VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Srovnání paradigmat strojového učení

Comparison of Machine Learning Paradigms

Martin Kaleta

Bakalářská práce

Vedoucí práce: Ing. Adam Albert

Ostrava, 2023

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/23/046805
vsboee23013DD7

Zadání bakalářské práce

Student: **Martin Kaleta**

Studijní program: B0613A140014 Informatika

Téma: Srovnání paradigmat strojového učení
Comparison of Machine Learning Paradigms

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je shrnou teorii strojového učení v multiagentních systémech, provést shrnutí základních paradigmat. V případové studii student vytvoří příklady algoritmů pro porovnání jejich rysů a funkcionality.

Práce bude obsahovat:
1. Shrnutí teorie strojového učení na základě symbolické reprezentace znalostí, genetických algoritmů, pravděpodobnosti a umělých neuronových sítí.
2. Případová stude, ve které student naimplementuje metody strojového učení na základě symbolické reprezentace a genetických algoritmů. Vstupem algoritmů budou data reprezentována pomocí klauzulí jazyka Prolog.

Seznam doporučené odborné literatury:

[1] LUGER, George F. Artificial intelligence: structures and strategies for complex problem solving. 6th ed. Boston: Pearson Addison-Wesley, c2009. ISBN 978-0-321-54589-3.
[2] MITCHELL, Tom M. Machine Learning. New York: McGraw-Hill, c1997. ISBN 00-704-2807-7.
[3] KUBÍK, Aleš. Inteligentní agenty. Brno: Computer Press, 2004. ISBN 80-251-0323-4.
[4] POOLE, David L., MACKWORTH Alan K. Artificial intelligence: foundations of computational agents. 2nd pub. Cambridge: Cambridge University Press, 2010. ISBN 978-0-521-51900-7.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Adam Albert**

Datum zadání: 01.09.2021

Datum odevzdání: 04.07.2023

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 30.05.2023 18:39:20

Abstrakt

Cílem práce je shrnout teorii strojového učení v multiagentních systémech, jejich základních paradigmat a pomocí implementace porovnat základní rysy a funkcionality vybraných metod strojového učení. První část práce se zabývá představením strojového učení a popisem jednotlivých paradigmat. Druhá část je věnovaná případové studii.

Klíčová slova

strojové učení, agent, multiagentní systém, symbolická reprezentace znalostí, genetické algoritmy, pravděpodobnost, umělé neuronové sítě

Abstract

The aim of this thesis is to summarize the theory of machine learning in multi-agent systems, their basic paradigms and compare the basic features and functionalities of selected machine learning methods through implementation. The first part of the thesis deals with the introduction of machine learning and the description of each paradigm. The second part devoted to a case study.

Keywords

machine learning, agent, multiagent system, symbol-based, genetic algorithms, stochastic, connectionist approaches

Poděkování

Rád bych na tomto místě poděkoval svému vedoucímu Ing. Adamu Albertovi za cenné rady a připomínky při tvorbě této práce. V neposlední řadě bych chtěl poděkovat také své rodině a přátelům za jejich neustálou podporu během mého studia.

Obsah

Seznam použitých zkratek a symbolů

KNN	–	Algoritmus k-nejbližších sousedů
DBSCAN	–	Prostorové shlukování aplikací se šumem na základě hustoty
MAS	–	Multiagentní systém
GA	–	Genetické algoritmy
ANN	–	Umělé neuronové sítě
PDP	–	Paralelně distribuované zpracování
HMM	–	Skrytý Markovův model
BFS	–	Algoritmus prohledávání do šířky

Seznam obrázků

Seznam algoritmů

Seznam tabulek

Kapitola 1

Úvod

Cílem této práce je poskytnout srovnání různých paradigmat strojového učení v multiagentních systémech. Poskytnout charakteristiku jednotlivých paradigmat včetně jejich výhod, nevýhod a vhodnosti použití při řešení různých druhů problémů. Celá práce je rozdělena na dvě části. První část je věnována představení multiagentních systémů a jejich návaznosti na strojové učení. Dále je v této části shrnuta teorie strojového učení na základě symbolické reprezentace znalostí, genetických algoritmů, pravděpodobnosti a umělých neuronových sítí. Druhá část se věnuje případové studii, která obsahuje implementaci a popis algoritmu založeném na symbolické reprezentaci znalostí a genetických algoritmech.

Celá práce je strukturována následovně. Kapitola 2 popisuje, co je to multiagentní systém (MAS), agent, prostředí a interakce v MAS. V kapitole 3 je stručně popsáno, co je to strojové učení, jaké jsou jeho základní rozdělení podle způsobu učení a jaké jsou základní přístupy zpracování dat ve strojovém učení. Kapitola 4 je zaměřena na popis a charakteristiku 4 základních paradigmat strojového učení.

Kapitola 5 obsahuje případovou studii, ve které jsem implementoval dva algoritmy. Prvním z nich je Winstonův algoritmus 5.1 aplikovaný na rozpoznávání tvarů, který je založen na symbolické reprezentaci znalostí. Algoritmus je možné použít na celou řadu problémů spojených s identifikací a rozpoznáváním (například rozpoznávání různých objektů). Druhým z nich je algoritmus inspirovaný přirozeným výběrem živočišných druhů, tj. evoluční biologií, který vyhledává cestu v bludišti 5.2. V závěru kapitoly 5 jsou oba popisované algoritmy porovnány z hlediska jejich přístupů. Kapitola 6 obsahuje závěr této práce.

Kapitola 2

Multiagentní systémy

Multiagentní systém je systém skládající se z několika autonomních (inteligentních) agentů, kteří interagují a spolupracují v daném prostředí za účelem dosažení individuálních nebo společných cílů. Každý agent jedná nezávisle na sobě v souladu se svými vlastními cíli. Klíčovou vlastností pro MAS a agenty je racionalita [2]. Tím se rozumí, že agenti jsou schopni plnit úkoly v prostředí, ve kterém se nacházejí. Inteligence agentů se může lišit, může být velmi omezená (například agent s omezenými schopnosti rozhodování a jednání, který je schopen pouze reagovat na signály, vyhýbat se překážkám, ale nemá schopnost plánování nebo učení) nebo velice rozvinutá.[1] Takovým příkladem může být osobní vozidlo, které pomocí svých senzorů pozoruje okolní prostředí, získává data, kterými se učí a podle situace dynamicky rozhoduje, co provede (například několik senzorů, které hlídají bezpečnou vzdálenost mezi vozidly a podle toho upravují rychlost vozidla).

S MAS se setkáme tam, kde centralizované systémy s předem daným chováním nevyhovují. Nejčastěji se jedná o systémy, kde je potřeba dynamického chování (například řízení leteckého provozu nebo v oblasti počítačových sítí při řízení vyvažování zátěže sítě). V takových situacích nemusí centralizovaný systém adekvátně fungovat a neposkytoval by potřebnou schopnost reagovat na dynamické požadavky. Proto jednou z motivací pro použití MAS je možnost distribuce výpočetních zdrojů a schopností mezi propojenou síť agentů. Tím jsou eliminovány problémy úzkého hrdla, omezených zdrojů nebo kritické poruchy, kterými centralizované systémy trpí. Když například použijeme MAS a nastane neočekávaná situace, během které dojde k selhání několik agentů, tak nemusí dojít k selhání celého systému, protože ostatní agenti můžou být schopni autonomně na danou situaci reagovat. Tím lze chápat například, že někteří agenti mohou dočasně zastoupit jiné v jejich práci.

Tímto lze podle [2] tvrdit, že mezi základní vlastnosti MAS patří:

- Autonomnost jednotlivých agentů. Agenti v MAS vykonávají nezávisle na ostatních své vlastní cíle na základě vlastního vnímání.

[1] Inteligenci agentů lze chápat jako schopnost vnímat, porozumět, rozhodovat a jednat v daném prostředí.

- Robustnost ve smyslu, že jsou MAS navrženy tak, aby byly odolné vůči poruchám a výpadkům. Když jeden agent selže, tak jsou ostatní agenti schopni dále pracovat na svých úkolech.
- Adaptace jako schopnost reagovat a přizpůsobit se změnám v prostředí.
- Decentralizace řízení v systému.

V následujících částech této kapitoly jsou dále stručně popsány jednotlivé složky, které tvoří MAS, a to agenti, prostředí a interakce mezi nimi.

2.1 Agent

Podle [2] je *agent* definován následovně:

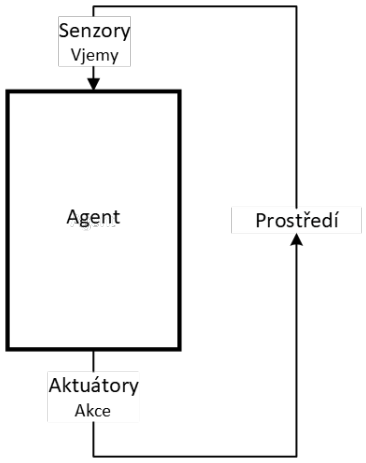
Definice 1 (Agent) *Agent je entita zkonstruovaná za účelem kontinuálně a do jisté míry autonomně plnit své cíle v adekvátním prostředí na základě vnímání prostřednictvím senzorů a prováděním akcí prostřednictvím aktuátorů. Agent přitom ovlivňuje podmínky v prostředí tak, aby se přibližoval k plnění cílů.*

Agenta můžeme popsat jako jakéhosi zástupce, který je pověřen k vykonávání určitých úkolů [2]. Měl by být schopen fungovat samostatně bez nutnosti zásahu člověka, případně minimálním. Člověk zde má pouze funkci dohledu nebo kontroly s minimálním vlivem na běh agenta. Agenta lze konstruovat, jak v hardwarové nebo softwarové podobě a je navržen tak, aby operoval v omezeném prostředí. Kromě případů, kdy agenti mají pevně daný úkol a opakují ho bez nutnosti učení je často nezbytné, aby se agenti nepřetržitě učili a zlepšovali svou činnost na základě změn v prostředí. Na obrázku 2.1 je představeno základní schéma agenta. Agent pomocí senzorů přijímá vstup (vjemy) z prostředí a na základě vstupu provede s aktuátory nějakou akci, která může ovlivnit prostředí, nebo selhat.

V porovnání s objektem, který je podle [2] nejpropracovanější abstrakcí reality a komponentou modelovaného systému v objektově-orientovaném inženýrstvím, představuje agent abstraktnější entitu s několika vlastnostmi navíc. Mezi tyto vlastnosti patří:

- autonomie (nezávislost na ostatních agentech)
- existence v prostředí
- propojení s prostředím pomocí senzorů a aktuátorů

Podle [2] lze agenty rozdělit do několika skupin na základě jejich racionality a vnitřní stavby jednotlivých částí (složitosti organizace vnitřních komponent). Konkrétně je můžeme rozdělit na reaktivní, deliberativní, sociální a hybridní agenty.



Obrázek 2.1: Základní schéma agenta.

2.1.1 Reaktivní agent

Reaktivní agent je nejjednodušším typem agentů. Podle [2] je definován následovně:

Definice 2 (Reaktivní agent) *Reaktivní agent je entita vykonávající akce bezprostředně a výlučně na základě stimulů přijatých z prostředí.*

Jedná se o agenta s nejjednodušší vnitřní architekturou, který se skládá z několika paralelních procesů tzv.druhů chování [2]. V krajním případě pouze z jednoho chování. Chování jsou aktivovány na základě podnětu z prostředí, vnitřního stavu agenta nebo jejich kombinací.

Reaktivní agent neobsahuje vnitřní reprezentaci prostředí a jeho akce jsou závislé na aktuálním stavu prostředí. Agent nedisponuje žádnou možností plánování a jedná okamžitě v reakci na vstupy ze senzorů. Jediné k čemu využívá vyrovnávací paměť je k uchovávání záznamu o stavu agenta.

Příkladem reaktivního agenta je podle [2] mechanická beruška se senzory (tykadly) a aktuátory (kolečky). Beruška se pohybuje po ploše stolu, která reprezentuje prostředí. V momentě, kdy tykadlo berušky klesne pod hranu stolu, tak pomocí koleček změní svůj směr pohybu, aby beruška nespadla.

2.1.2 Deliberativní agent

Deliberativní (uvažující) agent je agent uchovávající si symbolickou reprezentaci prostředí a vnitřních stavů na základě, kterých vytváří plány k dosažení svých cílů [2]. Vytvořené plány mohou být hierarchicky uspořádány a jednotlivé cíle ohodnoceny prioritou pro výběr vhodného plánu.

Liší se tedy od reaktivního agenta tím, že vytváří a uchovává symbolickou reprezentaci prostředí na základě, kterého je schopen plánovaní a rozhodování svých akcí.

Příkladem deliberativního agenta je robot doručující předměty z jednoho místa na druhé. V tomto případě agent obsahuje symbolicky reprezentovanou mapu prostředí s jejichž pomocí se po-

hybuje. Její součástí jsou informace o poloze předmětů, překážkách, cestách a cílových místech. Předtím než může agent doručovat, tak si musí namapovat pomocí senzorů celé prostředí.

2.1.3 Sociální agent

Dle [2] definice Sociálního agenta zní následovně:

Definice 3 (Sociální agent) *Sociální agenti rozšiřují své poznatky prostředí o modely ostatních agentů. Jedná se především o adresy, jména a specifikace jejich schopností pro případnou kooperaci vzájemných aktivit.*

Agenti mají zpravidla omezené schopnosti, kompetence, čas, finanční, výpočetní a jiné prostředky, které jsou nezbytné pro řešení přidělených úkolů [2]. Z tohoto důvodu sociální agenti vyhledávají pomoc ostatních agentů tak, aby odstranili nebo zmírnili nepříznivé důsledky omezenosti svých vlastních zdrojů.

Příkladem sociálního agenta je autonomní vozidlo, které vyhledává pomoc jiných vozidel ve svém okolí a vyměňuje si mezi nimi informace o rychlosti, trase jízdy, dopravní situaci. Tímto způsobem vozidlo získá rozšířený pohled na okolní prostředí a lépe se může přizpůsobit aktuálním podmínkám.

2.1.4 Hybridní agent

Agent s hybridní architekturou se rozumí takový agent, který kombinuje některé nebo všechny z uvedených architektur do jednoho celku [2]. Tímto se rozumí, že hybridní agent v sobě kombinuje charakteristiky a vlastnosti jiných druhů agentů.

Příkladem hybridního agenta je robot doručující objednávky kombinující architekturu reaktivních a deliberativních agentů. Reaktivní část agenta je zodpovědná za detekci překážek a okamžité vyhýbání se jim. Pokud robot narazí na překážku, reaktivní část robota okamžitě zareaguje a vyhledá alternativní cestu kolem překážky. Deliberativní část se stará o rozhodování a plánování. Na základě mapy prostoru a informacích o objednávkách deliberativní část robota vytváří optimální trasu, která minimalizuje čas doručení. Tímto způsobem robot efektivně plánuje svou trasu, minimalizuje čas doručování a reaguje na překážky v reálném čase.

2.2 Prostředí

Prostředím v MAS nazýváme prostor, ve kterém agent přichází do styku se vším, co se v něm nachází [20]. Agenti vnímají změny v prostředí pomocí senzorů a na základě těchto vjemů provádějí své akce, kterými jsou schopni do jisté míry ovlivňovat prostředí a vše, co se v něm nachází (například další agenty).

V [19] se dočteme, že prostředí lze klasifikovat podle toho, jestli je:

- Přístupné nebo nepřístupné.

- Deterministické nebo nedeterministické.
- Statické nebo dynamické.
- Spojité nebo diskrétní.

V přístupném prostředí může agent získat úplné, přesné a aktuální informace o stavu prostředí. Agent tak svými senzory vnímá celé prostředí, ve kterém se nachází, zatímco nepřístupné prostředí poskytuje agentovi pouze omezené informace o stavu prostředí.

Deterministické prostředí je charakterizováno tím, že každá akce má jediný zaručený účinek. Neexistuje žádná nejistota ohledně stavu, který bude výsledkem provedené akce.

Statické prostředí zůstává nezměněno, pokud nejsou prováděny žádné akce agentem. Naproti tomu dynamické prostředí je takové, které je ovlivňováno i jinými procesy než jen akcemi agenta.

Prostředí je diskrétní, pokud existuje pevný, konečný počet vjemů a akcí. Diskrétní prostředí má konečnou množinu stavů jako například šachová hra.

2.3 Interakce agentů v MAS

Interakce v MAS probíhá mezi agenty, kteří spolu komunikují a provádějí vzájemné akce v rámci prostředí. Interakce v MAS můžeme rozdělit následovně na:

- koordinaci
- kooperaci
- komunikaci

V [2] se dočteme, že koordinace mezi agenty hraje významnou roli při dosahování cílů na úrovni celého systému. Rozlišujeme zde dva druhy koordinace - centralizovaná a decentralizovaná. Centralizovaná ve smyslu, že existuje agent, který má řídicí postavení v systému a přiřazuje jednotlivým agentům jejich role a cíle. Naopak decentralizovaná koordinace si zakládá na spolupráci a vzájemné dohodě mezi agenty. V tomto případě koordinace prolíná do kooperace, kdy se agenti dohodnou na společném postupu a navzájem si pomáhají při dosažení společného cíle.

Kooperace představuje řízenou formu koordinace s účelovým uspořádáním agentů ve skupině s cílem dosáhnout společného řešení problému [2]. Každý agent má přesně stanovenou roli, kterou má plnit a také vztahy mezi ostatními agenty k tomu, aby bylo dosaženo globálního cíle.

V rámci MAS může být záměrem agenta ovlivnit vnitřní stav jiného agenta nebo s ním spolupracovat [20]. Komunikace v MAS, tak představuje jednu z nejdůležitějších rolí. Komunikace mezi agenty v MAS probíhá pomocí vyšších komunikačních jazyků, které poskytují způsob předávání zpráv [2]. Příkladem takového jazyka je KQML (Knowledge Query and Manipulation Language) a jazyk ACL (Agent Communication Language) vyvíjený organizací FIPA[2]. Jedná se o komunikační

[2]Foundations for Intelligent Physical Agents (FIPA) je organizace složená z velkých korporací, snažící se o vytváření a propagování standardů v oblasti agentových technologií.

jazyky navržený pro výměnu zpráv (řečových aktů) mezi agenty v MAS. Každá zpráva obsahuje informace o odesílateli, příjemci, obsahu a typu, který určuje význam zprávy. V současné době je primárně používaným jazykem pro komunikaci jazyk ACL, který je odvozený z jazyka KQML.

Podle [2] lze komunikaci rozdělit na základě toho, jak se zpráva od odesíláte dostane k příjemci následujícím způsobem:

- **Přímá** komunikace, kdy se zpráva odesílatele dostane přímo k jeho příjemci.
- **Nepřímá** komunikace, kdy agenti spolu komunikují prostřednictvím zprostředkovatele (mediátora), který pracuje jako poštovní služba. Rozesílá příchozí zprávy do schránek agentů. Agenti nemusí vědět informace o ostatních agentech, ale pouze o zprostředkovateli, který zpracovává jejich požadavky.

Kapitola 3

Strojové učení

Po představení MAS je nyní vhodné se přesunout ke strojovému učení, které představuje jeden z klíčových nástrojů v oblasti inteligentních systémů.

Strojové učení je jednou z vědních disciplín umělé inteligence, která se zabývá algoritmy a technikami, jakými je agentovi umožněno se učit [2]. Na základě získaných zkušeností je schopen se zlepšovat ve svém chování, zefektivnit svou schopnost přizpůsobit se změnám okolního prostředí, a podávat tak přesnější výsledky.

Podle Mitchella [4] je učení definováno následovně:

Definice 4 *Počítačový program se učí na základě zkušeností E vzhledem k nějaké třídě úkolů T a měřítku výkonu P, jestliže se výkon na úkolech T, měřený podle P zlepšil ze zkušeností E.*

Agent, který se učí, přijímá vjemy z prostředí a získává informace z naměřených dat. Získané zkušenosti (znalosti nebo schopnosti) mohou pocházet z různých zdrojů jako jsou například data, která poskytl učitel. Zkušenosti se poté využívají k vytvoření modelu. Model je reprezentací získaných znalostí a dovedností agenta, který mu umožňuje provádět úkoly a přizpůsobovat se změnám v prostředí. Na základě získaných zkušenosti může agent upravovat své akce tak, aby dosáhl lepších výsledků. Může optimalizovat svůj model a adaptovat se na nové situace. Zefektivněním chování lze chápat, že se agent snaží maximalizovat přesnost při plnění úkolů. Na základě zkušenosti může agent hledat způsoby (plánování, strategie), jak dosáhnout přesnějších výsledků.

Strojové učení jako vědní obor není příliš jednoduché správně zařadit, protože se mezi sebou jednotlivé obory prolínají [7]. Spadá mezi ně například obor pravděpodobnosti a statistiky, optimalizace či dolování dat.

Techniky strojového učení se využívají v mnoha odvětvích. Můžeme se s ním setkat v medicíně, kde se strojové učení používá například při identifikaci nádorů na základě obrázků z magnetické rezonance mozku. Proces učení začíná tím, že jsou sesbírány obrázky magnetické rezonance mozku, které obsahují anotace o přítomnosti či nepřítomnosti nádoru. Obrázky slouží jako trénovací data umělé neuronové sítě, která se postupně učí rozpoznávat nádory. Trénování probíhá tak, že se

postupně umělé neuronové sítí předkládají obrázky s anotací (obsahuje nádor či nikoliv), na základě které se síť snaží aktualizovat své váhy tak, aby minimalizovala chybu při klasifikaci obrázku mozku.

Obecně to může být agent, který se učí rozpoznávat obrázky. V počáteční fázi se agent učí identifikovat klíčové rysy, které charakterizují a rozdělují jednotlivé obrázky do skupin. Po dokončení fáze učení se model testuje na nových neznámých obrázcích a na základě předchozích zkušeností se snaží správně identifikovat nový obrázek do již známých skupin. V průběhu činnosti se může agent dále zlepšovat.

3.1 Základní rozdělení strojového učení

Tato část se zaměřuje na základní rozdělení strojového učení a jejich charakteristiku. Strojové učení lze rozdělit podle několika charakteristik do různých kategorií, a to podle druhu učení, způsobu zpracování nebo získávání dat. Nejjednodušeji lze strojové učení rozdělit podle druhu zpětné vazby do třech skupin:

- *učení s učitelem* (supervised learning)
- *učení bez učitele* (unsupervised learning)
- *učení posilováním* nebo také *zpětnovazebné učení* (reinforcement learning)

Učení s učitelem realizuje svou činnost na základě trénovacích příkladů, které jsou ohodnoceny. Podobně funguje *učení bez učitele*, jen v tomto případě chybí pro trénovací vzorky jejich ohodnocení. U *učení s posilováním* se využívá zpětné vazby za správně chování a trest za nežádoucí chování.

3.1.1 Učení s učitelem

V [5] je *učení s učitelem* definováno následovně:

Definice 5 (Supervised Learning) *Učení s učitelem je metoda, ve které učitel předkládá agentovi sadu trénovacích dat. Trénovací data popisují pomocí hodnot vstupních a výstupních atributů nějaký objekt tak, že mezi těmito hodnotami je neznámá funkční závislost. Existuje neznámá funkce* $y = f(x_1,..., x_n)$*, kde* x_1 *až* x_n *odpovídá hodnotám vstupních atributů a y odpovídá hodnota výstupního atributu. Cílem procesu učení s učitelem je nalézt funkci h (hypotézu), která aproximuje funkci f.*

Při učení s učitelem jsou agentovi předkládány příklady z trénovací sady dat, podle kterých vytváří svou hypotézu. Jednotlivé příklady jsou popsány hodnotami vstupních a výstupních atributů. Hodnoty vstupních atributů popisují charakteristiky nějakého objektu (například nějakého zvířete) a jeho hodnota výstupního atributu udává požadovanou odpověď (například, že popisované zvíře je pes).

