FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Líná kompilace v klasickém plánování
Student:	Bc. Zuzana Fílová
Vedoucí:	doc. RNDr. Pavel Surynek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Cílem práce ne navrhnout přístup pro línou kompilaci problému klasického plánování. Jako cílový formalismus kompilace předpokládáme problém splňování omezení (CSP), nebo výrokovou splnitelnost (SAT). Líným přístupem přitom rozumíme kompilaci, která nespecifikuje předem všechna omezení, ale omezení postupně přidává podle zjištěných diskrepancí v aktuálním řešení problému. Otevřenou otázkou je volba omezení, které je možné při prvotní kompilaci problému vynechat. Úkoly pro řešitele jsou následující:

1. Prostudujte formalismus a algoritmy klasického plánování. Zaměřte se zejména na techniky, které kompilují klasické plánování do jiného formalismu, jako je například CSP nebo SAT.
2. Na základě provedené rešerše navrhněte línou modifikaci existující kompilační techniky.
3. Svůj návrh implementujte formou softwarového prototypu a otestujte jej na relevantních testovacích úlohách například ze soutěže International Planning Competition (IPC).

[1] Malik Ghallab, Dana S. Nau, Paolo Traverso: Automated planning - theory and practice. Elsevier 2004, ISBN 978-1-55860-856-6, pp. I-XXVIII, 1-635

[2] Peter van Beek, Xinguang Chen: CPlan: A Constraint Programming Approach to Planning. AAAI/IAAI 1999: 585-590

[3] Henry A. Kautz, Bart Selman: Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. AAAI/IAAI, Vol. 2 1996: 1194-1201

[4] Nils Christian Froleyks, Tomás Balyo, Dominik Schreiber: PASAR - Planning as Satisfiability with Abstraction Refinement. SOCS 2019: 70-78

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 26. ledna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Líná kompilace v klasickém plánování

Bc. Zuzana Fílová

Katedra aplikované matematiky
Vedoucí práce: doc. RNDr. Pavel SURYNEK, Ph.D.

4. ledna 2022

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 4. ledna 2022 ……

Odkaz na tuto práci

Fílová, Zuzana. *Líná kompilace v klasickém plánování*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Předmětem této diplomové práce je problematika líné kompilace v klasickém plánování. V teoretické části práce jsou nejprve shrnuty základní informace o klasickém plánování, definovány důležité pojmy klasické reprezentace plánovacích problémů a představeny základní algoritmy pro jejich řešení, zejména prohledávání plánovacího stavového prostoru a techniky využívající plánovací graf. Poslední sekce se věnuje převodu plánování na problém výrokové splnitelnosti (SAT).

Na základě zjištění z teoretické části byla navržena metoda pro línou kompilaci plánovacích problémů do SAT, při které na rozdíl od klasické kompilace dochází k postupnému vytváření a úpravám formule výrokové logiky. V rámci praktické části práce byl implementován plánovač využívající dvě varianty kompilace – navrženou metodu pro línou kompilaci a kompilaci klasickou.

Plánovač byl testován na úlohách ze soutěže IPC (International Planning Competition). Experimenty se zaměřovaly na vyhodnocení úspěšnosti plánovače s línou kompilací a porovnání výsledků s plánovačem využívajícím klasický způsob kompilace. Celkem bylo využito 79 problémů různé obtížnosti ze čtyř domén, 63 z nich dokázal plánovač s línou kompilací vyřešit rychleji než plánovač s klasickou kompilací. Provedené experimenty poukázaly na výhody a možné nevýhody líné kompilace. Výsledky experimentů naznačují, že využití líné kompilace má potenciál ke zlepšení výkonu plánovače.

Klíčová slova automatické plánování, klasické plánování, plánování jako SAT, líná kompilace, líné kódování, SAT, výroková splnitelnost

Abstract

The subject of this diploma thesis is focused on a lazy compilation in classical planning. The theoretical part summarizes the basics of classical planning. Key concepts of the classical representation of planning problems are defined and basic planning algorithms are presented, in particular, the search in the planning state space and techniques using the planning graph. The compilation of the planning problem into the propositional satisfiability problem (SAT) is discussed at the end of this section.

Based on the obtained knowledge, a new method for lazy compilation of planning problems into SAT has been proposed. Different from the classical compilation, in this method the propositional formula is gradually created and modified. As part of the practical part of the work, a planner was implemented using two compilation variants - the proposed method for lazy compilation and classical compilation.

The planner was tested on planning problems from the International Planning Competition (IPC). The experiments focused on evaluating the success of the planner based on lazy compilation and comparing the results with the planner using the classical compilation method. A total of 79 problems of varying difficulty from four domains were used, of which the lazy planner was able to solve 63 faster than the classical planner. The performed experiments pointed out the advantages and possible disadvantages of lazy compilation. The results of the experiments indicate that the use of lazy compilation has the potential to improve the performance of the planner.

Keywords automated planning, classical planning, lazy compilation, SAT, planning as satisfiability, lazy encoding

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Automatické plánování je jedna z oblastí umělé inteligence, kterou lze uplatnit v mnoha odvětvích jako třeba plánování kosmických misí, doprava, výroba nebo robotika. Klasické plánování je podoblast automatického plánování, která pracuje se zjednodušeným modelem světa (akce trvají nulový čas, determinismus, …). I přes toto zjednodušení je klasické plánování velmi složitým problémem. Mnoho výzkumů v této oblasti bylo motivováno právě jeho složitostí.

Již na jednoduchých příkladech si lze všimnout, že problémy, které se pro člověka zdají přímočaré a snadné, mohou představovat pro počítač problém naopak velmi složitý, protože zde dochází ke kombinatorické explozi. Pokud budeme například řešit problém stavění kostek do věží (Blocks World) s pouhými několika kostkami (4-6) pomocí základních algoritmů prohledávání stavového prostoru, nalezení řešení může modernímu počítači trvat několik minut, hodin nebo i déle. Aby mohlo být automatické plánování použito v praxi (plánovací problémy jsou poté nesrovnatelně složitější), je potřeba vynalézat a využívat efektivnější algoritmy. Jedním z nich je převod plánovacích problémů do výrokové splnitelnosti (SAT), který je předmětem této práce.

Porovnáváním úspěšnosti plánovacích algoritmů se zabývá soutěž IPC, které se účastní se svými plánovači vědci z celého světa. Plánovače založené na převodu plánování na SAT již nejsou v současné době mezi těmi, které dosahují v soutěži IPC nejlepších výsledků. Po období jejich největšího úspěchu se objevily plánovače fungující na jiných principech, které jejich výkon překonaly. Ačkoliv se proto tento přístup k řešení plánovacích problémů může zdát jako překonaný, stále zde existuje určitý potenciál do budoucna, díky němuž má smysl se tomuto tématu věnovat.

Velkou výhodou převodu plánování na SAT je to, že každý pokrok v řešení SAT znamená automaticky také zlepšení výkonu plánovače. Může se tedy stát (stejně jako se stalo již v minulosti), že nastane vylepšení stávajících nebo objev nových algoritmů pro řešení SAT, což posune tyto plánovače zpět do

světové špičky. Musíme také zmínit, že některé nejmodernější portfoliové plánovače stále obsahují ve svém portfoliu i plánovače založené na převodu do SAT.

Líný způsob řešení problémů je velmi zajímavý a líná kompilace byla úspěšně použita například při řešení problému multiagentního hledání cest (Multi-Agent Path Finding – MAPF). [1] Nabízí se zde proto prostor pro další výzkum a otázka, zda půjde líný přístup úspěšně aplikovat i v oblasti klasického plánování.

Struktura práce

Teoretická část práce poskytuje základní ucelený vhled do problematiky klasického plánování. Jsou zde uvedeny definice základních pojmů, které jsou doplněny ukázkami a příklady. Dále jsou zde představeny některé algoritmy pro řešení plánovacích problémů, které jsou zpravidla uvedeny včetně pseudokódu. Poslední sekce teoretické části obsahuje informace o převodu plánování na problém splnitelnosti včetně popisu různých způsobů kódování. Praktická část se skládá z popisu návrhu metody pro línou kompilaci plánovacích problémů do SAT a experimentálního vyhodnocení. Výsledkem této části je teoretický návrh a následně implementovaný plánovač pracující s různými variantami kompilace – línou a klasickou. Poslední kapitola se věnuje popisu a výsledkům provedených experimentů.

Kapitola 1

Cíl práce

Cílem rešeršní části práce je shrnutí problematiky klasického plánování a vysvětlení základních pojmů z této oblasti. Cílem praktické části práce je navrhnout přístup pro línou kompilaci problému klasického plánování do výrokové splnitelnosti (SAT). Líným přístupem se rozumí taková kompilace, která nespecifikuje předem všechna omezení, ale postupně je přidává podle zjištěných chyb v aktuálním řešení problému. Cílem práce je dále implementovat plánovač využívající navrženou metodu pro línou kompilaci a experimentálně vyhodnotit jeho úspěšnost.

Část I

Teoretická část

Kapitola 2

Plánování

Plánování je abstraktní proces uvažování, který vybírá a řadí dostupné akce, přičemž bere v úvahu jejich očekávané výsledky. Cílem plánování je najít posloupnost kroků/akcí (plán), které vedou k dosažení daného cíle. V obecném kontextu se plánováním rozumí teoretická příprava posloupnosti těchto kroků, aniž bychom kterýkoliv krok během plánování provedli. V případě plánování automatického je cílem vytvořit plánovací systém (plánovač), který zvládne samostatně bez dalšího zásahu člověka vyhotovit plán, jehož případné provedení zadaný problém vyřeší. Automatické plánování je jedna ze základních oblastí umělé inteligence. [2] [3]

Plánování lze využívat v mnoha rozdílných oblastech. Jako příklad můžeme uvést robotiku, logistiku, plánování cest a pohybu, plánování komunikace, hry nebo montáž výrobků. Stejně tak jako existuje mnoho různých oblastí použití, tak existuje také mnoho různých akcí, které se ve vytvořených plánech uplatňují. V oblasti logistiky se bude jednat například o akce pohyb vozidel, naložení/vyložení balíku atd., naproti tomu při plánování komunikace plánovač využívá akce poslání/příjem zprávy. Obecně lze říci, že oblasti, ve kterých se plánování může uplatnit, jsou velmi rozdílné a platí v nich různé zákonitosti a pravidla.

Přístupy k plánování

Vzhledem k výše uvedenému můžeme rozlišit dva základní přístupy k automatickému plánování:

- plánování závislé na doméně (specifické pro určitou doménu),
- plánování doménově nezávislé.

V prvním případě je plánovací systém vytvořen na míru konkrétní doméně a může proto využívat doménově specifické znalosti a integrovat je do způsobu

řešení problémů. Tento přístup může v určitých případech/doménách hledání plánu velmi urychlit a zefektivnit. [4] [5]

Mezi nevýhody doménově závislého plánování naopak patří nutnost řešit problém plánování v každé doméně zvlášť, což je spojeno s velkými náklady (v tomto případě se nelze spoléhat na obecné nástroje). Tento přístup také nepostihuje všeobecné společné rysy plánování a je nedostatečný pro vytvoření autonomního inteligentního stroje, jehož uvažovací schopnosti při plánování nebudou omezeny pouze na určité oblasti. Z těchto důvodů se výzkum automatického plánování zaměřuje především na přístup doménově nezávislý. [3]

Doménově nezávislé plánování se opírá o abstraktní obecné modely akcí. Cílem automatického plánování je v tomto případě objevovat přístupy, které jsou využitelné obecně pro jakoukoliv doménu. Při řešení konkrétního problému je vstupem plánovače vedle samotného popisu problému také specifikace domény, ve které se daný problém řeší. V této práci se dále budeme zabývat pouze plánováním nezávislým na doméně.

2.1 Klasické plánování

Klasické plánování je část automatického plánování, která pracuje s velmi zjednodušeným modelem reálného světa, který je omezen určitými předpoklady viz 2.1.1. Základní konceptuální model klasického plánování můžeme definovat jako stavový prostor s přechody Σ. V tomto případě je přechodový stavový prostor trojice $\Sigma = (S, A, \gamma)$, kde

- $S = \{s_0, s_1, ..., s_n\}$ je konečná množina stavů,
- $A = \{a_1, a_2, ..., a_k\}$ je konečná množina akcí,
- $\gamma : S \times A \to S$ je přechodová funkce, $\gamma(s, a)$ je definovaná pro všechny akce a aplikovatelné ve stavu s.

Přechodový stavový prostor klasického plánování lze reprezentovat jako orientovaný graf, jehož uzly odpovídají stavům z množiny S a hrany mezi nimi představují provedené akce. Plánovací problém lze poté přímočaře popsat jako hledání cesty v grafu od počátečního do koncového stavu. Problém hledání cest v grafu je velmi dobře prozkoumaný a vyřešený, z tohoto pohledu by se řešení problémů v klasickém plánování mohlo zdát jednoduché. Kdybychom měli celý tento graf explicitně k dispozici, skutečně by tomu tak bylo. Komplikace zde ovšem nastává v tom, že i v jednoduché doméně s relativně malým počtem objektů je počet stavů tak velký, že tento způsob řešení nelze v praxi uplatnit.

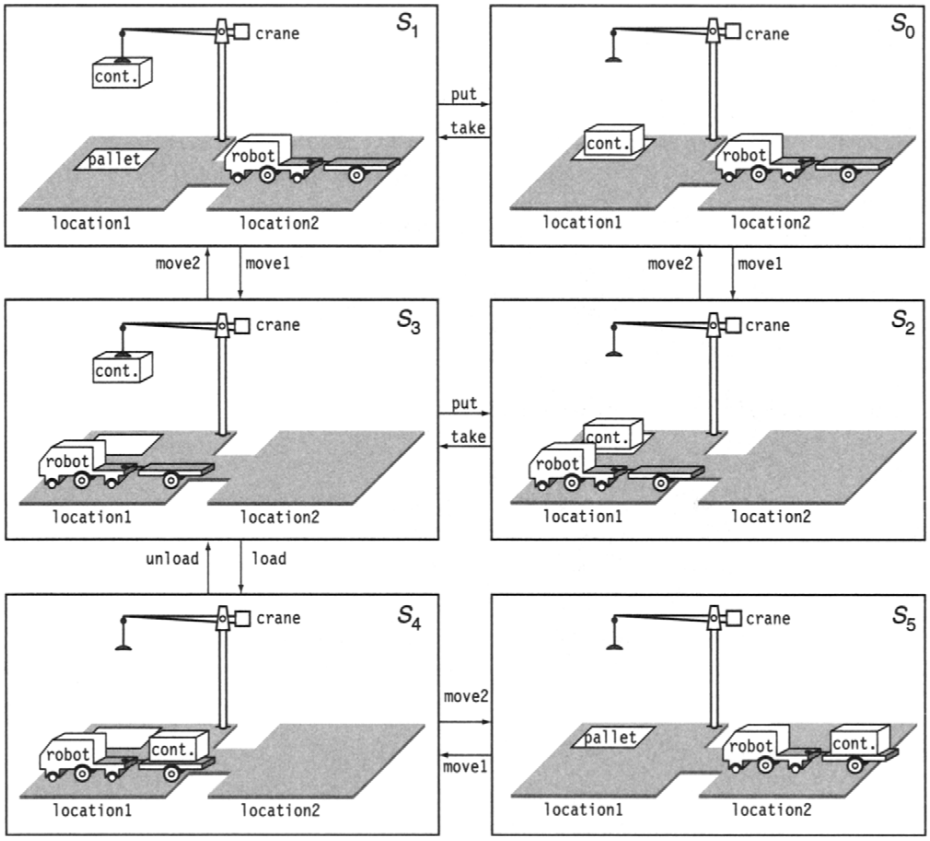
Na obrázku 2.1 je zobrazena ukázka stavového prostoru pro konkrétní doménu – přesun kontejnerů pomocí jeřábu a robota. Přechod mezi stavy S_0 až S_5 je zajišťován akcemi move, load, unload, put a take. Pokud by tato jednoduchá doména obsahovala pět lokací, tři vyhrazená místa pro kontejnery

na jednu lokaci, tři roboty a sto kontejnerů, potom by graf Σ měl přibližně 10^{277} stavů, což je asi 10^{190} krát více než odhadovaný počet všech částic ve vesmíru. [3] V důsledku toho je potřeba plánovací stavový prostor reprezentovat kompaktněji a popisovat určité podmnožiny S tak, aby v nich šlo snadno vyhledávat.

Hlavní výzvy a otázky k řešení v klasickém plánování jsou následující:

- Jakým způsobem je vhodné reprezentovat stavy a akce, aby nebylo potřeba explicitně vyjmenovat množiny S a A ? Bez vhodné reprezentace je nemožné vytvořit doménově nezávislý plánovací systém.

- Jak efektivně najít v této reprezentaci řešení plánovacího problému? Jaké použít algoritmy, heuristiky, techniky při hledání řešení?



Obrázek 2.1: Ukázka přechodu mezi stavy v jednoduché doméně [3]

2.1.1 Předpoklady

V případě klasického plánování vycházíme z následujících předpokladů:

- **Konečný Σ:** Stavový prostor Σ má konečnou množinu stavů.
- **Plně pozorovatelný Σ.**
- **Deterministický Σ:** pro každý stav s a každou akci a aplikovatelnou v s platí, že aplikace akce a ve stavu s dovede deterministický systém vždy pouze do jediného jiného stavu s_1.
- **Statický Σ:** stavy v systému Σ lze měnit pouze provedením akce, neexistují zde nedefinované události, které by měnily stavy
- **Implicitní čas:** akce trvají nulový čas, představují okamžitou změnu stavu.
- **Omezené cíle:** plánovač pracuje pouze s cílem, který je specifikován jako cílový stav s_g nebo množina cílových stavů S_g. Řešením je jakákoliv sekvence stavových přechodů, která vede k některému z cílových stavů. Rozšířená specifikace cílů, jako například zakázané stavy nebo podmínky na posloupnost stavových přechodů, nejsou v tomto zjednodušení podporovány.
- **Sekvenční plány:** řešením plánovacího problému je konečná posloupnost akcí.
- **Offline plánování:** plánovač nebere v úvahu žádnou nečekanou změnu, která by v systému Σ mohla nastat během plánování, vychází pouze z počátečního stavu.

2.1.2 Klasická reprezentace

Existuje několik různých způsobů reprezentace objektů v klasickém plánování. Všechny jsou navzájem ekvivalentní ve své vyjadřovací síle – každou doménu lze popsat ve všech těchto reprezentacích. V této práci se budeme věnovat pouze popisu klasické reprezentace, protože tento způsob využíváme v praktické části. Vycházíme z terminologie a definic pojmů uvedených v knize [3].

Klasická reprezentace vychází z množinové reprezentace a zobecňuje ji využitím predikátové logiky. Využívá vyjádření pomocí jazyka L, který je konečnou množinou konstant a predikátových symbolů. (Neexistují zde funkční symboly vyšší četnosti.) Atom je predikátový symbol s proměnnými nebo dosazenými konstantami (potom jej nazýváme základní atom). Literál je atom nebo negace atomu. Stavy jsou reprezentovány jako konečné množiny základních atomů. Předpokládáme uzavřenost světa. Základní atomy, které jsou ve stavu přítomné, jsou v dané interpretaci pravdivé, ostatní jsou v dané interpretaci považovány za nepravdivé.

2.1.2.1 Plánovací doména

Mějme jazyk L, který je konečnou množinou konstant a predikátových symbolů. Klasická plánovací doména v L je systém $\Sigma = (S, A, \gamma)$, kde

- $S \subseteq 2^{\{\text{všechny základní atomy v L}\}}$,
- $A = \{\text{všechny instance operátorů z } O\}$, kde O je množina operátorů viz definice 2.1.2.3,
- $\gamma(s, a) = (s \setminus \mathit{effects}^-(a)) \cup \mathit{effects}^+(a)$, pokud akce $a \in A$ je aplikovatelná ve stavu $s \in S$, jinak je $\gamma(s, a)$ nedefinovaná.

2.1.2.2 Plánovací problém

Plánovací problém je definován jako trojice $P = (\Sigma, s_0, g)$, kde

- s_0 (počáteční stav) je jakýkoliv stav z množiny S,
- g (cíl) je jakákoliv konečná množina základních atomů,
- $S_g = \{s \in S | s \text{ splňuje } g\}$, stav s splňuje cíl g, pokud platí

$$g^+ \subseteq s \wedge g^- \cap s = \emptyset$$

Zadání plánovacího problému můžeme zapsat jako $P = (O, s_0, g)$. Řešením plánovacího problému P je plán $\pi = (a_1, a_2, ..., a_k)$, kde $(a_1, a_2, ..., a_k)$ je posloupnost akcí odpovídající posloupnosti přechodů mezi stavy $(s_0, s_1, ..., s_k)$, pro kterou platí, že $s_1 = \gamma(s_0, a_1), ..., s_k = \gamma(s_{k-1}, a_k)$ a s_k patří do množiny cílových stavů S_g.

