Praktické plánování

V této části bude zmínka o existujících plánovačích, které jsou používány ve složitých, reálných doménách.

Montáž, integrace a verifikace kosmické lodi

European Space Agency používá plánovač nazvaný `Optimum-AIV` na pomoc montáži, integraci a verifikaci kosmických lodí (**AIV** = Assembly, Integration, and Verification). Systém se používá pro generování plánů a monitorování jejich provádění. Během monitorování systém upozorňuje uživatele na připravované činnosti a je schopen navrhovat úpravy plánu, když je nějaká činnost provedena pozdě, zrušena, nebo se objeví něco neočekávaného. Je to zejména schopnost systému změnit plán, co jej činí principiálně význačným. Systém sám neprovádí plány; ty jsou prováděny lidmi pomocí standardních konstrukčních a testovacích zařízení.

V podobně složitých projektech se obecně používají plánovací nástroje pocházející z oblasti operačního plánování, např. diagramy `PERT` nebo metoda kritické cesty. Tyto nástroje v zásadě používají jako vstup *ručně navržený* kompletní plán rozdělený na sub-plány v pořadí rozděleném na části a generují pro to optimální rozvrh činností. Akce jsou považovány za objekty, které vyžadují nějaký čas a mají omezení vzhledem k pořadí činností; jejich efekty se neuvažují. Tím se lze vyhnout použití znalostního inženýrství a pro jednorázové problémy to může obyčejně být zcela přiměřené řešení. Avšak pro většinu praktických aplikací existuje mnoho souvisejících problémů k řešení, takže se obvykle vyplácí popis domény a pak automatické generování plánů.

To je důležité zejména během *provádění* plánů—pokud krok plánu selže, je obyčejně nutno plán rychle změnit (upravit), aby se projekt znovu uvedl v život. Diagramy metody `PERT` neobsahují příčinná propojení a další informaci potřebnou pro to, aby bylo zjevné, jak změnit plán, a lidmi prováděné změny jsou často příliš pomalé.

Úspěch systémů umělé inteligence z reálného světa vyžaduje integraci do prostředí, v němž jsou aktivní. Plánovač musí být schopen přístupu do existujících databází s informacemi o projektu, nezávisle na formě v níž mohou informace být. Zároveň representace vstupů a výstupů plánovače by měly být ve formě, která je jak expresívní, tak snadno srozumitelná uživateli. Dříve (velmi stručně) zmíněný jazyk `Strips` je pro doménu AIV nedostačující, protože nedokáže vyjádřit čtyři základní koncepty:

1. **Hierarchické plánování:** Je zjevné, že vypuštění kosmické lodi je mnohem složitější než nakupování potravin v obchodě. Jednou ze schůdných cest zvládnutí složitosti je specifikace plánů na různých úrovních detailů.

   Plán na *vrcholné úrovni* může zahrnovat:
   příprava raketového stupně,
   příprava kosmické kabiny,
   naložení kosmické lodi potřebnými věcmi,
   vypuštění.

   K tomu může být velké množství meziúrovní před tím, než se dojde k úrovni zcela konkrétních prováděcích akcí:
   vlož matici *A* do otvoru *B* a sešroubuj se šroubem *C*.

   Přidání schopnosti representovat hierarchické plány může tvořit rozdíl mezi uskutečnitelnou a neuskutečnitelnou akcí a může způsobit snadnější porozumění výslednému plánu. Také pomáhá dovolit uživateli, aby poskytl plánovači vodítko ve formě částečně specifikovaného, abstraktního plánu, pro nějž plánovač doplní detaily.

2. **Komplexní podmínky:** Operátory `Strips` jsou v principu propositční—umožňují proměnné, ale jejich použití je velmi limitované. Neexistuje universální kvantifikace, bez níž není možno vyjádřit fakt, že operátor *Launch* (*Vypuštění*) způsobí přemístění *všech* objektů lodi na oběžnou dráhu. Podobně tomu, operátory `Strips` jsou nepodmíněné: nelze vyjádřit fakt, že pokud všechny systémy jsou vypuštěny, *Launch* umístí loď na oběžnou dráhu, jinak loď skončí v oceánu.

3. **Čas:** Jazyk `Strips` je založen na situačním kalkulu, proto předpokládá, že všechny akce se vyskytnou bezprostředně a že každá akce následuje po nějaké jiné bez přerušení. Problémy reálného světa vyžadují lepší model času—musí representovat fakt, že projekty mají konečné termíny (např. loď musí být vypuštěna 17. června), akce nějaký čas trvají (testování montáže *XYZ* trvá 6 hodin) a jednotlivé kroky plánu vyžadují časová okna (např. stroj pro testování montáže *XYZ* je k dispozici od 1. května do 1. června vyjma víkendů, ale musí se reservovat týden dopředu). Tradiční techniky operačního plánování měly jako hlavní přínos uspokojování časových omezení pro kompletní plán rozdělený na části.

