učení se z instrukcí (Learning from Instruction):** Agent přijímá explicitní pokyny od učitele.
- **Učení se z analogie (Learning by Analogy):** Agent aplikuje znalosti z jedné domény na novou, podobnou doménu.
- **Učení se na základě vysvětlení (Explanation-Based Learning):** Agent využívá existující znalosti k vysvětlení jednoho příkladu a z toho odvodí obecné pravidlo.
- **Učení se z příkladů (Inductive Learning):** Agent odvozuje obecné pravidlo z množiny konkrétních příkladů. Toto je nejběžnější forma strojového učení.
- **Učení se pozorováním a objevováním (Unsupervised Learning):** Agent hledá vzory a struktury v datech bez explicitní zpětné vazby.

1.2 Zpětná Vazba v Procesu Učení

Typ zpětné vazby definuje tři hlavní kategorie učení:

1. **Učení s učitelem (Supervised Learning):** Agent dostává trénovací sadu příkladů, kde každý vstup je spárován se správným výstupem (klasifikací). Učitel poskytuje "správné odpovědi".
2. **Učení bez učitele (Unsupervised Learning):** Agent dostává pouze vstupy a snaží se v nich najít skryté vzory. Typickým příkladem je **shlukování (clustering)**.
3. **Zpětnovazební učení (Reinforcement Learning):** Agent se učí na základě odměn a trestů. Po sérii akcí dostane zpětnou vazbu, která mu říká, jak dobře si vedl, ale neříká mu, která konkrétní akce byla správná nebo špatná.

2. Empirické Učení z Dat (Induktivní Učení)

Cílem je na základě **trénovací sady** příkladů `(x, y)` nalézt hypotézu (funkci) `h`, která dobře aproximuje neznámou cílovou funkci `f`. Očekáváme, že hypotéza bude dobře fungovat i na nových, dosud neviděných datech (tzv. **testovací sadě**).

2.1 Přeučení (Overfitting) vs. Podučení (Underfitting)

- **Přeučení (Overfitting):**
  - Nastává, když je model příliš složitý a "zapamatuje si" trénovací data včetně šumu, místo aby se naučil obecný vzor.
  - Projevuje se tak, že chyba na trénovacích datech je velmi nízká, ale chyba na testovacích datech je vysoká.
  - Typické pro složité modely (např. hluboké rozhodovací stromy, neuronové sítě s mnoha neurony).
- **Podučení (Underfitting):**
  - Nastává, když je model příliš jednoduchý a nedokáže zachytit základní strukturu dat.
  - Chyba je vysoká jak na trénovacích, tak na testovacích datech.

    - Typické pro příliš jednoduché modely (např. lineární regrese na nelineárních datech).

Cílem je najít model, který je "tak akorát" složitý a dobře **generalizuje**.

3. Paradigmy Učení

3.1 Učení jako Prohledávání

Učení lze chápat jako prohledávání **prostoru hypotéz** s cílem nalézt hypotézu, která nejlépe odpovídá datům.

- **Příklad:** Algoritmus pro učení **rozhodovacích stromů** prohledává prostor možných stromů a v každém kroku se chamtivě rozhoduje, kterým atributem data rozdělit, aby maximalizoval informační zisk. Hledá se optimální struktura i parametry modelu.

3.2 Učení jako Aproximace (Optimalizace)

V tomto pohledu máme předem danou strukturu modelu a snažíme se najít jeho optimální parametry.

- **Příklad:** U **neuronových sítí** je struktura (počet vrstev a neuronů) často daná a učení spočívá v nalezení optimálních vah w tak, aby se minimalizovala chybová funkce (loss function) na trénovacích datech. To se obvykle dělá pomocí **gradientního sestupu (gradient descent)**.

4. Neuronové Sítě

Neuronová síť je výpočetní systém inspirovaný strukturou a funkcí biologických neuronových sítí.

- Skládá se z propojených **výpočetních jednotek (neuronů)** uspořádaných do vrstev.
- Každé spojení má **váhu** w, která určuje sílu signálu.
- Každý neuron j spočítá vážený součet svých vstupů $in_j = \Sigma_i w_{ij} a_i$ a aplikuje na něj nelineární **aktivační funkci** g (např. sigmoid nebo ReLU), čímž získá svůj výstup $a_j = g(in_j)$.
- Učení sítě probíhá úpravou vah (např. algoritmem **zpětného šíření chyby - backpropagation**), aby se minimalizoval rozdíl mezi predikovaným a skutečným výstupem.

5. Teoretické Koncepty Učení

5.1 PAC Teorie (Probably Approximately Correct Learning)

Tato teorie poskytuje formální rámec pro analýzu, zda se model dokáže efektivně naučit a generalizovat.

- Zaručuje, že s pravděpodobností alespoň $1-\delta$ bude chyba naučené hypotézy menší než ε.
- Poskytuje odhady, kolik trénovacích vzorků je potřeba pro dosažení těchto záruk.
- Pomáhá formalizovat kompromis mezi složitostí modelu (měřenou např. **VC dimenzí**) a potřebným množstvím dat, čímž pomáhá předcházet přeučení.

