Principy řešení úloh. Dekompozice úloh na podúlohy, vedlejší efekty. Prohledávání stavového prostoru. Algoritmus A^*. Heuristické znalosti. Rozhodovací stromy a stromy řešení úloh, jejich prohledávání a optimalizace hledaného řešení.

Základní motivace - (počáteční model prostředí → požadovaný koncový model prostředí) ⇒ hledat vhodnou posloupnost akcí, jejichž aplikací lze přejít od počátečního modelu k cílovému ⇒ taková posloupnost se nazývá *plánem* a metody vytváření plánů označujeme jako metody řešení úloh. Každému modelu odpovídá jistý *stav* prostředí, množina všech stavů tvoří *stavový prostor* ⇒ lze reprezentovat orientovaným grafem ⇒ řešení úloh formulujeme jako hledání přijatelné cesty v orientovaném grafu.

Preciznější formulace problému řešení úloh
Stavový prostor χ je dán konečnou množinou stavů $\delta = \{s_i\}$ a konečnou množinou operátorů $\phi = \{\varphi_j\}$ ⇒

$$\chi = (\delta, \phi).$$

Každý operátor je parciálním zobrazením δ do δ. Úloha U nad stavovým prostorem χ je dvojice:

$$U = (s_é, \varepsilon),$$

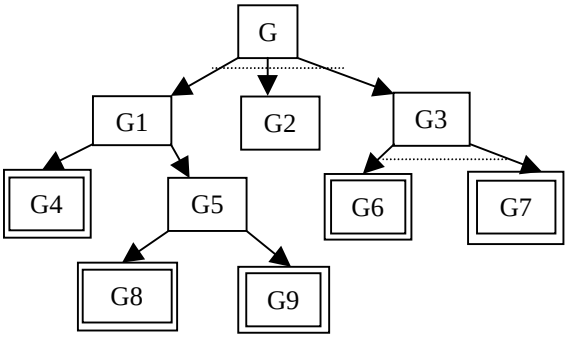
kde prvý symbol označuje počáteční stav a druhý množinu stavů cílových, $s_0 \in \delta$, $\varepsilon \subseteq \delta$.
Plánem P pro danou úlohu U (řešením úlohy U) je taková posloupnost operátorů:

$$P = (\varphi_1, \varphi_2, ... \varphi_n),$$

ke které lze přiřadit posloupnost stavů $(s_0, s_1, ...s_n)$, pro něž platí:

$$s_1 = \varphi_1 (s_0), ..., s_n = \varphi_n (s_{n-1}).$$

Rozklad úlohy na podúlohy. Úloha rozsáhlejšího charakteru ⇒ dekompozice na podúlohy, pokud podúloha není elementární, lze uskutečnit pokus opětovné dekompozice. Rozklad úlohy lze zobrazit orientovaným *AND/OR grafem* (obr. č.1).



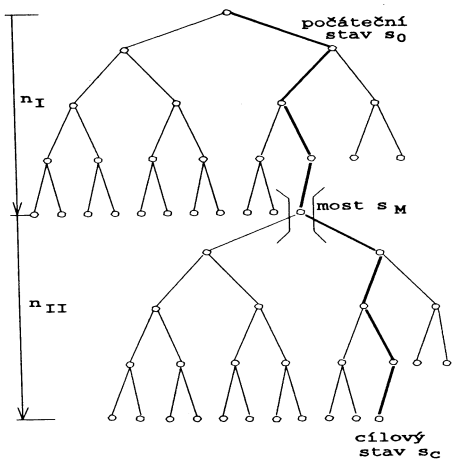
Obrázek č.1-Příklad rozkladu úlohy na pdoúlohy-znázornění ve formě AND/OR grafu (hrany vycházející z AND uzlu jsou spojeny čárkovanou linií).

AND – představují konjunkci podúloh (k řešení úlohy reprezentované uzlem AND je nezbytné, aby každá podúloha byla řešitelná).
OR – představují disjunkci poúloh (k řešení úlohy reprezentované uzlem OR stačí, aby alespoň jedna z podúloh byla řešitelná.

Užitečnost rozkladu. Vyjděme z obr. č.2.

Předpokládejme, že řešení musí procházet mezilehlým stavem (tzv. mostem) je možné původní úlohu rozdělit na dvě části: I s počátečním stavem s_0 a cílovým s_M , II s počátečním stavem s_M a cílovým s_c. Uvažujme nadále slepé prohledávání stavového prostoru, pak je vygenerováno max. počet stavů:

$$2^{nI} + 2^{nII},$$



Obrázek č.2 - Vliv rozkladu na rozsah prohledávaného stavového prostoru.

kde horní indexy reprezentují hloubku stromu. Evidentně platí:

$$2^{nI} + 2^{nII} < 2^{nI + nII},$$

přičemž úspora prohledávání může býti značná.

