Paralelní plánování jako problém splňování podmínek

Roman Barták[1]

[1]Katedra teoretické informatiky a matematiké logiky
Matematicko-fyzikální fakulta, Univerzita Karlova v Praze
Malostranské náměstí 2/25, 118 00 Praha 1, Česká republika
bartak@ktiml.mff.cuni.cz

Abstrakt. Klasické plánování se zabývá hledáním plánů – posloupností akcí převádějících stav světa z počáteční situace na situaci s požadovanými vlastnostmi. Při paralelním plánování se podobně hledá posloupnost množin akcí, ze které se plán získá libovolným uspořádáním akcí v těchto množinách. Výhodou tohoto přístupu je to, že symetrické plány, ve kterých je prohozeno pořadí dvou (či více) nezávislých akcí, se nemusí prozkoumávat zvlášť, protože tyto nezávislé akce mohou být sdruženy v jedné množině. Článek představuje nový přístup k hledání paralelních plánů technikami splňování omezujících podmínek. Navržený model je založený na myšlence časových os popisujících změny hodnot stavových proměnných v čase a synchronizaci mezi těmito osami pro různé stavové proměnné. Experimentální výsledky ukázaly, že tento přístup je pro řadu plánovacích domén výrazně efektivnější než sekvenční plánování.

Klíčová slova: plánování, paralelní plány, splňování podmínek

1 Úvod

Technologie splňování omezujících podmínek je založena na deklarativním popisu řešeného problému v podobě problému splňování podmínek a následné aplikaci obecného řešícího mechanismu kombinujícího prohledávání a inferenci. Problém splňování podmínek (PSP) je dán konečnou množinou proměnných popisujících hledané vlastnosti, jako je například obsah políčka v Sudoku, každá z proměnných má přiřazenu konečnou množinu hodnot, kterých může nabývat, v případě Sudoku je to množina {1,…,9}, a nakonec je tady konečná množina podmínek omezujících kombinace hodnot, které lze do proměnných přiřadit (u Sudoku podmínky popisují různost hodnot vybraných množin proměnných). Řešením PSP je ohodnocení proměnných splňující všechny podmínky. Jak již bylo řečeno hlavní řešící technikou je kombinace prohledávání a inference. Při prohledávání se vybere proměnná, jejíž hodnota ještě není rozhodnuta, a postupně se do této proměnné zkouší přiřadit hodnoty. V případě úspěchu (inference neodhaliła konflikt) se pokračuje k další proměnné, jinak se vrátí k předchozí proměnné. Jedná se tedy o klasické prohledávání do hloubky s navracením. Základní inferenční technikou je tzv. hranová konzistence, která postupně bere všechny podmínky a zjišťuje, zda hodnoty v zahrnutých proměnných mohou podmínku splnit. Pokud ne, tak se příslušná hodnota vyřadí z množiny kandidátů pro danou proměnnou. Při řešení reálných problémů je tedy úkolem uživatele naformulovat problém v podobě PSP, tzv. modelování, a případně určit strategie pro výběr proměnných a hodnot při prohledávání.

V tomto článku popisujeme, jak modelovat řešení plánovacích problémů pomocí omezujících podmínek. Jedním z prvních takových přístupů byl systém CPlan [15], kde se model konstruoval ručně pro každý plánovací problém. Tento přístup je poměrně efektivní, jak ukazuje například nový model pro plánovací problém Settlers [7], ale modely se nutně liší problém od problému. Zde se proto soustředíme na obecné modely, které lze plně automaticky generovat z popisu plánovacího problému. První odlišností plánovacích problémů od PSP je, že počet akcí v plánu není dopředu znám zatímco proměnné v PSP je potřeba specifikovat předem. Řešení tohoto koncepčního problému navrhli Kautz a Selman [10] pro kódování plánovacích problémů jako problém splnitelnosti logické formule (SAT) a je založeno na tom, že model se vždy definuje jen pro hledání plánu dané délky. V případě neúspěchu se délka hledaného plánu zvětší, vygeneruje se další model atd. Podobný přístup používají plánovače založené na plánovacím grafu [2], který se stal základem prvních obecných PSP modelů. Do a Kambhampati navrhli model GP-CSP [3], který kóduje fázi extrakce plánu tak, že pro každý plánovací výrok hledá akci (PSP proměnná), která tento výrok učinní pravdivým. Lopez a Bacchus navrhli model CSP-PLAN [12] používající Boolean proměnné, které určují, zda se daná akce v daném kroku plánu použije, a tyto proměnné propojili podmínkami následující stavu. Oba modely byly určeny pro hledání paralelních plánů v klasické plánovací reprezentaci, tj. plán je posloupnost množin akcí taková, že akce v jedné množině jsou nezávislé a lze je tedy aplikovat v libovolném pořadí. V [1] jsme popsali, jak přeformulovat tyto modely pro sekvenční plány v reprezentaci s více-hodnotovými proměnnými [8], které jsou podle nás vhodnější pro PSP modely. Sekvenční plány ale mají nevýhodu tzv. permutačních symetrií [11], tj. při plánování je potřeba prozkoumávat plány, kde jsou nezávislé akce organizovány v různých pořadích, což zvětšuje prohledávaný prostor. To lze částečně odstranit přidáním podmínek pro rozbití symetrií, jak bylo ukázáno v [1,5], což ale zvyšuje náklady na inferenci. Jiným přístupem k odstranění symetrií je použití částečně uspořádaných plánů jako v systému CPT [16], který má ale jistá omezení na obecnost hledaných plánů. Některé z myšlenek částečně uspořádaných plánů jsme integrovali do systému SeP [1], což zvýšilo efektivitu zvláště pro těžší plánovací problémy. Toto kódování poskytlo základní ideje pro model paralelních plánů s vícehodnotovými proměnnými, který představíme v tomto článku.

