Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE

Filip Stočes

Vizualizace plánů pro logistické úlohy

Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: RNDr. Pavel Surynek, Ph.D.

Studijní program: Informatika
Studijní obor: Obecná informatika
Praha 2011

Děkuji RNDr. Pavlu Surynkovi, Ph.D., vedoucímu této práce, za veškeré rady a čas, který mi věnoval.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne.....

Název práce: Vizualizace plánů pro logistické úlohy

Autor: Filip Stočes

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Pavel Surynek, Ph.D.

Abstrakt: Analýza plánů je důležitá při testování plánovacích systémů i při kontrole plánu před jeho provedením. Program VLP (Visualizace Logistických Plánů), který je výsledkem této práce, umožňuje uživateli orientaci v plánech pro logistické úlohy a pomáhá mu nalézt v nich případné nedostatky. S pomocí tohoto programu uživatel může vytvořit logistický problém a spustit plánovač, který vytvoří řešení (plán) pro tento problém. Program následně plán přehledně vizualizuje a simuluje jeho provedení, což uživateli usnadní analýzu tohoto plánu. Vše probíhá v grafickém uživatelském prostředí, takže se uživatel nemusí zabývat PDDL zápisem problému a plánu.

Klíčová slova: Plánování, Vizualizace, Logistické úlohy, PDDL

Title: Visualization of plans for logistics tasks

Author: Filip Stočes

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Pavel Surynek, Ph.D.

Abstract: Plan analysis is important for testing of planning systems and checking the plan before its execution. VLP application (result of this thesis) helps user with orientation in plans for logistical tasks and with finding possible flaws in plans. VLP offers tools for creating a logistical problem and running a planner, which creates a solution (plan) for this problem. The application then visualizes and simulates this plan, allowing the user to analyse it. VLP runs in a graphical user interface, so the user does not have to deal with writing problem and plan in PDDL.

Keywords: Planning, Visualisation, Logistics tasks, PDDL

Obsah

Úvod

Plánování je jedním z odvětví umělé inteligence. Zabývá se nalezením uspořádáné množiny akcí, které vedou k dosažení nějakého cíle. Plánovací systémy mohou být využívány k tvorbě plánů, které by jinak museli tvořit lidé (např. logistických plánů). Plánování je také důležitou součástí rozhodovacího procesu automatizovaných strojů. Pokud takový stroj není řízen soustavou instrukcí, které mu říkají jak reagovat na konkrétní situace, ale má jen schopnost pozorovat svět kolem sebe a zná cíl, kterého má dosáhnout, musí si vytvořit plán, který změní současný stav světa tak, aby byl cíl splněn.

Existuje mnoho nástrojů pro tvorbu plánu (plánovačů) a neustále jsou vyvíjeny další. S rostoucími schopnostmi plánovačů roste i velikost a komplexnost plánů, které sestavují. Je zřejmé, že pro člověka je orientace ve větších plánech velmi obtížná, až nemožná. Protože analýza plánů je nezbytná (testování plánovače při jeho tvorbě, kontrola plánu předtím, než je vykonán), jsou potřeba také nástroje, které ji usnadňují.

Většina existujících nástrojů pro vizualizaci a analýzu plánu není zaměřená na konkrétní typ úloh, ale soustředí se na plány obecně (např. itSIMPLE, GIPO). Proto tyto nástroje poskytují pouze vizualizaci na nízké úrovni, která je někdy nedostatečná pro odhalení nedostatků v plánu. Pokud se ale nástroj zaměří na vizualizaci pouze jednoho konkrétního typu úloh, může dosáhnout vizualizace na vyšší úrovni, která je mnohem přehlednější a přirozenější (např. program GraphRec [5], který se zabývá vizualizací pohybu entit po grafu)

Cílem této práce je vytvořit aplikaci, která umožňuje vizualizaci a analýzu logistických plánů (VLP). Konkrétně se zaměříme na plány řešící problém přepravy zboží od dodavatele k zákazníkovi, které se často vyskytují v praxi. Součástí práce je i návrh plánovací domény vhodné pro úlohy tohoto typu (viz příloha). Vedlejší funkcí aplikace je možnost editace zadání logistických úloh v grafickém prostředí. Uživatel se tak nemusí zabývat vytvářením popisu problému, který na vstupu požadují plánovače.

VLP tedy uživateli umožní vytvořit a uložit plánovací problém, a poté

graficky zobrazit plán, který jej řeší. Uživatel si může pustit simulaci plánu, při které probíhá i kontrola korektnosti plánu (aplikace uživatele upozorní, pokud je součástí plánu akce, kterou není v daném okamžiku možné provést). V průběhu simulace VLP zobrazuje informace o všech objektech, které jsou součástí problému. Uživatel tak může snadno sledovat přesun vozidel a zásilek mezi místy, což mu umožňuje v plánu odhalit zbytečné a neoptimální akce. To vše v přehledném grafickém prostředí. VLP může být zvláště užitečný pro vývojáře plánovacích systémů, kteří se snaží otestovat a vylepšit svůj plánovač.

Na začátku práce krátce popíšeme základy plánování, které je odvětvím umělé inteligence. Zejména se zaměříme na oblasti plánování, které jsou podstatné pro tuto práci. Dále představíme jazyk PDDL (Planning Domain Definitoin Language), který je používán pro zápis plánovacích domén a problémů. Ve druhé kapitole popíšeme plánovací doménu vytvořenou pro logistické úlohy, kterými se v této práci zabýváme. V další kapitole se zaměříme na to, jak je možné s pomocí VLP analyzovat logistické plány. Dále porovnáme VLP s jinými programy, které se zabývají editací problémů a analýzou plánů (itSIMPLE, VisPlan, GIPO). Předposlední kapitola popisuje, jak pracovat s programem VLP. Poslední kapitola se zabývá implementační stránkou programu.

1 Plánování

Plánování se zabývá nalezením uspořádané množiny akcí, které vedou k dosažení nějakého cíle. Přesněji, máme-li dán nějaký počáteční stav světa, množinu akcí, které můžeme provádět a popis cíle, kterého se snažíme dosáhnout, plánováním nazveme proces, který určí akce potřebné ke změně počátečního stavu tak, aby byly splněny cíle. Cílem může být:

- dosažení cílového stavu
- splnění nějaké podmínky
- optimalizace nějaké funkce

1.1 Plánovací model

Protože reálný svět je velmi komplexní, při plánovaní obvykle pracujeme s jeho zjednodušenným modelem. Ten obsahuje pouze objekty, které jsou pro náš problém důležité. Navíc zanedbává většinu jevů, které ovlivňují stav světa a soustředí se pouze na ty nejpodstatnější. (Řešíme-li například problém přepravy, nezahrnujeme většinou do modelu světa možnost, že vozidlo vezoucí zásilku při jízdě prorazí pneumatiku. Soustředíme se spíše na to, zda se balíky do vozidla vejdou nebo kolik paliva vozidlo spotřebuje).

V této práci se budeme zabývat pouze modelem zjednodušeného světa, který je deterministický, statický (pouze agent mění svět), plně pozorovatelný (máme všechny informace o světě) a konečný.

Model světa si reprezentujeme jako množinu stavů, ve kterých se svět může nacházet, a množinu akcí, pomocí kterých mezi těmito stavy přecházíme. Stav světa je vlastně popis situace, obsahuje tedy například informace o tom, kde se co nachází, kolik paliva zbývá v nádrži vozidla, a podobně.

1.2 Jazyk plánovacího problému[1]

Stavů v modelu světa je velké množství, stejně jako akcí, které jsou přechodovými funkcemi mezi nimi. Potřebujeme proto reprezentovat stavy a akce

1 Formální definice v této části jsou převzaty z [1]

tak, abychom je nemuseli všechny vyjmenovat. V této části se pokusíme vysvětlit, jak může taková reprezentace vypadat.

Mějme plánovací jazyk `L`, který je množinou predikátových symbolů a konstant. Atom budiž predikátový symbol s argumenty.

1.2.1 Stavy

Stav je určen množinou logických atomů (predikátových symbolů s argumenty z `L`), které v něm jsou pravdivé. Předpokládáme, že atomy, které nejsou ve stavu obsaženy, jsou v něm nepravdivé (svět je uzavřený). Proto se ve stavu nemusí nacházet žádné negace atomů.

Například takto by mohl vypadat stav světa, ve kterém převážíme balíky mezi místy:

```
vozidlo(v1) ∧ misto(m1) ∧ misto(m2) ∧
sousedi(m1,m2) ∧ balik(b1) ∧ na(v1,m1) ∧
na(b1,m1)
```

1.2.2 Operátory

Operátor je šablona, podle které můžeme vyrábět akce. Říká nám, jak můžeme přecházet z jednoho stavu do druhého. Operátor definuje podmínky, za kterých může být na stav použit (předpoklady) a změny, které se použitím operátoru provedou (efekty). Akce pak z operátoru vznikne, když dosadíme konkrétní hodnoty (konstanty z jazyka `L`) za proměnné v operátoru.

Formálně je operátor `o` trojice (`name(o)`, `precnod(o)`, `effects(o)`), kde:

- `name(o)` je jméno operátoru (sem patří i paramtery)
- `precond(o)` jsou literály, které musí být splněné, aby šlo operátor použít
- `effects(o)` jsou literály, které se stanou pravdivými po použití operátoru

Literály jsou predikátové symboly s proměnnými. Instanciací literálu tedy dostaneme atom. Jak předpoklady, tak efekty mohou být dvou druhů, pozitivní a negativní.

Negativní předpoklady jsou literály, které nesmí být ve stavu platné, pokud na něj chceme operátor použít. Negativní efekty jsou literály, které přestanou být pravdivými po jeho použití.

Příkladem operátoru pro svět z předchozího příkladu je:

```
Operátor(naloz(b,v,m)),
předpoklady: balik(b) ∧ vozidlo(v) ∧ misto(m) ∧ na(b,m) ∧ na(v,m)
efekty: not(na(b,m)) ∧ v(b,v)
```

1.2.3 Plánovací doména

Plánovací doména ∑ nad jazykem L a s operátory O je trojice (S,A,γ), kde:

- S je množina všech stavů světa
- A je množina všech akcí (instanciovaných operátorů z O nad L)
- γ je přechodová funkce mezi stavy, která popisuje, jak svět vypadá po použití akce a∈A na stav s∈S. (tedy γ(a,s) = (s - negativní efekty a) ∪ (pozitivní efekty a))

S je uzavřená vzhledem k funkci γ, takže pokud je a∈A použitelná na stav s∈S, pak γ(a,s)∈S.

1.2.4 Plánovací problém

Plánovací problém P je trojice (∑, s_0, g), kde:

- ∑(S,A,γ) je plánovací doména
- s_0∈S je počáteční stav
- g je dvojice množin atomů (g^+,g^-). Stav s splňuje g právě tehdy, když g^+ je obsažena v s a žádný atom z g^- není obsažen v s.

1.2.5 Plán

Plán π je posloupnost akcí (a_0, a_1, .., a_k). $\gamma^*(s_0, \pi) = \gamma(\gamma(...\gamma(s_0,a_0)...,a_{k-1}),a_k)$.

π je řešením problému P, právě když $\gamma^*(s_0, \pi)$ splňuje g.

1.2.6 Stavové proměnné

V této práci se budeme využívat plánování se stavovými proměnnými. Stavová proměnná popisuje vlastnost nějakého objektu v závislosti na stavu formou funkce. Takovou stavovou proměnnou může být například umístění balíku v závislosti na čase.

```
na: balk x cas -> misto
```

1.3 Plánování s časem

Plánovací model, tak jak jsme si ho představili, pracuje s implicitním časem. Akce v něm nemají žádné trvání, jsou prováděny okamžitě. To je ale pro logistické problémy přílišné zjednodušení. Přepravit zásilku z jednoho místa na druhé zjevně trvá nezanedbatelnou dobu, která navíc není vždy stejná. Proto musíme zavést u akcí další parametr, trvání.

Jakmile tak učiníme, narazíme na další problém. Kdy v průběhu akce máme vyžadovat splnění předpokladů? Kdy budou aplikovány efekty? Zjistíme, že na to neexistuje univerzální odpověď. Splnění některých podmínek nám stačí na začátku akce, jiné musí platit po celou dobu jejího trvání. Stejně tak je to s efekty. Předpoklady i efekty si tedy rozdělíme na tři skupiny podle toho, kdy jsou vyžadovány (kdy se projeví):

1. na začátku akce
2. po celou dobu průběhu akce
3. na konci akce

Zkusme si představit akci, která přesouvá vozidlo z jednoho místa na druhé. Doba **trvání** takové akce bude závislá na vzdálenosti těchto dvou míst. Jejím předpokladem bude, že **na začátku akce** se vozidlo nachází na prvním místě. Dále budeme požadovat, aby mezi místy existovala cesta **po celou dobu trvání akce**. Akce způsobí, že **na jejím začátku** vozidlo opustí první místo a **na jejím konci** dorazí na místo druhé.

1.4 Planning Domain Definition Language

Abychom mohli nějak zapsat plánovací problém, potřebujeme jazyk, který bychom k tomu mohli použít. Dlouhou dobu žádný takový standardní jazyk neexistoval a každý plánovač si tento problém řešil po svém. Až v roce 1998 byl pro po potřeby soutěže IPC (International Planning Competition) vytvořen jazyk zvaný PDDL (Planning Domain Definitoin Language).

Jazyk se rychle ujal a postupně se stal standardem pro popis plánovacích úloh. To mělo pozitivní dopad na rozvoj plánování jako celku. Od doby svého vzniku byl PDDL několikrát rozšířen, aby mohl reprezentovat složitější modely plánovacího světa. K zápisu problému budeme v této práci používat právě PDDL, proto si nyní tento jazyk trochu přiblížíme.

Plánovací úloha je v PDDL rozdělena do dvou souborů. Soubor s doménou obsahuje predikáty, funkce a akce, které lze v problému použít. V souboru problému je výčet objektů vyskytujících se v modelu světa, počáteční stavy funkcí a predikátů a popis cílů. Příklad PDDL domény a problému je možné nalézt v příloze.

VLP používá k zápisu logistických úloh právě PDDL, a proto si nyní představíme část jazyka, která je programem využívána.

V první verzi jazyka (PDDL 1.2 [7]) vytvořené pro první soutěž IPC v roce 1998 byly představeny jeho základy. Rozšíření jazyka pro plánování s časem přišlo v roce 2003 (PDDL 2.1 [2]). Dalšími rozšířeními se nebudeme zabývat, protože nejsou pro tuto práci podstatná.

Syntax PDDL je založena na syntaxi LISPu [10].

1.4.1 Doména

Proměnné v PDDL jsou řetězce začínající otazníkem (`?`).