<u>*Vedlejší efekty*</u> – nechť je každý stav popsán *R* parametry $s_i = (s_{i,1}, ... s_{i,R})$, potom lze úlohu *U* formulovat:

$$U: s_0 = (s_{0,1}, ... s_{0,R}) \rightarrow s_c = (s_{c,1}, ... s_{c,R}).$$

Pokus by si všechny stavy odpovídaly, je úloha vyřešena. U netriviálních úloh se však vyskytuje *diference* některých či všech složek. Úloha je vyřešena, nalezneme-li takové operátory, které diference odstraňují. Možnost rozkladu úlohy na *R* podúloh, z nichž první zabezpečuje odstranění diference u první složky atd. $\Rightarrow$ není použitelné vlivem *vedlejších (postranních) efektů* jednotlivých operátorů. Kromě diference, kterou operátor odstraní, přidá či zvětší jinou diferenci, navíc není známo jaké operátory budou použity $\Rightarrow$ není možné apriorně přesně stanovit cílové stavy podúloh (mosty), a tudíž ani počáteční stavy dalších navazujících podúloh.

<u>*Prohledávání stavového prostoru.*</u> Algoritmus, poskytující návod pro výběr pravidel z konfliktní množiny pravidel v každém kroku prohledávání stavového prostoru. Tento mechanismus generuje při prohledávání stavového prostoru strom, který je podgrafem orientovaného grafu reprezentující stavový prostor V případě přímého řízení se nejprve generuje a expanduje počáteční uzel s_0, v dalším procesu prohledávání se pak expandují některé z dříve expandovaných uzlů. Je-li vygenerován některý uzel z množiny cílů, pak prohledávání končí. Každá strategie, má-li býti úspěšnou musí splňovat:

1. Musí vést k *prohledávání* (způsobovat pohyb a zabraňovat cyklům v posloupnosti pravidel,
2. Musí být systematická.

Prohledávání lze omezit využitím znalostí o řešeném problému, o nichž víme, že jsou často užitečné při řešení, přičemž mnohdy ani nezaručují nalezení řešení. Tyto znalosti nazýváme *znalostmi heuristickými*. Ze dvou řešitelů stejného problémů, ten, který je vybaven lepším souborem heuristik , prohledává menší část stavového prostoru. Dle toho, zda jsou využity znalosti odané úloze či nikoliv, rozlišujeme *algoritmy informované a neinformované.*

<u>*Neinformované metody prohledávání*</u>. Definujme seznam neexpandovaných stavů *OPEN* a seznam již expandovaných stavů *CLOSED*.

<u>*Algoritmus prohledávání do šířky.*</u>

1. Zapiš počáteční stav do seznamu OPEN, seznam CLOSED je prázdný, Je-li počáteční stav stavem cílovým, konči prohledávání s pozitivním výsledkem.
2. Pokud je seznam prázdný, pak ukonči algoritmus s negativním výsledkem.

3. Vymaž první stav (označme jej *i*) v seznamu OPEN a zapiš tento stav do seznamu CLOSED.
4. Expanduj stav *i*. Pokud tento stav nemá již následovníky nebo všichni následovníci byli již expandováni (tj. jsou v seznamu CLOSED), pokračuj krokem č.2.
5. Zapiš všechny následovníky stavu *i*, kteří nejsou v seznamu CLOSED, na konec seznamu OPEN.
6. Pokud některý z následovníků stavu *i* je cílovým stavem, ukonči prohledávání s pozitivním výsledkem, jinak pokračuj krokem č.2.

Algoritmus prohledávání do hloubky je často spojeno s omezením maximální prohledávané hloubky, při jejímž dosažení se používá mechanismus *navracení (backtracking*). Změny oproti algoritmu prohledávání do šířky:

1. Slovní spojení na konec seznamu zaměň následujícím: na začátek seznamu.
2. Za krok č.3 vlož následující krok: Pokud se hloubka uzlu *i* rovná maximální přípustné hloubce, pokračuj krokem č.2.

Informované metody prohledávání. Definujme *hodnotící funkci f*, která pro každý uzel stromu řešení určí jeho ohodnocení, které je poté využito k výběru pro expanzi. Čím lépe tato funkce vystihuje charakter úlohy ⇒ budou vždy vybírány nejperspektivnější uzly ⇒ čím kvalitnější heuristické znalosti ⇒ efektivnější prohledávání.