Základní myšlenka navrhovaného plánovače je založena na modelování plánu jako evoluce hodnot stavových proměnných podobně jako u tzv. časových os [14]. To nám umožní přirozeně modelovat paralelní plány s vícehodnotovými stavovými proměnnými. Právě paralelní plány jsou jedním z pilířů efektivních SAT plánovačů tradičně pracujících s výrokovou reprezentací stavů světa. Tyto plánovače v poslední době dále zvýšily svoji efektivitu přechodem na více-hodnotové stavové proměnné [9], což bylo impulsem pro vývoj nového paralelního plánovače na principu splňování podmínek. V článku nejprve přesněji formulujeme řešený problém hledání paralelních plánů. Následně vysvětlíme základní koncept nového plánovače PaP (Parallel Planning), který využívá popisu evoluce hodnoty stavové proměnné pomocí konečného automatu. Po té představíme kódování jednotlivých konečných automatů a jejich synchronizace do PSP a popíšeme použitou prohledávací strategii. V závěru experimentálně porovnáme nový plánovač s moderními PSP plánovači SeP [1] a Constance [6] a ukážeme, že přes jednoduchost PSP modelu dokáže systém PaP tyto plánovače v řadě problémů výrazně překonat, pokud jde o efektivitu řešení.

2 Problém paralelního plánování

Klasické plánování se zabývá hledáním posloupnosti akcí, které převádí svět ze známého počátečního stavu do stavu splňujícího zadanou cílovou podmínku. Na rozdíl od klasické repezentace, kde jsou stavy světa popsány množinou výroků, které v daném stavu platí, používáme kompaktnější reprezentaci s více-hodnotovými stavovými proměnnými [8]. Její základní výhodou je, že přirozeně zachycuje vzájemně výlučné výroky, například, že robot se v daném čase může nacházet pouze v jedné lokaci. V této reprezentaci je tedy stav světa jednoznačně popsán hodnotami stavových proměnných, které se vybírají z konečné množiny možných hodnot pro každou stavovou proměnnou. Předpokládáme, že akce jsou instantní (nemají dobu trvání) a jsou popsány množinou požadavků na hodnoty stavových proměnných, tzv. předpoklady, a množinou požadovaných změn hodnot stavových proměnných, tzv. efekty. Formálně je předpoklad množina (konjunkce) výroků tvaru *stavová_proměnná = hodnota* nebo *stavová_proměnná* ∈ *množina_hodnot* a efekt je množina (konjunkce) výroků tvaru *stavová_proměnná* ← *hodnota*. Akci lze aplikovat na daný stav, pokud jsou splněny předpoklady (stavové proměnné mají požadované hodnoty) a nový stav se získá aplikací efektu, tj. nastavením hodnot příslušných stavových proměnných. Stavové proměnné, které se v efektu nevyskytují, zůstanou beze změny (tzv. axiom rámce). Příklad takové reprezentace je na Obr. 1.

Stavové proměnné

rloc ∈ {loc1,loc2} ;; lokace robota
cpos ∈ {loc1,loc2,r} ;; pozice kontejneru

Akce

1 : move(r, loc1, loc2) ;; robot r na lokaci loc1 se přesune do lokace loc2
Precond: *rloc* = loc1
Effects: *rloc* ← loc2

2 : move(r, loc2, loc1) ;; robot r na lokaci loc2 se přesune do lokace loc1
Precond: *rloc* = loc2
Effects: *rloc* ← loc1

3 : load(r, c, loc1) ;; robot r naloží kontejner c na lokaci loc1
Precond: *rloc* = loc1, *cpos* = loc1
Effects: *cpos* ← r

4 : load(r, c, loc2) ;; robot r naloží kontejner c na lokaci loc2
Precond: *rloc* = loc2, *cpos* = loc2
Effects: *cpos* ← r

5 : unload(r, c, loc1) ;; robot r vyloží kontejner c na lokaci loc1
Precond: *rloc* = loc1, *cpos* = r
Effects: *cpos* ← loc1