5.2 Teorém "No Free Lunch"

- **Znění:** Neexistuje žádný univerzálně nejlepší algoritmus strojového učení.
- **Vysvětlení:** Pokud je algoritmus A lepší než algoritmus B na jedné třídě problémů, pak musí existovat jiná třída problémů, na které je algoritmus B lepší než A.
- **Důsledek:** Výběr algoritmu musí být přizpůsoben konkrétnímu problému a datům.

5.3 Teorém Ošklivého Káčátka

- **Znění:** Z čistě formálního hlediska jsou si jakékoliv dva objekty stejně podobné (nebo nepodobné), pokud neuplatníme nějaký **bias** (předpoklad) o tom, které atributy jsou důležitější.
- **Vysvětlení:** Počet atributů, ve kterých se dva objekty shodují, je konstantní, pokud uvážíme všechny možné (i velmi absurdní) atributy.
- **Důsledek:** Jakákoliv klasifikace nebo učení na základě podobnosti implicitně předpokládá nějaký bias, který nám říká, co je a co není "důležitý" rys.

Lekce 8: Použití Znalostí v Učení

Tento průvodce se zaměřuje na to, jak mohou agenti využívat již existující (apriorní) znalosti k urychlení a zefektivnění procesu učení. Místo toho, aby se učili od nuly, mohou stavět na tom, co již vědí.

1. Klasifikace Znalostí

Znalosti můžeme dělit podle různých kritérií, což nám pomáhá pochopit jejich povahu a způsob, jakým je lze reprezentovat a využít.

1.1 Podle Formalizovatelnosti

- **Explicitní znalosti:** Jsou to znalosti, které jsou plně formalizované, artikulované a snadno sdělitelné. Příkladem jsou matematické vzorce nebo pravidla v manuálu.
- **Implicitní znalosti:** Tyto znalosti jsou primárně skryté v datech a nejsou přímo formulovány. Například vzory v nákupním chování zákazníků, které odhalí algoritmus strojového učení.
- **Tacitní znalosti:** Jde o nevědomé a těžko sdělitelné znalosti, které jsou skryty v myslích expertů. Příkladem je intuice zkušeného lékaře při stanovování diagnózy. Tacitní znalosti je velmi obtížné formalizovat.

1.2 Podle Obsahu

- **Deklarativní znalosti:** Popisují, co platí, tedy fakta o světě. Například: "Všechny kočky jsou savci." Jsou to výroky o stavu věcí.
- **Procedurální znalosti:** Popisují, jak postupovat, tedy návody a procesy. Například: "Postup pro výměnu pneumatiky." Jsou to znalosti o tom, jak provádět akce.

2. Požadavky na Znalosti

Aby byly znalosti v systémech umělé inteligence efektivně využitelné, měly by splňovat několik klíčových požadavků:

- **Transparentnost:** Znalosti by měly být srozumitelné a jejich fungování by mělo být snadno pochopitelné pro člověka.
- **Modularita:** Mělo by být možné snadno přidávat, odebírat nebo měnit jednotlivé části znalostní báze bez narušení celého systému.
- **Modifikovatelnost:** Systém by měl umožňovat snadnou aktualizaci a úpravu znalostí.
- **Užitečnost:** Znalosti musí být relevantní pro řešení daného problému a přispívat k dosažení cílů agenta.

3. Reprezentace Znalostí v AI

Existuje několik zavedených formalismů pro reprezentaci znalostí v systémech umělé inteligence.

3.1 Predikátová Logika

- Jedná se o rozšíření výrokové logiky, které umožňuje pracovat s objekty, jejich vlastnostmi a vztahy.
- Základními stavebními kameny jsou **predikáty** (vyjadřují vlastnosti nebo vztahy, např. `JeClovek(x)`), **funkce** (mapují objekty na jiné objekty, např. `Matka(x)`), **logické spojky** (∧, ∨, ¬, ⇒) a **kvantifikátory** (∀ – obecný, ∃ – existenční).
- Umožňuje formulovat složité a obecné výroky o světě, což je klíčové pro pokročilé usuzování.

3.2 Sémantické Sítě

- Grafová reprezentace, která popisuje realitu jako soubor **objektů (uzlů)** a **relací (hran)** mezi nimi.
- Například uzly "Pes" a "Savec" mohou být spojeny hranou s popiskem "JeDruh" (SubsetOf). Uzel "Fido" a "Pes" mohou být spojeny hranou "JeInstance" (MemberOf).
- Hlavní výhodou je intuitivní vizualizace a efektivní mechanismus pro **dědičnost** (inheritance), kdy objekty dědí vlastnosti od svých nadřazených kategorií.

