České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

Efektivní identifikace redundantních akcí v plánech

Efficient Identification of Redundant Actions in Plans

BAKALÁŘSKÁ PRÁCE

Studijní program:	Otevřená informatika
Specializace:	Základy umělé inteligence a počítačových věd
Vypracoval:	Jakub Med
Vedoucí práce:	RNDr. Lukáš Chrpa, Ph.D.
Rok:	2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Med** Jméno: **Jakub** Osobní číslo: **483428**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Efektivní identifikace redundantních akcí v plánech

Název bakalářské práce anglicky:

Efficient Identification of Redundant Actions in Plans

Pokyny pro vypracování:

Automatické plánování se zabývá problémem hledání posloupnosti akcí transformující prostředí z daného počátečního stavu do nějakého cílového stavu. Mnoho plánovacích systémů generuje korektní, ale suboptimální plány. Tyto systémy generují plány v malém čase (což může být nutné), ale často za cenu velké suboptimality. Úkolem je navrhnout techniky, které plány optimalizují pomocí detekce a odstranění redundantních akcí z plánů (redundantní akce lze odstranit z plánu bez porušení jeho validity), konkrétněji
- integrace a zobecnění technik Inverse Action Elimination do obecnějších technik Action Elimination a Greedy Action Elimination,
- identifikace pseudo-landmark akcí, které nelze z plánu odstranit, a jejich využití v technikách (Greedy) Action Elimination,
- vyhodnocení navržených a implementovaných technik na množině benchmarků z International Planning Competition na plánech generovaných několika existujícími (suboptimálními) plánovači.

Seznam doporučené literatury:

[1] Tomás Balyo, Lukás Chrpa, Asma Kilani: On Different Strategies for Eliminating Redundant Actions from Plans. SOCS 2014
[2] Lukás Chrpa, Thomas Leo McCluskey, Hugh Osborne: Optimizing Plans through Analysis of Action Dependencies and Independencies. ICAPS 2012
[3] Hootan Nakhost, Martin Müller: Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. ICAPS 2010: 121-128

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Lukáš Chrpa, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **04.01.2021** Termín odevzdání bakalářské práce: **21.05.2021**

Platnost zadání bakalářské práce: **30.09.2022**

RNDr. Lukáš Chrpa, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 19. května 2021 .....

Jakub Med

Poděkování

Děkuji RNDr. Lukáši Chrpovi, Ph.D. za vedení mé bakalářské práce, výbornou komunikaci a trpělivost v pandemické době a za rady a návrhy, jenž mou práci významně obohatily.

Jakub Med

Název práce:
Efektivní identifikace redundantních akcí v plánech

Autor: Jakub Med

Studijní program: Otevřená informatika
Obor: Základy umělé inteligence a počítačových věd
Druh práce: bakalářská práce

Vedoucí práce: RNDr. Lukáš Chrpa, Ph.D.
Elektrotechnická fakulta Českého vysokého učení technického v Praze, Katedra počítačů

Abstrakt: Při hledání plánů je často velmi složité najít ten optimální. Najít jakýkoli plán bývá mnohem snazší. Takto vzniklé plány mohou být často snadno vylepšeny a zkráceny. Jedním ze způsobů optimalizace plánu je identifikování zbytečných akcí a jejich následné odstranění. V této práci se zaměříme na implementaci několika metod, kterými toho lze dosáhnout. Představíme dvě optimalizace, jimiž lze zkoumané metody zrychlit. Navrhneme novou metodu, jenž bude inspirovaná původními metodami a námi vyvinutými optimalizacemi. Na závěr všechny metody porovnáme na problémech používaných na International Planning Competition a vzneseme podněty pro další výzkum.
Klíčová slova: klasické plánování, optimalizace, redundantní akce, inverzní cykly, lokální landmarky

Title:
Efficient Identification of Redundant Actions in Plans

Author: Jakub Med

Abstract: Finding an optimal plan can be very difficult. However, finding any plan is often much easier. Since these plans are suboptimal, they may be optimized and shortened. One way to achieve this is to identify and remove redundant actions. This work focuses on methods, which were designed to perform such optimization. We will introduce two approaches, which will speed up the examined methods. Then we will propose a new method inspired by the aforementioned ones and their new optimizations. In the end, we will compare all the methods on several benchmarks from the International Planning Competitions and propose topics for further research.
Key words: classical planning, optimalization, redundant actions, inverse cycles, local landmarks

Obsah

Seznam použitých zkratek

IPC International Planning Competition

AIPS The International Conference on Artificial Intelligence Planning Systems

MPT Multi-Valued Planning Task

AE Action Elimination

GAE Greedy Action Elimination

IAE Inverse Action Elimination

GCAE Greedy Chaining Action Elimination

AELS Action Elimination with Landmark Skip

AELSe Action Elimination with Enhanced Landmark Skip

GAELS Greedy Action Elimination with Landmark Skip

GAELSe Greedy Action Elimination with Enhanced Landmark Skip

AECD Action Elimination with Cycle Detection

GAECD Greedy Action Elimination with Cycle Detection

AECB Action Elimination with Combined Optimization of Landmark Skip and Cycle Detection

AECBe Action Elimination with Combined Optimization of Enhanced Landmark Skip and Cycle Detection

GAECB Greedy Action Elimination with Combined Optimization of Landmark Skip and Cycle Detection

GAECBe Greedy Action Elimination with Combined Optimization of Enhanced Landmark Skip and Cycle Detection

PNGS Plan Neighborhood Graph Search

MLR Minimal Lenght Reduction

MR Minimal Reduction

Seznam obrázků

Úvod

Automatické plánování je obor zaobírající se formálním popisem problému a následným hledáním posloupnosti akcí transformující prostředí z počátečního stavu do cílového (Ghallab; Nau; Traverso 2004). Klasické plánování probíhá v diskrétním statickém prostředí, jeho akce jsou deterministické a mají okamžitý účinek. Takové prostředí lze popsat grafem. Proto lze k hledání řešení použít například *Dijkstrův algoritmus* pro hledání nejkratší cesty (Dijkstra 1959). Tento algoritmus ale ve velkém grafu narazí na problém, že je často třeba prohledat nemalý počet vrcholů a hran. S tímto si umí částečně poradit algoritmus A^*, který pomocí dobře designované heuristiky dokáže odhadnout, kterým směrem se nachází cíl a tyto směry prohledává prioritně (Hart; Nilsson; Raphael 1968).

V obecnosti je klasické plánování *PSPACE-úplné* (Bylander 1994). Práce Helmert 2003 ukázala, že pro některé třídy problémů (použitých na *The International Conference on Artificial Intelligence Planning Systems* (AIPS) 1998[1] a 2000[2]) je časté, že nalezení optimálního plánu určité délky je znatelně složitější než nalezení libovolně dlouhého plánu. V případě, že nám nezáleží na kvalitě plánu, můžeme od požadavku na optimalitu upustit, čímž lze dosáhnout nalezení plánu, jenž sice není optimální, ale nalezneme jej rychleji.

I z tohoto důvodu vznikly různé relaxace algoritmu A^* a metody *Best-First Search* (Dechter; Pearl 1985), které hledají suboptimální cesty. Za modifikace algoritmu A^* lze považovat například použití nepřípustné heuristiky nebo algoritmus *Weighted* A^* a jeho obdoby (Pohl 1970) (Ebendt; Drechsler 2009). Současné suboptimální state-of-the-art plánovače často využívají algoritmus *Greedy Best-First Search* k nalezení prvního libovolného řešení. To sice zpravidla není optimální, ale stále nám přináší informaci o možné délce a ceně řešení, kterou můžeme v případném následném opětovném voláním algoritmu *Weighted* A^* použít jako horní mez při vyhledávání tak, že za váhu algoritmu zvolíme právě cenu, nebo délku již nalezeného plánu, a ořezat tak prohledávaný stavový prostor od již nezajímavých stavů (Richter; Westphal 2010) (Francès; Geffner; Lipovetzky; Ramírez 2018). Snižující se váha vede k postupném zlepšování plánu a ten se blíží optimu, dokud algoritmus neukončíme nebo nedojde ke spočtení iterace s váhou jedna, tedy k běhu již neváženého A^* (v tomto případě je optimalita takového plánu závislá na přípustnosti použité heuristiky).

Se vzniklým suboptimálním plánem se můžeme spokojit a prohlásit ho za řešení problému, nebo se můžeme pokusit plán optimalizovat. V této práci se zaměříme na vylepšování libovolných suboptimálních plánů klasického plánování. V plánech identifikujeme a následně odebereme takové akce, jejichž přítomnost v plánu není nezbytná pro dosažení cíle (a jsou tedy redundantní).

K tomu lze například použít algoritmus *Action Elimination* (Nakhost; Müller 2010a), jenž hlouběji probereme v této práci v sekci 2.2. Ten pracuje tak, že zkusí z existujícího plánu postupně odebrat každou akci (a spolu s ní i ty, jenž se v tom případě stanou neaplikovatelné) a v případě, že je vzniklý plán validní, tak ho prohlásí za optimálnější variantu. O něco sofistikovanější algoritmus vycházející z *Action Elimination* je algoritmus *Greedy Action Elimination* (Balyo; Chrpa; Kilani 2014). Ten nejprve spočte všechny

[1] Koenig; Simmons 1998.

[2] Bacchus; Kautz; Smith; Long; Geffner; Koehler 2000.

možné skupiny odebratelných akcí (stejně jako to dělá *Action Elimination*) a odebere právě tu, která má nejvyšší úhrnnou cenu. Tento algoritmus blíže přiblížíme v sekci 2.3. Jako poslední přejatý algoritmus popíšeme v kapitole 3 metodu *Inverse Action Elimination* (Chrpa; T. L. McCluskey; Osborne 2012), jenž funguje na odlišném principu. Místo přístupu "pokus, omyl", *Inverse Action Elimination* plán analyzuje na úrovni množin efektů a předpokladů. Díky tomu lze v plánu nalézt akce, které se nepodílí na dosažení cíle, a nebo třeba akce, jejichž efekty se vyruší s efekty nějaké jiné akce v plánu, tzv. inverzní páry, jenž jsou hlavní podstatou metody a po nich je také pojmenována. Abychom mohli metodu *Inverse Action Elimination* porovnat s metodami *Action Elimination* a *Greedy Action Elimination*, musíme ji reformulovat do SAS^+ formalismu (stejně jako i ostatní metody, u nichž je reformulace zcela přímočará), protože právě s ním pracuje naše prostředí.

Následně zobecníme metodu *Inverse Action Elimination* tak, aby podporovala podmíněné efekty z domén soutěže *International Planning Competition* (IPC) 2018 (Pommerening; Torralba; Balyo 2018a) a byla porovnatelná s metodami *Action Elimination* a *Greedy Action Elimination* i na těchto doménách.

Rovněž zavedeme dva optimalizační přístupy, jenž integrujeme do metod *Action Elimination* a *Greedy Action Elimination*. První z nich bude inspirovaná metodami pro hledání landmarků[3], které slouží k navádění prohledávání stavového prostoru (Hoffmann; Porteous; Sebastia 2004). Druhá bude založena na principech, které zkoumá metoda *Inverse Action Elimination*, a na poznatcích získaných při jejím reformulování a zobecňování. Následně oba přístupy zkombinujeme pro získání ještě efektivnější optimalizace.

Jako poslední optimalizační přístup zavedeme a popíšeme metodu *Greedy Chaining Action Elimination*, která se bude snažit redukovat opětovné výpočty množin redundantních akcí v *Greedy Action Elimination* a přiblížit se rychlostí metodě *Action Elimination* s tím, že si zachová kvalitnější optimalizaci podobnou té v *Greedy Action Elimination*. Při té příležitosti se zmíníme o nedostatcích všech metod a navrhneme možné řešení a předložíme podněty k dalšímu výzkumu.

Na závěr zanalyzujeme a porovnáme výkony zmíněných optimalizačních metod na doménách ze soutěží IPC (Helmert; Do; Refanidis 2010; Chrpa; Vallati; L. McCluskey 2014a; Pommerening; Torralba; Balyo 2018a), včetně soutěže IPC 2018, na které byly použity domény s podmíněnými efekty.

[3]Termín "landmark" lze přeložit do češtiny jako "milník", ale z důvodu zachování konzistence se zahraniční literaturou tak v této práci neučiníme.

Kapitola 1

Klasické plánování

Jak jsme již zmínili v úvodu, v automatickém plánování jde o automatizované hledání posloupnosti akcí, které nás dovedou do cíle. Automatické plánování rozdělujeme do několika tříd (Ghallab; Nau; Traverso 2016). Ty se liší například stochasticitou, zpracováním času anebo kompletností znalosti prostředí. Námi zkoumané klasické plánování probíhá v diskrétním statickém prostředí, jeho akce jsou deterministické, mají okamžitý účinek a formálně ho popíšeme v následující sekci.

1.1 Rozšířený SAS^+ formalismus

Domény a problémy klasického plánování bývají specifikovány ve *STRIPS* formalismu (Fikes; Nilsson 1971). Hlavním specifikem této formalizace je, že proměnná je atom, který nabývá vždy právě jedné z hodnot *platí*, *neplatí* (*pravda*, nebo *nepravda*). Existují plánovače, které plánují přímo v této reprezentaci jako třeba *YAHPS3* (Vidal 2014) nebo *LPG* (Gerevini; Saetti; Serina 2008). Nicméně my pro implementaci zmíněných optimalizačních metod využijeme SAS^+ (*Simplied Action Structures*) formalismus (Bäckström; Nebel 1995), která nabízí kompaktnější reprezentaci, kde každá proměnná má svoji doménu přípustných hodnot. Tento formalismus je dále rozšířen tak, aby podporoval podmíněné efekty akcí (Helmert 2006).

Definice 1.1. *Přiřazení je uspořádaná dvojice* (v, val)*. kde v je proměnná a val je hodnota proměnné z množiny všech možných hodnot* $dom(v)$*.*

Přiřazení spojujeme do množin. Takové množiny reprezentují stavy (vrcholy prohledávaného grafu) a jsou součástí definic akcí.

Definice 1.2. *Stav je množina přiřazení, ve kterých se každá proměnná vyskytuje nejvýše jednou. Hodnotu proměnné v ve stavu s označíme* $s[v]$ *a platí, že* $s[v] = val : (v, val) \in s$*.*

Plánovací úloha formálně modeluje konkrétní problém, jenž chceme řešit. Každá úloha klasického plánování musí mít přesně definované proměnné, které se v ní budou vyskytovat, jejich možné hodnoty, počáteční stav, popis situace, které chceme dosáhnout, a akce (hrany prohledávaného grafu), jenž popisují podmíněné přechody mezi stavy.

Definice 1.3. *Plánovací úloha (v rozšířeném* SAS^+ *formalismu) je uspořádaná čtveřice* $T = (X, O, s_I, s_G)$*.*

- $X = \{x_i, \ldots, x_n\}$ *je množina všech proměnných. Ke každé proměnné máme její doménu* $dom(x_i)$*.*
- *O je množina akcí (také operátorů). Každá akce* $a \in O$ *je trojice* $(pre(a), ceff(a), C(a))$*, kde* $pre(a)$ *je množina předpokladů akce a,* $ceff(a)$ *je množina podmíněných efektů akce a a* $C(a) \in \mathbb{N}_0$ *je cena akce a. Množina* $pre(a)$ *je množina přiřazení, v jejíž přiřazeních se každá proměnná vyskytuje nejvýše jednou. Množina* $ceff(a)$ *je množina*

uspořádaných dvojic (c, e)*, kde* c *je podmínka efektu a* e *efekt. Podmínka efektu* c *je množina přiřazení, v jejíž přiřazeních se každá proměnná vyskytuje nejvýše jednou, a efekt* e *je přiřazení. Efekt* e *nazýváme aktivní právě tehdy, když jsou v libovolném stavu* s *splněny všechny podmínky efektu* e*, tedy* $c \subseteq s$*.*

- s_I *je počáteční stav.*
- s_G *je množina cílových přiřazení, ve kterých se každá proměnná vyskytuje nejvýše jednou.*

Hlavní odlišností definice 1.3 od původního (nerozšířeného) SAS^+ formalismu je množina $\mathit{ceff}(a)$. V původní formalizaci šlo o množinu přiřazení $\mathit{eff}(a)$, která po aplikaci akce byla platná ve vzniklém stavu a to bezpodmínečně.

V rozšířeném formalismu má ale množina $\mathit{ceff}(a)$ složitější strukturu. Jak jsme již zmínili, jde o množinu dvojic (c, e), kde e je efekt a c je jeho předpoklad. Pokud $c = \emptyset$, pak je předpoklad splněný vždy a e je aktivní v libovolném stavu s, a proto se jedná o nepodmíněný efekt akce a (stejně jako tomu bylo v nerozšířeném formalismu). V ostatních případech je přítomnost efektu ve vznikajícím stavu podmíněna splněním podmínky c ve stavu s.

Také podotýkáme, že se od rozšíření SAS^+ z práce Helmert 2006, kde dostalo pojmenování *MPT* (*Multi-Valued Planning Task*), odlišujeme v tom, že součástí plánovací úlohy nemáme axiomy.

Přiřazení, kterých chceme dosáhnout, specifikujeme v množině s_G. Ta nemusí být splněna pouze v jediném stavu. Říkáme, že stav je řešením plánovací úlohy pokud platí, že každé přiřazení z množiny cílových přiřazení úlohy je i v množině přiřazení stavu. Stav tedy může mít libovolnou hodnotu proměnných, které množina cílového přiřazení úlohy nezmiňuje.

Definice 1.4. *Stav* s *nazveme cílovým pro danou plánovací úlohu* $T = (X, O, s_I, s_G)$*, pokud platí, že* $s_G \subseteq s$*.*

1.2 Aplikování akce

U aplikovatelnosti akce je to obdobně. Akce a je v daném stavu s aplikovatelná, pokud má splněné všechny předpoklady $\mathit{pre}(a)$. A to má přesně ve chvíli, kdy každé z přiřazení v předpokladu je i ve stavu s.

Definice 1.5. *Říkáme, že akce* a *je aplikovatelná ve stavu* s*, právě tehdy když* $\mathit{pre}(a) \subseteq s$*.*

Pro potřeby zjištění, zda daná proměnná figuruje v množině přiřazení, se nám bude hodit následující definice množiny asociovaných proměnných. Pomůže nám velice zjednodušit další definice a věty. Množina asociovaných proměnných není nic jiného, než množina proměnných, které jsou v množině přiřazení.

Definice 1.6. *Množina asociovaných proměnných* $vars(m)$ *pro množinu přiřazení* m *je množina* $vars(m) = \{v \mid (v, val) \in m\}$*.*

Abychom definovali stav vznikající aplikací aplikovatelné akce, potřebujeme vědět, které podmíněné efekty mají splněné podmínky, a tedy jejich efekt bude mít vliv.

Definice 1.7. *Nechť* $a = (\mathit{pre}(a), \mathit{ceff}(a), C(a))$ *je akce plánovací úlohy v rozšířeném* SAS^+ *formalismu a* s *je libovolný stav příslušný téže úloze.*
Množinou efektů $\mathit{eff}(a, s)$ *akce* a *ve stavu* s *myslíme množinu přiřazení* $\mathit{eff}(a, s) = \{e \mid (c, e) \in \mathit{ceff}(a), c \subseteq s\}$*.*

Pokud to bude z kontextu zřejmé (tedy bude jasné, o jaký stav se jedná), v práci budeme z důvodu přehlednosti často ztotožňovat termín “efekty akce” s termínem “efekty akce ve stavu”.

Aplikací akce v daném stavu dostaneme stav nový, jenž bude obsahovat všechna přiřazení z efektů a k tomu ještě ta přiřazení, která nejsou "přepsána" efekty. Tedy ta, jejichž asociované proměnné nejsou v efektech akce. V případě, že předpoklady akce v daném stavu splněny nejsou, operace není definována.

Definice 1.8. *Nechť $a = (pre(a), ceff(a), C(a))$ je akce plánovací úlohy v rozšířeném SAS$^+$ formalismu a s je libovolný stav příslušný téže úloze.*
Za podmínek, že akce a je ve stavu s aplikovatelná, notací $s' = \gamma(s, a)$ značíme stav s' vznikající aplikováním akce a ve stavu s, tedy $s' = \gamma(s, a) = eff(a, s) \cup \{(v, val) \mid (v, val) \in s, v \notin vars(eff(a, s))\}$.

Pro zjednodušení pozdějších zápisů a zvýšení expresivity definujeme množinu všech předpokladů podmíněných efektů.

Definice 1.9. *Nechť $a = (pre(a), ceff(a), C(a))$ je akce plánovací úlohy v rozšířeném SAS$^+$ formalismu.*
Množinou předpokladů podmíněných efektů $pre_{cond}(a)$ akce a myslíme množinu přiřazení $pre_{cond}(a) = \{c \mid (c, e) \in ceff(a)\}$.

Ještě definujeme množinu předpokladů, které jsou splněné v daném stavu. Například pokud je akce aplikovatelná, tak v množině předpokladů je celá množina $pre(a)$, případně i další předpoklady podmíněných efektů.

Definice 1.10. *Nechť $a = (pre(a), ceff(a), C(a))$ je akce plánovací úlohy v rozšířeném SAS$^+$ formalismu a s je libovolný stav příslušný téže úloze.*
Množinou splněných předpokladů $pre_{sat}(a, s)$ akce a ve stavu s myslíme množinu přiřazení $pre_{sat}(a, s) = (pre(a) \cup pre_{cond}(a)) \cap s$.

Poznamenáváme, že v tuto chvíli máme definované tři množiny předpokladů akce a. První z nich jsou nutné předpoklady $pre(a)$ akce a, které musí být splněny v nějakém stavu s, aby akce v onom stavu byla aplikovatelná. Druhou jsou podmíněné efekty $pre_{sat}(a, s)$, jejichž účelem je filtrování a ovlivňování efektů akce v závislosti na stavu s. Posledními definovanými předpoklady je množina splněných předpokladů $pre_{sat}(a, s)$, která obsahuje právě ta přiřazení, která jsou buďto v $pre(a)$ nebo v $pre_{cond}(a, s)$, a zároveň jsou splněna i ve stavu s.