2.1.2.3 Operátory a akce

Plánovací operátor je trojice $o = (\mathit{name}(o), \mathit{precond}(o), \mathit{effects}(o))$, kde

- $\mathit{name}(o)$ označuje jméno operátoru ve tvaru $n(x_1, ..., x_k)$, kde n je unikátní symbol operátoru (žádné dva operátory v L nemohou mít stejný symbol) a $(x_1, ..., x_k)$ jsou všechny proměnné vyskytující se v operátoru,
- $\mathit{precond}(o)$ označuje předpoklady operátoru, je to množina literálů, které musí být ve stavu pravdivé, aby operátor bylo možné aplikovat,
- $\mathit{effects}(o)$ označují efekty operátoru, je to množina literálů, které se stanou vykonáním operátoru pravdivé, rozlišujeme pozitivní a negativní efekty.

Akce je instance operátoru, ve které jsou za všechny proměnné $(x_1, ..., x_k)$ dosazeny konstanty. Pokud pro akci a a stav s platí, že

$$precond(a) \subseteq s$$

potom akce a je aplikovatelná ve stavu s a výsledek této aplikace je stav

$$\gamma(s, a) = (s \setminus effects^{-}(a)) \cup effects^{+}(a)$$

2.2 Příklad plánovacího problému

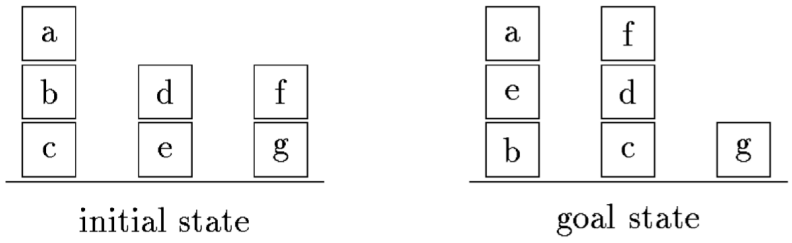
Pro lepší ilustraci a pochopení problému klasického plánování ukážeme výše uvedenou klasickou reprezentaci na konkrétním příkladě v doméně Blocks World.

2.2.1 Popis domény Blocks World

Blocks World (neboli svět kostek) je jednoduchá doména, která je již od počátku automatického plánování využívána jako modelová doména. Pomocí této domény jsou často ukazovány základní principy plánování, případně porovnávána úspěšnost jednotlivých plánovacích algoritmů. Tato doména také slouží často jako ukázková při výuce umělé inteligence. [6] Plánování v doméně Blocks World bylo široce zkoumáno zejména proto, že ačkoliv je tato doména velmi jednoduchá, lze na ní velice dobře ilustrovat obecné problémy plánování. Zvláště důležitá byla tato doména při zkoumání interakce cílů a podcílů - např. Sussmanova anomálie.

Základní verze domény Blocks World se skládá z konečného počtu stejně velkých kostek, stolu, který má dostatečnou velikost na to, aby se na něj všechny kostky vešly vedle sebe, a robotického ramene, které kostky přesouvá. Robotické rameno může najednou přesouvat pouze jednu kostku. Kostky mohou ležet buď přímo na stole nebo mohou být stavěny na sebe do různě vysokých věží. Robotické rameno může vždy vzít pouze jednu z vrchních kostek (takovou, na které v danou chvíli neleží žádná jiná kostka) a umístit ji buď na stůl nebo na vrchol již existující věže z kostek. Cílem je vytvořit plán popisující přesun kostek (jedné po druhé) od počátečního do cílového stavu. Počáteční a cílový stav jsou definovány konfigurací kostek na stole. [6]

Na obrázku 2.2 je ukázán konkrétní plánovací problém v popisované doméně. Na levé straně obrázku je zobrazena počáteční konfigurace kostek na stole, v pravé části poté požadovaná cílová konfigurace. Dole je uveden výsledný optimální plán pro zadaný problém. V případě domény Blocks World optimálním plánem myslíme takový plán, který má nejmenší možný počet kroků / přesunů kostek.



1. move a to table	2. move b to table	3. move d to c
4. move e to b	5. move a to e	6. move f to d

Obrázek 2.2: Ukázka plánovacího problému a optimálního plánu v doméně Blocks World. [4]

2.2.2 Klasická reprezentace

Pro popis domény Blocks World je použit jazyk L obsahující celkem pět predikátových symbolů:

- kostka x je položená na kostce y – (on ?x ?y),
- kostka je položená na stole – (ontable ?x),
- na kostce x není žádná jiná kostka – (clear ?x),
- robotické rameno nic nedrží – (handempty),
- robotické rameno drží kostku x – (holding ?x).

V doméně se vyskytují celkem čtyři operátory:

- zvednutí kostky ze stolu – (pick-up),
- položení kostky na stůl – (put-down),
- zvednutí kostky z vrcholu věže kostek – (unstack),
- položení kostky na jinou kostku – (stack).

V případě jednoduchého plánovacího problému P_1 viz obrázek 2.3 máme v jazyce L tři konstanty A, B, C. Počáteční stav s_0, kdy všechny kostky leží na stole, popíšeme jako množinu základních atomů:

$$s_0 = \{\text{ontable(A), ontable(B), ontable(C),}$$

clear(A), clear(B), clear(C),

handempty()}

a cíl g jako:

$$g = \{\text{ontable(A), ontable(C), on(B, A)}\}$$

Základní atomy, které jsou ve stavu přítomny, jsou v dané interpretaci pravdivé, ostatní nepravdivé – pro ilustraci například ve stavu s_0 je atom $ontable(A)$ pravdivý, zatímco atom $on(B, C)$ nepravdivý.

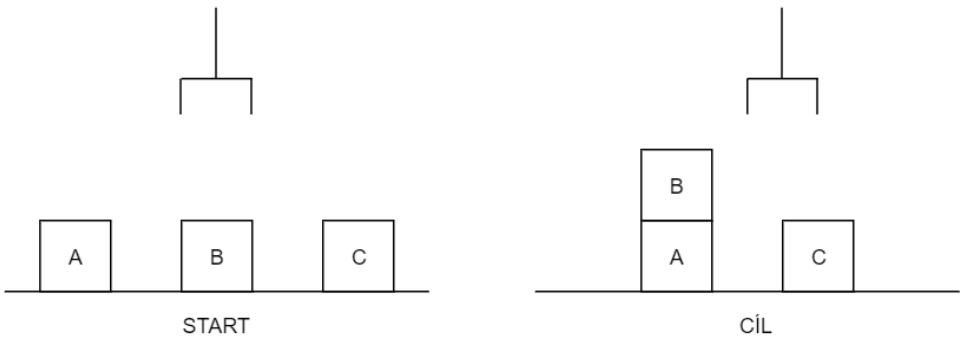
Řešením plánovacího problému je každá posloupnost akcí (odpovídající přechodům ve stavovém prostoru), které při jejich postupné aplikaci vedou ze stavu s_0 do stavu, ve kterém je splněn cíl g. Pro řešení problému může existovat více plánů. V tomto případě pro problém P_1 existuje například plán π_1:

$$\pi_1 = \text{(pick-up(B), stack(B,A))}$$

Tento plán π_1 označíme jako optimální, protože nelze najít plán, který by byl řešením problému P_1 a obsahoval méně akcí. Pro plánovací problém může existovat více různých optimálních plánů.

$$\pi_2 = \text{(pick-up(B), pick-up(C), put-down(C), stack(B,A))}$$

Plán π_2 je také řešením problému P_1, ale již nelze označit za optimální.



Obrázek 2.3: Jednoduchý problém v doméně Blocks World

2.2.3 Vstup plánovače

Již od roku 1998 je standardním jazykem pro popis plánovacích problémů a domén jazyk PDDL (The Planning Domain Definition Language). [7] Postupem času vznikala rozšíření jazyka PDDL, které poskytují vyjadřovací prostředky pro další konstrukty jako například pravděpodobnosti provedení akcí [8] nebo

domény s kontinuálním časem [9]. V našem případě si vystačíme se základní verzí jazyka PDDL.

Vstup plánovače v jazyce PDDL je rozdělen zpravidla do dvou souborů:

- soubor s popisem domény (predikáty a operátory),
- soubor s popisem problému (konstanty, počáteční a cílový stav).

Soubor se zápisem domény Blocks World v jazyce PDDL je vidět na obrázku 2.4. Výstupem plánovače je plán zapsaný jako posloupnost akcí, případně oznámení, že problém nemá řešení.

2.3 Složitost klasického plánování

Klasické plánování lze obecně označit za velmi složitý problém. Mnoho výzkumů v oblasti klasického plánování bylo motivováno právě jeho obtížností. Pro klasické plánování můžeme definovat dva druhy otázek jejichž složitost nás zajímá:

- Existence plánu – existuje řešení / plán?
- Délka plánu – existuje řešení / plán, které obsahuje $k \in \mathbb{N}$ akcí nebo méně?

2.3.1 Rozhodnutelnost

Pro klasické plánování je existence i délka plánu rozhodnutelný problém. V klasickém plánování je konečný počet možných stavů, proto lze vždy hrubou silou prohledat všechny stavy a zjistit, zda řešení existuje a jakou má délku. [3]

2.3.2 Složitost

Složitost problémů v klasickém plánování se liší v závislosti na podmínkách, kterými problém omezíme. V případě problému, kterým se zabýváme v této práci, tj. klasické plánování bez dalších doplňujících omezení, se jedná o exponenciální složitost. Problém existence plánu patří do třídy EXPSPACE-úplných problémů a problém délky plánu do třídy NEXPTIME-úplných problémů. [3]

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; 4 Op-blocks world
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x)
               )

  (:action pick-up
             :parameters (?x)
             :precondition (and (clear ?x) (ontable ?x)(handempty))
             :effect
             (and (not (ontable ?x))
                    (not (clear ?x))
                    (not (handempty))
                    (holding ?x)))