4. **Zdroje:** Projekt má normálně rozpočet, který nelze překročit, takže plán musí být omezený na spotřebu částky, která je k dispozici. Podobně existují limity na počet pracovníků k dispozici a na počet montážních a testovacích stanic. Omezení zdrojů se může týkat věcí potřebných v určitý čas (např. lidé) nebo celkového použitelného množství (např. peníze). Popis akcí proto vyžaduje zahrnutí spotřeby a vytváření zdrojů, stejně jako plánovací algoritmy musí být schopny zpracovat účinně omezení zdrojů.

`Optimum-AIV` je založen na plánovací architektuře z r. 1991, zvané `O-Plan`, která umožňuje používat jazyka se značnou možností výrazů pro representaci času, zdrojů a hierarchických plánů. Rovněž přijímá heuristiky pro vedení vyhledávání a zaznamenává důvody každého výběru, což usnadňuje pozdější úpravy plánu, pokud je zapotřebí. `O-Plan` byl použit na mnoho problémů, např. opatřování software v Price Waterhouse, plánování procesu montáže zadní osy u aut zn. Jaguar, a kompletní plánování produkce továrny Hitachi.

Plánování činnosti pracoviště

Problémy řešené různými pracovišti spočívají v obstarání surovin a dílů a jejich smontováním do konečných výrobků. Problém lze rozdělit na plánovací úlohu (rozhodnutí, které montážní kroky budou uskutečněny) a rozvrhovou úlohu (rozhodnutí, kdy a kde bude každý krok uskutečněn). V mnoha výrobních zařízeních se plánuje manuálně a rozvrh se dělá pomocí automatizovaného nástroje.

V Hitachi se O-Plan používá v plánovacím a rozvrhovém systému zvaném Tosca. Typický problém zahrnuje výrobu 350 různých výrobků, 35 montážních strojů a přes 2000 různých operací. Plánovač poskytuje rozvrhy na 30 dní pro osmihodinové směny. Obecně Tosca sleduje plán rozdělený na části, s nejmenším závazkem. Umožňuje také "nízkozávazková" rozhodování: volby, které kladou omezení na plán nebo na nějaký jeho konkrétní krok. Např. systém může vybrat rozvrh akce pro určitou třídu strojů bez konkretizace stroje.

Továrny s menší diversitou výroby často sledují pevný plán, ale stále potřebují automatizované rozvrhování. Systém ISIS z r. 1984 byl navržen speciálně pro rozvrhování a poprvé testován na výrobě částí turbíny firmy Westinghouse. Továrna produkuje tísíce různých lopatek turbín a pro každý typ existuje jeden či více plánů zvaných směrování procesu. Při obdržení objednávky je vybrán jeden z plánů a je pro něj vytvořen časový rozvrh. Čas závisí na kritičnosti objednávky: zda jde o urgentní výměnu lopatky činného zařízení, plánovanou údržbovou součástku se spoustou času, avšak s nutností přesně splnit datum, nebo prostě objednávka zásob k vytvoření reservy.

Tradiční metody plánování, jako PERT, jsou schopny najít vyhovující pořadí kroků vzhledem k časovým omezením, ale ukazuje se, že lidé-plánovači používající PERT spotřebují 80-90% svého času na komunikaci s pracovníky o tom, co jsou skutečná omezení. Úspěšný automatický plánovač potřebuje být schopen representovat a zdůvodnit tato přídavná omezení. Důležité faktory zahrnují cenu surovin, které jsou k dispozici, hodnotu vyrobeného, ale zatím neprodaného zboží, přesné předpovědi budoucích potřeb, a minimální narušení existujících postupů. ISIS používá hierarchické hledání s minimálním závazkem pro nalezení vysoce kvalitních plánů, které uspokojují všechny uvedené požadavky (minimální závazek—viz dříve uvedený popis representace plánů—v podstatě odklad ne právě v daném okamžiku aktuálních záležitostí do jiné fáze).

Plánování kosmických letů

Plánovací a rozvrhové systémy jsou v této oblasti využívány rozsáhle, a to jak pro plánování výprav lodí, tak i pro jejich konstrukci. Důvodem je především to, že kosmické lodě jsou mimořádně nákladné a také někdy obsahují lidi, což v případě havárie může mít extrémně vysokou cenu škody a přinést nenahraditelné ztráty. Kromě toho se lety uskutečňují v prostoru neobsahujícím mnoho dalších agentů, kteří by mohli narušit očekávané výsledky.

Plánovače byly a jsou používány pozemními týmy pro Hubblův teleskop a nejméně pro tři lodě: Voyager, UOSAT-II a ERS-1. V každém případě je cílem organizace pozorovacího zařízení, přenašečů signálů a kontrola mechanismů pro nastavení a udržování výšky a rychlosti, aby se maximalisovala informace získaná z pozorování za předpokladu dodržování omezení na čas a energii.