1.3 Maximální a minimální efekty

Dále definujeme množinu maximální a minimálních efektů. Množina maximální efektů reprezentuje největší možnou množinu efektů akce. Při sestavování množiny maximálních efektů ignorujeme všechny předpoklady podmíněných efektů a všechny považujeme za splněné. Proto jsou v maximálních efektech veškeré podmíněné efekty akce. Podotýkáme, že v maximální množině efektů nemusí být každá proměnná nejvýše jednou.

Na druhou stranu v množinou minimálních efektů myslíme množinu efektů, které jsou přítomné vždy. Ty získáme tak, že vezmeme právě ta podmíněná přiřazení, jenž nemají žádné podmínky.

Definice 1.11. *Nechť $a = (pre(a), ceff(a), C(a))$ je akce plánovací úlohy v rozšířeném SAS$^+$ formalismu.*
Množinou maximálních efektů $eff_{max}(a)$ akce a myslíme množinu přiřazení $eff_{max}(a) = \{e \mid (c, e) \in ceff(a)\}$. Množinou minimálních efektů $eff_{min}(a)$ akce a myslíme množinu přiřazení $eff_{min}(a) = \{e \mid (c, e) \in ceff(a), c = \emptyset\}$.

1.4 Plán

Jak jsme již několikrát zmínili, v plánování jde o hledání posloupnosti akcí, jejichž postupné aplikace nás dostanou do cílového stavu. Libovolnou posloupnost akcí nazýváme plán.

Definice 1.12. *Plán* Π *pro plánovací úlohu* $T = (X, O, s_I, s_G)$ *je posloupnost operátorů* $\Pi = \langle a_1, \ldots, a_n \rangle$*, kde* $a_i \in O$ *pro* $1 \leq i \leq n$*. Takovýto plán má délku* $L(\Pi) = n$ *a cenu* $C(\Pi) = \sum_{i=1}^{n} C(a_i)$*.*

Definice 1.13. *Aplikaci plánu* Π *ve stavu* s *značíme jako* $\gamma(s, \Pi)$ *a tuto operaci definujeme jako* $\gamma(s, \Pi) = \gamma(s, \langle a_1, a_2, \ldots, a_n \rangle) = \gamma(\gamma(s, a_1), \langle a_2, \ldots, a_n \rangle)$ *a* $\gamma(s, \langle\rangle) = s$*.*

Libovolný plán buď může svou plánovací úlohu řešit, nebo ne. Nás pochopitelně zajímají právě ty plány, kdy stav vznikající aplikací plánu v počátečním stavu splňuje podmínky cílového přiřazení. Pokud by v průběhu aplikace plánu nastalo, že akce, která má být aplikována, je v daném stavu neaplikovatelná, celý výraz $\gamma(s, \Pi)$ není definovaný.

Definice 1.14. *Říkáme, že plán* Π *řeší plánovací úlohu* $T = (X, O, s_I, s_G)$ *právě tehdy, když* $s' = \gamma(s_I, \Pi)$ *je cílový stav. Říkáme, že plán je validní právě tehdy, když řeší plánovací úlohu.*

Při aplikování plánu dochází ke vzniku průběžných stavů. O těchto stavech budeme v průběhu práce mluvit, a proto je definujeme formálně. Budeme jim říkat stavy trajektorie plánu. Trajektorie vždy začíná počátečním stavem příslušné plánovací úlohy. Ostatní stavy s_i vznikají aplikací plánu $\Pi' = \langle a_1, \ldots, a_{i-1} \rangle$ ve stavu s_I.

Definice 1.15. *Nechť* $\Pi = \langle a_1, \ldots, a_n \rangle$ *je plán pro plánovací úlohu* $T = (X, O, s_I, s_G)$*. Posloupnost* $\langle s_1, \ldots, s_{n+1} \rangle = \langle \gamma(s_I, \langle\rangle), \gamma(s_I, \langle a_1 \rangle), , \gamma(s_I, \langle a_1, a_2 \rangle), \ldots, \gamma(s_I, \Pi) \rangle$ *nazveme trajektorie plánu* Π*. Pro* $1 \leq i \leq n$ *stav* s_i *nazveme stav trajektorie plánu* Π *příslušný akci* $a_i \in \Pi$ *(také stav asociovaný akci* a_i*). Stav* s_{n+1} *nazveme koncový stav trajektorie plánu. Stav* s_{n+1} *nemá žádnou příslušnou akci.*

1.5 Redundantní akce

Předtím, než objasníme, které akce jsou redundantní, musíme formálně vyjasnit, jak budeme akce z plánu odebírat (plán je konečná posloupnost a redundantní akce budou budou tvořit množiny).

Definice 1.16. *Nechť* Π *je plán pro plánovací úlohu* T*.*
Plánem Π *bez akcí z množiny* $m = \{a_{r_1}, \ldots, a_{r_n}\} \subseteq \Pi$ *myslíme plán* $\Pi' = \Pi \setminus m = \langle a_1, \ldots, a_{r_1 - 1}, a_{r_1 + 1}, \ldots, a_n \rangle \setminus (m \setminus a_{r_1})$*. Plán* Π'' *bez akcí z prázdné množiny zůstane nezměněn, tedy* $\Pi'' = \Pi \setminus \emptyset = \Pi$*.*

Definice 1.16 nám zakazuje, aby v odebírané množině byla akce, která se nenachází v redukovaném plánu. Takové situaci se v práci budeme muset vyvarovat, jelikož takovou operaci nemáme definovanou.

Další věcí, kterou je třeba adresovat, je totožnost akcí. V plánu Π se mohou nacházet dvě akce, které mají shodnou uspořádanou trojici $(pre(a), ceff(a), C(a))$. Takové akce ale nejsou totožné, jestliže se liší indexem v posloupnosti. Nechť na stejném principu fungují i množiny akcí. V každé množině může být nejvýše jedna akce a_i s danou uspořádanou trojici $(pre(a), ceff(a), C(a))$ a indexem i (odkazující na $i$tou akci plánu Π), a počet akcí se shodnou uspořádanou trojicí není omezen.

Z praktických důvodů definujeme ještě operaci $\setminus$ i pro plán a podposloupnost téhož plánu.

Definice 1.17. *Nechť* Π *je plán pro plánovací úlohu* T*.*
Plánem Π *bez akcí z podposloupnosti* $\pi = \langle a_{r_1}, \ldots, a_{r_n} \rangle \subseteq \Pi$*, značeno* $\Pi \setminus \pi$*, myslíme plán* Π *bez akcí z množiny* $\{a_{r_i} \mid a_{r_i} \in \pi\}$*.*

Pro plánovací úlohu může existovat mnoho validních plánů. V takovém případě se plány mohou lišit kvalitou a obsahovat redundantní akce. Cílem této práce je identifikace a odstranění co nejvíce zbytečných akcí, a tím zapříčinění (potenciálního) snížení délky a ceny původního plánu. Následující definice tuto problematiku popíše formálně (Chrpa; T. L. McCluskey; Osborne 2012).

Definice 1.18. *Nechť* Π *je validní plán pro plánovací úlohu* T*.*
Množina akcí $m = \{a_{r_1}, \ldots, a_{r_n}\} \subseteq \Pi$ *je redundantní v* Π *právě tehdy, když je plán* $\Pi' = \Pi \setminus m$ *validní pro plánovací úlohu* T*. Plán* Π' *v takovém případě nazýváme redukce plánu* Π *a akce z množiny* m *nazveme redundantní v* Π*.*

Validní plán Π', jenž vznikl odebráním množiny redundantních akcí z Π, nazveme, stejně jako autoři Nakhost; Müller 2010a, redukce.

Pokud budeme mluvit o plánu optimálním, máme na mysli plán, jehož úhrnná cena je při nejhorším rovna jakémukoli jinému plánu. Za zmínku stojí, že délka plánu s jeho optimalitou nesouvisí. Nejkratší plán nemusí být optimální. Mohli bychom zavést definici optimality plánu odlišně tak, aby zohledňoval nejprve úhrnnou cenu, a v případě rovnosti pak délku obou plánů. Nicméně v této práci budeme uvažovat pouze jednokriteriální přístup.

Definice 1.19. *Říkáme, že plán* Π *je optimální pro plánovací úlohu* T *právě tehdy, když je validní a pro každý validní plán* Π' *(pro stejnou úlohu* T*) platí, že* $C(\Pi) \leq C(\Pi')$*.*

1.6 Minimální redukce

V této práci budeme plány optimalizovat pouze tak, že budeme odebírat množiny redundantních akcí (o dalším možném přístupu se krátce zmíníme v následující kapitole). Z toho důvodu naše metody často ani nemohou dosáhnout optimálního plánu.

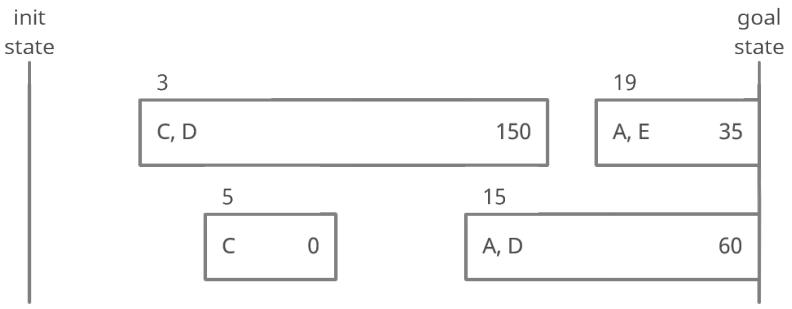
Pro plán Π existuje $2^{L(\Pi)}$ podplánů, z nichž pouze některé jsou validní pro původní úlohu. Každý takový validní podplán je redukcí plánu Π. V těch lze nalézt redukci, která bude mít přinejhorším stejnou cenu, jako každá jiná redukce stejného plánu.

Definice 1.20. *Nechť* Π *je validní plán pro plánovací úlohu* T *a* R *je množina všech možných redukcí* Π*.*
Redukci $\Pi' \in R$ *nazýváme minimální redukce právě tehdy, když pro každou redukci* Π'' *platí, že* $C(\Pi') \leq C(\Pi'')$*.*

Minimální redukce bude optimálnější výsledek pro veškeré zkoumané metody. Tato redukce nemusí být (a zpravidla ani není) shodná s optimálním plánem a její nalezení je *NP-úplný* problém (Nakhost; Müller 2010b).

1.7 Plánová schémata

Pro grafické znázornění rozdílů optimalizačních přístupů budeme potřebovat poukázat na možné existující závislosti redundantních množin. Právě k tomu nám slouží skupiny vzájemného vyloučení. Pro každou skupinu vzájemného vyloučení může být vždy odebrána nejvýše jedna množina redundantních akcí. Simulujeme tím, že redundance jedné množiny je závislá na přítomnosti každé další množiny ze skupiny vzájemného vyloučení.



Obrázek 1.1: ilustrační schéma

Schémata v práci několikrát použijeme a budeme s jejich pomocí prezentovat vlastnosti zkoumaných metod. Jsou v nich zakresleny a popsány množiny redundantních akcí v daném plánu. Každé schéma obsahuje dvě svislé úsečky znázorňující začátek a konec plánu, mezi kterými se nachází (za účelem dosažení větší přehlednosti ne nutně v jednom řádku) obdélníky, jež znázorňují nejmenší možné části plánu, v nichž se vyskytují všechny akce dané množiny redundantních akcí a všechny ostatní akce, které může odebrání akcí ovlivnit. Pokud "končí" dříve, než je dosaženo cílového stavu, znázorňujeme jím inverzní cyklus. Každá má svoji úhrnnou cenu (vpravo) a seznam skupin vzájemného vyloučení (vlevo). Nad levým horním rohem obdélníku se nachází pořadí první akce redundantní skupiny.

Na obrázku 1.1 jsou znázorněny čtyři množiny inverzních akcí, přičemž dvě z nich jsou inverzní cykly (začínající třetí a pátou akcí). Úhrnná cena akcí inverzního cyklu začínajícího pátou akcí je nula a mohou být odebrány pouze, pokud nedošlo k odebrání druhého cyklu, jenž začíná třetí akcí a má cenu sto padesát. Obdélníky, jenž "končí" až v koncovém stavu, znázorňují množiny redundantních akcí, u nichž si nejsme jisti, zda mohou následné akce ovlivnit, a tak musíme obdélník "protáhnout" do konce.

Kapitola 2

Současné optimalizační metody

2.1 Přehled literatury

Jak jsme se již zmínili, způsoby optimalizování plánů a metody samotné lze rozdělit do dvou skupin. Metody první skupiny se zaměřují na (námi již zmíněnou) identifikaci a odebírání nepotřebných akcí. Do této skupiny patří všechny metody, které v této práci podrobněji zmíníme.

V práci autorů Fink; Yang 1992 byly popsány čtyři algoritmy, jenž přísluší první skupině, přičemž jeden z nich byl později znovu objeven autory Nakhost; Müller 2010a a pojmenován *Action Elimination* (AE). Spolu s nimi bylo ukázáno, že jenom rozhodnout, zda plán obsahuje množinu redundantních akcí je *NP-úplný* problém.

Společně s metodou *Greedy Action Elimination* (GAE), které popíšeme později, byly publikovány další dva přístupy, *Minimal Lenght Reduction* a *Minimal Reduction*, příslušící první skupině. Identifikace redundantních akcí dosahují pomocí zakódování závislostí akcí v plánu do konjunktivní normální formy definující následně řešený problém vážené maximální splnitelnosti booleovské formule (*MAXSAT*) (Balyo; Chrpa; Kilani 2014). Jak již název napovídá, na rozdíl od metod AE a GAE, metoda *Minimal Reduction* (MR) nalézá minimální redukci, a proto řeší naši úlohu jako jediná (z přístupů zmíněných v této práci) optimálně (Balyo; Chrpa; Kilani 2014). Přístup *Minimal Lenght Reduction* (MLR) se od přístupu MR liší v tom, že neminimalizuje cenu plánu, ale jeho délku. I přes minimalizování jiné účelové funkce dosáhl obdivuhodných výsledků (Balyo; Chrpa; Kilani 2014).

Druhou skupinu tvoří metody prozkoumávající stavový prostor. Díky znalosti trajektorie validního plánu lze prozkoumat jeho okolí a nalézt případné zkratky. Metody z této skupiny se nejvíce liší přístupem k prozkoumávání a jeho rozsahem.

Spolu s metodou *Action Elimination* byla publikována metoda *Plan Neighborhood Graph Search* (Nakhost; Müller 2010a). Metoda expanduje předem daný počet stavů stavového prostoru v blízkosti trajektorie. V takovémto grafu je následně vyhledána nejkratší cesta do libovolného cílového stavu. Díky tomu může dosáhnout optimalizace, která je metodami, jenž odebírají redundantní akce, nedosažitelná. Jinými optimalizačními přístupy (jimiž byla ovlivněna i metoda *Plan Neighborhood Graph Search* (PNGS)) pracujících na podobných myšlenkách jsou *Iterative Tunneling Search with* A^* (Furcy 2006) a *Joint* a *Local Path* A^* (Ratner; Pohl 1986).

Přístupy, jenž (opakovaně) přeplánovávají pouze části plánů pomocí převodu úlohy na problém splnitelnosti booleovské formule, jsou popsány v pracích Balyo; Barták; Surynek 2012a a Balyo; Barták; Surynek 2012b. Další přístup, jenž optimalizuje plány skrze přeplánovávání jeho částí, je popsán v práci Siddiqui; Haslum 2015, díky němuž dokázali autoři často předčit plány, jenž poskytují současné *anytime* plánovače.

2.2 Action Elimination

Metoda *Action Elimination* (Nakhost; Müller 2010a) má nejnižší asymptotickou časovou složitost ze zkoumaných metod. Její hlavní myšlenka spočívá v tom, že procházíme plán (od začátku do konce), každou akci zkusíme vynechat a otestujeme, zda je plán bez této akce stále validní. Odebrání způsobí, že některé následující akce nemusí být aplikovatelné. Takové akce budeme ignorovat. Ostatní akce aplikujeme do námi drženého stavu s' a přejdeme na další. Pokud na konci plánu platí, že $s_G \subseteq s'$, můžeme všechny ignorované akce odebrat, protože se nepodílí na dosažení cíle.

Oproti algoritmu z původní práce autorů Nakhost; Müller 2010a jsme do algoritmu 1 integrovali optimalizaci. Akce redundantní skupiny neodebíráme okamžitě, nýbrž až na konci běhu algoritmu (řádek 19). Do té doby je máme pouze označené jako odebrané a ignorujeme je.

Algorithm 1: Action Elimination

```
   Input: plan Π = ⟨a_1, ..., a_n⟩, initial state s_I, goal state s_G
   Out  : a reduced plan
 1 removed ← ∅;
 2 marks ← ∅;
 3 s ← s_I;
 4 for i ← 1 to n do
 5     if i ∉ removed then
 6         marks ← marks ∪ {i};
 7         s' ← s;
 8         for j ← i + 1 to n do
 9             if j ∉ removed then
10                 if a_j is not applicable in s' then
11                     marks ← marks ∪ {j};
12                 else
13                     s' ← γ(s', a_j);
14         if s_G ⊆ s' then
15             removed ← removed ∪ marks;
16         else
17             s ← γ(s, a_i);
18         marks ← ∅;
19 Π ← ⟨a_i | a_i ∈ Π, i ∉ removed⟩;
20 return Π;
```

Také podotýkáme, že algoritmus 1 je pseudokód. Pro implementaci *marks* a *removed* jsme použili pole. Zkoušeli jsme i variantu podobnější pseudokódu, kde jsme nahradili pole *marks* množinou, což způsobilo významné zpomalení metody, a proto jsme od toho opustili a zvolili jsme rychlejší přístup.

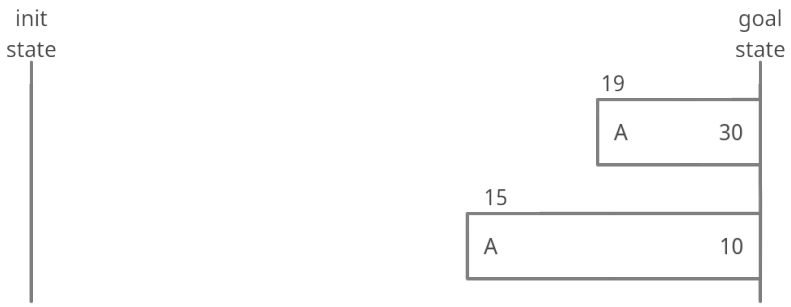
Protože pro každou akci v plánu procházíme nejvýše n následujících akcí, složitost tohoto algoritmu je tedy $O(n^2)$, kde n je délka plánu.

2.3 Greedy Action Elimination

Metoda *Greedy Action Elimination* (Balyo; Chrpa; Kilani 2014) je částečně prozřetelnější. Představme si situaci, kdy máme v plánu dvě skupiny redundantní akcí g_1 a g_2,

z nichž je vždy ale odebratelná nejvýše jedna (při odebrání obou skupin by došlo k porušení validity plánu). Metoda *Action Elimination* (Nakhost; Müller 2010a) vždy odebere právě tu skupinu, jejíž libovolná akce a předchází všem ostatním akcím obou redundantních skupin. Bez újmy na obecnosti předpokládejme, že a je ze skupiny g_1. Pokud má ale skupina g_1 nižší úhrnnou cenu než g_2, výsledný plán dosáhne horší kvality, než kdyby byla odebrána skupina g_2.

Tvrzení z předchozího odstavce graficky znázorňuje schéma[1] na obrázku 2.1. Skupina g_1 je znázorněna ve spodní části obrázku (začínající patnáctou akcí) a skupina g_2 v horní.



Obrázek 2.1: schéma znázorňující neoptimalitu metody Action Elimination

Tomuto, aby nebyla odebrána skupina g_1 a byla preferována hodnotnější množina, zabraňuje metoda *Greedy Action Elimination* tím, že před odebráním skupiny spočte úhrnnou cenu všech odebratelných skupin (pomocí opětovného volání algoritmu 3, který detekuje redundance obdobně jako vnitřní cyklus AE) a vybere tu, která ji má nejvyšší. Následně je skupina označena jako odebraná a výpočet se opakuje.

Algorithm 2: Greedy Action Elimination

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, initial state s_I, goal state s_G
Out : a reduced plan

$removed \leftarrow \emptyset$;
do
  $bestIndex \leftarrow 0$;
  $bestCost, bestMarks \leftarrow 0, \emptyset$;
  $s \leftarrow s_I$;
  for $i \leftarrow 1$ **to** n **do**
    if $i \notin removed$ **then**
      $cost, marks \leftarrow$ evaluateRemove$(\Pi, s, k, s_G, removed)$;
      if $cost \geq bestCost$ **then**
        $bestIndex \leftarrow i$;
        $bestCost, bestMarks \leftarrow cost, marks$;
      $s \leftarrow \gamma(s, a_i)$;
  if $bestIndex \neq 0$ **then**
    $removed \leftarrow removed \cup bestMarks$;
while $bestIndex \neq 0$;
$\Pi \leftarrow \langle a_i \mid a_i \in \Pi, i \notin removed \rangle$;
return Π;

[1] Schémata byly vysvětleny v podsekci 1.7.

Pro plán znázorněný na obrázku 2.1 metoda *Greedy Action Elimination* nalezne minimální redukci. To ale neznamená, že nalezení minimální redukce garantuje. Později, v kapitole 7, ukážeme, že tomu tak není.