Pomocí testovací sady dat je poté hypotéza ověřována, jestli je korektní. Testovací sada dat slouží k ověření korektnosti hypotézy a na rozdíl od trénovací sady dat obsahuje pouze hodnoty

vstupních atributů. Hypotéza je korektní, pokud agent předpovídá výstupní hodnoty atributů s nejvyšší přesností.

Tento postup se dá přirovnat k učení ve školách, kdy se učitel snaží předat žákům nové informace nebo vysvětlit nějaký problém. Učitel vysvětluje žákům na příkladech probíranou problematiku a k tomu jim poskytne vždy správný výsledek (zpětnou vazbu). Poté, co učitel vysvětlí danou problematiku, tak následuje testování, kterým se ověří nové znalosti žáků.

Například máme agenta, který se učí předpovědět diagnózu pacientů podle lékařské anamnézy. Pacient je charakterizován hodnotami vstupních atributů - krevní tlak, pulz, tělesná teplota, výška, váha a přítomnost bolesti. Na základě dostupných lékařských dat, které obsahují informace o různých pacientech a jejich odpovídající diagnóze je agent vytrénován. Agent se naučí predikovat diagnózu na základě anamnézy, kde anamnéza představuje hodnoty vstupních atributů a diagnóza je hodnota výstupního atributu. Poté na základě hodnot vstupních atributů vytrénovaný agent predikuje diagnózu pacientů. Tímto může agent pomoci lékařům s rychlejší a přesnější diagnózou pacientů.

Mezi nejznámější metody využívající typ učení s učitelem patří například umělé neuronové sítě a rozhodovací stromy. Rozhodovací stromy jsou v oblasti strojového učení populární metodou ke klasifikaci dat. Jedná se o klasifikátor v podobě stromové struktury, kde uzly představují jednotlivé atributy trénovací sady dat, hrany rozhodovací pravidla (hodnoty atributu) a listové uzly výstupní hodnoty (například predikovanou třídu) [7]. Vstupními daty pro rozhodovací stromy je sada trénovacích příkladů, kde jednotlivé příklady jsou popsány hodnotami vstupních atributů a hodnotou výstupního atributu.

Princip učení stromu spočívá v nalezení rozhodovacích bodů (atributů) v trénovací sadě dat, které minimalizují chybu klasifikace a maximalizují informační zisk. Rozhodovací body představují atributy trénovací sady dat, podle kterých lze nejpřesněji klasifikovat data do jednotlivých tříd. K nalezení těchto atributů a vytváření rozhodovacího stromu slouží například algoritmus ID3, který prochází jednotlivé atributy trénovací datové sady a hodnotí, jak dobře atributy rozdělují trénovací data do tříd. Výsledkem rozhodovacího stromu je hierarchická struktura rozhodovacích pravidel, která umožňuje klasifikovat nová data na základě hodnot jejich vstupních atributů. Uzly stromu reprezentují testy atributů a větve jednotlivé hodnoty daného atributu. Listy stromu představují konečnou klasifikaci (predikovanou výstupní hodnotu).

3.1.2 Učení bez učitele

Učení bez učitele rovněž anglicky Unsupervised learning je metoda učení, která obdobně jako učení s učitelem využívá sadu trénovacích dat, ale s jediným rozdílem, že data neobsahují hodnoty výstupních atributů. Tímto se eliminuje role učitele a nezbývá nic jiného než se naučit rozpoznávat souvislosti v datech samostatně. Souvislosti představují skryté vzorce nebo struktury v datech, které se snažíme najít.

Ve vstupní datech jsou hledány dosud neznámé souvislosti, podle kterých jsou rozděleny do tzv. shluků [7]. Shluky jsou oblasti tvořené daty, které jsou si vzájemně podobná. Data ve shluku sdílejí určité charakteristiky, vlastnosti nebo vzorce, které je odlišují od ostatních dat. Nalezené shluky pak mohou představovat nové třídy.

K nalezení podobnosti se využívá vzdálenosti (metrika) mezi jednotlivými objekty, které jsou například reprezentovány ve formě vektorů. Metrika je funkce $d : X \times X \to R$, jejíž výstupem je reálné číslo vyjadřující vzdálenost mezi dvěma prvky z množiny prvků X [7]. Při výpočtu metriky je důležité nejprve vzít v úvahu charakteristiky atributů jednotlivých prvků kvůli toho, že atributy mohou být rozdílných typů nebo rozsahů hodnot. Normalizace se provádí k vyvážení vlivu jednotlivých atributů, aby se zabránilo jejich nepřiměřené převaze[1]. Tím se zajistí, že každý atribut bude mít přiměřený vliv na celkovou vzdálenost. Nevhodně zvolená metrika zapříčiní, že vzdálenost bude například s velkou převahou určovat pouze jeden atribut nebo se vůbec neprojeví. Mezi nejčastěji používanou metriku patří Euklidovská vzdálenost. Nelze však tvrdit, že její použití je ve všech případech nejvhodnější. Existuje celá řada dalších metrik jako například Manhattanská vzdálenost odvozená podle struktury ulic města New York.

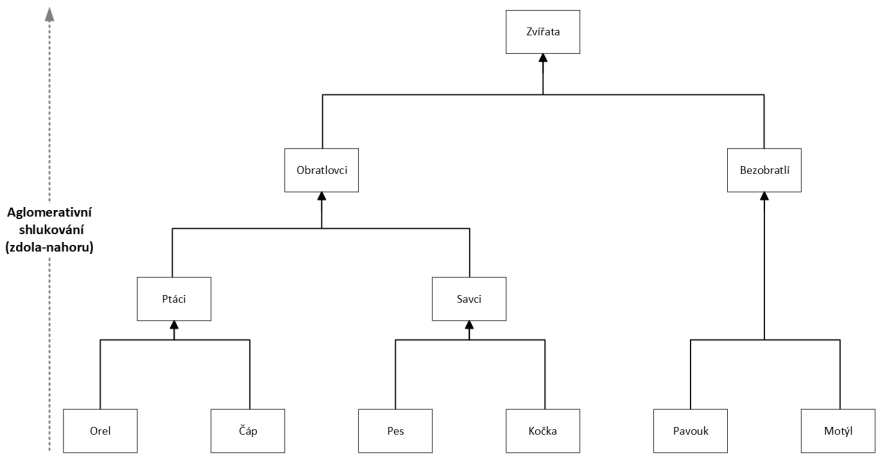
Metody založené na shlukování jsou dále rozděleny do 2 základních kategorií - hierarchické a nehierarchické shlukování [21]. Hierarchické shlukování se zaměřuje na vytváření stromové struktury shluků, kde každý uzel stromu představuje jeden shluk. Existují 2 způsoby, jak této stromové struktury docílit. Prvním z nich aglomerativní shlukování, kde jsou prvky nejprve spojeny do menších shluků, které se následně spojují do větších, dokud nevznikne jeden velký shluk obsahující všechny pod shluky. Druhým přístupem je divizní shlukování, které pracuje v opačném směru. Z jednoho velkého shluku dělením vytváří menší pod shluky. Hierarchické shlukování, tak vytvoří strom, kde jednotlivé uzly představují shluky a listy konkrétní data. Výsledek hierarchického shlukování je často znázorněn pomocí dendrogramu. Dendrogram je druh diagramu, který zachycuje hierarchickou strukturu shluků.

Na obrázku 3.1 je znázorněn příklad hierarchického shlukovaní, konkrétně aglomerativní shlukování nad datovou sadou příkladů popisující zvířata (kočka, pes, pavouk,...). Začneme tím, že každé popisované zvíře považujeme za jedinečný shluk. Na základě těchto jedinečných shluků jsou vytvořeny 2 další shluky (ptáci, savci) podle jejich podobností (například kočka a pes patří mezi savce). Tento proces pokračuje opakovaně, dokud nevznikne jeden shluk obsahující všechny zvířata. Výsledkem je tedy strom rozdělující zvířata do jednotlivých skupin a podskupin.

Nehierarchické shlukování se zaměřuje na rozdělení dat do předem definovaného počtu shluků, které spolu nevytvářejí stromovou strukturu. Nejznámější metodou shlukování tohoto typu je K-Means, který iterativně přiřazuje objekty k nejbližším shlukům podle jejich podobnosti.

V praxi se s učením bez učitele můžeme setkat například při analýze sociálních sítí, kde shlukování je použito k identifikaci uživatelů s podobnými zájmy nebo jinými společnými rysy. Předpo-

[1]Normalizace je úprava rozsahu veličiny. Garantuje velikost určitých parametrů (například rozsah možné hodnoty atributu) [7].



Obrázek 3.1: Příklad hierarchického shlukování na základě dat o zvířatech..

kládejme například, že máme sadu dat reprezentující uživatele nějaké sociální sítě. Každý uživatel je popsán několika atributy jako například věkem, pohlavím a oblíbenou aktivitou. Cílem je identifikovat skupiny uživatelů s podobnými vlastnostmi.

Identifikace skupin probíhá například pomocí algoritmu K-Means. Nalezení shluků algoritmem probíhá na základě několika kroků. Nejprve je nutné určit počet shluků, které chceme v datech nalézt. Podle definovaného počtu shluků jsou náhodně inicializovány centroidy a k nim přiřazeny jednotlivé datové body (data jednotlivých uživatelů ve formě vektorů). Centroid si lze v tomto případě přirovnat k fiktivnímu uživateli, který slouží k reprezentaci shluku. K nejbližšímu centroidu je přiřazen daný datový bod na základě jeho vzdálenosti (Euklidovské vzdálenosti) od centroidu. Poté, co jsou data přiřazena ke každému z centroidů, tak probíhá aktualizace centroidů tak, aby odpovídaly průměrné hodnotě k němu přiřazených dat. Tento proces se opakuje, dokud se centroidy nepřestanou výrazně měnit nebo neproběhne stanovený počet iterací. Tímto způsobem může například K-Means pomoci identifikovat podobné skupiny lidí na sociálních sítích a umožnit jim lépe personalizovat zobrazovaný obsah.

Učení bez učitele je dále popsáno například v [1, 6].

3.1.3 Učení posilováním

Podle [6] je *učení s posilováním* definováno následovně:

Definice 6 (Reinforcement Learning) *Při učení posilováním se agent učí na základě řady posílení - odměn nebo trestů.*

Jedná se o metodu učení pomocí zpětné vazby, kdy je agent umístěn do nějakého systému (stavového prostoru), ve kterém se učí chovat metodou odměňovaní za správné chování a trestáním za to špatné.

Zpětná vazba je zde skryta v podobě odměny za akce dosahující cíle, nebo trestu v opačném případě. Agent se snaží najít nejlepší možné chování.

Učení posilováním je vhodné v situacích, kde je obtížně přesně definovat všechna pravidla, které by vedla k požadovanému chování. V těchto situacích není možné poskytnout přesnou zpětnou vazbu pomocí učitele, který by přesně specifikoval, jak se má agent chovat v každé situaci. V případě her jako například šachy může být těžké specifikovat všechna možná pravidla, která by vedla k úspěchu, protože prostředí bývá často velmi dynamické s mnoha kombinacemi tahů (stavů). Učení posilováním umožňuje agentům objevovat nová pravidla a zlepšovat své výkony na základě interakce s prostředím.

Učení posilováním lze například prezentovat na hře šachů [6]. Agent jako šachový program se učí hrát šachy. Na počátku hry nemá žádné předchozí znalosti o hře, a tak se postupně učí na základě provádění tahů a sledování změn na hrací ploše. Získaný bod na konci šachové partie agentovi napoví, že provedl správnou sérii tahů a tím vyhrál partii. Naopak když žádný bod nezíská, tak to pro něho znamená, že provedl špatnou sekvenci tahů. Cílem agenta je dosáhnout vítězství, případně dosáhnout nejlepšího možného výsledku proti svým soupeřům. Agent zkouší hraním partií šachů různé možnosti tahů a podle zpětné vazby sestavuje strategii hraní (plán vedoucí k výhře) k dosažení nejlepšího výsledku.

Často se s tímto druhem učení setkáme ve videohrách, ale i v robotice. Ve videohrách je například učení s posilováním využito k tvorbě agentů, kteří ovládají počítačem řízené protivníky. Agent se učí interakcí s herním prostředím a podle svých akcí dostává zpětnou vazbu v podobě odměn nebo trestů za přiblížení ke svému cíli. Dále například opakovaným hraním si agent může optimalizovat strategie nebo posloupnost akcí k splnění cíle proti svým soupeřům.

Mezi algoritmy založené na učení posilováním například patří Q-učení [6]. Cílem algoritmu Q-učení je nalézt posloupnost akcí na základě aktuálního stavu agenta, která maximalizuje budoucí odměny. Agent využívá tabulku stavů a akcí k ukládání a aktualizaci hodnot, které reprezentují očekávané odměny za provádění akcí v daných stavech. Princip učení Q-učení spočívá v tom, že se agent postupně učí, jaké akce vykonávat v různých stavech prostředí na základě odměn, které za tyto akce obdrží. Během učení aktualizuje hodnoty v tabulce s cílem dosáhnout optimálních hodnot, které reprezentují nejlepší akce v jednotlivých stavech.

3.2 Základní přístupy

V této kapitole se zaměříme na základní druhy přístupů zpracování dat strojovým učením. Jedná se hlavně o tyto tři způsoby:

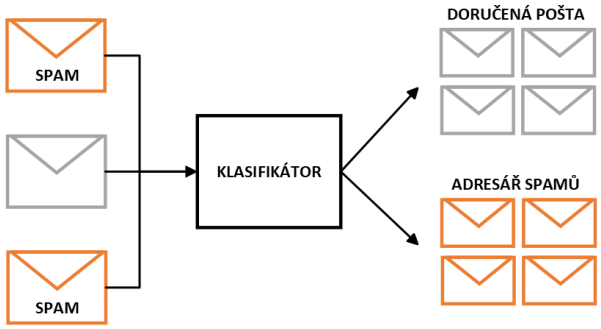
- klasifikace (classification)
- regrese (regression)
- shlukování (clustering)

3.2.1 Klasifikace

Jedná se o metodu strojového učení, která predikuje hodnotu výstupního atributu, jehož hodnota je diskrétní. Klasifikace k tomu používá vstupní datovou sadu, kterou rozdělí na tréninkovou a testovací. Na základě tréninkové sady se vytvoří klasifikační model, který je následně použit k zařazení jednotlivých vzorků z testovací sady do odpovídajících kategorií. Testovací sada dat se používá k vyhodnocení přesnosti vytvořeného klasifikačního modelu, kdy je určeno s jakou přesností dokáže vytvořený klasifikační model korektně predikovat nová data.

Klasifikovat lze jak binárně, tak i pro více tříd. K tomuto přístupu existuje řada algoritmů, které se ho snaží řešit. Příkladem algoritmů, které pomáhají řešit tento přístup jsou rozhodovací stromy, náhodné stromy (Random Forests) a umělé neuronové sítě.

S klasifikací se dnes setkává většina lidí na světě a ani o tom nemá povědomí. Používá se při rozlišení nevyžádané pošty v e-mailové komunikaci, která je znázorněna na obrázku 3.2. Obrázek demonstruje klasifikaci příchozí pošty. Každá příchozí pošta projde klasifikátorem, který rozhodne, jestli se jedná chtěný nebo nechtěný e-mail. Klasifikátor je program, který je naučen rozpoznat různé typy e-mailů podle jejich obsahu, klíčových slov a dalších rysů, a tak je automaticky zařadit do předem definovaných kategorií.



Obrázek 3.2: Klasifikace příchozí e-mailové komunikace.

Mezi nejznámější klasifikační algoritmy lze zařadit algoritmus K-nejbližších sousedů (K-Nearest Neighbors). Principem tohoto algoritmu je přiřadit nový datový bod do kategorie, která je nejpočetněji zastoupena mezi jeho k nejbližšími sousedy. Hodnotě k odpovídá počet nejbližších sousedů nového datového bodu. Nalezení nejbližších sousedů probíhá na základě Euklidovské vzdálenosti. Vzdálenost je vypočtena mezi novým datovým bodem a všemi dostupnými body. Podle vypočtené vzdálenosti je vybráno k nejbližších sousedů, na základě kterých algoritmus určují jejich četnost v jednotlivých kategoriích. Algoritmus následně přiřazuje nový datový bod do kategorie, která je nejpočetněji zastoupena mezi jeho nejbližšími sousedy.

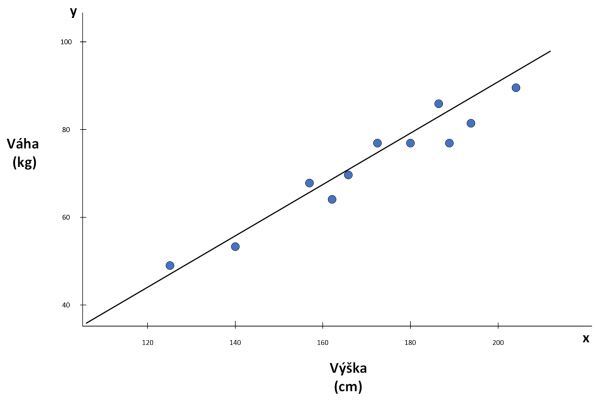
3.2.2 Regrese

Regrese je metoda predikce hodnoty výstupního atributu, která je spojitá. Cílem regrese je najít vztah nebo matematickou funkci mezi závislou výstupní hodnotou a několika nezávislými vstupními hodnotami. Nejběžnějším typem regrese, který se používá je lineární regrese. Tento přístup patří do učení s učitelem.

Regrese se používá nejčastěji tam, kde potřebujeme předpovědět nějakou číselnou hodnotu. Příkladem použití regrese je odhad ceny domů na základě velikosti, počtu místností, jeho lokality a dalších vlastností. Cílem regrese je najít matematickou funkci, která by nejlépe popisovala vztah mezi hodnotami vstupních a výstupních atributů a umožnila predikovat ceny domů pro nové vstupní hodnoty.

Na obrázku 3.3 je vizualizován příklad regresní analýzy, která se používá k modelování vztahu mezi nezávislou vstupní hodnotou na ose x a závislou výstupní hodnotou na ose y. Přímka v grafu reprezentuje regresní přímku, která nejpřesněji popisuje vztah mezi vstupní a výstupní hodnotou atributů na základě poskytnutých trénovacích dat. Přímka tedy představuje regresní model a je odvozena z datové sady. Body v grafu představují konkrétní data, jejich vstupní a odpovídající výstupní hodnoty. Na základě regresní přímky a její rovnice $y = ax+b$ je možné predikovat výstupní hodnotu y pro libovolnou vstupní hodnotu x.

Předpokládejme například, že máme historická data, ve kterých jsou zaznamenávána data o výšce a hmotnosti osob. Na základě těchto dat jsme schopni vytvořit lineární regresní model, který by predikoval váhu na základě výšky člověka. Pokud osa x reprezentuje výšky a osa y váhu člověka, pak podle toho můžeme predikovat, jakou váhu bude mít člověk v závislosti na jeho výšce.



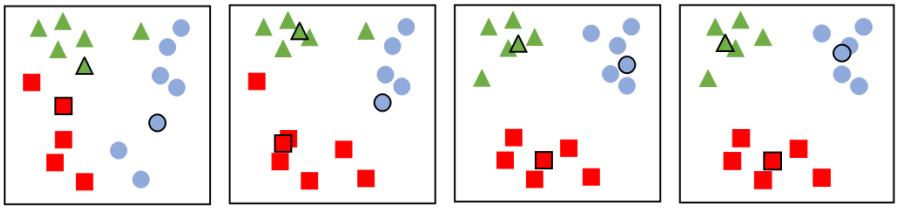
Obrázek 3.3: Grafické znázornění lineární regrese v datové analýze.

3.2.3 Shlukování

Shlukování neboli *clustering* je přístup strojového učení využívající způsobu učení bez učitele, který seskupuje data do shluků podle jejich podobných rysů. Cílem shlukování je analyzovat vstupní data a

nalézt v nich skryté souvislosti nebo podobnosti, podle kterých je identifikovat skupiny s podobnými vlastnosti. Na rozdíl od klasifikace zde nejsou předem definované žádné třídy nebo skupiny, podle kterých by se dala data rozdělit. Podobnost se zde určuje nejčastěji pomocí metriky.

Existuje široká řada algoritmů, které se zde aplikují. Mezi nejznámější můžeme zařadit K-Means a DBSCAN. Algoritmus K-Means je založen na Euklidovské vzdálenosti. Hodnota k odpovídá počtu shluků, které mají být vygenerovány. Jedná se o iterativní algoritmus, kde každá iterace se skládá z několika definovaných kroků - nalezení centroidů, přiřazení dat k nejbližšímu centroidu podle vzdálenosti, posun centroidů do hodnot průměru shluku. Na obrázku 3.4 je ilustrována ukázka procesu shlukování podle K-Means algoritmu, kde k odpovídá hodnotě 3 a jednotlivé shluky jsou odlišeny barvou a tvarem. Body, které jsou označeny černým rámečkem jsou centroidy. Okna v obrázku odpovídají jednotlivým iteracím, při kterých jsou přepočítávány pozice centroidů a znova přiřazeny do shluků jednotlivé data.



Obrázek 3.4: Ukázka procesu shlukování podle 3 shluků.

Například prostřednictvím shlukování nad daty profesionálních fotbalistů je možné na základě jejich statistik (počet gólů, asistencí, úspěšných přihrávek) identifikovat konkrétní skupiny hráčů s podobnými vlastnostmi. Nalezené shluky následně poskytují cenné informace trenérům a dalším členům týmu. Pomáhají přesněji identifikovat například talentované hráče, provést detailní analýzu jejich schopností a posoudit jejich potenciál. Tímto je umožněno lépe porozumět hráčům na základě jejich statistik a předpokládaného potenciálu.

V praxi může být dále shlukování použito v marketingu, kdy se společnosti snaží zjistit, na jaké skupiny lidí zaměřit svou reklamu. Dále se využívá při analýze sociálních sítí, kdy je možné shlukování použít k identifikaci skupin lidí s podobnými zájmy, chováním a tím jim personalizovat reklamu.

Kapitola 4

Shrnutí paradigmat

V této kapitole si shrneme teorii 4 základních paradigmat strojového učení, kterými jsou symbolická reprezentace znalostí, genetické algoritmy, pravděpodobnost a umělé neuronové sítě. Jednotlivé přístupy si zde popíšeme z hlediska základních charakteristik, využití, výhod či nevýhod a jejich používaných algoritmů.

4.1 Učení pomocí symbolické reprezentace znalostí

V této kapitole si rozebereme první ze čtyř paradigmat, kterým je symbolická reprezentace znalostí. Symbolická reprezentace znalostí je dána množinou symbolů a gramatikou. Obsahuje pravidla definovaná pomocí logických výrazů jako je například predikátová logika 1.řádu. Pomocí pravidel jsou následně reprezentovány entity (objekty) a vztahy mezi nimi. Jednotlivé algoritmy se snaží odvodit nové a užitečné zobecnění, které lze vyjádřit pomocí symbolů nebo pravidel [1].

Jak již bylo dříve uvedeno, data jsou reprezentována ve formě výrazů, například predikátové logiky prvního řádu nebo v jiné formě, která využívá symbolický zápis. Zápis pomocí symbolů a pravidel je pro člověka mnohem čitelnější a dokáže z něho pochopit jeho význam.

Jednoduchým ukázkovým příkladem může být *míč*, který je definován tvarem, velikostí a barvou. Malý červený míč může být potom symbolicky reprezentován pomocí následujících pravidel:

$$[velikost(x, malý) \wedge barva(x, červená) \wedge tvar(x, kulatý) \supset Míč(x)]$$

Jednotlivá pravidla tvoří jednoduchý popis míče a umožňují nám si představit, jak by takový míč mohl vypadat. Tímto způsobem je možné popsat libovolné objekty a jejich vlastnosti.