  (:action put-down
             :parameters (?x)
             :precondition (holding ?x)
             :effect
             (and (not (holding ?x))
                    (clear ?x)
                    (handempty)
                    (ontable ?x)))
  (:action stack
             :parameters (?x ?y)
             :precondition (and (holding ?x) (clear ?y))
             :effect
             (and (not (holding ?x))
                    (not (clear ?y))
                    (clear ?x)
                    (handempty)
                    (on ?x ?y)))
  (:action unstack
             :parameters (?x ?y)
             :precondition (and (on ?x ?y) (clear ?x) (handempty))
             :effect
             (and (holding ?x)
                    (clear ?y)
                    (not (clear ?x))
                    (not (handempty))
                    (not (on ?x ?y)))))
```

Obrázek 2.4: Zápis domény Blocks World v jazyce PDDL. [10]

Kapitola 3

Plánovací algoritmy

Algoritmy pro řešení plánovacích problémů můžeme obecně rozdělit do několika kategorií:

- **klasické plánovací techniky**
  - algoritmy prohledávání stavového prostoru
    - dopředné plánování
    - zpětné plánování
    - STRIPS
    - a další …
  - heuristické prohledávání stavového prostoru
  - plánování v prostoru plánů
  - a další …
- **neoklasické plánovací techniky**
  - techniky využívající plánovací graf
  - plánování jako splnitelnost (SAT)
  - plánování jako splňování omezení (CSP)
  - a další …

V této práci se budeme zabývat především způsoby řešení, které jsou založené na plánovacím grafu, a těmi, které plánování převádí na problém splnitelnosti.

3.1 Historie

Počátek řešení plánovacích problémů můžeme datovat do období okolo roku 1961.[11] Algoritmy prohledávání plánovacího stavového prostoru patří mezi nejstarší a mohou se zdát jako samozřejmý způsob řešení problému klasického plánování. Dlouhou dobu nebyly ovšem známy žádné efektivní způsoby a heuristiky pro prohledávání tak obrovského stavového prostoru. (Tyto techniky byly vyvinuty až později a ukázaly se jako velmi úspěšné - např. plánování pomocí heuristického prohledávání [12], Fast forward plánovač [13], Fast downward plánovač [14].)

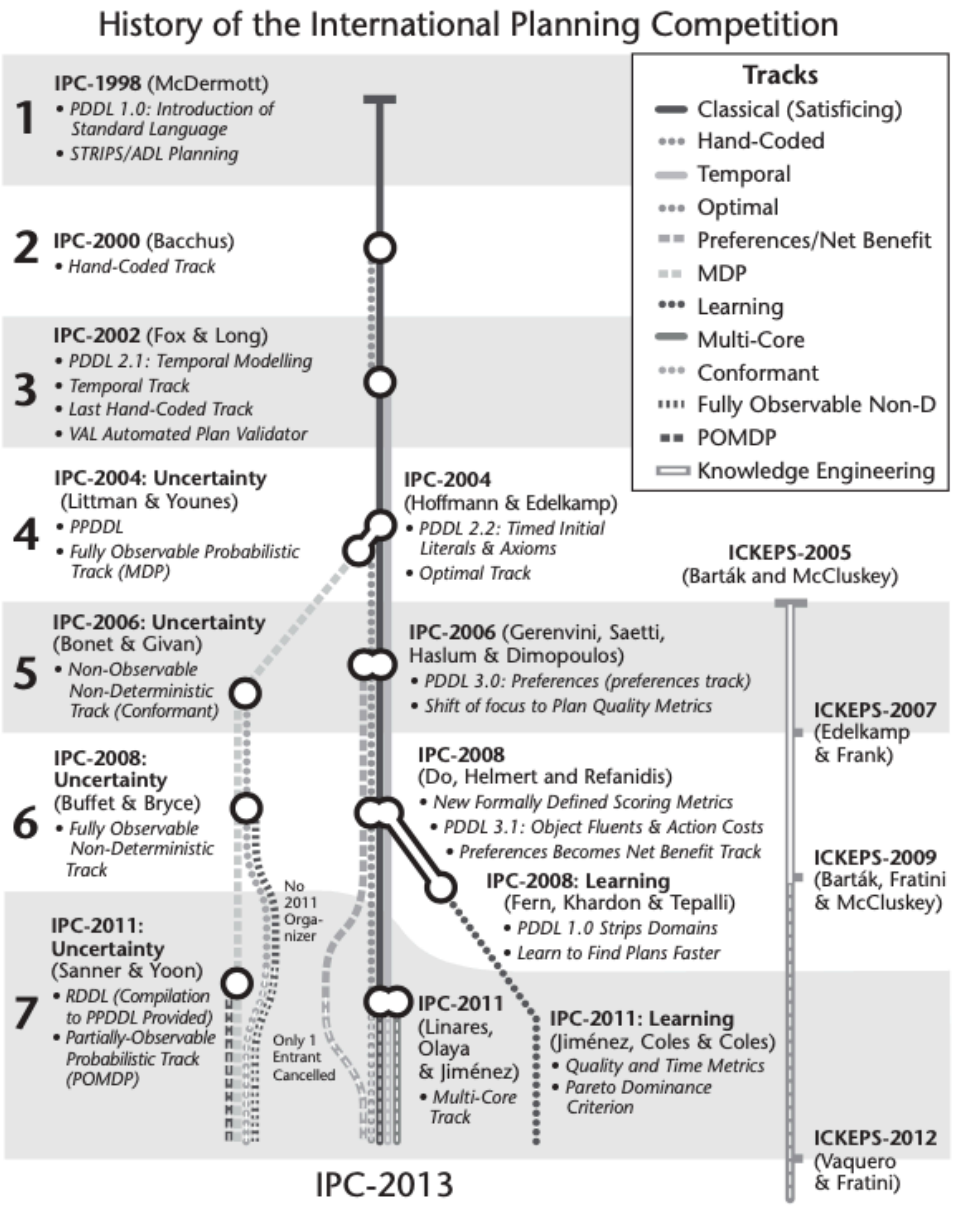
V průběhu historie se proto výzkum a rozvoj algoritmů v oblasti klasického plánování na nějakou dobu prakticky zastavil. Do té doby známé algoritmy (např. dopředné prohledávání 2, zpětné prohledávání 1, STRIPS [15], …) a přístupy dovolovaly vyřešit v akceptovatelném čase pouze velmi malé plánovací problémy. Velká složitost a kombinatorické potíže spojené s plánováním dostaly výzkum automatického plánování do slepé uličky. Další vývoj v této oblasti oživil až rozvoj neoklasických technik, které umožňují řešit v akceptovatelném čase výrazně větší problémy. Zásadní bylo objevení plánovacího grafu a navazujícího algoritmu Graphplan, který byl publikován v roce 1995. [16]

Oblast automatického plánování se poté začala více rozvíjet a v roce 1998 se konala první mezinárodní soutěž v automatickém plánování (1st International Planning Competition [10]). Finální části soutěže se zúčastnilo celkem pět plánovačů:

- tři plánovače založené na Graphplanu:
  - IPP plánovač [17]
  - SGP plánovač [18]
  - STAN plánovač [19]
- jeden plánovač založený na heuristickém prohledávání stavového prostoru (HSP plánovač [20]),
- jeden plánovač kompilující plánování do SAT (Blackbox plánovač [21]).

Plánovače řešily problémy v různých doménách a následně byly hodnoceny z hlediska času potřebného k vytvoření plánu a délky výsledného plánu. V průběhu soutěže se však nakonec ukázalo jako náročné určit, který z těchto systémů je obecně nejlepší. [22]

Velmi důležitým výsledkem prvního ročníku soutěže bylo ustanovení jazyka PDDL (The Planning Domain Definition Language [7]) jako standardu pro popis plánovacích domén a problémů. V dalších letech se postupně rozšiřovaly disciplíny a kategorie plánovací soutěže viz obrázek 3.1 a také se soutěží účastnilo více týmů výzkumníků z celého světa.



Obrázek 3.1: Historie mezinárodní plánovací soutěže IPC. [23]

V prvních letech soutěže IPC se v rámci klasického plánování rozlišovaly dvě kategorie:

- **Hand-coded track:** plánování s využitím doménově specifických znalostí,
- **Automated track:** plánování plně automatické – využití znalostí specifických pro doménu zde nebylo povoleno.

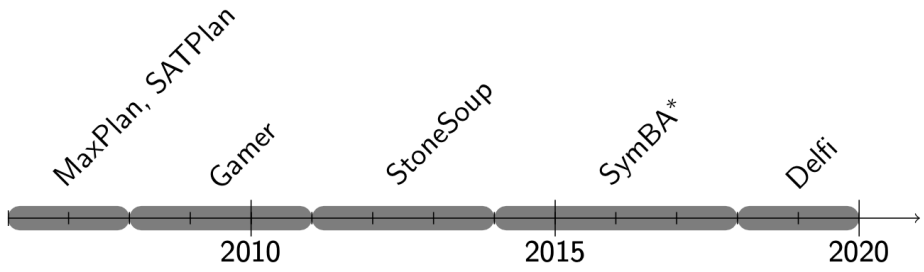
Později se klasické plánování rozdělilo do následujících kategorií:

- **Optimal track:** hledání optimálního plánu – cílem je nalezení nejlepšího možného plánu (optimálního z hlediska počtu kroků, případně s nejnižší cenou akcí apod.),
- **Satisficing track:** hledání uspokojivého plánu – v této kategorii je žádoucí nalézt také co nejlepší plán, ale nemusí být nutně optimální.

Na obrázcích 3.3 a 3.4 jsou zobrazeny trendy a průlomové objevy v průběhu let soutěže IPC v kategorii hledání optimálního plánu a v kategorii hledání uspokojivého plánu. Plánovací systémy založené čistě na Graphplanu byly rychle překonány plánovacími systémy kompilující problém plánování do SAT. Ty v prvních ročnících soutěže vítězily v obou kategoriích, později je vystřídaly plánovače založené na heuristickém prohledávání stavového prostoru (Satisficing track) a symbolickém prohledávání [24] (Optimal track). Podrobnější výsledky soutěží, které se konaly do roku 2008 jsou ukázány na obrázku 3.2.

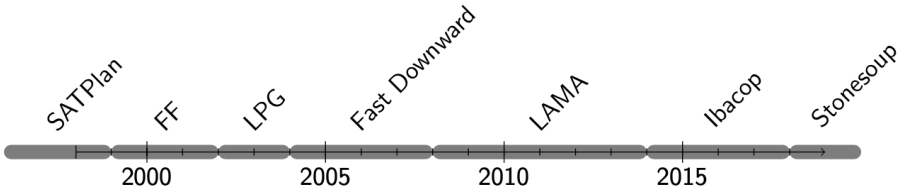
Year	Track	Winning Systems (approaches)
2008	Optimal	GAMER (model checking, bidirectional search)
2008	Satisficing	LAMA (fast downward search with FF heuristic)
2006	Optimal	SATPLAN, MAXPLAN (Boolean satisfiability)
2006	Satisficing	SGPLAN (forward search; partitions into independent subproblems)
2004	Optimal	SATPLAN (Boolean satisfiability)
2004	Satisficing	FAST DIAGONALLY DOWNWARD (forward search with causal graph)
2002	Automated	LPG (local search, planning graphs converted to CSPs)
2002	Hand-coded	TLPLAN (temporal action logic with control rules for forward search)
2000	Automated	FF (forward search)
2000	Hand-coded	TALPLANNER (temporal action logic with control rules for forward search)
1998	Automated	IPP (planning graphs); HSP (forward search)

Obrázek 3.2: Přehled nejlepších plánovacích systémů a přístupů v soutěži IPC do roku 2008. [11]



- SAT planners (MaxPlan, SATPlan)
- Symbolic Search planners (Gamer, SymBA*)
- Heuristic search planners
- Portfolios (StoneSoup, Delfi)

Obrázek 3.3: Trendy a průlomové objevy v soutěži IPC při hledání optimálních plánů [25]



- SAT-based planners (SAT-plan, Madagascar)
- Heuristic search planners (FF, LPG, Fast Downward, LAMA)
- Portfolios (Ibacop, Stonesoup)

Obrázek 3.4: Trendy a průlomové objevy soutěži IPC při hledání uspokojivých (ne nutně optimálních) plánů [25]

V prozatím posledním ročníku soutěže (IPC 2018) plánovače založené na kompilaci do SAT nebyly na aktuálních doménách označeny jako konkurenceschopné. K výsledkům soutěží IPC je však potřeba poznamenat, že v průběhu ročníků se ukázalo, že neexistuje jediný plánovač, který by byl vždy obecně nejlepším plánovačem pro každou doménu a každý problém v ní. Ačkoliv byl určitý plánovač v rámci soutěžní kategorie vyhodnocen jako vítězný podle nastavených metrik soutěže, vždy se našly nějaké problémy, ve kterých ostatní plánovače celkového vítěze předčily. [26]

V posledních letech se v soutěžích IPC velmi dobře umísťovaly tzv. portfoliové plánovací systémy (např. Ibacop [26], StoneSoup [27], Delfi [28]), které pro řešení každého konkrétního problému vybírají některý z nabídky již existujících plánovačů / plánovacích algoritmů. Některé z portfoliových plánovačů stále využívají ve svém portfoliu i plánovače založené na SAT. [29]

Role soutěže IPC a přidružené konference ICAPS (International Conference on Automated Planning and Scheduling) je v oboru automatického plánování velmi významná. Soutěže IPC se zúčastní nejmodernější a nejvýkonnější plánovací systémy. Cílem IPC a ICAPS je dále také podpora výzkumu v oblasti automatického plánování a rozvrhování, vzdělávací aktivity, poskytování benchmarků a zdůrazňování aktuálních výzev oboru.

Kapitola 4

Klasické plánovací techniky

Pro klasické plánovací techniky platí, že každý uzel prohledávaného prostoru odpovídá částečnému plánu – tj. posloupnosti akcí (ve stavovém prostoru) nebo částečně uspořádané množině akcí (v prostoru plánů). Každé řešení, které je z daného uzlu dosažitelné, obsahuje všechny akce z tohoto částečného plánu.

4.1 Algoritmy prohledávání stavového prostoru

Přechodový stavový prostor klasického plánování lze reprezentovat jako orientovaný graf, jehož uzly odpovídají stavům a hrany mezi nimi představují provedené akce. Tento prostor lze poté prohledávat pomocí algoritmů prohledávání stavového prostoru. Cílem je v tomto případě nalézt cestu mezi počátečním stavem s_0 a koncovým stavem $g \in S_g$.

Podle směru prohledávání můžeme tyto algoritmy rozdělit na:

- dopředné prohledávání (forward search),
- zpětné prohledávání (backward search).

4.1.1 Dopředné plánování

V případě dopředného plánování prohledávání začíná v počátečním stavu, ze kterého hledá cestu do některého z cílových stavů. Základní pseudokód algoritmu dopředného plánování je uveden níže (rekurzivní verze viz algoritmus 2, iterativní verze viz algoritmus 3). Algoritmus je nedeterministický. Vstupem algoritmu je plánovací problém $P = (O, s_0, g)$, pokud je problém P řešitelný, pak algoritmus vrátí plán, v opačném případě vrátí neúspěch. Algoritmus je korektní a úplný. [3] Algoritmus dopředného plánování můžeme implementovat deterministicky několika způsoby (prohledávání do hloubky, do šířky, hladové prohledávání, …).

Stavový prostor plánovacích problému je ohromný a větvící faktor při prohledávání je tak vysoký, že samotné dopředné plánování je při řešení větších plánovacích problémů nepoužitelné. Algoritmus zmiňujeme především kvůli jeho důležitosti z hlediska historie automatického plánování a jeho pozdějších vylepšení pomocí heuristik a dalších metod, těmi se ale nebudeme v této práci dále zabývat.

4.1.2 Zpětné plánování

V případě zpětného plánování prohledávání postupuje opačným směrem než při plánování dopředném – prohledávání začíná v cíli, ze kterého jsou aplikovány inverze pro cíl relevantních akcí, pomocí kterých získáváme podcíle. Prohledávání končí ve chvíli, kdy je množina podcílů splněna v počátečním stavu. Akce a je relevantní pro cíl g, pokud platí:

- $g \cap \mathit{effects}^+(a) \neq \emptyset$
- $g^+ \cap \mathit{effects}^-(a) = \emptyset$
- $g^- \cap \mathit{effects}^+(a) = \emptyset$

Pro akci a relevantní k cíli g potom definujeme:

$$\gamma^{-1}(g, a) = (g - \mathit{effects}(a) \cup \mathit{precond}(a))$$

Stejně jako v předchozím případě je algoritmus nedeterministický. Algoritmus je korektní a úplný. [3] Pseudokód algoritmu zpětného plánování je uveden zde 1.

Function `Backward-search`(O, s_0, g)**:**

$\pi \leftarrow$ the empty plan
Loop
  if s_0 *satisfies* g **then**
    return π
  end
  $relevant \leftarrow \{a \mid a$ is a ground instance of an operator in O that is relevant for $g\}$
  if $relevant = \emptyset$ **then**
    return failure
  end
  nondeterministically choose an action $a \in relevant$
  $\pi \leftarrow a.\pi$
  $g \leftarrow \gamma^{-1}(g, a)$
EndLoop

Algoritmus 1: Algoritmus zpětného plánování - backward search [3]

Function `Recursive-forward-search`(O, s_0, g)**:**

if s *satisfies* g **then**
  return empty plan
end
$active \leftarrow \{a \mid a$ is a ground instance of an operator in O, and $precond(a)$ is true in $s\}$
if $active = \emptyset$ **then**
  return failure
end
nondeterministically choose an action $a_1 \in active$
$s_1 \leftarrow \gamma(s, a_1)$
$\pi \leftarrow$ `Recursive-forward-search`(O, s_1, g)
if $\pi \neq$ *failure* **then**
  return $a_1.p$
else
  return failure
end

Algoritmus 2: Rekurzivní verze algoritmu dopředného plánování [3]

Function `Forward-search`(O, s_0, g)**:**

$s \leftarrow s_0$
$\pi \leftarrow$ the empty plan
Loop
  if s *satisfies* g **then**
    return π
  end
  $applicable \leftarrow \{a \mid a$ is a ground instance of an operator in O,
   and $precond(a)$ is true in $s\}$
  if $applicable = \emptyset$ **then**
    return failure
  end
  nondeterministically choose an action $a \in applicable$
  $s \leftarrow \gamma(s, a)$
  $\pi \leftarrow \pi.a$
EndLoop

Algoritmus 3: Iterativní verze algoritmu dopředného plánování [3]

Kapitola 5

Neoklasické plánovací techniky

Hlavní rozdíl oproti klasickým technikám spočívá v tom, že na každý uzel prohledávaného prostoru pohlížíme jako na soubor několika částečných plánů. V tomto případě neplatí, že každé řešení, které je z daného uzlu dosažitelné, obsahuje všechny akce z tohoto částečného plánu. Existují i akce z částečných plánů, které se poté v řešení nevyskytují.

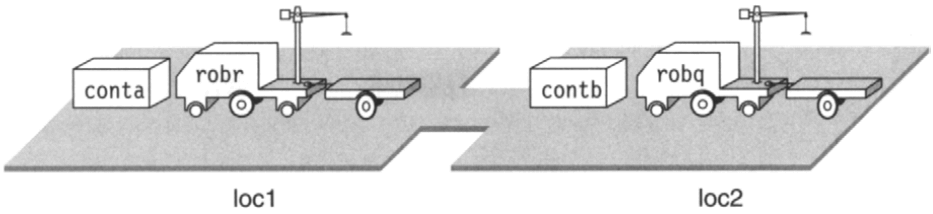
Objev neoklasických technik znamenal velký pokrok v oblasti automatického plánování. Jejich hlavní myšlenka je založena na komprimaci rozsáhlého stavového prostoru do struktur, které dokáží popsat plánovací prostor kompaktněji a lze v něm poté efektivněji hledat řešení. Mezi hlavní neoklasické techniky plánování patří následující:

- techniky využívající plánovací graf
- plánování jako splnitelnost (SAT)
- plánování jako splňování omezení (CSP)
- a další ...

5.1 Techniky využívající plánovací graf

Plánovací graf je struktura, která poskytuje efektivní způsob zobrazení toho, jaká množina základních atomů je eventuálně dosažitelná z počátečního stavu s_0, kterými akcemi a v kolika krocích. Pracuje s relaxovanou podmínkou dosažitelnosti.

Při popisu plánovacího grafu v této části práce vycházíme z [3] a [16]. Pro ukázku plánovacího grafu a souvisejících pojmů budeme používat zjednodušenou doménu DWR (Dock Worker Robots), která je znázorněna na obrázku 5.1.



move(r, l, l') ;; robot r at location l moves to a connected location l'
precond: at(r, l), adjacent(l, l')
effects: at(r, l'), ¬ at(r, l)

load(c, r, l) ;; robot r loads container c at location l
precond: at(r, l), in(c, l), unloaded(r)
effects: loaded(r, c), ¬ in(c, l), ¬ unloaded(r)

unload(c, r, l) ;; robot r unloads container c at location l
precond: at(r, l), loaded(r, c)
effects: unloaded(r), in(c, l), ¬ loaded(r, c)

Obrázek 5.1: Zjednodušená doména DWR (Dock Worker Robots) [3]

Pro přehlednost budeme používat zkratky pro základní atomy (např. r_1 pro at(robr, loc1), a_1 pro kontejner a na lokaci loc1, a_r pro kontejner a naložený na robotovi r, …) a akce (např. Mr12 pro move(robr, loc1, loc2), …).

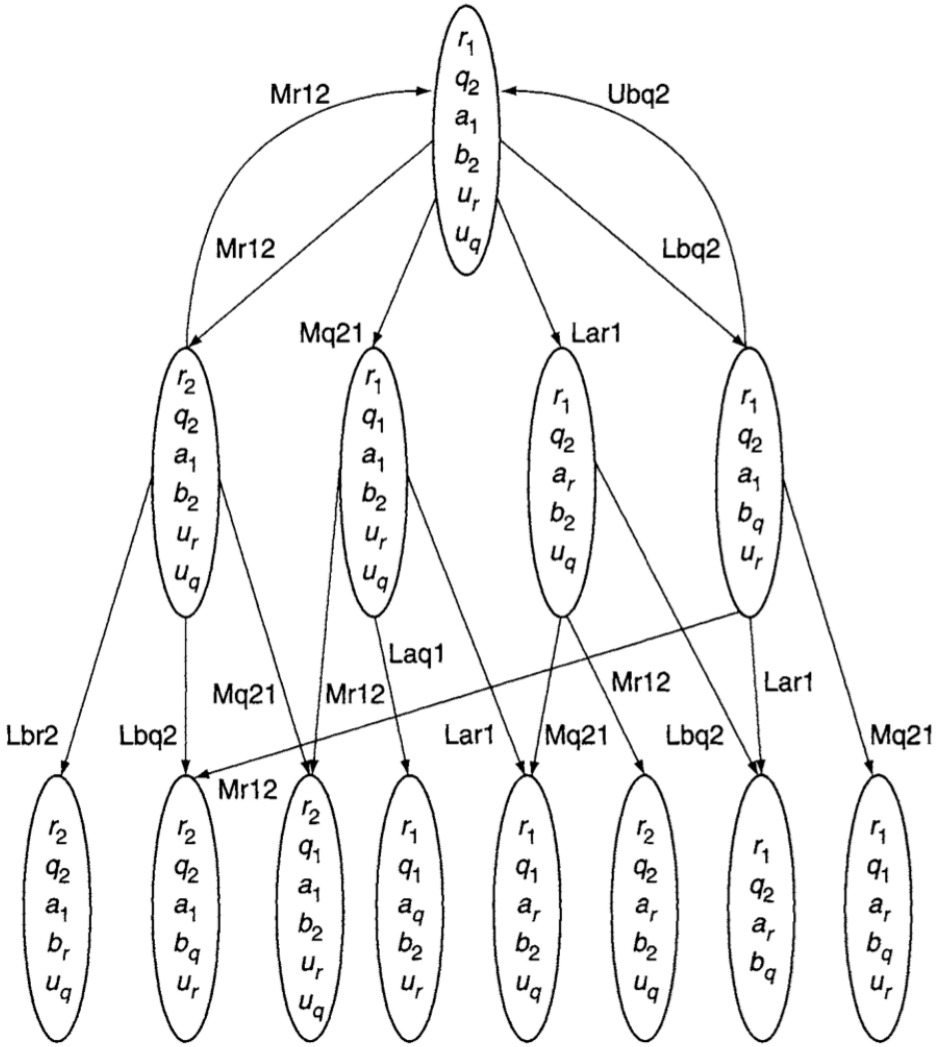
5.1.1 Dosažitelnost

Mějme množinu akcí A. Stav s je dosažitelný z počátečního stavu s_0, pokud existuje taková sekvence akcí z množiny A, která definuje cestu ze stavu s_0 do stavu s. Pro plánovací problém můžeme sestrojit strom dosažitelných stavů T. Uzly ve stromě T odpovídají stavům a hrany akcím z Σ. Kořenem stromu je počáteční stav s_0. Strom vytvořený do hloubky d řeší všechny plánovací problémy, jejichž cílový stav je dosažitelný z počátečního stavu s_0 za maximálně d akcí. Strom dosažitelných stavů obsahuje $O(k^d)$ uzlů, kde k je počet možných aplikovatelných akcí na stav.

Strom dosažitelných stavů můžeme upravit na graf dosažitelných stavů tím, že odstraníme duplikaci stavů ve stromu. Velikost grafu dosažitelných stavů může být ale také velmi velká, v nejhorším případě také $O(k^d)$. Příklad grafu dosažitelných stavů je zobrazen na obrázku 5.2.

Relaxovaná podmínka dosažitelnosti v plánovacím grafu je definována jako:

$$s_g \text{ je dosažitelný} \implies \text{je přítomný v plánovacím grafu}$$



Obrázek 5.2: Graf dosažitelných stavů [3]

Znamená to, že se v plánovacím grafu může vyskytnout i stav, který je ve skutečnosti nedosažitelný. Plánovací graf poté vychází z aproximace stavů pomocí sjednocení základních atomů ze všech stavů stromu / grafu dosažitelných stavů dosažitelných na dané úrovni.

5.1.2 Plánovací graf