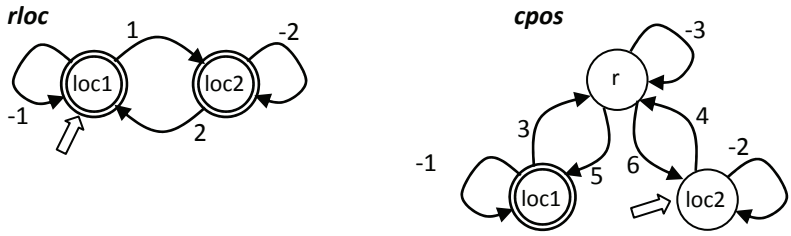
6 : unload(r, c, loc2) ;; robot r vyloží kontejner c na lokaci loc2
Precond: *rloc* = loc2, *cpos* = r
Effects: *cpos* ← loc2

Obr. 1. Příklad plánovací domény reprezentované pomocí vícehodnotových stavových proměnných; máme zde dvě lokace (loc1,loc2), robota (r) schopného cestovat mezi lokacemi a nakládat a vykládat kontejnery a jeden kontejner (c).

Plánovací problém je daný úplným popisem počátečního stavu, tj. hodnotami všech stavových proměnných, popisem vlastností cílového stavu ve stejném formátu, jako předpoklady akcí a množinou dostupných akcí. Úkolem je nalézt posloupnost akcí vedoucí z počátečního stavu do některého ze stavů splňujících cílovou podmínku. Při *paralelním plánování* se hledá posloupnost množin nezávislých akcí, ze které se plán získá libovolným uspořádáním akcí v těchto množinách. Akce jsou nezávislé, pokud je lze na daný stav aplikovat v libovolném pořadí a vždy se získá stejný stav. Toto lze pro dvojici akcí zaručit, pokud se stavová proměnná z efektu jedné akce nevyskytuje v předpokladech ani efektech druhé akce. V množině nezávislých akcí musí být všechny akce po dvojicích nezávislé. To nám zaručuje, že pokud jsou všechny akce z množiny aplikovatelné na daný stav (jsou splněny jejich předpoklady), potom tyto akce můžeme aplikovat v libovolném pořadí, protože aplikací akce se neporuší předpoklady následujících akcí ani efekty předchozích akcí v posloupnosti. Můžeme si představit, že akce z množiny jsou aplikovány najednou. Při plánování tedy stačí prozkoumávat množiny nezávislých akcí aplikovatelných na daný stav, kde se následující stav získá paralelní aplikací všech těchto akcí.

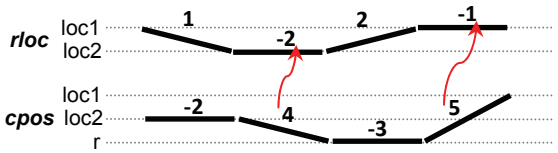
3 Základní koncept popisu plánu

Přirozeným způsobem, jak reprezentovat evoluci hodnoty více-hodnotové proměnné po aplikaci akcí, je popis konečným automatem (KA). Krátce řečeno, stavy automatu odpovídají hodnotám stavové proměnné a hrany popisují, jak akce tyto hodnoty mění. Hrana označená akcí A vede ze stavu P do Q, pokud A nastavuje v efektu proměnnou na hodnotu Q a buď má P mezi předpoklady nebo nemá danou stavovou proměnou vůbec mezi předpoklady a potom P může být libovolná hodnota (stav KA). Aby bylo možné popisovat nejen efekt akce, ale také její předpoklady, zavádíme do automatu ještě hrany se speciálními tzv. no-op akcemi. Jedná se vždy o smyčky u každého vrcholu, které říkají, že hodnota se nemění. Rozdíl od no-op akcí známých např. ze systému Graphplan [2] je v tom, že každá hodnota má svoji speciální no-op akci různou od no-op akce pro jinou hodnotu. Na Obr. 2 jsou tyto akce značeny zápornými čísly. V konečném automatu lze kromě hodnot stavové proměnné a akcí přirozeně modelovat také počáteční stav (počáteční hodnota) a cílový stav (cílové hodnoty). Na Obr. 2 popisujeme situaci, kdy je robot na lokaci *loc1*, kontejner na lokaci *loc2* a cílem je mít kontejner na lokaci *loc1* (robot tedy může být kdekoliv).



Obr. 2. Reprezentace plánovací domény z obrázku 1 pomocí konečných automatů. Paralelní plánování odpovídá synchronizovanému průchodu oběma automaty. Akce 3 musí být například synchronizována s no-op akcí -1 v KA pro *rloc*, aby byla splněna podmínka *rloc* = loc1.