Plánování misí často zahrnuje velmi složitá časová omezení, zejména ty mise, které obsahují periodické události. Např. ERS-1 (satelit vypuštěný z European Earth Resource Observation) uskuteční jeden oběh Země za 100 minut a do téhož bodu se vrátí každých 72 hodin. Pozorování každého bodu na zemském povrchu může být provedeno v libovolném z mnoha časových okamžiků, navzájem oddělených 72 hodinami, ale v jiných časech nikoliv. Satelity mají omezení zdrojů výstupní energie—nemohou překročit pevně stanovené maximum a musí se zajistit, aby nevyčerpaly příliš mnoho energie za časovou periodu.

Satelity lze považovat za typ pracoviště s plánovaným rozvrhem činností, kde teleskopy a další zařízení jsou výrobní stroje a pozorování jsou výrobky. Hubblův teleskop je dobrým příkladem potřeby automatického plánování a rozvrhování. Po svém vypuštění v dubnu 1990 bylo objeveno, že primární zrcadlo má chybu v nastavení ohniska. Pomocí Bayesových technik pro rekonstrukci obrazu byl pozemský tým schopen chybu kompensovat do stupně, který umožnil předávání nových a důležitých informací o Plutu, o gravitačních čočkách, o supernově, atd. V r. 1993 astronauti z raketoplánu opravili většinu problémů primárního zrcadla, čímž otevřeli nové možnosti pozorování. Pozemní tým neustále objevuje více možností, které má či nemá HST k dispozici a bylo by zcela nemožné aktualisovat plány pozorování bez automatických nástrojů.

Kterýkoliv astronom může předložit žádost. Žádosti jsou děleny na vysokoprioritní (70% pozorovacího času), nízkoprioritní (podle časových možností) a zamítnuté. Návrhy přicházejí v počtu cca 1 za den, což znamená, že je jich více, než je možno uskutečnit. Každý návrh obsahuje specifikaci, který pozorovací nástroj by měl být namířen na určitý nebeský objekt a jaký druh snímku by se měl udělat. Některá pozorování lze udělat kdykoliv, jiná závisejí na faktorech jako postavení planet a zda HST je v zemském stínu. Existují unikátní omezení pro tuto doménu—např. astronom může žádat o periodická pozorování quasaru (velmi vzdálená raná vývojová fáze galaxie) po několik měsíců nebo let za stejných podmínek stínu Země.

Plánovací systém HST je rozdělen na dvě části. Dlouhodobý plánovač SPIKE prvně dělá rozvrh pozorování v segmentech po jednom týdnu. *Heuristikou* je přiřazování vysokoprioritních návrhů tak, aby mohly být všechny provedeny, dokud zbývá nejméně 20% kapacity. To se dělá asi rok předem. Mnohaletý rozvrh s přibližně 5000 pozorováními lze vytvořit za méně než hodinu, takže změna plánu je snadná. Po naplánování každého segmentu se použije krátkodobý plánovač SPSS, jenž dělá detailní plánování pro každý segment a naplňuje čas mezi vysokoprioritními úlohami mnoha nízkoprioritními podle možností. Systém rovněž vytváří příkazy pro nastavení polohy základny tak, aby se pozorování mohla provést. Kontroluje proveditelnost návrhů a detailních rozvrhů mnohem rychleji, než dokáží lidé-experti.

Budovy, letadla a pivovary

SIPE (System for Interactive Planning and Execution monitoring—systém pro interaktivní plánování a sledování výkonu činností) byl prvním plánovačem, který se zabýval problémem změny plánu, a také prvním, který použil některé důležité kroky vůči expresívním operátorům. Je podobný systému O-Plan v rozsahu vlastnosti a aplikovatelnosti. Byl použit v některých demonstračních projektech pro několik domén, včetně plánování operací na palubě letadla a plánování výroby pro jeden pivovar v Austrálii. Jiná studie použila SIPE pro plánování konstrukce výškových budov—jedna z nejsložitějších domén, do které se plánovač pustil.

SIPE umožňuje deduktivním pravidlům operovat nad stavy, takže uživatel nemusí specifikovat všechny relevantní literály jako výsledky pro každý operátor. Tento systém je aplikovatelný na široký rozsah domén, ale jeho obecnost je spojena s velkými náklady. Např. v doméně konstrukce budov se zjistilo, že SIPE potřebuje čas úměrný $O(n^{2.5})$ pro n-poschoďovou budovu. Z toho plyne vysoký stupeň interakce mezi poschodími, přičemž ve skutečnosti je ten stupeň mnohem nižší: při sledování linie stavby od země nahoru a zajištění toho, aby výtahové šachty byly propojeny je možné získat výkonnost mnohem bližší složitosti $O(n)$.

Uvedené příklady poukazují na současnou ideu plánovacích systémů. Jsou dost dobré pro modelování složitých domén, ale dosud nebyly zcela prakticky přijaty mimo omezený počet pilotních projektů. Je zapotřebí dále zvýšit stupeň flexibility a výkonnosti, aby se tyto systémy staly běžným nástrojem.

Hierarchická dekomposice

Problém s nákupem, popsaný dříve, lze vyřešit použitím vhodných metod tak, že je získáno řešení na poměrně vysoké úrovni abstrakce (pro agenta). Například plán typu:

[*Go(Supermarket), Buy(Milk), Buy(Bananas), Go(Home)*]

je dobrý popis toho, co dělat, na vysoké úrovni, ale je velmi daleko od toho, co je zapotřebí agentovi předat formou instrukcí do jeho efektorů. Pro agenta je to nedostatečné z hlediska toho, co by měl skutečně *udělat*.