Predikáty slouží k reprezentaci stavů světa. Z predikátů se také skládají předpoklady a efekty akcí. Příkladem predikátu, který reprezentuje fakt, že dvě místa spolu sousedí je:

```
(sousedi ?m1 ?m2 - misto)
```

Funkce v PDDL přiřazuje hodnotu nějaké vlastnosti objektů nebo vztahu

mezi nimi. Součástí předpokladů akce může být porovnání hodnot funkcí a mezi efekty akce může patřit změna hodnot funkcí. Následující funkce určuje velikost balíku:

```
(velikost ?b – balik)
```

Akce v PDDL odpovídají operátorům definovaným v popisu plánovacího jazyka. Každá akce skládá ze svého jména, parametrů, předpokladů a efektů. My budeme používat akce, které mají navíc ještě definované trvání. Příklad takové akce:

```
(:durative-action Jed
     :parameters (?z ?do - misto ?v - vozidlo)
     :duration (= ?duration (trvani ?z ?do))
     :condition
          (and      (at start (na ?v ?z))
               (over all (sousedi ?z ?do)))
     :effect
          (and (at start (not (na ?v ?z)))
               (at end (na ?v ?do))
               (at end (increase (total-cost)
(spotreba ?z ?do ?v))))

)
```

1.4.2 Problém

Problém je v PDDL uložen v jiném souboru než doména, aby bylo možné použít jednu doménu pro více problémů. Tento soubor obsahuje výčet objektů, které se v modelovém světě nacházejí, popis počátečního stavu pomocí instanciovaných predikátů a funkcí a seznam predikátů, které musí být splněny v cílovém stavu. Problém navíc může mít specifikovánu i metriku, podle které se určuje kvalita plánu.

2 Problém přepravy

Budeme se zabývat vizualizací plánů pro logistické úlohy, konkrétně plánů řešících problém přepravy zboží od dodavatele k zákazníkovi. Abychom mohli takové problémy vůbec řešit, musíme si určit pravidla pro jejich definování a řešení. Jak již bylo zmíněno výše, k reprezentaci plánovacích problémů budeme používat jazyk PDDL. Je zřejmé, že v logistických problémech budeme pracovat s vozidly, které převážejí balíky mezi nějakými místy. Existuje ale více způsobů, jak takové problémy v PDDL reprezentovat, které se liší zejména podle toho, jak chce uživatel s problémy pracovat. Každý z těchto způsobů je možno v PDDL vyjádřit jinou doménou. Pro potřeby této práce jsme vytvořili doménu Přeprava (viz příloha), se kterou bude program VLP pracovat. Rozšíření VLP pro práci s dalšími doménami necháme jako možnost pro budoucí vylepšení programu. Nyní si vysvětlíme, jakým způsobem tato doména popisuje problematiku přepravy.

2.1 Objekty

```
(:types
     misto vec - object
     balik vozidlo - vec
     zastavka krizovatka - misto
     sklad zakaznik – zastavka)
```

Všechny objekty v doméně Přeprava jsou odvozeny od základního typu `Object`. Rozlišujeme objekty na ty, které představují místa (typ `Místo`), a ty, které se na místech mohou nacházet (typ `Věc`). Věci se pak dělí na `Vozidla` a `Balíky`. Vozidla se mohou přesouvat mezi místy a slouží k přepravě balíků. Balíky představují zásilky, které je potřeba doručit. Existují místa, na kterých se mohou nacházet balíky (to jsou `Zastávky`), a místa, která jsou jen křižovatkami cest (typ `Křižovatka`). Na křižovatkách samozřejmě nelze nakládat ani vykládat balíky, a tak jimi vozidla obvykle jen projíždějí na cestě do jiných míst. Zastávky jsou dvou druhů. `Zákazníci` jsou místa, na která je možné doručovat balíky, zatímco `Sklady` slouží především k reprezentaci míst, ze kterých jsou zásilky rozváženy.

2.2 Predikáty

Predikáty slouží k popisu stavů plánovacího světa. Každý stav je množinou predikátů, které v něm platí. Predikáty, které nejsou uvedeny jako platné jsou automaticky neplatné (např. když součástí popisu stavu není informace, že vozidlo `?v` se nachází na místě `?m`, předpokládáme, že vozidlo se na daném místě nenachází.

2.2.1 Predikát na

```
(na ?v - vec ?m – misto)
```

Predikát na vyjadřuje, že věc (balík, vozidlo) ?v se nachází na místě ?m. V problémech, na něž se program VLP zaměruje, budeme vyžadovat, aby počáteční stav definoval pozice všech věcí a cílový stav pozice všech balíků.

Vozidla, která se v počátečním stavu nenacházejí na žádném místě nemá smysl do problému vůbec zahrnovat, protože by s nimi nešlo nijak manipulovat. Možnost, že by se balík v počátečním stavu nacházel v nějakém vozidle nepřipouštíme, protože jde o zbytečnou komplikaci.

Logicky je nesmyslné v problému definovat balíky, které nemusí být doručeny na žádné místo. Naproti tomu, můžeme vypustit určení cílové pozice vozidla, pokud je nám jedno, kde po rozvezení všech zásilek skončí.

2.2.2 Predikát sousedi

```
(sousedi ?m1 ?m2 – misto)
```

Pro vyjádření skutečnosti, že mezi dvěma místy vede cesta (spojnice, na níž neleží žádná další místa) slouží predikát sousedi. Mezi takovými dvěma místy mohou vozidla přejíždět. Sousednost míst se určí v počátečním stavu a žádná akce ji pak už nemůže změnit.

Teoreticky by mohla být sousednost dvou míst jednostranná (jednosměrky), ale editor obsažený ve VLP tvoří pro přehlednost všechna sousedství symetricky. V definici problému se tedy predikát sousedi vždy vyskytuje v párech.

2.2.3 Predikát v

```
(v ?b - balik ?v – vozidlo)
```

Predikát `v` vyjadřuje, že balík `?b` se nachází ve vozidle `?v`. Pokud je balík naložen ve vozidle, nenachází se na žádném místě, a to nezávisle na tom, kde je

právě ono vozidlo.

2.2.4 Predikát doručen

```
(dorucen ?b - balik ?z – zastavka)
```

Predikát `doručen` slouží k vyjádření skutečnosti, že balík byl doručen na místo určení. Tento predikát by neměl být součástí definice počátečního stavu (takový balík je poměrně zbytečný) a měl by být definován v cílovém stavu pro všechny balíky.

2.2.5 Predikát doručitelný

```
(dorucitelny ?b - balik ?z - zastavka)
```

Predikát `doručitelný` vyjadřuje, že balík je možné doručit. To má smysl ve chvíli, kdy požadujeme, aby byl balík doručen v určitém termínu. Součástí definice počátečního stavu je nastavení doručitelnosti všech balíků v čase 0 na hodnotu `true` a zároveň nastavení tohoto predikátu na `false` (jeho negováním) v okamžiku vypršení lhůty, v níž měl být doručen. Plánovač tak musí nalézt plán, který zajistí doručení balíku dostatečně včas.

2.3 Funkce

Funkce v PDDL zpravidla popisují vlastnosti objektů nebo vztahy mezi nimi. V našem případě jsou hodnoty všech funkcí číselné. Akce mohou hodnoty funkcí měnit pomocí operátorů `increase` a `decrease`.

2.3.1 Funkce volná-kapacita

```
(volna-kapacita ?v – vozidlo)
```

Volná kapacita vyjadřuje objem balíků, které mohou být do vozidla naloženy. Na začátku jsou všechna vozidla prázdná, a tak se volná kapacita rovná jejich maximální kapacitě. V PDDL zápisu problému maximální kapacitu vozidel neuvádíme, protože to není potřeba.

2.3.2 Funkce velikost

```
(velikost ?b - balik)
```

Velikost vyjadřuje objem balíku. Je součástí definice problému pro každý balík, který problém obsahuje a v průběhu plánu se nemění.

2.3.3 Funkce trvání

`(trvani ?z ?do - misto)`

Hodnota funkce trvání vyjadřuje, jak dlouho trvá cesta z místa `?z` na místo `?do`. Funkce je definována pro každá dvě sousedící místa a její hodnota je v obou směrech stejná. Tak jako v předchozím případě, je tato hodnota neměnná.

2.3.4 Funkce spotřeba

`(spotreba ?z ?do - misto ?v - vozidlo)`

Spotřeba je definována pro každou trojici `?z`, `?do` a `?vozidlo`, kde `?z` a `?do` jsou sousedící místa. Hodnota funkce vyjadřuje množství paliva, které vozidlo spotřebuje při cestě z jednoho místa na druhé. Tato funkce je vlastně složením dvou funkcí: vzdálenst dvou míst a spotřeba vozidla, kde spotřeba vozidla vyjadřuje množství paliva potřebného k uražení jedné jednotky vzdálenosti. Teoreticky bychom místo spotřeby mohli používat tyto dvě funkce, pak by ale součástí akcí bylo násobení, což nemusí všechny plánovače podporovat.

2.3.5 Funkce total-cost

`Total-cost` vyjadřuje množství paliva, které je spotřebováno za celý průběh plánu. Tuto funkci používáme k hodnocení kvality plánu. Přirozeně totiž chceme, aby náklady na rozvezení zásilek byly co nejnižší.

2.4 Akce

Akce v PDDL slouží k přechodům mezi stavy. Každá akce definuje podmínky, za kterých může být použita, a efekty, kterými ovlivní stav světa. Hledání plánu spočívá v hledání posloupnosti akcí, které spojují počáteční stav s cílovým.

Vzhledem k tomu, že se zabýváme plánováním s časovými zdroji, budeme používat durativní akce (durative actions). Tyto akce mají určené trvání a jejich efekty a podmínky mohou být definovány zvlášť pro začátek, konec a celou dobu trvání akce.

2.4.1 Akce Jeď

Akce `Jeď`, přesouvá vozidlo z jednoho místa na druhé. Její trvání je rovno

trvání cesty z prvního místa na druhé. Smí být použita pouze pro sousedící místa a vozidlo, kterým manipuluje musí být v době začátku akce na prvním z míst. Na začátku akce vozidlo odjede z prvního místa a na jejím konci přijede na druhé.

Celková spotřeba (`total-cost`) se touto akcí zvýší o příslušnou hodnotu. Vzhledem k tomu, že se snažíme nalézt plán s pokud možno co nejnižší celkovou spotřebou, očekáváme, že plánovač se bude snažit těmito akcemi co nejvíce šetřit.

Balíky naložené ve vozidle samozřejmě cestují s ním, ale vzhledem k tomu, že pro ně nedefinujeme aktuální pozici, nemusíme se tím v definici akce zabývat.

2.4.2 Akce Nalož

Akcí `Nalož` nakládáme balík do vozidla. Balík se do vozidla musí vejít, což znamená, že volná kapacita vozidla musí být alespoň tak velká jako je velikost balíku. Splnění této podmínky vyžadujeme až na konci nakládání, protože v jeho průběhu mohly být jiné balíky z vozidla vyloženy (nebo do něj naloženy). Vozidlo a balík se musí pochopitelně po celou dobu nakládání nacházet na stejném místě.

V okamžiku, kdy nakládání skončí, balík se přestane nacházet na daném místě a je naložen ve vozidle. Volná kapacita vozidla se v tu chvíli zmenší o velikost balíku.

Tuto akci lze používat jen ve skladech. Na křižovatkách se žádné balíky nesmějí nacházet, a tak tam logicky nemohou být nakládány. Nakládání u zákazníků by zdánlivě nevadilo, ale pokud bychom to dovolili, mohla by vozidla používat zákazníky jako překladiště balíků, což nechceme.

2.4.3 Akce Vylož

Akce `Vylož` je symetrická k akci `Nalož`, až na to, že vykládat lze jak ve skladech, tak u zákazníků. Balík je vyložen na místo, na kterém se právě nachází příslušné vozidlo. Volná kapacita vozidla se zvětší o velikost balíku hned na začátku akce.

2.4.4 Akce Doruč

K doručení balíku na jeho místo určení slouží akce `Doruč`. Může být použita pouze pokud se balík nachází na správném místě a ještě nevypršela jeho lhůta doručitelnosti. Tato akce v podstatě balíkem nijak nemanipuluje, pouze jej označí

jako doručený a dále se nemůže účastnit žádných akcí (toho docílíme tak, že mezi efekty akce zařadíme odstranění balíku z aktuálního místa).

Poslední tři akce mají konstantní délku trvání, jednu časovou jednotku, což je ve srovnání s trváním akce Jeď zanedbatelná doba.

2.5 Metrika

Metrika plánovači říká, podle čeho má hodnotit kvalitu nalezených plánů. My jsme si za metriku zvolili celkovou spotřebu paliva a chceme ji minimalizovat. Spolu s nastavením termínů doručení balíků to má ten efekt, že plánovače se budou snažit ušetřit co nejvíce paliva, ale zároveň stihnout všechny termíny. Což přibližně odpovídá zjednodušenému modelu reálného problému přepravy zásilek.

3 Analýza logistického plánu

V této části se zaměříme na to, jaké skutečnosti nás zajímají při analýze plánu pro logistické úlohy a ukážeme si, jak k této analýze použít program VLP.

3.1 Přeprava zásilek

Představme si, že máme firmu, která se přepravou zabývá. Firma vlastní několik skladů, v nichž jsou umístěny balíky, které je potřeba doručit k zákazníkům. Každý balík má určen termín doručení, tedy čas, v němž nejpozději musí být doručen k zákazníkovi. Balíky nelze doručovat se zpožděním. Dále má firma k dispozici vozidla, která přepravu balíků zajišťují. Vozidla se mohou lišit podle toho, kolik balíků se do nich vejde a jakou mají spotřebu. Samozřejmě máme k dispozici mapu prostředí, v němž se budou vozidla pohybovat. Víme, mezi kterými místy vede cesta, jak je tato cesta dlouhá (tj. kolik paliva na ní vozidlo spotřebuje) a jak dlouho trvá projet touto cestou.

Nyní chceme vytvořit plán, jak všechny balíky doručit včas a přitom minimalizovat náklady na jejich doručení (celkové množství paliva spotřebovaného všemi vozidly).

3.2 Vizualizace problému a plánu

Řekněme, že jsme úlohu výše popsaného typu předložili plánovači a ten sestavil plán, který tuto úlohu řeší. Je poměrně snadné zjistit, zda je plán korektní, tedy zda úlohu skutečně řeší (balíky jsou doručeny včas, vozidla jezdí pouze po cestách, atp.). Také není problém zjistit kvalitu plánu, kterou je zde množství spotřebovaného paliva. Rozhodnout, zda by nebylo možné úlohu vyřešit lépe je už mnohem obtížnější.

I když je plán korektní, mohou se v něm vyskytovat různé nedostatky. Například akce, které k řešení problému nijak nepřispívají, nebo takové akce, které by mohly být nahrazeny efektivnějšími akcemi. Odstraněním těchto nedostatků bychom mohli dosáhnout zlepšení kvality plánu. Nalézt takové neoptimality v plánu ale nemusí být jednoduché. Pro člověka by bylo velmi obtížné zorientovat se v plánu

zapsaném jako seznam akcí, a proto je potřeba nějak si jej vizualizovat. Taková vizualizace by měla přehledně zobrazovat, co se v plánu kdy děje (přesun vozidel, kde se nachází které balíky, kolik toho které vozidlo najezdilo).