Při řešení plánovacího problému budeme synchronizovaně procházet všemi automaty, což byl důvod pro zavedení no-op akcí. Předpokládejme, že robot nakládá kontejner na lokaci *loc1*, tj. mění se hodnota stavové proměnné *cpos*. V KA pro *cpos* na Obr. 2 je to reprezentováno hranou s označením 3 (číslo akce z Obr. 1). Příslušná akce zároveň vyžaduje, aby také robot byl v lokaci *loc1*, což lze právě reprezentovat no-op akcí -1 v KA pro *rloc*. Jinými slovy, pokud v automatu pro *cpos* přejdeme po hraně 3, musíme v automatu pro *rloc* přejít pro hraně -1. Obecně řečeno, každá akce vyžaduje přesun po příslušné hraně v automatech, které reprezentují stavové proměnné z předpokladů a efektů akcí. Všimněme si dále, že pokud jsou akce nezávislé, můžeme přesun pro tyto akce udělat v příslušných automatech paralelně, protože každá akce pracuje s jinými automaty (vlastnost nezávislosti). Jedinou vyjímkou jsou dvě nezávislé akce sdílející proměnnou ve svých předpokladech. Tyto akce mohou být použity paralelně, pokud je jejich sdílený předpoklad stejný (přesněji průnik hodnot z předpokladů je neprázdný). Obr. 3 ukazuje synchronizovaný vývoj hodnot stavových proměnných (v tomto případě nemáme paralelní akce). Všimněme si, že v každém kroku je potřeba v každém automatu přejít po nějaké hraně a tyto přechody musí být mezi automaty synchronizovány. V následující kapitole popíšeme PSP model, kde přechody automatu a synchronizace přechodů jsou popsány podmínkami.



Obr. 3. Časové osy zobrazující vývoj hodnot jednotlivých stavových proměnných včetně no-op akcí (synchronizace je naznačena šipkami).

4 PSP model paralelního plánu

Jak již bylo řečeno v úvodu, tradičním způsobem jak reprezentovat hledání plánu předem neznámé délky pomocí PSP či SAT, je kódovat hledání plánu pevně dané délky n počínaje $n = 0$ a v případě, že plán nebyl nalezen, n postupně zvětšovat (v našem případě o 1). V této sekci se tedy dále soustředíme jen na PSP model popisující hledání paralelního plánu délky n. Připomeňme, že to znamená hledání n množin akcí takových, že v každé množině jsou akce navzájem nezávislé a tyto akce jsou aplikovatelné na stav předcházející dané množině. Výsledný plán tedy může mít více než n akcí za předpokladu, že se paralelismus akcí skutečně využije.

Nechť k je počet stavových proměnných v popisu plánovacího problému a n je délka hledaného paralelního plánu. Potom pro reprezentaci stavů použijeme $k(n+1)$ stavových proměnných S^i_j popisujících stavy "navštívené" paralelním plánem ($i = 1,\ldots,k$, $j = 0,\ldots,n$). Proměnná S^i_j může nabývat hodnoty odpovídající hodnotám i-té stavové proměnné. Hodnoty proměnných S^i_0 jsou dány popisem počátečního stavu a množiny hodnot pro S^i_n jsou omezeny cílovou podmínkou. Dále použijeme kn akčních proměnných A^i_j popisujících akce měnící příslušné stavové proměnné. Možnými hodnotami akční proměnné A^i_j jsou identifikátory akcí majících i-tou

stavovou proměnnou jako svůj efekt a no-op akce pro tutéž stavovou proměnnou. Například pro akční proměnnou A^{cpos}_j se jedná o množinu hodnot {-3,-2,-1,3,4,5,6}, viz Obr. 2.

V modelu zavádíme dva základní typy omezujících podmínek, podmínky popisující stavové přechody, tzv. *sekvencování*, a podmínky popisující *synchronizaci* mezi stavovými přechody různých automatů. Pro popis stavových přechodů KA existuje v mnoha systémech pro řešení podmínek vestavěná podmínka *regular* [13] říkající, že hodnoty v posloupnosti proměnných definují slovo přijímané zadaným konečným automatem. My jsme se rozhodli pro popis téhož omezení použít sadu podmínek definujících jednotlivé přechody. Takový model nesnižuje úroveň inference, ale navíc dává přímý přístup ke stavovým proměnným, které podmínka *regular* skrývá. Používáme tedy ternární *sekvencovací podmínku* definovanou nad všemi trojicemi proměnných S^i_{j-1}, A^i_j, S^i_j, která krátce řečeno popisuje hrany v automatu pro i-tou stavovou proměnnou. Přesněji řečeno, trojice hodnot (P,A,Q) splňuje podmínku pro i-tou stavovou proměnnou, pokud platí jedno z následujících tvrzení:

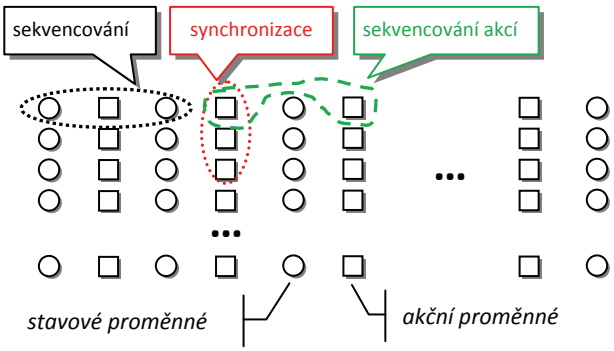
- A je skutečná akce (ne no-op) taková, že Q je hodnotou jejího efektu pro i-tou stavovou proměnnou a P je hodnota kompatibilní s jejími předpoklady (A používá P mezi svými předpoklady na i-tou stavovou proměnnou nebo A nemá i-tou stavovou proměnnou vůbec mezi předpoklady a potom může být P libovolná hodnota),
- A je no-op akce odpovídající hodnotě P i-té stavové proměnné a P = Q.