3.3 Rámce (Frames)

- Komplexní datová struktura, která se stala inspirací pro objektově orientované programování (OOP).
- Rámec reprezentuje typický objekt nebo koncept (např. rámec pro "Pokoj v hotelu").
- Obsahuje:
  - **Data (sloty):** Atributy, které popisují objekt (např. `Počet_lůžek`, `Má_koupelnu`).
  - **Meta-data:** Informace o datech (např. výchozí hodnota pro `Počet_lůžek` je 2).
  - **Procedury (démoni):** Funkce, které se aktivují při čtení nebo zápisu do slotu (např. procedura `if-added` pro přepočet ceny).

3.4 Pravidla (Rules)

- Znalosti jsou reprezentovány ve formě **IF-THEN** pravidel (podmínka-akce).
- Například: `IF (teplota < 18°C) AND (topení_je_vypnuté) THEN (zapni_topení)`.
- Tento přístup je základem **expertních systémů** (rule-based systems), kde se usuzování provádí řetězením těchto pravidel.

3.5 Případy (Cases)

- Znalosti jsou uchovávány ve formě konkrétních, v minulosti vyřešených problémů (případů).
- Při řešení nového problému systém hledá v databázi nejpodobnější starý případ a adaptuje jeho řešení.
- Tento přístup se nazývá **případové usuzování (Case-Based Reasoning - CBR)** a je užitečný v doménách, kde je těžké formulovat obecná pravidla, ale existuje bohatá historie zkušeností.

4. Učení s Využitím Apriorních Znalostí

Tradiční induktivní učení hledá hypotézu `h`, která nejlépe odpovídá datům. Učení založené na znalostech (Knowledge-Based Learning) rozšiřuje tento model o **apriorní znalosti (Background Knowledge)**.

Logická formulace učení vypadá takto: `Hypothesis ∧ Descriptions |= Classifications`

To znamená, že hypotéza spolu s popisy příkladů musí logicky implikovat jejich klasifikace. Pokud přidáme apriorní znalosti, rovnice se změní: `Background ∧ Hypothesis ∧ Descriptions |= Classifications`

Apriorní znalosti hrají dvě klíčové role:

1. **Omezují prostor hypotéz:** Generovaná hypotéza musí být konzistentní nejen s daty, ale i s apriorními znalostmi.
2. **Zjednodušují hypotézu:** Apriorní znalosti mohou "pomoci" s vysvětlením pozorování, takže samotná hypotéza může být mnohem jednodušší a snáze nalezitelná.

Příklad: Při učení se konceptu "Dědeček(x, y)", pokud systém již zná koncept "Rodič(x, y)", je výsledná hypotéza `∃z (Rodič(x, z) ∧ Rodič(z, y))` mnohem jednodušší než hypotéza, která by musela být odvozena pouze z primitivních predikátů jako `Matka` a `Otec`.

Lekce 9: Zpětnovazební Učení (Reinforcement Learning)

Zpětnovazební učení (Reinforcement Learning - RL) je oblast strojového učení, kde se agent učí optimálnímu chování na základě interakce s prostředím. Místo toho, aby dostával explicitní instrukce, co má dělat, učí se z následků svých akcí prostřednictvím odměn a trestů.

1. Charakteristiky Zpětnovazebního Učení

RL se liší od ostatních paradigmat učení (např. učení s učitelem) v několika klíčových aspektech:

- **Aktivní učení:** Agent není pasivním příjemcem dat, ale aktivně prozkoumává prostředí a svými akcemi ovlivňuje, jaká data získá.
- **Sekvenční povaha:** Interakce s prostředím probíhá v sekvenci kroků. Současná akce může ovlivnit nejen okamžitou odměnu, ale i budoucí stavy a odměny.
- **Orientace na cíl:** Cílem agenta je maximalizovat kumulativní (celkovou) odměnu v dlouhodobém horizontu, nikoliv jen okamžitý zisk.
- **Učení bez optimálních příkladů:** Agent se učí metodou pokus-omyl. Nemá k dispozici "správné odpovědi" (optimální akce) pro dané stavy, ale pouze signál odměny, který mu říká, jak dobře si vede.

Každý stav `s` v prostředí může být spojen s určitou **odměnou (reward) R(s)**. Agent se snaží naučit takovou strategii (politiku), která ho povede skrze stavy tak, aby součet (diskontovaných) odměn byl co nejvyšší.

Tento přístup je vhodný zejména pro problémy, kde je obtížné nebo nemožné poskytnout "oštítkovaná" data pro učení s učitelem, například při učení robota chodit nebo při hraní složitých her jako šachy nebo Go.

2. Typy Zpětnovazebního Učení

Rozlišujeme dva hlavní scénáře RL:

- **Pasivní Zpětnovazební Učení (Passive RL):** Agent má **fixní strategii (politiku) π**, která mu předepisuje, jakou akci má provést v každém stavu. Cílem agenta je pouze naučit se, jak dobrá tato strategie je, tj. naučit se **hodnoty užitku U(s)** pro jednotlivé stavy. Tento proces se podobá vyhodnocování politiky (policy evaluation) z Markovských rozhodovacích procesů.