4.1.1 Konceptuální učení

V [1] se uvádí, že konceptuální učení je typickým problémem induktivního učení. Pokouší se odvodit obecnou definici nějakého konceptu[1] (například kočky, psa, čehokoliv), která by umožnila agentovi správně rozpoznat budoucí případy tohoto konceptu. Je to proces učení, který rozpoznává podobnosti v datech a podle nich klasifikuje. Pro své učení využívá množiny příkladů (popis konceptů), ze kterých získává znalosti a buduje obecnou hypotézu.

Konceptuální učení je základem pro algoritmy *Version space search* (prohledávání prostoru verzí) a rozhodovací stromy [1].

4.1.1.1 Version space search

Prohledávání prostoru verzí (Version space search) je metoda, která se používá k prohledávání prostoru konceptů (příkladů, jak pozitivních i negativních), ze kterých se agent snaží nalézt hypotézu popisující vstupní data. Hledá symbolicky reprezentovaný popis, který je upravován na základě vstupních příkladů. Jedná se o inkrementální učení, kdy jsou příklady předkládaný agentovi postupně. K tomu, aby agent mohl nalézt hypotézu potřebuje 2 základní operace. Jsou jimi operace generalizace (zobecnění) a specializace (upřesnění). Vyhledávání v prostoru hypotéz využívá toho, že operace generalizace a specializace definuje částečné uspořádání na konceptech, které se používá k vedení vyhledávání [1].

Mezi primárně používané operace generalizace k úpravě hypotézy podle [1] lze zařadit:

- Nahrazení konstant proměnnými.
- Odstranění atributu z výrazu, který by specializoval hypotézu a vylučoval některé správně příklady konceptu.
- Přidání disjunkce do výrazu.
- Nahrazení hodnoty atributu jeho obecnější hodnotou (třídou).

Uvedené operace generalizace si popíšeme k čemu slouží, a jak můžou být použity k úpravě hypotézy nějakého konceptu. Předpokládejme například, že máme koncept míče popsaný atributy - velikost, barva a tvar, který se dá symbolicky reprezentovat následujícím způsobem:

$$[velikost(x, malý) \wedge barva(x, červená) \wedge tvar(x, kulatý) \supset Míč(x)]$$

Na základě tohoto konceptu, který stanovíme jako počáteční hypotézu popisující míče jsme schopni demonstrovat dříve vyjmenované operace generalizace následovně:

[1] Podle [1, 4] je koncept třída objektů, které popisuje hledaná hypotéza.

- Nahrazení konstant proměnnými. Máme hypotézu popisující míč:

$$[velikost(x, malý) \wedge barva(x, červená) \wedge tvar(x, kulatý) \supset Míč(x)]$$

Například na základě předloženého příkladu míče, který má odlišnou barvu můžeme konstantu *červená* představující barvu míče nahradit proměnnou y, která představuje jakoukoliv hodnotu v atributu barvy míče. Vzniká obecnější hypotéza míče, která není závislá na konkrétní hodnotě atributu barvy. Upravená hypotéza po nahrazení konstanty proměnnou vypadá následovně:

$$[velikost(x, malý) \wedge barva(x, y) \wedge tvar(x, kulatý) \supset Míč(x)]$$

Tato operace tedy umožňuje vytvořit obecnější popis konceptu míče, který není omezen pouze na míče červené. Na barvě míče nezáleží a lze ji nahradit proměnnou, za kterou je možné dosadit jakákoliv barva. Hypotézu, tak budou splňovat příklady libovolně barevných míčů, protože není kladena žádná striktní podmínka na konkrétní barvu míče.

- Odstranění atributu z výrazu, který dříve specializoval hypotézu a vylučoval některé pozitivní příklady konceptu. Máme hypotézu míče:

$$[tvar(x, kulatý) \wedge velikost(x, malý) \wedge barva(x, červená) \wedge \supset Míč(x)]$$

Například byl poskytnut příklad míče, který neobsahuje atribut *velikost*, protože velikost nehraje klíčovou roli při popisu konceptu míče. Na základě tohoto příkladu míče dojde k odstranění atributu *velikost* z hypotézy. Nově budou upravenou hypotézu splňovat i příklady míčů, které dříve kvůli atributu *velikost* nesplňovaly hypotézu míče. Upravená obecnější hypotéza po odstranění atributu vypadá následovně:

$$[tvar(x, kulatý) \wedge barva(x, červená) \wedge \supset Míč(x)]$$

- Přidání disjunkce do výrazu. Máme hypotézu popisující míč:

$$[velikost(x, malý) \wedge barva(x, červená) \wedge tvar(x, kulatý) \supset Míč(x)]$$

Například pomocí poskytnutých příkladů míče definujeme, že míč může mít hodnotu atributu barvy míče pouze červenou nebo žlutou barvu. Pomocí disjunkce umístěné mezi 2 stejné atributy *barva* dojde k rozšíření podmínky pro 2 hodnoty. Upravená obecnější hypotéza po rozšíření atributu *barva* vypadá následovně:

$$[tvar(x, kulatý) \wedge [barva(x, červená) \vee barva(x, žlutá)] \wedge velikost(x, malý) \supset Míč(x)]$$

Hypotéza míče nově umožňuje zahrnout do konceptu míče, jak míče červené barvy, tak i žluté barvy.

- Nahrazení hodnoty atributu jeho obecnější hodnotou (třídou). Operace se zaměřuje na nahrazení konkrétní hodnoty atributu na obecnější hodnotu (třídu). Například hodnotu atributu *barva* v hypotéze:

$$[velikost(x, malý) \wedge barva(x, červená) \wedge tvar(x, kulatý) \supset Míč(x)]$$

Hodnotu atributu *barva* nahradíme obecnou hodnotou (třídou) *primární_barva* na základě poskytnutého příkladu míče, který se liší od hypotézy barvou. Pro obě barvy, jak z hypotézy i příkladu existuje společná třída *primární_barva*, podle které proběhne zobecnění. Dojde tedy k nahrazení barvy v hypotéze obecnou hodnotou s tím, že hypotéza budou nově splňovat míče s barvou, která je součástí třídy. Zobecněná hypotéza vypadá takto:

$$[velikost(x, malý) \wedge barva(x, primární\_barva) \wedge tvar(x, kulatý) \supset Míč(x)]$$

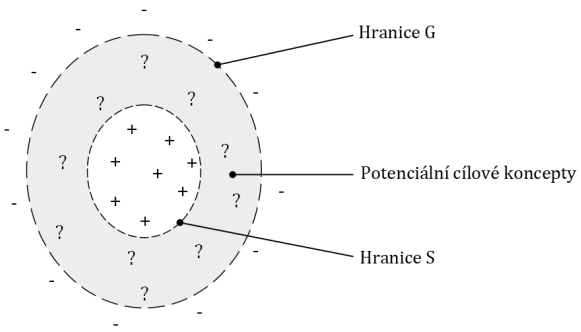
K operacím generalizace dále existují operace specializace, které fungují podobně jen slouží k upřesnění hypotézy.

Na základě operací generalizace a specializace vznikly algoritmy, které řeší prohledávání prostoru hypotéz (version space) a snaží se nalézt výslednou hypotézu. První dva algoritmy *Specific to General Search* a *General to Specific Search* se snaží zmenšovat prostor hypotéz ve směru prohledávání od obecného ke specifickému a od specifického k obecnému. Třetím z nich je *Candidate Eliminations algoritmus*, který využívá směry prohledávání prostoru hypotéz předchozích dvou algoritmů. Candidate Eliminations obsahuje dvě množiny konceptů. Množinu maximálně obecných konceptů G, kterou specializuje a množinu specifických konceptů S, kterou generalizuje. Tím dojde pomocí konvergence k eliminaci všech negativních konceptů a zahrnutí nových potenciálních cílových konceptů.

Obrázek 4.1 představuje zjednodušený popis algoritmu Candidate Eliminations [1]. V obrázku jsou požity symboly (+, -) k označení pozitivních a negativních trénovacích instancí konceptu. Vnitřní kruh zahrnuje známé pozitivní instance pokryté množinou S, zatímco vnější kruh obsahuje všechny instance pokryté množinou G. Šedá oblast obsahuje hledaný cílový koncept spolu s koncepty, které mohou být příliš obecné nebo specifické.

Algoritmus funguje tak, že zmenšuje hranici G, aby vyloučila všechny negativní příklady, a zároveň rozšiřuje vnitřní hranici S tak, aby zahrnovala nové pozitivní příklady. Úkolem je dosáhnout konvergence k cílovému konceptu.

Všechny výše uvedené algoritmy používají sadu, jak pozitivních i negativních příkladů výsledného konceptu. Pouze z pozitivních příkladů lze zobecňovat a nalézt hypotézu hledaného konceptu. S tímto vzniká problém přehnaného zobecnějí známý jako *overgeneralization*, kdy bude hypotéza



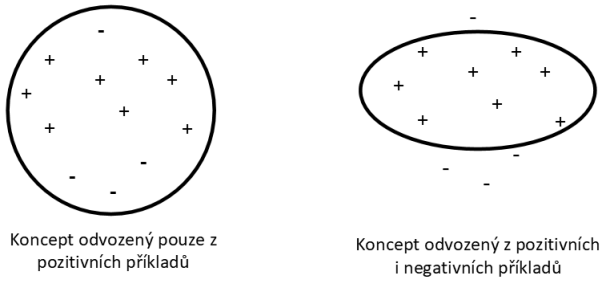
Obrázek 4.1: Konvergující hranice množin G a S v algoritmu Candidate Eliminations (převzato z [1]).

příliš obecná a bude zahrnovat i příklady, které nepatří pod hledaný koncept. Z toho důvodu se využívají negativní příklady, pomocí kterých se hypotéza specializuje a zabraňuje vzniku příliš obecné hypotézy. Negativní příklady pomáhají vyloučit příklady, které nespadají pod námi hledaný koncept a tím zvyšují přesnost hypotézy. V ideálním případě by měl být nalezený koncept dostatečně obecný, aby pokryl všechny pozitivní příklady, a zároveň dostatečně specifický, aby vyloučil negativní příklady.

Na obrázku 4.2 jsou znázorněny obě popisované situace. V prvním případě je hledaný koncept odvozen pouze na základě pozitivních příkladů, což vede k problému přílišného zobecnění kvůli toho, že do konceptu jsou zahrnuty i příklady (označené symbolem "-"), které daný koncept nepředstavují. V druhém případě je koncept odvozený pomocí pozitivních i negativních příkladů a tím je dostatečně obecný, aby pokryl pozitivní příklady, a zároveň vyloučil negativní příklady.

Toto téma bude dále popsáno v případové studii, která se zabývá implementací a popisem Winstonova algoritmu pro rozpoznávání tvarů.

Více lze o prohledávání prostoru verzí například nalézt v [1].



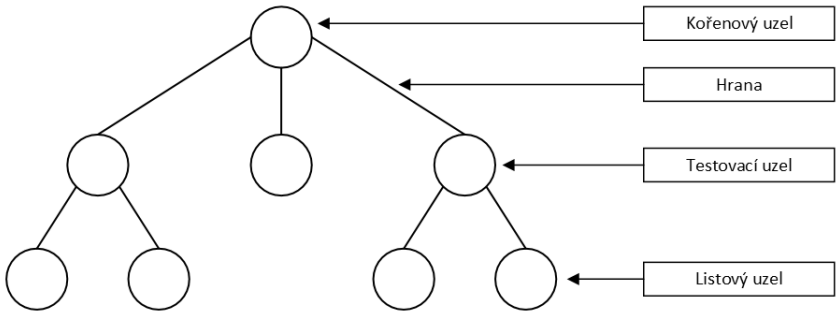
Obrázek 4.2: Role negativních příkladů v předcházení přílišné generalizace (převzato z [1]).

4.1.2 Rozhodovací stromy

Rozhodovací stromy patří mezi nejznámější symbolické metody strojového učení. Umožňují vytvářet modely sloužící ke klasifikaci příkladů do dvou nebo několika tříd. Podle [7] je rozhodovací strom hierarchický nelineární systém umožnující nalezení, uložení znalostí a jejich využití k analýze nových dat. Jejich nejčastější využití je při analýze a klasifikaci kvalitativních závislých proměnných na základě vstupních dat.

Mezi jejich základní charakteristiky patří srozumitelnost, čitelnost, flexibilita, hierarchická struktura a existence algoritmů, které umožňuji automatizovaně vytvářet rozhodovací stromy a extrahovat znalosti z dat [7]. Rozhodovací stromy jsou flexibilní, protože můžou zahrnovat jak kvantitativní, tak kvalitativní data. V uzlech rozhodovacího stromu je možné kombinovat několik atributů do vícerozměrných funkcí, které slouží k rozhodování.

Rozhodovací strom se skládá z kořenového uzlu a dalších uzlů, které jsou dvojího typu (listové a testovací uzly). Každý testovací uzel uvnitř stromu představuje test na jeden atribut a hrany odpovídají jednotlivým hodnotám daného atributu. Rozhodovací strom je tvořen testy na atributy, které odpovídají programátorské podmínce *if...then...else*. Hrany vycházející z testovacích uzlů jsou ohodnoceny hodnotami příslušného atributu. Dosažení listu při průchodu rozhodovacím stromem vede ke klasifikaci nebo predikci výstupní hodnoty analyzovaného záznamu. Může se například jednat o rozhodnutí pravda nebo nepravda. Na obrázku 4.3 je ilustrována struktura rozhodovacího stromu. V porovnání s umělými neuronovými sítěmi (vrstvy neuronů, váhové parametry, aktivační



Obrázek 4.3: Struktura rozhodovacího stromu

funkce, ...) je struktura rozhodovacích stromů mnohem jednodušší a čitelnější, což umožňuje snadnější porozumění procesu, jak strom dospěl k danému rozhodnutí.

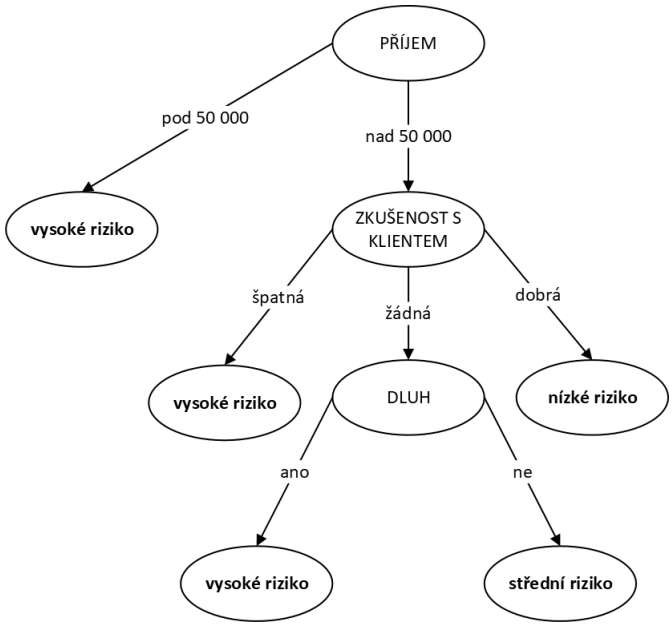
Jádrem učení s klasifikací objektů do skupin spočívá ve výběru kořene stromu a zařazením atributů do uzlů stromu [2]. Mezi základní algoritmy k vytváření rozhodovacích stromů patří ID3 a TDIDT (Top Down Induction of Decision Trees).

ID3 je algoritmus k vytváření rozhodovacího stromu a klasifikaci na základě nominálních (diskrétních) vstupních atributů [7]. Rozhodovací strom se rozvětvuje podle počtu hodnot atributů. Předností ID3 je jejich schopnost vybrat z velkého množství atributů jen ty, které nejvíce zvyšují

informační zisk (snižováním entropie) rozhodovacího stromu. Základem je entropie, jejíž hodnota vyjadřuje míru informace v sadě příkladů. Čím je hodnota entropie menší, tím přesněji lze klasifikovat jednotlivé příklady. Cílem rozhodovacího stromu je minimalizovat hodnotu entropie a tím dosáhnout největšího informačního zisku při rozdělování dat.

Algoritmus ID3 pracuje iterativně. V každém kroku vybírá atribut s nejvyšším informačním ziskem (nejnižší entropií) a používá ho jako rozhodovací pravidlo pro vytvoření uzlu v rozhodovacím stromu. Jednotlivé větve uzlu odpovídají hodnotám atributu. Tento postup se aplikuje pro každou větev, dokud není splněno určité podmínky ukončení vytváření stromu.

Na obrázku 4.4 je uveden příklad jednoduchého rozhodovacího stromu, který podle několika kritérií (atributů) klasifikuje klienty do tříd podle výšky rizika poskytnutí půjčky.



Obrázek 4.4: Příklad rozhodovacího stromu ke klasifikaci klientů podle několika atributů.

4.2 Genetické algoritmy

Genetický algoritmus (GA) je metoda, která se snaží napodobovat proces přirozené evoluce, kterou popsal Charles Darwin (1858) ve své teorii o vývoji druhů a přirozeném výběru [1]. Základní myšlenkou je napodobit vývoj živého organismu a podle něho vytvořený algoritmus aplikovat při řešení složitých problémů, kde dochází i k změnám prostředí, na které člověk nedokáže vhodně reagovat. Podobně jako umělé neuronové sítě jsou GA založeny na biologické metafoře, kdy učení chápou jako soutěž mezi populací vyvíjejících se jedinců.

GA se podle [1, 9] řadí do skupiny *evolučních algoritmů*, které se obecně používají k řešení složitých optimalizačních úloh a vyhledávacích problémů pomocí technik inspirovanými evolucí (dědičnost, mutace, selekce, křížení). Také jsou schopny pracovat s velkým množstvím dat a najít v nich informace, které by jinak byly obtížné nalézt.

Vznik prvních GA se datuje do 60. let dvacátého století, kdy jejich průkopník John Holland z Michiganské univerzity poprvé v [8] popsal základní principy a vlastnosti těchto algoritmů. Při svém výzkumu Holland definoval 2 cíle. Prvním z nich bylo zlepšit porozumění procesu přirozené adaptace a za druhé navrhnout systém s podobnými vlastnostmi jako mají systémy přírodní.

V GA se informace nebo vlastnosti reprezentují pomocí genetického kódu jedince. Tento kód může být reprezentován různými způsoby v závislosti na tom, jaký problém řeší. Může se jednat například o posloupnost jednotlivých bitů.

Následující uvedené pojmy jsou podle [1, 8, 9] nezbytné pro další pochopení základních principů GA. *Gen* je nejmenší část *chromozomu*, která je dále už nedělitelná. Přesně se jedná o konkrétní část chromozomu. *Chromozom* je soubor informací, které v sobě nese každý jedinec[2]. Nejčastěji jsou tyto informace reprezentovány ve formě řetězce nul a jedniček, ale může se také jednat o matice nebo vektory.

Dalším důležitým pojmem je *populace*. *Populaci* můžeme chápat jako množinu jedinců v rámci jedné *generace*. V GA *generace* představuje populaci jedinců v daném časovém okamžiku. Každá nová generace poté vzniká z rodičů a jejich potomků. Rodič v tomto případě je jedinec vstupující do rekombinace[3] a potomek je jedinec, který vznikne rekombinací rodičů. Na obrázku 4.5 jsou znázorněny zmiňované pojmy - gen, chromozom, populace. V tomto případě je gen reprezentován ve formě bitu. Dále chromozom jako řetězec 4 bitů a jako poslední je vizualizována populace složená ze 2 jedinců.

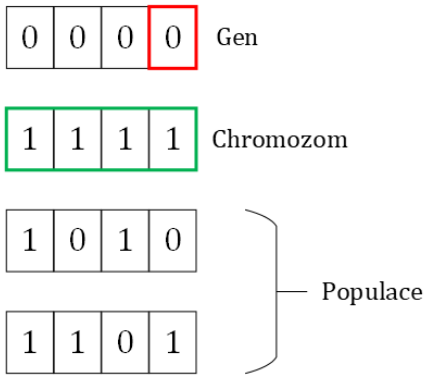
Fitness hodnota je číslo, které vyjadřuje kvalitu každého jedince. Pro každý problém je potřeba sestavit fitness funkci nebo hodnoticí funkci, která jako výsledek vrací požadovanou číselnou hodnotu. Fitness funkce hodnotí, jak dobře přispěje jedinec k další generaci řešení.

4.2.1 Operace

Jak již bylo zmíněno, GA využívají techniky podobné těm, které se vyskytují při přirozeném vývoji. Jsou jimi tyto operace:

- selekce (selection)
- křížení (crossover)
- mutace (mutation)
- inverze (inversion)

[2]Jedinec je nositelem genetické informace.
[3]Proces, který spočívá v kombinaci genetických informací rodičů s cílem vytvořit nové potomky.



Obrázek 4.5: Reprezentace genu, chromozomu a populace.

Z těchto vyjmenovaných technik se však řadí v [8] mezi základní a nejčastěji používané jen *selekce*, *křížení* a *mutace*. Zbylé operace se využívají jen zřídka. Operace se aplikují nad celou generací a výstupem je vždy nová generace. Dokud nejsou nalezeni jedinci s námi požadovanými vlastnosti, tak se jednotlivé operace opakují.

V následujících částech si popíšeme zmíněné 3 základní operace používané k nalezení vhodných jedinců.

4.2.1.1 Selekce

Selekce se využívá k výběru nejlepších jedinců, kteří se mohou stát rodiči pro další generaci. Existuje zde několik metod selekce. Jedná se o ruletovou, turnajovou selekci, metodu ořezáním a náhodným výběrem [9]. Většina těchto metod pracuje s kvalitou jedince (fitness hodnota). Každý jedinec má svoji fitness hodnotu jeho chromozomu.

- **Vážená ruleta** je metoda, kde každý jedinec má šanci být vybrán pro příští generaci, která je úměrná jeho fitness hodnotě. Každý jedinec dostane podíl na pomyslné ruletě podle své fitness hodnoty. Poté je prováděno losování pro výběr nových rodičů. Pravděpodobnost výběru a velikosti výseče je tedy určena kvalitou jedince. Pravděpodobnost, že bude jedinec vybrán je možné vypočítat následovně podle uvedeného vzorce:

$$p_i = \frac{f_i}{\sum_1^N f_i} \tag{4.1}$$

  - p_i pravděpodobnost, s jakou bude i-tý jedinec vybrán
  - f_i fitness hodnota i-tého jedince
  - N počet jedinců v populaci

  Dle uvedeného vzorce je možné sestavit ruletu rozdělenou na N částí úměrných k hodnotám p_i. Rozložení pravděpodobnosti na ruletě může například vypadat tak, jak je uvedeno v tabulce

4.1, kde ke každému jedinci je přiřazena odpovídající pravděpodobnost podle jejich fitness hodnoty.

Pořadí jedince	Fitness hodnota	Pravděpodobnost [%]
1	0.23	8.5
2	0.72	26.7
3	0.95	35.2
4	0.36	13.3
5	0.44	16.3

Tabulka 4.1: Pravděpodobnosti jednotlivých jedinců

- **Turnajová selekce** vybírá náhodně z populace skupinu jedinců. Minimálně musí však vybrat 2 a z nich následně vybere nejlepšího jedince s vyšší fitness hodnotou. Tento způsob řešení umožňuje přežití i slabších jedinců, protože jejich výběr je náhodný, a tak se můžou dostat do souboje s mnohem slabšími jedinci.
- **Ořezáním** funguje tak, že se všichni jedinci v populaci seřadí podle fitness hodnoty a následně podle vybraného parametru je populace rozdělena na dvě části. Jedinci jsou poté libovolným pravidlem vybírání z části, která obsahuje vyšší hodnoty fitness. Zachodí se tedy část, která obsahuje jedince s nízkou hodnotou fitness.
- **Náhodný výběr** je nejjednodušší metodou, která nijak neřeší hodnoty fitness a náhodně vybírá z celé populace.