Plánovací graf je orientovaný vrstevnatý graf, který obsahuje dva různé druhy uzlů a tři druhy hran. V grafu se pravidelně střídají predikátové vrstvy (obsahující uzly odpovídající predikátům s dosazenými konstantami – základním

atomům) a akční vrstvy (obsahující uzly odpovídající akcím). Hrany mohou spojovat pouze uzly ze sousedních vrstev. Graf můžeme rozdělit do úrovní, kdy každá úroveň obsahuje jednu akční a jednu predikátovou vrstvu, s výjimkou nulté úrovně. Tato nultá úroveň obsahuje pouze predikátovou vrstvu P_0, která obsahuje jeden uzel pro každý základní atom z počátečního stavu s_0.

První úroveň plánovacího grafu již obsahuje oba dva druhy vrstev: akční vrstvu A_1 a predikátovou vrstvu P_1.

- A_1 je množina akcí, jejichž předpoklady jsou uzly v predikátové vrstvě P_0
- P_1 je definována jako sjednocení množiny predikátů z P_0 a množiny obsahující všechny pozitivní efekty všech akcí z vrstvy A_1

Obdobným způsobem dochází k časové expanzi plánovacího grafu o další vrstvy: P_0 = predikáty pravdivé v čase 0, tj. v počátečním stavu s_0, A_1 = možné akce v čase 1, P_1 predikáty pravdivé v čase 1, A_2 = možné akce v čase 2, P_2 predikáty pravdivé v čase 2, atd.

Hrany v plánovacím grafu reprezentují vztahy mezi akcemi a predikáty (základními atomy). Můžeme rozlišit tři druhy hran:

- **hrany předpokladů:** akční uzly v akčních vrstvě A_i jsou spojeny s jejich předpoklady (preconds) ve vrstvě P_{i-1},
- **hrany pozitivních efektů:** akční uzly v akčních vrstvě A_i jsou spojeny s jejich pozitivními efekty ve vrstvě P_i,
- **hrany negativních efektů:** akční uzly v akčních vrstvě A_i jsou spojeny s jejich negativními efekty ve vrstvě P_i. Negativní efekty ze z plánovacího grafu neodebírají.

5.1.3 Plán v plánovacím grafu

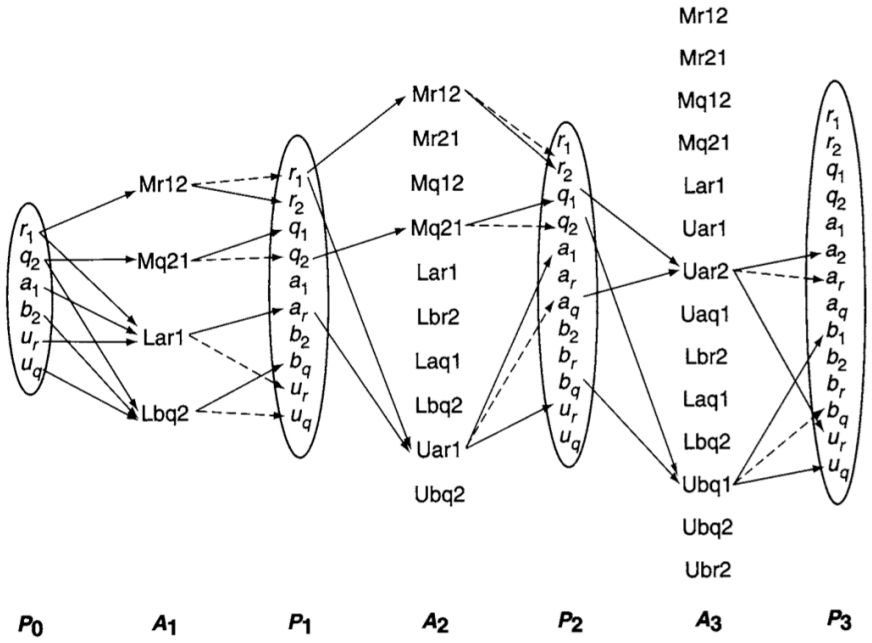
Zatímco klasické techniky pracují se sekvenčními plány, plánovací graf pracuje s vrstvenými plány. Plán v tomto případě je sekvence množin akcí.

$$\Pi = (\pi_1, \pi_2, \ldots, \pi_k)$$

Množiny akcí ve vrstveném plánu odpovídají vrstvám v plánovacím grafu, $\pi_i \subseteq A_i$. Vrstevnatý plán lze snadno převést na plán sekvenční tím, že vezmeme libovolnou permutaci akcí z π_1, za ní připojíme libovolnou permutaci akcí z π_2 atd.

5.1.4 Mutexy

Mutexy (vzájemná vyloučení) popisují nekompatibilitu některých dvojic akcí/atomů v plánovacím grafu. Jedná se o zpřesnění aproximace.



Obrázek 5.3: Ukázka plánovacího grafu ve zjednodušené doméně DWR [3]

Nezávislost akcí

Dvě akce a a b jsou nezávislé, pokud platí:

- $effects^{-}(a) \cap (precond(b) \cup effects^{+}(b)) = \emptyset$
- $effects^{-}(b) \cap (precond(a) \cup effects^{+}(a)) = \emptyset$

Množina akcí π je nezávislá, pokud každá dvojice akcí z π je nezávislá.

Akční mutexy

Dvě akce a a b jsou mutex v úrovni A_i, pokud platí alespoň jeden z následujících bodů:

- akce a a b jsou závislé
- nějaký z předpokladů akce a je mutex s nějakým z předpokladů akce b v předchozí predikátové vrstvě P_{i-1}

Závislost akcí je vlastností domény. Závislé akce jsou mutex v jakékoliv vrstvě plánovacího grafu. Druhá podmínka pro mutexy se ale v průběhu expanze grafu může změnit, proto se obecně může stát, že akce a a b jsou mutex v nějaké vrstvě A_i, ale v některé z následujících vrstev již mutexem nejsou. Množinu akčních mutexů na vrstvě A_i značíme jako μA_i.

Predikátové mutexy

Dva atomy p a q jsou mutex ve vrstvě P_i, pokud:

- každá akce z předchozí akční vrstvy A_i, která má atom p jako pozitivní efekt, je mutex s každou akcí z předchozí akční vrstvy A_i, která má atom q jako pozitivní efekt, a zároveň
- v předchozí akční vrstvě A_i není žádná akce, která by měla za pozitivní efekt p i q současně.

Množinu predikátových mutexů na vrstvě P_i značíme jako μP_i.

Level	*Mutex elements*
A_1	{Mr12} × {Lar1}
	{Mq21} × {Lbq2}
P_1	$\{r_2\} \times \{r_1, a_r\}$
	$\{q_1\} \times \{q_2, b_q\}$
	$\{a_r\} \times \{a_1, u_r\}$
	$\{b_q\} \times \{b_2, u_q\}$
A_2	{Mr12} × {Mr21,Lar1,Uar1}
	{Mr21} × {Lbr2,Lar1*,Uar1*}
	{Mq12} × {Mq21,Laq1,Lbq2*,Ubq2*}
	{Mq21} × {Lbq2,Ubq2}
	{Lar1} × {Uar1,Laq1,Lbr2}
	{Lbr2} × {Ubq2,Lbq2,Uar1,Mr12*}
	{Laq1} × {Uar1,Ubq2,Lbq2,Mq21*}
	{Lbq2} × {Ubq2}
P_2	$\{b_r\} \times \{r_1, b_2, u_r, b_q, a_r\}$
	$\{a_q\} \times \{q_2, a_1, u_q, b_q, a_r\}$
	$\{r_1\} \times \{r_2\}$
	$\{q_1\} \times \{q_2\}$
	$\{a_r\} \times \{a_1, u_r\}$
	$\{b_q\} \times \{b_2, u_q\}$
A_3	{Mr12} × {Mr21,Lar1,Uar1,Lbr2*,Uar2*}
	{Mr21} × {Lbr2,Uar2,Ubr2}
	{Mq12} × {Mq21,Laq1,Uaq1,Ubq1,Ubq2*}
	{Mq21} × {Lbq2,Ubq2,Laq1*,Ubq1*}
	{Lar1} × {Uar1,Uaq1,Laq1,Uar2,Ubr2,Lbr2,Mr21*}
	{Lbr2} × {Ubr2,Ubq2,Lbq2,Uar1,Uar2,Ubq1*}
	{Laq1} × {Uar1,Uaq1,Ubq1,Ubq2,Lbq2,Uar2*}
	{Lbq2} × {Ubr2,Ubq2,Uaq1,Ubq1,Mq12*}
	{Uaq1} × {Uar1,Uar2,Ubq1,Ubq2,Mq21}*
	{Ubr2} × {Uar1,Uar2,Ubq1,Ubq2,Mr12}*
	{Uar1} × {Uar2,Mr21*}
	{Ubq1} × {Ubq2}
P_3	$\{a_2\} \times \{a_r, a_1, r_1, a_q, b_r\}$
	$\{b_1\} \times \{b_q, b_2, q_2, a_q, b_r\}$
	$\{a_r\} \times \{u_r, a_1, a_q, b_r\}$
	$\{b_q\} \times \{u_q, b_2, a_q, b_r\}$
	$\{a_q\} \times \{a_1, u_q\}$
	$\{b_r\} \times \{b_2, u_r\}$
	$\{r_1\} \times \{r_2\}$
	$\{q_1\} \times \{q_2\}$

Obrázek 5.4: Akční a predikátové mutexy pro plánovací graf z obrázku 5.3. [3]

5.2 Graphplan

Algoritmus Graphplan [16] je plánovací algoritmus založený na plánovacím grafu. V algoritmu se střídají dvě fáze:

- **Expanze:** rozšiřování plánovacího grafu o další úroveň (akční a predikátovou vrstvu);
- **Extrakce:** pokus o nalezení plánu;

V první expanzi se plánovací graf rozšiřuje o další vrstvy až do té doby, dokud poslední predikátová vrstva neobsahuje bezmutexovou množinu atomů z cíle. V případě neúspěšné extrakce plánu pokračuje algoritmus expanzí o další úroveň. Po určité době dojde ke stabilizaci plánovacího grafu. Další úrovně grafu – akce, atomy, akční i predikátové mutexy – již zůstávají stejné. Tento stav stabilizace je v algoritmu Graphplan označován jako tzv. fixed point. Pseudokód části Expanze z Graphplanu je k dispozici viz algoritmus 4. Časová a prostorová složitost expanze plánovacího grafu je polynomiální vzhledem k velikosti řešeného problému (počtu akcí a atomů).

Část extrakce plánu se skládá z rekurzivních metod GP-Search a Extract, pomocí nichž algoritmus pro každou akční vrstvu hledá bezmutexovou množinu akcí splňující aktuální cíle. Využívá se zde tabulka ∇ se zakázanými cíli pro jednotlivé vrstvy plánovacího grafu. Pseudokód pro část extrakce je uveden viz algoritmus 5.

Výsledkem algoritmu je plán nebo indikace, že řešený problém nemá řešení. Algoritmus Graphplan je korektní a úplný. [3] Pseudokód hlavní části algoritmu Graphplan je uveden viz algoritmus 6. Konstrukce plánovacího grafu je nezávislá na cíli g, jednou vytvořený plánovací graf může být použit pro všechny problémy řešené ve stejné doméně (mající stejnou množinu operátorů O) se stejným počátečním stavem s_0.

Function
`Expand`$(< P_0, A_1, \mu A_1, P_1, \mu P_1, \ldots, A_{i-1}, \mu A_{i-1}, P_{i-1}, \mu P_{i-1} >)$**:**

$A_i \leftarrow \{a \in A \mid \mathit{precond}(a) \subseteq P_{i-1} \wedge \mathit{precond}^2(a) \cap \mu P_{i-1} = \emptyset\}$
$P_i \leftarrow \{p \mid \exists a \in A_i : p \in \mathit{effects}^+(a)\}$
$\mu A_i \leftarrow \{(a,b) \in {A_i}^2, a \neq b \mid$
$\mathit{effects}^-(a) \cap (\mathit{precond}(b) \cup \mathit{effects}^+(b)) \neq \emptyset \vee$
$\mathit{effects}^-(b) \cap (\mathit{precond}(a) \cup \mathit{effects}^+(a)) \neq \emptyset \vee$
$\exists (p,q) \in \mu P_{i-1} : p \in \mathit{precond}(a), q \in \mathit{precond}(b)\}$
$\mu P_i \leftarrow \{(p,q) \in {P_i}^2, p \neq q \mid \forall a, b \in A_i, a \neq b :$
$p \in \mathit{effects}^+(a), q \in \mathit{effects}^+(b) \implies (a,b) \in \mu A_i\}$
foreach $a \in A_i$ **do**
link a with:
precondition arcs to $\mathit{precond}(a)$ in P_{i-1}
positive arcs to $\mathit{effects}^+(a)$ in P_i
negative arcs to $\mathit{effects}^-(a)$ in P_i
end
return $< P_0, \ldots, A_{i-1}, \mu A_{i-1}, P_{i-1}, \mu P_{i-1}, A_i, \mu A_i, P_i, \mu P_i >$

Algoritmus 4: Část z algoritmu Graphplan – expanze plánovacího grafu. [3]

Function `Extract`(G, g, i)**:**
- **if** $i = 0$ **then**
  - **return** $<>$
- **end**
- **if** $g \in \nabla(i)$ **then**
  - **return** failure
- **end**
- $\pi_i \leftarrow$ GP-Search$(G, g, \emptyset, i)$
- **if** $\pi_i \neq$*failure* **then**
  - **return** π_i
- **end**
- $\nabla(i) \leftarrow \nabla(i) \cup \{g\}$
- **return** failure

Function `GP-Search`(G, g, π_i, i)**:**
- **if** $g = \emptyset$ **then**
  - $\Pi \leftarrow$ Extract$(G, \bigcup\{precond(a) | \forall a \in \pi_i\}, i-1)$
  - **if** $\Pi =$ *failure* **then**
    - **return** failure
  - **end**
  - **return** $\Pi. < \pi_i >$
- **else**
  - select any $p \in g$
  - resolvers $\leftarrow \{a \in A_i | p \in effects^+$ and $\forall b \in \pi_i : (a, b) \notin \mu A_i\}$
  - **if** *resolvers* $= \emptyset$ **then**
    - **return** failure
  - **end**
  - nondeterministically choose $a \in$ resolvers
  - **return** GP-Search$(G, g - effects^+(a), \pi_i \cup \{a\}, i)$
- **end**

Algoritmus 5: Část z algoritmu Graphplan – extrakce plánu. [3]

Algorithm `Graphplan`(A, s_0, g)**:**

$i \leftarrow 0$
$\nabla \leftarrow \emptyset$
$P_0 \leftarrow s_0$
$G \leftarrow P_0$
do
  $i \leftarrow i + 1$
  $G \leftarrow \text{Expand}(G)$
until $(g \subseteq P_i$ *or* $g^2 \cap \mu P_i \neq \emptyset)$ *or Fixedpoint(G)*
if $g \nsubseteq P_i$ *or* $g^2 \cap \mu P_i = \emptyset$ **then**
  return failure
end
$\Pi \leftarrow \text{Extract}(G, g, i)$
if *Fixedpoint(G)* **then**
  $\eta \leftarrow |\nabla_{(K)}|$
else
  $\eta \leftarrow 0$
end
while $\Pi = failure$ **do**
  $i \leftarrow i + 1$
  $G \leftarrow \text{Expand}(G)$
  $\Pi \leftarrow \text{Extract}(G, g, i)$
  if $\Pi =$ *failure or Fixedpoint(G)* **then**
    if $\eta = |\nabla_{(K)}|$ **then**
      return failure
    end
    $\eta \leftarrow |\nabla_{(K)}|$
  end
end
return Π

Algoritmus 6: Algoritmus Graphplan. [3]

Kapitola 6

Plánování jako SAT

S prvotní myšlenkou převodu klasického plánování do SAT (výrokové splnitelnosti) přišli již v roce 1992 Kautz a Selman. [30] Problém SAT je velmi známý problém a je v popředí zájmu výzkumu. Jednou z výhod převodu plánování na SAT je proto to, že každý pokrok v řešících algoritmech pro splnitelnost vede automaticky i ke zlepšení efektivity plánovačů, které převod využívají.

V prvních letech nebyly plánovače založené na převodu plánování do SAT konkurenceschopné vůči ostatním algoritmům a metodám používaným v automatickém plánování. V tabulce 6.1 je ukázáno porovnání výkonu SAT řešičů při řešení plánovacích problémů v doméně Airport ze soutěže IPC-4. Lze si všimnout, že mezi roky 1997 a 2001 došlo k velkému skoku ve výkonnosti SAT řešičů. (Například výslednou formuli pro problém p18 s přibližně 34 tisíci proměnnými a 750 tisíci klauzulemi nebyl do roku 1997 schopen vyřešit žádný z řešičů ve lhůtě 24 hodin. Řešiče vzniklé po roce 2001 již tento samý problém zvládly vyřešit v čase od tří do čtrnácti sekund.)

Toto velké zlepšení tedy znamenalo automaticky i zlepšení výkonu plánovačů založených na převodu do SAT, které se tímto staly efektivní a konkurenceschopné. V soutěži IPC poté dosahovaly dobrých výsledků zvláště v kategorii hledání optimálních plánů.

Základní myšlenkou použití SAT pro plánování je zakódování problému existence plánu určité délky $n \in \mathbb{N}$ do formulí výrokové logiky. Formule pro dané n je splnitelná právě tehdy, když existuje plán délky n. Hledání plánu se redukuje na testování splnitelnosti formulí pro různé hodnoty n.

Vzhledem ke složitosti 2.3 problému klasického plánování neexistuje garance jeho převodu do SAT v polynomiálním čase. Formule, které představují plánovací problém, mohou být v nejhorším případě exponenciální vzhledem k velikosti plánovacího problému. Většina tříd plánovacích úloh je však v praxi řešitelná pomocí plánů polynomiální délky, což znamená, že odpovídající formule mají polynomiální velikost. V praxi je tedy možné převádět problém plánování na problém splnitelnosti. [32]

wff	vars	clauses	sato 1997	satz 1997	zChaff 2001	jerusat 2002	MiniSat 2003	siege 2004
p05	3,656	31,089	13.23	0.61	0.01	0.01	0.02	0.01
p15	10,671	143,838	x	4.85	0.05	0.13	0.09	0.03
p18	34,325	750,269	x	x	13.92	6.59	2.55	4.85
p20	40,304	894,643	x	x	14.75	10.35	10.03	8.68
p28	249,738	13,849,105	x	x	846.72	79.59	27.80	12.74

Obrázek 6.1: Porovnání výkonu SAT řešičů při řešení plánovacích problémů v doméně Airport ze soutěže IPC-4. V tabulce je uveden celkový čas v sekundách potřebný pro vyřešení výsledné formule při hledání plánu optimální délky. (x = řešič nezvládl vyřešit problém do 24 hodin) [31]

Obecný postup při plánování jako SAT:

- plánovací problém zakódujeme jako výrokovou formuli,
- pomocí algoritmů pro řešení SAT zjistíme splnitelnost formule,
- zpětně dekódujeme řešení plánovacího problému z pravdivostního ohodnocení proměnných.

Mezi známé plánovače založené na převodu plánování do SAT patří například Blackbox [21], SATPlan [33] nebo Madagascar [34].

6.1 Výroková logika

V této sekci se věnujeme definování základních pojmů z výrokové logiky, vycházíme při tom z [35] a [32]. Základem výrokové logiky jsou výroky. Prvotním výrokem rozumíme jednoduchou oznamovací větu, u které má smysl hodnotit, zda je či není pravdivá. Předpokládáme takové výroky, které buď platí nebo neplatí (nic mezi tím).

Jazyk výrokové logiky

Jazyk výrokové logiky obsahuje:

- symboly označující výroky neboli prvotní formule (A, B, C),
- symboly pro logické spojky $(\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow)$,
- závorky.

Formule

Formule je konečná posloupnost symbolů z jazyka, která vznikne pomocí následujících pravidel:

- Prvotní formule je formule.
- Jsou-li A a B formule, pak i $(\neg A), (A \wedge B), (A \vee B), (A \Rightarrow B), (A \Leftrightarrow B)$ jsou formule.
- Každá formule vznikne pomocí dvou předchozích pravidel v konečně mnoha krocích.

Konjunktivní normální forma – CNF

- **Literál** je prvotní formule nebo negace prvotní formule.
- **Klauzule** je literál nebo disjunkce několika literálů.
- **Formule** je v **konjunktivní normální formě**, pokud je klauzulí nebo konjunkcí několika klauzulí.

Příklad formule v konjunktivní normální formě:

$$(A \vee \neg B \vee C) \wedge (\neg A \vee D) \wedge (B \vee C)$$

Pravdivostní ohodnocení

Pravdivostní ohodnocení w prvotních výroků z množiny P je jejich zobrazení do množiny $\{0, 1\}$.

$$w : P \to \{0, 1\}$$

Pokud pro prvotní výrok A platí $w(A) = 1$, označíme tento výrok jako pravdivý při ohodnocení w, v opačném případě $(w(A) = 0)$ jako nepravdivý při ohodnocení w.

Pravdivostní hodnota $w(F)$ formule F za předpokladu, že w je pravdivostní ohodnocení všech prvotních formulí vyskytujících se ve výrokové formuli F, se stanoví induktivně postupným určováním pravdivosti podformulí od prvotních až po finální formuli. Využíváme při tom pravdivostní tabulku pro jednotlivé logické spojky. Formule F je pravdivá při ohodnocení w, pokud $w(F) = 1$, v opačném případě je nepravdivá.

Splnitelnost formule

Formule je splnitelná právě tehdy, když existuje takové pravdivostní ohodnocení prvotních výroků z formule, při kterém je tato formule pravdivá.

Problém splnitelnosti – SAT

Problémem SAT myslíme úlohu, jejíž vstupem je výroková formule v CNF a výstupem je odpověď, zda existuje takové ohodnocení w, ve kterém je formule pravdivá. Problém SAT je známý NP-úplný problém.

6.1.1 Algoritmy řešení

Základním algoritmem využívaným v současnosti v systematických SAT řešičích je algoritmus Conflict-Driven Clause Learning (CDCL). Jedním z hlavních důvodů pro rozšířené používání SAT v mnoha aplikacích je to, že řešení SAT pomocí algoritmu CDCL je v praxi velmi efektivní. Od svého vzniku v polovině devadesátých let proto byly CDCL SAT řešiče úspěšně využity v mnoha praktických aplikacích. Algoritmus CDCL je založen na využití backtrackingu/backjumpingu, jednotkové propagace a učení klauzulí. Pseudokód algoritmu CDCL je uveden zde: 7. Detailní popis algoritmu je dostupný například v [32] (strany 131-149).