Díky tomu, že se zaměřujeme pouze na jeden typ problémů, můžeme si je vizualizovat velmi přirozeným a přehledným způsobem, což v jiných programech zaměřujících se na plánovací problémy obecně (např. GIPO) nelze.

Vytvoříme si mapu míst a propojíme je cestami podle toho, jak spolu sousedí. Mapa je tedy vlastně graf, v němž vrcholy reprezentují místa a hrany cesty mezi nimi. Vzdálenost míst na mapě nemusí odpovídat jejich skutečné vzdálenosti, ale v popisu každého místa jsou uvedeny vzdálenosti jeho sousedů. Součástí popisu místa jsou také seznamy vozidel a balíků, které se na něm nachází.

Plán pak vizualizujeme jako simulaci jeho provedení. Na mapě tedy zobrazujeme přesouvající se vozidla a po každém kroku simulace aktualizujeme informace o všech objektech.

Tento přístup nám umožňuje sledovat celkový pohyb vozidel na mapě i zaměřit se na konkrétní objekt a pozorovat, jak se jeho stav mění v průběhu simulace.

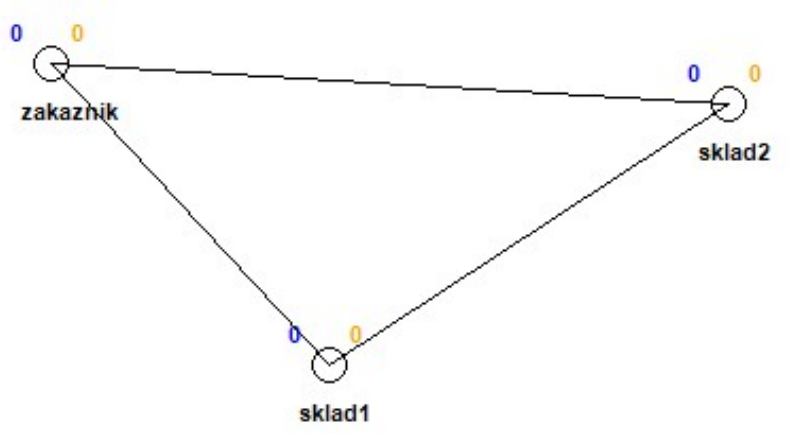
3.3 Analýza plánu

Pokud analyzujeme plán, zpravidla si potřebujeme nejprve vytvořit přehled o tom, co se v jeho průběhu děje, a potom se zaměřit na ty jeho části, které chceme prozkoumat. VLP plán simuluje a přitom zaznamenává informace o pohybu vozidel a balíků.

Při analýze plánu s pomocí VLP je nejlepší nejprve shlédnout celou simulaci, přičemž si uživatel může utvořit představu o tom, jak plán vypadá jako celek. Například zda jsou využívána všechna vozidla, která má firma k dispozici nebo jestli přepravu zásilek zvládá jen několik z nich, zda jsou využívány všechny cesty na mapě nebo jen některé, a podobně. Po skončení simulace se může zaměřit na jednotlivé objekty (vozidla, balíky) a podívat se, jak se během ní přesouvaly (tyto informace jsou k dispozici i v průběhu simulce). Při dalším spuštění simulace pak uživatel může sledovat, jak se mění stavy těch objektů, které ho zaujaly.

Podíváme se na několik typů neoptimalit, které se v plánu řešícím logistický problém mohou vyskytnout (a výrazně zhoršit jeho kvalitu) a na jednoduchých příkladech ukážeme, jak využít VLP k jejich odhalení. Všechny plány uvedené v příkladech jsou vytvořeny úpravou plánů vygenerovaných plánovačem LPG-td spuštěným s paramterem `-speed` (při použití tohoto parametru LPG-td vrací první plán, který je řešením problému). V původních plánech se nacházely neoptimality stejných typů. Plán je i s těmito neoptimalitami korektní, ale jejich odstraněním může dojít k výraznému zlepšení kvality plánu. PDDL zápis všech uvedených příkladů (problém i plán) je součástí testovacích dat přiložených k programu VLP.

Všechny příklady budou zasazeny do stejného prostředí (na stejnou mapu), ve kterém se vyskytují tři místa, dva sklady a jeden zákazník. Všechna tato místa jsou propojena cestami a pro vzdálenosti mezi nimi platí trojúhelníková nerovnost. Nejkratší cesta mezi dvěma z nich vede tedy přímo po jejich spojnici (Obrázek 1). Pro každý příklad zvolíme trochu jinou kombinaci vozidel a balíků.



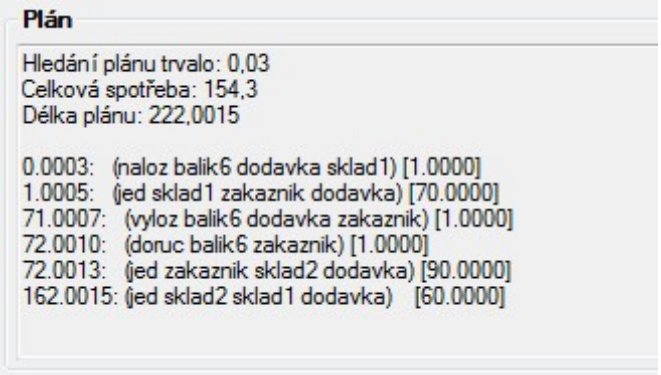
Obrázek 1: Mapa prostředí

3.3.1 Vozidlo spotřebuje příliš mnoho paliva

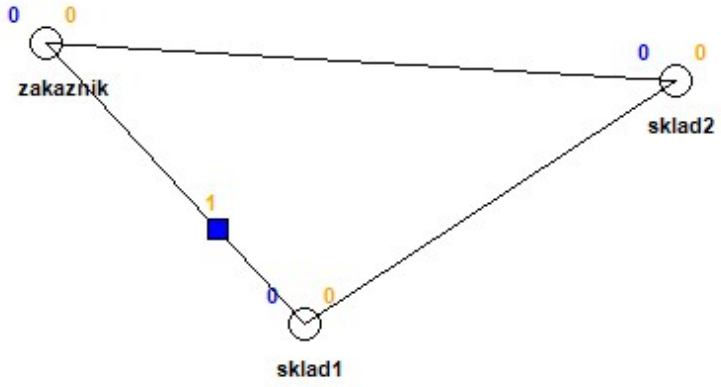
Do skladu 1 umístíme balík, který má být doručen k zákazníkovi. Na doručení balíku je dostatek času. Ve skladu 1 se také nachází vozidlo, do kterého se balík vejde. Toto vozidlo se má po doručení balíku vrátit do skladu 1.

Plánovač nám pro tento problém vytvořil plán, který můžeme vidět na Obrázku 2. Spustíme-li simulaci plánu, můžeme vidět, že vozidlo dle očekávání naložilo balík a zamířilo přímo k zákazníkovi (Obrázek 3). Po doručení balíku ale

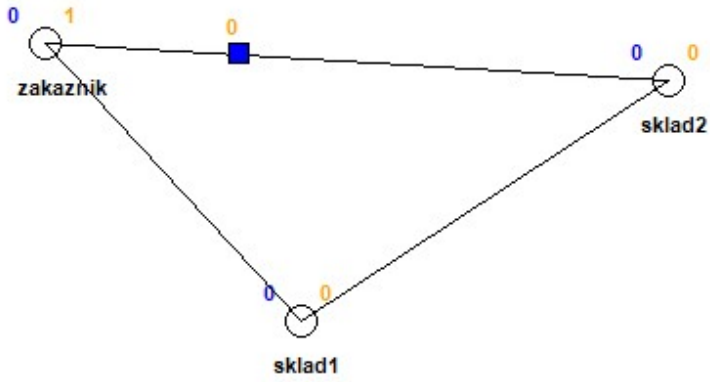
vozidlo nezamíří přímo zpět do skladu 1, jak bychom předpokládali. Místo toho zvolí objížďku přes sklad 2, jak vidíme na Obrázku 4, což, jak víme, je nevýhodné, protože vozidlo tak spotřebuje více paliva.



Obrázek 2: Plán



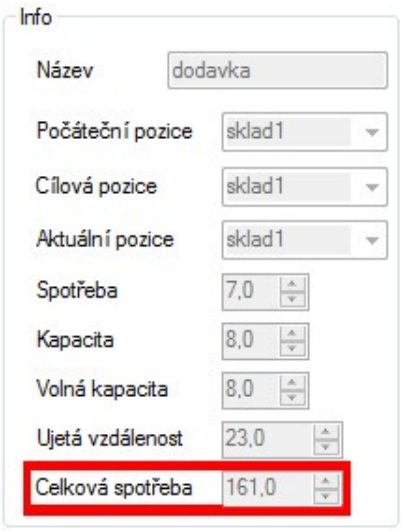
Obrázek 3: Vozidlo jede dle očekávání



Obrázek 4: Vozidlo se vydalo špatným směrem

V takto jednoduchém případě jsme nevhodné chování vozidla odhalili

okamžitě, pokud bychom ale měli složitější problém, který by obsahoval více objektů, nemuseli bychom si tohoto nedostatku při pouhém sledování simulace všimnout. VLP proto u každého vozidla zobrazuje informaci o tom, kolik paliva vozidlo projezdilo od začátku simulace plánu (Obrázek 5). Snadno tedy zjistíme, pokud má vozidlo neočekávaně vysokou spotřebu. Při opakovaném spuštění simulace se na toto vozidlo zaměříme a sledujeme jeho cestu na mapě. Okamžitě vidíme, kudy vozidlo jede a kde přitom nakládá a vykládá balíky. Na první pohled je pak vidět, pokud vozidlo prázdné chaoticky jezdí po mapě nebo pokud zbytečně zajíždí na místa, na nichž nenakládá ani nevykládá balíky a prodlužuje si tím trasu.



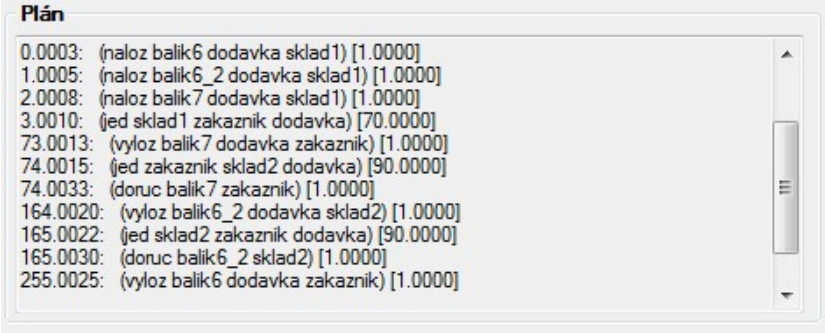
Obrázek 5: Celková spotřeba vozidla

3.3.2 Balík zbytečně cestuje

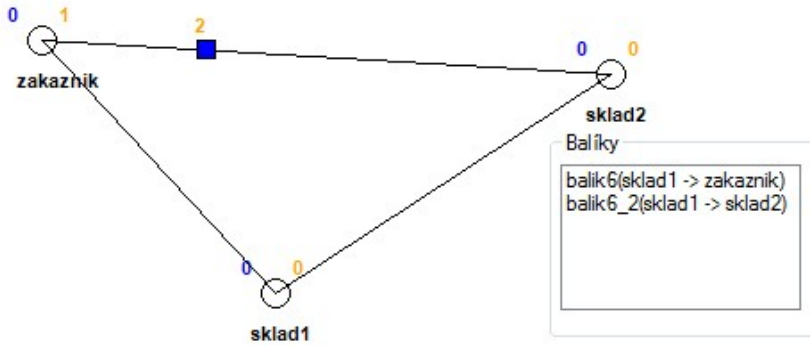
Nyní do skladu 1 umístíme tři balíky, které se všechny najednou vejdou do jednoho vozidla. Cílem dvou z nich bude zákazník a cílem třetího sklad 2. Ve skladu je k dispozici vozidlo, do něhož se balíky vejdou.

Předpokládejme, že pro tento problém nám plánovač vytvořil plán, jehož důležitá část je zobrazena na Obrázku 6. Spustíme-li simulaci plánu, vidíme, že vozidlo dle očekávání naložilo všechny tři balíky a vydalo se s nimi na cestu k zákazníkovi. Tam ale vyložilo jen jeden z nich a se zbylými dvěma balíky pokračuje do skladu 2 (Obrázek 7). Je jasné, že lepší by bylo, kdyby vozidlo rovnou vyložilo oba balíky určené k zákazníkovi, protože takto se s druhým balíkem ještě

bude muset vracet.

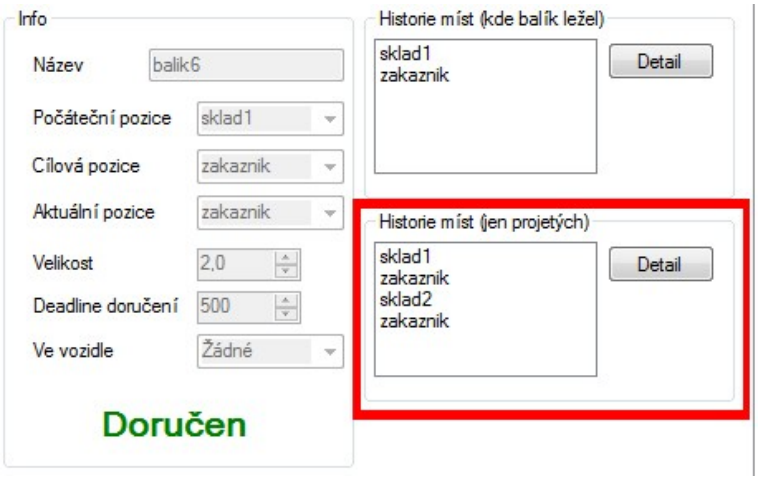


Obrázek 6: Plán



Obrázek 7: Vozidlo nevyložilo balík

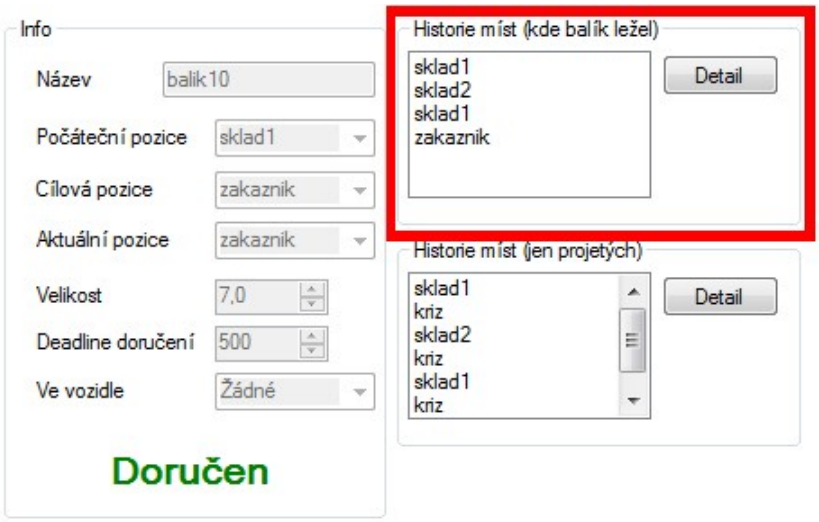
Ve VLP tento problém snadno odhalíme, i kdybychom si ho nevšimli v průběhu simulace. Program totiž zaznamenává místa, kterými balík procestoval. Pokud si uživatel po skončení simulace prohlédne seznam těchto míst, okamžitě by viděl, kdyby balík procestoval místem svého určení, a pak byl ještě provezen několika dalšími místy, než byl doručen (Obrázek 8).