Podle [8, 9] je selekce důležitou technikou, protože nám umožňuje přenést nejlepší vlastnosti rodičů na své potomky, a tím přispět k postupnému zlepšování kvality populace.

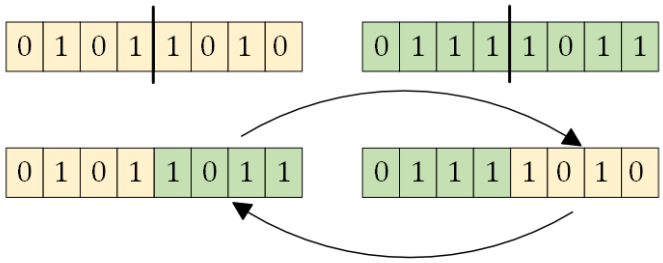
4.2.1.2 Křížení

Jedná se o operaci, která navazuje na selekci. Křížení je operace, která umožňuje kombinovat vlastnosti rodičů, a tím vytvářet nové jedince (potomky), kteří jsou kombinací genů svých rodičů. Křížením vznikají jedinci nové populace.

Křížení probíhá tak, že jsou nejdříve náhodně vybráni dva jedinci z populace, kteří jsou určeni selekcí k rozmnožování [2]. Náhodně je vybrána pozice, na které budou chromozomy jedinců, konkrétně jejich řetězce rozděleny. Následně na to jsou řetězce rozděleny podle určené pozice a zkříženy mezi sebou, čímž vzniknou řetězce nových jedinců. Tento proces je inspirován přirozeným křížením organismů.

Existuje několik metod křížení, které se využívají v GA [8, 9]. Mezi ně patří například jednobodové a vícebodové křížení. Jednobodové rozdělí chromozom na dvě části a ty se mezi potomky vymění. U vícebodového je možné provádět libovolné kombinace z více než dvou rodičů.

Na následujícím obrázku 4.6 je graficky znázorněno, jak se provede křížení nad dvěma bitovými řetězci, které jsou rozděleny na dané pozici na dvě části. Na obrázku jsou šipky znázorňující výměnu koncových částí chromozomů mezi rodiči, což vede k vzniku nových potomků.



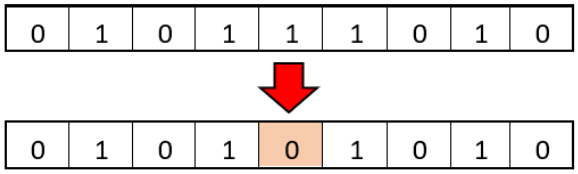
Obrázek 4.6: Příklad křížení dvou chromozomů reprezentovaných ve formě bitového řetězce.

4.2.1.3 Mutace

Mutace je poslední operací GA [8, 9]. Je to proces, kdy se mění genetický kód jedince. Tato technika vezme jednoho potomka a náhodně mu změní některý jeho aspekt. Může například vybrat jeden jeho bit a náhodně ho změnit z 0 na 1 nebo naopak. Mutace je důležitá, protože hned při výběru počáteční populace může být vyloučena podstatná část řešení, a tím pádem by populace jedinců pouze reprodukovala své genetické vlastnosti a nedocházelo by k žádným změnám. S mutací se nám mohou objevit nové kombinace, které můžou mít lepší vlastnosti než původní potomci.

Obdobně probíhá mutace jedinců, kteří používají jinou formu kódovaní chromozomu. Například pokud je chromozom složen z reálných hodnot, tak může být být mutován obdobně, jak při binární mutaci chromozomu. V tomto případě vybrané číslo je mutováno tak, že je bud zvýšeno nebo sníženo o předem danou hodnotu nebo je číslo nahrazeno novým, které je vygenerováno z předem definovaného intervalu.

Na obrázku 4.7 je znázorněna binární mutace jednoho genu, kdy dochází k změně bitu 1 na hodnotu 0, a tím k vytvoření nového jedince.



Obrázek 4.7: Příklad mutace na bitovém řetězci.

4.2.2 Fitness funkce

Hodnotící nebo fitness funkce je funkce jejíž vstupem je jedinec (chromozom) a výstupem je číselná hodnota, která představuje jeho kvalitu [7]. Pomocí této funkce je možné ohodnotit jedince a určit, kteří jsou nejlepší. Na základě této hodnoty, pak probíhá operace selekce, která vybírá vhodné jedince k vytvoření nové populace.

Jako příklad jednoduché fitness funkce lze uvést například funkci $f(x) = 1 + 0.45 \cdot x$. Převedením chromozomu do dekadické podoby a dosazením do funkce obdržíme funkční hodnotu $f(x)$ v bodě, která zároveň odpovídá kvalitě daného jedince.

4.2.3 Princip činnosti genetického algoritmu

Když je použit GA k řešení problémů, probíhá během toho několik fází. Jednotlivé fáze si teď popíšeme podle algoritmu 1 z [1, 9]. Prvotní fází je inicializace a stanovení velikosti populace. Během ní se musí vytvořit počáteční populace řešení, která se skládá z jednotlivých jedinců (chromozomů). Chromozomy jsou na počátku náhodně vygenerovány pro všechny jedince. Ještě než budou jedinci použiti, tak se většinou musí přeložit jejich prezentace do podoby, která umožňuje operace rekombinace (selekce, křížení, mutace). Nejčastější používanou reprezentací bývá bitový řetězec. Tímto je vytvořena počáteční generace.

Druhou fází je fitness funkce. Provede se vyhodnocení kvality každého jedince pomocí zmiňované fitness funkce. Pokud vznikl jedinec splňující požadované vlastnosti k řešení problému, tak v tomto místě algoritmus končí.

Třetím krokem je selekce. Výběr jedinců na základě fitness hodnoty, kteří předají své vlastnosti do další generace. Poté následuje operace křížení a mutace.

Posledním krokem v činnosti GA je návrat zpět k vyhodnocení fitness funkce a rozhodnutí, jestli bylo nalezeno požadované řešení. Pokud ne, tak se proces opakuje po několik generací.

Celý tento postup činnosti GA si můžeme shrnout do několika kroků:

1. Inicializace
2. Vyhodnocení fitness funkce
3. Selekce
4. Křížení
5. Mutace
6. Návrat do bodu 2.

Algoritmus 1: Pseudokód genetického algoritmu

```
t = 0;
inicializace P(t);
while Dokud není nalezen vhodný jedinec pro řešení problému do
    vyhodnotit fitness hodnotu jedinců v P(t);
    selekce z P(t) na základě fitness hodnoty;
    provedení operace křížení a mutace;
    nahradit původní generaci novou;
    t = t + 1;
end
```

4.2.4 Rozšíření genetických algoritmů

V předchozích částech jsme si popsali charakteristiky a principy základního modelu GA na nejnižší úrovni, který popsal Holland při svém výzkumu. Poté přišla řada odborníků (Goldberg, Mitchell, Koza), kteří se zapříčinili ke vzniku nových oblastí výzkumu jako genetické programování, umělý život, kde se techniky GA aplikují na složitější reprezentace (například části počítačového programu) a umožňuji jejich širší použití. Holland používal ve své práci jednoduchou, nízko-úrovňovou bitovou reprezentaci. Řetězec bitů složený z nul a jedniček.

Genetické programování rozšiřuje oblast GA. Je to metoda, která je schopna optimalizovat zdrojový kód programu a vytvářet nové programy. Zaměřuje se tedy na návrh a implementaci programů. Průkopníkem tohoto je John Koza (1992) [1], který přišel se svým návrhem, kde se počítačový program může vyvíjet postupnou aplikací genetických operátorů. Počátečním krokem genetického programování je vytvoření počáteční populace, která je složena z náhodně vygenerovaných programů skládajících se z částí potřebných pro nalezení řešení. Části se mohou skládat z běžných aritmetický, logických operací nebo matematických funkcí. Po provedení inicializace se provedou podobné kroky, které jsme si už dříve popisovali. Provede se operace selekce, křížení a mutace nad kódem programu. Algoritmy pro tyto operace však musí být uzpůsobeny pro vytváření počítačových programů, protože se manipuluje s hierarchicky uspořádanými moduly. Na závěr je stanovena fitness hodnota každého programu. Například podle toho, jak si vedl dobře při řešení zadaných problémů. Tímto se určí, které programy přežijí a budou využity k vytváření nových potomků.

Vhodnou formou reprezentace vývoje generací programů je strom, ve kterém uzly reprezentují struktury programu (cykly, podmínky, funkce) a listy reprezentují proměnné nebo konstanty. Názorný příklad křížení dvou programů z [2] je ilustrován pomocí stromů na obrázku B.1. Rodičovské programy pracující na seznamech prvků jsou navzájem kříženy tak, že část jednoho rodiče je odstraněna z jeho programu a doplněna do programu druhého rodiče. Obdobně je toto provedeno na druhém rodiči.

Genetické programování má řadu využití. Od optimalizace a tvorby algoritmů, dále využití při projektování, návrhu inženýrských nástrojů, vývoji nových léčiv a chemických látek.

Podrobněji je problematika genetického programování a dalších technik například popsána v [1].

4.3 Stochastické a dynamické modely učení

V této kapitole se seznámíme s přístupem strojového učení, založeným na pravděpodobnosti. Tento přístup využívá teorii pravděpodobnosti, souhrnně Stochastiku (obor pravděpodobnosti a statistiky) k modelování závislostí mezi různými proměnnými a k vytváření předpovědí na základě předchozích pozorování. Pomocí teorie pravděpodobnosti můžeme určit pravděpodobnost výskytu jevů, případně popsat, jak se navzájem tyto jevy ovlivňují.

K použití pravděpodobnosti v oblasti strojové učení, k pochopení a předvídání jevů existují 2 hlavní důvody [1]. Prvním z nich je modelování složitých vztahů v měnícím se světě, které se nejlépe zachycují pomocí stochastických modelů a za druhé události mohou být skutečně pravděpodobnostně spolu provázány. Díky toho měla pravděpodobnost a stochastika značný vliv na návrh a účinnost algoritmů strojového učení.

Součástí pravděpodobnosti existují dva druhy modelů učení [1]. Prvním z nich jsou *Stochastické modely* učení, které se zabývají náhodností a pravděpodobnostním rozdělováním na základě dat. Jsou schopny odhadnout pravděpodobnost několika možných událostí. Druhým typem jsou *Dynamické modely* učení zabývají se učením, které se mění v průběhu času (přizpůsobit se změnám na základě času).

Konkrétně mezi modely učení, které využívají pravděpodobnostní přístup můžeme zahrnout například *Bayesovské sítě* a *Markovovy Modely*.

Modely učení, založené na pravděpodobnosti, se používají v řadě oblastech. Příkladem může být zpracování řeči, analýza jazyka, kde je pomocí pravděpodobnosti vybírán nejlepší možný překlad nebo nejlepší stavba slov pro daný kontext. Dále při diagnostice zdravotního stavu může být pravděpodobnost využita pro analýzu dat a následné predikci vzniku nemocí. Tímto je možné u pacientů předpověď podle jejich zdravotních anamnézy, jaké onemocnění se u nich mohou objevit.

Jednou z hlavních nevýhod pravděpodobnosti ve strojovém učení je podle [1] jejich výpočetní náročnost. Jejich další nevýhodou jsou vyšší nároky na trénovací množinu dat než běžné metody strojového učení.

4.3.1 Bayesovské sítě

Bayesovské sítě jsou pravděpodobnostní modely využívající grafovou reprezentaci [1]. Podporují interpretaci nových zkušeností na základě dříve naučených vztahů (porozumění současným událostem je funkcí naučených jevů z předchozích událostí). Bayesovská síť je orientovaný acyklický graf (v grafu neexistuje cyklus), který pomocí hran zachycuje závislosti mezi proměnnými a systémem náhodných pravděpodobnostních distribucí. Každý uzel má přiřazenou pravděpodobnostní distribuci

(podle Bayesova pravidla) a ta popisuje pravděpodobnostní závislost mezi vybranými proměnnými [12].

Bayesovo pravidlo je věta teorie pravděpodobnosti, kterou využívají Bayesovské sítě při stanovení hodnot pravděpodobnostní distribuce na základě hodnot jiných uzlů (proměnných) v sítí [1, 6, 13]. Tento princip popisuje, jak se pravděpodobnost jedné události mění v závislostí na jiné. Tento vztah lze zapsat takto:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.2}$$

- P(A|B) je podmíněná pravděpodobnost jevu A za předpokladu, že již došlo k jevu B.
- P(B|A) představuje podmíněnou pravděpodobnost výskytu jevu B za předpokladu, že již došlo k výskytu jevu A.
- P(A) a P(B) představují pravděpodobnosti výskytu jevu A a B.

Proces učení těchto sítí se skládá z 2 hlavních kroků. Jde o strukturální a parametrické učení. Strukturální učení rozhoduje, které proměnné budou součástí sítě a jaké budou mezi nimi závislosti, zatímco parametrické učení řeší pravděpodobnostní rozdělení pro každou proměnnou v síti.

Bayesovské sítě mají řadu využiti, jedním z nich bývá diagnostika a predikce, například v medicíně. Hlavní výhodou Bayesovských sítí je jejich schopnost zpracovávat složité vztahy mezi proměnnými a práce s neúplnými daty.

4.3.2 Markovovy modely

Jedná se o pravděpodobnostní modely, které popisují náhodný proces měnící se v závislosti na čase. Základem těchto modelů je vlastnost *Markov Property*, která se odborně označuje jako *Markovova vlastnost* [1, 10, 11]. Vlastnost obecně říká, že pravděpodobnost přechodu z jednoho stavu do následujícího závisí pouze na aktuálním stavu, a ne na předchozích stavech. Nezáleží tedy na tom, jak jsme se do aktuálního stavu dostali, ale pouze jen na stavu bezprostředně předcházejícím. Není potřeba si pamatovat historii, stačí jen aktuální stav. Díky tomu je možné reprezentovat procesy jako konečné stavové automaty, kde ke každé hraně je připsána její pravděpodobnost. Existují i Markovovy modely vyšších řádů, ve kterých závisí na konkrétním počtu předcházejících stavů.

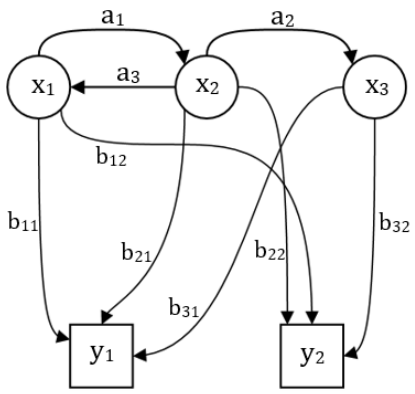
Existuje několik druhů Markovových modelů, můžeme zmínit tyto:

- Markovův řetězec
- Skryté Markovovy modely

Skryté Markovovy modely nebo zkráceně HMM jsou zobecněním Markovova řetězce [1, 10, 11]. V Markovově řetězci reprezentoval každý stav jednu pozorovatelnou událost (například počasí v určitou dobu). HMM je tvořen 2 druhy stavů. Prvním z nich jsou skryté stavy, které nejsou viditelné a druhým typem stavů jsou stavy pozorovatelné stejně jako u Markovova řetězce. HMM lze popsat

jako pozorovatelný stochastický proces pozorovaný dalším skrytým procesem. Na obrázku 4.8 je znázorněn HMM se svými skrytými stavy x_n, pravděpodobnostními přechody a_n, pozorovatelnými výstupními stavy y_n a jejich výstupními přechody b_n.

Aplikaci HMM lze demonstrovat na problému uren, které obsahují míčky [1]. Urny jsou umístěny v místnosti, kam nemá pozorovatel přístup a jen vidí posloupnost vytažených míčku, ale neví, ze kterých uren byly vytaženy. Markovovy modely se používají v mnoha oblastech. Můžeme se s nimi



Obrázek 4.8: Příklad skrytého Markovova modelu.

setkat například v biologii, kde se s jejich pomocí analyzují a modelují DNA sekvence. Používají se také při analýze, generování hlasu nebo psaného textu.

4.4 Umělé neuronové sítě

Umělá neuronová síť (ANN) je modelem, který je inspirován nervovou soustavou, snaží se o napodobení činnosti lidského mozku. Je to jeden z přístupů, kde se uplatňuje biologie, podobně jako při GA. Někdy jsou ANN označovány jako konekcionistické nebo paralelně distribuované systémy (PDP) [1]. Slovo paralelně se zde aplikuje, protože ANN je složena z vrstev propojených umělých *neuronů*, kde každý neuron ve vrstvě zpracovává své vstupní data současně a nezávisle na ostatních. Vazby neboli spojení mezi neurony jsou reprezentovány vahami, které se průběžně upravují podle toho, jestli neuron vede k správné nebo špatné odpovědi. Neuron je tedy základním stavebním kamenem (základní výpočetní jednotka) při tvorbě ANN a v pozdější části si ho rozebereme více.

Kromě vlastností neuronu je ANN dále charakterizována globálními vlastnostmi jako je topologie sítě, algoritmus učení a schéma kódovaní.

Nejčastějším použitím pro ANN přístup je klasifikace, rozpoznávání vzorů, predikce, optimalizace, filtrace šumu [1]. Praktickým příkladem může být rozpoznávání, obrazů nebo hlasů, překlad textu, analýza dat a predikce nemocí. Metody ANN jsou také vhodné k řešení problémů, které

nedokážou symbolické přístupy vyřešit. Typicky u úloh, kde není přesně definována syntaxe nebo jsou vyžadovaný schopnosti založené na vnímání.

Výhodou ANN je jejich výpočetní výkon, díky kterému jsou schopny tréninku sítě na velkém množství dat. Další výhodou je použiti na širokou řadu úloh (klasifikace, regrese, detekce atd.). Dále dokážou v datech najít takové vzory, které člověk nebo běžné metody strojového učení nedokážou identifikovat.

Mezi nevýhody určitě patří jejich nadměrné přizpůsobení určitým datům, když jsou ANN trénovány na příliš malé sadě dat. Nastane to, že ANN bude příliš specializovanou na konkrétní sadu dat a později nebude schopna se vyvíjet na nových datech. Dalším problémem bývá špatné nastavení jejich parametrů, kdy dojde k špatnému vytrénování sítě. V poslední řadě patří mezi zásadní problém komplexnost sítě, kdy příliš rozsáhlou síť bude velice náročné navrhnout.

4.4.1 Model neuronu

Jedním z prvních modelů neuronu byl ten, který navrhli McCulloch a Pitts (1943) [1] a jejich model se dodnes používá. Základním prvkem ANN je neuron, který je představen na obrázku 4.9. Vstupy neuronu $x_1, x_2, ...x_n$ přijímají vstupní signály. Může jít například o již zpracovaná data předchozích vrstev neuronů nebo data ze senzorů. Podoba těchto vstupních dat bývá často ve formě vektorů nebo matic a je určena v závislosti na typu řešeného problému. Ke každému vstupu odpovídá jeho váha (koeficient), která popisuje jeho význam. Na základě $w_1, w_2, ...w_n$ vah se potom rozhoduje, které vstupy upřednostnit při výpočtu.

Po vstupech přichází na řadu výpočetní část neuronu, kde probíhají dva procesy, které jsou podle [14, 15] popsány následovně. Matematicky je lze funkcí neuronu zapsat takto:

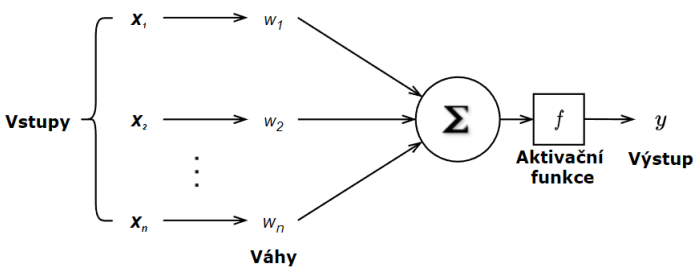
$$y = f(\sum_{i=1}^{n} w_i x_i - \theta) \tag{4.3}$$

Prvním z procesů je vnitřní potenciál neuronu. Vnitřní potenciál je součet vstupujících signálů škálovaných váhou spojení w_i. Prahová hodnota θ určuje, při jaké hodnotě bude neuron aktivován (probuzen). Když bude součet všech vstupů menší než prahová hodnota, tak nedojde k aktivaci a výstup zůstane nezměněn. Druhým procesem je aplikování aktivační funkce f, která vypočte výslednou výstupní hodnotu y.

4.4.2 Návrh umělé neuronové sítě

Při návrhu ANN je potřeba dodržet několik kritérií podle toho, co přesně má síť vykonávat. Jednotlivé kritéria lze souhrne popsat těmito body:

- Definování cíle.
- Vhodný výběr typu a struktury sítě.



Obrázek 4.9: Model umělého neuronu.

- Nastavení parametrů sítě.
- Příprava dat a trénink sítě.

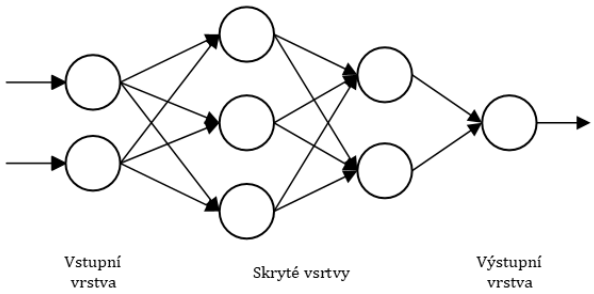
Prvním krokem je stanovit cíl řešení. Dále je důležitý výběr vhodného typu sítě, jestli půjde o použití perceptronu nebo vícevrstvé sítě a k tomu vhodně zvolit strukturu navrhované sítě (množství vstupů, výstupů, skrytých vrstev). Dalším důležitým požadavkem je před připravení dat a následné trénování na těchto datech. Data většinou nelze použít v surovém stavu a musíme je upravit (normalizovat) a poté rozdělit na dvě množiny - trénovací a testovací. Síť se následně trénuje na trénovací množině dat a během toho se vyhodnocuje chybová funkce, která určuje, jak přesně daná síť řeší konkrétní problém. Podle zjištěné chybovosti se upravují jednotlivé váhy a učení sítě se opakuje znova.

4.4.3 Typy sítí

Jednou z prvních sítí podle [1] byl *Perceptron* a jednalo se o síť složenou pouze z jednoho neuronu. Model umělého neuronu jsme si popsali již dříve. Tato síť dokázala řešit jen jednoduché, lineárně separovatelné (binárně klasifikační) problémy. McCulloch a Pitts demonstrovali použití Perceptronu na učení booleovských funkcí součinu a součtu. Při použití Perceptronu na učení složitějších, lineárně neseparovatelných problémů, jakým je například booleovská funkce XOR se zjistilo, že nejde tyto problémy tímto způsobem řešit. Tohle vedlo k návrhu sítí s větší výpočetní sílou, složených z několika perceptronů tzv. vícevrstvé perceptronové sítě.

Vícevrstvá síť je složena z vstupní, výstupní vrstvy a libovolného počtu skrytých vrstev, které se starají o výpočetní část. Podle obrázku 4.10 si lze vícevrstvou síť představit jako orientovaný graf. Z každé vrstvy sítě, která obsahuje libovolný počet neuronů (uzlů) vedou vždy spojení do vyšších vrstev. Počet skrytých vrstev a počet neuronů v nich závisí na složitosti funkce, jakou má síť vykonávat a na typu sítě.