```
Algorithm CDCL(φ, ν):
    if UnitPropagation(φ, ν) == CONFLICT then
        return UNSAT
    end
    dl ← 0 (decision level)
    while not AllVariablesAssigned(φ, ν) do
        (x, v) = PickBranchingVariable(φ, ν)
        dl ← dl + 1
        ν ← ν ∪ {(x, v)}
        if UnitPropagation(φ, ν) == CONFLICT then
            β = ConflictAnalysis(φ, ν)
            if β < 0 then
                return UNSAT
            else
                Backtrack(φ, ν, β)
                dl ← β
            end
        end
    end


    return SAT
```

Algoritmus 7: Typický algoritmus CDCL [32]

6.2 Převod plánování na SAT

Při převodu plánování na SAT je klíčové zakódovat plánovací problém (stavy a přechody mezi nimi) do formule Φ. Konstrukce Φ je založená na dvou hlavních myšlenkách:

- omezení plánovacího problému na problém hledání plánu určité délky n (tento plán můžeme rozdělit na jednotlivé časové kroky $i, 0 \leq i \leq n$),
- transformace takto omezeného problému do problému splnitelnosti (stavy a přechody mezi nimi jsou namapovány na prvotní formule, které popisují stavy a akce v každém kroku).

Rozdělení plánování do jednotlivých časových kroků vypadá technicky takto: každý predikátový symbol s k proměnnými se rozšíří o proměnnou i určující časový krok na predikátový symbol s $k+1$ proměnnými. Například v doméně Blocks world máme predikátový symbol $on(x, y)$ popisující, že kostka x leží na kostce y, tento symbol bude nově $on(x, y, i), 0 \leq i \leq n$ - kostka x leží na kostce y v časovém kroku i. Operátory se rozšíří podobným způsobem taktéž o proměnnou i, například $put - down(x)$ bude nově $put - down(x, i), 0 \leq i \leq n$, kde i určuje v jakém časovém kroku je akce provedena.

Problém hledání plánu určité délky n může být rozšířen na problém hledání plánu délky $\leq n$ pro určité n. Pokud existuje plán, jeho délka musí být menší nebo rovna celkovému počtu stavů $|S|$, což znamená $n \leq 2^{|D|^{A_p}}$, kde $|D|$ je celkový počet konstant a A_p je maximální počet proměnných v predikátu. Hledání takto dlouhých plánů ale v praxi nelze uskutečnit a ani nedává smysl. Nadějí je, že může existovat plán relativně malé délky a není nutné procházet takto velký prostor. V praxi však není délka plánu známá předem. Vzhledem k tomu, že plánování jako SAT dovede pracovat pouze výše popsaným omezením problému (délkou plánu n), je třeba řešení problému opakovat pro různé hodnoty n. (Např. spouštět algoritmus iterativně pokaždé s jinou pevně nastavenou délkou n (2,4,8, …), dokud plán není nalezen.) [3]

Úspěšnost a efektivita přístupu plánování jako SAT je ovlivněna třemi významnými faktory: [32]

- Způsob reprezentace plánovacího problému pomocí výrokové formule Φ.
- Efektivita algoritmů pro testování splnitelnosti Φ.
- Algoritmy pro výběr hodnot n. (Jaké hodnoty n použít pro nalezení splnitelné formule Φ v co nejrychlejším čase?)

6.2.1 Zakódování plánovacího problému do problému splnitelnosti

Mějme klasický plánovací problém $P = (\Sigma, s_0, S_g)$ a funkci *Enc*, jejímiž argumenty jsou plánovací problém P a požadovaná maximální délka plánu n. Výstupem je výroková formule Φ.

$$Enc(P, n) = \Phi$$

Funkce *Enc* kóduje plánovací problém P do problému splnitelnosti, pokud platí následující: Φ je splnitelná právě tehdy, když pro problém P existuje řešení/plán délky n. Ve zkratce říkáme, že funkce *Enc* kóduje plánování do splnitelnosti.

Existují různé možnosti tohoto kódování. Dvě z nich zmíníme v následující sekci.

6.2.1.1 Lineární kódování

Lineární kódování lze považovat za nejzákladnější způsob kódování. Toto kódování bylo poprvé popsáno Kautzem a Selmanem v roce 1992. [30] Při jeho popisu budeme využívat označení f_i pro základní atom (fluent) v časovém kroku i a a_i pro akci provedenou v časovém kroku i. Formule Φ je sestavena podle následujících pravidel.

1. Počáteční stav s_0 je reprezentován jako konjunkce všech základních atomů f_0 (pokud jsou v s_0 přítomny) nebo jejich negace (pokud přítomny nejsou).

$$\bigwedge_{f \in s_0} f_0 \wedge \bigwedge_{f \notin s_0} \neg f_0$$

2. Koncový stav je reprezentován jako konjunkce základních atomů, které jsou definovány v g.

$$\bigwedge_{f \in g^+} f_n \wedge \bigwedge_{f \in g^-} f_n$$

3. Pokud platí akce a_i, potom její předpoklady musí platit v čase i a efekty v čase $i+1$. Pro každou akci $a \in A$ a pro každý časový krok $0 \leq i \leq n-1$ máme formuli:

$$a_i \implies \left(\bigwedge_{p \in precond(a)} p_i \wedge \bigwedge_{e \in effects(a)} e_{i+1} \right)$$

4. Klasické axiomy rámce: pokud akce a nemá atom f ve svých pozitivních nebo negativních efektech, v případě vykonání akce a_i se pravdivostní hodnota atomu f nezmění ($f_i \iff f_{i+1}$). V každém časovém kroku je provedena minimálně jedna akce.

$$f_i \wedge a_i \implies f_{i+1}$$

$$\bigvee_{a \in A} a_i$$

5. Exkluzivita akcí: v každém časovém kroku může nastat pouze jedna akce. U všech dvojic akcí zakážeme jejich provedení ve stejném časovém kroku. Pro každý časový krok $0 \leq i \leq n-1$ a každou dvojici akcí $a_i, b_i \in A, a_i \neq b_i$ máme formuli:

$$\neg a_i \vee \neg b_i$$

Výsledná formule vzniká konjunkcí všech formulí vzniklých podle výše uvedených pravidel. Počet proměnných využitých v tomto kódování je $O(n|A| + n|F|)$ a velikost (počet literálů) výsledné formule je $O(n|A|^2 + n|A||F|)$, kde n je požadovaná délka plánu, $|A|$ je celkový počet akcí a $|F|$ je celkový počet základních atomů v plánovacím problému. Kritický vliv na praktickou použitelnost tohoto způsobu kódování má arita (počet proměnných) operátorů a predikátů. Lze říci, že pokud se v doméně vyskytují operátory s aritou ≥ 3 (např. operátor přesunu – move(A, B, C)), je tento způsob kódování spíše nepoužitelný. [36]

Existují různé modifikace tohoto kódování, které mají za cíl redukovat počet proměnných a klauzulí výsledné formule.

- **Rozdělení operátorů:** operátory s velkou aritou jsou rozděleny na konjunkci více atomů s menší aritou.

přesun(a, b, c, 3)

$\Downarrow$

start(a, 3) $\wedge$

cíl(b, 3) $\wedge$

objekt(c, 3)

- **Vysvětlující axiomy rámce:** pokud atom f neplatí v časovém kroku i a platí v časovém kroku $i+1$, potom v časovém kroku i musí platit alespoň jedna z akcí, které mají f jako pozitivní efekt. Obdobně je to i v opačném případě, kdy atom f platí v časovém kroku i a neplatí v časovém kroku $i+1$. Pro každý atom f a každý časový krok $0 \leq i \leq n-1$ máme formule:

$$\neg f_i \wedge f_{i+1} \implies \left(\bigvee_{a \in A | f_i \in effects^+(a))} a_i \right)$$

$$f_i \wedge \neg f_{i+1} \implies \left(\bigvee_{a \in A | f_i \in effects^-(a))} a_i \right)$$

Lineární kódování (varianta s vysvětlujícími axiomy rámce) může být upraveno také na paralelní kódování 6.2.1.2 změnou pravidla exkluzivity akcí pouze na mírnější zákaz konfliktních akcí. [36]

6.2.1.2 Paralelní kódování

Počet časových kroků n má vliv na počet proměnných i velikost výsledné formule. Pro zlepšení těchto parametrů je možné povolit více akcí v jednom časovém kroku. Způsob kódování, který paralelní akce využívá, je založen na plánovacím grafu a nazývá se paralelní kódování. Formule Φ je sestavena podle plánovacího grafu pomocí následujících pravidel. Každý atom a akce vyskytující se v plánovacím grafu představuje prvotní formuli. Časové kroky i v tomto případě odpovídají vrstvám plánovacího grafu. Používáme značení vrstev, které bylo uvedeno v sekci 5.1.2, a_i značí akci a provedenou v akční vrstvě A_i a p_i základní atom z predikátové vrstvy P_i.

1. Všechny atomy z počátečního stavu jsou pravdivé ve vrstvě P_0.

$$\bigwedge_{p \in P_0} p_0$$

2. Všechny atomy z cílového stavu jsou pravdivé ve vrstvě P_n.

$$\bigwedge_{p \in g} p_n$$

3. Operátory implikují jejich předpoklady. Pro každou akci a ve vrstvě A_i máme formuli:

$$a_i \implies \left(\bigwedge_{p \in precond(a)} p_{i-1} \right)$$

4. Každý atom ve vrstvě P_i implikuje disjunkci všech akcí v předchozí vrstvě A_{i-1} jejichž je pozitivním efektem. Pro každý atom p ve vrstvě P_i máme formuli:

$$p_i \implies \left(\bigvee_{a \in A_{i-1} | p_i \in effects^+(a))} a_{i-1} \right)$$

5. Konfliktní akce jsou zakázány. Pro každou akční vrstvu A_i a každou dvojici akcí $a_i, b_i \in A_i$ pro které platí, že a_i a b_i jsou závislé, máme formuli:

$$\neg a_i \vee \neg b_i$$

Výsledná formule vzniká stejně jako v předchozím případě konjunkcí všech formulí sestavených podle výše uvedených pravidel. Počet proměnných využitých v tomto kódování je opět $O(n|A| + n|F|)$ a velikost výsledné formule odpovídá taktéž lineárnímu kódování (variantě s vysvětlujícími axiomy rámce). V nejhorším případě jsou všechny dvojice akcí navzájem konfliktní, tím pádem se ztrácí výhoda paralelních akcí. V praxi ale existuje mnoho plánovacích problémů, které paralelní akce mohou obsahovat, což vede k možnému nižšímu n a tudíž kratšímu kódování. [36]

Variant paralelního kódování existuje celá řada a není jednoznačně dáno, která z nich obecně poskytuje nejlepší výsledky. Některé plánovače (např. SatPlan verze z roku 2004) dávají na výběr několik variant kódování akcí (podrobněji viz [37]). Pozdější verze plánovače SatPlan-2006 do paralelního kódování zahrnuje i klauzule pro predikátové mutexy. [33]. Některé varianty kódování zahrnují klauzule i pro pozitivní a negativní efekty akcí nebo používají různé způsoby rozdělování operátorů. [38]

Obecným problémem kompilace plánování do SAT je velikost výsledných formulí a s tím spojená paměťová náročnost. Formule pro problémy ze soutěže IPC dosahují velikosti milionů klauzulí. (Například výsledná formule pro problém v doméně Blocks world s 15 kostkami a délkou plánu $n = 28$ obsahuje při kódování založeném na plánovacím grafu cca 2,5 milionu klauzulí. [39]) Různé nové varianty kódování mají proto většinou za cíl redukovat výsledný počet proměnných a klauzulí. (Je však potřeba poznamenat, že ne vždy má redukce jejich počtu i pozitivní vliv na rychlost řešení.)

Využití plánovacího grafu má zastoupení i v moderních variantách kódování. Důvodem je to, že získané akční a predikátové mutexy jsou velmi užitečné a zároveň ne nadbytečné, protože nejsou moderními SAT řešiči dedukovány nezávisle. [38]

Část II

Praktická část

Kapitola 7

Popis problému

Cílem praktické části práce je navrhnout a otestovat přístup pro línou kompilaci plánování do SAT. Řešený problém odpovídá problému klasického plánování, využíváme klasickou reprezentaci. Klasické plánování i jeho klasická reprezentace byly podrobně popsány v teoretické části.

Zadání problému popisuje trojice $P = (O, s_0, g)$, kde O je množina všech operátorů a určuje plánovací doménu, s_0 je počáteční stav a g je požadovaný cíl. Řešením problému je plán neboli posloupnost akcí, které při jejich postupné aplikaci dovedou systém z počátečního stavu s_0 do stavu, který splňuje všechny atomy z cíle g.

Téma práce je založeno na převodu plánování do SAT. Řešení plánovacího problému je proto získáno pomocí jeho převedení do formule výrokové logiky. Tato formule je následně vyřešena SAT řešičem a z pravdivostních hodnot proměnných je zpětně dekódován plán. V experimentální části porovnáváme navrženou metodu pro línou kompilaci s kompilací klasickou. Obě varianty jsou podrobně popsány v následujících kapitolách.

V případě převodu plánování do SAT je potřeba využít omezení plánovacího problému na problém hledání plánu určité délky n nebo $\leq n$. Abychom mohli porovnat výkon plánovače s normální a línou kompilací definujeme si dvě varianty problému (a s tím spojený způsob určování hodnot délek plánu n, pro které hledáme řešení), na které se odkazujeme v dalších kapitolách a kterými se budeme zabývat v experimentální části:

- **Varianta O:** hledáme optimální vrstvený plán o maximálně n vrstvách, který řeší zadaný plánovací problém. Jednotlivé vrstvy vrstveného plánu mohou obsahovat různé počty akcí. Pro stejné n proto mohou vznikat sekvenční plány různé délky. Cílem je získat plán, který obsahuje co nejmenší počet vrstev akcí (v našem případě tedy vznikl z plánovacího grafu s nejmenším možným počtem úrovní), na počet akcí v sekvenčním plánu nehledíme. Strategie volby hodnot délky plánu n je v tomto případě postupovat od 1 (případně jiné nejnižší hodnoty, která má v daném

případě smysl) postupně až do maximální délky n (případně do maximální délky menší než n, která má v daném případě smysl).

- **Varianta NI:** hledáme vrstvený plán s n vrstvami, který řeší zadaný plánovací problém. Vznikl tedy z plánovacího grafu o n úrovních. V této variantě dochází nejprve k jednorázové expanzi plánovacího grafu na požadovaný počet úrovní a až poté k sestavení a řešení formule Φ.

Kapitola 8