V řeči konečných automatů můžeme říci, že A označuje hranu mezi stavy P a Q automatu. Sekvencovací podmínka tedy definuje korektní evoluci hodnot stavové proměnné neboli přechodovou funkci konečného automatu.

Druhý typ podmínek popisuje synchronizaci mezi evolucemi jednotlivých automatů, jak bylo naznačeno v předchozí kapitole. Potřebujeme totiž zajistit, že pokud akce mění několik stavových proměnných, potom musí být tato akce přiřazena do akčních proměnných pro všechny tyto stavové proměnné v dané vrstvě (vrstvou nazýváme všechny akční proměnné A^i_j s identickým indexem j). Dále potřebujeme zaručit, že pokud akce používá nějakou stavovou proměnnou jako předpoklad (ale ne efekt), potom musí mít příslušná stavová proměnná hodnotu odpovídající předpokladu, což zajistíme tak, že odpovídající akční proměnná ve stejné vrstvě má přiřazenu hodnotu odpovídající no-op akce. Při synchronizaci tedy svazujeme pouze akční proměnné v dané vrstvě a nemusíme zahrnovat stavové proměnné. V principu bychom mohli synchronizaci popsat jednou *k*-ární podmínkou nad proměnnými A^1_j,..., A^k_j, ale extenzionální reprezentace takové podmínky by byla příliš velká. Navrhli jsme proto použití *k k*-árních *synchronizačních podmínek*, z nichž každá popisuje synchronizační požadavky akcí použitých v jednom stavovém přechodu. Nechť *i* je index stavové proměnné, potom *i*-tá synchronizační podmínka popisuje pro každou akci z *i*-té akční proměnné jaké akce z ostatních akčních proměnných jsou s ní kompatibilní. Je-li A reálná (ne no-op) akce přiřazená do proměnné A^i_j, potom podmínka vyžaduje následující přiřazení do ostatních akčních proměnných:

- pokud má A efekt v p-té stavové proměnné potom $A^p_j = A$,
- pokud má A předpoklad (ale ne efekt) v p-té stavové proměnné a B je no-op akce odpovídající hodnotě tohoto předpokladu, potom $A^p_j = B$.

Pokud je A no-op akce, tak podmínka nijak neomezuje akce v jiných proměnných, abychom udrželi extenzionální reprezentaci podmínky kompaktní. Zde je třeba si uvědomit, že synchronizační podmínky pro další akční proměnné mohou tuto no-op akci svázat s jinými reálnými akcemi, proto tento vztah nemusí být popsán v aktuální podmínce. Na tomto místě je vhodné ještě zmínit, jak lze tyto podmínky realizovat v praxi. Pro jednoduchost výkladu jsme všechny podmínky definovali jako *k*-ární, ale v praxi každá podmínka omezuje jen několik dalších akčních proměnných. Důvodem je to, že akce zpravidla používají jen malou podmnožinu stavových proměnných ve svých předpokladem a efektech. Akční proměnné, které nejsou podmínkou nijak omezeny, tedy můžeme z podmínky vyřadit, snížit tak její aritu a prakticky zefektivnit inferenci. Obr. 4 ukazuje, jak vypadá základní struktura PSP modelu.



Obr. 4. Základní struktura modelu s podmínkami, kolečka reprezentují stavové proměnné, čtverečky akční proměnné.

Sekvenční a synchronizační podmínky definují základní PSP model. Pokud řešící systém najde řešení splňující tyto podminky, lze z něho snadno zkonstruovat paralelní plán tak, že reálné akce ohodnocující akční proměnné v jedné vrstvě tvoří příslušnou množinu nezávislých akcí. Nezávislost akcí je zajištěna splněním synchronizačních podmínek, které nedovolí, aby dvě akce ovlivňovaly stejné stavové proměnné a aby jedna akce měla nějakou stavovou proměnnou jako efekt zatímco jiná jako předpoklad (to jsou přesně dvě části definice synchronizační podmínky).

Základní PSP modely definující problém bývají často doplněny o redundantní podmínky, jejichž cílem je posílit inferenci založenou na hranové konzistenci. V navrženém modelu jsme podobně odhalili slabší inferenci v sekvencování aktivit, která je dána tím, že sekvence jsou definovány po jednotlivých stavových proměnných. Konkrétně akce B může přímo následovat akci A v časové ose nějaké stavové proměnné, pokud je efekt A v této stavové proměnné kompatibilní s předpokladem B v téže stavové proměnné. Může se ale stát, že v jiné stavové proměnné tato kompatibilita neplatí a A nemůže být před B (v žádné časové ose). Tato inference v sobě vlastně kombinuje několik sekvencovacích podmínek a synchronizační podmínku, systém ale případnou nekompatibilitu odhalí až při ohodnocování proměnných. Z tohoto důvodu jsme do modelu přidali ještě sekvencovací podmínku pro každou dvojici přímo následujících akčních proměnných v každé časové ose. Nechť A a B jsou dvě akce z KA popisujícího změny nějaké

stavové proměnné. Potom dvojice (A,B) splňuje příslušnou *akční sekvencovací podmínku*, právě když nastává některý z následujících případů:

- A a B jsou identické no-op akce,
- A je no-op akce kompatibilní s předpokladem reálné akce B,
- B je no-op akce odpovídající efektu reálné akce A,
- všechny efekty reálné akce A jsou kompatibilní se všemi předpoklady reálné akce B.