Obrázek 8: Historie míst

3.3.3 Zbytečné nakládání a vykládání balíku

Balík může být před svým doručením několikrát převezen mezi sklady, ve kterých čeká na vozidlo, které ho později dopraví k zákazníkovi. To je často výhodnější, než aby byl balík prostě naložen na své výchozí pozici a rovnou dopraven na místo svého určení. Může se ale stát, že je balík takto překládán, až se ocitne na místě, na kterém už jednou byl. To jistě nechceme. Tato neoptimalita se vyskytuje spíše u komplikovanějších problému, při jejichž řešení jsou balíky hodně překládány. Nebudeme zde proto uvádět konkrétní příklad, v němž k ní dochází, jen ukážeme, jak ji odhalit.



Obrázek 9: Balík je vyložen na místě, kde už ležel

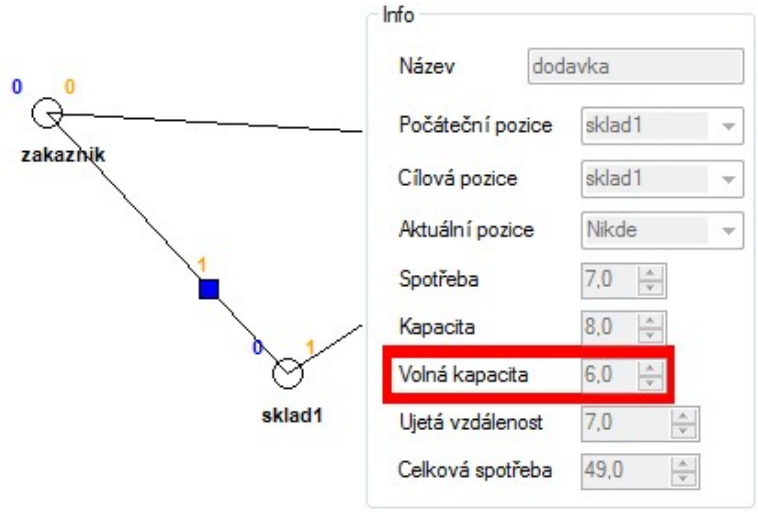
VLP sleduje nejen to, kterými místy byl balík provezen, ale také na kterých místech byl uskladněn. Prohlédne-li si uživatel tento seznam (Obrázek 9), rychle zjistí, opakuje-li se v něm nějaké místo dvakrát (v tomto případě je to sklad 1). Je jasné, že veškeré manipulace s balíkem mezi těmito dvěma jeho uskladněními byly zbytečné.

3.3.4 Vozidlo jezdí poloprázdné

V dalším příkladu budeme mít ve skladu 1 dva balíky, které je potřeba doručit na stejné místo. Ve skladu je opět k dispozici vozidlo, do kterého se oba balíky vejdou. Problém předáme plánovači, který vytvoří plán (Obrázek 10).

Obrázek 10: Plán

Spustíme-li simulaci tohoto plánu, vidíme, že vozidlo vyrazilo ze skladu pouze s jedním balíkem, přestože v jeho parametrech můžeme vidět, že má dostatečnou kapacitu pro naložení obou balíků (Obrázek 11). Později se pak vozidlo bude pro druhý balík vracet a doručovat ho zvlášť.



Obrázek 11: Vozidlo naložilo jen jeden balík

Odhalit podobnou neoptimalitu v složitějším plánu by pochopitelně bylo obtížnější. Uživatel by musel sledovat konkrétní vozidlo a pokaždé, když by se mu zdálo, že je nedostatečně naložené, by se musel podívat, zda na místě, kterým vozidlo projíždí není balík, který by vozidlo mohlo naložit. Je samozřejmě možné, že i kdyby vozidlo balík naložilo, kvalita plánu by se nezměnila, ale to už si musí uživatel vyzkoušet.

3.3.5 Výsledek analýzy

Uživatel prováděl analýzu plánu, aby v něm nalezl případné nedostatky. Aplikace VLP je určena k tomu, aby mu při této analýze pomohla, ale už se nezabývá

nápravou nalezených nedostatků. Chce-li uživatel přimět plánovač k tomu, aby tvořil kvalitnější plány, musí sáhnout pro jiné nástroje. Pokud je uživatel vývojářem plánovače, který plán sestavil, může plánovač upravit na základě výsledku analýzy, kterou s pomocí VLP provedl. Jestliže uživatel nemůže nebo nechce zasahovat do algoritmu, který plánovač používá k nalezení plánu, může se pokusit spustit plánovač s jinými parametry (např. inicializovat generátor náhodných čísel). Další možností je změnit termíny doručení u balíků tak, aby je bylo nutné doručit co nejpřímější cestou. Případně by uživatel mohl upravit plánovací doménu (např. přidat objektivní funkci, která penalizuje zbytečné nakládání a vykládání balíku). VLP zatím dokáže pracovat pouze s jednou doménou, ale je možné, že v budoucnu bude rozšířen tak, aby podporoval různé domény.

3.4 Srovnání s jinými programy

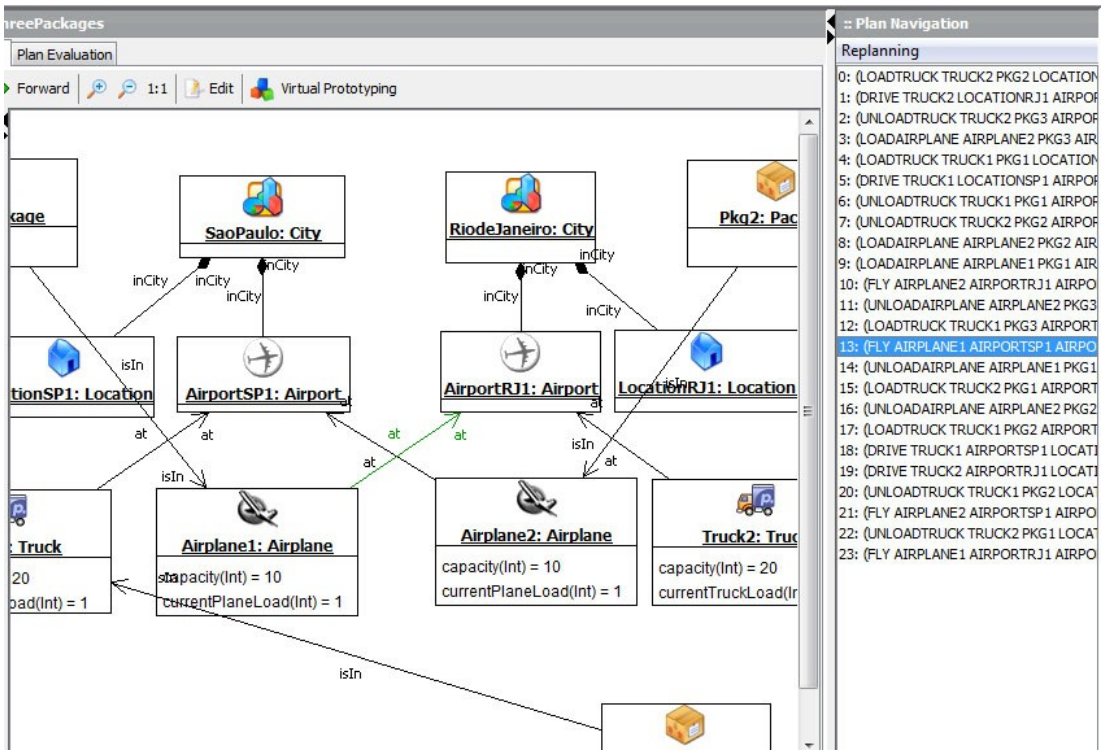
Pro analýzu plánů samozřejmě již existují jiné nástroje. Krátce si zde několik z nich popíšeme a porovnáme je s programem VLP.

3.4.1 itSIMPLE

itSimple (Integrated Tools Software Interface for Modeling PLanning Environments) [9] je nástroj, který byl vyvinut pro usnadnění tvorby plánovacích domén. Pro reprezentaci plánovacích modelů používá jazyk UML (Unified Modeling Language) [11], který je určen pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.

itSIMPLE umožňuje modelování plánovacích domén i problémů v přehledném grafickém prostředí. Tyto domény a problémy je možno přeložit do jazyka PDDL. Program také nabízí možnost vytvoření plánu pro zadaný problém pomocí některého z přiložených plánovačů (např. Metric-FF, FF, SGPlan, MIPS-xxl, LPG-td, LPG, hspsp, SATPlan, Plan-A, blackbox, LPRPG, Marvin).

Plán, ať už vytvořený pomocí přiloženého plánovače nebo externě jinak, může být vizualizován pomocí nástroje Movie Maker. Ten vytvoří sérii diagramů, které zobrazují, jak plán mění stavy světa (Obrázek 12).



Obrázek 12: itSIMPLE Movie Maker

Movie Maker přehledně zobrazuje, jak akce změní stav světa. V jednom okně může uživatel vidět, stav před akcí, ve druhém po ní. Přitom jsou změny, které akce provede, zvýrazněny.

itSIMPLE navíc umožňuje sledování změn proměnných (např. volné kapacity vozidla, ale ne třeba aktuální pozice vozidla) v průběhu plánu.

S pomocí tohoto programu by bylo na rozdíl od VLP obtížné například sledovat, zda balík není nakládán a vykládán zbytečně (viz 3.3.3). Uživatel by si musel zaznamenávat, kterými místy balík prošel, což VLP dělá za něj. Zjištění, zda balík necestuje zbytečně i poté, co byl provezen svou cílovou pozicí (3.3.2) by bylo o něco snažší, ale uživatel by se musel na balík soustředit po celou dobu simulace. Ve VLP stačí podívat se na seznam míst, kterými byl balík provezen.

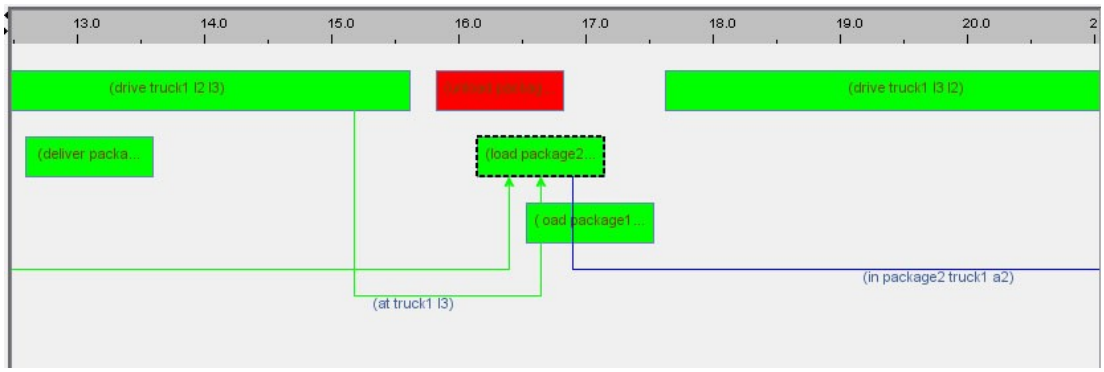
Je také těžké rozhodnout, zda vozidlo někde cestuje zbytečně, protože v Movie Makeru sice můžeme vidět, která místa spolu sousedí, ale tyto vztahy jsou znepřehledněny dalšími vztahy mezi objekty (např. kde se nachází který balík nebo vozidlo).

Celkově VLP vizualizuje logistické plány přirozeněji než itSIMPLE a nabízí

více informací o jednotlivých objektech potřebných pro analýzu plánu.

3.4.2 VisPlan

VisPlan (Visualization and Verification of Plans) [3] je nástroj pro vizualizaci a verifikaci plánů. Zaměřuje se na hledání a zobrazování kauzálních vztahů mezi akcemi. To znamená, že pro každou akci vyhledá akce, které změnily stav světa tak, aby ji bylo možno použít (pokud není předpoklad akce platný už v počátečním stavu). Program zobrazí plán jako graf, v němž vrcholy představují akce a hrany kauzální vztahy mezi nimi (Obrázek 13).



Obrázek 13: VisPlan - vizualizace plánu

VisPlan dokáže zpracovat i nekorektní plány. V těch vyhledá akce, jejichž předpoklady nejsou splněny a vyznačí je červeně. U těchto akcí kauzální vztahy pochopitelně zobrazeny nejsou.

Program nabízí možnost manuálně upravovat plán přidáváním, odebíráním a editací akcí. Změny se v plánu projeví okamžitě, takže uživatel může hned vidět, zda svou úpravou například opravil chybu v plánu.

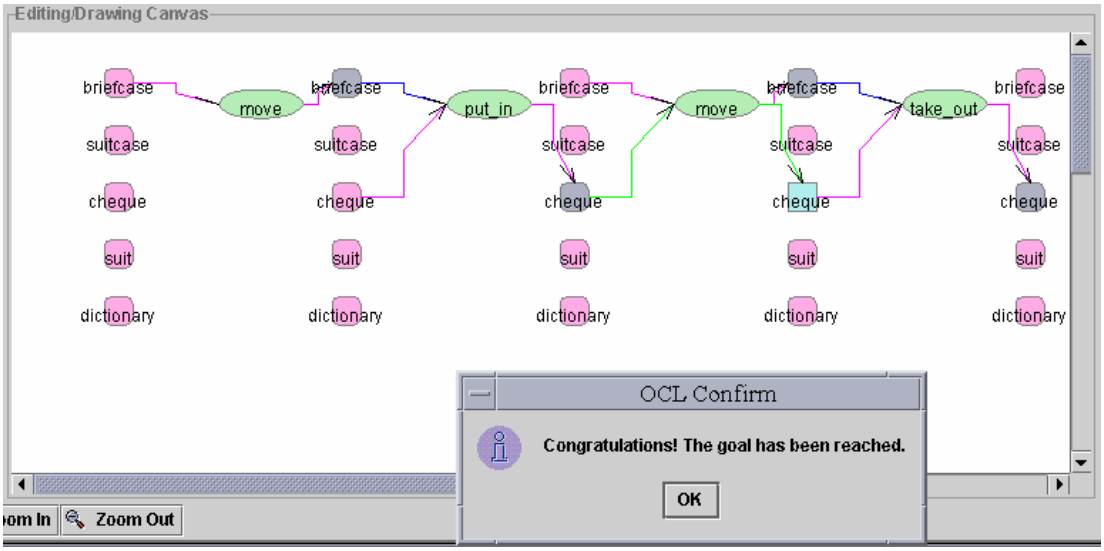
VisPlan zobrazuje informace o tom, jak akce mění stav stav světa, ale není schopen popsat stav, v němž se nachází nějaký konkrétní objekt. Pokud by tedy uživatel při analýze logistického plánu chtěl například vědět, kde je právě umístěn nějaký balík, musel by se podívat na aktuální stav světa a tuto informaci z něj vyčíst. Program nijak nevizualizuje strukturu světa, takže není možné vidět, jak vypadá mapa, ani jak se vozidla přesouvají mezi místy. Kdyby se uživatel při analýze plánu například snažil zjistit, zda vozidlo jezdí po optimálních trasách, musel by procházet

akce, které vozidlo přesunují, a zjišťovat, zda má vozidlo důvod jet tam, kam jede (bude tam vykládat nebo nakládat balíky).