Plánovač

Výše popsaný problém byl řešen pomocí návrhu a implementace plánovače, který podporuje dvě varianty kompilace plánovacího problému do SAT – klasickou a línou kompilaci. Program je napsán v jazyce C++.

Výsledný plánovač se skládá ze tří hlavních částí:

- PDDL parser,
- plánovací část,
- SAT řešič.

Plánovač s oběma variantami kompilace využívá stejný níže popsaný PDDL parser a SAT řešič. Rozdíly nastávají v plánovací části a jsou podrobně popsány dále.

8.1 PDDL parser

PDDL parser zajišťuje převod pddl souborů do vnitřní reprezentace plánovací domény a problému. Jako parser PDDL souborů využíváme již hotovou implementaci parseru [40] od Thiaga P. Buena z univerzity University of São Paulo. Tento parser je integrován do kódu našeho programu. Parser je napsán v jazyce C++ a využívá Flex [41] a Bison [42]. Tento parser neumí zpracovat soubory pddl obsahující predikáty bez proměnných (např. handempty() v doméně Blocks World). Využíváme proto malou úpravu této domény s použitím konstanty pro robotické rameno (handempty(?r)). Tato úprava nemá vliv na počet atomů nebo akcí v plánovacím grafu.

8.2 SAT řešič

SAT řešič zajišťuje řešení výrokových formulí. Rozhodli jsme se využít jeden z mnoha již existujících SAT řešičů. Mezi kritéria pro výběr vhodného řešiče patřilo především:

1. **dostupnost:** volně přístupný kód,
2. **výkonnost:** efektivita řešiče,
3. **snadná integrace do našeho plánovače:** implementace v C++, snadno pochopitelný kód a jeho dokumentace,
4. **podpora inkrementálního způsobu řešení SAT:** při líné kompilaci je potřeba řešit výrokovou formuli opakovaně s postupným přidáváním dalších nových klauzulí,
5. **možnost řešení pro různé délky plánu n:** odebírání klauzulí popisujících cíle z řešené formule.

V naší práci využíváme řešič MiniSat [43]. Jedná se o minimalistický (původní verze měla pouze přibližně 600 řádků kódu), ale zároveň výkonný, systematický SAT řešič založený na algoritmu CDCL. Více detailů k implementaci MiniSatu je k dispozici v článku [44]. Tento řešič vyhovoval všem našim požadavkům. Co se týče posledního bodu – MiniSat sice nedovoluje odebírání klauzulí, ale je možné formuli řešit postupně s různými předpoklady pravdivostních hodnot některých proměnných (tzv. assumptions). Toto je ve spojení s podporou inkrementálního způsobu řešení pro náš účel dostačující. Kód SAT řešiče je oddělen od ostatních částí programu a je ho možné v případě potřeby snadno vyměnit za jiný. Lze také přidat SAT řešičů více a porovnávat jejich výsledky.

8.3 Plánovací část

Plánovací část je stěžejní částí plánovače. Dochází zde k vytvoření plánovacího grafu, jeho převodu do formule výrokové logiky a následně ke zpětnému převodu vyřešené formule do výsledného plánu. V případě líné kompilace zde probíhá také kontrola plánu a postupná úprava výrokové formule. Přesný postup práce plánovače se liší při využití líné a klasické kompilace a také mezi variantou problému O a NI.

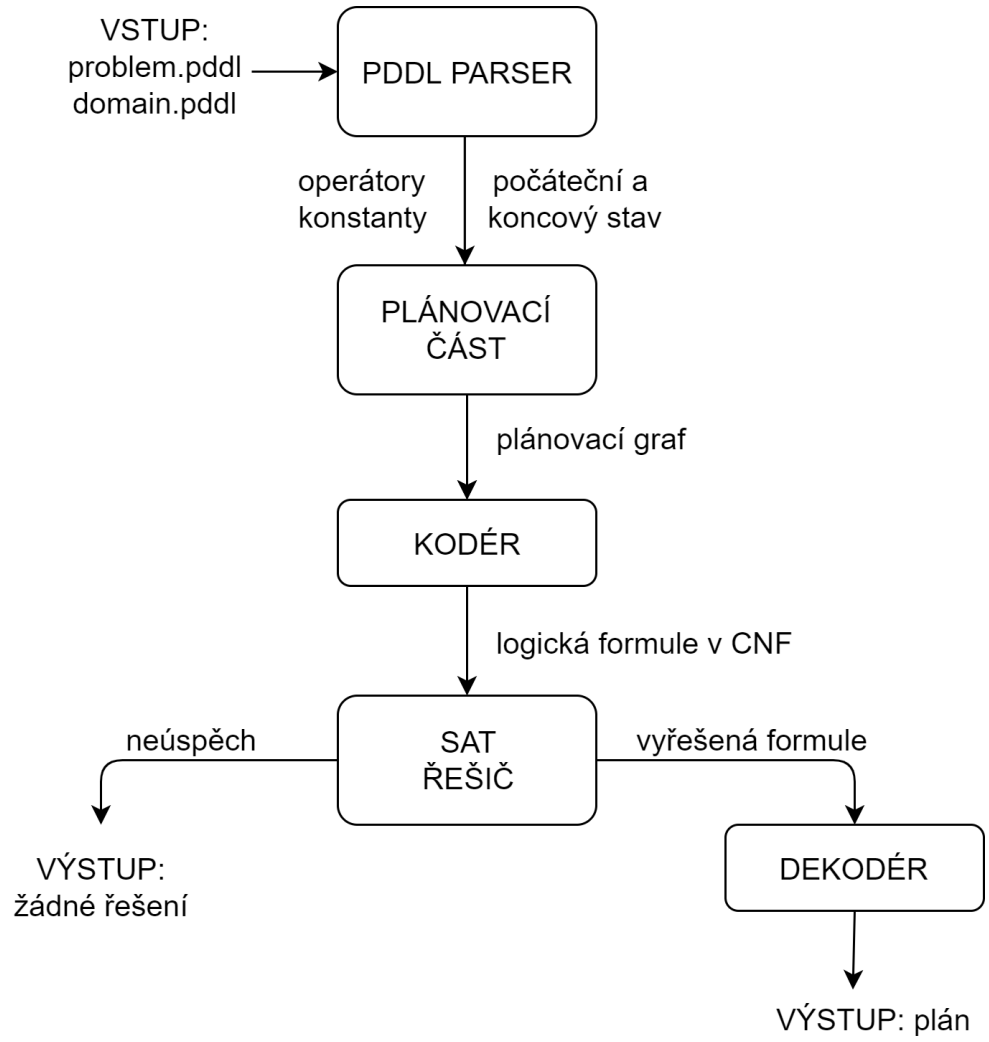
Na diagramu 8.1 je znázorněna práce plánovače s klasickou kompilací pro variantu problému NI. V tomto případě je vytvořen plánovací graf, který je jednorázově expandován na n úrovní. Poté je hledáno řešení pomocí převodu na SAT.

Při hledání optimálního plánu (varianta problému O) program využívá postupnou expanzi plánovacího grafu (a tím určuje, pro která n bude probíhat

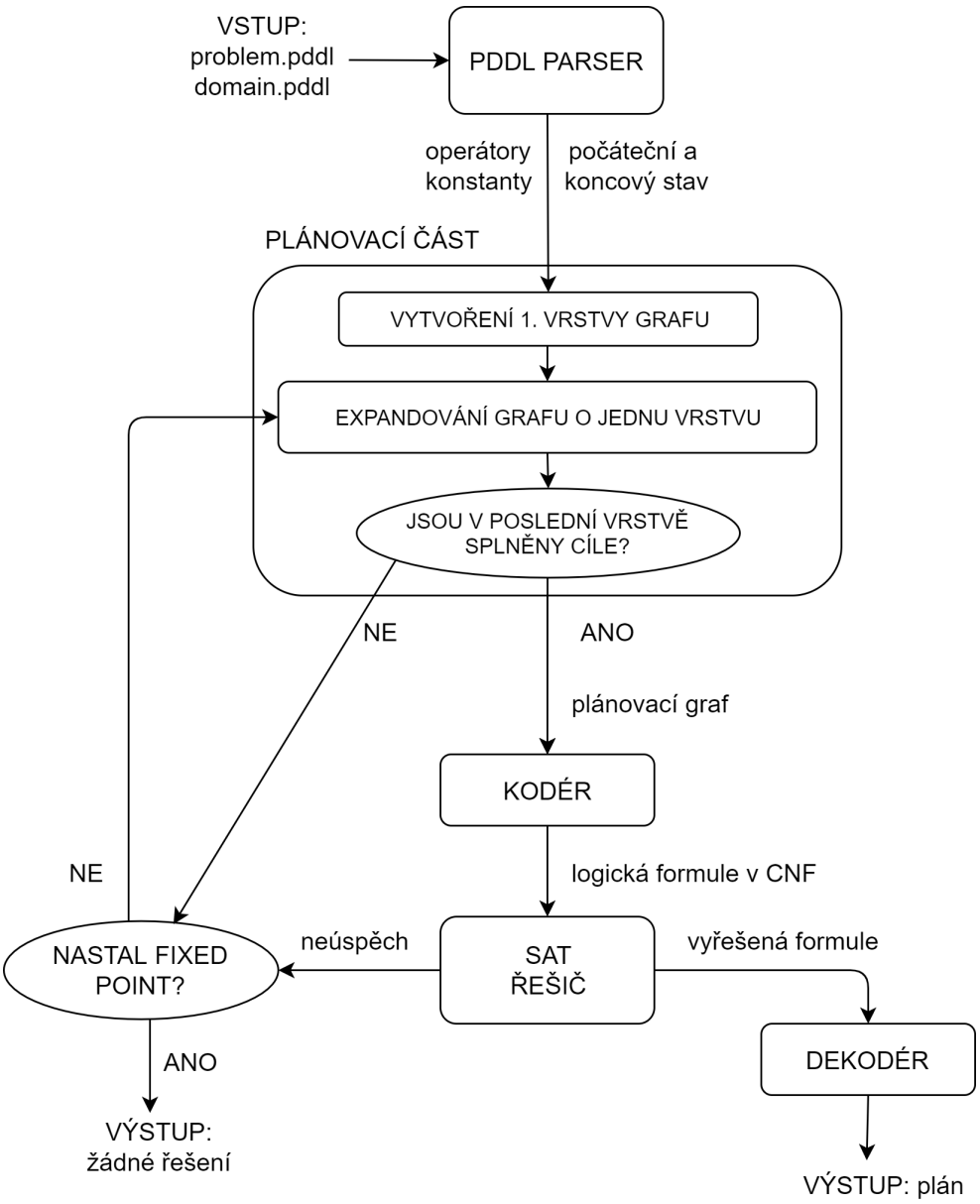
hledání plánu) obdobně jako algoritmus Graphplan 6. V první fázi je plánovací graf expandován do té doby, než poslední vrstva grafu obsahuje bezmutexovou množinu atomů cíle. Poté dochází k převodu plánovacího grafu do formule Φ a jejímu řešení. Pokud nedojde k nalezení plánu, dochází k expanzi grafu o další úroveň. Dojde-li ke stabilizaci grafu – nastane tzv. fixed point – program již s hledáním plánu pro vyšší n nepokračuje, protože je jasné, že řešení již nebude nalezeno. Postup je znázorněn na diagramu 8.2.

Na diagramu 8.3 je znázorněna práce plánovače s línou kompilací pro variantu problému NI. V tomto případě je vytvořen plánovací graf, který je jednorázově expandován na n úrovní. Při líné kompilaci nejsou používány mutexy, jejich konstrukce je proto při expandování grafu vynechána. Převod plánovacího problému do formule Φ probíhá líně. Formule je řešena opakovaně s postupným přidáváním klauzulí.

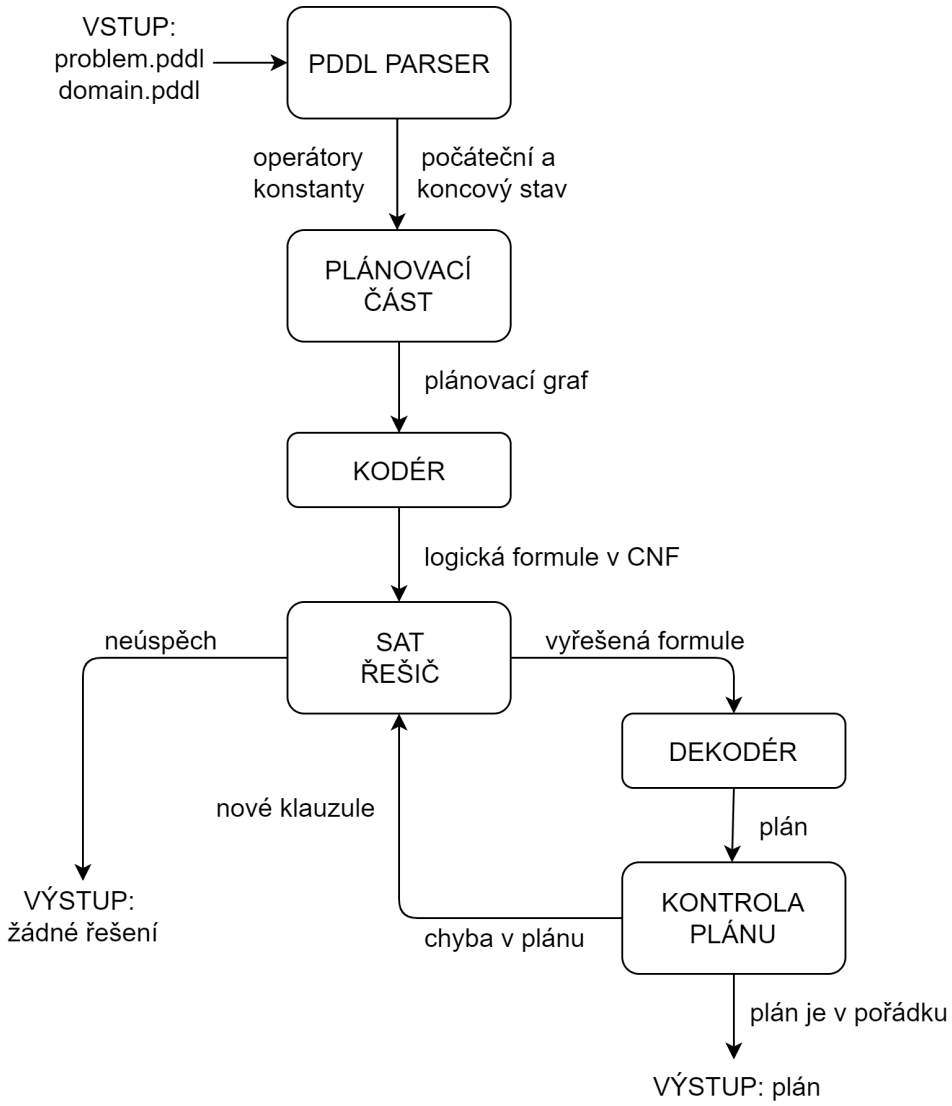
V případě líné kompilace a hledání optimálního plánu (varianta problému O) plánovač využívá postupnou expanzi plánovacího grafu (a tím určuje, pro která n bude probíhat hledání plánu). V první fázi je plánovací graf expandován do té doby, než poslední vrstva grafu obsahuje všechny atomy z cíle. Konstrukce mutexů je při expandování grafu vynechána, nelze tedy určit, zda množina atomů cíle je na dané vrstvě bezmutexová. Poté dochází k převodu plánovacího grafu do formule Φ a jejímu línému řešení. Pokud nedojde k nalezení plánu, dochází k expanzi grafu o další úroveň. Na rozdíl od klasické kompilace nelze určit, kdy došlo k opravdové stabilizaci plánovacího grafu. Expanzi grafu proto nelze ukončit předčasně. Místo využití fixed pointu zde musí být určen maximální počet úrovní, do kterého chceme graf expandovat. Postup je znázorněn na diagramu 8.4.



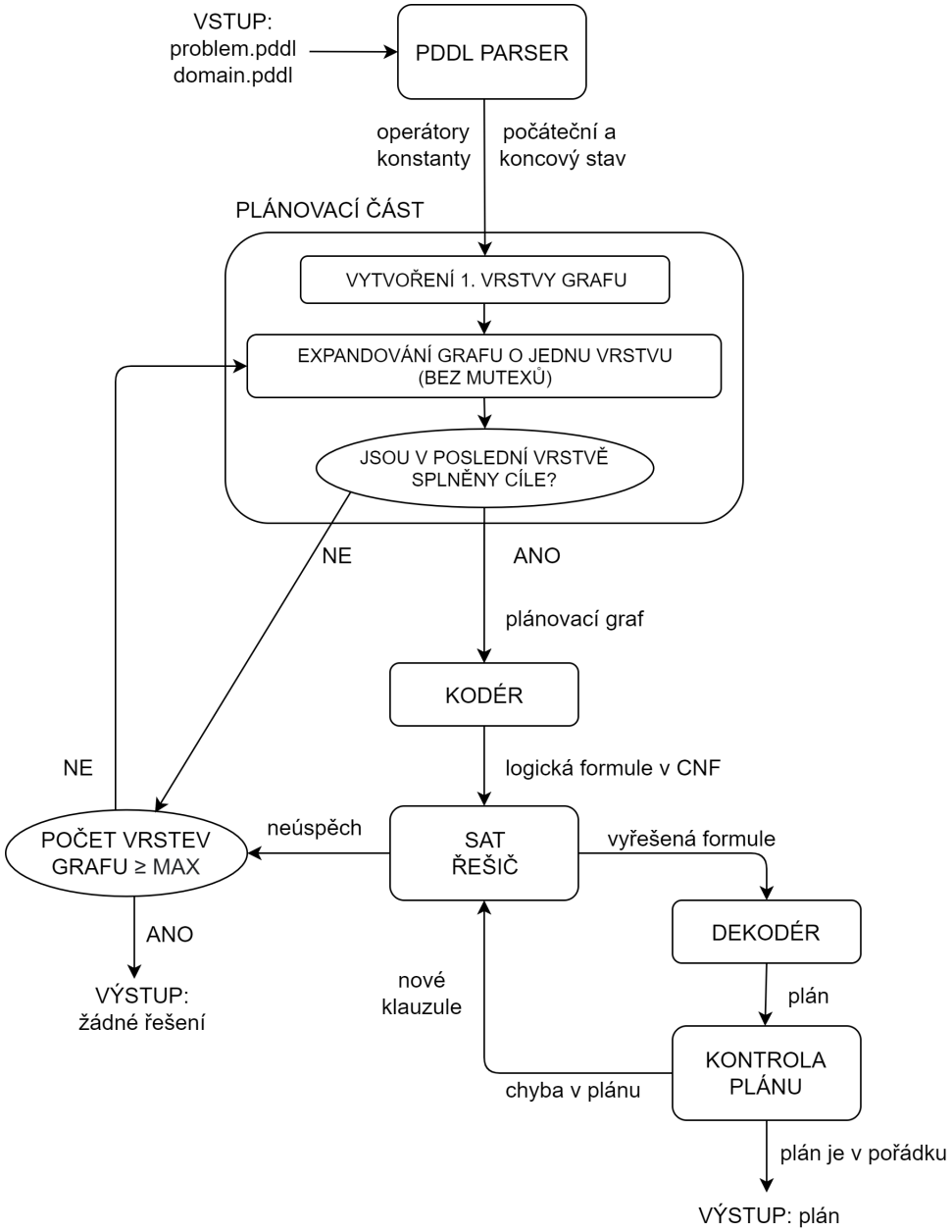
Obrázek 8.1: Diagram znázorňující práci plánovače s klasickou kompilací pro variantu problému NI.



Obrázek 8.2: Diagram znázorňující práci plánovače s klasickou kompilací pro variantu problému O.



Obrázek 8.3: Diagram znázorňující práci plánovače s línou kompilací pro variantu problému NI.



Obrázek 8.4: Diagram znázorňující práci plánovače s línou kompilací pro variantu problému O.

Kapitola 9

Klasická kompilace

Klasickou kompilací myslíme takový postup, kdy pro danou délku plánu n převedeme celý plánovací problém najednou do formule výrokové logiky Φ. Pokud je pomocí SAT řešiče nalezeno řešení této formule, plán Π_K vzniklý zpětným dekódováním z pravdivostních hodnot proměnných je řešením zadaného plánovacího problému. Pokud řešení nalezeno nebylo, plánovací problém nemá řešení.

9.1 Kódování

V případě klasické kompilace využíváme paralelní kódování založené na plánovacím grafu, konkrétní varianta s příklady je popsána níže.

Příklad: Uvažujeme plánovací problém v doméně Blocks World 2.3 s požadovanou délkou plánu $n = 2$.

Všechny atomy a akce, vyskytující se v plánovacím grafu pro daný problém, jsou označeny za prvotní formule – jako symboly pro prvotní formule používáme čísla 1,2,3, …, m, kde m je celkový počet atomů a akcí v plánovacím grafu. Ukázka:

- ontable(A) ve vrstvě $P_0 = 1$
- ontable(B) ve vrstvě $P_0 = 2$
- ontable(C) ve vrstvě $P_0 = 3$
- handempty() ve vrstvě $P_0 = 4$
- …
- pickup(A) ve vrstvě $A_1 = 5$
- pickup(B) ve vrstvě $A_1 = 6$

- pickup(C) ve vrstvě $A_1 = 7$
- ontable(A) ve vrstvě $P_1 = 8$
- ontable(B) ve vrstvě $P_1 = 9$
- ontable(C) ve vrstvě $P_1 = 10$
- …
- pickup(A) ve vrstvě $A_2 = 11$
- putdown(B) ve vrstvě $A_2 = 12$
- …
- on(B,A) ve vrstvě $P_2 = 13$

Přiřazení symbolů pro prvotní formule v této ukázce stejně jako výběr jednotlivých atomů a akcí je pouze ilustrační. Na stejném principu je vždy zpracován celý plánovací graf pro daný problém.

Výsledná formule je sestavena z formulí vzniklých podle následujících pravidel.

1. Všechny atomy z počátečního stavu jsou pravdivé ve vrstvě P_0.

$$\bigwedge_{p \in P_0} p_0$$

   Všechny atomy z cíle jsou pravdivé ve vrstvě P_n.

$$\bigwedge_{p \in g} p_n$$

   Příklad: Máme-li atomy počátečního stavu 1,2,3,4 a cílový atom 13, dostáváme formuli $1 \wedge 2 \wedge 3 \wedge 4 \wedge 13$.

2. Operátory implikují jejich předpoklady. Pro každou akci a ve vrstvě A_i máme formuli:

$$a_i \implies \left(\bigwedge_{p \in precond(a)} p_{i-1} \right)$$

   Příklad: Máme-li akci 44 s předpoklady 20, 25 a 27, získáváme:

$$44 \implies (20 \wedge 25 \wedge 27)$$

   Po převodu do CNF máme formuli:

$$(\neg 44 \vee 20) \wedge (\neg 44 \vee 25) \wedge (\neg 44 \vee 27)$$

3. Každý atom ve vrstvě P_i implikuje disjunkci všech akcí v předchozí vrstvě A_{i-1} jejichž je pozitivním efektem. Pro každý atom p ve vrstvě P_i máme formuli:

$$p_i \implies \left(\bigvee_{a \in A_{i-1} | p_i \in effects^+(a))} a_{i-1} \right)$$

   Příklad: Máme-li atom 60, který je pozitivním efektem akcí 55, 56 a 58, získáváme $60 \implies (55 \vee 56 \vee 58)$, po převodu do CNF máme formuli $\neg 60 \vee 55 \vee 56 \vee 58$

4. Všechny mutexové dvojice akcí jsou zakázány. Pro každou dvojici akcí $a, b \in M_A$ máme formuli:

$$\neg a \vee \neg b$$

   Všechny mutexové dvojice atomů jsou zakázány. Pro každou dvojici atomů $p, q \in M_P$ máme formuli:

$$\neg p \vee \neg q$$

   Příklad: Jsou-li akce 5 a 6 mutex, máme formuli: $\neg 5 \vee \neg 6$.

9.2 Získání plánu z vyřešené formule Φ

Řešení výsledné formule Φ koresponduje s řešením příslušného plánovacího problému. Formule Φ je splnitelná právě tehdy, když pro problém P existuje řešení/plán délky n.

$$\Phi \text{ je splnitelná} \iff P \text{ má řešení pro dané } n$$

Pokud je výsledná formule Φ splnitelná, lze z pravdivostního ohodnocení proměnných vždy zpětně dekódovat vrstevnatý plán Π_K, který lze převést na sekvenční plán π_K, který je řešením zadaného plánovacího problému. Součástí plánu jsou všechny akce, pro které platí, že ve výsledném ohodnocení formule byly proměnné odpovídající těmto akcím pravdivé. Pokud je formule Φ nesplnitelná, poté pro daný plánovací problém neexistuje plán délky $\leq n$.

Kapitola 10

Líná kompilace

V případě tzv. líné kompilace je v první fázi do formule převedena pouze částečná specifikace zadaného plánovacího problému. Plán Π_L vzniklý vyřešením formule a zpětným dekódováním z příslušných pravdivostních hodnot proměnných nemusí být validní. V některých případech nelze aplikovat postupně všechny akce z plánu Π_L a stav po provedení všech aplikovatelných akcí nemusí obsahovat všechny atomy z cílového stavu. Plán Π_L je proto potřeba vždy zkontrolovat a v případě zjištění chyb doplnit nově zjištěné klauzule do formule a opakovaně řešit pomocí SAT řešiče.

10.1 Kódování

Navržená metoda pro línou kompilaci využívá taktéž paralelní kódování založené na plánovacím grafu. Výsledná formule je sestavena podle následujících pravidel. (Tato pravidla jsou totožná s výše uvedenými pravidly 1 až 3 u klasické kompilace. Poslední pravidlo č. 4 zakazující mutexové akce a atomy je při tvorbě formule zcela vynecháno.)

1. Všechny atomy z počátečního stavu jsou pravdivé ve vrstvě P_0.

$$\bigwedge_{p \in P_0} p_0$$

   Všechny atomy z cíle jsou pravdivé ve vrstvě P_n.

$$\bigwedge_{p \in g} p_n$$

2. Operátory implikují jejich předpoklady. Pro každou akci a ve vrstvě A_i máme formuli:

$$a_i \implies \left(\bigwedge_{p \in precond(a)} p_{i-1} \right)$$

3. Každý atom ve vrstvě P_i implikuje disjunkci všech akcí v předchozí vrstvě A_{i-1} jejichž je pozitivním efektem. Pro každý atom p ve vrstvě P_i máme formuli:

$$p_i \implies \left(\bigvee_{a \in A_{i-1} | p_i \in effects^+(a))} a_{i-1} \right)$$

Níže uvádíme důvody, které při návrhu metody líné kompilace vedly právě k vynechání klauzulí pro akční a predikátové mutexy:

- Konstrukce mutexů je při tvorbě plánovacího grafu časově nejnáročnější částí [16], její vynechání způsobí úsporu času ve fázi před převodem plánování do logické formule.

- Při kódování plánování do SAT je velkým problémem počet proměnných a klauzulí ve výsledné formuli. Vynecháním klauzulí pro mutexy se celkový počet klauzulí signifikantně sníží. (Mutexů bývá v plánovacím grafu obecně značné množství.) V určitých případech může dojít i ke snížení počtu proměnných ve formuli.

- Při tvorbě plánu pro konkrétní problém může být potřeba pouze malá podmnožina všech mutexů. (Může zde existovat velmi mnoho mutexů, které se týkají části prostoru, jejíž prohledávání a popis pro nás při řešení tohoto konkrétního problému není důležité.)

10.2 Získání plánu z vyřešené formule Φ

Použité kódování pro línou kompilaci je neúplné. Řešení výsledné formule Φ nemusí korespondovat s řešením příslušného plánovacího problému. Pokud je formule Φ splnitelná, příslušný plán nemusí být validní a plánovací problém nemusí mít řešení. Pokud je však formule Φ nesplnitelná, pak platí, že ani plánovací problém pro dané n nemá řešení.

$$\Phi \text{ je splnitelná } \not\implies P \text{ má řešení pro dané } n$$

$$\Phi \text{ není splnitelná } \implies P \text{ nemá řešení pro dané } n$$

10.3 Kontrola plánu