Obr. 5 ukazuje, jak vypadá extenzionální reprezentace všech podmínek v PSP modelu pro příklad z Obr. 2. Pro úplnost je potřeba říci, že v tomto jednoduchém problému nejsou podmínky pro akční sekvencování potřeba (nezlepšují inferenci) a že paralelní plány obsahují v každé vrstvě právě jednu reálnou akci. Abychom neměli vrstvy obsahující pouze no-op akce, stačí přidat podmínku zajišťující, že v každé vrstvě je alespoň jedna reálná akce. Čtenář snadno nahlédne, že plán představený v Obr. 3 splňuje všechny definované podmínky a je tedy korektním paralelním plánem. Za zmínku možná ještě stojí pozorování, že v každé vrstvě může být maximálně *k* různých akcí, což přímo plyne z požadavku nezávislosti akcí. Řada těchto akcí ale budou no-op akce, které se ve výsledném plánu neobjeví (zajišťují splnění předpokladů reálných akcí a axiom rámce).

sekvencování

S^{rloc}_{i-1}	A^{rloc}_{i}	S^{rloc}_{i}
loc1	1	loc2
loc2	2	loc1
loc1	-1	loc1
loc2	-2	loc2

sekvencování akcí

A^{rloc}_{i}	A^{rloc}_{i+1}
1	{-2,2}
2	{-1,1}
-1	{-1,1}
-2	{-2,2}

S^{cpos}_{i-1}	A^{cpos}_{i}	S^{cpos}_{i}
loc1	3	r
loc2	4	r
r	5	loc1
r	6	loc2
loc1	-1	loc1
loc2	-2	loc2
r	-3	r

A^{cpos}_{i}	A^{cpos}_{i+1}
3	{-3,5,6}
4	{-3,5,6}
5	{-1,3}
6	{-2,4}
-1	{-1,3}
-2	{-2,4}
-3	{-3,5,6}

synchronizace

A^{rloc}_{i}	A^{cpos}_{i}
1	{-1,-2,-3}
2	{-1,-2,-3}
{-2,-1}	{-3,…,6}

A^{cpos}_{i}	A^{rloc}_{i}
3	-1
4	-2
5	-1
6	-2
{-3,-2,-1}	{-2,…,2}

Obr. 5. Kompaktní reprezentace ad-hoc podmínek pro plánovací problém z obrázku 2.

4.1 Prohledávací strategie

V úvodu bylo zmíněno, že použití technologie splňování podmínek obsahuje konstrukci PSP modelu, ten definuje, jak probíhá inference, a popis prohledávací strategie, tj. v jakém pořadí se mají ohodnocovat proměnné a v jakém pořadí se mají zkoušet hodnoty. Na začátek je dobré zmínit, že v našem modelu stačí ohodnocovat pouze akční proměnné, protože hodnoty stavových proměnných se nastaví automaticky pomocí inference.

Pro *výběr proměnné* jsme použili mírně modifikovanou verzi osvědčené heuristiky známé pod názvem *dom* [4]. Tato heuristika vybírá pro ohodnocení proměnnou s nejmenším počtem hodnot pro přiřazení a je typickým reprezentantem principu prvního neúspěchu (doporučuje se nejdříve ohodnocovat proměnné, jejichž ohodnocení pravděpodobně neuspěje, aby se to odhalilo při prohledávání dříve). Heuristiku jsme modifikovali tak, že výběr provádíme pouze mezi proměnnými, do kterých lze ještě přiřadit reálnou akci. Proměnné, jejichž aktuální množiny možných hodnot obsahují pouze no-op akce, při prohledávání ignorujeme, protože jejich hodnota se podobně jako u stavových proměnných nastaví inferencí. Tímto způsobem jsme přirozeně zmenšili prohledávaný prostor.

Pro *výběr hodnoty* pro proměnnou se používá princip prvního úspěchu (nejprve se zkouší hodnota, která má největší šanci být v řešení). Na rozdíl od výběru proměnné zde ale neexistují obecně aplikovatelné a široce používané heuristiky realizující zmíněný princip a zpravidla se používá problémově závislá heuristika. Pro náš model jsme navrhli následující způsob větvení. Nejprve se množina možných hodnot pro akční proměnnou rozdělí na no-op akce a reálné akce a jako první se prozkoumá větev s no-op akcemi. Nejedná se sice o klasický výběr hodnoty pro proměnnou, ale o obecnější metodu dekompozice problému. Motivací za tímto větvením byla snaha o minimalizaci počtu reálných akcí v plánu (proto se nejprve zkouší no-op akce). Pokud k ohodnocení přijde proměnná, do které lze přiřadit už pouze reálné akce, potom se akce zkouší v pořadí, jak byly definovány na vstupu problému. Navržený princip větvení znamená, že stejná akční proměnná se typicky objeví ve dvou větvících uzlech prohledávacího stromu.