VLP na rozdíl od VisPlanu zobrazuje mapu světa a přehledný popis objektů v aktuálním stavu, což umožňuje lepší analýzu optimality plánu. VisPlan se naproti tomu zaměřuje spíše na verifikaci plánu, tedy na analýzu toho, proč plán je nebo není korektní.

3.4.3 GIPO

GIPO (Graphical Interface for Planning with Objects) [6] je nástroj pro tvorbu a editaci plánovacích domén v grafickém uživatelském prostředí a jejich export do PDDL.



Obrázek 14: GIPO - Animator

Program uživateli umožňuje tvorbu objektových typů, predikátů a akcí, z nichž lze sestavit doménu. Dále si uživatel může pro doménu vytvořit zadaní problémů, s jejichž pomocí ji pak testuje. Součástí programu je plánovač, který lze použít k sestavení plánu pro problém. Uživatel si plán může vizualizovat pomocí nástroje Animator (Obrázek 14), který znázorňuje vliv akcí na objekty v modelovém světě. Další nástroj, který je součástí programu, Stepper, umožňuje dokonce manuální tvorbu plánu, při níž jsou akce vizualizovány stejným způsobem jako v Animatoru.

Animator uživateli sice pomůže orientovat se v tom, kterými objekty akce

manipulují, ale nezobrazí už stav, ve kterém se objekty nacházejí po aplikaci akce. S pomocí tohoto nástroje tedy téměř není možné odhalovat jakékoliv neoptimality v plánu. Kdyby uživatel například chtěl sledovat trasu vozidla, musel by si u každé akce, která vozidlo přesouvá, zaznamenat, kde se bude nacházet po jejím provedení. Pokud uživatel například zkoumal, zda vozidlo nedělá zbytečné zajížďky, musel by si vypisovat i to, kde vozidlo nakládá a vykládá balíky. Podobně obtížné by bylo i sledování cesty balíku nebo změn volné kapacity vozidla.

Zatímco GIPO je nástroj pro práci s plánovacími úlohami obecně, VLP je zaměřen pouze na úlohy logistické. Proto může poskytnout mnohem vyšší stupeň vizualizace (GIPO vizualizaci plánů s časem vůbec nepodporuje) a bližší informace o stavech světa. GIPO je vhodný spíše pro editaci domén, kdežto VLP se zaměřuje na analýzu plánu.

3.4.4 Výsledky srovnání

VLP se hodí k analýze logistických plánů více než výše zmíněné programy. Na rozdíl od nich se více zaměřuje na popis stavu světa z pohledu jednotlivých objektů, o kterých poskytuje velké množství informací. Způsob, jakým plány vizualizuje, vytváří lepší představu o tom, jak by jejich provedení vypadalo. Tabulka 1 zobrazuje přehled vybraných vlastností všech čtyřech programů.

Srovnání vlastností programů				
	VLP	**itSIMPLE**	**VisPlan**	**GIPO**
doménově nezávislé problémy	ne	ano	ano	ano
tvorba domén	ne	ano	ne	ano
tvorba problému	ano	ano	ne	ano
export problému do PDDL	ano	ano	-	ano
vizualizace plánu	ano	ano	ano	částečně
simulace plánu	ano	ano	ne	částečně
mapa logistického problému	ano	částečně	ne	ne
simulace pohybu vozidel	ano	ne	ne	ne
historie cesty balíků	ano	ne	ne	ne

Tabulka 1: Přehled vybraných vlastností programů

4 Uživatelská dokumentace

V této části se budeme zabývat použitím programu VLP a jeho popisem z hlediska uživatele. Hlavním účelem programu je umožnit uživateli testování plánovače a analýzu plánů, které tvoří. Proto nabízí nástroje pro snadnou editaci problému, spuštění plánovače a vizualizaci plánu. Uživatel by se měl před použitím programu seznámit s doménou `Přeprava`, která popisuje modelový svět, s nímž VLP pracuje. Doménu lze nalézt v příloze, její podrobný popis se nachází v kapitole 3 této práce.

4.1 Instalace programu

Program je určen pro operační systém Windows XP nebo novější.

Instalaci aplikace uživatel provede spuštěním souboru setup.exe a následným postupem podle instrukcí.

Součástí instalačního balíčku je PDDL doména, se kterou aplikace pracuje, několik problémů a plánů, na nichž je možné aplikaci vyzkoušet a plánovač LPG-td[2].

4.2 Spuštění programu

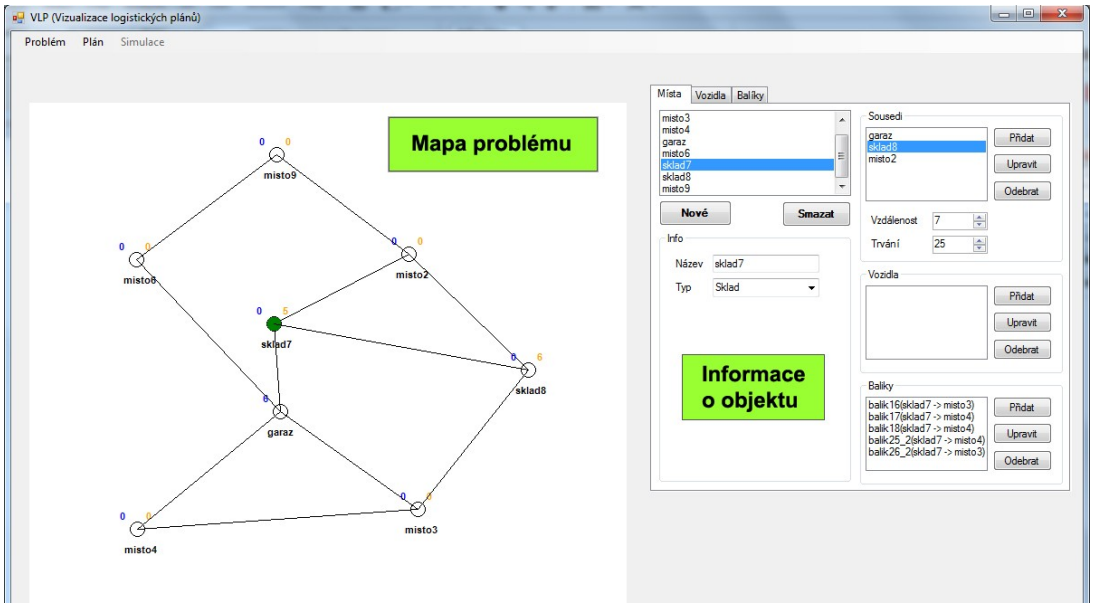
Aplikace se spustí souborem vlp.exe, v módu editace problému. Po spuštění programu se otevře okno s novým (prázdným) problémem. V horní části okna je hlavní menu, nalevo se nachází oblast, do níž je kreslena mapa a napravo jsou zobrazeny informace o objektech, které jsou součástí problému. Uživatel má možnost buďto rovnou začít tvořit nový problém nebo pokračovat v práci na některém již uloženém. Načíst rozpracovaný problém může volbou Problém → Otevřít... v hlavním menu.

4.3 Editace problému

V módu editace problému uživatel tvoří a upravuje logistický problém (Obrázek 15). V problému existují tři základní typy objektů: místo, vozidlo a balík. V pravé části okna se nachází seznamy těchto objektů (na každé záložce jeden typ) a informace o nich (jejich vlastnosti). Pod každým seznamem je tlačítko Nový/é,

2 http://zeus.ing.unibs.it/lpg/

kterým uživatel vytvoří nový objekt daného typu a tlačítko Smazat, kterým vybraný objekt smaže. Každý objekt musí mít unikátní název. Novému objektu je název automaticky vygenerován (<typ objektu><nějaké číslo>). Uživatel jej může libovolně měnit (v kolonce Název), pokud dodrží pravidlo, že žádné dva objekty se nejmenují stejně.



Obrázek 15: Editace problému

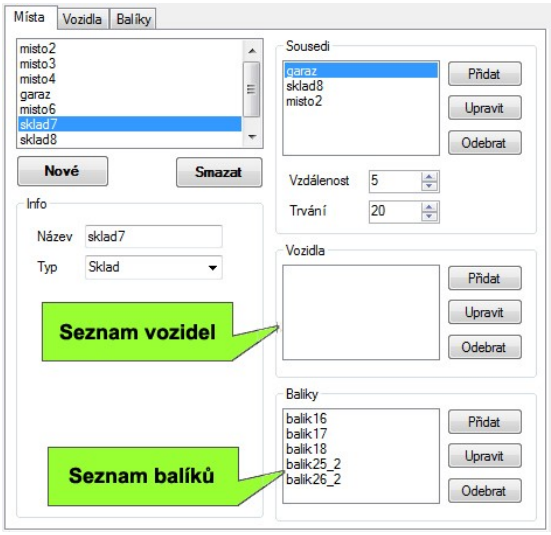
4.3.1 Místa

Při tvorbě nového místa jej musí uživatel po klepnutí na tlačítko Nové umístit na mapu (klepnutím myší). Místo je na mapě zobrazeno jako kružnice s popiskem (prvních 8 písmen názvu). Klepnutím myší dovnitř kružnice uživatel zobrazí informace o příslušném místě. U každého místa je zobrazeno modré číslo, které vyjadřuje počet vozidel, které se na něm právě nachází. Pokud je místo zastávka, je u něj oranžově zobrazen počet balíků, které se na něm právě nachází.

Počáteční typ místa je vždy křižovatka. Typ může uživatel změnit výběrem z roletky Typ. Po změně typu na zákazníka nebo křižovatku se ověří, zda místo není počáteční (případně cílovou) pozicí nějakého balíku. Pokud tomu tak je, pozice se ve vlastnostech balíku změní na „Nikde“ (neurčená pozice).

Vlastnosti místa (Obrázek 16) obashují také seznay vozidel a balíků, které se na něm nacházejí (v editaci to znamená balíky a vozidla, jejichž počáteční pozicí je

dané místo) a seznam jeho sousedů. Do těchto seznamů lze přidávat tlačítkem Přidat, které se nachází vedle každého z nich. Po jeho stisknutí se zobrazí dialogové okno s nabídkou objektů, které lze přidat (všechny objekty příslušného typu, které v seznamu ještě nejsou). Volbu objektu uživatel potvrdí tlačítkem OK (tlačítka Cancel a X okno také zavřou, ale ponechají seznam nezměněný). Klepnutím na tlačítko Odebrat uživatel odebere vybraný objekt ze seznamu. Tlačítkem Upravit zobrazí informace o vybraném objektu a následně může upravovat jeho vlastnosti.



Obrázek 16: Vlastnosti míst

U každého souseda je uvedena jeho vzdálenost od daného místa a trvání cesty k němu. Zobrazí se v oblasti pod seznamem sousedů po výběru položky v něm. Počáteční hodnota obou těchto vlastností je 100 a uživatel ji může měnit v rozmezí 1-10.000 (pouze celá čísla). Sousedství je vždy oboustranné, pokud tedy uživatel přidá místu X jako souseda místo Y, místu Y se mezi sousedy zařadí místo X se stejnými parametry. Na mapě je sousedství dvou míst znázorněno úsečkou mezi středy kružnic reprezentujících tato místa.

4.3.2 Vozidla

V seznamu vozidel je za názvem vozidla uvedena jeho počáteční a cílová pozice (`<počáteční pozice> → <cílová pozice>`).

Počáteční a cílovou pozici vozidla lze změnit výběrem místa v roletce vedle příslušného popisku. Počáteční pozice každého místa musí být určena, cílová pozice může být nastavena na „Nikde“. Po vytvoření vozidla jsou obě pozice nastaveny na „Nikde“.

Dále je u každého vozidla uvedena jeho spotřeba a kapacita. Počáteční hodnota obou těchto vlastností je nastavena na 10 a uživatel ji může měnit v rozmezí 1-100 (s přesností na jedno desetinné místo). Spotřeba udává množství paliva, které vozidlo spotřebuje uražením jedné jednotky vzdálenosti. Celkové množství spotřebovaného paliva je pak měřítkem kvality plánu.

4.3.3 Balíky

Stejně jako u vozidel je v seznamu balíků za názvem balíku uvedena jeho počáteční a cílová pozice.

Počáteční a cílovou pozici balíku uživatel mění stejným způsobem jako u vozidel. Počáteční pozice balíků jsou ale omezeny pouze na sklady a cílové pozice na sklady a zákazníky. Navíc balík na rozdíl od vozidla musí mít určenou nejen počáteční, ale i cílovou pozici.

Další vlastností balíku je velikost. Ta udává jak velkou část kapacity vozidla balík zabere, je-li do něj naložen. Počáteční velikost balíku je 1 a uživatel ji může měnit v rozmezí 0,1-20 (s přesností na jedno desetinné místo).

Nakonec u balíku uvádíme deadline doručení (nejpozdější čas, ve kterém může být balík doručen).

4.3.4 Uložení problému

Uživatel problém uloží výběrem položky Problém → Uložit. Pokud byl problém už někdy předtím uložen, uloží se do stejného souboru (přepíše ho a stará verze problému tedy bude ztracena). Jinak program uživateli nabídne dialog, jehož prostřednictvím bude moci vybrat soubor, do kterého problém chce uložit. Pokud chce uživatel starou verzi problému zachovat, může jej uložit volbou Problém → Uložit jako..., kterou vyvolá ukládací dialog. Problém se uloží v PDDL formátu.

Součástí problému je také rozmístění míst na mapě, vzdálenosti mezi sousedícími místy a spotřeba jednotlivých vozidel. Tyto informace plánovače nepoužívají a jsou tedy uvedeny v komentářích PDDL souboru popisujícího problém. Program umí načíst i soubor, který tyto informace neobsahuje, ale nezobrazí místa na mapě a všechny vzdálenosti a spotřeby nastaví na jejich počáteční hodnoty. Je tedy doporučeno načítat pouze problémy vytvořené pomocí této aplikace nebo vhodně upravné.