- **Aktivní Zpětnovazební Učení (Active RL):** Agent nezná optimální strategii a musí se ji naučit. Cílem je nejen zjistit užitek stavů, ale také nalézt optimální politiku π*(s). Klíčovým problémem je zde tzv. **dilema průzkumu vs. využití (exploration vs. exploitation)** – agent se musí rozhodovat, zda provede akci, o které ví, že je dobrá (využití), nebo zda zkusí novou, neprozkoumanou akci, která by mohla být ještě lepší (průzkum).

3. Komponenty Učícího se Agenta

Obecný RL agent se skládá z několika částí:

- **Výkonná komponenta (Performance Component):** Zodpovídá za výběr a provádění akcí v prostředí na základě aktuální strategie.
- **Učící se komponenta (Learning Component):** Aktualizuje znalosti agenta (např. užitkové funkce nebo politiku) na základě zkušeností (sekvencí stavů, akcí a odměn).
- **Kritik (Critic):** Poskytuje zpětnou vazbu učící se komponentě tím, že hodnotí, jak dobré byly provedené akce. Tato zpětná vazba je odvozena od odměn získaných z prostředí.
- **Generátor problémů (Problem Generator):** Navrhuje nové akce nebo sekvence akcí, které mohou vést k prozkoumání nových a potenciálně užitečných částí stavového prostoru (podporuje exploraci).

4. Formulace Úlohy: Markovský Rozhodovací Proces (MDP)

Formálním rámcem pro zpětnovazební učení je **Markovský rozhodovací proces (Markov Decision Process - MDP)**. MDP je matematický model pro sekvenční rozhodování ve stochastickém (pravděpodobnostním) prostředí.

4.1 Definice MDP

MDP je definováno jako čtveřice `(S, A, P, R)`:

- `S`: Množina stavů.
- `A`: Množina akcí.
- `P(s' | s, a)`: **Přechodová funkce**, která udává pravděpodobnost, že provedením akce `a` ve stavu `s` se agent dostane do stavu `s'`. Klíčová je **Markovská vlastnost**: pravděpodobnost přechodu závisí pouze na aktuálním stavu a akci, nikoliv na celé historii předchozích stavů.
- `R(s)`: **Funkce odměn**, která přiřazuje číselnou hodnotu (odměnu) každému stavu.

Prostředí je **plně pozorovatelné**, což znamená, že agent v každém okamžiku přesně ví, ve kterém stavu se nachází.

4.2 Cíl Agenta v MDP

Cílem agenta je najít *optimální politiku π(s)**. Politika (strategie) je pravidlo, které agentovi říká, jakou akci `a` má provést v každém stavu `s`. Optimální politika je taková, která maximalizuje **očekávaný kumulativní zisk** (součet diskontovaných odměn):

`Uπ(s) = E[ Σ (γ^t * R(St)) ]`

kde `γ` je diskontní faktor ($0 \leq \gamma < 1$), který určuje váhu budoucích odměn, a `St` je stav v čase `t`.

V kontextu RL agent typicky nezná přechodovou funkci `P` ani funkci odměn `R`. Tyto parametry se musí naučit z interakce s prostředím.

Lekce 10: Počítačové Vidění

Počítačové vidění je vědní a technologická disciplína, jejímž cílem je vytvářet stroje, které jsou schopné "vidět" a interpretovat vizuální informace z okolního světa, podobně jako lidé.

1. Typické Úlohy Počítačového Vidění

Počítačové vidění řeší širokou škálu úloh, které lze rozdělit do několika kategorií:

- **Rozpoznávání (Recognition):** Identifikace a klasifikace objektů ve scéně. Například rozpoznání obličeje, značky auta nebo typu zvířete.
- **Analýza pohybu (Motion Analysis):** Sledování objektů v čase, odhad jejich rychlosti a trajektorie. Příkladem je sledování chodců ve videu.
- **Rekonstrukce scény (Scene Reconstruction):** Vytváření 3D modelů scény z 2D obrázků. Toho se využívá například ve virtuální realitě nebo robotice.
- **Restaurace obrazu (Image Restoration):** Odstraňování šumu, rozmazání a jiných degradací z obrazu za účelem zlepšení jeho kvality.

Typický proces zpracování obrazu má několik kroků:

1. **Snímání a digitalizace:** Pořízení obrazu (např. kamerou) a jeho převod do digitální podoby. Zahrnuje i OCR (Optical Character Recognition) pro text.
2. **Předzpracování:** Úpravy obrazu pro zlepšení kvality, např. odstranění šumu.
3. **Segmentace obrazu:** Rozdělení obrazu na smysluplné části nebo objekty.
4. **Popis objektů:** Extrakce charakteristických rysů (features) z každého objektu.
5. **Porozumění obsahu obrazu:** Interpretace a klasifikace objektů a celé scény.