5 Experimentální výsledky

Navržený model jsme implementovali pomocí *clpfd* knihovny SICStus Prolog 4.1.2 a přímo ho porovnali s plánovačem SeP [1] používajícím stejnou knihovnu. Pro porovnání jsme použili standardní sady testů z mezinárodních soutěží IPC. STRIPS verze problémů jsme nejprve převedli do reprezentace s vícehodnotovými stavovými proměnnými pomoci programu Malte Helmerta (některé problémy v doménách openstack a airport ale nebyl schopen přeložit), čas převodu není ve výsledcích započítán. Měření jsme provedli na počítači vybaveném procesorem Intel Xeon CPU E5335 2.0 GHz s 8GB RAM pod systémem Ubuntu Linux 8.04.2 (Hardy Heron). Každý plánovač měl na řešení každého problému 30 minut, měřený čas zahrnoval jak vytvoření PSP reprezentace problému, tak vlastní hledání plánu. Na úvod je potřeba ještě zdůraznit, že oba plánovače řeší mírně odlišný problém. SeP hledá nejkratší

sekvenční plán, zatímco PaP nejkratší paralelní plán, jehož sekvenční verze nemusí být nutně nejkratší. Jak ale ukazují výsledky, délka nalezených sekvenčních plánů se příliš neliší od optima nalezeného systémem SeP. Tabulka 1 ukazuje počty vyřešených problémů a jednoznačně ukazuje, že PaP je pro většinu typů problémů efektivnější pokud jde o schopnost nalézt plán. Tabulka 2 potom ukazuje podrobnější pohled na jednotlivé řešené problémy ve vybraných plánovacích doménách, konkrétně délky plánu (v případě PaP počet vrstev, tj. délka paralelního plánu, i počet akcí, tj. délka sekvenčního plánu) a běhové časy. Je vidět, že PaP je o několik řádů výkonnější než SeP. Z časových důvodů jsme neprováděli přímé porovnání se systémem Constance, ale podle výsledků prezentovaných v [6], které ukazovaly, že Constance je lepší než SeP v doménách driverlog, zenotravel a tpp, lze jednoznačně říci, že v těchto doménách dosahuje PaP lepšího výkonu než Constance.

Tabulka 1. Počet vyřešených problémů v každé plánovací doméně (číslo v závorce je celkový počet problémů v doméně).

domain	***SeP***	***PaP***
airport (15)	4	**6**
blocks (16)	**7**	**7**
depots (10)	**2**	**2**
driverlog (15)	4	**12**
elevator (30)	**30**	27
freecell (10)	1	**3**
openstacks (7)	**5**	0
rovers (10)	4	**6**
tpp (15)	4	**8**
zenotravel (15)	6	**11**

6 Závěr

Článek prezentuje nový model pro paralelní plánování založený na splňování omezujících podmínek. Představený model je překvapivě jednoduchý, používá pouze dva základní typy podmínek a další typ podmínky pro posílení inference. Také prohledávací strategie je založena na tradičních technikách splňování podmínek. Přes svoji jednoduchost nový systém PaP v efektivitě poráží současné moderní plánovače SeP a Constance založené také na splňování podmínek. V rámci objektivity je ale potřeba říci, že tyto plánovače hledají nejkratší možné sekvenční plány, zatímco PaP hledá nejkratší paralelní plány, jejichž sekvenční podoba může být delší. Na druhou stranu v systému PaP je ještě hodně prostoru pro zahrnutí pokročilejších plánovacích technik a heuristik, které již používají plánovače SeP a Constance, a lze tedy ještě očekávat nárůst výkonu především pro typy problémů, kde zatím PaP zaostává.

Poděkování. Výzkum je podporován Grantovou agenturou České republiky v rámci projektu P103/10/1287. Autor by rád poděkoval Danielovi Toropilovi za provedení experimentů a Malte Helmertovi za poskytnutí převaděče reprezentací.

Tabulka 2. Délka nalezených plánů (sekvenčních i paralelních) a doba běhu (v milisekundách) pro vybrané problémy.