U všech vozidel a balíků definovaných v problému musí být určena jejich počáteční pozice a u všech balíků i pozice cílová. Při ukládání problému tedy musí být tyto pozice definovány (nastaveny na jinou hodnotu než „Nikde“).

4.4 Vizualizace plánu

Chce-li uživatel vizualizovat plán, musí mít v módu editace otevřený problém, jehož je plán řešením. Poté může buďto načíst již vytvořený plán volbou Plán → Načíst... v hlavním menu nebo předat problém externímu plánovači volbou Plán → Vytvořit. V takovém případě program spustí vybraný plánovač a počká na jeho výstup, který pak zpracuje.

4.4.1 Tvorba plánu

Pokud uživatel chce vytvořit plán volbou Plán → Vytvořit, může si vybrat ze dvou alternativ. První je spustit libovolný plánovač (volba Vlastní). Pokud zvolí tuto možnost, stačí aby vybral soubor, který spouští plánovač, a VLP už mu předá problém, který má řešit. Jakmile plánovač doběhne, načte se plán, který vytvořil. Vybraný plánovač je spuštěn s parametry `-f <soubor, v němž je uložen problém> -o <soubor, v němž je uložena doména> -out <soubor, do kterého se uloží plán>`. Pokud jej chce uživatel spustit s jinými parametry, musí tak učinit externě a plán pak načíst volbou Načíst.

Druhou možností je vytvořit plán s pomocí přiloženého plánovače LPG-td. Uživatel si může vybrat, zda jej chce spustit s parametrem „`-speed`“ (Rychle) nebo „`-quality`“ (Kvalitně). V módu Rychle LPG-td skončí po nalezení prvního plánu, který řeší problém. Pokud je ale spuštěn v módu Kvalitně, rozhodne se po nalezení prvního řešení, kolik času věnuje snaze o jeho zlepšení (a zpravidla nějaké lepší řešení najde).

Nevýhodou plánovače LPG-td je, že nenalezne řešení vždy, když existuje. Při řešení komplexnějších problémů může skončit chybou „`REACHABILITY ANALYSIS ERROR`“. V takovém případě žádný plán nevytvoří. Uživatel může zkusit takový problém předložit jinému plánovači (např. SGPLAN6[3]**) a plán pak načíst volbou** Plán → Načíst...

3 Ke stažení na stránkách soutěže IPC-2008, http://ipc.informatik.uni-freiburg.de/

Již hotový plán uživatel načte volbou Plán → Načíst.... V takovém případě aplikace očekává vstupní soubor se seznamem akcí, přičemž každá akce je zapsána na samostatném řádku následujícím způsobem:

```
<začátek akce>: (<název akce> <parametry akce>)
                [<trvání akce>]
```

Přitom začátek a trvání akce musí býr zapsán jako desetinné číslo, ve kterém je desetinná část oddělena tečkou, a parametry akce musí být odděleny mezerou. Na velikosti písmen nezáleží. Tedy například:

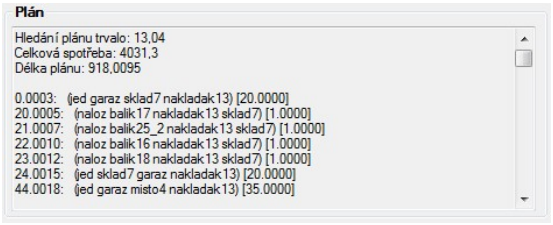
```
298.046: (JED MISTO2 MISTO9 NAKLADAK14) [60.0000]
```

Program je navíc schopen zpracovat informace o tom, jak dlouho trvalo plán vytvořit a jaká je jeho kvalita, pokud je soubor obsahuje v komentářích zapsané následovně:

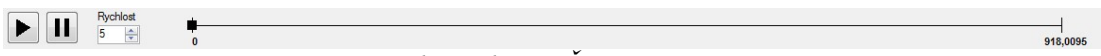
```
; Time <doba tvoření plánu>
; Quality <kvalita plánu>
```

4.4.2 Simulace plánu

Jakmile je plán načten, vypíše se v pravém dolním rohu okna spolu s informacemi o jeho délce, o tom, jak dlouho trvalo jej vytvořit, a o jeho kvalitě (pokud jsou k dispozici) (Obrázek 17). V dolní části okna se navíc objeví časová osa znázorňující, v jaké fázi je simulace plánu (na začátku 0), tlačítka spouštějící a pozastavující simulaci a kolonka, v níž je možno nastavit rychlost simulace. Rychlost simulace vyjadřuje, o kolik časových jednotek se simulace posune za jednu sekundu (Obrázek 18).



Obrázek 17: Výpis plánu

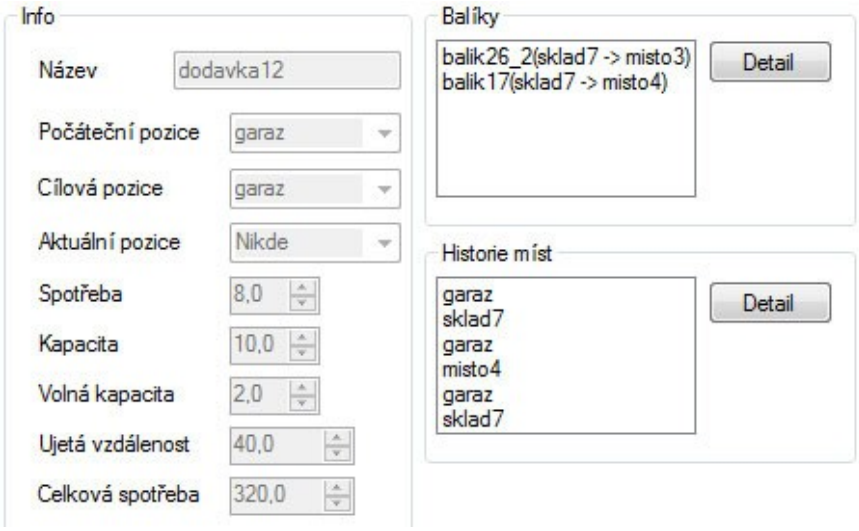


Obrázek 18: Časová osa

V pravé části okna zůstanou informace o vlastnostech objektů, ale v módu

simulace je není možné měnit. Navíc několik vlastností přibude. Konkrétně jsou to u vozidel (Obrázek 19):

- Aktuální pozice – vyjadřuje, na kterém místě se vozidlo právě nachází. Pokud je někde na cestě mezi dvěma místy, je nastavena na „Nikde“.
- Ujetá vzdálenost – kolik jednotek vzdálenosti už vozidlo urazilo. Při přesunu mezi dvěma místy se přičte vzdálenost mezi nimi v okamžiku, kdy vozidlo vyrazí na cestu.
- Celková spotřeba – množství paliva, které vozidlo dosud spotřebovalo
- Balíky – seznam balíků, které jsou aktuálně naloženy ve vozidle.
- Historie míst – seznam míst, kterými vozidlo projelo od začátku simulace. Místo se do seznamu přidá v okamžiku, kdy jej vozidlo opustí.



Obrázek 19: Vlastnosti vozidla

Nové vlastnosti u balíků:

- Aktuální pozice – podobně jako u vozidel. Pokud je balík naložen ve vozidle, má aktuální pozici nastavenou na „Nikde“, nezávisle na tom, kde je vozidlo.
- Ve vozdile – vyjadřuje, ve kterém vozidle je balík právě naložen. Pokud není ve vozidle, je vlastnost nastavena na „Žádné“.
- Doručen – objeví se v momentě doručení balíku. Znamená, že balík byl úspěšně a včas doručen.
- Historie míst (kde balík ležel) – seznam míst, na kterých byl balík uskladněn

- Historie míst (jen projetých) – seznam míst, kterými byl balík provezen

Pomocí tlačítka Detail, uživatel zobrazí informace o objektu vybraném v příslušném seznamu.

Vozdila, která právě nejsou na žádném místě, ale cestují mezi nimi, jsou na mapě znázorněny čtverečky se středem na spojnici mezi příslušnými místy. Klepnutím myší na čtvereček uživatel zobrazí informace o vozidle, které reprezentuje.

Simulace se spustí volbou Simulace → Spustit v hlavním menu nebo tlačítkem **>**. Každou vteřinu se odsimuluje tolik jednotek času, kolik je nastaveno v políčku Rychlost. Vyhodnotí se akce, které v uplynulém okamžiku proběhly a vozidla na mapě se posunou o úsek, který mezitím stihly ujet.

Při vyhodnocování akcí se nejen mění vlastnosti objektů, ale také se ověřuje, zda lze akci v daném momentě provést. Pokud to není možné, akce se nevyhodnotí, simulace je přerušena a zobrazí se chybové hlášení.

Simulaci je kdykoliv možné zastavit volbou Simulace → Pozastavit nebo tlačítkem **||**. Pokud se uživatel rozhodne pustit simulaci znovu od začátku, učiní tak volbou Simulace → Od začátku.

Kdykoliv v průběhu simulace je možné vrátit se zpět k editaci problému volbou Problém → Upravit v hlavním menu.

5 Programátorská dokumentace

Nyní se zaměříme na popis implementace programu. Vysvětlíme, jak je v programu reprezentován problém, jak probíhá simulace plánu a jak je tento proces vizualizován.

Použité algoritmy nemají optimální složitost, ale pro rozumně velké problémy je odezva aplikace dostatečně rychlá (nepředpokládáme, že by aplikace byla používána k vizualizaci příliš velkých problémů, protože s velikostí problému rychle roste náročnost hledání plánu a ta se brzy stává pro plánovače neúnosnou).

5.1 Operační prostředí a rozhraní

Aplikace byla vytvořena v prostředí Microsoft Visual Studio 2010 v jazyce C# a je určena pro operační systém Windows XP nebo novější. Uživatelské prostředí je vytvořeno s pomocí technologie Windows Forms.

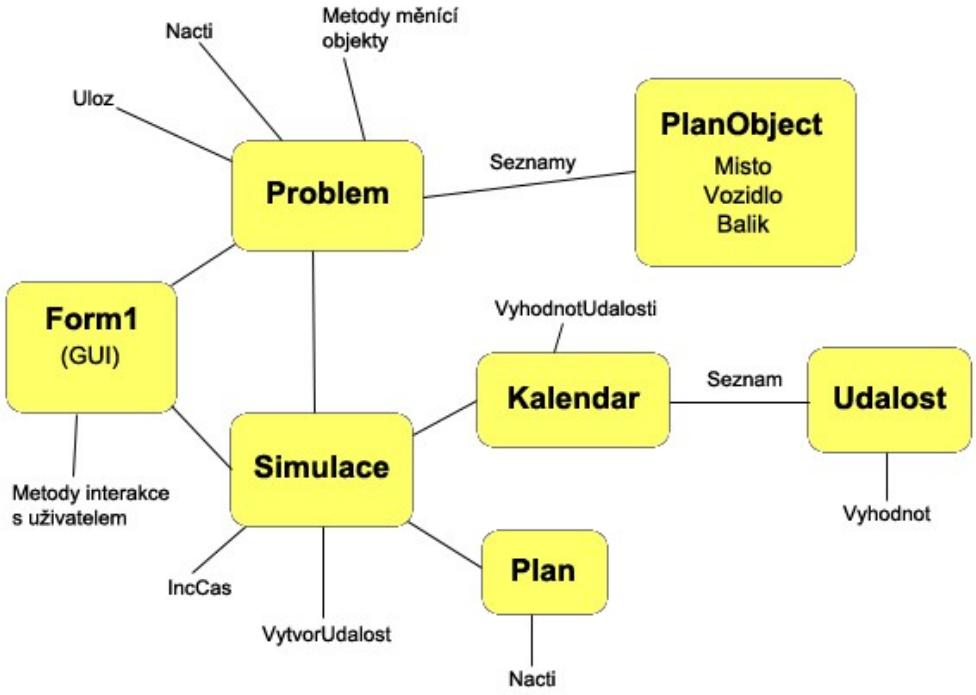
5.2 Struktura nejdůležitějších tříd a jejich metod

Program se skládá ze tří základních částí. První z nich zajišťuje reprezentaci problému, druhá se stará o simulaci plánu a třetí komunikuje s uživatelem (graficky zobrazuje problém i plán a reaguje na uživatelovy akce). Strukturu nejdůležitějších částí programu ilustruje Obrázek 20.

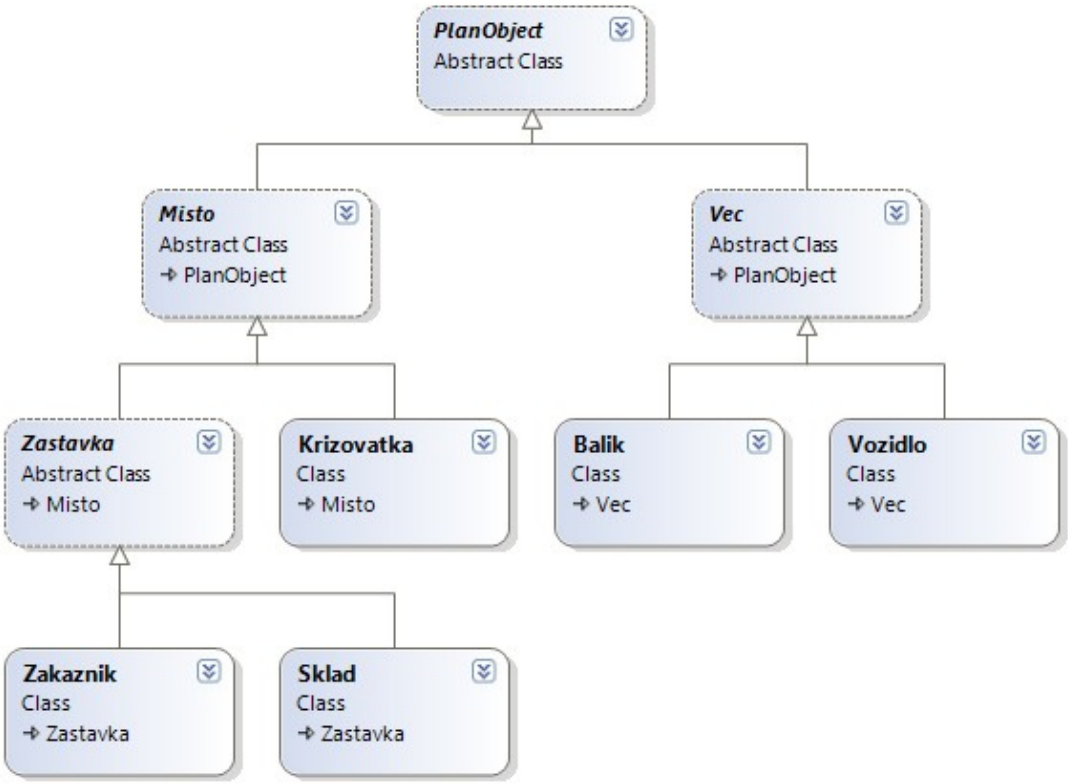
5.3 Reprezentace problému

V PDDL je problém popsán výčtem objektů, které se v něm vyskytují, vlastnostmi, které objekty mají, vztahy mezi jednotlivými objekty a podmínkami, které musí být splněny v cílovém stavu. VLP se touto reprezentací inspiruje, ale vzhledem k tomu, že se zabývá pouze problémy z jedné konkrétní domény, reprezentace je přizpůsobena právě této doméně.