Stejně jako u metody *Action Elimination*, i zde jsme provedli určité optimalizace. Akce opět neodebíráme hned, ale až po dokončení běhu algoritmu. Dále jsme upravili algoritmus *evaluteRemove* z původní práce (Balyo; Chrpa; Kilani 2014, Figure 2) tak, aby vracel dvojici *cena*, *označené akce*. Tím zajistíme to, že algoritmus *remove* z původní práce (Balyo; Chrpa; Kilani 2014, Figure 2) již nebudeme potřebovat. Jako poslední optimalizaci přidáme trajektorii plánu do výpočtu, abychom nemuseli stavy počítat vícekrát (algoritmus 2, řádky 5 a 12) (algoritmus 3, řádek 1).

Algorithm 3: Evaluate Remove

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, state s, action index to start with evaluating k, goal state s_G, set of removed actions *removed*
Out : a pair of cumulative cost and set of indeces of redundant actions

```
1  s' ← s;
2  cost ← C(a_k);
3  marks ← {k};
4  for i ← k + 1 to n do
5      if i ∉ removed then
6          if a_i is applicable in s' then
7              s' ← γ(a_i, s');
8          else
9              cost ← cost + C(a_i);
10             marks ← marks ∪ {i};
11 if s_G ⊆ s' then
12     return (cost, marks);
13 else
14     return (−1, ∅);
```

Hlavní cyklus metody se v nejhorším případě provede $n$krát (pokud se v každé iteraci odebere právě jedna akce). Pro každou takovou akci dojde k volání nejvýše n ohodnocení všech možných skupin (každé se složitostí $O(n)$), kde n je délka plánu. V původní práci autorů Balyo; Chrpa; Kilani 2014 by ještě došlo k jednomu vyhodnocení algoritmu odebrání se složitostí $O(n)$. Právě díky této optimalizaci jsme "snížili" složitost algoritmu z $O(n(n^2 + n))$ na $O(n(n^2) + 1)$, a proto je asymptotická složitost metody $O(n^3)$.

Kapitola 3

Inverse Action Elimination

Metoda *Inverse Action Elimination* (Chrpa; T. L. McCluskey; Osborne 2012) je, jak již název napovídá, postavena na hledání dvojic akcí (tzv. inverzních párů), jejichž efekty se vzájemně vyruší. K identifikaci těchto situací využívá později definovanou relaci $\rightarrow$ (Chrpa; T. L. McCluskey; Osborne 2012, Definition 3), která popisuje závislosti mezi akcemi. Už jen pomocí neexistence relace mezi konkrétními akcemi lze některé redundantní akce odebrat. Následně jsou identifikovány inverzní páry, u nichž je zkontrolována redundance.

V této práci se vyhneme uvádění veškerých postupů a principů metody *Inverse Action Elimination* (IAE) ve *STRIPS* formalismu tak, jak ho uvedla původní práce Chrpa; T. L. McCluskey; Osborne 2012, a rovnou metodu reformulujeme a rozšíříme. Proto se pro pochopení této kapitoly bude čtenáři hodit znalost metody IAE popsaná v předchozí (již zmíněné) práci.

3.1 Metoda a podmíněné efekty

Oproti metodám *Action Elimination* (AE) a *Greedy Action Elimination* (GAE) máme u metody *Inverse Action Elimination* problém s podmíněnými efekty. Metoda totiž analyzuje přímo efekty akcí. V původní práci (Chrpa; T. L. McCluskey; Osborne 2012) nebyly podmíněné efekty brány v potaz. Celá teorie metody byla vypracována s množinou efektů ve *STRIPS* formalismu, jenž je po transformaci metody do (nerozšířeného) SAS$^+$ formalismu ekvivalentní množině $\mathit{eff}_{min}(a)$.

Množina minimálních efektů $\mathit{eff}_{min}(a)$ nebude pochopitelně v doméně, která obsahuje podmíněné efekty, dostačující, a proto musíme metodu přeformulovat pro SAS^+ formalismus a nalézt sofistikovanější přístup, jenž si s podmíněnými efekty poradí.

V následujících dvou sekcích probereme možné přístupy k rozšíření metody IAE tak, aby metoda podporovala podmíněné efekty akcí a dala se tak použít i pro složitější domény.

3.2 Pesimistická reformulace

K tomu, abychom korektně reformulovali metodu IAE, máme více možných přístupů. Prvním a nejsnazším z nich je, že namísto množiny $\mathit{eff}_{min}(a)$ použijeme množinu $\mathit{eff}_{max}(a)$, tedy připravíme se na nejhorší možnou situaci, která může nastat.

Pokud i v takovémto případě lze určitou akci odebrat, tak je jistě odebratelná i ve všech případech, kdy nemusí být aktivní všechny efekty. (I zde bychom museli odladit určité krajní případy, pro něž předchozí věta není zcela pravdivá.) Důkazy pro tuto reformulaci neuvedeme a namísto toho definujeme a dokážeme reformulaci situační.

3.3 Situační reformulace

Další možná formulace je založená na využití informací ze stavů trajektorie plánu, které nám říkají, jaké budou efekty akce v dané situaci. Pokud dodržíme určitá pravidla, dokážeme vytvořit reformulaci, která potenciálně odebere více akcí než výše zmíněná pesimistická. K tomu využijeme množinu $\mathit{eff}(a, s)$, kde s je asociovaný stav akce a.

Na této myšlence v následujících sekcích přeformulujeme metodu IAE pro rozšířený SAS^+ formalismus.

3.4 Relace závislosti

Relaci závislosti akcí zachováme takovou, jaká byla ve *STRIPS* reformulaci. Pouze zaměníme množiny efektů, předpokladů a přidáme asociované stavy. Tím způsobíme, že veškeré aktivní podmíněné efekty budou zohledněny v relaci závislosti.

Definice 3.1. *Nechť* $\Pi = \langle a_1, \ldots, a_n \rangle$ *je plán pro plánovací úlohu* T *a* $\langle s_1, \ldots, s_{n+1} \rangle$ *je trajektorie plánu* Π.
Říkáme, že akce a_j *je přímo závislá na akci* a_i*, značeno* $a_i \rightarrow a_j$*, právě tehdy, když* $i < j$*,* $\mathit{eff}(a_i, s_i) \cap \mathit{pre}_{sat}(a_j, s_j) \neq \emptyset$ *a* $\mathit{eff}(a_i, s_i) \cap \mathit{pre}_{sat}(a_j, s_j) \nsubseteq \bigcup_{k=i+1}^{j-1} \mathit{eff}(a_k, s_k)$.
Dále říkáme, že akce a_j *je závislá na akci* a_i *právě tehdy, když* $a_i \rightarrow^* a_j$*, kde* $\rightarrow^*$ *je reflexivní tranzitivní uzávěr relace* $\rightarrow$.
Relace $\nrightarrow$ *(resp.* $\nrightarrow^*$*) značí, že akce nejsou přímo závislé (resp. závislé).*

Jak jsme již zmínili v úvodu kapitoly, už pomocí této relace dokážeme rozpoznat některé redundantní akce. Jako první metoda nalezne akce, v jejichž reflexivním tranzitivním uzávěru relace závislosti $\rightarrow$ se nenachází cíl (akce tedy nepřispívají k dosažení cíle). K tomu je třeba upravit plán tak, aby obsahoval speciální akce $a_I = (\emptyset, \{(\emptyset, e) \mid e \in s_I\}, 0)$ a $a_G = (s_G, \emptyset, 0)$ (Chrpa; T. L. McCluskey; Osborne 2012). Akce simulují počáteční a cílový stav původního problému T, tak aby "stavy" mohly být závislé na akcích z plánu $\Pi = \langle a_1, \ldots, a_n \rangle$.

Pomocí těchto akcí definujeme nový plán $\Pi' = \langle a_I, a_1, \ldots, a_n, a_G \rangle$, ve které nalezneme závislosti. Může být užitečné zmínit, že plán Π' řeší libovolnou úlohu T právě tehdy, když ji řeší plán Π. Pokud $a_I \nrightarrow^* a_G$ znamená to, že je úloha T nezávislá na počátečním stavu s_I. Akce $a_i \in \Pi : a_i \nrightarrow^* a_G$ můžeme z plánu Π odebrat z důvodu redundance v obou plánech.

Proto při optimalizaci plánu jako první nalezneme akce, které nemají v reflexivním tranzitivním uzávěru relace $\rightarrow$ akci a_G. Tyto akce označíme jako redundantní a dále se nimi nebudeme zaobírat.

3.5 Inverzní páry

Následně identifikujeme a odebereme dvojice inverzních akcí, které mohou být odebrány bez porušení validity plánu. Nejprve nalezneme všechny dvojice inverzních akcí, jejichž redundanci následně zkontrolujeme v algoritmu 5.

Díky formulaci přechodové funkce γ již nemusíme při definici inverzní akce nic upravovat. Pár akcí je dle definice inverzní, pokud sousledná aplikace obou z nich dospěje do původního stavu před aplikací.

Definice 3.2. *Akce* a_i *a* a_j *tvoří inverzní pár ve stavu* s *právě tehdy, když* $i < j$ *a* $\gamma(\gamma(s, a_i), a_j) = s$.

Takové dvojice mohou být z plánu odebrány v případě, že druhá akce v plánu zvrátí všechny efekty akce první (akce již nemusí být aplikovány bezprostředně po sobě), mezilehlé akce jsou stále aplikovatelné a stav po aplikaci páru je stejný, jako kdyby byly akce páru při aplikaci vynechané.

Věta 3.1. *Nechť* $\langle a_1, \ldots, a_n \rangle$ *je plán* Π *pro plánovací úlohu* $T = (X, O, s_I, s_G)$, $\langle s_1, \ldots, s_{n+1} \rangle$ *je trajektorie plánu* Π *a* a_i, $a_j \in \Pi, i < j$ *je dvojice inverzních akcí ve stavu* s_i. *Pokud* $\mathit{eff}(a_j, s_j) = \mathit{eff}(a_j, \gamma(s_i, a_i))$ *a neexistuje* a_k, *pro* $\forall k : i < k < j$, *že*

- $a_i \rightarrow a_k$ *nebo*
- $\exists (v, val) : (v, val) \in \mathit{eff}(a_j, s_j), v \in vars(\mathit{eff}(a_k, s_k)), (v, val) \notin \mathit{eff}(a_k, s_k)$ *nebo*
- $\exists c \in pre_{cond}(a_k) : \{(v, val) \mid (v, val) \in c, v \in vars(\mathit{eff}(a_i, s_i))\} \neq \emptyset \wedge \{(v, val) \mid (v, val) \in c, v \in vars(\mathit{eff}(a_i, s_i))\} \subseteq \{(v, s_i[v]) \mid v \in vars(\mathit{eff}(a_i, s_i))\}$,

pak je množina $\{a_i, a_j\}$ *redundantní v* Π.

Důkaz. Nechť a_i, $a_j \in \Pi, i < j$ je dvojice inverzních akcí ve stavu s_i.

Aby byla množina akcí redundantní v plánu Π, musí být redukovaný plán Π' validní, tedy řešit původní plánovací úlohu ($s_G \subseteq \gamma(s_I, \Pi')$). Proto musíme zaručit, že absence odebraných akcí nezpůsobí neaplikovatelnost libovolné akce (v takovém případě není γ definovaná) a zároveň bude cílový stav stále dosažen.

Nejprve ukážeme, že pokud $\mathit{eff}(a_j, s_j) = \mathit{eff}(a_j, \gamma(s_i, a_i))$ a neexistuje a_k, pro $\forall k : i < k < j$, že $a_i \rightarrow a_k$ nebo $\exists (v, val) : (v, val) \in \mathit{eff}(a_j, s_j), v \in vars(\mathit{eff}(a_k, s_k))$, $(v, val) \notin \mathit{eff}(a_k, s_k)$, pak jsou všechny akce v Π' aplikovatelné v jejich asociovaných stavech.

Protože každá akce ovlivňuje pouze stavy následující, akce $a_1, \ldots, a_{i-1}$ jsou jistě stále aplikovatelné. Akce $a_{i+1}, \ldots, a_{j-1}$ mohou být obecně ovlivněny akcí a_i. Akce $a_{j+1}, \ldots, a_n$ mohou obecně ovlivnit obě odebíratelné akce.

Pokud by odebrání akce a_i mělo způsobit neaplikovatelnost libovolné akce $a_{k'}$ z $\langle a_{i+1}, \ldots, a_{j-1} \rangle$, pak by muselo platit $a_i \rightarrow^* a_{k'}$. Jestliže $a_i \rightarrow^* a_{k'}$, pak nutně existuje $a_k, k \leq k'$, že $a_i \rightarrow a_k$. (Toto z důvodu reflexivity uzávěru relace $\rightarrow$ neplatí obecně, v našem případě ale $a_i \neq a_{k'}$, a proto je tvrzení pravdivé). Obměnou implikace dostaneme, že pokud $\neg \exists a_k : i < k < j \wedge a_i \rightarrow a_k$, pak $\neg \exists a_{k'} : i < k' < j \wedge a_i \rightarrow^* a_{k'}$. Z podmínky dokazované věty víme, že neexistuje a_k, pro $\forall k : i < k < j$, že $a_i \rightarrow a_k$. Akce a_i proto nemohla způsobit neaplikovatelnost libovolné akce z $\langle a_{i+1}, \ldots, a_{j-1} \rangle$.

Pokud zaručíme, že aplikací akce a_j zvrátíme veškeré efekty akce a_i, pak přímo z definice plyne, že neexistuje $a_l \in \langle a_{j+1}, \ldots, a_n \rangle : a_i \rightarrow a_l$, tedy (obdobně jako v předchozím odstavci) akce a_i neovlivní aplikaci ani těchto akcí.

Pokud zároveň zaručíme, že aplikací akce a_j zvrátíme veškeré efekty akce a_i a žádný jiný efekt akce a_j mít nebude a žádná další akce $a_k : i < k < j$ nezmění hodnotu žádné proměnné, kterou by a_j také zvrátila, pak pro každou akci $a_l : a_j \rightarrow a_l$ existuje akce $a_m : 1 \leq m < j, m \neq i$ taková, že se po odebrání akce a_j stane akce a_l závislá na akci a_m. To proto, že po aplikaci akce a_j má libovolná proměnná z $vars(\mathit{eff}(a_j, s_j))$ hodnotu stejnou, jako měla ve stavu s_i, takže musí existovat akce, která tuto hodnotu nastavila (podotkneme, že náš plán obsahuje i speciální akci a_I). Zároveň neexistuje akce, která by hodnotu této proměnné ještě následně změnila (taková akce by totiž musela být mezi akcemi a_i a a_j, a my z předpokladu věty víme, že tam není).

Pro zajištění aplikovatelnosti všech akcí stačí ukázat, že akce a_j zvrátí veškeré efekty akce a_i a žádný jiný efekt mít nebude. Z definice inverzního páru je toto splněno pro situace, kdy $i + 1 = 1$. Pro případy, kdy tato rovnost splněna není, musíme zajistit, že žádná mezilehlá akce nezmění efekty akce a_j ve stavu s_j. To máme zajištěno nutnou podmínkou dokazované věty, že $\mathit{eff}(a_j, s_j) = \mathit{eff}(a_j, \gamma(s_i, a_i))$.

Jako poslední věc musíme ukázat, že aplikací plánu Π' dosáhneme cílového stavu. Již víme, že nepřítomnost akcí a_i, a_j nezpůsobí neaplikovatelnost žádné z akcí plánu Π'. Ještě musíme ukázat, že veškeré ostatní akce mají totožné efekty. S akcemi $a_1, \ldots, a_{i-1}$ jistě není problém. U ostatních akcí mohou nastav dvě situace. Efekty mohou v obecnosti

získat nebo ztratit libovolný počet podmíněných efektů.

Víme, že libovolná akce a_k z $\langle a_{i+1}, \ldots, a_{j-1} \rangle$ nemůže přijít o žádný ze svých podmíněných efektů. Došlo by tím k porušení podmínky, že neexistuje $a_k : i < k < j, a_i \rightarrow a_k$. Aby akce a_k získala po odebrání akce a_i nové efekty, musela by akce a_i "kazit" $\rightarrow$ mezi a_k a některou předcházející akcí. To znamená, že akce a_k musí mít předpoklad podmíněného efektu, jehož největší možná podmnožina obsahující pouze přiřazení proměnných, které a_i mění, je ve stavu s_i splněna. Tuto situaci zachycuje podmínka, že $\exists c \in pre_{cond}(a_k) : \{(v, val) \mid (v, val) \in c, v \in vars(eff(a_i, s_i))\} \neq \emptyset \wedge \{(v, val) \mid (v, val) \in c, v \in vars(eff(a_i, s_i))\} \subseteq \{(v, s_i[v]) \mid v \in vars(eff(a_i, s_i))\}$. Pokud by taková největší možná podmnožina existovala, pak pár nemusí být redundantní. Pokud ale žádná taková podmínka neexistuje, pak absence akce a_i nemohla aktivovat nový podmíněný efekt. Jelikož jsme vyloučili přidání i odebrání podmíněného efektu, efekty akce a_k se nemění.

Již jsme ukázali, že pro každou akci a_l, která je přímo závislá na a_j, existuje po odebrání akce a_j jiná akce, která akci a_j v závislosti nahradí. Proto akce a_l nemohou ztratit podmíněné efekty. Protože efekty akcí $a_1, \ldots, a_{i-1}$ a $a_{i+1}, \ldots, a_{j-1}$ nemění svoje efekty a akce a_j zvrátila veškeré efekty akce a_i, tak nový asociovaný stav akce a_{j+1} je shodný se stavem s_{j+1}. Proto nemohou vzniknout žádné nové přímé závislosti, které by akci a_l způsobily získání nového podmíněného efektu. Pokud by nějaké takové měly vzniknout, nalezli bychom je už předtím. Akce a_l má tedy totožné efekty jako před odebráním inverzního páru.

Protože jsou všechny akce v Π' aplikovatelné, mají totožné efekty jako před odebráním inverzního páru a $\gamma(s_j, a_j) = \gamma(s'_{j-1}, a_{j-1})$, kde s'_{j-1} je nový asociovaný stav plánu Π', tak byl pár redundantní.

□

Jak je z věty 3.1 a důkazu zřejmé, mohli bychom zesílit podmínky, abychom získali přesnější popis redundance. Například pokud akce a_k mění proměnné, které mění i akce a_j, pak to nutně neznamená, že pár není redundantní. Může totiž existovat akce $a'_k, k < k' < j$, která nastaví hodnotu dotčené proměnné na stejnou hodnotu, jako by to udělala akce a_j. Stejně tak existence splněné podmnožiny předpokladu podmíněného efektu neimplikuje neredundanci. To proto, že doplněk podmnožiny předpokladu nemusí být splněn nebo podmíněný efekt mění nepodstatnou proměnnou.

3.6 Pořadí kontroly párů

Z věty 3.1 plyne, že pokud mezi inverzními akcemi žádná další není, můžeme je odebrat. Problém ale nastane ve chvíli, kdy se začne pár prolínat s dalším párem. Potom se může stát, že pokud páry zkontrolujeme ve špatném pořadí, budeme muset kontrolu zbytečně opakovat. Tomu můžeme částečně zabránit seřazením párů.

Definice 3.3. *Nechť $\Pi = \langle a_1, \ldots, a_n \rangle$ je plán a a_i, $a_{i'} \in \Pi, i < i'$ je dvojice inverzních akcí. Definujeme binární relaci $\prec$ tak, že pro každý další pár inverzních akcí $(a_k, a_{k'})$ platí $(a_i, a_{i'}) \prec (a_k, a_{k'})$ právě tehdy, když $i \geq k$.*

Lze dokázat (Chrpa; T. L. McCluskey; Osborne 2012), že pokud akce odebíráme v pořadí daném relací $\prec$, můžeme stanovit podmínky, za kterých je třeba pár opětovně zkontrolovat. Nicméně už samotné procházení páru v pořadí daném relací urychluje výpočet tím, že většina vzájemných poloh párů je kontrolována ve "správném" pořadí.

V práci Chrpa; T. L. McCluskey; Osborne 2012 jsou také popsány situace, kde máme dva páry inverzních akcí, jenž si vzájemně brání v odebrání, a mohou být odebrány pouze současně. Takovéto akce lze rozšířením metody popsat, detekovat a odstranit (Chrpa; T. L. McCluskey; Osborne 2012). Jde o konkrétní situaci, kterou jsme v naší implementaci nezohlednili, jelikož s danou situací si přístupy z pozdějších kapitol poradí i bez detailního popisu.

3.7 Implementace

Pro nalezení efektů akcí potřebujeme nejprve napočítat trajektorii plánu. Podotkneme, že stavy podél plánu není po odebrání inverzního páru a_i, a_j třeba přepočítávat. Stavy $s_1, \ldots, s_{i-1}$ jsou jistě netknuté, stejně tak stavy $s_{j+1}, \ldots, s_n$, protože stav $\gamma(s_j, a_j) = \gamma(s'_{j-1}, a_{j-1})$. Stavy $s_{i+1}, \ldots, s_{j-1}$ se sice liší, ale pouze v proměnných, které akce a_{i+1}, $\ldots, a_{j-1}$ nemají v předpokladech, popřípadě v předpokladech podmíněných efektů, které by mohly být splněny.

Algorithm 4: Inverse Action Elimination

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, initial state s_I, goal state s_G
Out : a reduced plan

```
1 determine action dependencies;
2 mark such action on which the goal is not dependent;
3 determine pairs of inverse actions and sort them with respect to ≺;
4 do
5 |  check pairs of inverse actions for redundancy and mark them if redundant;
6 while no action has been marked;
7 remove marked actions from the plan;
8 return a reduced plan;
```

Pokud páry neodebíráme okamžitě, je třeba se při hledání vypořádat s již označenými akcemi. Odebrání obecně může způsobit změnu stavů a závislostí, čímž může dojít k porušení předtím splněných podmínek. Víme, že stavy ovlivněny nejsou, a pokud ano, tak pouze tak, že to neovlivní efekty a splněné předpoklady. Proto musíme dát pozor pouze na vznik nové relace závislosti mezi a_i a a_k (algoritmus 5, řádek 8 a 10). Pokud ke vzniku takové závislosti došlo, pár již nemůžeme odebrat.