Na druhé straně by nízko-úrovňový plán typu:

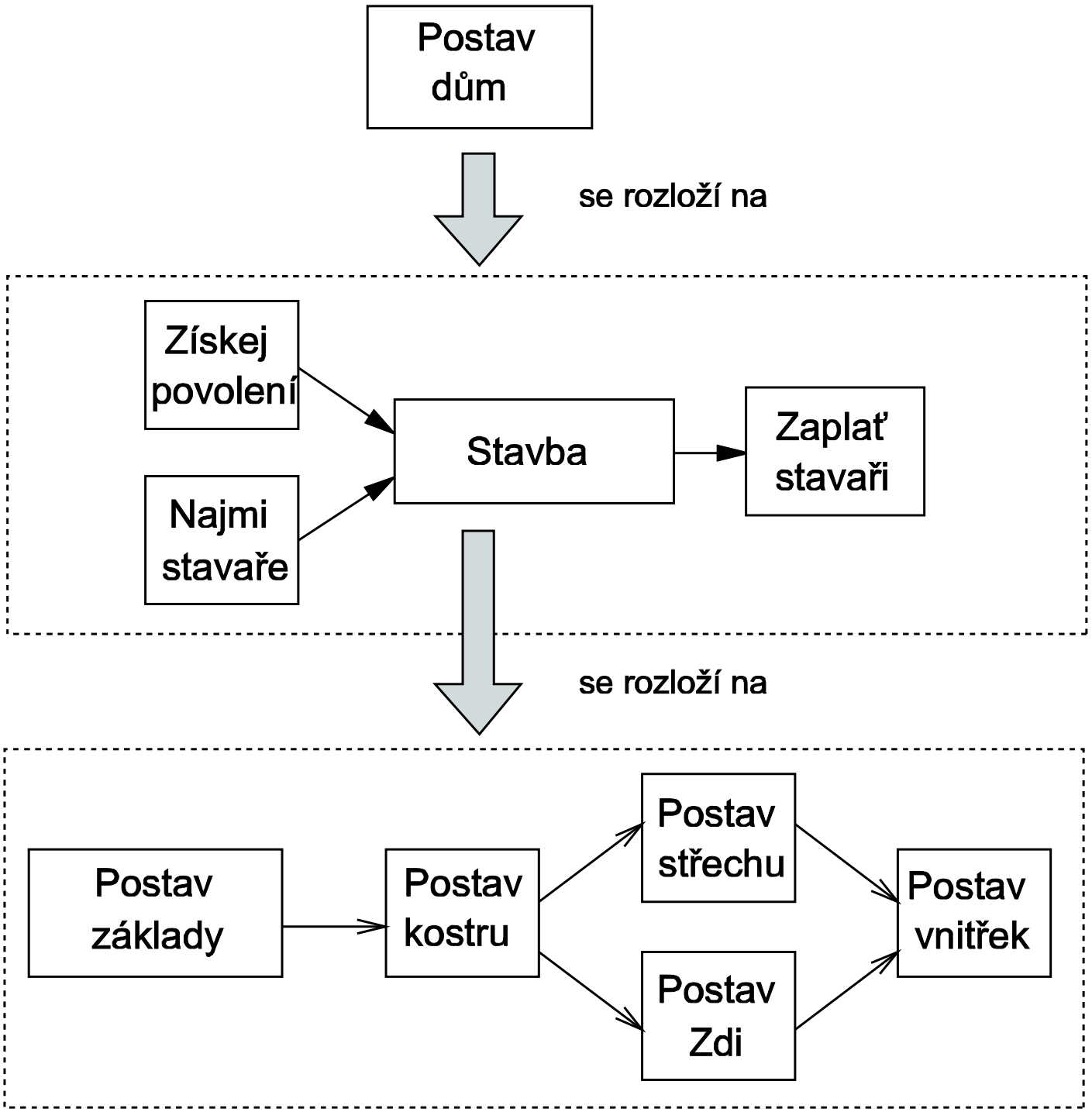
[*Forward* (1cm), *Turn* (1deg), *Forward* (1cm), ...]

obsahoval mnoho tisíc kroků a řešení problému nákupu by bylo nesmírně dlouhé. Zejména by vznikl problém extrémně velikého prostoru plánů velké délky a je pravděpodobné, že by metody hledání nenašly řešení v rozumném čase.

Uvedené dilema se v praktických plánovačích řeší pomocí myšlenky zavedení **hierarchické dekomposice**, která vychází z dekomposice **abstraktních operátorů** na skupinu kroků, které vytvářejí plán, který implementuje operátor.

Jako příklad lze uvážit problém stavby domu. Abstraktní operátor *Build(House)* (postav dům) lze rozložit na plán, který se skládá za čtyř kroků: *Obtain Permit* (získej povolení), *Hire Builder* (najmi stavitele), *Construction* (stavba), a *Pay Builder* (zaplať staviteli).