problém	*délka plánu*			*runtime (ms)*	
	SeP	*PaP*		*SeP*	*PaP*
		par	*seq*		
airport-p03	17	9	17	8780	**810**
airport-p06	41	21	41	1736 250	**13 670**
airport-p07	≥38	21	41	-	**13 720**
airport-p12	39	21	39	1459 030	**19 560**
airport-p13	37	21	37	1548 350	**20 440**
airport-p15	≥31	22	58	-	**104 720**
driverlog-p01	7	6	8	110	**40**
driverlog-p02	19	9	21	1017 570	**100**
driverlog-p03	12	7	12	11 650	**110**
driverlog-p04	≥14	7	18	-	**250**
driverlog-p05	≥13	8	21	-	**190**
driverlog-p06	11	5	11	83 940	**160**
driverlog-p07	≥11	6	18	-	**190**
driverlog-p08	≥13	7	25	-	**2 690**
driverlog-p09	≥16	10	24	-	**12 680**
driverlog-p10	≥12	7	20	-	**446 580**
driverlog-p11	≥13	9	21	-	**13 130**
rovers-p01	10	5	10	940	**60**
rovers-p02	8	4	8	370	**20**
rovers-p03	11	7	12	2 190	**120**
rovers-p04	8	4	8	440	**60**
rovers-p05	≥13	5	22	-	**150**
rovers-p06	≥15	≥8		-	-
rovers-p07	≥11	5	18	-	**120**
tpp-p01	5	5	5	10	**0**
tpp-p02	8	5	8	20	**10**
tpp-p03	11	5	11	160	**30**
tpp-p04	14	5	14	2 110	**20**
tpp-p05	≥17	7	23	-	**100**
tpp-p06	≥15	9	29	-	**4 110**
tpp-p07	≥14	9	38	-	**3 170**
tpp-p08	≥14	9	44	-	**5 930**
zenotravel-p01	1	1	1	**10**	20
zenotravel-p02	6	5	6	60	**50**
zenotravel-p03	6	5	9	300	**130**
zenotravel-p04	8	5	11	970	**130**
zenotravel-p05	11	5	14	153 990	**240**
zenotravel-p06	11	5	12	530 390	**510**
zenotravel-p07	≥12	6	16	-	**560**
zenotravel-p08	≥10	5	15	-	**1 690**
zenotravel-p09	≥11	6	24	-	**145 760**
zenotravel-p10	≥12	6	24	-	**252 040**
zenotravel-p11	≥9	6	16	-	**41 780**

Reference

[1] Barták, R., Toropila, D. Solving Sequential Planning Problems via Constraint Satisfaction. *Fundamenta Informaticae*, Volume 99, Number 2, IOS Press, 125-145, 2010.

[2] Blum, A. and Furst, M. Fast planning through planning graph analysis. *Artificial Intelligence* 90, 281-300, 1997.

[3] Do, M.B. and Kambhampati, S. Solving planning-graph by compiling it into CSP. *Proceedings of the Fifth International Conference on Artificial Planning and Scheduling* (AIPS-2000), AAAI Press, 82-91, 2000.

[4] Golomb, S., Baumert, L. Backtrack programming. *Journal of the ACM* 12, 516-524, 1965.

[5] Grandcolas, S., Pain-Barre, C. Filtering, Decomposition and Search Space Reduction for Optimal Sequential Planning. *Proceedings of AAAI-2007*, 993-998, 2007.

[6] Gregory, P., Long, F., Fox. M. Constraint Based Planning with Composable Substate Graphs. *Proceedings of 19th European Conference on Artificial Intelligence* (ECAI), IOS Press, 2010.

[7] Gregory, P., Rendl, A. A Constraint Model for the Settlers Planning Domain. *Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group* (PlanSIG), Heriot-Watt University, 2008.

[8] Helmert, M. The Fast Downward Planning System. *Journal of Artificial Intelligence Research 26*, 191-246, 2006.

[9] Huang, R., Chen, Y., Zhang, W. A Novel Transition Based Encoding Scheme for Planning as Satisfiability. *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligenc*e (AAAI-10), AAAI Press, 89-94, 2010.

[10] Kautz, H. and Selman, B. Planning as satisfiability. *Proceedings of ECAI*, 359-363, 1992.

[11] Long, D., Fox, M., Plan Permutation Symmetries as a Source of Planner Inefficiency, *Proceedings of Workshop of the UK Planning and Scheduling Special Interest Group* (PlanSIG), 2003.

[12] Lopez, A. and Bacchus, F. Generalizing GraphPlan by Formulating Planning as a CSP. *Proceedings of IJCAI*, 954-960, 2003.

[13] Pesant, G. A Regular Language Membership Constraint for Finite Sequences of Variables. *Principles and Practice of Constraint Programming*, LNCS 3285, Springer, 482-495, 2004.

[14] Pralet, C., Verfaillie, G. Forward Constraint-Based Algorithms for Anytime Planning. *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling* (ICAPS), AAAI Press, 265-272, 2009.

[15] van Beek, P. and Chen, X. CPlan: A Constraint Programming Approach to Planning. *Proceedings of AAAI-99*, 585-590, 1999.

[16] Vidal, V. and Geffner, H. Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. *Proceedings of AAAI-04*, 570-577, 2004.

Annotation:

A Novel Constraint Model for Parallel Planning

Parallel plan is a sequence of sets of actions such that any ordering of actions in the sets gives a traditional sequential plan. Parallel planning was popularized by the Graphplan algorithm and it is one of the key components of successful SAT-based planners. Recently, SAT-based planners started to exploit the multi-valued state variables – area which seems traditionally more suited for constraint-based planners – and they improved their performance further. In this paper we propose a novel view of constraint-based planning that uses parallel plans and multi-valued state variables. Rather than starting with the planning graph structure like other parallel planners, this novel approach is based on the idea of timelines and their synchronization.