Algorithm 5: Marking redundant inverse actions

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, plan trajectory $\langle s_1, \ldots, s_{n+1} \rangle$, initial state s_I, goal state s_G, sequence σ of inverse pairs

```
1  foreach (a_i, a_j) in the sequence σ ordered by ≺ do
2  |  if a_i or a_j is marked then
3  |  |  continue;
4  |  violated ← false;
5  |  atoms ← ∅;
6  |  for k ← i + 1 to j − 1 do
7  |  |  if a_k is marked then
8  |  |  |  atoms ← atoms ∪ (eff(a_i, s_i) ∩ eff(a_k, s_k));
9  |  |  else
10 |  |  |  violated ← a_i → a_k ∨ pre_sat(a_k, s_k) ∩ atoms ≠ ∅ ∨ (∃(v, val) : (v, val) ∈
   |  |  |    eff(a_j, s_j), v ∈ vars(eff(a_k, s_k)), (v, val) ∉ eff(a_k, s_k)) ∨ (∃c ∈
   |  |  |    pre_cond(a_k) : {(v, val) | (v, val) ∈ c, v ∈ vars(eff(a_i, s_i))} ≠
   |  |  |    ∅ ∧ {(v, val) | (v, val) ∈ c, v ∈ vars(eff(a_i, s_i))} ⊆ {(v, s_i[v]) | v ∈
   |  |  |    vars(eff(a_i, s_i))});
11 |  |  |  if violated then
12 |  |  |  |  break;
13 |  if ¬ violated then
14 |  |  mark both a_i and a_j;
```

Metoda *Inverse Action Elimination* se dá tedy shrnout do následující postupu. Nejprve nalezneme závislosti mezi akcemi. Poté odebereme akce, na kterých není závislá cílová akce. Ve zbylých akcích nalezneme všechny páry inverzních akcí a seřadíme je dle $\prec$. Tento seřazený seznam a vypočtené stavy předáme algoritmu 5. Ten ověří redundantnost daných párů, a pokud označí libovolný pár, tento krok opakujeme. Nakonec můžeme všechny označené akce odebrat a prohlásit plán za optimalizovaný.

Přístup také nemusí odstranit veškeré redundantní akce. V případě, že efekty akce nejsou zvráceny jedinou (inverzní) akcí, ale třeba čtyřmi, nedojde metodou IAE k jejich identifikaci. Takové množiny akcí nazveme inverzními cykly a na jejich identifikaci se zaměříme v kapitole 5.

3.8 Algoritmická složitost

Nalezení akcí, které ve svém reflexivním tranzitivním uzávěru nemají speciální akci a_G, má složitost $O(n^2)$, kde n je délka plánu. Nalezení párů inverzních akcí lze provést se složitostí $O(n^2)$. Složitost kontroly redundance páru je $O(l)$, kde l je největší počet akcí mezi akcemi každého z párů inverzních akcí. Tato kontrola je v každé iteraci provedena nejvýše $k$krát, kde k je počet párů. Celkový počet iterací je nejvýše $\frac{n}{2} + 1$ (každá iterace musí označit nejméně dvě akce, jinak dojde k ukončení). Asymptotická časová složitost metody je proto $O(n^2 + n^2 + (\frac{n}{2} + 1)(kl)) = O(n^2 + n^2 + \frac{nkl}{2} + kl)$, a protože $l \leq n$ a $k \leq n^2 - n$, tak platí, že asymptotická složitost metody je $O(n^4)$. Nutno podotknout, že jde o pesimistický odhad, který předpokládá, že v plánu délky n může být $n^2 - n$ párů, z nichž bude v každé iteraci odebrán právě jeden.

Kapitola 4

Lokální landmarky

Landmarky[1], tak jak je definuje literatura, jsou tvrzení nad přiřazeními či akcemi, jenž popisují, jaké přiřazení či akce musí být splněny v každém validním plánu (Hoffmann; Porteous; Sebastia 2004). Jejich použití při formování heuristického odhadu použité při prohledání stavového prostoru prokázalo jejich velký a praktický význam (Richter; Westphal 2010; Helmert; Domshlak 2009). Takovéto obecné landmarky budeme nadále v této práci nazývat globálními. Oproti tomu, landmarky, tak jak je budeme hledat v této kapitole, které najdeme s pomocí plánu, budeme nazývat lokálními.

Navrhneme metodu, s níž dokážeme nalézt lokální landmarky. Díky těm budeme moci identifikovat, jaké akce nelze z daného plánu odebrat a urychlit tak optimalizaci díky dodatečnému ořezávání.

4.1 Definice

Jak jsme již nastínili, na rozdíl od přístupů pro hledání globálních landmarků, kde máme k dispozici pouze plánovací úlohu a hledáme přiřazení, jenž musí být splněná v každém plánu, máme výhodu v tom, že zde máme k dispozici i plán. Jedním z postupů, jimiž se hledají globální landmarky, je zpětná propagace přiřazení, kterých je třeba dosáhnout (Hoffmann; Porteous; Sebastia 2004). Inspirováni tímto přístupem budeme postupovat i při hledání lokálních. Tím nalezneme přiřazení, která jsou podstatná. Oproti definicím globálních landmarků, my budeme pojmem landmark myslet akci, a ne přiřazení. Námi hledané lokální landmarky pak budou akce určené nalezenými přiřazeními, jenž navíc nemůžeme odebrat z plánu bez porušení jeho validity.

Definice 4.1. *Nechť* Π *je plán příslušný plánovací úloze* T.
Akci $a \in \Pi$ *nazveme lokální landmark pro plánovací úlohu* T *a plán* Π *právě tehdy, když neexistuje množina* m*, taková, že* $a \in m$ *a* m *je množina redundantních akcí v plánu* Π.

Jestliže se podíváme, jaká přiřazení jsou v počátečním stavu a jaké efekty mohou akce mít, a spočítáme výskyty jednotlivých přiřazení v obou těchto skupinách, tak ta přiřazení, která se v těchto efektech vyskytují právě jednou, jsou v plánu ojedinělá. Počáteční stav nebo akci, jenž poskytují určité přiřazení nazveme neformálně jako poskytovatele.

Definice 4.2. *Nechť* $T = (X, O, s_I, s_G)$ *je plánovací úloha a* Π *je validní plán příslušný téže plánovací úloze* T.
Přiřazení p *nazveme ojedinělé pro plánovací úlohu* T *a plán* Π *právě tehdy, když* $|\{a \mid a \in \Pi, p \in \mathit{eff}_{max}(a)\}| + |\{s_I \mid p \in s_I\}| = 1$.

[1] Termín "landmark" lze přeložit do češtiny jako "milník", ale z důvodu zachování konzistence se zahraniční literaturou tak v této práci neučiníme.

Taková přiřazení jsou kandidáty na to, aby byla pro plán nezbytné. Může se totiž stát, že takové přiřazení bude součástí definice cílového stavu. V takovém případě je přítomnost přiřazení v plánu jistě nezbytná, a proto ho nazveme významné přiřazení.

Tato situace nám implikuje další důležité pozorování. Protože je přiřazení ojedinělé, má právě jednoho poskytovatele. Pokud by tento poskytovatel v plánu nebyl (například bychom se ho pokusili odebrat), nedosáhli bychom cílového stavu.

Definice 4.3. *Nechť $T = (X, O, s_I, s_G)$ je plánovací úloha a Π je validní plán příslušný téže plánovací úloze T.*
Přiřazení p nazveme významné pro plánovací úlohu T a plán Π právě tehdy, když je ojedinělé pro T a Π a platí, že $p \in s_G$ nebo existuje $a \in \Pi$ takové, že a je lokální landmark a $p \in pre(a)$.

Libovolný poskytovatel významného přiřazení je akce, kterou z plánu odebrat nemůžeme, tedy jde i o lokální landmark. Proto musíme zaručit, že neporušíme jeho aplikovatelnost. Předpoklady takové akce musí být poskytnuty předchozími akcemi. Pokud je přiřazení z předpokladů takové akce ojedinělé, označíme ho také jako významné, protože bez něj nebude plán validní (stejně jako přiřazení, na kterém je závislý cílový stav), a jeho poskytovatel je další lokální landmark.

Tvrzení 4.1. *Nechť $T = (X, O, s_I, s_G)$ je plánovací úloha a Π je validní plán příslušný téže plánovací úloze T a a je libovolná akce z Π.*
Jestliže množina maximálních efektů $eff_{max}(a)$ obsahuje významné přiřazení pro T a Π, pak je a lokální landmark pro Π a T.

Důkaz. Nechť $p \in eff_{max}(a)$ je významné přiřazení pro T a Π. Pak víme, že p je ojedinělé, tedy a je jeho jediný poskytovatel. Při odebrání akce a z Π by již přiřazení nemohlo být splněno.

Protože je p významné, je buď součástí definice cílového stavu s_G nebo je v předpokladu jiné akce a', která je zároveň lokální landmark.

Pokud by p bylo součástí definice cílového stavu s_G, odebrání akce a by způsobilo, že nový plán již nebude validní (v novém plánu by nebyl žádný poskytovatel), a proto a nemůže být součástí žádné množiny redundantních akcí.

Pokud by p bylo součástí předpokladů jiné akce a', jenž je lokální landmark, pak by odebrání akce a způsobilo, že předpoklady akce a' nemohou být splněny (v novém plánu by nebyl žádný poskytovatel). To by ale způsobilo nevaliditu plánu, protože by akce a' nemohla být aplikovatelná.

V obou případech proto akce a nemůže být odebrána a nepatří tak do žádné množiny redundantních akcí.

□

4.2 Hledání lokálních landmarků

Pro nalezení lokálních landmarků potřebujeme nejprve spočítat výskyty přiřazení v maximálních efektech akcí a v počátečním stavu. Následně projdeme plán od konce a budeme propagovat vlastnost landmarků až k první akci.

Tvrzení 4.2. *Algoritmus 6 vrací množinu, jenž obsahuje pouze lokální landmarky.*

Důkaz. První část algoritmu pouze hledá možné poskytovatele.

Díky tomu, že do množiny *landmarks* se přidávají akce pouze v případě, že zkoumané přiřazení (z jeho maximálních efektů) má jediného poskytovatele, a právě tento poskytovatel je přidán, pak se v množině *landmarks* nacházejí pouze akce, jenž jsou poskytovatelé pro ojedinělé přiřazení. Další podmínkou pro přidání poskytovatele do množiny je, že

Algorithm 6: Local Landmark Search

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, initial state s_I, goal state s_G
Out : set of landmark's indeces

```
1  achievers ← empty map;
2  for p ∈ s_I do
3      achievers[p] ← {0};
4  for i ← 1 to n do
5      for p ∈ eff_max(a_i) do
6          if p ∉ achievers then
7              achievers[p] ← ∅;
8          achievers[p] ← achievers[p] ∪ {i};
9  landmarks ← ∅;
10 for p ∈ s_G do
11     if |achievers[p]| = 1 then
12         landmarks ← landmarks ∪ achievers[p];
13 for i ← n to 1 do
14     if i ∈ landmarks then
15         for p ∈ pre(a_i) do
16             if |achievers[p]| = 1 then
17                 landmarks ← landmarks ∪ achievers[p];
18 return landmarks;
```

zkoumané přiřazení (z jeho maximálních efektů) je součástí definice cílového stavu, nebo předpokladu jiného lokálního landmarku. Zkoumané přiřazení je proto, spolu s ojedinělostí, významné. Proto algoritmus vrací množinu akcí, z nichž každá akce je lokální landmark. □

Postupem použitým v algoritmu 6 nemusíme nalézt (a zpravidla nenalezneme) všechny lokální landmarky. K nalezení všech lokálních landmarků bychom museli identifikovat všechny množiny redundantních akcí.

Jako příklad situace, kdy nejsou nalezeny veškeré lokální landmarky, a k demonstraci nedostatků algoritmu 6 poslouží libovolný plán, v němž jsou akce a_i, a_j, a_k a existuje přiřazení p, kde $p \notin \mathit{eff}_{min}(a_j)$, $p \in \mathit{eff}_{max}(a_j)$ a $p \in \mathit{eff}_{min}(a_i)$, $p \in \mathit{pre}(a_k)$ a a_k je lokální landmark, a navíc se p v efektech žádné jiné akce a ani v počátečním stavu nevyskytuje. Pak je počet výskytů p roven dvěma. Proto dle naší definice není p ojedinělý, a proto ani významný, ačkoli se nachází v předpokladech lokálního landmarku. Pokud by ale podmínky podmíněného efektu a_j nikdy v plánu nemohly být splněné, nikdy by efekt nemohl být aktivní. S takovou znalostí bychom mohli poskytovatele a_j ignorovat a prohlásit tak přiřazení p za ojedinělé a významné a jeho jediného poskytovatele označit za lokální landmark, protože je v plánu jistě nezbytný.

4.3 Vylepšené hledání lokálních landmarků

S ohledem na poznatek z příkladu z předchozí kapitoly vylepšíme algoritmus 6 tak, aby nalezl více nezbytných akcí.

Algoritmus 7 je vylepšenou variantou algoritmu 6, jehož vylepšení je založeno na myšlence, že pokud existuje přiřazení p z maximálního efektu poslední akce a_n, které má x poskytovatelů v plánu $\Pi = \langle a_1, \ldots, a_n \rangle$, pak v plánu $\Pi' = \langle a_1, \ldots, a_{n-1} \rangle$ má p právě

$x - 1$ poskytovatelů. Protože se akce samy ovlivňovat nemohou a nemají na ně vliv ani akce následující, můžeme při zpětném průchodu snižovat počty poskytovatelů přiřazení. Takto detekujeme přiřazení p', která jsou ojedinělá pro zkrácené plány a původní plánovací úlohu.

Definice 4.4. *Nechť $T = (X, O, s_I, s_G)$ je plánovací úloha a $\Pi = \langle a_1, \ldots, a_n \rangle$ je plán příslušný téže plánovací úloze T.*
Přiřazení p nazveme striktně významné pro plánovací úlohu T a plán Π právě tehdy, když platí:

- *$p \in s_G$ a p je ojedinělé pro T a Π, nebo*
- *existuje a_i takové, že a_i je lokální landmark, $p \in pre(a_i)$ a p je ojedinělé pro T a $\Pi' = \langle a_1, \ldots, a_{i-1} \rangle$.*

Je zřejmé, že každé významné přiřazení je i striktně významné. Následujícím tvrzením a důkazem ukážeme, že tvrzení 4.1 platí i pro obecnější pojem striktně významného přiřazení.

Tvrzení 4.3. *Nechť $T = (X, O, s_I, s_G)$ je plánovací úloha a Π je validní plán příslušný téže plánovací úloze T a a je libovolná akce z Π.*
Jestliže množina maximálních efektů $eff_{max}(a)$ obsahuje striktně významné přiřazení pro T a Π, pak je a lokální landmark pro Π a T.

Důkaz. Nechť $p \in eff_{max}(a)$ je striktně významné přiřazení pro T a Π. Pak nastane alespoň jedna ze dvou následujících možností. První z nich je, že $p \in s_G$ a p je ojedinělé pro T a Π. Druhou situací je, že existuje a_i takové, že a_i je lokální landmark, $p \in pre(a_i)$ a p je ojedinělé pro T a $\Pi' = \langle a_1, \ldots, a_{i-1} \rangle$.

V prvním případě je p součástí definice cílového stavu s_G. Odebrání akce a by způsobilo, že nový plán již nebude validní (v novém plánu by nebyl žádný poskytovatel, protože p je v T a Π ojedinělé), a proto a nemůže být součástí žádné množiny redundantních akcí.

V druhém případě je p součástí předpokladů jiné akce a_i, jenž je lokální landmark. Pak by odebrání akce a způsobilo, že předpoklady akce a_i nemohou být splněny (v novém plánu by nebyl žádný poskytovatel, protože p je pro $\langle a_1, \ldots, a_{i-1} \rangle$ a T ojedinělé). To by ale způsobilo nevaliditu plánu, protože by akce a_i nemohla být aplikovatelná, jelikož by nemohla mít splněný předpoklad.

V obou případech proto akce a nemůže být odebrána a nepatří tak do žádné množiny inverzních akcí.

□

Pro nalezení nově definovaných striktně významných přiřazení a díky nim i odhalení více lokálních landmarků nám stačí jemně modifikovat algoritmus 6. Před kontrolou zda je daná akce prokazatelně lokální landmark a zda je přiřazení lokálního landmarku nebo cílového stavu ojedinělé, musíme pro každé přiřazení z maximálních efektů akce upravit množinu poskytovatelů tak, aby neobsahovala danou akci (akce nemůže ovlivnit svoje předpoklady, ani předpoklady akcí předcházejících). Tím si udržíme poskytovatele v daných podplánech, a díky těm můžeme hledat ojedinělé výskyty i v nich. V algoritmu 7 se tato úprava nachází na řádku 15.

Tvrzení 4.4. *Algoritmus 7 vrací množinu, jenž obsahuje pouze lokální landmarky.*

Důkaz. Protože akce a_i vždy ovlivňuje pouze následující akce $a_j,\ j > i$, může být akce poskytovatelem přiřazení pouze pro následné akce. Jestliže tedy v algoritmu 7 dojdeme ke zpracování akce a_i, již nás jako poskytovatel nebude zajímat, protože její efekty nemohly mít na akce $a_k,\ k \leq i$ vliv. Z toho důvodu je můžeme odebrat.

Jestliže tak učiníme, zajistíme, že v množině *achievers* budou v každé chvíli běhu právě ti poskytovatelé, kteří mohou zpracovávanou akci ovlivnit.

Algorithm 7: Enhanced Local Landmark Search

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, initial state s_I, goal state s_G
Out : set of landmark's indeces

```
1  achievers ← empty map;
2  for p ∈ s_I do
3      achievers[p] ← {0};
4  for i ← 1 to n do
5      for p ∈ eff_max(a_i) do
6          if p ∉ achievers then
7              achievers[p] ← ∅;
8          achievers[p] ← achievers[p] ∪ {i};
9  landmarks ← ∅;
10 for p ∈ s_G do
11     if |achievers[p]| = 1 then
12         landmarks ← landmarks ∪ achievers[p];
13 for i ← n to 1 do
14     for p ∈ eff_max(a_i) do
15         achievers[p] ← achievers[p] \ {i};
16     if i ∈ landmarks then
17         for p ∈ pre(a_i) do
18             if |achievers[p]| = 1 then
19                 landmarks ← landmarks ∪ achievers[p];
20 return landmarks;
```

Zbytek důkazu je analogický s důkazem tvrzení funkčnosti algoritmu 6, pouze je jedinečnost přiřazení v podplánu, který mohl aplikace příslušné akce, či splnění cílového stavu ovlivnit.

□

4.4 Integrace

Algoritmy znázorňující integraci zkoumaných poznatků v práci neuvedeme, ale pouze ji popíšeme, jelikož je velice přímočará. Nejdříve si pomocí algoritmů 6 a 7 spočteme lokální landmarky. Protože lokální landmarky nejsou součástí žádné množiny redundantních akcí (dle definice 4.1), víme, že se ani nemusíme snažit je odebrat (v našem případě je jakkoli označit, tedy je přidat do množiny *marks*). Pokud by k tomu v algoritmech *Action Elimination* (AE) a *Greedy Action Elimination* (GAE) mělo dojít, prohlásíme skupinu za neredundantní a prohledávání ukončíme dříve, popřípadě ani prohledávat nezačneme.

Podotýkáme, že ve shrnutí uvedeme dvě varianty tohoto přístupu. První z nich, založenou na definici významného přiřazení 4.3 a využívající algoritmus 6, pojmenujeme - *with Landmark Skip* (zkráceně -LS), a druhou, založenou na silnější definici 4.4 a využívající algoritmus 7, nazveme - *with Enhanced Landmark Skip* (zkráceně -LSe). Ukážeme, že na námi zkoumaných metodách nemělo postupné snižování počtu poskytovatelů zásadní vliv, ačkoli metoda dosáhla v průměru lepších výsledků. V některých případech totiž časová investice vložená do nalezení více landmarků převyšuje čas ušetřený při samotném výpočtu algoritmu.

V implementaci algoritmů pro hledání lokálních landmarků reprezentujeme množinu

landmarks polem booleovských hodnot, jenž každá přísluší právě jedné akci a indikuje, zda daná akce je, či není lokální landmark. Plány bývají "krátké" a schraňování této informace u každé akce nezpůsobí paměťové problémy.

4.5 Algoritmická složitost

Asymptotická složitost obou algoritmů 6 a 7 je $O(2(n+1)) = O(2n+2) = O(n)$, kde n je délka plánu. Při prvním průchodu musíme spočítat výskyty všech přiřazení v počátečním stavu a v n maximálních efektech. Při druhém průchodu procházíme přiřazení cílového stavu a n předpokladů akcí.

Obě optimalizované varianty metod AE a GAE vyžadují právě jeden výpočet algoritmu 6 (resp. 7), proto po integrování výpočtu lokálních landmarků těchto metod stoupne asymptotická složitost o $O(2(n+1)) = O(n)$, kde n je délka plánu. Protože obě metody jsou více než lineárně asymptoticky složité, ve výsledné složitosti se neprojeví. Asymptotická složitost metody *Action Elimination* s (vylepšeným) přeskakováním landmarků (AELS(e)) je proto $O(n^2)$ a metoda *Greedy Action Elimination* s (vylepšeným) přeskakováním landmarků (GAELS(e)) má asymptotickou složitost $O(n^3)$.