Existuje celá řada dalších druhů ANN, které se liší v jejich architektuře, algoritmech učení [16]. Dalšími typy sítí je například Hopfieldova nebo Kohoenova síť.



Obrázek 4.10: Vícevrstvá neuronová síť.

4.4.4 Proces učení umělých neuronových sítí

Proces učení začíná připravením dostatečné datové sady příkladů, na kterých bude ANN trénována. Kdyby nebyla množina dat dostatečně velká mohlo by dojít k tomu, že se síť příliš specializuje na konkrétní příklady a později se nedokáže vyvíjet dále nad novými příklady.

Na začátku učení se ve většině případech nastavují vstupní váhy na náhodné hodnoty. Pro své učení podle [16] využívají ANN nejčastěji algoritmus zpětného šíření chyby (Backpropagation). Jeho cílem je minimalizovat výstupní chybovou funkci (chybu nebo odchylku), která vzniká jako rozdíl mezi očekávaným výstupem (z trénovací sady dat) a výstupem z ANN. Podle této chyby se zpětně upravují jednotlivé váhy neuronů v síti, aby se docílilo přesnějšího výsledku při dalším opakování učení. Proces učení se vždy provádí v několika iteracích.

Kapitola 5

Případová studie

Tato kapitola je zaměřena na případovou studii, jejíž cílem byla implementace algoritmů na základě symbolické reprezentace znalostí a GA. Oba zmiňované přístupy byly popsány v předchozích kapitolách teoretické části práce.

Cílem prvního algoritmu založeném na symbolickém přístupu je hledat obecnou hypotézu, pomocí které je možné rozpoznávat objekty spadající do třídy konceptu. V tomto případě hledat obecnou hypotézu konceptu oblouku. Algoritmus přijímá na vstupu příklady objektů, které jsou definované jako jednoduché klauzule jazyka Prolog [18] a snaží se podle příkladů oblouku vytvářet obecnou hypotézu, která bude správně rozhodovat, jestli se jedná o oblouk nebo ne.

Druhý algoritmus je zaměřen na hledání cesty v bludišti za pomoci GA. Algoritmus pracuje s populací jedinců, kteří zkoušejí projít bludištěm k cíli a postupně se během toho vyvíjí podle jedinců, kteří byli efektivní v průchodu bludištěm. Základním principem algoritmu je genetický vývoj, kdy postupnou evolucí se jedinci v populaci zlepšují až do doby než se dostanou k cílové pozici v bludišti.

Pro implementaci obou algoritmů jsem zvolil programovací jazyk Python a jeho knihovny, konkrétně ve verzi Python interpretu 3.11. Jeho výhodou je jeho přenositelnost, což znamená, že je spustitelný prakticky na kterémkoliv operačním systému. Jedná se o vyšší programovací jazyk, patřící mezi skriptovací jazyky umožnující ve většině případech psát kratší, lépe čitelné a pochopitelné programy. Naproti tomu nízkoúrovňové programovací jazyky (například jazyk symbolických, programovací jazyk C) poskytují nižší abstrakci a jsou snadněji převeditelné na strojové instrukce.

V následujících dvou částech 5.1 a 5.2 budou zmíněné algoritmy popsány a na závěr této kapitoly budou oba popisované algoritmy srovnány na základě jejich přístupů.

5.1 Vytváření hypotézy podle konceptů

V této části případové studie bylo úkolem naimplementovat algoritmus s využitím symbolické reprezentace znalostí, jejíž teorie byla shrnuta v kapitole 4.1. Konkrétně se jedná o Winstonův algoritmus, který patří do skupiny algoritmů využívající typ učení s učitelem. Cílem algoritmu je nalézt obecnou

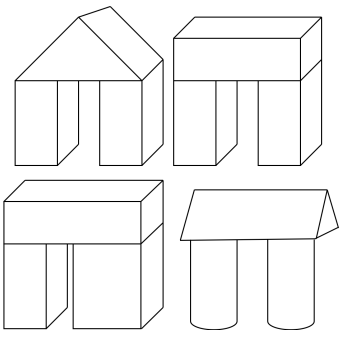
hypotézu, která bude identifikovat nějaký koncept. V tomto případě koncept oblouku složeného z geometrických těles.

Základním principem Winstonova algoritmu je kategorizace objektů prostřednictvím uvádění příkladů objektů, které do dané skupiny patří a objektů, které do ní nepatří [2]. Algoritmus hledá symbolickou reprezentaci nějakého konceptu postupem generalizace a specializace. Generalizace slouží k zobecnění hypotézy a je dosažena prostřednictvím uvádění pozitivních příkladů. Vyloučení vlastností, které objekty dané skupiny nesmí mít, a potvrzení vlastností, které musí mít je prováděno specializací. Konkrétně uváděním negativních příkladů, které do dané skupiny nepatří. Winston ve svém algoritmu pro negativní příklady používá speciální označení *near-miss* neboli "téměř" příklady. Jedná se o příklady podobné pozitivním příkladům, ale liší se od nich v jediném rozdílu, který zahrnuje chybějící nebo nadbytečnou vlastnost.

Cílem algoritmu je tedy nalézt hypotézu (symbolickou reprezentaci popisu konceptu), která v tomto případě popisuje vlastnost "být oblouk". Oblouky se skládají z geometrických těles různých tvarů, barev a pozic. Nalezený koncept by měl být dostatečný obecný, aby identifikoval správně objekty, kteří patří do třídy oblouků, a zároveň dostatečně specifický, aby vyloučil objekty, které do třídy oblouku nepatří.

Algoritmus pracuje s trénovací sadou příkladů poskytnutou učitelem na vstupu, která je složena z příkladů oblouků (pozitivní a negativní příklady oblouku) popsaných pomocí symbolické reprezentace znalostí. Negativní příklady představují objekty, které jsou popsány některými vlastnostmi konceptu oblouku, ale nelze je zařadit do třídy oblouku (konkrétně jim chybí nebo mají navíc nějakou vlastnost).

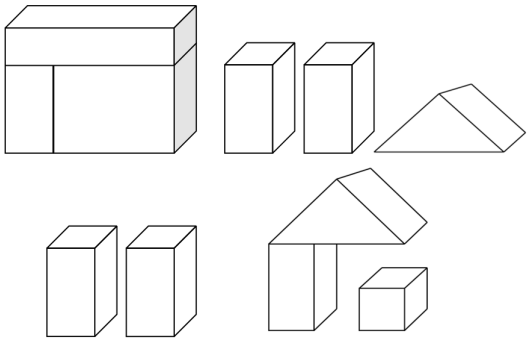
Na obrázku 5.1 je několik příkladů oblouku (pozitivních příkladů), které se liší vzhledem, ale mají několik společných vlastností nebo rysů, jako jsou dva podpěrné sloupy oddělené mezerou, na kterých je postavena střecha. Následně na obrázku 5.2 jsou ilustrovány "téměř" příklady oblouku



Obrázek 5.1: Příklady oblouků.
(pozitivní příklady)

(negativní příklady), které se podobají příkladům oblouku, avšak obsahují jeden rozdíl (některou vlastnost mají navíc nebo jim některá vlastnost chybí), a proto jsou vyloučeny z třídy oblouku. Z obrázku si lze všimnout, že na jednom z příkladů "téměř" oblouku chybí mezera mezi sloupy nebo

na jiném příkladu není střecha umístěna na sloupech. Těmito negativními příklady je prováděna specializace hypotézy, kdy jsou vyloučeny vlastnosti, které oblouk nesmí mít (například sloupy nesmí být vedle sebe) a také potvrzeny vlastnosti, které musí obsahovat (například střecha musí být umístěna na sloupech).



Obrázek 5.2: Příklady "téměř" oblouků.
(negativní příklady)

5.1.1 Popis algoritmu

V této části je popsán princip algoritmu hledající hypotézu. Vstupem algoritmu je sada pozitivních a negativních trénovacích příkladů. Spolu s trénovacím sadou dat je poskytnuta ontologie, která je potřebná k generalizaci modelu a dále seznam vzájemně vylučujících se hodnot pravidel.

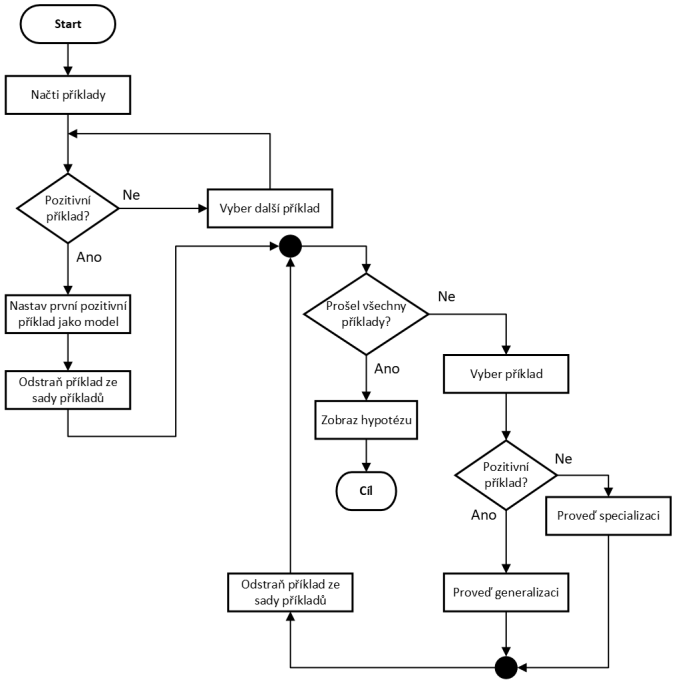
Prvním krokem algoritmu je zvolit pozitivní příklad jako počáteční model hypotézy, který následně bude algoritmus upravovat pomocí trénovacích příkladů. V procesu hledání je hypotéza nazývána modelem hypotézy. Jako počáteční model hypotézy je zvolen první pozitivní příklad nalezený v trénovací sadě příkladů, která mu byla poskytnuta jako vstupní data. Příklad, který byl zvolen jako model je potom vyloučen ze sady příkladů.

Následně jsou postupně vybírány příklady z trénovací sady příkladů a porovnávány s modelem. Pokud je vybrán pozitivní příklad představující oblouk, tak je prováděna generalizace modelu, která slouží k zobecnění modelu a také identifikaci vlastností, které nejsou z pohledu kategorizace důležité (například barva, velikost). V opačném případě, když je vybrán negativní příklad je prováděna specializace modelu, kdy se snaží algoritmus model upřesnit, aby vylučoval příklady, které mají podobné vlastnosti jako oblouk, ale nelze je mezi oblouky zařadit.

Na obrázku 5.3 je znázorněn vývojový diagram popisovaného Winstonova algoritmu hledající hypotézu.

5.1.2 Vstupní data

Před zahájením hledání obecné hypotézy je nezbytné popsat jednotlivé vstupy algoritmu a způsob, jakým jsou reprezentována data.



Obrázek 5.3: Vývojový diagram Winstonova algoritmu hledající hypotézu na základě konceptů.

Vstupem pro tento algoritmus jsou data reprezentována pomocí jednoduchý klauzulí jazyka Prolog. Prolog je logický programovací jazyk, který se skládá z predikátů, logických výrazů a pravidel, pomocí kterých popisuje vztahy mezi různými objekty a pravdivostními hodnotami. Zápis dat pomocí Prologu byl zvolen kvůli tomu, že algoritmy využívající symbolickou reprezentaci znalostí pracují s daty popsánými pomocí logických výrazů a pravidel. Podrobnější dokumentaci jazyka Prolog je možné nalézt v publikaci [18].

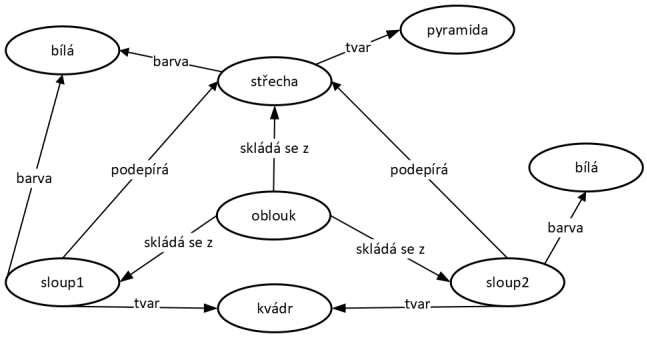
Obsahem vstupních dat je trénovací sada příkladů složená z pozitivních i negativních příkladů oblouku. Tyto příklady jsou uloženy v souboru formátu CSV a jsou poskytnuty algoritmu při jeho spuštění. Příklad oblouku popsány pomocí jazyka Prolog je reprezentován následovně:

$$skládá\_se\_z(oblouk,\ sloup1) \wedge skládá\_se\_z(oblouk,\ sloup2) \wedge skládá\_se\_z(oblouk,\ střecha)$$
$$\wedge$$
$$tvar(sloup1,\ kvádr) \wedge tvar(sloup2,\ kvádr) \wedge tvar(střecha,\ pyramida)$$
$$\wedge$$
$$barva(sloup1,\ bílá) \wedge barva(sloup2,\ bílá) \wedge barva(střecha,\ bílá)$$
$$\wedge$$
$$podepírá(sloup1,\ střecha) \wedge podepírá(sloup2,\ střecha) \supset Oblouk(oblouk)$$

Zapsaný příklad oblouku v jazyce Prolog je tvořen pomocí jednoduchých pravidel, které jsou spojeny logickou spojkou konjunkce. Předchozí uvedený zápis, ale není zcela korektní, protože Prolog pro spojku konjunkce používá znak čárky. Správný zápis oblouku pomocí Prolog klauzulí tak vypadá trochu odlišně. Logické spojky konjunkce jsou nahrazeny čárkami jako na následujícím upraveném příkladu:

$$\textit{skládá\_se\_z(oblouk, sloup1)}, \textit{skládá\_se\_z(oblouk, sloup2)}, \textit{skláda\_se\_z(oblouk, střecha)},$$
$$\textit{tvar(sloup1, kvádr)}, \textit{tvar(sloup2, kvádr)}, \textit{tvar(střecha, pyramida)},$$
$$\textit{barva(sloup1, bílá)}, \textit{barva(sloup2, bílá)}, \textit{barva(strecha, bílá)},$$
$$\textit{podepira(sloup1, střecha)}, \textit{podepira(sloup2, střecha)}$$

Podle výše uvedeného příkladu lze daný oblouk převést pro lepší srozumitelnost do grafické podoby. Používáme k tomu sémantickou síť, kde objekty jsou reprezentovány pomocí uzlů a vztahy mezi nimi hranami. Na obrázku 5.4 je ilustrována tato sémantická síť oblouku, kde je oblouk složen ze 3 základních prvků (sloup1, sloup2 a střecha), které mají určité atributy jako je tvar, barva, a co podepírají. Samozřejmě je možné definovat další pravidla, a tím rozšířit definici oblouku. Například sloupy by měly mezi sebou mít volný prostor.



Obrázek 5.4: Pozitivní příklad oblouku prezentovaný pomocí sémantické sítě.

5.1.3 Heuristiky, generalizace a specializace

Jak bylo uvedeno dříve v této kapitole a také v kapitole 4.1, Winstonův algoritmus pro identifikaci konceptů je založen na heuristikách[1]. Heuristiky vedou k úpravě modelu hypotézy během procházení sady trénovacích příkladů tak, aby nalezená hypotéza dokázala predikovat nad dosud neznámými (neviděnými) daty. Mezi 2 hlavní operace, které využívá algoritmus při úpravě modelu jsou generalizace (zobecnění) a specializace (zpřesnění).

[1] Jedná se o metodu používanou k řešení problémů nebo rozhodování v situacích, kdy neexistuje jednoznačné řešení.

K budování správně hypotézy je nejvhodnější volit trénovací sadu příkladů, která obsahuje pozitivní i negativní příklady oblouku. Pokud by byla hypotéza tvořena jen za pomoci pozitivních příkladů, tak by mohla nastat situace, že hypotéza bude příliš obecná neboli *over-generalization* a důsledkem toho může být, že do skupiny osvojeného konceptu (oblouku) zařadí také objekty, které zde nepatří. Tento problém řeší použití negativních příkladů, jejíž úkolem je zpřesnit hypotézu pouze na oblouky.

5.1.3.1 Specializace

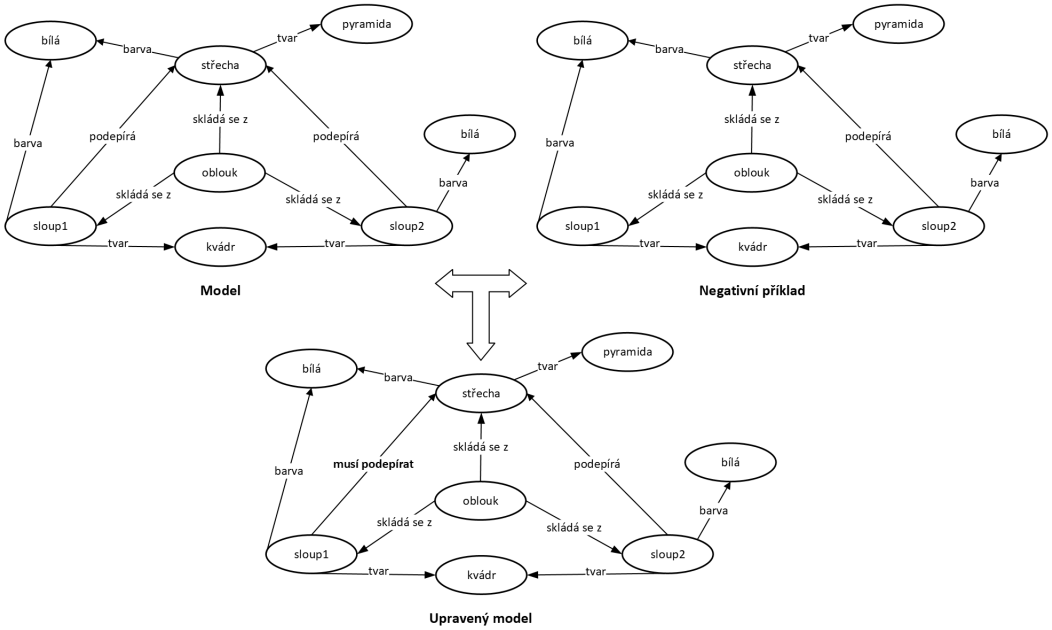
Specializace se používá k upřesnění modelu hypotézy na základě negativního příkladu oblouku. Prvním krokem specializace je porovnání hledané hypotézy (modelu hypotézy) s negativním příkladem a nalezení jednoho zásadního rozdílu (*signicicant difference*) mezi nimi. Jako zásadní rozdíl si lze představit jeden z důležitých prvků, ze kterých je oblouk složen. Oblouk je složen ze 2 sloupů s mezerou mezi nimi a střechou, kterou sloupy podepírají. Pokud je zásadní rozdíl mezi modelem a negativním příkladem nalezen, tak dochází podle [5] k úpravě modelu na základě jedné z těchto operací:

- **Require-link**, pokud model hypotézy obsahuje atribut, který není součástí negativního příkladu, tak je tento atribut označen jako povinný.
- **Forbid-link**, pokud negativní příklad obsahuje atribut, který není součástí modelu hypotézy, tak je tento atribut přidán do modelu hypotézy a označen jako vylučující.

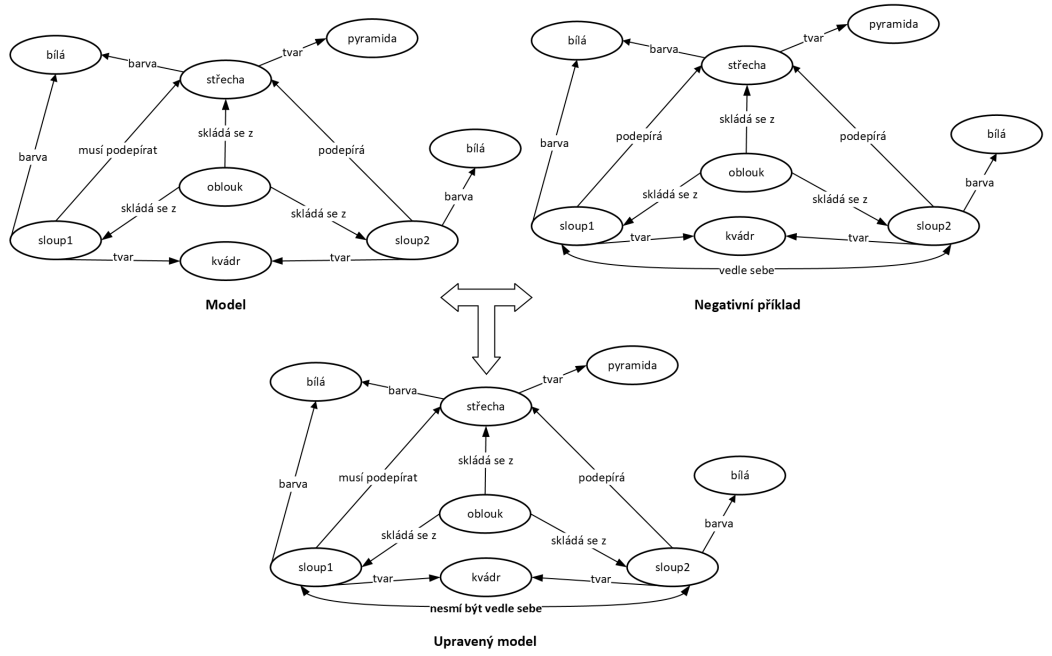
V případě, že během porovnání nebyl nalezen žádný zásadní rozdíl, tak žádná z uvedených úprav neproběhne. Negativní příklad je ignorován a je místo něho poskytnut algoritmu další.

Require-link je heuristika používaná k označení atributu v modelu, který chybí v porovnávaném negativním příkladu. Z takto označených atributů se následně stávají povinné atributy, které musí být součástí modelu, aby bylo možné správně predikovat oblouk. Označené atributy jako povinné nelze dále už odstranit, ale je možné na ně aplikovat heuristiky zaměřené na generalizaci. Označení atributu jako povinného může být realizováno například použitím prefixu *musí být* (must-be). Na obrázku 5.5 je ilustrován příklad aplikace této heuristiky, kde je model hypotézy oblouku a negativní příklad, kterému chybí atribut *podepírá(sloup1, střecha)*. V upraveném modelu hypotézy je atribut označen jako povinný pomocí "musí", protože je nezbytný k predikci oblouku.

Ve **Forbid-link** heuristice jde o podobný princip jako v Require-link. Jen je zde rozdíl v tom, že se používá ve chvíli, kdy v modelu chybí atribut, který je obsažen v negativním příkladu. Chybějící atribut je přidán do modelu a označen opět prefixem (*nesmí být*) jako vylučující. Na obrázku 5.6 je ilustrován případ, kdy poskytnutý negativní příklad obsahuje atribut, který říká, že *sloup1* a *sloup2* stojí vedle sebe. Tento atribut avšak chybí v modelu hypotézy. Model hypotézy je upraven přidáním tohoto atributu s označením "nesmí být". Tím je zajištěno vyloučení příkladů se sloupy postavenými vedle sebe.



Obrázek 5.5: Příklad použití Require-link k úpravě modelu hypotézy.



Obrázek 5.6: Příklad použití Forbid-link k úpravě modelu hypotézy.