2. Základní Charakteristiky Digitálního Obrazu

Digitální obraz je reprezentován mřížkou pixelů. Jeho kvalita je určena dvěma hlavními parametry:

- **Vzorkování (Sampling):** Určuje **rozlišení** obrazu, tedy počet pixelů, ze kterých se skládá. Vyšší rozlišení znamená více detailů.
- **Kvantování (Quantization):** Určuje počet možných hodnot (úrovní jasu nebo barev), které může každý pixel nabývat. Například 8bitový černobílý obraz má 256 úrovní šedi.

3. Detekce Hran

Hrany jsou místa v obraze, kde dochází k prudké změně jasu. Jsou to základní stavební kameny pro rozpoznávání objektů.

- **Základní myšlenka:** Hrany lze detekovat hledáním míst s největším **gradientem** (změnou) jasu, což se matematicky provádí pomocí **derivace** obrazové funkce.
- **Problém se šumem:** Reálné obrazy obsahují šum, který způsobuje malé, náhodné změny jasu. Přímá aplikace derivace by vedla k detekci velkého množství falešných hran.
- **Řešení - Vyhlazení:** Před derivováním je nutné obraz **vyhladit**, aby se potlačil šum. To se provádí zprůměrováním hodnoty každého pixelu s hodnotami jeho sousedů.

3.1 Konvoluce

Vyhlazení je příkladem operace zvané **konvoluce**. Při konvoluci je nová hodnota každého pixelu vypočtena jako **vážená lineární kombinace** hodnot pixelů v jeho okolí. Váhy jsou definovány malou

maticí, která se nazývá **konvoluční jádro (kernel)**. Pro vyhlazení se často používá Gaussovo jádro, které dává největší váhu centrálnímu pixelu a váhy se postupně snižují směrem od středu.

4. Rosenblattův Perceptron

- Navržen Frankem Rosenblattem v roce 1958, je to jeden z prvních a nejjednodušších modelů umělé neuronové sítě.
- Byl inspirován fungováním lidského oka a jeho původním úkolem bylo rozpoznávat znaky abecedy z pole optických snímačů o rozměrech 20x20.
- Jedná se o algoritmus pro **binární klasifikaci**. Na základě váženého součtu vstupních hodnot rozhoduje, zda vstup patří do jedné ze dvou tříd. `Výstup = f(Σ(váha_i * vstup_i))`
- **Struktura Perceptronu:**
  1. **Receptory (S-jednotky):** Vstupní vrstva, která přijímá data (např. pixely obrazu).
  2. **Asociativní elementy (A-jednotky):** Mezivrstva s pevně danými vahami, která provádí jednoduchou extrakci rysů.
  3. **Reagující elementy (R-jednotky):** Výstupní vrstva, která sečte vážené signály z A-jednotek a na základě prahové funkce vydá výstup (např. 0 nebo 1). Váhy vedoucí do této vrstvy se během učení upravují.

5. Konvoluční Neuronová Síť (CNN)

- Moderní a velmi úspěšný typ **hluboké (vícevrstvé), dopředné (feed-forward)** neuronové sítě, který je standardem pro úlohy analýzy obrazu.
- **Klíčová vlastnost:** Neurony v jedné vrstvě nejsou propojeny se všemi neurony v předchozí vrstvě, ale pouze s **malou lokální oblastí** (tzv. receptivním polem). Tato operace je matematicky ekvivalentní konvoluci, odtud název.
- Tato architektura umožňuje síti postupně se učit hierarchii rysů:
  - První vrstvy se naučí detekovat jednoduché rysy jako hrany a rohy.
  - Další vrstvy kombinují tyto jednoduché rysy do složitějších vzorů (textury, části objektů).
  - Hluboké vrstvy rozpoznávají celé objekty.
- Výstupem CNN je obvykle **vektor pravděpodobností**, který udává, s jakou pravděpodobností vstupní obraz patří do jednotlivých předdefinovaných kategorií (např. 80% kočka, 15% pes, 5% liška).

Lekce 11: Agenti a Roboti

Tento průvodce se zabývá robotikou, autonomními agenty a architekturami, které řídí jejich chování.

1. Robotika

Robotika je disciplína na pomezí informatiky, strojírenství a umělé inteligence, která se zabývá návrhem, konstrukcí a provozem robotů – fyzických agentů, kteří manipulují s fyzickým světem.

1.1 Typy Robotů

Roboty lze kategorizovat do několika hlavních skupin:

- **Průmyslové roboty (Manipulátory):** Jsou to robotická ramena, která jsou fyzicky ukotvena na svém pracovišti, například na montážní lince. Jejich hlavním úkolem je manipulace s objekty. Jsou nejrozšířenějším typem průmyslových robotů.

- **Mobilní roboty:** Pohybují se ve svém prostředí pomocí kol, nohou nebo jiných mechanismů. Příklady zahrnují autonomní vozidla (UGV), drony (UAV), podvodní vozidla (AUV) a planetární rovery.
- **Humanoidní roboty:** Svým vzhledem a často i pohybem napodobují člověka. Jsou příkladem mobilních manipulátorů, kteří kombinují mobilitu s schopností manipulace.