Kapitola 5

Inverzní cykly

5.1 Definice

V této kapitole inspirováni metodou *Inverse Action Elimination* (IAE) popíšeme obecnější případ párů inverzních akcí (definice 3.2). Podstatou páru inverzních akcí je to, že druhá akce páru zvrátí všechny efekty akce první a žádné jiné efekty nemá. V důsledku toho se může stát, že pár můžeme odebrat, protože nemusí mít vliv na zbytek plánu a stav po aplikaci páru může být stejný, jako kdybychom akce z páru v plánu vynechali.

Na tuto vlastnost budeme cílit při definování inverzních cyklů. Inverzním cyklem myslíme podposloupnost akcí z plánu, jenž při vynechání nezmění příslušný stav v původní trajektorii. Z toho důvodu je pár inverzních akcí velice podobný inverznímu cyklu délky dva.

Definice 5.1. *Nechť $T = (X, O, s_I, s_G)$ je plánovací úloha, $\Pi = \langle a_1, \ldots, a_n \rangle$ je plán příslušný téže plánovací úloze T a $\langle s_1, \ldots, s_n, s_{n+1} \rangle$ je trajektorie stavů podél plánu Π. Inverzní cyklus (též skupina inverzních akcí) je podposloupnost $I = \langle a_{\iota_1}, \ldots, a_{\iota_m} \rangle$ akcí z Π splňující podmínku $\gamma(s_{\iota_1}, \langle a_{\iota_1}, a_{\iota_1+1}, \ldots, a_{\iota_m-1}, a_{\iota_m} \rangle \setminus I) = s_{\iota_m+1}$.*

Z rovnosti stavů pak víme, že nepřítomnost akcí z cyklu nebude mít vliv na libovolné následující akce. Dále také podotkneme, že rovnost $\gamma(s_{\iota_1}, \langle a_{\iota_1}, a_{\iota_1+1}, \ldots, a_{\iota_m-1}, a_{\iota_m} \rangle \setminus I) = s_{\iota_m+1}$ implikuje, že veškeré mezilehlé akce cyklu jsou aplikovatelné (jinak by totiž γ nebyla definovaná, a proto by neplatila rovnost).

Tvrzení 5.1. *Nechť $T = (X, O, s_I, s_G)$ je plánovací úloha, $\Pi = \langle a_1, \ldots, a_n \rangle$ je plán příslušný téže plánovací úloze T, $S = \langle s_1, \ldots, s_n, s_{n+1} \rangle$ je trajektorie stavů podél plánu Π. Pokud je $I = \langle a_{\iota_1}, \ldots, a_{\iota_m} \rangle$ inverzní cyklus a Π je validní, pak je množina $\{a_{\iota_1}, \ldots, a_{\iota_m}\}$ redundantní v Π.*

Důkaz. Vezmeme plán $\Pi' = \Pi \setminus I$.

Z toho důvodu, že akce neovlivňují předešlé stavy, trajektorie stavů S' takového plánu obsahuje zpočátku stejné stavy jako S až do stavu vznikajícího aplikací první odebrané akce, tedy S' začíná stavy $s_1, \ldots, s_{\iota_1}$.

Dále v trajektorii následuje $\iota_m - \iota_1 + 1 - m$ stavů, které vytváří mezilehlé akce $\langle a_{\iota_1}, a_{\iota_1+1}, \ldots, a_{\iota_m-1}, a_{\iota_m} \rangle \setminus I$. Poslední z těchto stavů je stav $\gamma(s_{\iota_1}, \langle a_{\iota_1}, a_{\iota_1+1}, \ldots, a_{\iota_m-1}, a_{\iota_m} \rangle \setminus I)$. Z toho, že I je inverzní cyklus, víme, že tento stav je shodný se stavem s_{ι_m+1} z trajektorie S. Předpoklady akcí, jenž následují až po inverzním cyklu, odebrání cyklu porušit nemohlo (protože se v obou trajektoriích nachází s_{ι_m+1}). Oba plány Π a Π' tak končí stavy $s_{\iota_m+1}, \ldots, s_{n+1}$.

Ze závěrů dvou předcházejících odstavců víme, že příslušné stavy akce a_{ι_m+1} z obou plánů jsou shodné, a také jsou následovány stejnou posloupností akcí $a_{\iota_m+2}, \ldots, a_n$. Proto i obě trajektorie S' a S končí stejnými stavy $s_{\iota_m+1}, \ldots, s_{n+1}$. Protože je Π validní, tak platí $s_G \subseteq s_{n+1} = \gamma(s_I, \Pi')$.

Zbývá ukázat, že jsou všechny akce v Π aplikovatelné. Akce před cyklem jistě ovlivněny nejsou. Aplikovatelnost mezilehlých akcí nám je garantována tím, že I je inverzní cyklus. Rovnost $\gamma(s_{\iota_1}, \langle a_{\iota_1}, a_{\iota_1+1}, \ldots, a_{\iota_m-1}, a_{\iota_m}\rangle \setminus I) = s_{\iota_m+1}$ nám totiž implikuje, že veškeré mezilehlé cyklu jsou aplikovatelné (jinak by totiž γ nebyla definovaná, a proto by neplatila rovnost). Díky této rovnosti víme, že jsou aplikovatelné i akce po cyklu (stavy obou trajektorií jsou si rovny).

Z aplikovatelnosti všech akcí plánu Π' společně s tím, že $s_G \subseteq \gamma(s_I, \Pi')$, víme, že Π' je validní. Tedy množina $I = \{a_{\iota_1}, \ldots, a_{\iota_m}\}$ je redundantní v Π.

□

5.2 Integrace

Díky tomu, že inverzní cyklus popisuje množinu redundantní akcí, můžeme na tomto poznatku založit detekci redundantních akcí, jako to udělala metoda IAE. My ale využijeme poznatku zmíněného v důkazu tvrzení 5.1. Jestliže po provedení inverzního cyklu máme stejný stav, jako když akce z cyklu ignorujeme, tak víme, že nepřítomnost akcí cyklu nemůže ovlivnit žádné následující akce.

Na základě toho můžeme optimalizovat algoritmy *Action Elimination* (AE) a *Greedy Action Elimination* (GAE). Při hledání množin inverzních akcí budeme kontrolovat, zda odebrané akce netvoří cyklus. Pokud ano, tak víme, že tvoří množinu redundantní akcí a také, že neovlivní žádné další akce. Díky tomu již nemusíme pokračovat v hledání dalších akcí, které by do oné skupiny mohly ještě patřit, můžeme vyhledávání ukončit a akce označit jako redundantní.

5.2.1 Úplná detekce

Nejpřímějším přístupem je rozšíření obou metod o další stav, ve kterém budeme počítat původní trajektorii a kontrolovat ji se stavem vznikajícím bez akcí z cyklu. Abychom omezili počet porovnávání stavů, jakožto složité operace, držíme si množinu přiřazení m, jenž sleduje, jak akce z cyklu mění stavy původní trajektorie. S její pomocí můžeme kontrolu rovnosti stavů obou trajektorií provést pouze ve chvíli, kdy jsou všechna přiřazení z této množiny splněna i ve stavu vznikajícím ignorování akcí z inverzního cyklu.

Tento přístup jsme implementovali. Bohužel čas potřebný k vypočítávání druhého průběžného stavu významně převyšoval časovou úsporu získanou ořezáváním, a to u obou metod AE a GAE. Dále jsme se pokusili tento přístup zrychlit tím, že si předpočítáme trajektorii stavů, kterou si budeme chytře přepočítávat pouze, když bude třeba. S tímto přístupem jsme snížili časovou náročnost, a dokonce jsme výjimečně dosáhli i lepšího času než metoda, jenž popíšeme v následující podsekci (a která časově předčila obě původní metody AE a GAE). Pseudokódy, ani detailní výsledky pro tyto přístupy neuvádíme. Pouze v sekci 8.4.1 shrneme poznatky popsané v této kapitole v číslech.

5.2.2 Částečná detekce

S vědomím, že se nevyplatí trávit výpočetní čas udržováním trajektorie původního plánu, jsme ustoupili z požadavku jisté detekce inverzního cyklu. Namísto toho jsme přistoupili, podobně jako v metodě IAE, k obecnější podmínce, která způsobí, že ne všechny cykly identifikujeme, ale ušetříme tím výpočetní čas, a to převážně v případech, kdy daná skupina ověřovaných akcí netvoří inverzní cyklus.

Našim cílem bude prokázat rovnost stavů bez toho, abychom museli stav z původní trajektorie počítat. Podobně jako u původní metody i zde použijeme množinu přiřazení m, která bude sledovat, jaké proměnné by ověřované akce změnily při aplikaci původního plánu. Pokud budou později všechny změny splněné i ve stavu z nové trajektorie, naše

ověřované akce mohou tvořit inverzní cyklus. V takovém případě nám zbývá zkontrolovat, zda jsou proměnné, jenž ověřované akce neměnily, stejné v obou stavech.

K ověření zbylých proměnných již nemůžeme použít porovnání stavů, protože stav původní trajektorie nemáme k dispozici. Místo toho zaručíme, že se hodnoty zbylých proměnných v původní trajektorii nemohou lišit od hodnot v nové trajektorii. To zajistíme tím, že se efekty zbylých mezilehlých (neověřovaných) akcí nezmění. Nepodmíněné efekty budou jistě stejné, protože akce musí být aplikovatelné (jinak by byly označeny a ignorovány). Proto jediné, co musíme zajistit, je shodnost podmíněných efektů. To zaručíme tím, že zakážeme jakékoli změny proměnných, které mají neověřované akce v předpokladech podmíněných efektů. Právě v tomto místě se vzdáváme toho, že nalezneme všechny inverzní cykly. Akce totiž proměnné z podmínek podmíněných efektů měnit mohou, pokud tím nezpůsobí aktivaci efektu (podmínkou může být nadále nesplněná množina přiřazení) nebo pokud bude podmíněný efekt později zvrácen.

Na rozdíl od přístupu v IAE můžeme povolit, že mezilehlá akce může měnit ovlivněné proměnné. Můžeme tak učinit, protože máme k dispozici novou trajektorii. S její pomocí můžeme snadno odhalit případy, kdy je proměnná mezilehlými akcemi sice změněna, ale později je navrácena na původní hodnotu. Takový případ nám cyklus nenarušuje. V případě, že navrácena není, cyklus nemáme, což odhalíme pomocí kontroly, zda je každá změněná proměnná splněná ve stavu z nové trajektorie (to v takovém případě splněno nebude).

Implementovali jsme jak variantu s tímto povolením, tak bez něj. V sekci 8.4.1 ukážeme, že jestliže tyto změny povolíme, pak významně zvýšíme dobu výpočtu na námi zvolených doménách a problémech. V nich totiž k navrácení hodnot z pravidla nedochází, a proto při změně ovlivněné proměnné dojde k nenávratnému porušení cyklu. Varianta, která toto zakazuje, zjistí, že cyklus nenalezne mnohem dříve, než varianta, jenž stále "doufá" v to, že hodnota proměnné se jednou změní zpět.

Posledním problémem, s nímž si musíme poradit kvůli nepřítomnosti stavu z původní trajektorie, je jakým způsobem zjistit podobu množiny $\mathit{eff}(a_i, s_i)$ (kde a_i ověřovaná akce, s_i je stav původní trajektorie a s'_i je stav trajektorie nové). Pomocí této množiny totiž zjišťujeme, jaké efekty měla akce a_i v původní trajektorii ve stavu s_i, a tedy jaké jsou ovlivněné proměnné a jejich hodnoty. Víme, že s_i je vlastně s'_i, ale beze změn ovlivněných proměnných.

Stav s'_i bychom tedy mohli sestavovat z těchto dvou množin. To by byl ale krok zpátky. Stav jsem měli v úplné detekci vypočtený průběžně a nevedlo to na uspokojivé výsledky. Proto i tu musíme ustoupit z nároku na optimalitu a spokojit se s tím, že víme, v jakém případě si musí efekty být efekty $\mathit{eff}(a_i, s'_i)$ a $\mathit{eff}(a_i, s_i)$ rovny a kdy ne.

Rovnost množin efektů $\mathit{eff}(a_i, s'_i)$ a $\mathit{eff}(a_i, s_i)$ zaručíme stejně jako u neověřovaných akcí. Pokud zakážeme změny v proměnných, které jsou součástí předpokladů podmíněných efektů, víme, že rozdílnost nenastane.

Nyní stačí poznatky z předchozích odstavců integrovat do vnitřního cyklu obou metod AE a GAE. Úpravy, jenž transformují metodu AE v *Action Elimination* s detekcí cyklů (AECD) jsou uvedeny v algoritmu 8. Pseudokód pro metodu *Greedy Action Elimination* s detekcí cyklů (GAECD) neuvedeme. Úpravy pro *Greedy Action Elimination with Cycle Detection* (GAECD) jsou analogické úpravám v metodě *Action Elimination with Cycle Detection* (AECD).

5.3 Algoritmická složitost

Oproti optimalizaci s lokálními landmarky není pro optimalizaci pomocí inverzních cyklů třeba žádných dalších algoritmů. Veškeré výpočty se provádějí za běhu a pouze v rámci dané akce. Proto se asymptotická složitost, v níž figuruje pouze délka plánu, nezmění a bude stejná, jako u původních variant metod.

Algorithm 8: Action Elimination with Cycle Detection

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, initial state s_I, goal state s_G
Out : a reduced plan

```
1  removed ← ∅;
2  marks ← ∅;
3  s ← s_I;
4  for i ← 1 to n do
5      if i ∉ removed then
6          marks ← marks ∪ {i};
7          s' ← s;
8          cycle ← false;
9          violated ← false;
10         changed ← eff(s', a_i);
11         for j ← i + 1 to n do
12             if j ∉ removed then
13                 if not violated and
                    ∃(v, val) ∈ changed, ∃c ∈ pre_cond(a_j) : v ∈ vars(c), (v, var) ∉ c then
14                     violated = true;
15                 if a_j is not applicable in s' then
16                     marks ← marks ∪ {j};
17                     if not violated then
18                         changed ← {(v, val) | v ∉ eff(a_j, s')} ∪ eff(a_j, s');
19                         if changed ⊆ s' then
20                             cycle ← true;
21                             break;
22                 else
23                     if not violated then
24                         if ∃(v, val) : (v, val) ∈ eff(a_j, s'), v ∈
                            vars(changed), (v, val) ∉ changed then
25                             violated ← true;
26                     s' ← γ(s', a_j);
27         if cycle or s_G ⊆ s' then
28             removed ← removed ∪ marks;
29         else
30             s ← γ(s, a_i);
31         marks ← ∅;
32 Π ← ⟨a_i | a_i ∈ Π, i ∉ removed⟩;
33 return Π;
```

Kapitola 6

Kombinace obou přístupů

Dosud jsme popsali dva optimalizační přístupy. Přirozenou myšlenkou je vyzkoušení jejich kombinace a zda-li je jejich kombinace vůbec možná. Ukážeme, že v našem případě se nejedná o problém.

6.1 Kompatibilita

Optimalizace pomocí lokálních landmarků nám říká, kterou akci nemůžeme v žádném případě odebrat. Takové akce přeskakujeme, pokud popřípadě došlo k narušení aplikovatelnosti, přerušujeme ověřování. Na druhou stranu optimalizace založená na inverzních cyklech nám identifikuje skupiny akcí, u nichž nemusíme ověřovat redundanci dál, protože jsme si již v průběhu ověřování jistí, že je skupina redundantní a žádná další akce do ní nepatří. Proto můžeme přístupy snadno zkombinovat a dosáhnout ještě silnější optimalizace.

U obou metod nejprve zkontrolujeme, zda je akce a_j z vnitřního cyklu lokální landmark. Pokud není, můžeme přistoupit k dalšímu kroku optimalizace, kde zkontrolujeme, zda nedošlo k porušení podmínek pro detekci cyklu. Následně provedeme kontrolu aplikovatelnosti, tak jako u původních metod. Pokud ale není aplikovatelná, upravíme si množinu změněných přiřazení a zkontrolujeme cyklus.

6.2 Integrace

Stejně jako v kapitole 5, i zde uvedeme pouze algoritmus rozšiřující metodu *Action Elimination* (AE), neboť rozšíření metody *Greedy Action Elimination* (GAE) je analogické. Rozšíření metody AE znázorňuje algoritmus 9.

6.3 Algoritmická složitost

Stejně jako u obou předchozích přístupů, ani jejich kombinace nenavýší (ani nesníží) asymptotickou složitost optimalizovaných algoritmů.

Algorithm 9: Action Elimination with Landmark Skip and Cycle Detection

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, initial state s_I, goal state s_G
Out : a reduced plan

```
1  removed ← ∅;
2  marks ← ∅;
3  landmarks ← landmarkSearch(Π, s_I, s_G);
4  s ← s_I;
5  for i ← 1 to n do
6      if i ∉ removed then
7          if i ∉ landmarks then
8              marks ← marks ∪ {i};
9              s' ← s;
10             cycle ← false;
11             violated ← false;
12             changed ← eff(s', a_i);
13             for j ← i + 1 to n do
14                 if j ∉ removed then
15                     if not violated and
                        ∃(v, val) ∈ changed, ∃c ∈ pre_cond(a_j) : v ∈ vars(c), (v, var) ∉ c
                        then
16                         violated = true;
17                     if a_j is not applicable in s' then
18                         if j ∉ landmarks then
19                             marks ← marks ∪ {j};
20                             if not violated then
21                                 changed ← {(v, val) | v ∉ eff(s', a_j)} ∪ eff(s', a_j);
22                                 if changed ⊆ s' then
23                                     cycle ← true;
24                                     break;
25                         else
26                             break;
27                     else
28                         if not violated then
29                             if ∃(v, val) : (v, val) ∈ eff(a_j, s'), v ∈
                                vars(changed), (v, val) ∉ changed then
30                                 violated ← true;
31                         s' ← γ(s', a_j);
32             if cycle or s_G ⊆ s' then
33                 removed ← removed ∪ marks;
34             else
35                 s ← γ(s, a_i);
36             marks ← ∅;
37         else
38             s ← γ(s, a_i);
39 Π ← ⟨a_i | a_i ∈ Π, i ∉ removed⟩;
40 return Π;
```

Kapitola 7

Greedy Chaining Action Elimination

7.1 Závislosti redundantních množin

Jedna z největších slabin metody *Greedy Action Elimination* (GAE) je opětovný výpočet redundantních množin. Algoritmus v první iteraci spočte až n množin redundantních akcí. Následně z nich vybere tu s největší cenou, odebere ji a celý výpočet se opakuje. Algoritmus obecně provádí $n - a$ výpočtů ohodnocující smyčky při každé iteraci, kde a je počet již odebraných akcí. Proto se může snadno stát a to hned v případě, že plán obsahuje dvě množiny redundantních akcí začínajících jinou akcí, že je některý výpočet množiny zbytečně opakován. Pokud by si ale algoritmus skupiny zapamatoval, tak za předpokladu, že jsou všechny odebratelné současně, nemusel by je počítat znovu a stačilo by mu pro běh pouze n výpočtů ohodnocující smyčky. Tento nedostatek se u metody *Action Elimination* (AE) neobjevuje díky její jednoduchosti. Na druhou stranu se pomocí hladového přístupu metody GAE dají nalézt (a jak později ve výsledcích experimentů ukážeme, i nalézají) lepší plány.

Jestliže bychom uměli ukázat, které skupiny jsou na sobě nezávislé, a proto jsou odebratelné současně, mohli bychom je spojit v jednu skupinu a odebrat je najednou, čímž bychom se vyhnuli opětovnému výpočtu pro druhou skupinu. Přesně na této myšlence je založena metoda *Greedy Chaining Action Elimination.*

Metoda vychází z toho, že právě inverzní cykly nám tuto informaci poskytují. Máme-li totiž dvě množiny redundantních akcí, z nichž je jedna skupina inverzní cyklus a všechny akce cyklu jsou před akcemi druhé skupiny, pak cyklus tuto množinu nemůže ovlivnit. Díky tomu můžeme cyklus "přilepit" na následující množinu redundantních akcí a vytvořit tak jednu větší množinu.

Tvrzení 7.1. *Nechť $T = (X, O, s_I, s_G)$ je plánovací úloha a $\Pi = \langle a_1, \ldots, a_n \rangle$ je validní plán příslušný téže plánovací úloze T.*
Pokud $I = \langle a_{\iota_1}, \ldots, a_{\iota_m} \rangle$ je inverzní cyklus, $P = \{a_{\rho_1}, \ldots, a_{\rho_o}\}$ je množina redundantních akcí a $\forall \rho_i \in P : \iota_m < \rho_i$, pak $P \cup \{a_{\iota_i} \mid a_{\iota_i} \in I\}$ je množina redundantních akcí v Π.

Důkaz. Pokud je $P = \emptyset$, důkaz je triviální. Dále předpokládáme, že $P \neq \emptyset$.

Musíme ukázat, že obě množiny P a $\{a_{\iota_i} \mid a_{\iota_i} \in I\}$ jsou redundantní v plánech bez akcí z druhé skupiny. Z toho bychom věděli, že nezáleží na pořadí, v kterém skupiny odebereme jednotlivě, a proto je můžeme odebrat najednou.

Vezměme plán $\Pi_I = \Pi \setminus P$. Z důvodu, že trajektorie plánů Π a Π_I je shodná alespoň do stavu $s_{min(\{\rho_1,\ldots,\rho_o\})}$ a všechny akce z I předcházejí libovolné akci z P, je posloupnost I inverzní cyklus i pro plán Π_I. Protože I inverzní cyklus pro Π_I a Π_I je validní, tak I je redundantní v Π_I (Tvrzení 5.1).

Vezměme plán $\Pi_P = \Pi \setminus I$. Z definice inverzního cyklu v trajektorii plánu Π_P nalezneme stav s_{ι_m+1} z původní trajektorie Π, od kterého jsou obě trajektorie plánů Π a Π_P shodné až do konce. Protože inverzní cyklus I neměl na tento stav stav vliv, nemůže ho mít ani na žádný další. Redundance množiny P je zároveň určena stavem $s_{min(\{\rho_1,\ldots,\rho_o\})}$, ve kterém je aplikována nejdřívější akce $a_{min(\{\rho_1,\ldots,\rho_o\})}$ z P. Protože $\iota_m < min(\{\rho_1,\ldots,\rho_o\})$, tak $\iota_m + 1 \leq min(\{\rho_1,\ldots,\rho_o\})$. Jinými slovy stav, od kterého jsou trajektorie shodné, je shodný s, nebo předchází stavu, na kterém je závislá redundance množiny akcí P v plánu Π_P. Proto je P redundantní v Π_P.