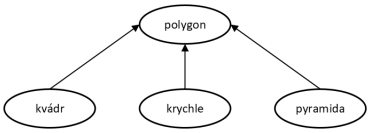
5.1.3.2 Generalizace

Generalizace probíhá podobně jako specializace jen je v tomto případě porovnáván model hypotézy s pozitivním příkladem. Úkolem generalizace je upravit model hypotézy oblouku tak, že zobecňuje atributy jako například barva a tvar, které nejsou striktně omezeny na konkrétní hodnoty k predikci oblouku. Pro každý nalezený rozdíl mezi modelem hypotézy a pozitivním příkladem jsou prováděny následující úpravy modelu:

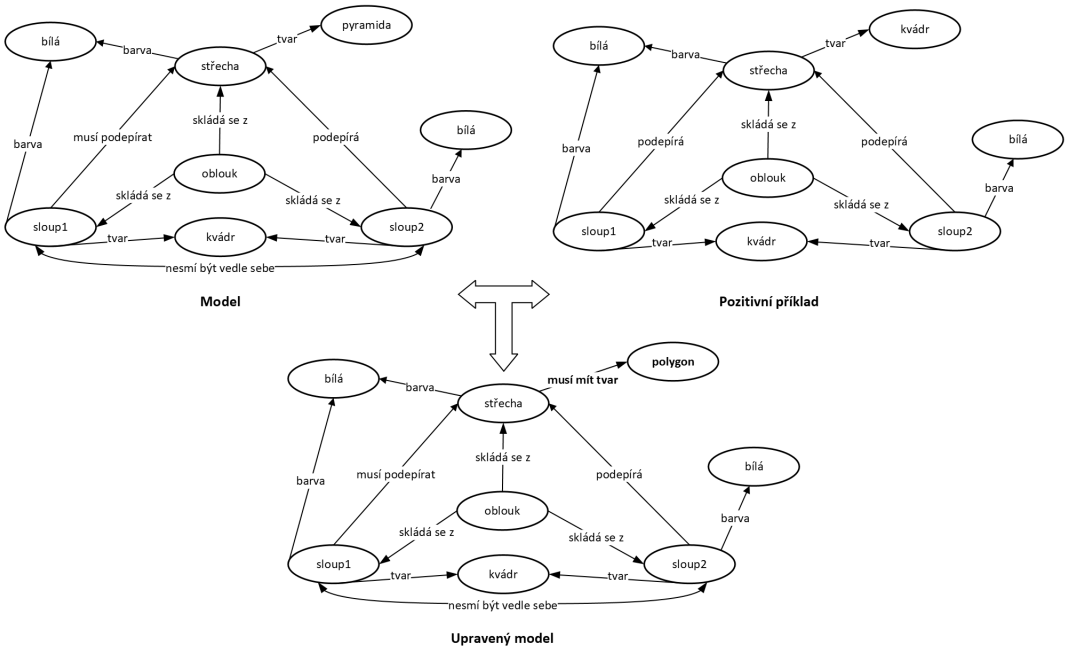
- **Climb-tree**, pokud pro atribut s rozdílnou hodnotu v modelu hypotézy i pozitivním příkladu existuje obecná třída (ontologie), kterou je možné obě hodnoty nahradit.
- **Drop-link** je použit v případě, že model hypotézy a pozitivní příklad obsahují vylučující se hodnoty ve stejném atributu. Úprava odstraňuje konkrétní atribut z modelu hypotézy.
- **Enlarge-set**, pokud model hypotézy a pozitivní příklad obsahují ve stejném atributu odlišné hodnoty, pro které neexistuje žádná obecná třída, tak je atribut rozšířen o hodnotu z pozitivního příkladu.
- **Drop-link**, pokud v modelu hypotézy je atribut, který není součástí pozitivního příkladu, tak je atribut z modelu hypotézy odstraněn.
- **Close-interval** je použit v případě, kdy se atribut v modelu hypotézy a příkladu liší v číselné hodnotě nebo intervalu hodnot. Pokud atribut modelu hypotézy už obsahuje interval hodnot, tak je tento interval rozšířen o hodnotu z pozitivního příkladu. Pokud však atribut modelu hypotézy obsahuje jedinou číselnou hodnotu, tak je pomocí hodnoty z pozitivního příkladu upraven vytvořením intervalu těchto dvou hodnot.

V případě, že nebyl nalezen žádný rozdíl mezi modelem hypotézy a pozitivním příkladem, tak je tento příklad ignorován a je poskytnut algoritmu další. Všechny zmíněné úpravy si dále podrobněji popíšeme na příkladech.

Climb-tree heuristika je používána v situacích, kdy je model hypotézy příliš specializovaný. Cílem této heuristiky je nalézt pro rozdílné hodnoty stejného atributu nějakou obecnou třídu, která obě hodnoty nahradí a zobecní tím upravený model. Algoritmu je poskytnuta ontologie jako na obrázku 5.7, která obsahuje informace o konkrétním atributu, jeho obecné třídě a z čeho je složena. Obecně řečeno máme nějakou abstraktní třídu (například polygon), ze které dědí několik podobných specifičtějších tvarů (například kvádr, krychle, pyramida). Podle ontologie a informací, které obsahuje potom algoritmus rozhodne, jak zobecnit daný atribut, když se model hypotézy a poskytnutý pozitivní příklad liší v hodnotě atributu, která je obsažena v ontologii. Na obrázku 5.8 je uveden příklad zobecnění modelu hypotézy podle ontologie z obrázku 5.7. Vidíme, že model hypotézy a pozitivní příklad se liší tvarem střechy, který je obsažen v ontologii. Proběhne upravení modelu nahrazením specifické hodnoty obecnější hodnotou polygon a označení atributu jako povinného (*musí mít*).



Obrázek 5.7: Příklad ontologie tvaru střechy.



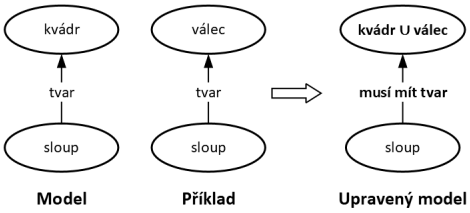
Obrázek 5.8: Příklad použití Climb-tree k úpravě modelu hypotézy.

Enlarge-set heuristika je používána v případech, kdy máme v modelu hypotézy a poskytnutém pozitivním příkladu stejný atribut s rozdílnou hodnotou, pro které neexistuje žádná společná třída k zobecnění. Je nutné modifikovat atribut tak, aby akceptoval i jinou hodnotu než, která je momentálně definována v modelu hypotézy. Řešením je sjednocení, pomocí kterého se rozšíří atribut o další hodnotu. Na obrázku 5.9 je znázorněn příklad této úpravy, kde část modelu hypotézy předpokládá, že sloup má tvar kvádru, avšak je zde poskytnut pozitivní příklad, který se liší v tom, že má tvar válce. Pro obě hodnoty atributu neexistuje žádná obecná třída, a tak je nutné použít Enlarge-set k rozšíření atributu modelu pomocí sjednocení o hodnotu z pozitivního příkladu. Upravený model hypotézy bude zobecněn na možnost mít tvar kvádru nebo válce.

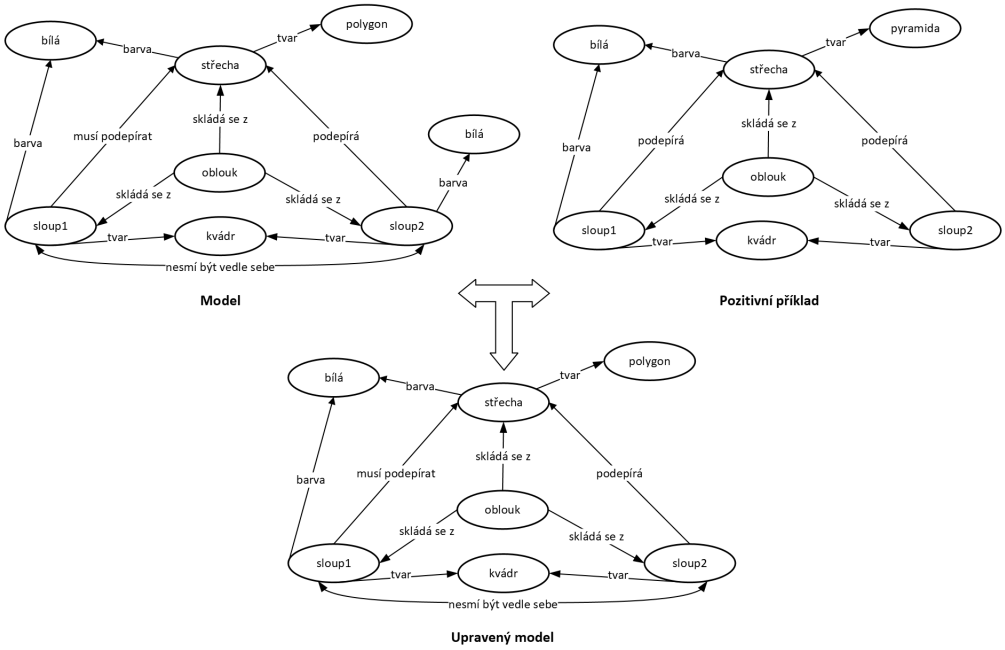
Drop-link je používán k odstranění atributu z modelu hypotézy, který není součástí poskytnutého pozitivního příkladu nebo pokud se dvě související hodnoty atributu mezi sebou vylučují (například sloup nemůže ležet a stát zároveň). Nejčastěji nastává situace, kde je právě odstraněn atribut z modelu hypotézy, který není důležitý k rozpoznávání.

Příkladem může být barva, která je pro sloup v modelu zadána, ale v poskytnutém pozitivním příkladu tato informace chybí. Dojde k jejímu odstranění v modelu, protože je zanedbatelné, jakou

bude mít sloup barvu. Popisovaný příklad je ilustrován na obrázku 5.10.

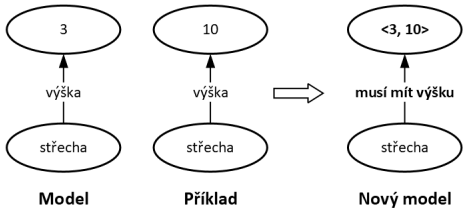


Obrázek 5.9: Příklad použití Enlarge-set k úpravě modelu hypotézy.



Obrázek 5.10: Příklad použití Drop-link k odstranění nedůležitého atributu z modelu hypotézy.

Close-interval je heuristika používaná v případech, kdy atributy v modelu hypotézy a pozitivním příkladu obsahují číselné hodnoty nebo intervaly. Pokud je nalezen atribut, který se liší pouze v číselných hodnotách, tak je z těchto hodnot vytvořen interval hodnot. V případě, že atribut v modelu hypotézy již obsahuje interval hodnot, tak je pouze rozšířen tak, aby zahrnoval i hodnotu z poskytnutého pozitivního příkladu. Na obrázku 5.11 je ilustrován příklad použití uvedené heuristiky pomocí sémantické sítě. Na obrázku je zobrazena část modelu, která popisuje střechu, která má výšku o hodnotě 3. K tomuto modelu je poskytnut pozitivní příklad, který uvádí, že střecha má výšku 10. Na základě tohoto příkladu je upraven atribut v modelu nahrazením hodnoty intervalem $< 3, 10 >$ a označením prefixem *musí mít*.



Obrázek 5.11: Příklad použití Close-interval k úpravě modelu hypotézy.

5.1.4 Příklad hledání konceptu oblouku

V této části bude prezentováno hledání konceptu oblouku na základě několika příkladů z trénovací sady dat uvedených níže.
Pozitivní příklady:

1. $$skládá\_se(oblouk,\ sloup1), skládá\_se(oblouk,\ sloup2), skládá\_se(oblouk,\ střecha),$$
$$tvar(sloup1,\ kvádr), tvar(sloup2,\ kvádr), tvar(střecha,\ pyramida),$$
$$barva(sloup1,\ bílá), barva(sloup2,\ bílá), barva(střecha,\ bílá),$$
$$podepírá(sloup1,\ střecha), podepírá(sloup2,\ střecha)$$

2. $$skládá\_se(oblouk,\ sloup1), skládá\_se(oblouk,\ sloup2), skládá\_se(oblouk,\ střecha),$$
$$tvar(sloup1,\ kvádr), tvar(sloup2,\ kvádr), tvar(střecha,\ pyramida),$$
$$barva(sloup1,\ bílá), barva(sloup2,\ bílá),$$
$$podepírá(sloup1,\ střecha), podepírá(sloup2,\ střecha)$$

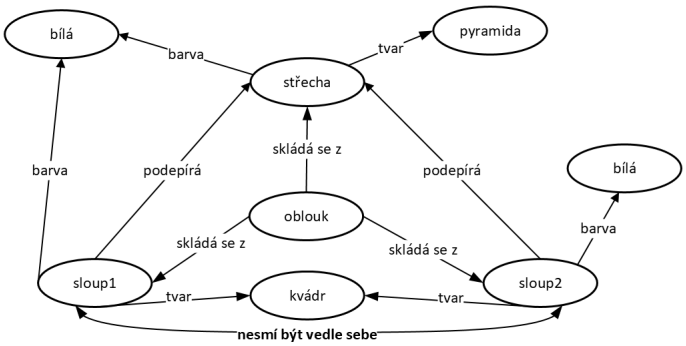
Negativní příklady:

1. $$skládá\_se(oblouk,\ sloup1), skládá\_se(oblouk,\ sloup2), skládá\_se(oblouk,\ střecha),$$
$$tvar(sloup1,\ kvádr), tvar(sloup2,\ kvádr), tvar(střecha,\ pyramida),$$
$$barva(sloup1,\ bílá), barva(sloup2,\ bílá), barva(střecha,\ bílá),$$
$$podepírá(sloup1,\ střecha), podepírá(sloup2,\ střecha), vedle\_sebe(sloup1,\ sloup2)$$

2. $$skládá\_se(oblouk,\ sloup1), skládá\_se(oblouk,\ sloup2), skládá\_se(oblouk,\ střecha),$$
$$tvar(sloup1,\ kvádr), tvar(sloup2,\ kvádr), tvar(střecha,\ pyramida),$$
$$barva(sloup1,\ bílá), barva(sloup2,\ bílá),$$
$$podepírá(sloup1,\ střecha)$$

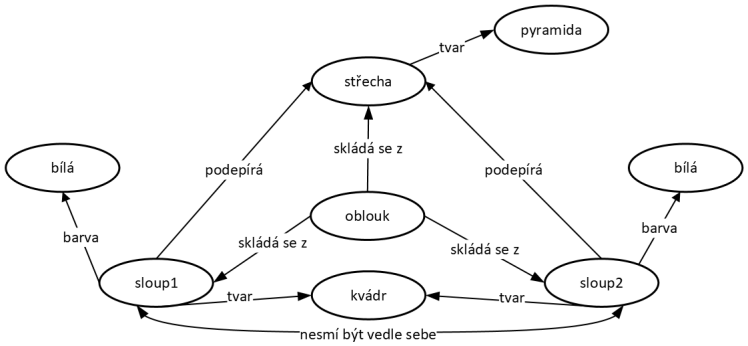
V prvním kroku algoritmus nastaví jako počáteční model první pozitivní příklad ze sady trénovacích příkladu. Počátečnímu modelu odpovídá sémantická síť na obrázku 5.4.

Po nastavení počátečního modelu hypotrézy je následně algoritmu poskytnut první negativní ("téměř") příklad, který se od modelu hypotézy odlišuje tím, že jsou sloupy vedle sebe. Když je poskytnut negativní příklad, tak algoritmus provede specializaci. V tomto případě bude provedena heuristika Forbid-link, která rozšíří model hypotézy o atribut říkají, že sloupy nesmí být vedle sebe. Na obrázku 5.12 je ilustrována sémantická síť upraveného modelu hypotézy.



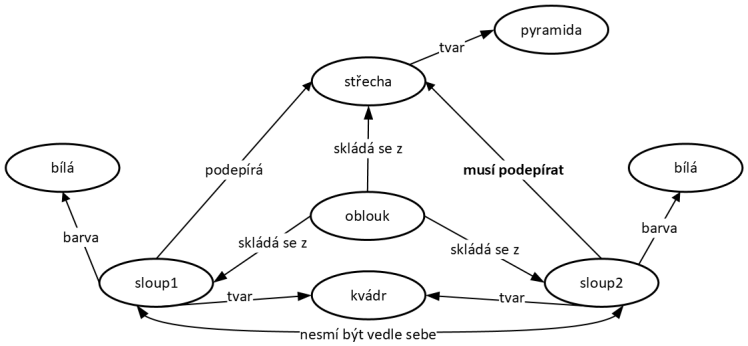
Obrázek 5.12: Sémantická síť upraveného modelu hypotézy
po aplikaci heuristiky Forbid-link.

Nyní bude poskytnut algoritmu druhý pozitivní příklad, který se od modelu liší v tom, že neobsahuje atribut o barvě střechy. Proběhne tedy generalizace modelu pomocí heuristiky Drop-link, kdy bude nepotřebný atribut odstraněn jako na obrázku 5.13. Uvedená sémantická síť představuje upravený model hypotézy.



Obrázek 5.13: Sémantická síť upraveného modelu hypotézy
po aplikaci heuristiky Drop-link.

Dále je poskytnut druhý negativní příklad, který má jediný rozdíl v tom, že střecha je podepírána pouze jedním ze sloupů. Proběhne specializace modelu pomocí heuristiky Require-link jako na obrázku 5.14, kdy bude chybějící atribut z negativního příkladu označen jako povinný.



Obrázek 5.14: Sémantická síť upraveného modelu hypotézy
po aplikaci heuristiky Require-link.

Tímto způsobem jsou aplikovány heuristiky generalizace a specializace na další předložené příklady.

5.1.5 Výstup

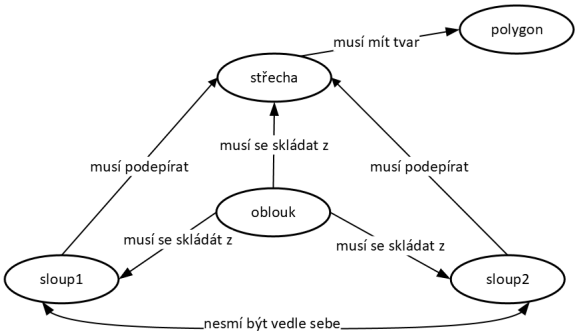
Výsledkem této části případové studie byla implementace Winstona algoritmu pro rozpoznávání konceptů, konkrétně rozpoznávání konceptu oblouku. Algoritmus vytváří obecnou hypotézu pomocí trénovacích příkladů, které jsou mu postupně překládány od učitele. Výstupem algoritmu je symbolická reprezentace hypotézy oblouku, která je uvedena níže:

$$musí\_se\_skládat\_z(Oblouk,\ Sloup1), musí\_se\_skládat\_z(Oblouk,\ Sloup2),$$
$$musí\_se\_skládat\_z(Oblouk,\ Střecha), musí\_mít\_tvar(Střecha,\ polygon),$$
$$musí\_podepírat(Sloup1,\ Střecha), musí\_podepírat(Sloup2,\ Střecha)$$
$$nesmí\_být\_vedle\_sebe(Sloup1,\ Sloup2), nesmí\_být\_vedle\_sebe(Sloup2,\ Sloup1)$$

Tato hypotéza je dále ilustrována pomocí sémantické sítě na obrázku 5.15. Hypotéza definuje oblouky tak, že musí být složeny ze tří částí (sloupy a střecha). Sloupy musí podepírat střechu a nesmí být vedle sebe. Střecha oblouku musí odpovídat tvarům definovaným v ontologii (krychle, kvádr, pyramida) algoritmu.

Na základě nalezené hypotézy lze následně otestovat její přesnost správné identifikace předkládáním testovacích příkladů, které se liší od trénovacích v tom, že algoritmus neví, zda mu byl poskytnut pozitivní nebo negativní příklad.

Součástí implementace je možnost si nechat vykreslit nalezenou hypotézu pomocí jednoduché sémantické sítě a dále je možnost testování příkladů podle nalezené hypotézy. Testování hypotézy probíhá náhodně pro celou sadu tetovacích příkladů, kdy jsou postupně vybírány testovací příklady (trénovací příklady bez hodnoty identifikující oblouk) a porovnávány s hypotézou. U každého příkladu je prováděna kontrola jednotlivých atributů, jestli obsahuje všechny povinné atributy a



Obrázek 5.15: Sémantická síť výsledné hypotézy oblouku.

neobsahuje atributy, které nesmí oblouk obsahovat. Dále je prováděna kontrola pro obecné hodnoty, jestli hodnota příkladu a modelu spadá pod nějakou obecnější hodnotu, která je společná pro oba.

Poté co jsou všechny atributy ověřeny, tak je zobrazen výsledek s informací, jestli se jedná o oblouk nebo ne. Tento výsledek je na závěr porovnán s očekávaným výsledkem pro daný příklad.

5.2 Hledání cesty bludištěm

Cílem tohoto algoritmu je nalezení cesty mezi dvěma body ve čtvercovém bludišti libovolně definované velikosti. Algoritmus hledání cesty v bludišti je založen na GA, které byly popsány v kapitole 4.2. Základem implementovaného algoritmu je genetický vývoj náhodně vytvořené počáteční populace jedinců (prvotní generace), kde každý jedinec disponuje náhodně vygenerovaným chromozomem, který určuje, jak se má jedinec postupně pohybovat v bludišti. V chromozomu jsou uloženy informace o pohybu jedince bludištěm, kde jednotlivé geny, ze kterých je chromozom jedince složen představuje jeden krok nebo pohyb bludištěm.

Genetický vývoj probíhá na základě úspěšnosti jedinců. Každému jedinci je pomocí speciální funkce (fitness funkce) přiřazena číselná hodnota, která vyjadřuje jeho úspěšnost (fitness hodnota) v průchodu bludištěm k cílové pozici. Fitness funkce počítá vzdálenost mezi cílovou pozicí a jedincem dosaženou pozicí. Podle fitness hodnoty jsou následně prováděny operace vývoje (selekce, křížení a mutace), pomocí kterých vzniká nová populace jedinců (nové generace). Z populace jsou vybráni nejlepší jedinci, kteří se dostali nejblíže k cílové pozici, podle kterých je prováděno křížení s ostatními jedinci v populaci. Během křížení dojde k vytvoření nových jedinců (potomků), kteří vznikli kombinací chromozomů dvou jedinců (rodičů).

Na křížení navazuje operace mutace, jejíž úkolem je zavést do nově vytvořených potomků náhodnost nebo diversitu tím, že je náhodně vybraným jedincům pozměněna část jejich chromozomu (gen).

Proces vývoje se pak opakuje, dokud není nalezeno požadované řešení (nalezena cesta v bludišti) nebo pokud není algoritmus zastaven nějakým zastavujícím pravidlem jako například maximálním počtem vytvořených generací jedinců.

V následujících částech je představený algoritmus podrobněji popsán.

5.2.1 Popis algoritmu

Základem algoritmu je vytvoření počáteční populace jedinců, kde každý jedinec obsahuje náhodně vygenerovaný seznam kroků (chromozom), který určuje, jak se má v bludišti postupně pohybovat. Dále jsou součástí každého jedince kromě jeho chromozomu následující informace:

- **Aktuální pozice** v bludišti, kterou je potřeba zaznamenávat kvůli pohybu v bludišti a také k vyhodnocení fitness funkce.
- **Fitness hodnota** vyjadřující kvalitu jedince. V tomto případě vzdálenost mezi aktuální pozicí jedince a cílovou pozici. Čím menší hodnota, tím blíže se jedinec nachází k cíli.
- **Seznam navštívených míst** v bludišti evidující již navštívené místa. Slouží k eliminaci pohybu jedince na pozice, které už dříve navštívil.
- Informace o **nalezení cesty** k cílové pozici.
- **Schopnost pohybu** vyjadřující, jestli se jedinec zastavil nebo se stále pohybuje v bludišti.