Po líné kompilaci plánovacího problému $P = (O, s_o, g)$ do splnitelnosti, vyřešením formule a následně zpětným dekódováním z pravdivostních hodnot dostaneme vrstevnatý plán Π_L, z něhož snadno získáme sekvenční plán π_L. Tento plán ale nemusí být validní (akce nelze aplikovat postupně) a ani výsledný stav nemusí po aplikaci všech aplikovatelných akcí obsahovat všechny atomy z cíle g.

Ověřit, zda π_L je řešením plánovacího problém P, lze velmi jednoduchým způsobem. Vezmeme počáteční stav s_0, provedeme postupnou aplikaci akcí z plánu a ověříme, že výsledný stav obsahuje cíl g. Pokud zjistíme, že plán π_L je validním řešením zadaného plánovacího problému, problém byl úspěšně vyřešen a práce programu končí.

V opačném případě musíme identifikovat k jakým chybám v plánu došlo, doplnit nová zjištění do formule Φ a pokračovat s opakovaným řešením formule. Postup je znázorněn na již výše zmiňovaném diagramu 8.3.

10.3.1 Chyby v plánu

Pokud se zamyslíme obecně nad tím, jaké chyby se v plánu mohou vyskytovat a nad možnostmi, jak je opravit, zjistíme, že tato otázka není vůbec jednoduchá. V plánu může být akce, která zruší některý z předpokladů následující akce, která tím pádem nejde vykonat. Toto ovlivnění se nemusí stát jen ve stejné vrstvě akcí plánovacího grafu, ale i ve vrstvách libovolně daleko od sebe. Stejně tak zde může nějaká důležitá akce chybět. Teoreticky se může stát, že všechny akce v plánu jsou chybné a validní plán lze sestavit pouze z úplně jiných akcí. Oprava plánu navíc v našem případě musí jít vyjádřit úpravou formule Φ. Výše uvedené bylo bráno v úvahu již při návrhu metody pro línou kompilaci.

Námi využitý způsob kódování vykazuje dobré vlastnosti a případné chyby v plánu lze proto snadno identifikovat i odstranit doplněním klauzulí do formule Φ.

- Pravidlo číslo 1 zaručuje pravdivost atomů z počátečního stavu a cíle.
- Pravidlo číslo 2 zaručuje, že se nemůže stát, že by byla v plánu provedena akce, jejíž předpoklady by nebyly v předchozí predikátové vrstvě pravdivé.
- Pravidlo číslo 3 zaručuje, že pro každý pravdivý atom je pravdivá alespoň jedna akce, jež ho má za pozitivní efekt. Nemůže se tedy stát, že by se atom stal najednou pravdivý v některé vrstvě plánovacího grafu, aniž by ho způsobila provedená akce.

Všechny chyby, které v plánu mohou vzniknout, jsou zapříčiněny chybějícím pravidlem číslo 4 (z kódování pro klasickou kompilaci). V některé vrstvě

vrstevnatého plánu Π_L muselo dojít k porušení nezávislosti akcí (5.1.4) a vyskytuje se zde tedy alespoň jedna dvojice akcí, které jsou závislé. Pokud pro některou dvojici akcí a, b ze stejné vrstvy plánu Π_L platí alespoň jedna z těchto podmínek:

- $\mathit{effects}^-(a) \cap (\mathit{precond}(b) \cup \mathit{effects}^+(b)) \neq \emptyset$
- $\mathit{effects}^-(b) \cap (\mathit{precond}(a) \cup \mathit{effects}^+(a)) \neq \emptyset$

jedná se o závislé akce a, b. Tyto akce způsobují chybu v plánu a je potřeba je zakázat.

10.3.2 Přidávání nových klauzulí do formule Φ

Pro další iteraci řešení jsou do formule Φ doplněny klauzule popisující zákaz zjištěných dvojic závislých akcí a, b:

$$\neg a \vee \neg b$$

Metoda líné kompilace je založena na postupné identifikaci těchto závislých dvojic akcí a opakovaném řešení rovnice Φ s nově zjištěnými klauzulemi zakazujícími některé z dvojic závislých akcí, které se v plánu vyskytují. Níže uvádíme dvě mírně rozdílné varianty, které se liší v rychlosti přidávání nových klauzulí do formule Φ.

V krajním případě se může stát, že budou postupně identifikovány a přidány do formule úplně všechny dvojice závislých akcí a formule Φ bude tedy obsahovat úplnou specifikaci plánovacího problému. Ve většině případů ale dříve dojde buď k nalezení plánu nebo se naopak formule Φ stane nesplnitelnou, což znamená, že plánovací problém pro danou délku plánu n nemá řešení.

10.3.3 Varianty kontroly plánu a přidávání nových klauzulí do Φ

Varianta A – s prováděním akcí

Po ohodnocení proměnných formule Φ se zabýváme pouze těmi proměnnými, které jsou pravdivé a označují akce. Akce jsou seřazené podle pořadí, ve kterém byly v plánovacím grafu. Postupujeme od počátečního stavu s_0 a provádíme postupně pravdivé akce. Pravdivé akce, které nelze provést, počítáme. Pokud počet akcí, které nelze provést, překročí námi určenou hranici, ukončíme provádění plánu. V dosud pravdivých akcích najdeme závislé akce a přidáme klauzule do formule. Řešíme formuli Φ s novými klauzulemi. V této variantě děláme výjimku a ve vrstevnatém plánu dovolujeme v jedné vrstvě i akce, které jsou závislé, pokud tyto akce lze provést za sebou v takovém pořadí, v jakém se nalézají v plánovacím grafu.

Varianta N – pouze zjišťování nezávislosti

Po ohodnocení proměnných formule Φ se zabýváme pouze těmi proměnnými, které jsou pravdivé a označují akce. V této variantě se nesnažíme akce postupně provádět, ale rovnou mezi nimi najdeme všechny dvojice závislých akcí. To znamená, že v každé vrstvě plánu najdeme všechny možné dvojice akcí a pro každou dvojici zjistíme, jestli se jedná o závislé akce. Klauzule zakazující tyto dvojice akcí přidáme do formule Φ. Poté řešíme formuli Φ s novými klauzulemi.

10.4 Srovnání klasické a líné kompilace

Výsledky soutěže IPC (srovnávání výkonu plánovačů založených na různých principech při řešení různých variant plánovacích problémů v různých doménách) zatím naznačují, že je nepravděpodobné objevení principu, který by byl obecně nejlepší a poskytoval při řešení všech problémů lepší výsledky než do té doby známé přístupy. Stejně tak se lze domnívat, že líná kompilace bude pro některé problémy dosahovat lepších výsledků než kompilace klasická, zatímco na jiných problémech zlepšení výkonu nedosáhne nebo dokonce dojde ke zhoršení.

Neznamená to však, že objevování takovýchto nových přístupů, které neposkytnou konzistentně lepší výkon, je zbytečné. Spíše je důležité využít pozitivní stránky každého z nich. (Na tomto principu ostatně fungují v posledních letech úspěšné portfoliové plánovače.) Otevírá se zde otázka, jak pro konkrétní problém předem identifikovat, jaký způsob kompilace je vhodnější zvolit.

Předpokládáme, že líná kompilace bude lépe fungovat ve výrazně paralelních doménách a naopak může být horší v doménách, kde se vyskytuje málo paralelních akcí. Zde se může častěji stávat, že plán Π_L nebude validní z důvodu závislosti akcí. Proces postupného doplňování klauzulí a řešení rovnice Φ bude opakován tolikrát, že se v konečném součtu nemusí vyplatit.

I ve stejné doméně však záleží na každém konkrétním problému, který zrovna řešíme. Přestože vynechání konstrukce mutexů v plánovacím grafu přináší časovou úsporu, ztráta informací s sebou také nese určité nevýhody. Domníváme se, že v některých případech (např. nesmyslný cíl – robot, který se nachází na dvou místech zároveň) může být tato ztráta informací významnější než úspora času při konstrukci plánovacího grafu. V případě řešitelného plánovacího problému a varianty problému O může absence informací o mutexech v plánovacím grafu způsobovat to, že plánovač s línou kompilací bude (na rozdíl od plánovače s línou kompilací) hledat řešení i na vrstvách, kde to nemá smysl, protože zde množina atomů cíle obsahuje mutexy.

V případě varianty problému NI je také velmi důležitá volba požadované délky plánu n. Zatímco plánovač s klasickou kompilací může díky informacím z plánovacího grafu skončit s řešením již před dosažením maximálního n, plánovač s línou kompilací tyto informace nemá k dispozici. Pokud se jedná

o problém, který nemá řešení, neadekvátně volená n z tohoto důvodu mohou působit větší časové ztráty v případě líné kompilace.

Obecně lze říci, že princip líné kompilace nabízí oproti klasické kompilaci určité výhody, které jsou však vykoupeny i některými nevýhodami. Experimentálnímu srovnání výkonu plánovače s klasickou a línou kompilací se budeme věnovat v poslední části práce.

Kapitola 11

Experimentální vyhodnocení

V následující kapitole jsou uvedeny výsledky výše popsaného plánovače s klasickou a línou kompilací. Výsledky experimentů jsou znázorněny v tabulkách a vizualizovány pomocí grafů. V některých případech pro přehlednost uvádíme stejné údaje ve formě tabulky i grafu. Všechny časové údaje uvádíme v sekundách. Experimenty jsou rozděleny do čtyř sekcí odpovídajících doménám, ve kterých byly prováděny. V každé doméně jsou experimenty ještě dále děleny do tří částí:

- experimenty zaměřené na výslednou formuli Φ pro různé varianty kompilace (počet proměnných a klauzulí),
- čas řešení problémů různé obtížnosti pomocí plánovače s různými variantami kompilace,
- podrobnější měření u vybraných problémů.

Všechny experimenty uvedené v této kapitole byly provedeny s variantou problému O (postupná expanze plánovacího grafu), kvůli odstranění vlivu volby požadované délky plánu n ve variantě problému NI.

Na konci kapitoly je uvedeno krátké shrnutí výsledků. V tabulkách a grafech používáme následující zkratky pro popis použitých variant plánovače a problémů:

- L – A: líná kompilace, kontrola plánu varianta A,
- L – N: líná kompilace, kontrola plánu varianta N,
- K: klasická kompilace,
- O: zadání problému varianta O.

V doménách Logistika a Blocks World ukazujeme na některých grafech i srovnání našeho plánovače s plánovačem BlackBox. Tento plánovač byl vybrán proto, že je založen na stejném principu jako náš plánovač (vytvoření plánovacího grafu a jeho převedení do splnitelnosti) a účastnil se soutěže IPC 2000, ze které pochází tyto dvě domény. Časy uvedené pro BlackBox byly naměřeny přímo v soutěži IPC 2000. Srovnání je pouze ilustrativní, při provádění experimentů nelze dosáhnout stejných podmínek, což si ani za cíl neklademe. (Experimenty provádíme na jiné architektuře než byla použita v soutěži IPC apod.) Časy plánovače BlackBox mohou kromě porovnání s naším plánovačem poskytnout také představu o rozdílu v obtížnosti jednotlivých problémů.

11.1 Popis domén

Výše popsaný plánovač testujeme na problémech z několika domén ze soutěže IPC (ročník 1998, 2000 a 2002).

Logistika a Blocks World

Domény Logistika a Blocks World byly využívány v soutěži IPC v roce 2000. Doméně Blocks world jsme se podrobně věnovali v teoretické části v kapitole 2.2.1. Doména logistiky popisuje svět, ve kterém existují města rozdělená na jednotlivé lokace. Po městech se pohybují dodávky, které převáží balíčky mezi jednotlivými lokacemi. Dále se zde vyskytují letiště (každé letiště je také jedna z lokací) a letadla, která převáží balíčky mezi letišti. Cílem je dopravit všechny balíčky z jejich startovní do cílové lokace. Kapacita letadel a dodávek je neomezená. Doménu vytvořili Bart Selman a Henry Kautz na základě dřívější domény od Manuela Velosa.

Tyto dvě domény byly zvoleny záměrně proto, že jsou navzájem velmi odlišné. Zatímco doména logistiky je ze své podstaty hodně paralelní (jednotlivé dodávky a letadla se pohybují nezávisle na sobě), v doméně Blocks World naopak vůbec neexistují paralelní akce (robotické rameno zvládne manipulovat vždy pouze s jednou kostkou). Další odlišností je fakt, že v doméně Blocks world je mnohem větší interakce mezi jednotlivými atomy z cíle než v doméně Logistika. Kostky jsou stavěny postupně na sebe a například vynechání jedné kostky již později nelze jednoduše napravit, protože lze přidávat a ubírat kostky pouze z vrcholu věže. Velmi zde záleží na pořadí, ve kterém bude s jednotlivými kostkami manipulováno. V doméně Logistika na sobě naopak pozice jednotlivých balíčků nezávisí.

Jednotlivé problémy jsou označeny podle vzoru probNAME-x-y. Číslo x odkazuje na velikost problému, čím větší číslo, tím více konstant problém obsahuje. V případě domény Blocks world odpovídá počtu kostek.

ZenoTravel

Doména ZenoTravel pochází ze soutěže IPC 2002 [45] a popisuje svět, ve kterém je cílem přepravit lidi v letadlech ze startovních do cílových destinací. Letadla potřebují na létání palivo a mohou využívat dva druhy pohybu – rychlý a pomalý let. Při využití rychlého letu se spotřebuje paliva více.

Mystery

Doména Mystery pochází z prvního ročníku soutěže IPC [10]. Obsahuje rovinný graf uzlů, přičemž na každém uzlu se mohou nacházet vozidla, předměty a také určité množství paliva. Předměty lze nakládat na vozidla (až do výše do jejich kapacity) a převážet mezi uzly za předpokladu, že na uzlu je dostatečné množství paliva. Cílem je dostat předměty ze startovních do cílových uzlů. Jméno Mystery odkazuje na zvláštnost této domény – aby se zamaskoval skutečný význam predikátů a akcí, všechny názvy jsou přejmenovány. Uzly jsou pojmenovány jako emoce, kapacity vozidel jako geografické pojmy atd. Doménu vytvořil Drew McDermott.

11.2 Experimenty v doméně Logistika