□

7.2 Integrace

Nyní máme efektivní nástroj ke slučování cyklů s libovolnými následujícími množinami redundantních akcí a můžeme ho využít k částečnému eliminování opakování výpočtu v metodě GAE. Metoda *Greedy Chaining Action Elimination* (GCAE) je pozměněná GAE tak, aby ověřovala skupiny od konce plánu a mohla tím využít principy dynamického programování. Pokud totiž při ověřování detekujeme cyklus, můžeme ho "přilepit" na libovolnou množinu redundantních akcí, jejíž nejdřívější akce následuje až po poslední akci cyklu.

K tomu použijeme pole, do kterého si budeme pro každou počáteční akci ukládat množinu redundantních akcí, která má největší úhrnnou cenu. V případě, že nalezneme inverzní cyklus, sloučíme ho s nejlepším výsledkem, který lze získat za jeho koncem, a v případě, že je toto sloučení hodnotnější než dosavadní nejlepší výsledek, prohlásíme ho za novou nejlepší množinu.

Podotýkáme, že v algoritmu 10 je integrována i optimalizace pomocí lokálních landmarků, která není nezbytnou součástí metody. Při porovnávání metod na to bude třeba brát ohled. I z toho důvodu na řádku 13 referujeme na algoritmus ohodnocení skupiny z metody GAE s oběma optimalizacemi (jenž v práci není přímo uveden).

Dalším věc, kterou je třeba uvést na pravou míru, je podmínka na řádku 27 a její větve. O problematice, kterou tyto řádky řeší, jsme se již zmínili v kapitole s cykly u úplné metody. Z důvodu, že metoda GCAE zkoumá množiny od konce, musíme zajistit rozumný výpočet stavu, ve kterém má kontrola započít. O to se stará řádek 3, kde dojde k zavolání algoritmu (jehož pseudokód neuvádíme), který vrátí trajektorii plánu. Algoritmu, který se stará o výpočet skupiny, poté předáme tento stav a máme tak zaručeno, že stejně jako u metody GAE spočteme každý stav pouze jednou za iteraci. Nicméně díky tomu, že pracujeme s cykly a stavy máme na rozdíl od GAE uložené, nemusíme je vždy počítat všechny znovu. Stačí přepočítat ty, které následují po poslední akci v odebírané množině. V případě, že odebíraná množina redundantních akcí je inverzní cyklus, můžeme opětovný výpočet stavů zastavit při dosažení poslední akce z cyklu (nově vypočtený následující stav by byl shodný s původním). Přesně o toto přepočítání stavů se stará metoda *updatePlanTrajectory* (jejíž pseudokód opět neuvádíme) ve větvích podmínky na řádku 27.

7.3 Srovnání metod

Tak, jak jsme definovali metodu GCAE, jsme dosáhli potenciální úspory výpočetního času. Co ale nemusí být zřejmé, že metoda GCAE obecně nedosahuje stejných výsledků jako GAE, či AE. Tím, že chamtivě přidáváme inverzní cykly před následující množiny redundantních akcí, docílíme chování, které může dosáhnout horšího či lepšího výsledku než metoda GAE.

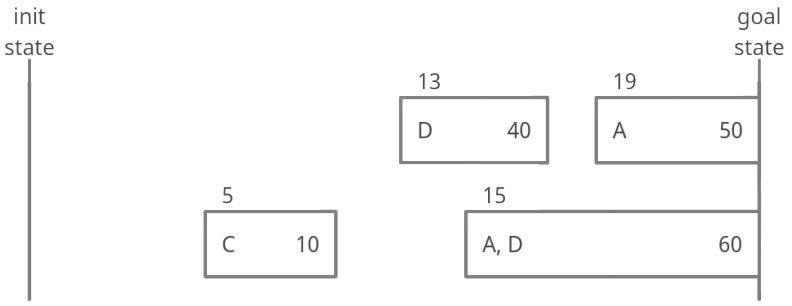
Jedním z případů, kde je metoda GAE dominována metodou GCAE, vidíme na obrázku 7.1. Pokud se tento plán pokusíme optimalizovat metodou GAE, algoritmus nejprve

Algorithm 10: Greedy Chaining Action Elimination with Landmark Skip

Input: plan $\Pi = \langle a_1, \ldots, a_n \rangle$, initial state s_I, goal state s_G
Out : a reduced plan

```
1  removed ← ∅;
2  landmarks ← landmarkSearch(Π, s_I, s_G);
3  states ← getPlanTrajectory(Π, removed, s_I);
4  bests ← empty map;
5  do
6      bestIndex ← 0;
7      bestCost ← 0;
8      bestCycleDetected ← true;
9      bestMarks ← ∅;
10     for i ← n to 1 do
11         if i ∉ removed then
12             if i ∉ landmarks then
13                 cost, cycleDetected, marks ←
                    evaluateRemoveLSCD(Π, s_i, k, s_G, removed, landmarks);
14                 if cycleDetected and max(marks) + 1 ≤ n then
15                     previousCost, previousCycleDetected, previousMarks ←
                        bests[max(marks) + 1];
16                     cost ← cost + previousCost;
17                     cycleDetected ← previousCycleDetected;
18                     marks ← marks ∪ previousMarks;
19                 if cost ≥ bestCost then
20                     bestIndex ← i;
21                     bestCost ← cost;
22                     bestCycleDetected ← cycleDetected;
23                     bestMarks ← marks;
24         bests[i] ← (bestCost, bestCycleDetected, bestMarks);
25     if bestIndex ≠ 0 then
26         removed ← removed ∪ bestMarks;
27         if bestCycleDetected then
28             updatePlanTrajectory(states, Π, removed, min(bestMarks),
                max(bestMarks));
29         else
30             updatePlanTrajectory(states, Π, removed, min(bestMarks), n);
31 while bestIndex ≠ 0;
32 Π ← ⟨a_i | a_i ∈ Π, i ∉ removed⟩;
33 return Π;
```

nalezne všechny množiny a odebere tu, jejíž úhrnná cena je nejvyšší, tedy šedesát. Následně provede výpočet znovu (v redukovaném plánu) a nalezne pouze poslední skupinu, začínající pátou akcí. Proto je úhrnná cena odstraněných akcí rovna sedmdesáti.



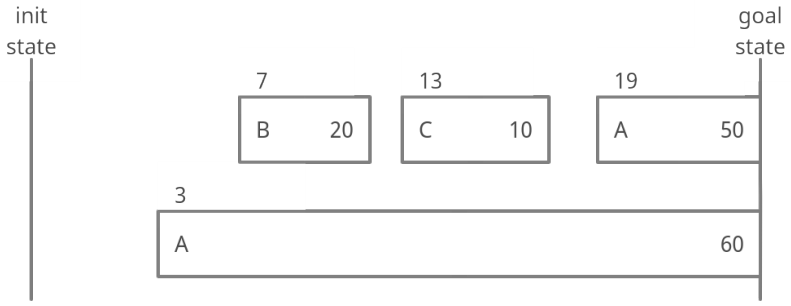
Obrázek 7.1: schéma znázorňující slabinu metody *Greedy Action Elimination*

Pustíme-li na stejný plán metodu GCAE, dojde k zcela odlišnému chování. Algoritmus postupuje od konce plánu. Nejprve proto nalezne množinu s úhrnnou cenou padesát. Poté pokračuje dále a nalezne množinu začínající patnáctou akcí. Ta je lepší než množina začínající akcí devatenáctou, a proto si změní nejlepší doposud nalezený výsledek na šedesát. Když ale metoda dojde k třinácté akci a spočte si hodnotu množiny redundantních akcí začínajících touto akcí, zjistí, že nalezla inverzní cyklus s úhrnnou cenou čtyřicet. To je sice méně než dosavadních šedesát, ale metoda může k tomuto cyklu "přilepit" nejlepší možný nález začínající za svou poslední akcí. Taková množina je tam v našem případě pouze jedna a má úhrnnou cenu padesát. To dohromady se čtyřiceti z právě nalezeného cyklu dá více, než je dosavadní nejlepší výsledek, a proto si poznamená nové nejlepší řešení. Naposled se změní nejlepší výsledek při nalezení inverzního cyklu začínajícího pátou akcí, který se v našem případě přidá k dosud nejlepšímu výsledku. Při prvním průchodu tedy metoda GCAE nalezne množinu redundantních akcí, která je sloučením množin začínajících akcemi pět, třináct a devatenáct, jejíž úhrnná cena je sto. V další iteraci již žádnou další redundantní množinu nenajde a ukončí svůj běh. Nalezené řešení je lepší než řešení metody GAE.

Zmíníme, že na plánu ze schématu na obrázku 7.1 je efektivnější i metoda AE. Ta díky svému naivnímu přístupu odebere stejné množiny jako metoda GCAE, pouze "opačně". Nejprve nalezne množinu s cenou deset, poté množinu s cenou čtyřicet, jejímž odebráním způsobí, že množina začínající patnáctou akcí již nebude nalezena, a naposled množinu začínající akcí devatenáctou. Ve výsledku metoda AE odebere množiny redundantních akcí s úhrnnou cenou sto, stejně jako metoda GCAE.

Může dokonce nastat situace, kdy bude výkon metody AE největší. Pokud do schématu z obrázku 7.1 přidáme ještě jednu množinu redundantních akcí (která není inverzní cyklus) začínající třetí akcí, s jedinou skupinou vzájemného vyloučení C a cenou 20, pak metoda GAE odstraní akce v úhrnné ceně osmdesát, metoda GCAE sto a metoda AE dokonce sto deset. Stejně tak může nastat situace, kdy GCAE předčí obě původní metody. Jestliže změníme úhrnnou cenu přidávané množiny z dvaceti na pět, pak metoda GAE odstraní akce v úhrnné ceně sedmdesát, metoda AE devadesát pět a nová metoda GCAE sto.

Jak již intuice napovídá, jistě také existují i situace, kdy je metoda GCAE horší než obě metody AE a GAE. Bez hlubšího popisu situací, které nastanou při aplikaci metod na plán ze schéma z obrázku 7.2. Metody AE a GAE naleznou řešení s úhrnnou cenou devadesát, zatímco metoda GCAE pouze s cennou osmdesát.



Obrázek 7.2: schéma znázorňující slabinu metody *Greedy Chaining Action Elimination*

7.4 Další výzkum

Jak jsme ukázali v předchozím odstavci, všechny zmíněné metody obsahují nedostatky, kvůli kterým je mohou předčit i méně sofistikované metody. Poukázali jsme na fakt, že žádná metoda není dominována výkonem jiné metody. Příčinou tohoto jevu je, že všechny metody odstraňují množiny redundantních akcí (víceméně) naslepo. Metody AE a GAE se o skupiny vzájemného vyloučení nestarají vůbec a metoda GCAE pouze částečně. Proto se může stát, že metoda v dobré víře odebere, dle svých pravidel, "nejlepší" množinu, čímž ale zabrání dalšímu (a úhrnně hodnotnějšímu) odebrání jiných množin.

Díky informacím a poznatkům, které nám přinesla metoda *Inverse Action Elimination* (IAE) a ní inspirovaný optimalizační přístup pomocí inverzních cyklů, můžeme některé souvislosti množin velice snadno vyloučit. To inspirovalo využití této informace při hledání množin redundantních akcí v metodě GCAE.

V dalším výzkumu v této oblasti bychom se rádi zaměřili na hlubší prozkoumání souvislostí množin redundantních akcí a jejich případného užití v další optimalizační metodě.

Kapitola 8

Experimenty

8.1 Prostředí

Zkoumané metody byly implementovány v jazyce *C++*. Experimenty běželi na stroji se čtyřjádrovým procesorem AMD A10-6800K Black Edition (4,1 GHz, L2 Cache 4 MB) a 12 GB RAM (1333 MHz). Plánovače a optimalizační metody běžely dle pravidel agilní části *International Planning Competition* (IPC) 2018 (Pommerening; Torralba; Balyo 2018a), tedy pouze na jednom jádře a s maximálně 8 GB RAM po dobu maximálně 5 minut.

8.2 Plánovače

Všechny použité plánovače (až na plánovač *YAHSP3*) se účastnili agilní části soutěže IPC 2018. Plánovač *YAHSP3* se účastnil soutěže IPC 2014 (Chrpa; Vallati; L. McCluskey 2014a) (kterou i vyhrál). Kódy byly staženy přímo z repozitáře soutěží a následně integrovány do našeho optimalizačního prostředí.

Mimo následně zmíněných plánovačů jsme využili překladač, jenž je součástí *Fast Downward* plánovače (Helmert 2006). Použili jsme ho pro překlad domén ze *STRIPS* (Fikes; Nilsson 1971) do rozšířeného SAS^+ formalismu (Helmert 2006), se kterým pracuje naše implementace.

8.2.1 Plánovač LAMA[1]

Algoritmus nejprve nalezne řešení pomocí *Greedy Best-First Search*, a poté spouští iterativně *Weighted* A^*, dokud nemá dojít k ukončení hledání. V agilním části soutěží dojde k pouze vyhledání prvního plánu. Při prohledávání se využívají dvě fronty, z nichž má každá svoji heuristiku: heuristiku orientačních bodů a upravenou *FF* heuristiku (Richter; Westphal 2010).

Plánovač se umístil na "druhém" místě (byl použit jako referenční plánovač) agilní části soutěže IPC 2018.

8.2.2 Plánovač LAPKT-BFWS-Preference[2]

Polynomiální algoritmus *Best-First Width Search*, který v roce 2017 překonal *state-of-the-art* algoritmus *LAMA* (Richter; Westphal 2010), obnovil myšlenku využití novosti stavů v prohledávání. Jeho varianta *BFWS-Preference* v každém kroku expanduje stav, který má nejnižší *novelty* (velikost nejmenší ještě nedosažené $n$tice v totožných (ostatních) heuristických hodnotách) (Francès; Geffner; Lipovetzky; Ramírez 2018) a v případě

[1] Richter; Westphal 2010.
[2] Francès; Geffner; Lipovetzky; Ramírez 2018.

rovnosti se rozhoduje pro ten, který má nižší počet nedosažených proměnných cílového přiřazení. Tento plánovač se stal vítězem IPC 2018.

8.2.3 Plánovač Freelunch Madagascar[3]

Tento plánovač hledá řešení plánovací úlohy převodem na úlohu splnitelnosti booleovské formule. Při překladu dojde ke snaze vyřešit plánovací úlohu heuristicky. V případě, že tato snaha selže, se hledá řešení *SAT* problému.

8.2.4 Plánovač Cerberus[4]

Vylepšená verze plánovače *Mercury* (Katz; Hoffmann 2018), který se účastnil soutěže IPC 2014, kde i přes ignorování podmíněných proměnných dosáhl zajímavých výsledků.

Cerberus i *Mercury* při prohledávání využívají *red-black* heuristiku, která je jakýmsi kompromisem mezi delete-relaxací problému a problémem samotným.

Cerberus se také inspiroval metodou *LAPKT-BFWS-Preference* (Francès; Geffner; Lipovetzky; Ramírez 2018), kde se krom jiných kritérií využívá pro prohledávání i *novelty* s *red-black* heuristikou.

8.2.5 Plánovač YAHSP3[5]

Plánovač *YAHSP3* využívá predikce stavového prostoru na základě analýzy relaxované úlohy. Plánovač analyzuje doménu a rozdělí akce do dvou skupin - akce *užitečné* a *záchranné*. Následně jsou při výběru stavu k expanzi preferované ty stavy, které "obsahují" *užitečné* akce, a to i proti stavům s nižší heuristickou hodnotou. V roce 2014 *YAHSP3* vyhrál agilní část IPC.

8.3 Domény

Pro provedení experimentů jsme vybrali šestnáct domén ze soutěží IPC. Bez hlubšího popisu domén se odkážeme na stránky soutěží. Z ročníku 2018 jsme použili veškeré domény (Pommerening; Torralba; Balyo 2018b). Ze starších soutěží jsme vybrali domény *Elevators* z ročníku 2008 (Helmert; Do; Refanidis 2009) a *Barman*, *Hiking* a *Tetris* z ročníku 2014 (Chrpa; Vallati; L. McCluskey 2014b).

8.4 Výsledky

Kompletní výsledky experimentů spolu s detailnějšími tabulkami jsou uvedeny v přiložených souborech práce (příloha B).

Pro provedení experimentů bylo použito 330 úloh z výše zmíněných domén. Na každou z nich byly spuštěny plánovače. Ty nalezly celkem 525 plánů (v ostatních 1125 případech jim došly zdroje, či plánování skončilo chybou, a byly ukončeny). Výsledných 525 plánů bylo následně optimalizováno. U každého běhu byl měřena původní cena a délka plánu, cena a délka optimalizovaného plánu, čas využitý k plánování a čas využitý k optimalizování.

V následujících odstavcích budeme často uvádět zaokrouhlené hodnoty.

[3] Balyo; Gocht 2018.
[4] Katz 2018.
[5] Vidal 2014.

8.4.1 Srovnání úplné a částečné detekce inverzních cyklů

Varianta částečné detekce, v níž jsme povolili změny ovlivněných proměnných, dosáhla na experimentech neuspokojivého výsledku. Stejně tak metoda úplného přístupu.

Výpočty původní metody *Action Elimination* (AE) (skrze všechny níže popsané problémy a domény) trvaly 50,522 sekundy. Přístup s úplnou detekcí spotřeboval více, konkrétně 52,698 sekund. Přístup částečné detekce (s povolením) se umístil mezi těmito dvěma výsledky s časem 51,481 vteřin. V 79,96 % případů je metoda částečné detekce (s povolením) rychlejší než metoda úplné detekce. Oproti tomu původní metoda AE je rychlejší v 69,13 % případů než varianta s částečnou detekcí (s povolením) a v 80,32 % případů než varianta s úplnou detekcí. "Optimalizace", na námi zkoumaných úlohách, nejsou efektivní.

U metody *Greedy Action Elimination* (GAE) je tomu podobně. Původní metoda spotřebovala 1427 vteřin. Integrace metody úplné detekce opět dosáhla nejvyššího času 1644 vteřin. Běh integrace přístupu částečné detekce (s povolením) trval 1577 sekund a byla rychlejší než úplná detekce v 74,91 % případů. Stejně jako u integrací do AE, i pro metodu GAE nebyla optimalizace úspěšná.

Jak jsme již zmínili, přístup částečné detekce se zakázáním jakýchkoli změn si vedl lépe. Oproti přístupu úplné detekce běžel při integraci do metody GAE pouze 866 vteřin (resp. 43,892 vteřin při integraci do AE), tj. 54,45 % (resp. 89,57 %) času přístupu úplné detekce, rychlejší byl v 73,71 % (resp. v 80 %) případů, a jeho výpočet průměrně zabral 85,45 % (resp. 82,07 %) času úplného přístupu. Proto, jestliže budeme následně mluvit o částečné detekci, máme tím na mysli (nespecifikujeme-li jinak) přístup se zákazem změn, tak jak jsme ji uvedli v kapitole 5.

8.4.2 Efektivita

Nejprve popíšeme výsledky z pohledu efektivity optimalizace plánu. Podotýkáme, že každá optimalizace dosáhla stejného výsledku jako metoda původní a nenarušili jsme ní vlastnosti původní metody.

Potvrdili jsme, že metoda *Greedy Action Elimination* dosahuje na doménách soutěže IPC (Pommerening; Torralba; Balyo 2018a; Chrpa; Vallati; L. McCluskey 2014a; Helmert; Do; Refanidis 2010) obecně lepší výsledků než AE, a to i na doménách soutěže IPC 2018 (Pommerening; Torralba; Balyo 2018a). Metoda GAE (a i její optimalizace) odebrala akce v celkové ceně 17772, což odpovídá průměrně 6,71 % ceny optimalizovaného plánu, zatímco AE jen 15964, odpovídajících 6,46 %.

Námi navrhnutá metoda *Greedy Chaining Action Elimination* (GCAE) bohužel na použitých problémech nedosáhla tak kvalitní optimalizace jako GAE. Celkem nalezla množiny redundantních akcí v úhrnné ceně 16344, tj. průměrné zlepšení plánu o 6,56 %.

Letmo jsme experimentovali i s možnými modifikacemi metody GCAE tak, abychom se přiblížili efektivitě GAE. Dospěli jsme k zajímavému výsledku. Jestliže v GCAE nahradíme hledání skupiny s nejvyšší úhrnnou cenou hledáním skupiny s nejdražší akcí, pak můžeme dosáhnout zcela odlišného výsledku. Na námi zkoumaných problémech jsme touto úpravou dosáhli odebrání akcí v ceně 17271 (průměrně zlevnila plán o 6,64 %), což je asi o 5 % více než původní metoda. Rádi bychom tento přístup více prozkoumali v další práci, neboť si myslíme, že by to mohlo vést k více zajímavým výsledkům.

Posledním zkoumaným přístupem je rozšíření metody *Inverse Action Elimination* (IAE). Ta, stejně jako původní nerozšířená varianta, nedosahuje významné efektivity. Odebrala množiny akcí, jejichž úhrnná cena je nejmenší ze všech metod, konkrétně 9511 a zlevnila tak v průměru plán o 3,87 %.