Každý jedinec vykonává vygenerované kroky pohybu, dokud nenarazí do zdi, nenajde hledanou cílovou pozici nebo pokud se nepokusí dalším krokem vrátit zpět na pozici, kterou již má uloženou v seznamu navštívených míst. V kódu A.1 je ukázka toho, jak je kontrolováno, jestli je možné udělat následující krok ze seznamu kroků v chromozomu jedince a jeho následné provedení. Z kódu si lze všimnout, že je zabráněno zpětnému pohybu jedince tím, že se kontroluje, jestli nově vypočtená pozice jedince není již obsažena v seznamu, který ukládá navštívené pozice. Pokud tato situace nastane, tak jedinec nezastaví svůj pohyb bludištěm, ale pokusí se najít jiný možný krok a ten provede. Tím se docílí toho, že jedinec bude více prozkoumávat bludiště než, aby se ihned zastavil poté, co se pokusí vrátit na místo, které už jednou navštívil.

V momentě, kdy jedinci skončí s pohybem po bludišti, tak dochází k vyhodnocení fitness hodnoty jedinců. Hodnota fitness je měřítkem toho, jak úspěšný byl jedinec v procházení bludištěm. K tomu, aby bylo možné získat ohodnocení každého jedince je potřeba fitness funkce, která vyhodnocuje, jak blízko se jedinec dostal k cílové pozici. Existuje celá řada metrik zabývajících se vzdáleností jako je například Euklidova nebo Manhattanská vzdálenost, které vyjadřují vzdálenost mezi dvěma body. Pro tento problém byla zvolena metrika, která počítá vzdálenost jako počet dílků od aktuální pozice k cílové pozici a neignoruje zdi v bludišti. Úkolem fitness funkce je nalézt cestu mezi aktuální pozicí jedince a cílovou pozicí.

Po vyhodnocení celé populace dochází k selekci, křížení a mutaci nad populací. Během operace selekce je nejprve celá populace seřazena od nejmenšího po největší podle fitness hodnoty, která udává, jak daleko jsou jedinci od cílové pozice a nalezení cesty bludištěm. Následně na to je vybráno z populace několik lepších jedinců, kteří jsou použiti ke křížení s ostatními. Vybere se lepší jedinec a provede se s ním křížení se všemi horšími jedinci s tím, že je eliminována možnost provést křížení s identickými jedinci.

Operace křížení dostane dva vybrané jedince jako rodiče a vytvoří z nich nového, lepšího potomka, který dědí informace z obou svých rodičů. Nově vytvořený potomek získá seznam kroků, který je částečné složen z jeho rodičů. Z ukázky kódu 5.1 si můžete všimnout, že je nejprve náhodně vygenerováno celé číslo, podle kterého jsou seznamy rodičovských jedinců rozděleny na dvě části a následně spojeny do nového seznamu obdobně jako na obrázku 4.6 s tím, že bude ve většině případů zvýhodňován lepší jedinec. Nové vytvořený seznam kroků (chromozom) je následně uložen do seznamu, do kterého se postupně ukládají nově vytvoření potomci. Takto probíhá proces křížení, dokud není vytvořen stejný počet potomků jako bylo jedinců v populaci.

```
def crossover(array1, array2):
    # vygenerování náhodné pozice k rozdělení pole
    cut_point = random.randint(0, len(array1) - 1)
    # 95% času se bude podmínka splněna
    if random.random() <= 0.95:
        # vzniká nové pole složené z částí array1 a array2
        new_arr = np.concatenate((array1[:cut_point], array2[cut_point:]))
    else:
        # vzniká nové pole složené z částí array2 a array1
        new_arr = np.concatenate((array2[:cut_point], array1[cut_point:]))
    return new_arr
```

Zdrojový kód 5.1: Křížení dvou jedinců.

Dále je nutné provést mutaci u potomků, kteří vznikli křížením. Účelem mutace je zavést do populace rozmanitost. Mutace umožňuje například pomocí náhodné změny některých genů přinést do populace nová řešení, která by nemohla vzniknout jen za pomocí selekce a křížení. Pro mutaci byl zvolen postup, kdy je nejprve stanoveno pro kolik jedinců z populace bude mutace prováděna, a náhodně vybráno, na kterých pozicích přesně se má v seznamu obsahující všechny seznamy kroků provést mutace. Poté už se jen přepočte pozice na konkrétního potomka, na kterého má být aplikována mutace a na pozici v jeho seznamu směrů, která má být změněna. Pomocí mutace je možné zabránit uvíznutí populace v místech vhodných pro procházení, ale nevedoucích k žádnému lepšímu výsledku. Například se může jednat o mrtvé konce v bludišti, které mohou být optimálními řešeními, ale vedou k uvíznutí.

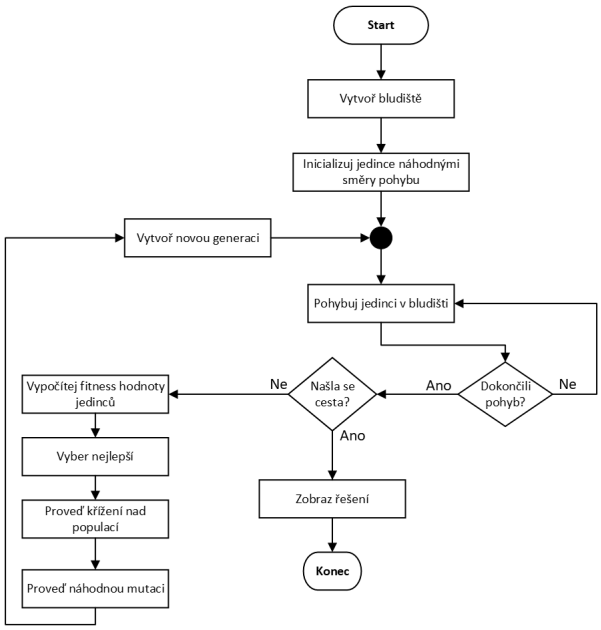
Po dokončení křížení a mutace přichází na řadu poslední část algoritmu, kdy je vytvořena nová generace jedinců. Jsou vynulovány počítadla, která zaznamenávají aktuální tah a zvýšen počet již vytvořených generací.

Popisovaný algoritmus se následně opakuje, dokud určité procenta jedinců z populace nenalezne cestu bludištěm nebo skončí, když do definovaného počtu generací nenajde žádný jedinec řešení.

Na závěr, když je nalezena cesta bludištěm, tak je na obrazovku do nového okna vykreslena výsledná cesta.

Součástí programu jsou parametry, které slouží k nastavení bludiště, definování velikosti populace, počtu kroků nebo maximálního počtu generací k nalezení řešení.

Jednoduchý vývojový diagram celého programu je ilustrován na obrázku 5.16.

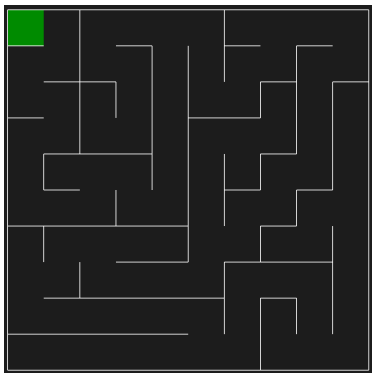


Obrázek 5.16: Vývojový diagram programu hledající cestu v bludišti.

5.2.2 Vytvoření bludiště

Generování bludiště a jeho vizualizace do okna na obrazovku je realizována pomocí modulu Pyamaze, který slouží k usnadnění vytváření náhodného bludiště. Modul je postaven na frameworku Tkinter, který se používá k tvorbě grafických uživatelských rozhraní. Tímto je schopen vygenerované bludiště včetně nalezené cesty zobrazit na obrazovku. Součástí modulu je celá řada před připravených metod a funkcí, kterými je možné ovlivnit jak vzhled, ale také nastavit různé parametry generovaného bludiště. Například generovat bludiště podle uživatelem zadaných rozměrů nebo měnit složitost bludiště podle počtu smyček a cest uvnitř.

Na obrázku 5.17 je ilustrován konkretní příklad vygenerovaného bludiště pomocí modulu Pyamaze, které je dále v tomto textu používáno. Vygenerované bludiště je velikosti 10x10 dílků a obsahuje vždy jen jedinou možnou cestu mezi cílovou pozicí a libovolně vybranou startovní pozicí. Zeleně označená pozice v bludišti reprezentuje cílovou pozici, ke které bude hledána cesta.



Obrázek 5.17: Příklad vygenerovaného bludiště velikosti 10x10 dílků pomocí modulu pyamaze.

5.2.3 Vstupní data

Vstupem algoritmu je několik parametrů ovlivňující jeho činnost, a podle kterých je náhodně vytvořena počáteční populace jedinců. Při spuštění algoritmus přijímá prostřednictvím jednoduchého grafického rozhraní na vstupu následující parametry:

- **Velikost bludiště** udávající, jak velké chceme vytvořit bludiště pomocí modulu Pyamaze.
- **Velikost populace**, která definuje z kolika jedinců bude složena populace.
- **Maximální počet generací**, které je možné vytvořit. Po dosažení maximálního počtu generací algoritmus ukončuje činnost.
- **Míra mutace** vyjadřující s jakou pravděpodobností a na jaké části populace bude prováděna mutace.
- **Mez selekce** určující část lepší populace, ze které budou vybíraní jedinci ke křížení se zbytkem populace.

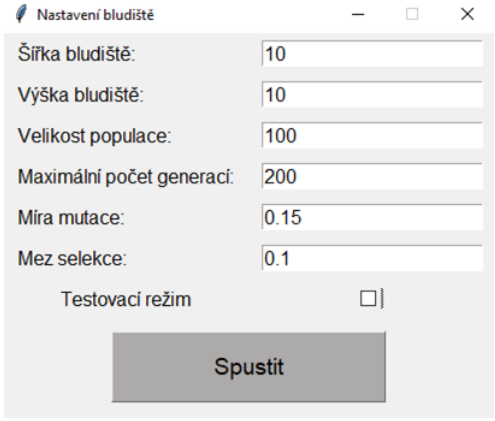
Nastavením těchto parametrů je vytvořena počáteční populace jedinců. Důležitým rozhodnutím při implementaci byla volba podoby dat popisující směr pohybu jedince v bludišti. Jeden krok pohybu je reprezentován jedním ze 4 základním směrů odvozených podle světových stran (sever, jih, východ, západ). Hlavním důvodem, proč byl zvolen tento způsob reprezentace byl kvůli tomu, že modul Pyamaze umožňuje provádět pohyb jedince po bludišti pomocí světových stran. Vygenerováním bludiště tímto modulem vzniká slovník všech pozic ilustrovaný na obrázku 5.18, mapující

strukturu bludiště a jakými směry je možné se pohybovat z daných pozic. Ve slovníku jsou uvedeny všechny pozice pomocí jejich souřadnic v bludišti, ke kterým je přiřazena informace o tom, jakými směry je možné se pohybovat. Hodnota 0 u daného směru definuje, že v tomto směru je zeď. Když je hodnota u směru nastavena na 1, tak je tímto směrem možné se posunout na další pozici.

```
(1, 1): {'E': 1, 'W': 0, 'N': 0, 'S': 0},
(2, 1): {'E': 1, 'W': 0, 'N': 0, 'S': 1},
(3, 1): {'E': 1, 'W': 0, 'N': 1, 'S': 0},
(4, 1): {'E': 1, 'W': 0, 'N': 0, 'S': 1},
(5, 1): {'E': 0, 'W': 0, 'N': 1, 'S': 1},
(6, 1): {'E': 1, 'W': 0, 'N': 1, 'S': 0},
(7, 1): {'E': 0, 'W': 0, 'N': 0, 'S': 1},
(8, 1): {'E': 1, 'W': 0, 'N': 1, 'S': 1},
(9, 1): {'E': 1, 'W': 0, 'N': 1, 'S': 0},
(10, 1): {'E': 1, 'W': 0, 'N': 0, 'S': 0},
...
(10, 10): {'E': 0, 'W': 1, 'N': 1, 'S': 0}
```

Obrázek 5.18: Příklad slovníku mapující strukturu bludiště 10x10.

Na obrázku 5.19 je prezentováno úvodní uživatelské rozhraní, pomocí kterého jsou jednotlivé parametry nastavovány před spuštěním samotného algoritmu. Konkrétně je nastaveno, že bludiště bude velikosti 10x10 dílků, populace bude složena ze 100 jedinců a bude možné algoritmem maximálně vytvořit 200 generací k nalezení řešení. Míra mutace, udávající jak velká část populace bude mutována, je nastavena na hodnotu 0.15. V tomto případě bude mutována pouze čtvrtina populace. Mez selekce je nastavena na 0.1. Před spuštěním algoritmu je možné spustit testovací režim, při kterém je po nalezení cesty bludištěm generováno nové bludiště, a tak je možné testovat algoritmus na další vytvořené bludiště.



Obrázek 5.19: Uživatelské rozhraní k nastavení parametrů algoritmu.

Na základě reprezentace jednoho kroku pohybu bylo možné vytvořit seznam kroků (chromozom) jedince. Pomocí náhodného výběru ze 4 možností popsaných dříve se pro každého jedince v populaci vytvoří počáteční seznam kroků o specifikované délce, která odpovídá druhé mocnině velikosti bludiště. Vytvořený seznam kroků (chromozom) poté vypadá následovně jako na obrázku 5.2, kde jednotlivé kroky v seznamu určují, jakým směrem se má jedinec pohybovat po bludišti velikosti 10x10 dílků. Při této uvedené velikosti bludiště má každý jedinec v populaci seznam kroků o délce 100.

```
['W','W','N','W','W','W','W','W','W','W','S','S','E','N','E','N','E','E','W','W',
'E','N','E','S','E','S','N','E','W','W','E','S','E','S','S','N','S','S','S','W',
'W','W','W','N','S','N','S','N','E','S','E','S','S','S','S','E','W','W','W','W',
'N','W','N','S','S','N','S','N','W','W','E','S','E','N','W','W','S','E','E','N',
'N','W','W','E','S','N','S','E','S','N','S','W','E','S','N','W','E','W','S','W']
```

Zdrojový kód 5.2: Příklad chromozomu jedince reprezentovaný jako seznam kroků v bludišti 10x10.

V této části byly představeny konkrétní příklady vstupů algoritmu, na kterých bude dále představena činnost dalších částí algoritmu.

5.2.4 Pohyb po bludišti

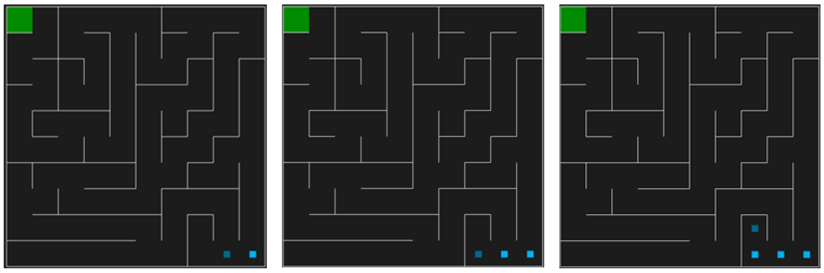
Pohyb jedince po bludišti je realizován pomocí vygenerované seznamu kroků (chromozom). Pohyb vždy začíná ze startovní pozice, která je každému jedinci při vytvoření počáteční generace nastavena na pozici v pravém dolním rohu bludiště. Pohyb každého jedince, tak probíhá krok po kroku podle jeho seznamu kroků. Postupně vybírá ze seznamu kroků následující krok, který má udělat. Nejprve podle vybraného kroku zjistí, jestli je krok možné provést. Vypočítá se pozice, na kterou se krokem dostane. Výpočet probíhá tak, že se k aktuální pozici jedince přičte nebo odečte hodnota 1 podle toho, jakým směrem se zrovna má posunout.

Pro příklad pohybu použijeme chromozom z 5.2. Předpokládejme, že jedinec je ve startovní pozici [10,10] a pokusí se provést první krok. Prvnímu kroku odpovídá 'W', který reprezentuje pohyb o jednu pozici vlevo. Dojde tak k posunu jedince o jednu pozici vlevo, které odpovídá souřadnice [10,9]. Podle nově vypočtené pozice proběhne kontrola, jestli se jedinec nepokusil projít zdí v bludišti. Ve slovníku všech pozic v bludišti se vyhledá, jestli je tento krok možný provést. Pokud je tedy všechno v pořádku, tak se tento krok provede a zaznamená se do seznamu navštívených míst předchozí pozice. Pokud však není možné provést krok vybraným směrem, tak se jedinec ihned nezastaví, ale pokusí se najít krok, který by z daného místa mohl provést. Pro aktuální pozici se vyhledají všechny možné kroky, které je možné provést bez toho, že by jedinec narazil do zdi a nebo provedl krok na již dříve navštívenou pozici. Z nalezených možností se poté náhodně vybere jeden krok, který jedinec provede. Tento krok si pak uloží na aktuální pozici v seznamu kroků. Pokud nebyl

nalezen žádný možný proveditelný krok, tak jedinec zastavuje svůj pohyb. Tímto je realizováno, že jedinci se pokouší prozkoumávat bludiště a nezastaví se ihned po prvním nárazu do zdi.

Na obrázku 5.20 je ilustrován popisovaný pohyb jedince po bludišti pomocí seznamu kroků z 5.2, podle kterého jedinec provedl 3 kroky - vlevo, vlevo, nahoru ('W', 'W' a 'N') a zastavil se, protože narazil na slepou uličku a zpět se taky vrátit nemůže.

Takto postupně provádí jednotlivé kroky po bludišti celá populace, dokud se všichni jedinci nezastaví nebo pokud nenaleznou cílovou pozici a tím cestu v bludišti. Na dokončení pohybu jedinců následně navazuje výpočet kvality jedinců podle fitness funkce, která bude popsána dále.



Obrázek 5.20: Příklad pohybu jedince podle seznamu kroků.

5.2.5 Fitness funkce

Fitness funkce v tomto algoritmu ohodnocuje jedince podle vzdálenosti mezi jejich aktuální pozicí a cílovou pozicí. Úkolem fitness funkce je pro každého jedince vypočítat, kolik dílků je potřeba projít, aby byla nalezena cesta mezi aktuální pozicí jedince a cílovou pozicí. Funkce bere v potaz strukturu bludiště a počítá pouze dílky, které na sebe navazují. Vzdálenost tak není vyjádřena vzdušnou čarou skrz zdi bludiště. Pro tuto fitness funkci je využit algoritmus Prohledávání do šířky (BFS), pomocí kterého je hledána cesta mezi dvěma body. Algoritmus z pozice jedince prochází do šířky bludiště a hledá cílovou pozici. Princip algoritmu je takový, že postupně prochází vrstvu po vrstvě, přičemž začíná v počáteční pozici. Postupně z tohoto místa prochází své sousedy, poté sousedy sousedů a tak dále, dokud nenavštíví všechny dosažitelné pozice nebo nenajde cestu k cílové pozici.

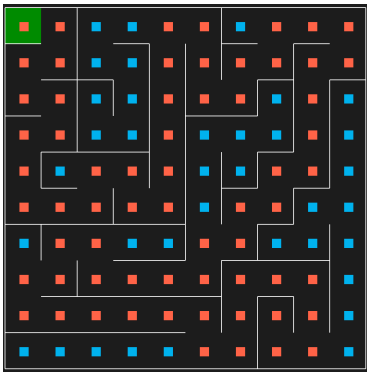
Prakticky to znamená, že se nejprve procházejí všechny dosažitelné sousední pozice, které se nacházejí kolem počáteční pozice. Dosažitelné ve smyslu, že mezi pozicemi neexistuje zeď. Poté se přejde do vrstvy sousedů, ze které se procházejí sousedi sousedů. Takto se tento postup opakuje, dokud se nenajde cesta nebo dokud nebyly navštíveny všechny dosažitelné pozice.

Výsledkem fitness funkce je potom číselná hodnota vyjadřující, jak daleko se jedinec nachází od cílové pozice. Čím je tato hodnota menší tím blíže se jedinec nachází k cíli. Pokud je hodnota rovna 0, tak se jedinec dostal na cílové místo a nalezl cestu.

Jako příklad využijeme jedince, na kterém byl dříve demonstrován pohyb bludištěm. Jedinec se dostal do slepého místa a zastavil. Z tohoto místa je hledána cesta bludištěm k cílové pozici.

Nejprve jsou z aktuální pozice vyhledány dosažitelné sousední pozice. Podle obrázku 5.20 existuje pouze jediný možný soused. Algoritmus se tak posune na tuto pozici, zaznamená navštívenou pozici a zase hledá sousedy pozice, na kterou se v předchozím kroku posunul. Tento postup se opakuje dokud není nalezen cíl. Výsledek hledání je prezentován na obrázku 5.21, kde je červeně vyznačena nalezená vzdálenost mezi cílovou pozicí a aktuální pozicí jedince ve slepé uličce. Modře vyznačená místa představují ostatní pozice, které byly navštíveny za běhu hledání cíle. Výsledná hodnota fitness je 63 a toto číslo odpovídá počtu dílků mezi těmito body.

Na základě ohodnocení všech jedinců je dále realizována selekce.



Obrázek 5.21: Příklad prohledávání bludiště do šířky.

5.2.6 Selekce

V návaznosti na vyhodnocení fitness hodnoty probíhá operace selekce, kdy jsou vybíráni lepší jedinci, podle kterých bude prováděno křížení se zbytkem populace.

Během selekce je nejprve celá populace seřazena podle fitness hodnoty, kdy na začátku se nacházejí jedinci, kteří mají nejmenší hodnotu fitness. Jedinci s nejmenší hodnotou jsou ti, kteří si nejlépe vedli při průchodu bludištěm a jsou nejlepšími kandidáty ke křížení a vytvoření nových potomků, kteří dědí vlastnosti svých rodičů.

Ze seřazené populace je vybráno určité procento lepších jedinců, na základě kterých se provede křížení s ostatními horšími jedinci. Kolik bude vybráno jedinců jako lepších je určeno parametrem mez selekce, který byl uveden a popsán v předchozích částech. Pokud byl tedy parametr nastaven na hodnotu 0.1, tak bude z populace vybráno prvních 10% nejlepších jedinců. Následně jsou postupně vybíráni vždy lepší a horší jedinec ke křížení, dokud není počet vytvořených potomků roven velikosti populace.

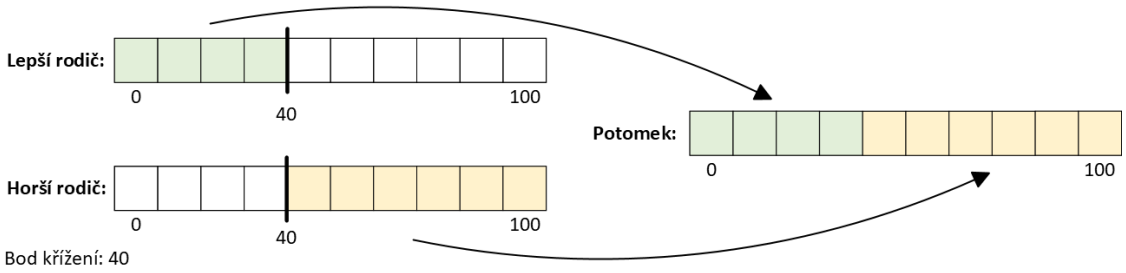
5.2.7 Křížení

Křížení je technika, která provádí reprodukci neboli vytvoření nových jedinců, kteří jsou potomky svých rodičů. Kombinuje genetické informace svých rodičů (části chromozomů) a snaží se vytvořit

lepšího jedince.

Součástí algoritmu je křížení naimplementováno tak, že jeho výsledkem je vždy jeden nový potomek, který dědí část chromozomu z obou svých rodičů. Rozdělení chromozomů je založeno na náhodnosti, kdy je předem vygenerováno číslo v intervalu od 0 po délku chromozomu. Podle vygenerovaného čísla, které reprezentuje pozici jsou následně chromozomy rodičů rozděleny na dvě části a z každého z rodičů vybrána jedna část k vytvoření nového chromozomu potomka. Výběr jednotlivých částí je založen na pravděpodobnosti, kdy je ve většině případech chromozom nového potomka složen ve tvaru, že první část je složena z lepšího rodiče a druhá z toho horšího. Tímto je zaručeno, že se do nové populace dostanou i horší, různorodější jedinci.