Kroky ukázaného vysokoúrovňového plánu lze dále rozložit na ještě specifičtější plány. Na obrázku je znázorněna dekomposice kroku stavby domu. V dekomposici lze pokračovat tak dlouho, až je dosažena nejnižší úroveň typu *Hammer(Nail)* [Zatlouci(Hřebík)]. Plán je kompletní v okamžiku, kdy je každý krok ve formě **primitivního operátoru**, tj. operátoru, který může být přímo proveden agentem. Hierarchická dekomposice je nejúčinnější, pokud lze operátory dekomponovat několika způsoby. Např. operátor *Build(Walls),* čili *Postav(Zdi)*, lze rozložit na plán zahrnující dřevo, cihly, beton nebo vinyl (zdi lze stavět z různého materiálu a plán by to měl respektovat).



K uskutečnění nastíněné myšlenky *dekomposice* je nutno udělat dvě věci:

- Do jazyka typu Strips dát rozšíření, která umožňují neprimitivní operátory.
- Modifikace plánovacích algoritmů tak, aby bylo možné nahradit neprimitivní operátory jejich dekomposicemi.

Rozšíření jazyka

Pro začlenění *hierarcické dekomposice* je zapotřebí při popisu každé problémové domény upravit dvě věci:

- Rozdělení souboru operátorů na primitivní a neprimitivní. V doméně stavby domů je *Zatlouci(Hřebík)* operátor primitivní a *Postavit(Dům)* neprimitivní. Obecně je rozdíl mezi primitivními a neprimitivními operátory relativní a záleží na agentovi, který bude plán vykonávat. Pro dodavatele stavby je operátor typu *Namontovat(PodlahováPrkna)* primitivní (protože pouze objedná dělníka k montáži podlahy), pro dělníka neprimitivní (a pro něj bude primitivní *Zatlouci(Hřebík)*).
- Přidání souboru dekomposičních metod. Každá metoda je výraz typu *Decompose(o,p)*, což znamená, že neprimitivní operátor, který je spojen s *o*, může být rozložen do plánu *p*. Dekomposice *Stavby* (*Construction*) z předchozího obrázku:

Decompose(*Construction,*
Plan(`Steps:`{S_1: *Build(Foundation)*, S_2: *Build(Frame),*
S_3: *Build(Roof)*, S_4: *Build(Walls)*,
S_5: *Build(Interior)*}
`Orderings:` $\{S_1 \prec S_2 \prec S_3 \prec S_5, S_2 \prec S_4 \prec S_5\}$,
`Bindings:` {},
`Links:` $\{S_1 \xrightarrow{Foundation} S_2, S_2 \xrightarrow{Frame} S_3,$
$S_2 \xrightarrow{Frame} S_4, S_3 \xrightarrow{Roof} S_5, S_4 \xrightarrow{Walls} S_5$
}))

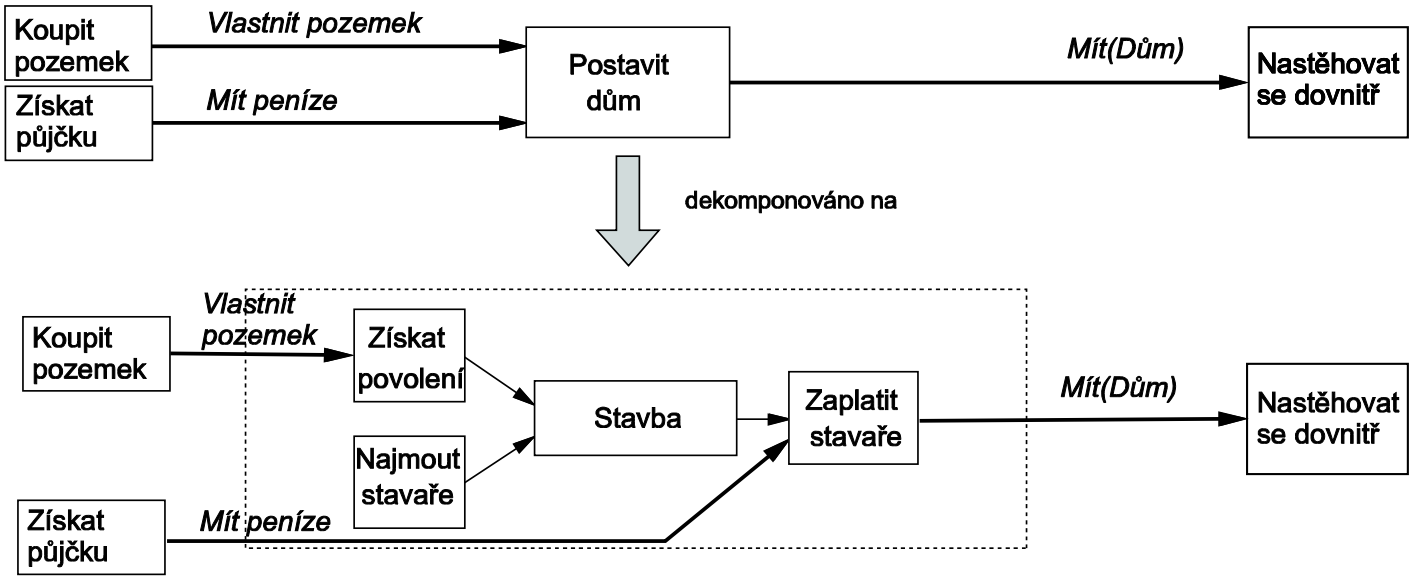
Dekomposiční metoda je v podstatě *procedura* nebo *makro* definující operátor. Musí být zajištěno, aby dekomposice byla korektní implementací operátoru. Platí, že plán *p* korektně implementuje operátor *o*, pokud se jedná o kompletní a konsistentní plán pro daný problém k dosažení účinků *o* za předpokladů pro *o*:

- *p* musí být konsistentní (tj. neexistuje žádná kontradikce v uspořádáních nebo vazbách proměnných).
- Každý účinek (efekt) operátoru *o* musí být prosazen nejméně jedním krokem plánu *p* (a není popřen nějakým jiným, pozdějším krokem plánu *p*).
- Každá podmínka kroků v *p* musí být dosažena nějakým krokem v *p* nebo být jednou z podmínek operátoru *o*.

Tím je zaručeno, že je možné nahradit neprimitivní operátor jeho složkami (dekomposicemi) a vše je správně navázáno na sebe. Je samozřejme vždy nutné zkontrolovat možná ohrožení pocházející z interakcí mezi nově zavedenými kroky s podmínkami a již existujícími kroky s podmínkami—není ale nutno si dělat starosti s interakcemi mezi kroky obsaženými v samotné dekomposici (ta *musí* být sama o sobě udělána zcela korektně).

Za předpokladu, že není příliš mnoho interakcí *mezi* různými částmi plánu, hierarchické plánování umožňuje vybudovat velmi složité plány kumulativně z jednodušších sub-plánů. Je také umožněno vygenerovat plány, uložit je, a pak znovu použít v pozdějších plánovacích problémech.

Následující obrázek ukazuje detailnější diagram příkladu dekomposice kroku plánu v kontextu rozsáhlejšího plánu:



Analýza hierarchické dekomposice

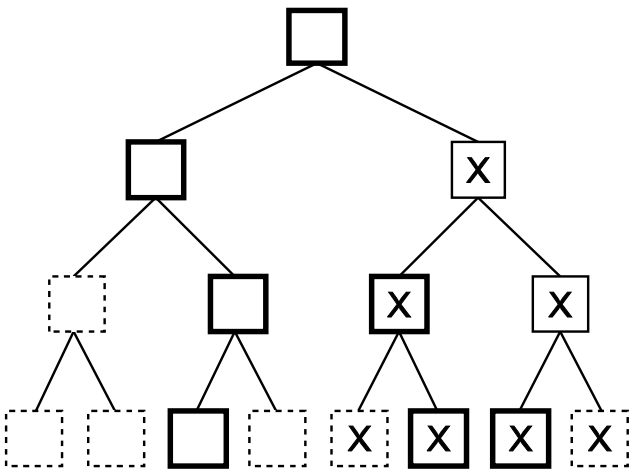
Hierarchiská dekomposice zde byla ukázána stručně, ve své základní části. Je založena na obdobné myšlence jako použití maker nebo procedur v programování—umožnit programátorovi nebo znalostnímu inženýrovi specifikaci problému po částech majících rozumnou velikost. Části mohou pak být hierarchicky kombinovány pro vytváření velkých plánů, aniž by vznikaly enormní kombinatorické náklady spojené s konstrukcí z primitivních operátorů.

Předpokládejme, že existuje způsob vzájemného propojení čtyř abstraktních operátorů k postavení domu (viz obrázek z podkapitoly *Hierarchická dekompozice*). Tento způsob se nazývá **abstraktní řešení**—konsistentní a kompletní plán obsahující abstraktní operátory. Nalezení malého abstraktního řešení (pokud existuje), by nemělo být příliš náročné. Pokračování tohoto procesu by mělo vést k dosažení primitivního plánu bez přílišného navracení se zpět po cestách hledání.

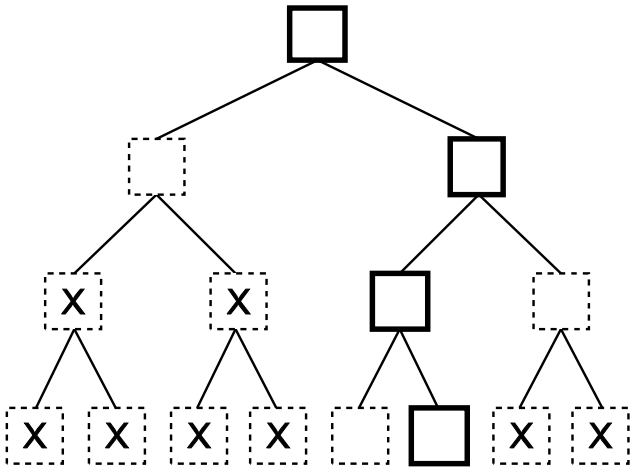
Zde se ovšem předpokládá, že hledání abstraktního řešení—a odmítání jiných abstraktních plánů jako nekonsistentních—je účelné. Měly by platit tyto vlastnosti:

- Je-li plán *p* abstraktním řešením, pak existuje primitivní řešení, pro něž je *p* abstrakcí. Pokud to skutečně platí, pak jakmile je nalezeno abstraktní řešení, lze odstranit z vyhledávacího stromu všechny ostatní abstraktní plány.Tato vlastnost se nazývá **řešení směrem dolů**.
- Je-li abstraktní plán nekonsistentní, pak neexistuje primitivní řešení, pro něž je ten plán abstrakcí. Pokud to skutečně platí, pak lze odříznout veškeré potomky libovolného nekonsistentního abstraktního plánu. Tato vlastnost se nazývá **řešení směrem nahoru**, protože to rovněž znamená, že všechny kompletní abstrakce primitivních řešení jsou abstraktními řešeními.

Tyto dvě vlastnosti lze znázornit graficky:



(a) Řešení směrem dolů

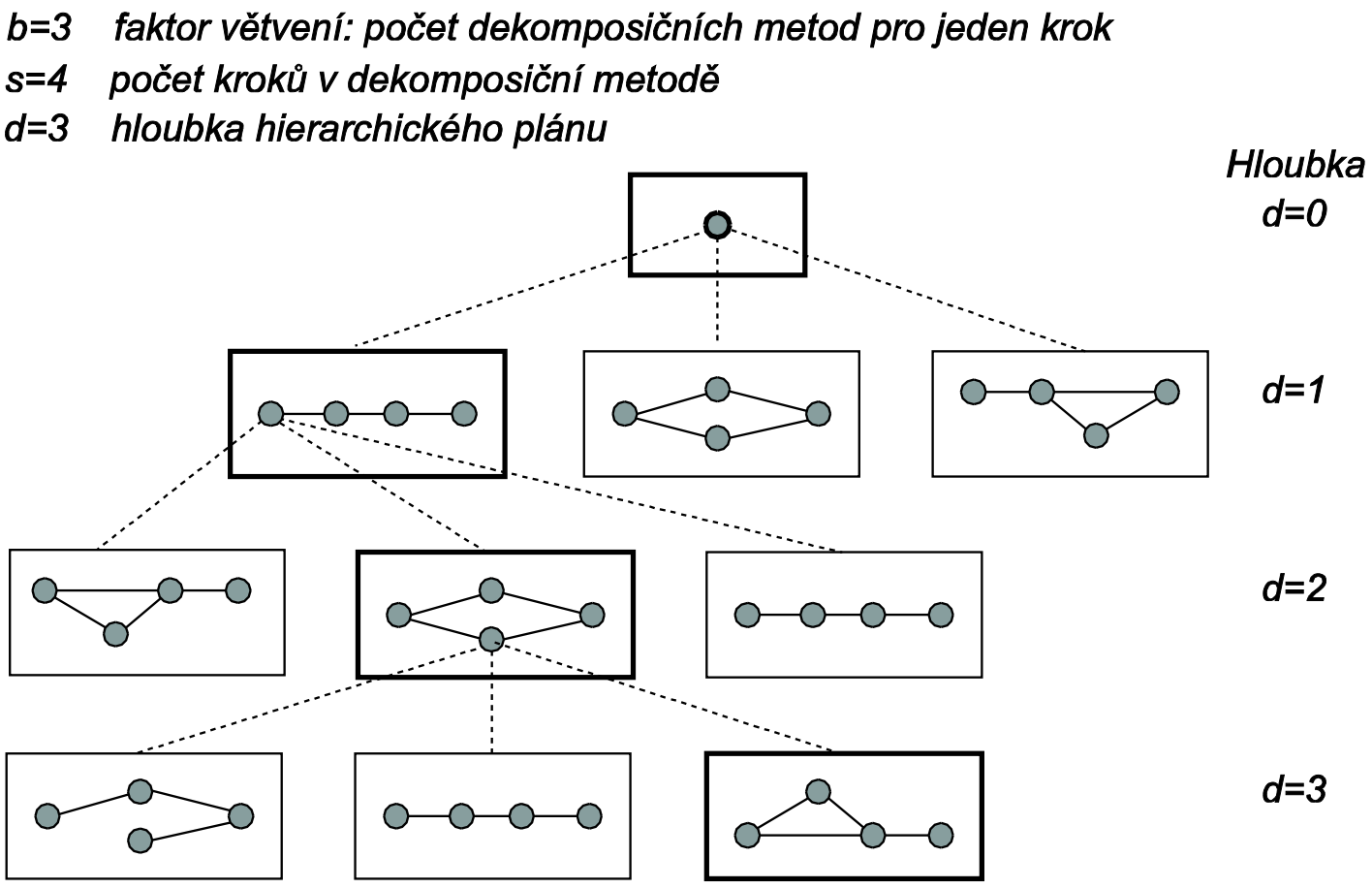


(b) Řešení směrem nahoru

Každý uzel representuje celý plán (*nikoliv* pouze jeden krok), každá hrana representuje dekomposiční krok, v němž je abstraktní operátor expandován.