V kapitole 7 jsme se zmínili o neporovnatelnosti metod AE, GAE a GCAE, neboť žádná z jich není obecně lepší než jiná. Jak z našich výsledků vyplývá, metoda GAE si vedla na našich doménách nejlépe. Výsledky metody GAE byly metodou AE předčeny ve 4 případech a 477 si ze stránky výkonu byly rovny. Horší byla ve 44 případech. Metoda

GCAE byla o něco málo úspěšnější. Byla lepší také pouze ve 4 situacích, zato rovna metodě GAE byla ve 498 a horší jen ve 23 případech. Pokud porovnáme metody AE a GCAE, tak nejjednodušší metoda AE zvítězila v 7, remizovala ve 483 a horší byla v 35 případech. Experimentálně jsme tedy potvrdili poznatky z dané části kapitoly 7.

8.4.3 Výpočetní čas

Jak znázorňuje tabulka 1, námi implementované optimalizace byly obecně na zkoumaných problémech užitečné, přestože v určitých případech, primárně v situacích, kdy v plánu nebyly odhaleny žádné akce, mohlo dojít vylepšeným přístupem ke zpomalení původních metod.

Metoda *Action Elimination with Cycle Detection* (AECD) s částečnou detekcí cyklů při výpočtech spotřebovala 92,1 % času AE. U *Greedy Action Elimination with Cycle Detection* (GAECD) došlo k výraznějšímu zlepšení a to na 62,61 %. Jak u GAECD, tak u AECD došlo k zajímavému a jedinečnému výsledku. Přestože v celkovém srovnání je metoda rychlejší, tak při pohledu na konkrétní situace zjistíme, že je metoda AECD rychlejší než AE pouze na 37,52 % problémů. U dvojice GAECD a GAE je tomu obdobně, neboť je tato hodnota ještě nižší, konkrétně 27,62 %. V 72,38 % zkoumaných problémů metoda GAECD nevedla ke snížení výpočetního času. Zdá se, že metoda je na krátkých plánech pomalejší kvůli "zbytečné" režii a k výraznému zlepšení časů dojde při běhu na dlouhých plánech s velkým počtem redundantních akcí. Tyto dva přístupy jsou tímto ojedinělé. Žádné další přístupy, jenž měly snížit výpočetní čas k takto špatnému výsledku nedošly a všechny dosáhly zrychlení alespoň v 91 % případů. To potvrzuje i to, že geometrický průměr hodnot udávajících kolik % času metody AE (resp. GAE) spotřebuje metoda AECD (resp. GAECD) při výpočtu na témže plánu, je roven 110 (resp. 112) %. Maximální naměřená hodnota je 546 (resp. 403) %, tedy skoro pěti a půlnásobné (resp. čtyřnásobné) zpomalení. Největší zrychlení bylo 35 (resp. 42) %, tedy skoro trojnásobné (resp. dvou a půlnásobné) zrychlení. Žádná další metoda (krom úplné detekce) takhle malého největšího zrychlení nedosáhla.

Jak jsme zmínili v kapitole 4, varianty optimalizací s vylepšeným hledáním landmarků využívajících algoritmu 7 nedosáhly znatelně lepších výsledků oproti metodám založených na algoritmu 6. Metoda *Action Elimination with Enhanced Landmark Skip* (AELSe) spotřebovala 95,37 % času *Action Elimination with Landmark Skip* (AELS). U *Greedy Action Elimination with Enhanced Landmark Skip* (GAELSe) to bylo pouze 98,82 % času *Greedy Action Elimination with Landmark Skip* (GAELS). Změny v procentech vzhledem k časům původních metod byly pouze 1,99 % a 0,7 %. Metoda AELSe předčila metodu AELS v 84,57 % případů a metoda GAELSe metodu GAELS v 71,81 %.

Optimalizace s pomocí (vylepšených) lokálních landmarků si vedla lépe než optimalizace s cykly. Algoritmus *Action Elimination with Enhanced Landmark Skip* běžel pouze 41,12 % výpočetního času metody AE. U *Greedy Action Elimination with Enhanced Landmark Skip* už tak k velké změně oproti detekci cyklů už nedošlo. Výpočet trval 58,9 % času GAE. Na rozdíl od dvou předchozích odstavců, když porovnáme jednotlivé časy zjistíme, že metody jsou znatelně konzistentnější, jelikož jsou obě ve více jak 92 % rychlejší než původní metoda.

I kombinace obou optimalizačních přístupů přinesla ovoce. Integrace detekce cyklů a vylepšeného hledání lokálních landmarků do AE zabrala pouze 31,34 % a pro GAE 19,27 % časů příslušných metod. Varianta kombinace se slabšími landmarky dosáhla nepatrně horšího výsledku a to 33,26 % pro AE a 20,18 % pro GAE. I zde jsou metody velmi konzistentní ve své dominanci nad původními metodami. Nicméně oproti metodám AELSe a GAELSe zde došlo ke snížení počtu situací, kde došlo ke zrychlení. Kombinace obou metod dominuje AE ve 93,33 % případů, zatímco AELSe v 95,62 %. U variant GAE došlo k poklesu z 96,95 % na 93,71 %. Když porovnáme jednotlivé instance, tak metody kombi-

nované optimalizace byly (pouze u variant GAE) v nepatrné nadpoloviční většině případů pomalejší, ale v nejlepším případě dosahovaly až osminásobného zrychlení a v nejhorším pouze tří a půlnásobného zpomalení. I přes tuto skutečnost stále došlo k výraznému snížení celkového času z doposud nejlepšího času metod AELSe a GAELSe.

Metodu *Greedy Chaining Action Elimination* (GCAE) srovnáme s metodami, jenž kombinují oba přístupy, protože GCAE tak činí též. Z dat je bez zaváhání hned jasné, že metoda založená na AE a kombinující oba přístupy je v naprosté většině a i celkově rychlejší. Zajímavější situace nastává u porovnání časů GCAE a GAE s oběma optimalizacemi. Metoda GCAE sice spotřebovala pouze 24,48 % času srovnávané metody, ale byla rychlejší pouze ve 39,62 % případů. Na druhou stranu v nejlepším případě byla 25krát rychlejší a v nejhorším pouze 3krát pomalejší.

Metoda *Inverse Action Elimination* (IAE) má ze všech zkoumaných metod nejvyšší asymptotickou složitost. I přes tuto skutečnost je její celkový čas "kompromisem" mezi AE a GAE. Zajímavé je, že přestože metoda GAE běžela přibližně pětkrát déle než IAE, její nejsilnější optimalizace spotřebovala dokonce pouze 91,09 % času IAE a za ten dosáhla i dosud nejlepší optimalizace.

8.4.4 Metody a algoritmy

Nezdá se, že by použití některé metody na konkrétní algoritmus přineslo nějaké nové zajímavé výsledky, vyjma dvojice *Freelunch Madagascar* a optimalizační metody skupiny AE. V tomto případě si skupina AE vedla o 0,02 % cen plánů lépe, než GAE, a to 16,82 % oproti 16,8 % cen optimalizovaných plánů. Ve všech ostatních případech je stejné uspořádání metod dle efektivity jako u souhrnných výsledků. Pro plánovač *Freelunch Madagascar* použití metod skupiny AE přineslo úhrnně efektivnější výsledky než použití libovolných jiných metod.

Když se podíváme na ceny odebraných akcí u jednotlivých metod, zjistíme, že pro plánovač *Cerberus* byly plány v průměru optimalizovány o 2,41 %. Pro plánovač *LAPKT-BFWS-Preference* to bylo podobné, a to o 2,54 %. K dvojnásobné optimalizaci došlo u plánovače *LAMA-first*, kde optimalizační metody odebraly v průměru 5,47 %. Zbylé dva plánovače obsahují ještě hodnotnější množiny redundantních akcí, jenž byly odhaleny. Pro plánovač *Freelunch Madagascar* měly cenu 16,67 % a pro plánovač *YAHSP3* dokonce 23,35 % ceny původního plánu.

Rozdíly výkonů optimalizačních přístupů různých skupin jsou na všech plánech minimální, v jednotkách setin a desetin procent. Jen u plánovače *YAHSP3* je tento rozdíl roven 1,89 %.

Absolutní hodnoty optimalizovaných cen porovnávat nemůžeme. Ne každý plánovač úspěšně naplánoval řešení pro nějaký problém, jenž úspěšně vyřešil jiný. Lze ale říci, že *YAHSP3* generuje nejdelší plány, jenž jsou pak poměrně úspěšně vylepšeny. Plány z plánovače *Freelunch Madagascar* byly v průměru dostupných dat nejkratší a také v průměru dobře optimalizovány, nicméně podotýkáme, že také nalezl nejméně řešení. Délky ostatních plánovačů i jejich následné zlepšení si byly velice podobné.

8.4.5 Shrnutí

Ověřili jsme, že metody skupin AE, GAE a GCAE nelze obecně uspořádat podle efektivity, a k výjimkám dochází i v praxi.

Ukázali jsme, že metoda GAE na doménách soutěží IPC dosahuje v průměru nejlepšího výsledku, a to za cenu vyššího výpočetního času. Námi navrhnuté optimalizace metod AE a GAE zapříčinily znatelné zrychlení obou metod. Významnější optimalizace s pomocí lokálních landmarků byla efektivní v naprosté většině případů, zatímco metoda hledající inverzní cykly byla užitečná převážně pro dlouhé plány obsahující větší počet množin redundantních akcí.

Také jsme přišli s novou metodou GCAE, jenž snižuje výpočetní čas metody GAE. V obecnosti může dosahovat i lepších výsledků, nicméně v průměru na doménách soutěží IPC dosáhla menší efektivity. Je proto jistým kompromisem mezi AE a GAE a spolu s našimi podněty i inspirací k dalšímu výzkumu.

Závěr

Pro realizaci experimentů jsme vyvinuli prostředí pro reprezentaci úloh a plánů v rozšířeném SAS^+ formalismu. S pomocí tohoto prostředí jsme úspěšně implementovali námi upravené varianty metod *Action Elimination* (AE) a *Greedy Action Elimination* (GAE).

Přeložili jsme metodu *Inverse Action Elimination* (IAE) ze *STRIPS* do SAS^+ formalismu. Následně jsme reformulaci dále upravili a rozšířili tak, aby podporovala podmíněné efekty rozšířeného SAS^+ formalismu, a integrovali jsme ji do naší implementace.

Navrhli jsme a prodiskutovali několik možných přístupů, jimiž lze dosáhnout snížení výpočetního času existujících metod AE a GAE. Ty jsou založeny na dvou principech.

První diskutovaný princip byl založen na identifikaci lokálních landmarků. Primární inspirací pro tento přístup byla úspěšnost metod, které hledají (globální) landmarky, jenž jsou platné pro libovolný validní plán. V našem případě jsme omezeni pouze na jeden, a proto dokážeme tyto (lokální) landmarky hledat velice efektivně.

Druhým principem bylo hledání inverzních cyklů. Tato myšlenka vzešla z poznatků nabytých při reformulaci a rozšíření metody IAE. Jestliže sledujeme efekty akcí, můžeme popsat a identifikovat situace, v nichž již ověřování, tak jak ho provádí algoritmy AE a GAE, nepřinese žádné nové informace, a tak můžeme prohledávání ukončit dříve.

Efektivitu a časovou náročnost veškerých metod a přístupů jsme experimentálně změřili na šestnácti doménách z *International Planning Competition* (IPC), včetně všech dvanácti domén použitých v roce 2018, jenž některé z nich obsahují podmíněné efekty.

Metoda *Inverse Action Elimination* nedosáhla významných výsledků ani v nerozšířeném, ani v rozšířeném SAS^+ formalismu. Ale právě její reformulace a rozšíření sloužily jako základ k definování a popsání dalších principů optimalizace.

U principu optimalizace s pomocí lokálních landmarků jsme navrhli dva přístupy, lišící se schopností jejich identifikace. Námi provedené experimenty ukázaly, že oba tyto přístupy zrychlují původní metody AE a GAE, a to velice konzistentně. Také vzešlo na povrch, že námi navrhovaný složitější přístup dosáhl lepších výsledků, přestože zlepšení nebylo zcela významné.

U druhého principu, jenž je založen na inverzních cyklech, jsme zmínili dva různé přístupy. První (přesnější) z nich kvůli své složité režii dosáhl výsledku, jenž byl znatelně horší než původní metoda samotná. Následné slevení z nároků dalo vzniknout druhému přístupu, jenž sice na krátkých a "kvalitních" plánech strávil více času než původní metody, ale na dlouhých zajistil, že celkový čas metody byl výrazně snížen, a tak i tato optimalizace si našla své místo.

Také jsme obě optimalizace spojili. Po integraci druhého přístupu do metod AE a GAE s optimalizací pomocí lokálních landmarků došlo ještě k dalšímu snížení výpočetního času.

Na závěr, inspirováni prací na druhém principu a nedostatky metody GAE, jsme představili námi navržený algoritmus *Greedy Chaining Action Elimination* (GCAE), jenž je navržen tak, aby částečně řešil (často) velký výpočetní čas metody GAE a zároveň si zachoval efektivitu odebírání. Hlavní myšlenkou je zřetězování množin redundantních akcí s cykly. Tím dokážeme (částečně) zajistit to, že je-li to možné a výhodné, tak můžeme množiny odebrat v jedné iteraci vnějšího cyklu algoritmu GAE. Bohužel se v provedených

experimentech ukázalo, že GCAE není na zkoumaných problémech tak efektivní jako GAE. Díky němu jsme ale dosáhli významného snížení výpočetního času, a to převážně pro dlouhé plány. Algoritmus *Greedy Chaining Action Elimination* je tedy jistým kompromisem mezi AE a GAE.

I přesto, že metody AE, GAE a GCAE jsou obecně neporovnatelné co se týče schopnosti identifikace a odebrání množin inverzních akcí (což jsme experimentálně potvrdili), na námi zkoumaných úlohách se projevil trend, jenž poukazuje na skutečnost, že v praktických situacích je metoda GAE nejefektivnější, za což ale platíme (často znatelně) vyšším výpočetním časem. Metoda AE, jakožto asymptoticky nejrychlejší ze zkoumaných metod, splnila naše očekávání. Na úlohách sice projevila nižší efektivitu, ale za výrazně nižší výpočetní čas. Jak jsme již zmínili, metoda GCAE se umístila mezi AE a GAE, a to jak výkonem, tak spotřebovaným výpočetním časem.

V průběhu práce jsme nastínili několik dalších možných směrů výzkumu.

Už při analýze metod jsme přišli s další variantou metody *Greedy Chaining Action Elimination*, kde je jako účelová funkce místo úhrnné ceny množiny použito maximum ceny všech akcí v množině. Ukázali jsme, že už tento algoritmus dosahuje na námi zkoumaných problémech lepší úhrnné ceny odebraných akcí. Ukázali jsme, že metody AE, GAE a GCAE jsou neporovnatelné převážně z důvodu závislostí mezi množinami (jenž GCAE bere částečně v úvahu). V další práci bychom se proto rádi zaměřili na podrobnější prozkoumání této problematiky a na to, jak nám v tom mohou inverzní cykly (navíc) pomoci.

Dalším možným doposud nezmíněným směrem je nalezení sofistikovanějšího hledání lokálních landmarků, inspirované například metodami, jenž hledají globální landmarky pomocí *Domain Transition Graph*, neboť si myslíme, že tento způsob a analýza tohoto grafu by mohla odhalit i nové informace o závislostech množin inverzních akcí.

Bibliografie

1. GHALLAB, Malik; NAU, Dana S.; TRAVERSO, Paolo. *Automated planning - theory and practice*. Elsevier, 2004. ISBN 978-1-55860-856-6.
2. DIJKSTRA, Edsger W. A note on two problems in connexion with graphs. *Numerische Mathematik*. 1959, roč. 1, s. 269–271. Dostupné z DOI: 10.1007/BF01386390.
3. HART, Peter E.; NILSSON, Nils J.; RAPHAEL, Bertram. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Sci. Cybern.* 1968, roč. 4, č. 2, s. 100–107. Dostupné z DOI: 10.1109/TSSC.1968.300136.
4. BYLANDER, Tom. The Computational Complexity of Propositional STRIPS Planning. *Artif. Intell.* 1994, roč. 69, č. 1-2, s. 165–204. Dostupné z DOI: 10.1016/0004-3702(94)90081-7.
5. HELMERT, Malte. Complexity results for standard benchmark domains in planning. *Artif. Intell.* 2003, roč. 143, č. 2, s. 219–262. Dostupné z DOI: 10.1016/S0004-3702(02)00364-8.
6. KOENIG, Sven; SIMMONS, Reid. *The International Conference on Artificial Intelligence Planning Systems* [online]. 1998 [cit. 2021-04-30]. Dostupné z: http://www.cs.cmu.edu/~aips98/.
7. BACCHUS, Fahiem; KAUTZ, Henry; SMITH, David E.; LONG, Derek; GEFFNER, Hector; KOEHLER, Jana. *The Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems* [online]. 2000 [cit. 2021-04-30]. Dostupné z: https://ipc00.icaps-conference.org/.
8. DECHTER, Rina; PEARL, Judea. Generalized Best-First Search Strategies and the Optimality of A*. *J. ACM*. 1985, roč. 32, č. 3, s. 505–536. Dostupné z DOI: 10.1145/3828.3830.
9. POHL, Ira. Heuristic Search Viewed as Path Finding in a Graph. *Artif. Intell.* 1970, roč. 1, č. 3, s. 193–204. Dostupné z DOI: 10.1016/0004-3702(70)90007-X.
10. EBENDT, Rüdiger; DRECHSLER, Rolf. Weighted A* search - unifying view and application. *Artif. Intell.* 2009, roč. 173, č. 14, s. 1310–1342. Dostupné z DOI: 10.1016/j.artint.2009.06.004.
11. RICHTER, Silvia; WESTPHAL, Matthias. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.* 2010, roč. 39, s. 127–177. Dostupné z DOI: 10.1613/jair.2972.
12. FRANCÈS, Guillem; GEFFNER, Hector; LIPOVETZKY, Nir; RAMÍREZ, Miquel. Best-First Width Search in the IPC 2018: Complete, Simulated, and Polynomial Variants. In: 2018, s. 23–27.

13. NAKHOST, Hootan; MÜLLER, Martin. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement. In: BRAFMAN, Ronen I.; GEFFNER, Hector; HOFFMANN, Jörg; KAUTZ, Henry A. (ed.). *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*. AAAI, 2010, s. 121–128. Dostupné také z: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS10/paper/view/1420.

14. BALYO, Tomáš; CHRPA, Lukáš; KILANI, Asma. On Different Strategies for Eliminating Redundant Actions from Plans. In: EDELKAMP, Stefan; BARTÁK, Roman (ed.). *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014*. AAAI Press, 2014. Dostupné také z: http://www.aaai.org/ocs/index.php/SOCS/SOCS14/paper/view/8915.

15. CHRPA, Lukáš; MCCLUSKEY, Thomas Leo; OSBORNE, Hugh. Determining Redundant Actions in Sequential Plans. In: *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*. IEEE Computer Society, 2012, s. 484–491. Dostupné z DOI: 10.1109/ICTAI.2012.72.

16. POMMERENING, Florian; TORRALBA, Álvaro; BALYO, Tomáš. *International Planning Competition 2018 Classical Tracks* [online]. 2018 [cit. 2021-01-13]. Dostupné z: https://ipc2018-classical.bitbucket.io.

17. HOFFMANN, Jörg; PORTEOUS, Julie; SEBASTIA, Laura. Ordered Landmarks in Planning. *J. Artif. Intell. Res.* 2004, roč. 22, s. 215–278. Dostupné z DOI: 10.1613/jair.1492.

18. HELMERT, Malte; DO, Minh; REFANIDIS, Ioannis. *International Planning Competition 2008 Deterministic Part* [online]. 2010 [cit. 2021-01-13]. Dostupné z: https://ipc08.icaps-conference.org/deterministic/.

19. CHRPA, Lukáš; VALLATI, Mauro; MCCLUSKEY, Lee. *International Planning Competition 2014 Deterministic Part* [online]. 2014 [cit. 2021-01-13]. Dostupné z: https://helios.hud.ac.uk/scommv/IPC-14/index.html.

20. GHALLAB, Malik; NAU, Dana S.; TRAVERSO, Paolo. *Automated Planning and Acting*. Cambridge University Press, 2016. ISBN 978-1-107-03727-4. Dostupné také z: http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/automated-planning-and-acting?format=HB.

21. FIKES, Richard; NILSSON, Nils J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artif. Intell.* 1971, roč. 2, č. 3/4, s. 189–208. Dostupné z DOI: 10.1016/0004-3702(71)90010-5.

22. VIDAL, Vincent. YAHSP3 and YAHSP3-MT in the 8th International Planning Competition. In: 2014, s. 64–65.

23. GEREVINI, Alfonso; SAETTI, Alessandro; SERINA, Ivan. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artif. Intell.* 2008, roč. 172, č. 8-9, s. 899–944. Dostupné z DOI: 10.1016/j.artint.2008.01.002.

24. BÄCKSTRÖM, Christer; NEBEL, Bernhard. Complexity Results for SAS+ Planning. *Comput. Intell.* 1995, roč. 11, s. 625–656. Dostupné z DOI: 10.1111/j.1467-8640.1995.tb00052.x.

25. HELMERT, Malte. The Fast Downward Planning System. *J. Artif. Intell. Res.* 2006, roč. 26, s. 191–246. Dostupné z DOI: 10.1613/jair.1705.

26. NAKHOST, Hootan; MÜLLER, Martin. Action Elimination and Plan Neighborhood Graph Search: Two Algorithms for Plan Improvement - Extended Version. *Technical Report TR 10-01, Dept. of Computing Science. University of Alberta*. 2010. Dostupné z DOI: 10.7939/R3FM9H.

27. FINK, Eugene; YANG, Qiang. Formalizing Plan Justications. In: GLASGOW, Janice I.; HADLEY, Robert F. (ed.). *Proceedings of the Ninth Conference of the Canadian Society for Computational Studies of Intelligence*. 1992, s. 9–14. Dostupné z DOI: /10.1184/R1/6605831.v1.