Příklad křížení je znázorněn na obrázku 5.22, kde vstupem jsou dva rodiče, ze kterých vzniká nový potomek. Podle náhodně vygenerovaného čísla v intervalu 1 až 100 je určeno, ve kterém místě budou seznamy kroků (chromozomy) rozděleny. V tomto případě jsou chromozomy rozděleny na 40. pozici a z prvního rodiče je vybráno prvních 40 kroků, zatímco z druhého horšího rodiče jsou vybrány kroky od 40. pozice do konce.



Obrázek 5.22: Příklad křížení chromozomů rodičů a vznik jejich potomka skládajícího z první části lepšího rodiče a z druhé části horšího rodiče.

5.2.8 Mutace

Mutace je poslední operací před vytvořením nové populace (generace) jedinců. Úkolem mutace je zavést do nové populace náhodnost nebo rozmanitost změnou některých hodnot uvnitř chromozomu. Mutací jsou tak zavedeny do populace nové varianty potomků, kteří by nemohli vzniknout ze selekce, křížení a mají potenciál být správným řešením problému.

V algoritmu je mutace prováděna následovně. Podle hodnoty míry mutace je určeno s jakou pravděpodobností (jak často) bude prováděna mutace a kolik genů (kroků) v celé populaci bude celkem mutováno. Následně na to jsou náhodně vybíráni jedinci z populace, kterým bude pozměněna některá část jejich chromozomu (seznamu kroků). Změna probíhá tak, že je pro vybraného jedince náhodně určeno, kterou část chromozomu pozměnit. Poté je už jen náhodně vygenerován nový krok, který je umístěn na vybrané místo s tím, že se nesmí jednat o stejný krok jako před mutací.

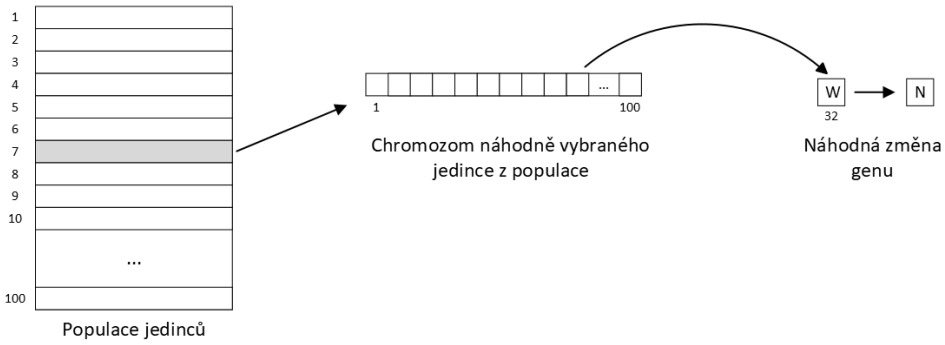
Předpokládejme tedy, že míra mutace odpovídá hodnotě 0.15, velikost populace 100 jedincům, každý jedinec má chromozom o délce 100 kroků a máme bludiště o velikosti 10x10 dílků. Na základě těchto informací je určeno, kteří jedinci v populaci budou mutováni, a které kroky v jejich chromozomech. Podle míry mutace (hodnota 0.15) a celkového počtu kroků (součin počtu jedinců a délky chromozomu) je určeno, kolik genů bude mutováno v celé populaci. Následně je z intervalu <0, 10000> představující celkový počet genů náhodně vybráno vždy unikátní číslo, ze kterého bude vyjádřen konkrétní jedinec a pozice genu v jeho chromozomu. Pořadové číslo jedince (7) je určeno podílem vybraného čísla z intervalu (732) a počtem kroků (100). Zbytkem při dělení je stanovena pozice genu (32), který bude nahrazen náhodně vybraným krokem. Z populace bude vybrán jedinec s pořadovým číslem 7 a chromozom bude mutován na pozici 32. Popisovaný proces je ilustrován na obrázku 5.23.

Počet genů k mutaci: $0.15 \times (100 \times 100) = 1500$ genů

Náhodně vygenerované číslo v intervalu <0, 10 000>: 732

Určení jedince: $\frac{732}{100} = 7$

Určení pozice genu v chromozomu: $732 \mod 100 = 32$



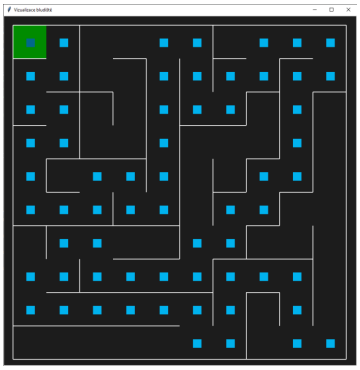
Obrázek 5.23: Příklad mutace chromozomu vybraného jedince z populace.

Po provedení mutace je vytvořena nová generace jedinců, která znovu bude procházet bludištěm a hledat cestu skrz. Jednotlivé operace jsou prováděny opakovaně, dokud není nalezeno řešení nebo nedošlo k překročení maximálního možného počtu vytvořených generací.

5.2.9 Výstup

Výstupem implementovaného algoritmu je nalezená cesta mezi startovní a cílovou pozicí v bludišti. Výsledky jsou uživateli graficky vizualizovány pomocí okna na obrazovce, ve kterém se vykreslí vytvořené bludiště a následně na to vykreslí výsledná cesta bludištěm. Na obrázku 5.24 je ilustro-

váno vykreslené okno s bludištěm a nalezenou cestou. Cesta je vyznačena pomocí malých modrých čtverečků v bludišti.



Obrázek 5.24: Nalezená cesta v bludišti 10x10.

Součástí výstupu je také výpis do konzole, kde je možné pozorovat činnost algoritmu před nalezením řešení a vizualizací na obrazovku. Do konzole jsou vypisovány informace o vytvořených generacích a jejich statistikách. Po nalezení řešení bludiště je na závěr do konzole vypsána hláška o tom, že bylo nalezeno řešení pro dané bludiště a jaké procento jedinců z populace nalezlo cestu bludištěm. Na obrázku 5.25 je ilustrován konkrétní příklad výstupu do konzole. Algoritmus pro každou generaci vyhodnocuje statistiky jako je nejlepší jedinec v populaci a průměrná hodnota fitness. Z obrázku si lze všimnout, že bylo vytvořeno celkově 7 generací, kdy už při 3. generaci někteří jedinci nalezli cestu bludištěm, ale algoritmus se nezastavil. Důvodem proč algoritmus pokračoval dále ve své činnosti je ten, že jen malá část populace nalezla řešení. V mém případě je nastaveno zastavení algoritmu tak, že minimálně 80% jedinců z celé populace musí nalézt řešení. Dále je součástí výstupu průměrná hodnota každé generace, která indikuje, jak se daná generace zlepšila, případně zhoršila ve svém vývoji.

```
Nejlepsi fitness hodnota  1 generace je 28 a prumerna fitness hodnota populace je 62.400
## 2 generace vytvorena ##
Nejlepsi fitness hodnota  2 generace je 13 a prumerna fitness hodnota populace je 33.050
## 3 generace vytvorena ##
Nejlepsi fitness hodnota  3 generace je 0 a prumerna fitness hodnota populace je 21.510
## 4 generace vytvorena ##
Nejlepsi fitness hodnota  4 generace je 0 a prumerna fitness hodnota populace je 33.940
## 5 generace vytvorena ##
Nejlepsi fitness hodnota  5 generace je 0 a prumerna fitness hodnota populace je 12.150
## 6 generace vytvorena ##
Nejlepsi fitness hodnota  6 generace je 0 a prumerna fitness hodnota populace je 4.690
## 7 generace vytvorena ##
Jedinci nalezli cestu bludistem
90.00% jedincu z populace nalezlo reseni
```

Obrázek 5.25: Textový výstup do konzole obsahující informace o vytvořených generacích a nalezeném řešení.

K nalezení cesty v bludišti existuje celá řada dalších algoritmů, které nejsou založeny na GA. Jedná se o algoritmy, které byly primárně navrženy k prohledávání grafů a hledání nejkratší cesty v grafu. Zde bych uvedl například Dijkstrův algoritmus, který byl navržen k nalezení nejkratší cesty mezi dvěma uzly v kladně ohodnoceném grafu.

5.3 Porovnání přístupů

V této části jsou srovnány z hlediska výhod a nevýhod oba přístupy implementovaných algoritmů.

Nejprve se zaměříme na symbolickou reprezentace znalostí, která bylo využita v případě implementovaného Winstona algoritmu. Hlavní výhodou učení pomocí symbolické reprezentace znalostí je její interpretovatelnost na rozdíl od jiných přístupů jako například ANN nebo GA, kde o datech nic nevíme. V případě symbolické reprezentace jsou data reprezentována pomocí symbolů a logických pravidel, jak bylo popsáno v 4.1, která jsou relativně jednoduché k pochopení. Jako příklad lze uvést již dříve popisovaný oblouk zapsaný pomocí jednoduchý klauzulí v jazyce Prolog, který je jednoduše čitelný a pochopitelný.

Podle [3] patří mezi nevýhody symbolického přístupu citlivost na šum v datech nebo špatně vybraná data k učení. Pokud je hledána hypotéza na trénovací sadě dat, která obsahuje špatně definována nebo neúplná data, tak může nastat případ, kdy výsledná hypotéza nebude korektně rozpoznávat objekty. Například bude hypotéza příliš specifická nebo v druhém případě příliš obecná kvůli toho, že v datech chyběla nebo byla navíc pravidla, která by omezila množinu možných případů daného konceptu. Proto je nezbytné, aby byla poskytovaná data kompletní a správně navržena.

Učení na základě symbolické reprezentace znalostí v některých případech vyžaduje existenci poznatků, které tvoří potřebnou teorii ke generalizaci a specializaci (například jak je možné generalizovat některé tvary v oblouku na obecnější pomocí ontologie) [7]. Na rozdíl od toho GA nepotřebují takové informace znát, protože pracují na základě operací odvozených z evoluce (selekce, křížení a mutace), kdy jsou jedinci vybíráni a vyhodnocováni na základě jejich schopnosti plnit stanovené cíle. Jsou vytvářeni jedinci, kteří se dokážou postupně adaptovat na daný problém.

Podle [4] hlavní výhodou GA je jejich síla při řešení optimalizačních úloh a schopnost prohledávat širokou množinu možných řešení. Použitím operací z evoluce, jako selekce, křížení a mutace, mohou GA efektivně hledat řešení komplexních problémů (například sady pravidel pro ovládání robotů nebo počítačové programy). GA totiž pracují s celou populací možných řešení a po dokončení běhu GA může populace obsahovat více optimálních řešení daného problému.

Dále je možné GA používat paralelně, což umožňuje současně ve stejnou chvíli prohledávat více potenciálních řešení problému. Paralelismus může vést k rychlejšímu nalezení optimálního řešení problému.

Mezi nevýhody GA patří jejich časová a výpočetní složitost při řešení komplexních problémů. Jejich náročnost roste v závislosti na řešeném problému, velikosti populace a dalších kritériích, kdy je potřeba pro celou populaci jedinců provádět mnoho operací během selekce, křížení a mutace. Další

nevýhodou bývá jejich horší interpretovatelnost ve srovnání například se symbolickou reprezentací znalostí. GA pracují s daty, které jsou hůře čitelná nebo nesrozumitelná pro člověka. Data můžou být například ve formě binárních řetězců, ze kterých nelze jednoznačného vyčíst, co představují, a jak byly vytvořeny. Další nevýhodou GA je to, že potřebují přesně definovanou fitness funkci. Navržení vhodné fitness funkce pro daný úkol může být často velmi složitý proces.

GA nejsou nejvhodnějším způsobem řešení všech problémů, ačkoliv mohou představovat jednu z možností pro jejich řešení. Například navážeme-li na řešení problému hledaní cesty bludištěm, tak pro tento problém existuje celá řada efektivnějších a rychlejších algoritmů, které byly konkrétně pro problém hledání cesty v bludišti (grafu) navrženy. Příkladem je například Dijkstrův algoritmus k hledání nejkratší cesty v grafu s nezáporně ohodnocenými hranami. Algoritmus začíná v počátečním uzlu a postupně prochází sousední uzly, přičemž aktualizuje délky cest ke všem dosud navštíveným uzlům.

Kapitola 6

Závěr

Cílem této práce bylo nejprve shrnout teorii strojového učení v multiagentních systémech a na základě symbolické reprezentace znalostí, GA, pravděpodobnosti a ANN. Všechny tyto vyjmenované pojmy byly popsány v teoretické části mojí práce.

V případové studii byly podle zadání naimplementovány a následně popsány algoritmy založené na symbolické reprezentaci znalostí a GA. Oba vytvořené algoritmy se povedlo dovést do funkčního spustitelného stavu, na kterých lze demonstrovat charakteristiky GA a symbolické reprezentace znalostí.

Prvním z nich byl algoritmus založený na symbolické reprezentaci znalostí, přesněji Winstonův algoritmus, který vytváří obecnou hypotézu k rozpoznávání tvarů (konceptu oblouku). Druhým z nich byl algoritmus postavený na principech GA, který hledá cestu ve čtvercovém bludišti. Během testování GA pro hledání cesty v bludišti bylo zjištěno, že s rostoucí náročností bludiště a větším množstvím slepých uliček dochází k tomu, že se jedinci přestávají dostatečně vyvíjet. Tento problém byl nakonec vyřešen volbou lepší fitness funkce, která počítá cestu (počet dílků) mezi aktuální pozicí jedince a cílovou pozicí k nalezení cesty bludištěm.

Oba algoritmy je dále možné rozšířit. Jedním z možných příkladů rozšíření pro první algoritmus je jeho zobecnění, aby ho bylo možné použít pro rozpoznávání širší množiny problémů. Druhý algoritmus by bylo možné rozšířit pro řešení úloh zabývajících se optimalizací a hledání nejkratší nebo nejefektivnější cesty v podobné struktuře jako je bludiště.

Literatura

[1] LUGER, George F. *Artificial intelligence: structures and strategies for complex problem solving*. 6th ed. Boston: Pearson Education, 2008. ISBN 978-0-321-54589-3.

[2] KUBÍK, Aleš. *Inteligentní agenty*. Brno: Computer Press, 2004. ISBN 80-251-0323-4.

[3] POOLE, David L. a Alan K. MACKWORTH. *Artificial intelligence: foundations of computational agents*. 2nd pub. Cambridge: Cambridge University Press, 2010. ISBN 978-0-521-51900-7.

[4] MITCHELL, Tom M. *Machine learning*. Boston: McGraw-Hill, 1997. ISBN 00-704-2807-7.

[5] MENŠÍK, M., M. DUŽÍ, A. ALBERT, V. PATSCHKA a M. PAJR. Machine learning Using TIL. *Frontiers in Artificial Intelligence and Applications*. 2020, (321), 344-362.

[6] RUSSELL, Stuart J. a Peter NORVIG. *Artificial intelligence: a modern approach*. 3rd ed. Upper Saddle River: Prentice Hall, 2010. Prentice Hall series in artificial intelligence. ISBN 978-0-13-604259-4.

[7] HONZÍK, Petr. *Strojové učení* [online]. Brno, 2006 [cit. 2022-12-02]. Dostupné z: `http://vision.uamt.feec.vutbr.cz/STU/others/Honzik%20-%20Strojove_uceni_S.pdf`

[8] BHASIN, Harsh a Surbhi BHATIA. *International Journal of Computer Science and Information Technologies: Application of Genetic Algorithms in Machine learning* [online]. 2. 2011 [cit. 2023-01-16]. ISSN 0975-9646.

[9] OLIVKA, Petr. *Genetické algoritmy* [online]. [cit. 2023-01-16]. Dostupné z: `https://poli.cs.vsb.cz/edu/isy/down/ga.pdf`

[10] HLAVÁČ, Václav. *Rozpoznávání s markovskými modely* [online]. České vysoké učení technické v Praze, institut informatiky, robotiky a kybernetiky. [cit. 2023-01-24]. Dostupné z: `http://people.ciirc.cvut.cz/~hlavac/TeachPresCz/31Rozp/62MarkovianPR-Cz.pdf`

[11] BYSTRÝ, Vojtěch. *Markovovy modely v Bioinformatice* [online]. Masarykova Univerzita, Brno. [cit. 2023-01-24]. Dostupné z: `https://is.muni.cz/el/fi/jaro2017/PB051/um/prednaskaHMM1.pdf`

[12] MATOUŠEK, Kamil. *Pravděpodobnostní reprezentace neurčitosti: Bayesovské sítě* [online]. Fakulta Elektrotechnická, České vysoké učení technické Praha. [cit. 2023-01-24]. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/a7b33sui/bayesovske_site.pdf

[13] BERKA, Petr. *Bayesovská klasifikace* [online]. In: . [cit. 2023-01-24]. Dostupné z: https://sorry.vse.cz/~berka/docs/izi456/kap_5.6.pdf

[14] VOLNÁ, Eva. *Neuronové sítě 1* [online]. In: . Ostravská univerzita, Ostrava, 2008 [cit. 2023-01-30]. Dostupné z: https://web.osu.cz/~Volna/Neuronove_site_skripta.pdf

[15] VONDRÁK, Ivo. *Neuronové sítě* [online]. Fakulta elektrotechniky a informatiky, VŠB - Technická univerzita Ostrava, 2009 [cit. 2023-01-30]. Dostupné z: http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf

[16] SIEGER, Tomáš. Neuronové sítě. In: *ČVUT FEL CourseWare Wiki* [online]. [cit. 2023-01-30]. Dostupné z: https://cw.fel.cvut.cz/b181/_media/courses/a6m33dvz/dvz2017-05-nnet.pdf

[17] AHSAN NAEEM, Muhammad. Pyamaze. In: *Github* [online]. [cit. 2023-04-06]. Dostupné z: https://github.com/MAN1986/pyamaze

[18] BRATKO, Ivan. *Prolog: programming for artifical intelligence*. 3rd ed. Harlow: Pearson, 2001. International computer science series. ISBN 02-014-0375-7.

[19] WOOLDRIDGE, Michael J. *An introduction to multiagent systems* [online]. 2nd ed. Chichester: Wiley, c2009 [cit. 2023-06-10]. ISBN 978-0-470-51946-2.

[20] NETRVALOVÁ, Arnoštka. *Úvod do problematiky multiagentních systémů* [online]. In: . Plzeň, [cit. 2023-06-10]. Dostupné z: https://www.kiv.zcu.cz/~netrvalo/phd/MAS.pdf

[21] KEITA, Z. An Introduction to Hierarchical Clustering in Python. In *Datacamp*. [online]. 2023 [cit. 2023-07-01]. Dostupné z: https://www.datacamp.com/tutorial/introduction-hierarchical-clustering-python

Příloha A

Zdrojový kód pohybu jedince

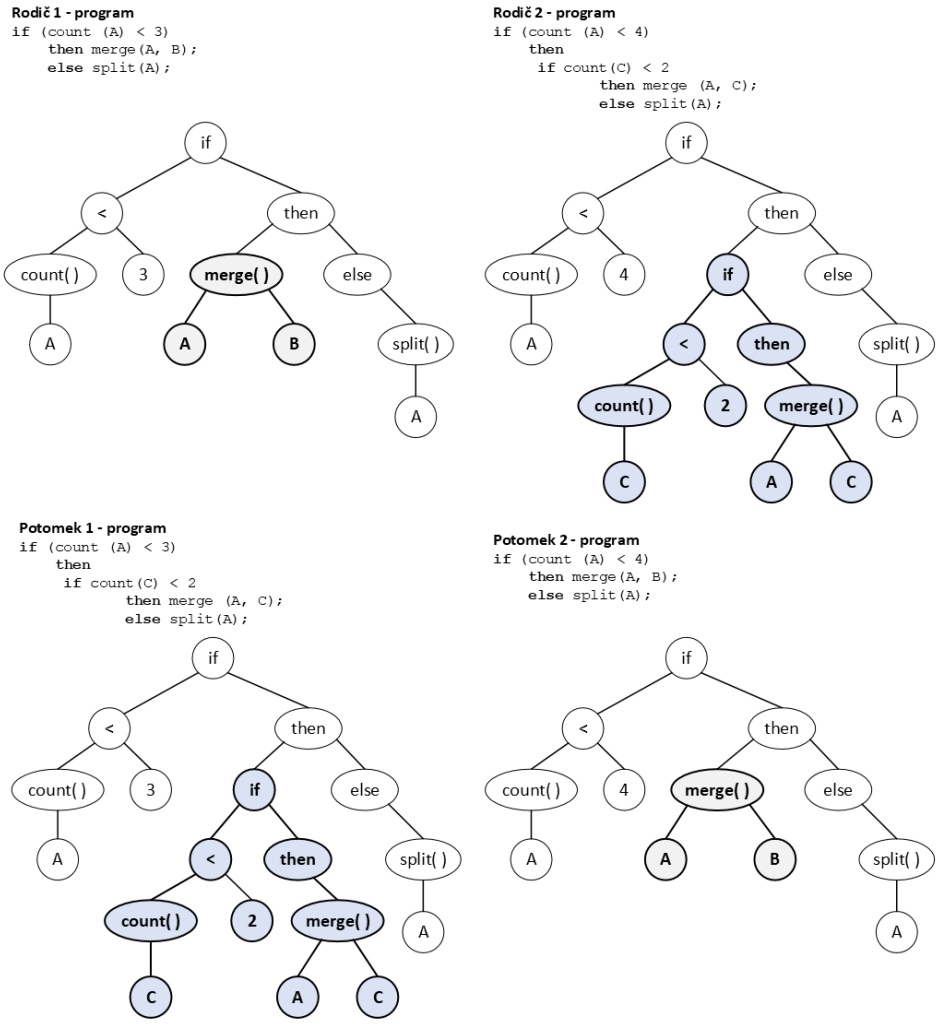
```
def next_move(self, move, maze_map, turn):
    if self.can_walk is False:
        return
    next_position = self.next_position(move)
    # pokud je možné se daným směrem pohybovat a nová pozice ještě nebyla navští
        vena
    if maze_map[(self.x, self.y)][move] == 1 and next_position not in self.
        visited_positions:
        self.move(move)
    # není možné daným směrem jít nebo na daném místě už byl
    else:
        # vyhledá dostupné směry pohybu
        possible_moves=[k for k, v in maze_map[(self.x, self.y)].items() if v==1]
        if len(possible_moves) == 1:
            self.can_walk = False
            self.fitness += 300
        else:
            # zjistí se možné cesty, které nebyly navštíveny
            correct_moves = self.correct_moves(possible_moves)
            # vyloučí se cesta, která už byla použita
            difference = set(correct_moves) - set(move)
            difference = [i for i in difference]
            # pokud existuje více možných cest
            if len(difference) > 1:
                # náhodně se vybere jedna cesta a ta se použije k pohybu
                tmp = random.choice(difference)
```

```
            # uloží se na pozici aktuálního tahu, nový pohyb
            self.move_array[turn] = tmp
            self.move(tmp)
        # pokud neexistuje žádná možná cesta, tak se nemůže dále pohybovat
        elif len(difference) == 0:
            self.can_walk = False
        else:
            # existuje pouze jedna možná cesta k pohybu
            self.move_array[turn] = difference[0]
            self.move(difference[0])
```

Zdrojový kód A.1: Ověření a pohyb jedince.

Příloha B

Příklad křížení v genetickém programování



Obrázek B.1: Příklad křížení v genetickém programování (převzato z [2]).