2. Autonomní Robot

Autonomní robot je inteligentní stroj schopný vykonávat úkoly samostatně bez přímé lidské pomoci.

- **Nejdůležitější vlastnost:** Schopnost vnímat své okolí a reagovat na nepředvídatelné změny v něm.
- **Základní části:**
  1. **Senzory (Sensors):** Umožňují robotovi vnímat prostředí (např. kamery, lasery, gyroskopy).
  2. **Řídicí jednotka (Controller):** "Mozek" robota, který zpracovává data ze senzorů a rozhoduje o akcích.
  3. **Efektory (Effectors) / Aktuátory (Actuators):** Vykonávají fyzické akce (např. motory v kolech, kloubech ramene, chapadla).

3. Prostředí Agenta

Vlastnosti prostředí zásadně ovlivňují návrh agenta. Prostředí může být:

- **Plně pozorovatelné (Fully Observable) / Částečně pozorovatelné (Partially Observable):** Má agent v každém okamžiku přístup k úplnému stavu prostředí? Reálný svět je pro roboty téměř vždy částečně pozorovatelný.
- **Deterministické (Deterministic) / Stochastické (Stochastic):** Je další stav prostředí plně určen aktuálním stavem a akcí agenta? Pohyb reálných robotů je stochastický kvůli skluzu kol, nepřesnostem motorů atd.
- **Epizodické (Episodic) / Sekvenční (Sequential):** Skládá se život agenta z nezávislých epizod, nebo současná akce ovlivňuje budoucí rozhodnutí? Robotika je typicky sekvenční.
- **Statické (Static) / Dynamické (Dynamic):** Mění se prostředí, zatímco agent přemýšlí? Svět, ve kterém se roboti pohybují, je dynamický.
- **Diskrétní (Discrete) / Spojité (Continuous):** Je stavový prostor, čas a akce agenta tvořen konečným počtem hodnot, nebo spojitým rozsahem? Robotika se typicky odehrává ve spojitém prostoru a čase.
- **Jednoagentové (Single-agent) / Multiagentní (Multi-agent):** Působí v prostředí jeden agent, nebo více? Doprava je typickým multiagentním prostředím.

4. Senzorický Subsystém

- **Pasivní senzory:** Zachytávají signály z prostředí (např. **kamera**, **gyroskop**, **tlačítka**).
- **Aktivní senzory:** Vysílají do prostředí signál a detekují jeho odraz (např. **sonar**, **laserové dálkoměry (lidar)**, **GPS**).
- **Lokální vs. Distribuované:** Senzory mohou být součástí robota (lokální) nebo rozmístěné v prostředí (distribuované).

5. Příklady Historických Robotů

- **Shakey (konec 60. let):** První univerzální mobilní robot schopný vnímání, plánování a provádění akcí. Během jeho vývoje vznikl mimo jiné i slavný prohledávací algoritmus A*.

- **Genghis (80. léta):** Šestinohý chodící robot od Rodneyho Brookse, který demonstroval principy reaktivního řízení a subsumpční architektury.
- **Cog a Kismet (90. léta):** Humanoidní roboti z MIT zaměření na kognitivní vědu a sociální interakci mezi člověkem a robotem.
- **Coco:** Humanoidní robot inspirovaný tělem gorily pro studium pohybu.

6. Typy Agentů

6.1 Reaktivní Agent (Simple Reflex Agent)

- Nejjednodušší typ agenta. Jedná na základě **pravidel typu podmínka-akce**.
- Rozhoduje se pouze na základě **aktuálního vjemu**, ignoruje historii.
- Nevytváří si vnitřní model světa ani složité plány.
- Příklad: Jednoduchý robotický vysavač, který změní směr, když narazí do zdi.

6.2 Deliberativní Agent (Model-based / Goal-based Agent)

- Pokročilejší typ agenta, který překonává omezení reaktivních agentů.
- Udržuje si **vnitřní stav (model světa)**, který reprezentuje aspekty prostředí, jež nejsou aktuálně viditelné.
- Má explicitně definované **cíle**, kterých se snaží dosáhnout.
- **Plánuje** sekvence akcí, které vedou ke splnění cílů.
- Příklad: Robot, který plánuje optimální trasu pro doručení balíku, přičemž zvažuje překážky a alternativní cesty na základě mapy (svého modelu světa).

7. Subsumpční Architektura

- Navržena Rodneym Brooksem. Je to způsob dekompozice chování agenta do několika **vrstev**.
- Každá vrstva implementuje určité chování (např. "vyhni se překážce", "jdi vpřed").
- Vrstvy spolu "soupeří" o řízení agenta. Nižší, základnější vrstvy (např. vyhýbání se) mají **přednost** a mohou "potlačit" (subsume) příkazy z vyšších vrstev (např. "jdi k cíli").
- Jedná se o reaktivní, zdola nahoru budovanou architekturu.