V kořeni stromu je velmi abstraktní plán, v listech jsou plány s pouze primitivními kroky. Uzly s tučně znázorněnými obrysy jsou (abstraktními) řešeními. Uzly s tečkovanými obrysy jsou nekonsistentní (tj. nejsou bezesporné). Plány označené **X** nemusí být plánovacím algoritmem zkontrolovány. Mohou být (hledáním zleva doprava) odříznuty.

Předpokládejme pro příklad jednoduchý model prohledávaného prostoru. Nechť existuje nejméně 1 řešení s n primitivními kroky a dále, že možná ohrožení a omezení lze řešit v zanedbatelném čase—vše, co bude nutno uvažovat, je pouze čas potřebný k výběru správné množiny kroků. Obrázek ukazuje prostor hledání z hlediska parametrů b, s, a d:



Řešení bude mít $n = s^d$ kroků (zde je $n = 64$). Předpokladem je $1/b$ dekomposic vedoucích k řešení.

Nehierarchický plánovač by vygeneroval plán s *n*-kroky, přičemž by pro každý krok vybíral z *b* možností (předpokládejme zde, že počet dekomposic pro každý neprimitivní krok, *b*, je tentýž jako počet aplikovatelných nových operátorů pro podmínku primitivního kroku).

V nejhorším případě je tedy složitost $O(b^n)$. *Pro hierarchické plánovače lze použít strategii hledání pouze těch dekomposic, které vedou k abstraktním řešením* (v ukázaném zjednodušeném modelu na obrázku je právě 1 z každých *b* dekomposic řešením—v mnohem realističtějších modelech je zapotřebí uvážit, co dělat, když je počet řešení nulový nebo je jich více než 1).

Plánovač musí prohlédnout *sb* kroků v hloubce $d = 1$. V hloubce $d = 2$ je to dalších *sb* kroků pro každý dekomponovaný krok, ale zde musí dekomponovat jen $1/b$ z disponibilních kroků pro celkové množství bs^2. Celkový počet uvažovaných plánů je tedy

$$\sum_{i=1}^{d} bs^i = O(bs^d)$$

Rozdíl mezi nehierarchickým a hierarchickým plánovačem je v nutnosti prohlédnout 3×10^{30} plánů (nehierarchický) oproti 252 (hierarchický).

Řešení směrem vzhůru a dolů se jeví jako enormě výkonná—pokud ovšem není žádná ze tří vlastností pro *o* (uvedených v podkapitole o rozšíření jazyka) garantována, pak hierarchický plánovač nebude lepší než nehierarchický v nejhorším případě (v průměru může být lepší).

Jako shrnutí lze říci, že původní plánovací jazyk umělé inteligence vycházel z representace stavů a operátorů. Pro jednoduché či uměle sestavené problémy je takový přístup použitelný, avšak pro zpracování složitějších a realističtějších domén je zapotřebí plánovací jazyk rozšířit.

Existuje celá řada rozšíření, každé z nich vyžaduje odpovídající změny v plánovacím algoritmu a většinou dochází k obtížnému kompromisu mezi schopností vyjadřovat výrazy (expresivitou) a výkonností z hlediska nejhoršího případu. V dosud existujících experimentech se ukázalo, že expresívnější representace, použité uvážlivě a s rozmyslem, mohou vést k nárůstu efektivnosti, přestože se jedná o poměrně složitou úlohu.

Situační kalkul a dokazování teorémů jsou z praktického hlediska nepoužitelné, ale vedly ke vzniku srozumitelného paradigmatu (plánování typu `Strips`).

Jazyk `Strips` je příliš omezený pro komplexní, realistické domény, ale lze jej rozšířit několika směry:

- Plánovače, založené na rozšířených versích `Strips` a na algoritmech používajících částečné uspořádání a nejmenší závazek, se dosud ukázaly jako schopné zpracovat složité domény typu výprava kosmické lodi nebo výroba.

- **Hierarchická dekomposice** umožňuje zahrnutí neprimitivních operátorů do plánů (je ovšem nutno znát dekomposici na primitivnější kroky).

- Hierarchická dekomposice je nejúčinnější tehdy, pokud slouží ke zmenšení prohledávaného prostoru. Zmenšování (prořezávání stromu) je garantováno, pokud platí buď **řešení směrem dolů** (každé abstraktní řešení může být dekomponováno do primitivních řešení) nebo **řešení směrem nahoru** (nekonsistentní abstraktní plány nemají žádná primitivní řešení).

Oblast plánování patří v současnosti mezi oblasti umělé inteligence, kterým je věnováno význačné výzkumné úsilí vzhledem k potřebnosti rychlých a efektivních plánování v množství praktických problémů reálného světa.