Algoritmus uspořádaného prohledávání – seznamy OPEN, CLOSED, uzel je tvořen: <jméno uzlu,hodnota *f*, jméno rodičovského uzlu>. Nyní následuje podrobný popis,v němž expandujeme stavy dle minimální hodnoty funkce *f*:

1. Počáteční stav zapiš do seznamu OPEN, seznam CLOSED je prázdný.
2. Pokud je seznam OPEN prázdný, řešení neexistuje, ukonči prohledávání.
3. Ze seznamu OPEN vyber stav *i* s nejmenší hodnotou *f(i)*. V případě většího počtu stavů se stejnou minimální hodnotou *f(i)* prověř, zda některý z těchto stavů není stavem cílovým, v takovém případě jej vyber; jinak vyber mezi nimi libovolně.
4. Vymaž stav *i* ze seznamu OPEN a zařaď jej do seznamu CLOSED.
5. Je-li stav *i* stavem cílovým, pak ukonči prohledávání s pozitivním výsledkem.
6. Expanduj stav *i* ; pro každého následovníka *j* stavu *i* vypočti hodnotu *f(j)*. Pokud stav není v seznamu OPEN ani CLOSED, zařaď jej do seznamu OPEN. Nachází-li se již v seznamu OPEN nebo CLOSED, avšak s větším ohodnocením, pak změň ohodnocení na *f(j)* a jméno rodičovského uzlu a zařaď ho do seznamu OPEN.
7. Pokračuj krokem č.2.

Algoritmus A. Využívá následujícího tvaru hodnotící funkce :

$$f(i) = g(i) + h(i),$$

kde *g(i)* je součet ceny optimální cesty z počátečního stavu do stavu *i* a *h(i)*cena optimální cesty ze stavu *i* do některého z cílů. Poté lze funkci *f(i)* chápat jako cenu přechodu z počátečního stavu do cílového stavu přes stav *i*. Většinou tyto hodnoty nahrazujeme jejich odhady, protože skutečnou cenu nelze zjistit: $g^*(i)$, $h^*(i)$. Funkce $h^*(i)$ je tedy *nositelem heuristické informace* a bývá proto nazývána *heuristickou informací.*

Algoritmus A^*. Přípustnost algoritmu prohledávání – algoritmus je přípustný, jestliže vždy nalézá optimální cestu, pokud tato cesta existuje.Lze dokázat, *iff* $h^*(i)$ je nezáporná a menší nebo rovna skutečné ceně přechodu ze stavu *i* do cílového stavu, je algoritmus *A* přípustný ⇒ *algoritmus* A^*. Čím je $h^*(i)$ lepším spodním odhadem funkce *h(i)*, tím menší část stavového prostoru se prohledává při hledání optimálního řešení. Monotónnost funkce $h^*(i)$: $h^*(i) - h^*(j) \leq cena\ (i,j)$.

Stromy řešení úloh – viz. Prohledávání stavového prostoru. Prohledávání stavového prostoru spolu s expanzí generuje strom, který je podgrafem orientovaného grafu reprezentující stavový prostor. Lze nalézt analogii s *Rozhodovacími stromy*. V každém stavu stavového prostoru provádíme rozhodnutí o volbě stavu pomocí nějž provedeme expanzi. V případě neinformovaného prohledávání je toto rozhodování velice primitivní a závislé pouze na postupu generování jednotlivých stavů, avšak v případě informovaného prohledávání je provedeno rozhodnutí na základě hodnotící funkce, která vyjadřuje *naději*, že daný stav leží na cestě z počátečního stvavu do cílového stavu.

Prohledávání stromu řešení úloh a jejich optimalizace. Základní algoritmy prohledávání stavového prostoru:

1. *Algoritmus prohledávání do šířky* –viz. výše.

2. *Algoritmus prohledávání do hloubky* – viz. výše.
3. *Algoritmus iterativního prohlubování DFID* – neinformované, v každé iteraci roste povolená hloubka prohledávání o číslo 1.
4. *Algoritmus gradientní* – expanduje ten uzel, který je na dané hloubce st. Prostoru ohodnocen nejlépe.Rodič a sourozenci jsou zapomenuti. Nebezpečí *sklouznutí* do lokálního extrému a nekonečné smyčky (neuchovává rodiče a alternativy).
5. *Algoritmus uspořádaného prohledávání* – viz. výše.
6. *Algoritmus A* – viz. výše.
7. *Algoritmus* A^* - viz. výše.
8. *Algoritmus se stejnoměrnou cenou* – viz. A^*, ale $h^*(i) = 0$.
9. *Algoritmus větví a mezí* – doprohledávání stavového prostoru. Je-li nalezeno řešení, pak je zapamatována jeho cena. Poté jsou automaticky ze seznamu OPEN vyřazeny ty stavy, jejichž ohodnocení je vyšší. Je-li nalezeno nové řešení s menší cenou, pak je opět zapamatováno. Algoritmus pracuje dokud OPEN není prázdný.
10. *Algoritmus IDA**. Kombinace A^* a DFID, prohledávání do hloubky, přičemž vylučuje z uvažování ty uzly, jejichž cena přesahuje určitou prahovou hodnotu. Hodnota se mění, zpočátku je rovna odhadu ceny počátečního uzlu. V každé iteraci se práh zvyšuje o minimální hodnotu, o kterou byl v předchozím kroku překročen.