28. FURCY, David. ITSA*: Iterative tunneling search with A*. In: *Proceedings of the AAAI 2006 Workshop on Heuristic Search, Memory-Based Heuristics and Their Applications*. 2006, s. 21–26. Dostupné také z: https://aaai.org/Library/Workshops/2006/ws06-08-005.php.

29. RATNER, Daniel; POHL, Ira. Joint and LPA*: Combination of Approximation and Search. In: KEHLER, Tom (ed.). *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1: Science*. Morgan Kaufmann, 1986, s. 173–177. Dostupné také z: http://www.aaai.org/Library/AAAI/1986/aaai86-028.php.

30. BALYO, Tomáš; BARTÁK, Roman; SURYNEK, Pavel. On Improving Plan Quality via Local Enhancements. In: BORRAJO, Daniel; FELNER, Ariel; KORF, Richard E.; LIKHACHEV, Maxim; LÓPEZ, Carlos Linares; RUML, Wheeler; STURTEVANT, Nathan R. (ed.). *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*. AAAI Press, 2012. Dostupné také z: http://www.aaai.org/ocs/index.php/SOCS/SOCS12/paper/view/5388.

31. BALYO, Tomáš; BARTÁK, Roman; SURYNEK, Pavel. Shortening Plans by Local Re-planning. In: *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*. IEEE Computer Society, 2012, s. 1022–1028. Dostupné z DOI: 10.1109/ICTAI.2012.144.

32. SIDDIQUI, Fazlul Hasan; HASLUM, Patrik. Continuing Plan Quality Optimisation. *J. Artif. Intell. Res.* 2015, roč. 54, s. 369–435. Dostupné z DOI: 10.1613/jair.4980.

33. HELMERT, Malte; DOMSHLAK, Carmel. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In: GEREVINI, Alfonso; HOWE, Adele E.; CESTA, Amedeo; REFANIDIS, Ioannis (ed.). *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*. AAAI, 2009. Dostupné také z: http://aaai.org/ocs/index.php/ICAPS/ICAPS09/paper/view/735.

34. BALYO, Tomáš; GOCHT, Stephan. The Freelunch Planning System Entering IPC 2018. In: 2018, s. 5–8.

35. KATZ, Michael. Cerberus: Red-Black Heuristic for Planning Tasks with Conditional Effects Meets Novelty Heuristic and Enchanced Mutex Detection. In: 2018, s. 47–50.

36. KATZ, Michael; HOFFMANN, Jörg. Good Old Mercury Planner. In: 2018, s. 51–52.

37. POMMERENING, Florian; TORRALBA, Álvaro; BALYO, Tomáš. *International Planning Competition 2018 Classical Tracks, Domains* [online]. 2018 [cit. 2021-05-12]. Dostupné z: https://ipc2018-classical.bitbucket.io/domains.html.

38. HELMERT, Malte; DO, Minh; REFANIDIS, Ioannis. *International Planning Competition 2008 Deterministic Part, Elevators Domain* [online]. 2009 [cit. 2021-05-12]. Dostupné z: https://ipc08.icaps-conference.org/deterministic/Elevators.html.

39. CHRPA, Lukáš; VALLATI, Mauro; MCCLUSKEY, Lee. *International Planning Competition 2014 Deterministic Part, Sequential Domains* [online]. 2014 [cit. 2021-01-13]. Dostupné z: https://helios.hud.ac.uk/scommv/IPC-14/domains_sequential.html.

Přílohy

A Tabulky

A.1 Legendy

Legenda pro tabulku 1

Optimalizovaný (výstupní) plán je označen Π' a původní (vstupní) plán je označen Π. Čas t_o je čas potřebný k běhu optimalizační metody na daném plánu a čas t_p je čas, jenž spotřeboval daný plánovač.

Ve sloupcích: celková délka optimalizovaných plánů $\sum L(\Pi')$; celkové absolutní zlepšení délky plánů $\sum \Delta L(\Pi')$; celková cena optimalizovaných plánů $\sum C\,(\Pi')$; celkové absolutní zlepšení cen plánů $\sum \Delta C(\Pi')$; průměr relativního zlepšení plánu $\mathrm{E}\dfrac{C\,(\Pi')}{C(\Pi)}$; směrodatná odchylka relativního zlepšení plánu $\sigma\dfrac{C\,(\Pi')}{C(\Pi)}$; maximum relativního zlepšení plánu $\max\dfrac{C\,(\Pi')}{C(\Pi)}$; celkový absolutní čas optimalizace v milisekundách $\sum t_o$; průměr relativního času optimalizace (vůči celkovému času) $\mathrm{E}\dfrac{t_o}{t_p + t_o}$; směrodatná odchylka relativního času optimalizace (vůči celkovému času) $\sigma\dfrac{t_o}{t_p + t_o}$; maximum relativního času optimalizace (vůči celkovému času) $\max\dfrac{t_o}{t_p + t_o}$.

V řádcích: zkratky názvů optimalizačních metod.

Legenda pro tabulku 2

Označením t_{m_i} myslíme čas, který spotřebovala metoda m_i optimalizací daného plánu. Skalárem n myslíme počet provedených optimalizací danou optimalizační metodou $n = n_{m_1} = n_{m_2}$.

Ve sloupcích: první metoda m_1; druhá metoda m_2; poměr součtů absolutní času metod $\dfrac{\sum t_{m_1}}{\sum t_{m_2}}$; počet situací, kdy byla m_1 rychlejší než m_2, $t_{m_1} < t_{m_2}$; počet situací, kdy byla m_1 rychlejší než m_2, vzhledem k celkovému počtu provedených experimentů n $\dfrac{t_{m_1} < t_{m_2}}{n}$; geometrický průměr relativního času t_{m_1} vzhledem k t_{m_2} $\mathrm{GM}\dfrac{t_{m_1}}{t_{m_2}}$; směrodatná odchylka relativního času t_{m_1} vzhledem k t_{m_2} $\sigma\dfrac{t_{m_1}}{t_{m_2}}$; maximum relativního času t_{m_1} vzhledem k t_{m_2} $\max\dfrac{t_{m_1}}{t_{m_2}}$; minimum relativního času t_{m_1} vzhledem k t_{m_2} $\min\dfrac{t_{m_1}}{t_{m_2}}$.

Legenda pro tabulky 3, 4 a 5

Optimalizovaný (výstupní) plán je označen Π' a původní (vstupní) plán je označen Π. Čas t_o je čas potřebný k běhu optimalizační metody na daném plánu a čas t_p je čas, jenž spotřeboval daný plánovač. Tabulka je pro každý plánovač obohacena o řádek "total", jenž obsahuje agregovaná data skrze všechny (v této tabulce zobrazené) optimalizační metody.

Ve sloupcích: počet plánů n; celková délka optimalizovaných plánů $\sum L(\Pi')$; celkové absolutní zlepšení délky plánů $\sum \Delta L(\Pi')$; celková cena optimalizovaných plánů $\sum C(\Pi')$; celkové absolutní zlepšení cen plánů $\sum \Delta C(\Pi')$; průměr relativního zlepšení plánu $\mathrm{E}\dfrac{C(\Pi')}{C(\Pi)}$; maximum relativního zlepšení plánu $\max \dfrac{C(\Pi')}{C(\Pi)}$; celkový absolutní čas optimalizace v milisekundách $\sum t_o$; průměr relativního času optimalizace (vůči celkovému času) $\mathrm{E}\dfrac{t_o}{t_p + t_o}$; maximum relativního času optimalizace (vůči celkovému času) $\max \dfrac{t_o}{t_p + t_o}$.

V řádcích: první písmena názvů plánovačů (a následně jejich hodnoty).

B Přiložené soubory

K práci přikládáme zdrojové kódy našeho prostředí, LaTeX zdrojové kódy této práce, *csv* soubor s výsledky experimentů vygenerovanými prostředím a soubor aplikace *Microsoft Excel*, v němž jsou zpracovány tabulky výsledků a je možné si je i libovolně filtrovat a upravovat.

	$\sum L(\Pi')$	$\sum \Delta L(\Pi')$	$\sum C(\Pi')$	$\sum \Delta C(\Pi')$	$\mathrm{E}\frac{C(\Pi')}{C(\Pi)}$	$\sigma\frac{C(\Pi')}{C(\Pi)}$	$\max\frac{C(\Pi')}{C(\Pi)}$	$\sum t_o$ (ms)	$\mathrm{E}\frac{t_o}{t_p+t_o}$	$\sigma\frac{t_o}{t_p+t_o}$	$\max\frac{t_o}{t_p+t_o}$
AE	79 705	15 964	281 246	29 957	6,46 %	12,83 %	66,89 %	47 654	1,00 %	2,57 %	20,76 %
AECB	79 705	15 964	281 246	29 957	6,46 %	12,83 %	66,89 %	15 850	0,25 %	0,69 %	6,73 %
AECBe	79 705	15 964	281 246	29 957	6,46 %	12,83 %	66,89 %	14 934	0,25 %	0,69 %	6,72 %
AECD	79 705	15 964	281 246	29 957	6,46 %	12,83 %	66,89 %	43 892	0,83 %	1,92 %	17,01 %
AELS	79 705	15 964	281 246	29 957	6,46 %	12,83 %	66,89 %	20 544	0,45 %	1,53 %	15,24 %
AELSe	79 705	15 964	281 246	29 957	6,46 %	12,83 %	66,89 %	19 593	0,43 %	1,52 %	15,35 %
GCAE	79 325	16 344	280 576	30 627	6,56 %	12,97 %	67,37 %	65 215	1,09 %	4,05 %	47,50 %
GAE	77 897	17 772	278 742	32 461	6,71 %	13,44 %	74,09 %	1 382 631	6,94 %	18,97 %	96,84 %
GAECB	77 897	17 772	278 742	32 461	6,71 %	13,44 %	74,09 %	278 989	3,58 %	12,58 %	87,73 %
GAECBe	77 897	17 772	278 742	32 461	6,71 %	13,44 %	74,09 %	266 418	3,49 %	12,41 %	87,61 %
GAECD	77 897	17 772	278 742	32 461	6,71 %	13,44 %	74,09 %	865 621	6,56 %	17,80 %	95,19 %
GAELS	77 897	17 772	278 742	32 461	6,71 %	13,44 %	74,09 %	824 014	4,73 %	16,22 %	95,53 %
GAELSe	77 897	17 772	278 742	32 461	6,71 %	13,44 %	74,09 %	814 322	4,69 %	16,20 %	95,53 %
IAE	86 158	9 511	295 344	15 859	3,87 %	9,79 %	51,95 %	292 478	3,84 %	8,82 %	78,26 %

Tabulka 1: Tabulka shrnující úhrnnou efektivitu a časovou náročnost optimalizačních metod na použitých problémech.

m_1	m_2	$\frac{\sum t_{m_1}}{\sum t_{m_2}}$	$t_{m_1} < t_{m_2}$	$\frac{t_{m_1} < t_{m_2}}{n}$	GM$\frac{t_{m_1}}{t_{m_2}}$	$\sigma\frac{t_{m_1}}{t_{m_2}}$	max $\frac{t_{m_1}}{t_{m_2}}$	min $\frac{t_{m_1}}{t_{m_2}}$
AE	GAE	3,45 %	423	80,57 %	40,66 %	46,12 %	222,94 %	0,32 %
AELSe	AELS	95,37 %	444	84,57 %	92,98 %	13,92 %	199,39 %	41,73 %
GAELSe	GAELS	98,82 %	377	71,81 %	94,65 %	13,06 %	282,22 %	38,95 %
AECBe	AECB	94,22 %	355	67,62 %	95,54 %	19,33 %	338,99 %	34,05 %
GAECBe	GAECB	95,49 %	378	72,00 %	93,73 %	11,37 %	175,50 %	43,20 %
AELS	AE	43,11 %	484	92,19 %	35,42 %	40,02 %	579,17 %	2,84 %
AELSe	AE	41,12 %	502	95,62 %	32,94 %	34,49 %	358,33 %	2,76 %
AECD	AE	92,10 %	197	37,52 %	109,69 %	48,11 %	546,26 %	35,49 %
AECB	AE	33,26 %	482	91,81 %	31,11 %	39,88 %	362,50 %	2,76 %
AECBe	AE	31,34 %	490	93,33 %	29,72 %	41,04 %	420,83 %	2,78 %
GAELS	GAE	59,60 %	487	92,76 %	32,55 %	34,79 %	222,31 %	2,69 %
GAELSe	GAE	58,90 %	509	96,95 %	30,81 %	33,10 %	208,11 %	2,68 %
GAECD	GAE	62,61 %	145	27,62 %	112,42 %	40,98 %	403,11 %	42,35 %
GAECB	GAE	20,18 %	484	92,19 %	30,16 %	38,50 %	211,76 %	2,20 %
GAECBe	GAE	19,27 %	492	93,71 %	28,27 %	37,98 %	202,70 %	2,10 %
AECB	AELS	77,15 %	327	62,29 %	87,82 %	31,11 %	380,74 %	14,12 %
AECBe	AELSe	76,22 %	297	56,57 %	90,23 %	32,16 %	353,40 %	13,19 %
GAECB	GAELS	33,86 %	253	48,19 %	92,64 %	24,92 %	217,73 %	14,57 %
GAECBe	GAELSe	32,72 %	253	48,19 %	91,74 %	24,78 %	223,49 %	13,41 %
AECBe	GCAE	22,90 %	512	97,52 %	47,98 %	27,73 %	330,09 %	3,76 %
GCAE	GAECBe	24,48 %	208	39,62 %	89,08 %	51,55 %	311,59 %	3,19 %

Tabulka 2: Tabulka znázorňující časové porovnání dvojic optimalizačních metod.

	n	metoda	$\sum L(\Pi')$	$\sum \Delta L(\Pi')$	$\sum C(\Pi')$	$\sum \Delta C(\Pi')$	$\mathrm{E}\frac{C(\Pi')}{C(\Pi)}$	$\max \frac{C(\Pi')}{C(\Pi)}$	$\sum t_o$ (ms)	$\mathrm{E}\frac{t_o}{t_p+t_o}$	$\max \frac{t_o}{t_p+t_o}$
C.	139	AE	15 437	555	45 389	1 021	2,49 %	24,24 %	7 103	0,08 %	0,71 %
		AECBe	15 437	555	45 389	1 021	2,49 %	24,24 %	2 337	0,03 %	0,31 %
		AECD	15 437	555	45 389	1 021	2,49 %	24,24 %	7 034	0,08 %	0,82 %
		AELSE	15 437	555	45 389	1 021	2,49 %	24,24 %	2 505	0,03 %	0,30 %
		GCAE	15 401	591	45 353	1 057	2,52 %	24,24 %	4 966	0,08 %	3,12 %
		GAE	15 401	591	45 353	1 057	2,52 %	24,24 %	32 463	0,40 %	6,79 %
		GAECBe	15 401	591	45 353	1 057	2,52 %	24,24 %	7 779	0,13 %	4,74 %
		GAECD	15 401	591	45 353	1 057	2,52 %	24,24 %	32 553	0,42 %	7,43 %
		GAELSe	15 401	591	45 353	1 057	2,52 %	24,24 %	8 455	0,13 %	4,40 %
		IAE	15 758	234	45 823	587	1,53 %	14,39 %	37 795	0,47 %	4,79 %
	1390	total	154 511	5 409	454 144	9 956	2,41 %	24,24 %	142 991	0,19 %	7,43 %
F.	34	AE	1 551	394	8 278	3 448	16,82 %	64,06 %	524	0,04 %	0,18 %
		AECBe	1 551	394	8 278	3 448	16,82 %	64,06 %	298	0,01 %	0,05 %
		AECD	1 551	394	8 278	3 448	16,82 %	64,06 %	525	0,05 %	0,24 %
		AELSE	1 551	394	8 278	3 448	16,82 %	64,06 %	303	0,01 %	0,05 %
		GCAE	1 550	395	8 291	3 435	16,80 %	58,33 %	2 770	0,07 %	0,72 %
		GAE	1 549	396	8 290	3 436	16,80 %	58,33 %	8 306	0,31 %	2,35 %
		GAECBe	1 549	396	8 290	3 436	16,80 %	58,33 %	3 502	0,09 %	0,91 %
		GAECD	1 549	396	8 290	3 436	16,80 %	58,33 %	8 389	0,34 %	2,38 %
		GAELSe	1 549	396	8 290	3 436	16,80 %	58,33 %	3 504	0,09 %	0,90 %
		IAE	1 584	361	8 678	3 048	15,45 %	51,95 %	1 587	0,13 %	0,61 %
	340	total	15 534	3 916	83 241	34 019	16,67 %	64,06 %	29 709	0,12 %	2,38 %

Tabulka 3: Tabulka znázorňující efektivitu a časovou náročnost metod pro konkrétní plánovače.

	n	metoda	$\sum L(\Pi')$	$\sum \Delta L(\Pi')$	$\sum C(\Pi')$	$\sum \Delta C(\Pi')$	$\mathrm{E}\frac{C(\Pi')}{C(\Pi)}$	$\max \frac{C(\Pi')}{C(\Pi)}$	$\sum t_o$ (ms)	$\mathrm{E}\frac{t_o}{t_p+t_o}$	$\max \frac{t_o}{t_p+t_o}$
L.	167	AE	27 058	2 455	87 763	8 499	5,54 %	52,44 %	20 511	0,78 %	8,35 %
		AECBe	27 058	2 455	87 763	8 499	5,54 %	52,44 %	4 644	0,14 %	1,62 %
		AECD	27 058	2 455	87 763	8 499	5,54 %	52,44 %	19 911	0,73 %	7,73 %
		AELSE	27 058	2 455	87 763	8 499	5,54 %	52,44 %	5 712	0,20 %	2,22 %
		GCAE	26 846	2 667	87 447	8 815	5,71 %	52,44 %	26 636	0,61 %	19,29 %
		GAE	26 799	2 714	87 355	8 907	5,76 %	52,44 %	220 511	5,41 %	64,63 %
		GAECBe	26 799	2 714	87 355	8 907	5,76 %	52,44 %	50 940	1,40 %	27,32 %
		GAECD	26 799	2 714	87 355	8 907	5,76 %	52,44 %	211 697	5,29 %	62,66 %
		GAELSe	26 799	2 714	87 355	8 907	5,76 %	52,44 %	63 180	1,84 %	28,29 %
		IAE	28 574	939	90 800	5 462	3,75 %	39,28 %	120 956	2,92 %	22,69 %
	1670	total	270 848	24 282	878 719	83 901	5,47 %	52,44 %	744 700	1,93 %	64,63 %
B.[a]	168	AE	18 745	817	103 947	2 528	2,67 %	29,08 %	13 444	0,66 %	8,27 %
		AECBe	18 745	817	103 947	2 528	2,67 %	29,08 %	6 549	0,17 %	2,51 %
		AECD	18 745	817	103 947	2 528	2,67 %	29,08 %	13 433	0,67 %	8,49 %
		AELSE	18 745	817	103 947	2 528	2,67 %	29,08 %	6 831	0,18 %	2,63 %
		GCAE	18 739	823	103 941	2 534	2,68 %	29,08 %	19 813	0,43 %	9,32 %
		GAE	18 737	825	103 935	2 540	2,70 %	29,08 %	57 281	1,76 %	22,21 %
		GAECBe	18 737	825	103 935	2 540	2,70 %	29,08 %	29 417	0,56 %	11,19 %
		GAECD	18 737	825	103 935	2 540	2,70 %	29,08 %	56 942	1,77 %	21,92 %
		GAELSe	18 737	825	103 935	2 540	2,70 %	29,08 %	30 465	0,58 %	11,89 %
		IAE	19 262	300	105 340	1 135	1,24 %	22,30 %	91 108	2,99 %	24,48 %
	1680	total	187 929	7 691	1 040 809	23 941	2,54 %	29,08 %	325 283	0,98 %	24,48 %

Tabulka 4: Tabulka znázorňující efektivitu a časovou náročnost metod pro konkrétní plánovače.

[a] Z důvodu kolize prvních písmen je plánovač *LAPKT-BFWS-Preference* označován jako "B".

	n	metoda	$\sum L(\Pi')$	$\sum \Delta L(\Pi')$	$\sum C(\Pi')$	$\sum \Delta C(\Pi')$	$\mathrm{E}\frac{C(\Pi')}{C(\Pi)}$	$\max \frac{C(\Pi')}{C(\Pi)}$	$\sum t_o$ (ms)	$\mathrm{E}\frac{t_o}{t_p+t_o}$	$\max \frac{t_o}{t_p+t_o}$
Y.	46	AE	18 988	11 743	40 315	14 461	23,94 %	66,89 %	6 812	5,94 %	20,76 %
		AECBe	18 988	11 743	40 315	14 461	23,94 %	66,89 %	1 272	1,60 %	6,72 %
		AECD	18 988	11 743	40 315	14 461	23,94 %	66,89 %	3 724	4,14 %	17,01 %
		AELSE	18 988	11 743	40 315	14 461	23,94 %	66,89 %	4 408	3,49 %	15,35 %
		GCAE	18 863	11 868	39 990	14 786	24,37 %	67,37 %	11 305	8,35 %	47,50 %
		GAE	17 485	13 246	38 255	16 521	25,83 %	74,09 %	1 064 798	51,77 %	96,84 %
		GAECBe	17 485	13 246	38 255	16 521	25,83 %	74,09 %	174 948	32,29 %	87,61 %
		GAECD	17 485	13 246	38 255	16 521	25,83 %	74,09 %	556 784	47,72 %	95,19 %
		GAELSe	17 485	13 246	38 255	16 521	25,83 %	74,09 %	708 885	44,20 %	95,53 %
		IAE	23 054	7 677	49 149	5 627	10,04 %	41,91 %	55 279	21,31 %	78,26 %
	460	total	187 809	119 501	403 419	144 341	23,35 %	74,09 %	2 588 217	22,08 %	96,84 %

Tabulka 5: Tabulka znázorňující efektivitu a časovou náročnost metod pro konkrétní plánovače.