Všechny objekty v modelovém světě jsou odvozeny od abstraktní třídy `PlanObject` a každý z nich má unikátní id a název. Tato třída definuje virtuální metodu `VypišTyp()`, která vrací typ objektu. Strukturu tříd reprezentujících objekty modelového světa zobrazuje Obrázek 21.



Obrázek 20: Struktura nejdůležitějších tříd a metod



Obrázek 21: Struktura objektů

5.3.1 Místo

U každého místa je uveden seznam míst, se kterými sousedí, a pro každého souseda je zde jeho vzdálenost od místa a trvání cesty k němu. Vzdálenost mezi dvěma místy je pak v PDDL souboru uváděna pouze v komentářích, pro potřeby aplikace VLP. Při ukládání problému se vzdálenost spolu se spotřebou vozidla uloží pro každé dvě sousedící místa jako spotřeba vozidla na cestě z jednoho místa do druhého. Všechny tyto parametry se mění pouze při editaci problému.

Dále je u každého místa uveden seznam vozidel, které na něm aktuálně jsou. V definici problému to znamená vozidla, která mají dané místo uvedeny jako počáteční pozici. Poté při simulaci se obsah tohoto seznamu mění podle toho, jak vozidla přijíždějí a odjíždějí.

V souvislosti s tím třída `Místo` obsahuje metody `Odjelo(Vozidlo)` a `Dorazilo(Vozidlo)`. První z nich odebere vozidlo ze seznamu a druhá jej na něj přidá.

Abstraktní třída `Místo` má dva potomky. Jsou jimi `Křižovatka` a `Zastávka`. `Křižovatka` představuje místo, na kterém nemohou být žádné balíky, zatímco na `Zastávce` ano. `Zastávka` tedy obsahuje seznam balíků, které jsou na ní umístěny a metodu `Přivez(Balík)`, která balíky do seznamu přidává. Na všechny zastávky lze balíky přivážet, ale odvážet je je možno pouze ze skladů. Proto má třída `Zastávka` ještě potomky `Sklad` a `Zákazník`.

`Sklad` obsahuje na rozdíl od `Zákazníka` metodu `Odvez(Balík)`, protože reprezentuje místo, na němž balíky začínají. `Zákazník` představuje místo, na které jsou balíky doručovány, a proto z něj balíky nemohou být odváženy.

5.3.2 Vozidlo

Třída `Vozidlo` obsahuje veškeré informace o vozidle. Pro přehled o kapacitě vozidla jsou zde proměnné `kapacita` a `volná_kapacita`. První z nich se mění pouze při editaci problému a vyjadřuje celkovou kapacitu vozidla, druhá se mění v průběhu simulace podle aktuální volné kapacity a na začátku simulace je rovna celkové kapacitě. V problému se pak uvádí pouze volná kapacita.

Proměnná `spotřeba` udává množství paliva, které vozdilo spotřebuje za ujetou jednotku vzdálenosti a do problému se ukládá pouze v komentářích, protože

slouží k výpočtu spotřeby vozidla mezi jednotlivými místy.

`historie_míst` a `ujetá_vzdálenost` slouží pouze pro lepší přehled o pohybu vozidla a v problému ani plánu se vůbec neuváději.

Proměnné `na_cestě`, `na_cestě_z`, `na_cestě_do` a `začátek_cesty_čas` slouží k popisu akce `Jeď` a více si o nich řekneme v části popisující simulaci plánu.

Nakonec třída obsahuje seznam balíků, které jsou aktuálně ve vozidle. Tento seznam je na začátku prázdný a v průběhu simulace se do něj přidávají a z něj odebírají balíky metodami `Nalož(Balík)` a `Vylož(Balík)`.

5.3.3 Balík

Třída `Balík` obsahuje informace o velikosti balíku, nejpozdějším termínu jeho doručení a vozidle, ve kterém se balík nachází. Je-li balík včas doručen, nastaví se proměnné `doručen` a `doručen_včas` na hodnotu `true` (na začátku simulace mají hodnotu `false`).

Součástí třídy jsou dále metody sloužící k naložení do a vyložení z vozidla (změna hodnot proměnných `vozidlo` a `aktuální_pozice`) a doručení balíku.

5.3.4 Věc

Třídy `Vozidlo` a `Balík` jsou odvozeny od abstraktní třídy `Věc` reprezentující objekt, který může být přesouván mezi místy. Informace o počáteční, cílové a aktuální pozici jsou tedy uvedeny již v definici této třídy. Platí, že počáteční pozici musí mít neustále uvedenu jak vozidla, tak balíky, zatímco cílová pozice je povinná pouze pro balíky, vozidla ji mohou mít nastavenu na hodnotu `-1`. Aktuální pozice je na začátku simulace stejná jako počáteční a v jejím průběhu udává místo, na němž se věc právě nachází. Pokud se na žádném místě nenachází, je nastavena na hodnotu `-1` (pokud je balík naložen ve vozidle, jeho aktuální pozice je vždy `-1`, bez ohledu na to, kde je vozidlo).

Vždy, když je u objektu uveden seznam objektů, které obsahuje (např. u místa seznam vozidel, které se na něm nacházejí), jde o seznam jejich id, ne odkazů na ně.

Jak si čtenář může všimnout, struktura těchto tříd odpovídá struktuře typů definovaných v doméně `Přeprava` (viz Příloha). Každá třída pak obsahuje metody, které mění stav objektu podle toho, jak změnu definují akce uvedené v doméně. To je pak velmi užitečné při simulaci plánu.

5.3.5 Problém

Instance třídy `Problém` obsahuje seznam všech objektů, které jsou v daném problému definovány. Pro rychlejší přístup k jednotlivým objektům navíc obsahuje zvlášť seznamy míst, vozidel a balíků. Protože v PDDL jsou objekty rozlišovány podle jejich názvů, je součástí třídy i převodní tabulka z názvu na id (opačně to není potřeba, protože podle id si snadno najedeme objekt v seznamu všech objektů a u něj už je pak uveden i jeho název).

Třída také obsahuje metody, které umožňují editaci problému, tedy vytváření, modifikace a odebírání objektů. Tyto metody jsou vesměs velmi přímočaré a čtenář je snadno pochopí při čtení jejich zdrojového kódu, nebudeme je zde tedy explicitně popisovat.

5.3.6 Převod do PDDL a zpět

Námi zvolenou reprezentaci problému převedeme do PDDL poměrně snadno. Zajišťuje to metoda `Ulož(soubor)`, která je součástí třídy `Problém`.

Nejprve se vypíše seznam objektů obsažených v problému a jejich typů (`Vypiš_Objekty()`).

Poté se vypíší hodnoty funkcí a predikáty popisující počáteční stav (`Vypiš_Init()`). Výpis funkcí je celkem přímočarý, až na výpis spotřeby vozidla a lhůty doručení balíku. Spotřeba se totiž v PDDL zápisu problému uvádí pro každou trojici `místo1`, `místo2` a `vozidlo`, kde `místo2` je sousedem `místa1`. Pro každá dvě sousedící místa a každé vozidlo se tedy vzdálenost mezi místy vynásobí spotřebou vozidla a zapíše v PDDL. Lhůta doručení balíku se zapisuje pomocí časem iniciovaných literálů (timed-initial-literals). To znamená, že pro každý balík se v čase 0 nastaví jeho doručitelnost na true a v momentě, kdy vyprší lhůta jeho doručení se nastaví na false a tento balík již nebude možné doručit.

Dále se vypíší cíle, tedy predikáty, které musí být splněny v cílovém stavu. To

je doručení všech balíků na určená místa (včas, jinak to nejde) a přítomnost vozidel na určených cílových pozicích, pokud je mají nastaveny (`Vypiš_cíle()`).

Nakonec se vypíší komentáře generované editorem (`Vypiš_pozice()`, `Vypiš_spotřeby` a `Vypiš_Vzdálenosti()`). Ty obsahují pozice míst na mapě, vzdálenosti mezi místy a spotřeby vozidel. Pozice míst jsou zapsány takto (každá na samostatné řádce):

```
;#[x,y]<název místa>
```

Spotřeba vozidla a vzdálenost mezi místy je pak zapsána stejně jako funkce v PDDL, ale je uvozena znaky `;%`, aby se dalo poznat, že jde o funkci určenou pro VLP.

Převod problému z PDDL do naší reprezentace je o něco obtížnější. Implementuje jej funkce `Načti(soubor)`, která je součástí třídy `Problém`.

Postupujeme tak, že nejprve načteme celý soubor, v němž je problém uložen, do paměti. Při tom z načítaného textu odstraňujeme komentáře. Pokud jde o komentáře určené pro VLP (pozice míst na mapě, vzdálenosti mezi místy, spotřeba vozidel), odkládáme si je stranou. Ostatní komentáře zahazujeme.

Načtený text pak zpracováváme po blocích. Blok je úsek textu ohraničený závorkami, které k sobě patří nebo jedno slovo ohraničené bílými znaky (tedy např. `(= (trvání místo1 místo2) 100.0)` je blok, zatímco `(at 500.0 (not (doručitelný balík místo))` blok není, protože první závorka v něm nebyla dosud uzavřena). K načtení bloku slouží metoda `NactiBlok(ref string s)`, která vrací první blok obsažený v řetězci s a odebírá jej z něj.

Celý problém zapsaný v PDDL je vlastně jedním blokem, který obashuje několik menších bloků (`(:objects ...)`, `(:init ...)`, atd...). Každý z těchto bloků zpracovává zvláštní metoda, pokud nejde o blok triviální (např. `(:domain <jméno domény>)`). Tyto bloky jsou opět složeny z několika menších bloků.

Metoda `Načti_objects(blok)` čte názvy objektů definovaných v problému a vždy, když dostane jejich typ, vytvoří odpovídající objekty (VLP sice vypisuje ke každému objektu jeho typ, ale PDDL umožňuje i zápis několika objektů stejného typu oddělených mezerou za sebou, a až poté určení jejich typu). Vždy při

výrobě se objekt zařadí do seznamu všech objektů, do příslušného seznamu podle typu a jeho název a id se přidá do převodní tabulky.

Metoda `Načti_init(blok)` načítá funkce a predikáty definované v problému. Blok, který je zápisem funkce, začíná znakem „=“ na rozdíl od bloku, který je zápisem predikátu. Podle hodnoty funkce nebo predikátu se nastavují vlastnosti objektů, kterých se týkají. Paramtery funkcí a predikátů (včetně jejich názvu) musí být odděleny právě jednou mezerou.

Blok obsahující cíle je zpracováván metodou `Načti_goal(blok)`. Vzhledem k tomu, že obsahuje predikáty popisující cílový stav, zpracovává se podobně jako blok obsahující predikáty popisující počáteční stav.

Nakonec, když je celý problém načten, zpracují se pomocné funkce a pozice míst uložené v komentářích souboru. Pomocné funkce se zpracovávají úplně stejně jako běžné funkce. Pozice míst se pak ukládají do seznamu, který je využíván při kreslení mapy.

5.4 Simulace plánu

Plán je simulován tak, že akce v něm použité mění vlastnosti objektů definovaných v problému. Při simulaci plánu používáme stejnou třídu `Problém`, jako při tvorbě problému, který je plánem řešen. Proto musí být před načtením plánu otevřen příslušný problém.

Plán je vlastně seznam akcí, z nichž každá má určený začátek a trvání. Jednotlivé efekty akce se projeví buďto na jejím začátku nebo konci. Každou akci proto můžeme snadno převést na jednu nebo dvě události. První událost nastane na začátku akce a druhá po skončení jejího trvání. Tyto události pak vyhodnocujeme ve chvíli, kdy nastanou.

5.4.1 Načtení plánu

Nejprve si vytvoříme instanci třídy Kalendář, která slouží právě k uložení událostí.

Akce jsou v souboru obsahujícím plán zapsány takto:

```
<začátek akce>: (<název akce> <parametry akce>)
                [<trvání akce>]
```

Přitom každá akce je na samostatném řádku. Akce se tedy skládá ze tří bloků, a tak k jejímu načtení můžeme využít již zmíněnou metodu `NactiBlok()`.

Pro každou akci vytvoříme událost, která nastane na jejím začátku. Vzhledem k tomu, že pouze akce `Jeď` má trvání delší než jednu časovou jednotku, budeme vytvářet koncovou událost pouze pro tuto akci. Ostatní akce jsou v porovnání s ní téměř okamžité, a tak je jejich trvání při simulaci zanedbatelné. Všechny efekty těchto akcí tedy odsimulujeme hned na jejich začátku. Vytvořenou událost přidáme do seznamu událostí v kalendáři.

5.4.2 Události

Existuje pět druhů událostí. `Odjeď`, `Doraž`, `Nalož`, `Vylož` a `Doruč`. První dva odpovídají akci `Jeď`, zbylé tři stejnojmenným akcím. Každý z těchto pěti druhů je reprezentován třídou, která je potomkem abstraktní třídy `Událost`. Tato třída obsahuje virtuální metodu `Vyhodnoť()`, která je spuštěna `Kalendářem` ve chvíli, kdy má příslušná událost nastat. Každá událost mění vlastnosti objektů, kterých se týká, na základě efektů, které měla akce, z níž byla událost vytvořena.

Událost `Odjeď` spustí při svém vyhodnocení metodu `Odjeď(odkud, kam, čas, vzdálenost)` vozidla, kterého se týká, a metodu `Odjelo(Vozidlo)` místa `odkud`. Metoda vozidla nastaví jeho proměnnou `na_cestě` na hodnotu `true`, přiřadí proměnným `na_cestě_z` a `na_cestě_do` id příslušných míst, přičte `vzdálenost` k `ujeté_vzdálenosti` a zaznamená `čas` začátku cesty. Metoda místa pak odebere vozidlo ze seznamu vozidel, které se na místě nacházejí.

Když nastane událost `Doraž` ukončující akci, kterou událost `Odjeď` započala, spustí se metoda `Doraž()` příslušného vozidla, která nastaví parametry vozidla na potřebné hodnoty a metoda `Dorazilo(Vozidlo)` místa, na které se vozidlo přesunulo.