Hledání v hierarchicky uspořádaném prostoru. Využití abstrakce. Každý stav představuje několik stavů nižší hierarchické úrovně, takový prostor je mnohem menší.

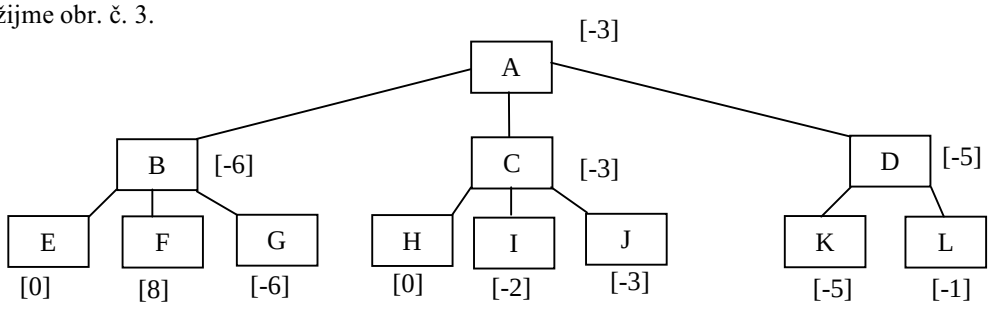
Metaznalosti. Využívání znalostí o úloze ve formě samostatných pravidel., kterými se řídí používání pravidel definujících úlohu.

Hry. – odlišný *způsob* rozhodování.

Metoda minimaxu. – prohledávání do hloubky s omezenou hloubkou. Volba statické ohodnocovací funkce *f*, která se vypočítává pro daný uzel na *i*-té úrovni grafu využitím tohoto iterativního postupu:

1. Daný uzel se expanduje a pro všechny následovníky se určí hodnota *f*.
2. Z takto určených hodnot se vybere nejlepší a použije se zpětně pro určení rodičovského uzlu.

Použijme obr. č. 3.



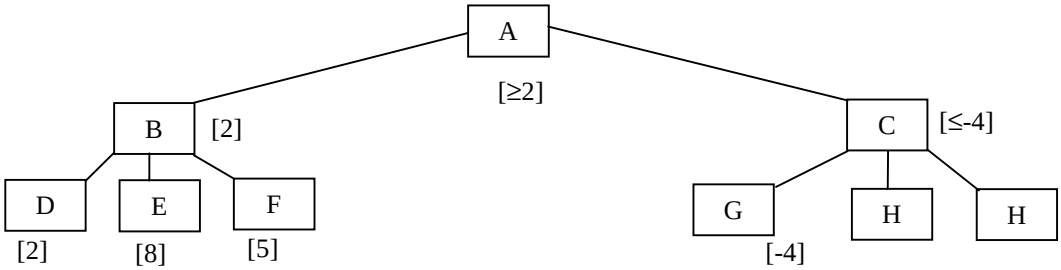
Obrázek č.3

Uvažujme lichou úroveň (rozhoduje se počítač), je nutné brát maximální hodnotu nejbližší sudé úrovně. Nyní uvažujme sudou úroveň (rozhoduje se protihráč), je nutné brát minimální hodnotu nejbližší liché úrovně. Lichou úroveň nazýváme maximalizující, sudou minimalizující.

Prořezávání alfa-beta. Optimalizace prohledávání do hloubky. V velmi ranných stádiích jsou zavrhována řešení evidentně horší než dosud nalezená. Strategie větví a mezí (viz. výše) modifikovaná pro hru dvou hráčů – *prořezávání alfa-beta.* Definujme dvě meze:

1. Hodnotu *alfa* reprezentující dolní mez ohodnocení uzlu, v němž je na tahu hráč A (maximalizační úroveň).
2. Hodnotu *beta* představující horní mez ohodnocení uzlu, odpovídajícího tahu hráče B (minimalizační úroveň).

Uvažujeme obr. č.4.



Obrázek č.4

Ve stavu B jsme zpětně získali ohodnocení rovné 2. Poté jsme ohodnotili stav G hodnotou rovnou –4. V tuto chvíli nemusíme rozvíjet následující stavy, protože: V C provádíme minimalizaci, tudíž maximální hodnota (beta) bude rovna –4. V uzlu A je hodnota (alfa) rovna 2, a protože se jedná a maximalizující úroveň, bude jeho hodnota rovna minimálně dvěma, proto hodnota v uzlu C nemá na toto ohodnocení vliv.