8. Celulární Automat (Cellular Automaton)

- Dynamický systém, který je **diskrétní v prostoru a čase**.
- Skládá se z pravidelné mřížky **buněk**, kde každá buňka může být v jednom z konečného počtu stavů.
- Stav každé buňky v dalším časovém kroku je určen **lokální přechodovou funkcí**, která závisí na aktuálním stavu buňky a stavech jejích **sousedů**.
- **Využití:**
  - **Simulace přírodních jevů:** Šíření požáru, růst krystalů, šíření epidemií.
  - **Teoretická informatika:** Modelování výpočetních procesů (např. Turingův stroj).
  - **Fyzika:** Modelování dynamiky plynů a termodynamických procesů.
  - **Kryptografie:** Generování pseudonáhodných čísel pro šifrovací algoritmy. Jeden z nejznámějších příkladů je "Hra života" (Game of Life) od Johna Conwaye.

Appendix: AI Theory Cheat Sheet

4IZ431 AI Theory Sheet

1. Search Algorithms

Breadth-First Search (BFS)

- **Definition:** An uninformed search strategy that expands the shallowest nodes in the search tree first. It uses a **FIFO (First-In-First-Out) Queue** to store the frontier.
- **Properties:**
  - **Complete:** Yes (if branching factor b is finite).
  - **Optimal:** Yes (if path cost is a non-decreasing function of the depth of the node, e.g., all step costs are equal).
  - **Time Complexity:** $O(b^d)$ where d is the depth of the solution.
  - **Space Complexity:** $O(b^d)$ (keeps all nodes in memory).

Depth-First Search (DFS)

- **Definition:** An uninformed search strategy that expands the deepest node in the current frontier of the search tree. It uses a **LIFO (Last-In-First-Out) Stack** to store the frontier.
- **Properties:**
  - **Complete:** No (fails in infinite spaces or loops without check).
  - **Optimal:** No (can find a deeper, more expensive solution first).
  - **Time Complexity:** $O(b^m)$ where m is the maximum depth.
  - **Space Complexity:** $O(b \cdot m)$ (linear space).

Uniform Cost Search (UCS)

- **Definition:** An uninformed search strategy that expands the node with the lowest path cost $g(n)$ from the frontier. It uses a **Priority Queue** ordered by $g(n)$.
- **Properties:**
  - **Complete:** Yes (if step costs $\ge \epsilon > 0$).
  - **Optimal:** Yes.
  - **Time/Space:** $O(b^{C^*/\epsilon})$ where C^* is optimal cost.

A* Search

- **Definition:** An informed best-first search that uses an evaluation function $f(n) = g(n) + h(n)$, where:
  - $g(n)$: Cost from start to node n.
  - $h(n)$: Estimated cost from node n to goal (heuristic).
- **Algorithm:** Expands the node with lowest $f(n)$.
- **Optimality:** Guaranteed if the heuristic is Admissible (for Tree Search) or Consistent (for Graph Search).

Heuristics: Admissible vs. Consistent

- **Admissible Heuristic:**
  - A heuristic $h(n)$ is admissible if it *never overestimates* the cost to reach the goal.
  - **Formula:** $0 \le h(n) \le h^*(n)$, where $h^*(n)$ is the true optimal cost from n to goal.
  - **Benefit:** Ensures optimality in Tree Search.
- **Consistent Heuristic (Monotonicity):**
  - A heuristic is consistent if, for every node n and every successor n' generated by action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost from n' to the goal.

- **Formula:** $h(n) \le c(n, a, n') + h(n')$
- **Benefit:** Ensures optimality in Graph Search (no need to re-open closed nodes). Consistency implies admissibility.

---

2. Game Theory / Multi-Agent

Minimax Algorithm

- **Definition:** A recursive algorithm for choosing the next move in a two-player, zero-sum game (MAX vs. MIN). MAX tries to maximize the utility value, while MIN tries to minimize it.
- **Pseudocode Concept:**

```
function MINIMAX-DECISION(state) returns an action
  return argmax_[a in ACTIONS(s)] MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state)
  if TERMINAL-TEST(state) return UTILITY(state)
  v = -infinity
  for each a in ACTIONS(state) do
    v = MAX(v, MIN-VALUE(RESULT(state, a)))
  return v

function MIN-VALUE(state)
  if TERMINAL-TEST(state) return UTILITY(state)
  v = infinity
  for each a in ACTIONS(state) do
    v = MIN(v, MAX-VALUE(RESULT(state, a)))
  return v
```

Alpha-Beta Pruning

- **Definition:** An optimization technique for Minimax that prunes away branches in the game tree that cannot possibly influence the final decision.
- **Parameters:**
  - α: The value of the best (highest-value) choice found so far at any choice point along the path for MAX.
  - β: The value of the best (lowest-value) choice found so far at any choice point along the path for MIN.
- **Pruning Condition:**
  - Prune if $\alpha \ge \beta$.
- **Benefits:** Does not affect the result. Can double the search depth reachable in the same amount of time ($O(b^{m/2})$ instead of $O(b^m)$ in best case).