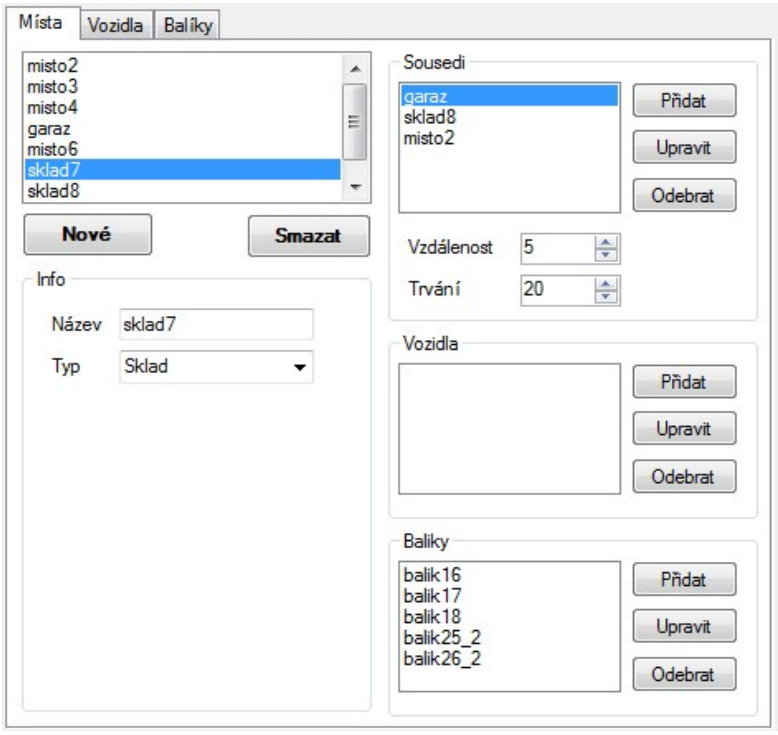
Vyhodnocování ostatních událostí funguje podobně. Vždy platí, že když událost mění vlastnosti objektů, zkontroluje, zda jsou splněny podmínky definované příslušnou akcí. Pokud nejsou, simulace se zastaví a je vyvolána výjimka, která uživateli oznámí, kde došlo k nepovolenému použití akce.

5.4.3 Krok simulace

Simulace vizualizuje průběh plánu po krocích nastavených uživatelem (proměnná `rychlost`). Každou sekundu reálného času zvýší čas simulace o hodnotu `rychlosti`. Poté vezme seznam událostí, který je setříděn podle času, v němž událost nastane, a postupně provádí události, které měly nastat před aktuálním okamžikem. Po vyhodnocení je událost odebrána ze seznamu, takže se v dalším kroku nevyhodnotí znova.

5.5 Vizualizace problému a plánu

Uživatelské prostředí je vytvořeno pomocí technologie Windows Forms. Uživatel má k dispozici `PictureBox`, na kterém je nakreslena mapa problému. Mapa je interaktivní, takže když uživatel klepne myší na nějaký objekt na ní znázorněný, zobrazí se mu informace o něm (načtou se z paramterů vybraného objektu). Informace o všech objektech jsou zobrazovány v pravé části okna (Obrázek 22). Tyto informace, stejně jako mapa, se obnovují při každé změně vlastností nějakého objektu a po provedení každého kroku simulace plánu. K tomu slouží metoda `Obnov()`.



Obrázek 22: Informace o místech

Identifikace objektu, na který uživatel klikl, probíhá tak, že se určí okolí pozice myši nad mapou. Pak se vybere okolí této pozice a procházením seznamu objektů na mapě se zjišťuje, zda je v tomto okolí umístěn nějaký objekt. Pokud je takový objekt nalezen, je nastaven jako aktivní a zobrazí se informace o něm. Nejprve se prohledává seznam míst, a až pak seznam vozidel. Někdy se může stát, že v oblasti kliknutí je více různých objektů. V takovém případě je jako aktivní nastaven první, který byl nalezen.

Při přepnutí z editace problému na simulaci plánu (tedy při načtení plánu) je vlastnost `Enabled` všech prvků, které mění problém, nastavena na hodnotu `false`. To uživateli znemožní v průběhu simulace upravovat problém. Při přepnutí zpět se tato vlastnost opět nastaví na `true`.

5.5.1 Vykreslování mapy

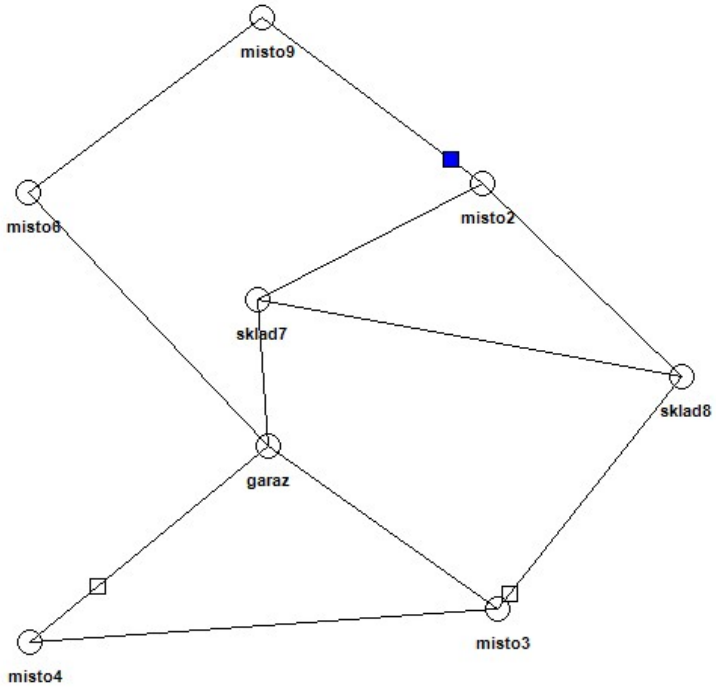
K danému problému máme k dispozici seznam pozic na mapě všech míst, které problém obsahuje. Každé místo v tomto seznamu znázorníme kružnicí se středem na příslušné pozici. Poté vedeme úsečku z této pozice k pozici každého místa, které s ním sousedí.

Při simulaci problému musíme vykreslit také vozidla, která se pohybují mimo místa. Při každém kroku simulace vypočteme, kde se vozidlo nachází na cestě mezi místy a tuto pozici uložíme. Následně pak vozidlo znázorníme čtverečkem se středem na jeho pozici. Tak dosáhneme toho, že vozidla se při simulaci zdánlivě pohybují (Obrázek 23).

5.5.2 Úprava vlastností objektů

Pokud uživatel změní vlastnost některého objektu, nejprve se ověří, zda je nová hodnota vlastnosti přípustná (například při změně názvu se ověřuje jeho unikátnost, a zda se skládá z povolených znaků), a pak je zavolána metoda problému, která daný objekt změní.

Podobně i vytváření a mazání objektů probíhá prostřednictvím metod problému. Z uživatelského rozhraní nelze do problému zasahovat přímo.



Obrázek 23: Mapa modelu

Závěr

Program VLP, který je výsledkem této práce, poskytuje nástroje k editaci logistických úloh a následné vizualizaci jejich řešení. S jeho pomocí může uživatel přirozeně a přehledně reprezentovat problém přepravy zásilek a exportovat tuto reprezentaci do PDDL. VLP umožňuje přímé předání problému externímu plánovači a následné převzetí řešení, které plánovač sestaví. Dále je program schopen řešení vymodelovat a zobrazit v podobě simulace.

V průběhu simulace může uživatel sledovat vlastnosti objektů v modelu i veškeré změny, které jsou důsledky prováděných akcí. Program navíc při simulaci akce ověří, zda ji lze na aktuální stav použít. VLP je tedy užitečný například při testování plánovače nebo při kontrole plánu před jeho skutečným provedením.

VLP je zaměřený na velmi úzkou část spektra plánovacích úloh, a tak na rozdíl od vetšiny existujících programů podobného zaměření přizpůsobuje vizualizaci konkrétnímu typu úloh. To umožňuje velmi přehlednou simulaci plánu.

V budoucnu by bylo možné program rozšířit tak, aby sám vyhledával některé neoptimality v plánu (např. opakované vykládání balíku na stejném místě), aby pracoval s více doménami, případně aby se s jeho pomocí daly tvořit domény logistickým úlohám na míru. Nabízí se také implementace podpory pro úlohy zapsané v novějších verzích PDDL. Dále může být vylepšeno grafické zobrazení modelu, které je v současnosti velmi jednoduché.

Seznam použité literatury

[1] Roman Barták: Plánování a rozvrhování, Webová stránka přednášky, http://kti.mff.cuni.cz/~bartak/planovani/index.html, Univerzita Karlova v Praze (červenec 2011)

[2] Fox, M.; Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, volume 20, 61-124, AAAI Press

[3] Radoslav Glinský: VisPlan – Visualization and Verification of Plans, Webová stránka projektu, http://glinsky.org/visplan/, Radoslav Glinský (červenec 2011)

[4] S. Koenig: IPC - International Planning Competition, Webová stránka soutěže, http://ipc.icaps-conference.org/, University of Southern California (červenec 2011)

[5] Petr Koupý: GraphRec, Webová stránka projektu, http://www.koupy.net/graphrec.php, Petr Koupý (srpen 2011)

[6] T. L. McCluskey: GIPO – Graphical Interface for Planning with Objects, Webová stránka projektu, http://scom.hud.ac.uk/planform/gipo/, University of Huddersfield (červenec 2011)

[7] D. McDermott. PDDL: the Planning Domain Definition Language. Technical Report. Yale Center for Computational Vision and Control, Yale University, CT, USA, 1998.

[8] S. Russel, P. Norvig: Artificial Intelligence - A modern approach, Prentice Hall, 2003, ISBN 0-13-790395-2.

[9] T. Vaquero, F. Tonidandel, J. Silva: itSIMPLE – Integrated Tools Software Interface for Modeling PLanning Environments, Webová stránka projektu, http://dlab.poli.usp.br/twiki/bin/view/ItSIMPLE/WebHome, University of Sao Paulo

(červenec 2011)

[10] Wikipedia, internetová encyklopedie: Lisp (programming language), http://en.wikipedia.org/wiki/Lisp_%28programming_language%29, Wikimedia Foundation (červenec 2011)

[11] Wikipedia, internetová encyklopedie: Unified Modeling Language http://en.wikipedia.org/wiki/Unified_Modeling_Language, Wikimedia Foundation (červenec 2011)

Seznam použitých zkratek

GIPO – Graphical Interface for Planning with Objects (grafické rozhraní pro plánování s objekty)

IPC – International Planning Competition (mezinárodní plánovací soutěž)

itSIMPLE – Integrated Tools Software Interface for Modeling PLanning Environments (integrované nástroje softwarového rozhraní pro modelování plánovacích prostředí)

PDDL – Planning Domain Definitoin Language (jazyk pro definici plánovacích domén)

UI – Umělá Inteligence

UML – Unified Modeling Language (jednotný jazyk pro modelování)

VisPlan - Visualization and Verification of Plans (vizualizace a verifikace plánů)

VLP – Vizualizace Logistických Plánů

Příloha

PDDL doména Přeprava

```
(define (domain Preprava)
(:requirements :numeric-fluents :typing :durative-actions
:timed-initial-literals)

(:types
     misto vec - object
     balik vozidlo - vec
     zastavka krizovatka - misto
     sklad zakaznik - zastavka)

(:predicates
     (na ?v - vec ?m - misto)
     (sousedi ?m1 ?m2 - misto)
     (v ?b - balik ?v - vozidlo)
     (dorucen ?b - balik ?z - zastavka)
     (dorucitelny ?b - balik ?z - zastavka))

(:functions

     (volna-kapacita ?v - vozidlo)
     (velikost ?b - balik)
     (trvani ?z ?do - misto)
     (spotreba ?z ?do - misto ?v - vozidlo)
     (total-cost)
     )




(:durative-action Jed
     :parameters (?z ?do - misto ?v - vozidlo)
     :duration (= ?duration (trvani ?z ?do))
     :condition
          (and  (at start (na ?v ?z))
                (over all (sousedi ?z ?do)))
     :effect
          (and  (at start (not (na ?v ?z)))
                (at end (na ?v ?do))
                (at end (increase (total-cost) (spotreba ?z ?
do ?v))))

)
```

```
(:durative-action Naloz
      :parameters (?b - balik ?v - vozidlo ?s - sklad)
      :duration (= ?duration 1)
      :condition
            (and  (over all (na ?v ?s))
                  (over all (na ?b ?s))
                  (at end (>= (volna-kapacita ?v) (velikost ?
b))))
      :effect
            (and  (at end (not (na ?b ?s)))
                  (at end (v ?b ?v))
                  (at end (decrease (volna-kapacita ?v)
                                    (velikost ?b))))
)

(:durative-action Vyloz
      :parameters (?b - balik ?v - vozidlo ?m - zastavka)
      :duration (= ?duration 1)
      :condition
            (and  (over all (na ?v ?m))
                  (at start (v ?b ?v)))
      :effect
            (and  (at end (na ?b ?m))
                  (at end (not (v ?b ?v)))
                  (at start (increase (volna-kapacita ?v)
                                      (velikost ?b))))
)




(:durative-action Doruc
      :parameters (?b - balik ?z - zastavka)
      :duration (= ?duration 1)
      :condition
            (and  (at start (na ?b ?z))
                  (over all (na ?b ?z))
                  (at end (dorucitelny ?b ?z)))
      :effect
            (and  (at end (not (na ?b ?z)))
                  (at end (dorucen ?b ?z)))
)
)
```

Příklad PDDL problému

```
(define (problem preprava_prob)
(:domain preprava)
(:objects

     misto1 - sklad
     misto2 - krizovatka
     misto3 - zakaznik
     vozidlo4 - vozidlo
     balik5 - balik
     balik6 - balik
 )
(:init

     (sousedi misto1 misto2)
     (= (trvani misto1 misto2) 70)
     (= (spotreba misto1 misto2 vozidlo4) 50)
     (na balik6 misto1)
     (na balik5 misto1)
     (sousedi misto2 misto1)
     (= (trvani misto2 misto1) 70)
     (= (spotreba misto2 misto1 vozidlo4) 50)
     (sousedi misto2 misto3)
     (= (trvani misto2 misto3) 90)
     (= (spotreba misto2 misto3 vozidlo4) 75)
     (na vozidlo4 misto2)
     (sousedi misto3 misto2)
     (= (trvani misto3 misto2) 90)
     (= (spotreba misto3 misto2 vozidlo4) 75)
     (= (volna-kapacita vozidlo4) 10)
     (= (velikost balik5) 8)
     (at 0 (dorucitelny balik5 misto3))
     (at 1000 (not (dorucitelny balik5 misto3)))
     (= (velikost balik6) 7)
     (at 0 (dorucitelny balik6 misto3))
     (at 350 (not (dorucitelny balik6 misto3)))
     (= total-cost 0)
 )
(:goal

     (and

          (dorucen balik5 misto3)
          (dorucen balik6 misto3)
 ))
(:metric minimize (total-cost))
 )
```