Expectimax

- **Definition:** An adaptation of Minimax for games with chance/random elements (e.g., dice, stochastic ghosts in Pacman).
- **Mechanism:**
  - **MAX nodes:** Take the maximum value of successors (like Minimax).
  - **CHANCE nodes:** Take the *weighted average* (expected value) of successors based on probability distribution.

    - **Formula for Chance Node:** $V(s) = \sum_{s'} P(s'|s, a) V(s')$
- **Note:** Expectimax does not prune efficiently (pruning is only possible if bounds on utility values are known).

---

3. Reinforcement Learning

Q-Learning Update Formula

- **Definition:** An off-policy TD (Temporal Difference) control algorithm that learns the value of the optimal policy independently of the agent's actions.
- **Formula:** $$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') \right]$$
    - Or equivalently: $Q(s,a) \leftarrow Q(s,a) + \alpha [\text{difference}]$
    - Where:
        - α: Learning rate ($0 < \alpha \le 1$).
        - γ: Discount factor ($0 \le \gamma \le 1$).
        - r: Reward received.
        - $\max_{a'} Q(s', a')$: Estimate of optimal future value (greedy).

Value Iteration vs. Policy Iteration

- **Value Iteration (VI):**
    - **Algorithm:** Iteratively updates utility values $V(s)$ for all states until convergence. The policy is derived from the final values.
    - **Update Rule (Bellman Optimality Update):** $$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') [R(s,a,s') + \gamma V_k(s')]$$
- **Policy Iteration (PI):**
    - **Algorithm:** Alternates between two steps:
        1. **Policy Evaluation:** Calculate $V^\pi(s)$ for the current policy π (solving linear equations or iterative evaluation).
        2. **Policy Improvement:** Update $\pi(s)$ to be greedy with respect to new $V^\pi(s)$.
    - **Comparison:** PI often converges in fewer iterations than VI, but each iteration is more computationally expensive (solving systems of equations).

Epsilon-greedy Exploration

- **Definition:** A strategy to balance exploration and exploitation.
- **Mechanism:**
    - With probability $1 - \epsilon$: **Exploit** (choose the action with the highest current Q-value).
    - With probability ϵ: **Explore** (choose a random action uniformly).
- **Purpose:** Ensures the agent continues to try new actions to find potentially better rewards (exploration) while maximizing reward based on current knowledge (exploitation).

---

4. Key Concepts from Papers

BERT (from cv08 paper: Devlin et al.)

- **Full Name:** Bidirectional Encoder Representations from Transformers.

- **Core Innovation:** Designed to pre-train deep **bidirectional** representations from unlabeled text by jointly conditioning on both left and right context in all layers.
- **Pre-training Tasks:**
    1. **Masked Language Model (MLM):** Randomly masks 15% of input tokens and predicts them based on context (allows bidirectional training unlike left-to-right LMs).
    2. **Next Sentence Prediction (NSP):** Predicts if sentence B actually follows sentence A (helps with sentence-pair tasks like QA).
- **Usage:** Two steps: **Pre-training** (on large corpus) and **Fine-tuning** (on specific downstream tasks with minimal architectural changes).

Deep RL (from cv10 paper: Mnih et al.)

- **Contribution:** The **Deep Q-Network (DQN)** agent that learns control policies directly from high-dimensional sensory input (raw pixels) using reinforcement learning.
- **Key Innovations:**
    1. **Deep Convolutional Neural Network (CNN):** Approximate the Q-value function $Q(s, a; \theta)$. Input is raw pixels (84x84x4 after preprocessing).
    2. **Experience Replay:** Stores transitions (s, a, r, s') in a replay memory buffer and samples *random minibatches* for training. This breaks correlations between consecutive samples and smooths data distribution.
    3. **Target Network:** Uses a separate network $\hat{Q}$ to generate target values, updated periodically. Improves stability.
- **Result:** Achieved human-level performance on Atari 2600 games.

Samuel's Checkers (from cv06 paper: A. L. Samuel)

- **Overview:** Two machine-learning procedures applied to the game of checkers (1959).
- **Rote Learning:**
    - Saves board positions and their backed-up scores.
    - Uses "effective ply" (looking up a saved board effectively extends the search depth).
    - Required cataloging and "forgetting" (culling) to manage memory.
- **Generalization Learning:**
    - Learns a **scoring polynomial** (evaluation function) $V(s) = \sum w_i f_i(s)$.
    - Adjusts weights w_i based on the difference between the current evaluation and the backed-up evaluation from a look-ahead search ("delta").
    - Alpha-Beta pruning (referred to as "looking ahead" with pruning conditions) was utilized to deepen the search.
- **Outcome:** The program learned to play a better-than-average game, verifying that computers can learn to outperform their programmers.