Následující grafy a tabulky popisují výsledky experimentů prováděné v doméně Logistika. Problémy probLOGISTICS-4-0 až probLOGISTICS-15-1 byly použity v prvním kole soutěže IPC 2000. Problémy probLOGISTICS-18-0 až probLOGISTICS-20-1 pochází z druhého kola stejného ročníku soutěže. Všechny použité problémy mají řešení. Všechny zkoumané problémy z domény Logistika se podařilo vyřešit plánovačem s klasickou i línou kompilací v časovém limitu, který byl určen s ohledem na výsledky plánovačů ze soutěže IPC jako 15 minut na jeden problém. V experimentech jsme se zaměřovali na parametry výsledné formule Φ a na zjištění potřebného času pro vyřešení problému.

11.2.1 Výsledná formule Φ

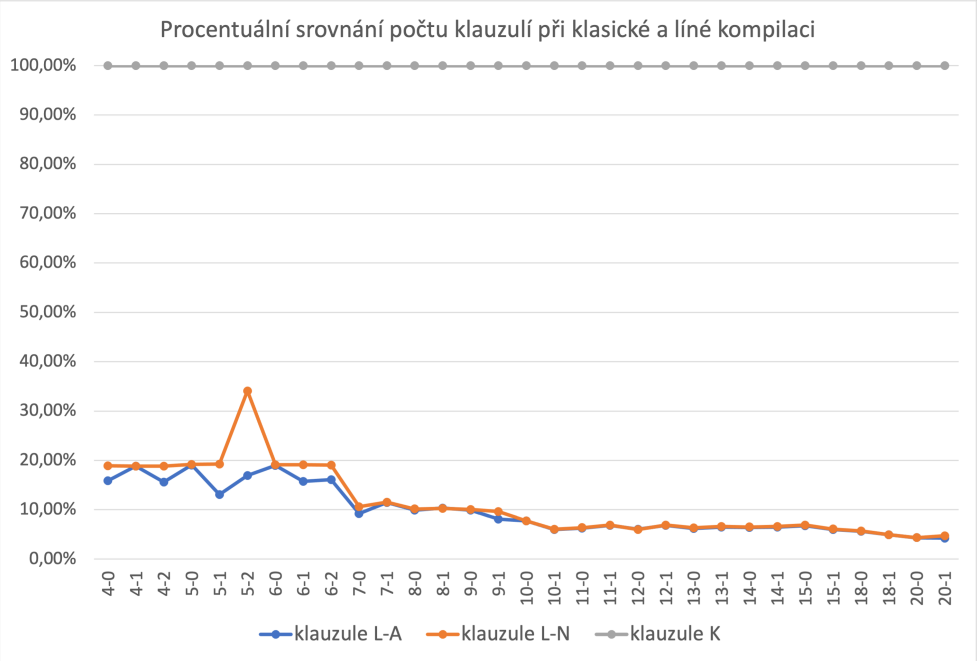
Tabulka 11.1 zobrazuje počet proměnných a klauzulí ve výsledné formuli Φ pro různé druhy kompilace. Formule pro řešené problémy obsahovaly od 348 do 30770 proměnných. Proměnné představují atomy a akce vyskytující se ve výsledném plánovacím grafu pro daný problém, jejich počet tedy odpovídá součtu počtu atomů a akcí.

V případě klasické kompilace obsahovala formule Φ od cca 1500 až do dvou milionů klauzulí. Ve všech případech to bylo znatelně více klauzulí než při využití líné kompilace (varianta A i N). Experimenty ukázaly, že v případě líné kompilace stačilo pro vyřešení formule ve většině případů méně než 20 % klauzulí oproti klasické kompilaci, pro větší problémy se toto číslo pohybovalo

kolem 6 %. Počet klauzulí při líné kompilaci se mezi jednotlivými variantami A a N příliš neliší. Formule Φ při variantě A obsahuje zpravidla o něco méně klauzulí než při variantě N. Podrobné výsledky – procentuální srovnání pro všechny problémy – jsou znázorněny na grafu 11.1.

probLOGISTICS	proměnné	klauzule K	klauzule L-A	klauzule L-N
4-0	1536	17590	2791	3330
4-1	1536	17590	3311	3316
4-2	1536	17590	2745	3311
5-0	1536	17655	3366	3383
5-1	1536	17590	2297	3385
5-2	348	1467	249	500
6-0	1536	17590	3339	3368
6-1	1536	17590	2772	3361
6-2	1536	17590	2836	3350
7-0	4106	94460	8699	10014
7-1	4540	96300	11017	11085
8-0	3672	86973	8623	8842
8-1	4106	94460	9760	9701
9-0	3672	86973	8624	8740
9-1	3238	79486	6440	7641
10-0	8769	291483	22618	22515
10-1	6590	274053	16480	16630
11-0	7316	292393	18419	18649
11-1	8768	325844	22340	22588
12-0	6590	274053	16560	16476
12-1	8768	323823	22194	22432
13-0	12896	589315	36538	37295
13-1	12896	562518	36573	37212
14-0	11592	507694	32529	33069
14-1	12896	560283	36304	37044
15-0	12896	542748	36842	37437
15-1	11592	541629	32601	32999
18-0	21442	1133404	64203	64941
18-1	17786	1057252	51972	52472
20-0	28330	1996557	86657	87232
20-1	30770	2086989	89002	98863

Tabulka 11.1: Počty proměnných a klauzulí výsledné formule Φ pro různé druhy kompilace pro problémy z domény Logistika. (varianta problému O)



Obrázek 11.1: Procentuální srovnání počtu klauzulí výsledné formule Φ pro problémy z domény Logistika pro různé druhy kompilace. Počet klauzulí při klasické kompilaci bereme jako 100 %.

11.2.2 Čas potřebný pro vyřešení problémů

Mezi hlavní ukazatele úspěšnosti plánovače patří schopnost nalézt řešení zadaných problémů v co nejkratším čase. Hlavní část našich experimentů se proto zabývala právě touto problematikou. Na grafech 11.2 (normální měřítko) a 11.3 (logaritmické měřítko) a v tabulce 11.2 je vidět celkové srovnání časů pro náš plánovač s klasickou a línou kompilací (varianty A a N) a plánovače BlackBox. Na grafu 11.4 je znázorněno procentuální srovnání časů.

Ve všech zkoumaných problémech byla líná kompilace (varianta A i N) rychlejší než kompilace klasická. Na vyřešení problémů stačilo v případě varianty A průměrně pouze 22,7 % času oproti klasické kompilaci, pro variantu N bylo toto číslo 27,8 %.

Pokud se zaměříme na detailnější srovnání dvou variant líné kompilace, všimneme si, že tyto dvě varianty jsou, co se týče potřebného času pro vyřešení problémů, přibližně shodné. Varianta A dosahovala obecně mírně lepších výsledků, při řešení některých problémů ale byla naopak mírně rychlejší varianta N. Rozdíl v časech řešení byl nejvýraznější v případě problému probLOGISTICS-20-1. Zde byla varianta A výrazně lepší než varianta N (153 sekund oproti 573 sekundám).

Pokud porovnáme výkon našeho plánovače s hodnotami naměřenými pro

plánovač BlackBox, v případě menších problémů byl BlackBox lepší než náš plánovač s klasickou kompilací a srovnatelně rychlý jako náš plánovač s línou kompilací. Na středních a těžších problémech již BlackBox potřeboval času více. Z grafů je patrné, že zvláště od problému probLOGISTICS-10-0 dosahoval BlackBox výrazně horších výsledků než náš plánovač s klasickou i línou kompilací. Průměrně BlackBox potřeboval 291 % času oproti našemu plánovači s klasickou kompilací.

Do druhého kola soutěže IPC 2000 v doméně Logistika postoupilo pouze 6 plánovačů ze 14 a BlackBox již nebyl mezi nimi. Časy řešení pro plánovač BlackBox pro problémy probLOGISTICS-18-0 až probLOGISTICS-20-1 jsou tedy neznámé. Problémy probLOGISTICS-25-0 a těžší již byly příliš náročné i pro náš plánovač, který je nebyl schopný vyřešit v určeném časovém limitu 15 minut.

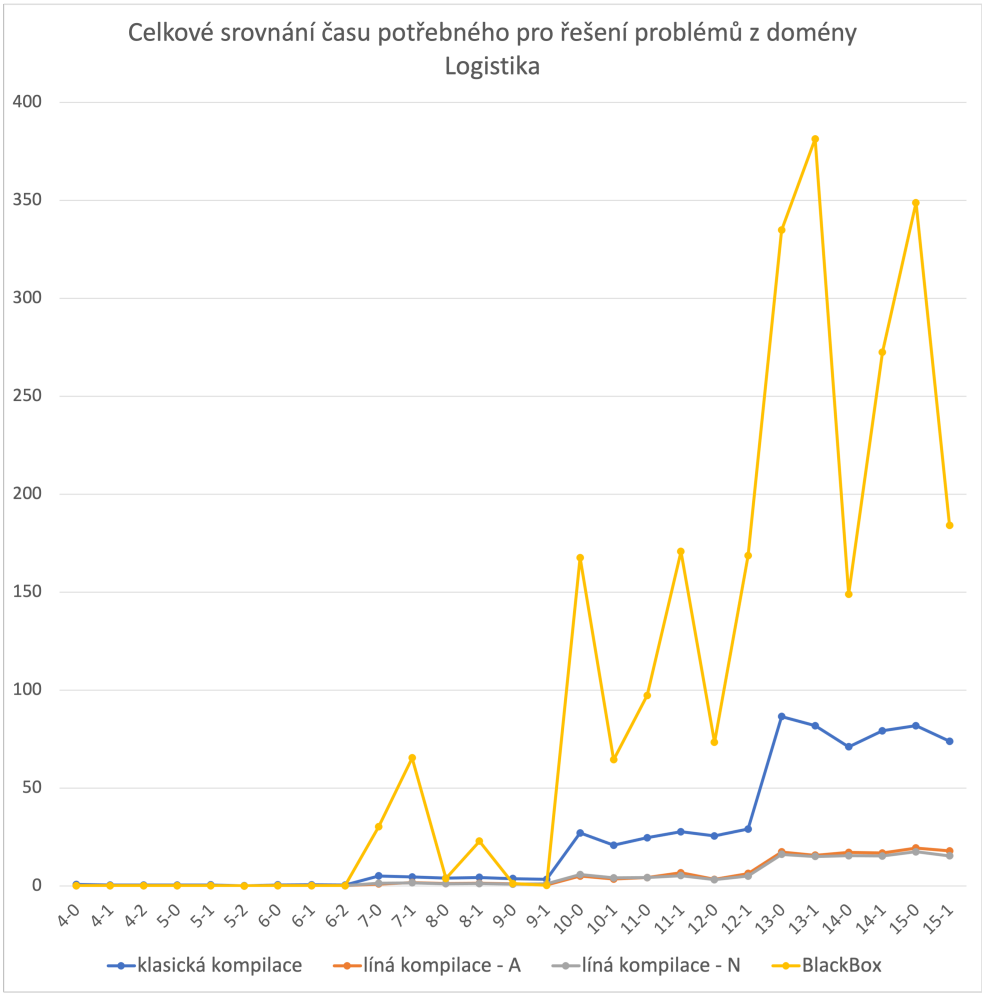
11.2.3 Podrobnější měření u vybraných problémů

Následující sekce se zaměřuje na podrobnější průzkum času potřebného pro vyřešení SAT u problémů probLOGISTICS9-1 a probLOGISTICS13-0. První z problémů byl zvolen proto, že je zde líná kompilace ve variantě N téměř dvakrát pomalejší než ve variantě A. U problému probLOGISTICS13-0 dochází oproti předchozímu problému (12-1) k nárůstu času u klasické kompilace.

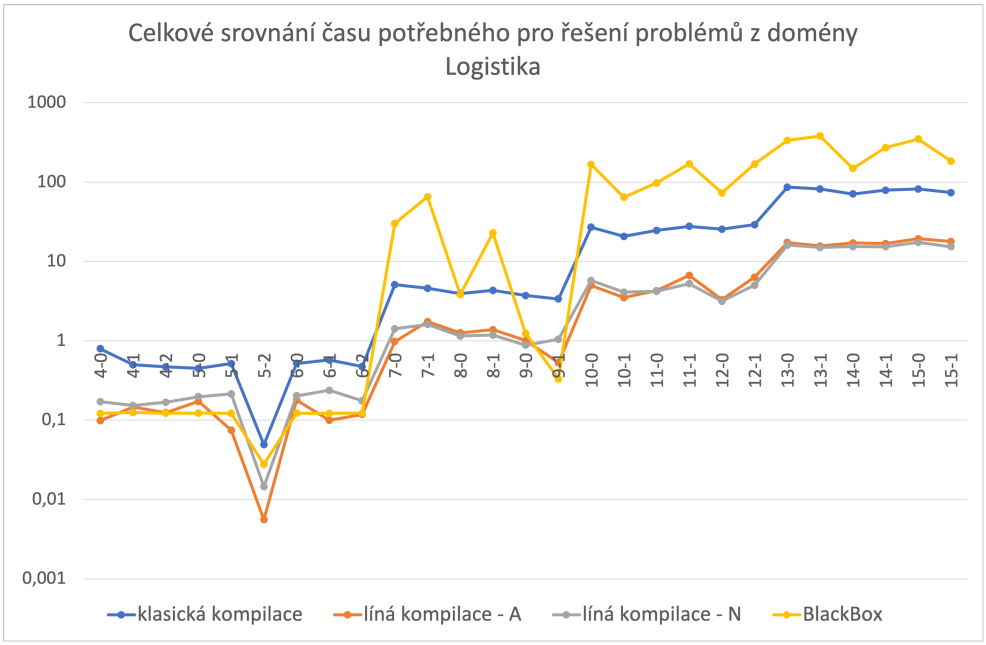
V tabulkách 11.3, 11.4 a grafech 11.5, 11.6 jsou uvedeny časy řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je zde uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.

U problému probLOGISTICS9-1 bylo zjištěno, že líná kompilace ve variantě A potřebuje k vyřešení problému o jednu vrstvu plánovacího grafu méně než varianta N. Tím si můžeme vysvětlit rozdíl v jejich rychlosti řešení problému.

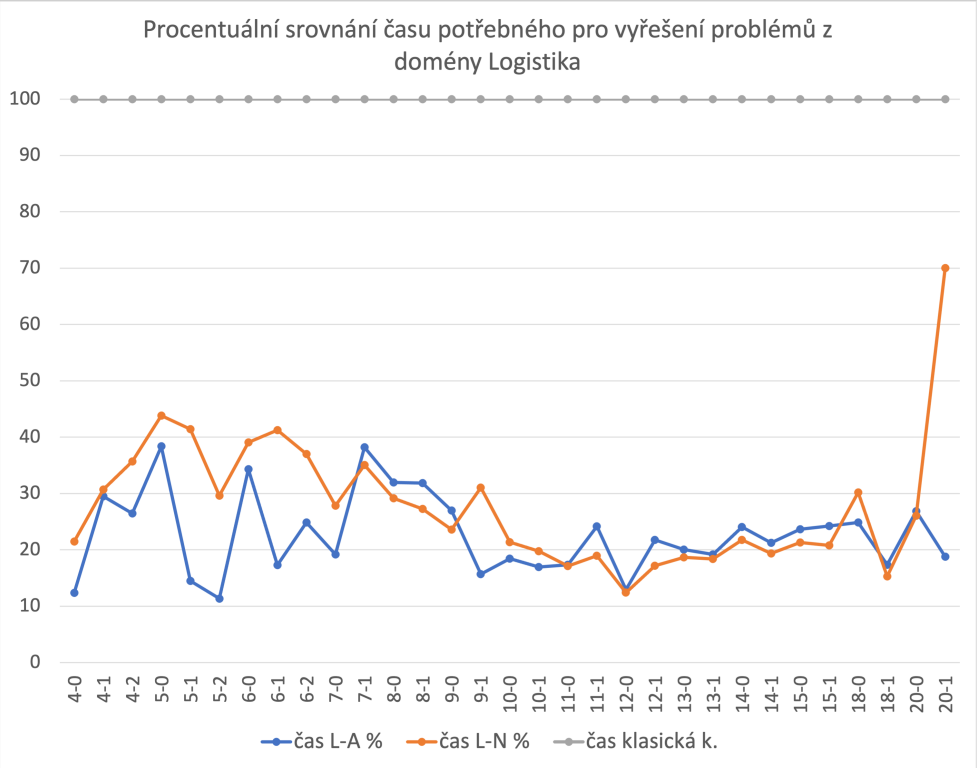
Časy řešení SAT na jednotlivých vrstvách grafu nám ukazují, že výkon plánovače s klasickou kompilací není v tomto případě zpomalen časem řešení SAT. Čistý čas práce SAT řešiče je menší než v případě líné kompilace, přesto je plánovač s klasickou kompilací při řešení problémů celkově pomalejší.



Obrázek 11.2: Celkové srovnání našeho plánovače s různými variantami kompilace a plánovače BlackBox. (Osa y znázorňuje čas v sekundách a osa x jednotlivé problémy z domény Logistika.)



Obrázek 11.3: Celkové srovnání našeho plánovače s různými variantami kompilace a plánovače BlackBox. (Osa y znázorňuje čas v sekundách v logaritmickém měřítku a osa x jednotlivé problémy z domény Logistika.)



Obrázek 11.4: Procentuální srovnání času potřebného pro vyřešení problémů z domény Logistika pro různé druhy kompilace. Čas pro klasickou kompilaci bereme jako 100 %.

LOGISTICS	klasická k.	líná k. - A	líná k. - N	BlackBox
4-0	0,80	0,10	0,17	0,12
4-1	0,50	0,15	0,15	0,13
4-2	0,47	0,12	0,17	0,12
5-0	0,45	0,17	0,20	0,12
5-1	0,52	0,07	0,21	0,12
5-2	0,05	0,01	0,01	0,03
6-0	0,52	0,18	0,20	0,12
6-1	0,58	0,10	0,24	0,12
6-2	0,47	0,12	0,18	0,12
7-0	5,11	0,98	1,42	30,18
7-1	4,58	1,75	1,61	65,36
8-0	3,94	1,26	1,15	3,82
8-1	4,33	1,38	1,18	22,86
9-0	3,73	1,01	0,88	1,24
9-1	3,37	0,53	1,05	0,33
10-0	27,02	4,99	5,76	167,55
10-1	20,75	3,52	4,10	64,43
11-0	24,65	4,28	4,22	97,24
11-1	27,70	6,69	5,26	170,73
12-0	25,50	3,30	3,17	73,30
12-1	29,08	6,33	4,99	168,58
13-0	86,50	17,34	16,13	334,78
13-1	81,74	15,68	15,04	381,34
14-0	71,02	17,08	15,44	148,92
14-1	79,17	16,81	15,33	272,48
15-0	81,79	19,36	17,44	348,74
15-1	73,86	17,87	15,34	184,12
18-0	266,30	66,24	80,41	-
18-1	207,99	36,10	31,85	-
20-0	726,47	195,23	189,31	-
20-1	819,13	153,75	573,64	-

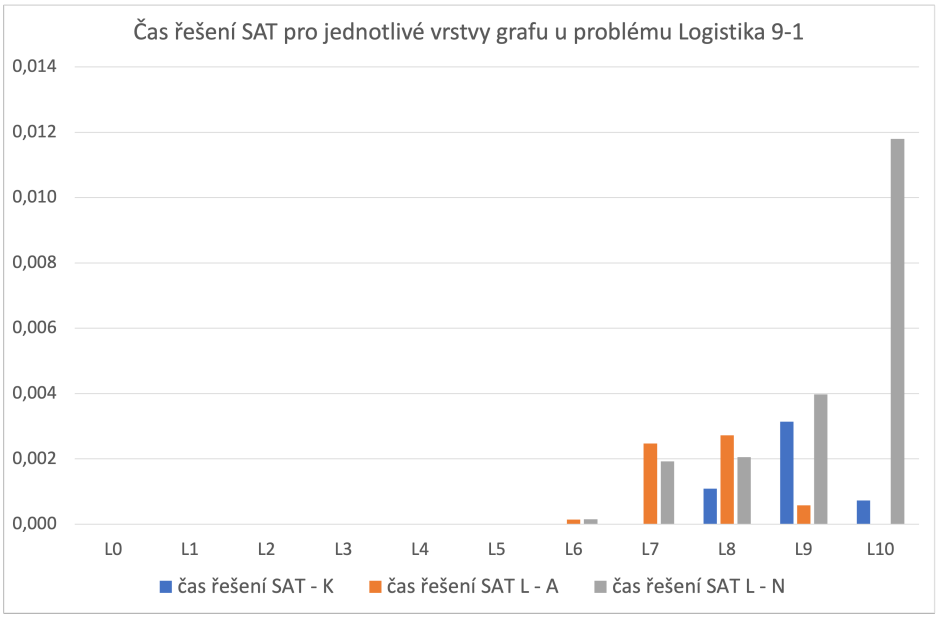
Tabulka 11.2: Čas v sekundách potřebný pro vyřešení problémů z domény Logistika pro různé druhy kompilace. (varianta problému O)

9-1	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	0	0	0	0	0	0	0
L4	0	0	0	0	0	0	0
L5	0	0	0	0	0	0	0
L6	0	1,44E-04	1,49E-04	7,21E-05	7,44E-05	2	2
L7	0	2,47E-03	1,92E-03	1,76E-04	2,14E-04	14	9
L8	1,09E-03	2,72E-03	2,05E-03	3,89E-04	2,57E-04	7	8
L9	3,14E-03	5,82E-04	3,97E-03	2,91E-04	3,97E-04	2	10
L10	7,29E-04	-	1,18E-02	-	5,12E-04	-	23

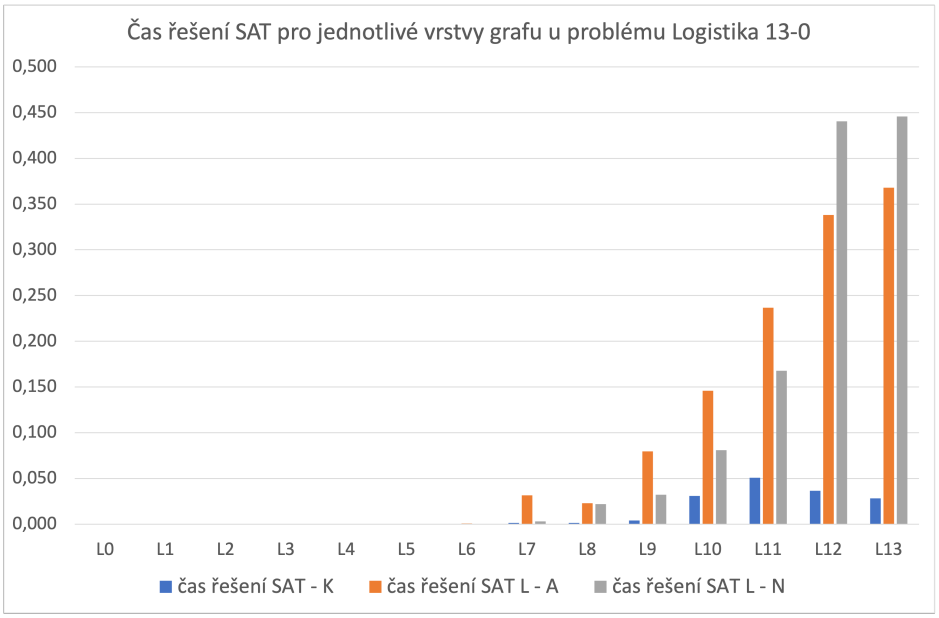
Tabulka 11.3: Problém probLOGISTICS9-1: čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.

13-0	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	0	0	0	0	0	0	0
L4	0	0	0	0	0	0	0
L5	0	0	0	0	0	0	0
L6	0	5,80E-04	4,49E-04	2,90E-04	2,25E-04	2	2
L7	1,42E-03	3,16E-02	3,03E-03	6,44E-04	4,33E-04	49	7
L8	1,50E-03	2,28E-02	2,20E-02	1,52E-03	7,85E-04	15	28
L9	4,00E-03	7,96E-02	3,22E-02	2,57E-03	2,48E-03	31	13
L10	3,09E-02	1,46E-01	8,09E-02	4,86E-03	5,39E-03	30	15
L11	5,07E-02	2,37E-01	1,68E-01	8,77E-03	1,05E-02	27	16
L12	3,67E-02	3,38E-01	4,40E-01	9,94E-03	1,38E-02	34	32
L13	2,82E-02	3,68E-01	4,46E-01	6,57E-03	5,37E-03	56	83

Tabulka 11.4: Problém probLOGISTICS13-0: čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.



Obrázek 11.5: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém probLOGISTICS9-1.



Obrázek 11.6: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém probLOGISTICS13-0.

11.3 Experimenty v doméně Blocks world

Následující grafy a tabulky popisují výsledky experimentů prováděné v doméně Blocks World. Problémy probBLOCKS-4-0 až probBLOCKS-11-2 byly použity v prvním kole soutěže IPC 2000. Všechny použité problémy mají řešení. V experimentech jsme se stejně jako v předchozím případě zaměřovali na parametry výsledné formule Φ a na zjištění potřebného času pro vyřešení jednotlivých problémů. Všechny zkoumané problémy z domény Blocks World se podařilo vyřešit plánovačem s klasickou kompilací v časovém limitu, který byl určen s ohledem na výsledky plánovačů ze soutěže IPC jako 15 minut na jeden problém. V případě využití našeho plánovače s línou kompilací se podařilo v časovém limitu vyřešit jenom problémy do probBLOCKS-7-0, ve většině těžších problémů již docházelo k překročení nastaveného časového limitu.

11.3.1 Výsledná formule Φ

Tabulka 11.6 zobrazuje počet proměnných a klauzulí ve výsledné formuli Φ pro různé druhy kompilace. Formule pro řešené problémy obsahovaly od 421 do 15509 proměnných. Plánovač s línou kompilací vyřešil v časovém limitu pouze problémy, které měly maximálně cca 4000 proměnných.

V případě klasické kompilace obsahovala formule Φ od cca 6100 až do 1100000 klauzulí. Ve všech případech to bylo znatelně více klauzulí než při využití líné kompilace (varianta A i N). Experimenty ukázaly, že v případě líné kompilace stačilo pro vyřešení formule ve většině případů přibližně 10 % klauzulí. Počet klauzulí při líné kompilaci je ve variantách A a N podobný. Formule Φ při variantě A obsahuje zpravidla o něco méně klauzulí než při variantě N. Podrobné výsledky – procentuální srovnání pro problémy probBLOCKS-4-0 až probBLOCKS-7-0 – jsou uvedeny v tabulce 11.5.

probBLOCKS	**klauzule L-A**	**klauzule L-N**	**klauzule K**
4-0	12,00%	13,31%	100%
4-1	10,58%	10,36%	100%
4-2	11,22%	11,64%	100%
5-0	9,78%	10,93%	100%
5-1	10,92%	11,13%	100%
5-2	10,87%	11,99%	100%
6-0	7,54%	8,65%	100%
6-1	10,47%	11,37%	100%
6-2	15,63%	16,41%	100%
7-0	8,03%	9,09%	100%

Tabulka 11.5: Procentuální srovnání počtu klauzulí výsledné formule Φ pro problémy z domény Blocks World pro různé druhy kompilace. Počet klauzulí u klasické kompilace je 100 %.

BLOCKS	proměnné	klauzule K	klauzule L - A	klauzule L - N
4-0	421	6151	738	819
4-1	625	9470	1002	981
4-2	403	5938	666	691
5-0	1185	25886	2531	2829
5-1	1023	22344	2441	2487
5-2	1573	34711	3774	4162
6-0	1689	52665	3972	4556
6-1	1495	44780	4687	5090
6-2	2751	58333	9116	9575
7-0	3499	125871	10113	11440
7-1	4315	134840	-	-
7-2	3863	129762	16635	-
8-0	4649	230081	-	-
8-1	5171	252046	-	-
8-2	4037	195254	41247	-
9-0	9057	502886	-	-
9-1	8001	352549	-	-
9-2	7609	379435	-	-
10-0	12529	766757	-	-
10-1	12325	738462	-	-
10-2	12529	731466	-	-
11-0	14885	1006791	-	-
11-1	14704	1162693	-	-
11-2	15509	1106900	-	-

Tabulka 11.6: Počty proměnných a klauzulí výsledné formule Φ pro různé druhy kompilace pro problémy z domény Blocks World. (varianta problému O)

11.3.2 Čas potřebný pro vyřešení problémů

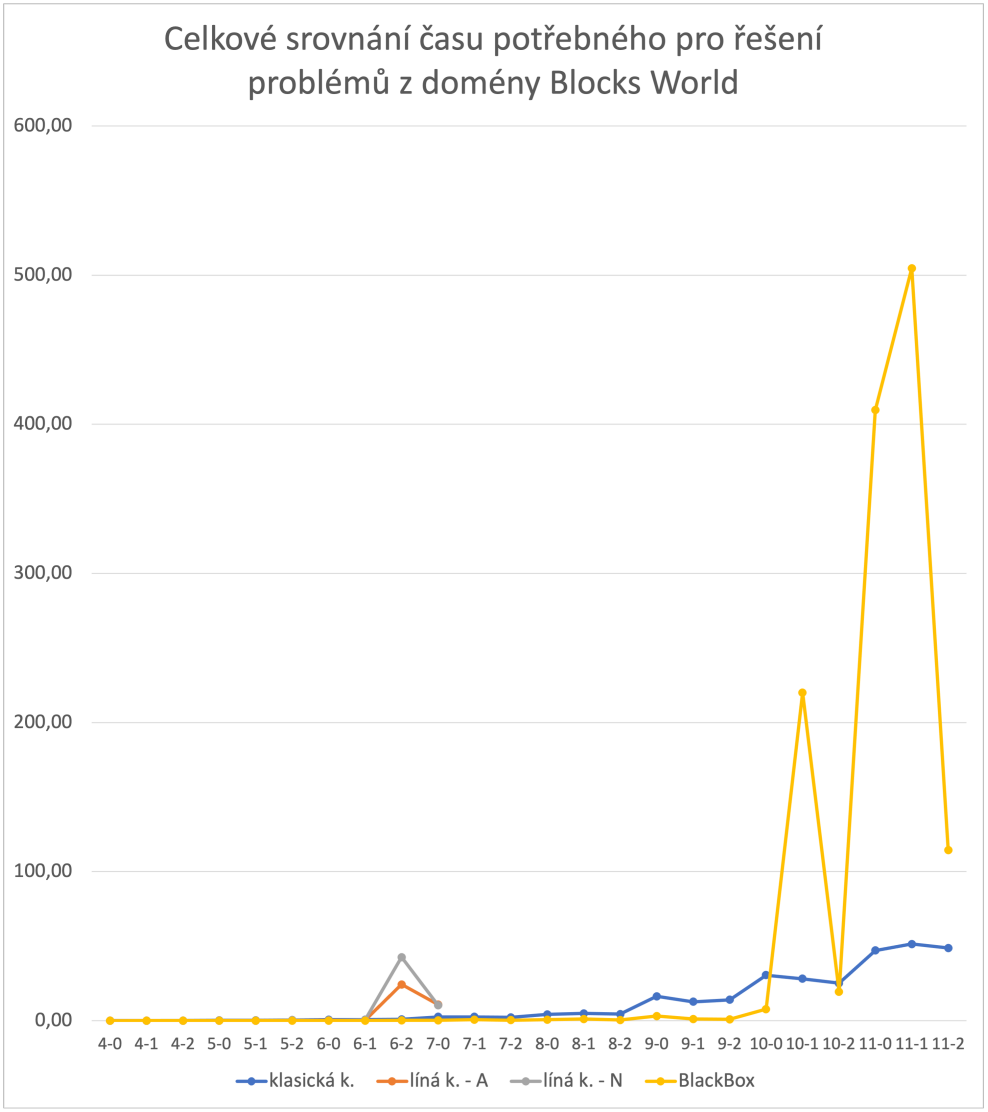
Na grafech 11.7 (normální měřítko) a 11.8 (logaritmické měřítko) a v tabulce 11.7 je vidět celkové srovnání časů potřebných pro vyřešení jednotlivých problémů z domény Blocks World pro náš plánovač s klasickou a línou kompilací (varianty A a N) a plánovače BlackBox.

V doméně Blocks World je úspěšnost líné kompilace značně odlišná od výsledků z domény Logistika. V tomto případě byla líná kompilace rychlejší než kompilace klasická pouze na lehčích problémech (do problému probBLOCKS-6-1), poté nastává prudký zlom.

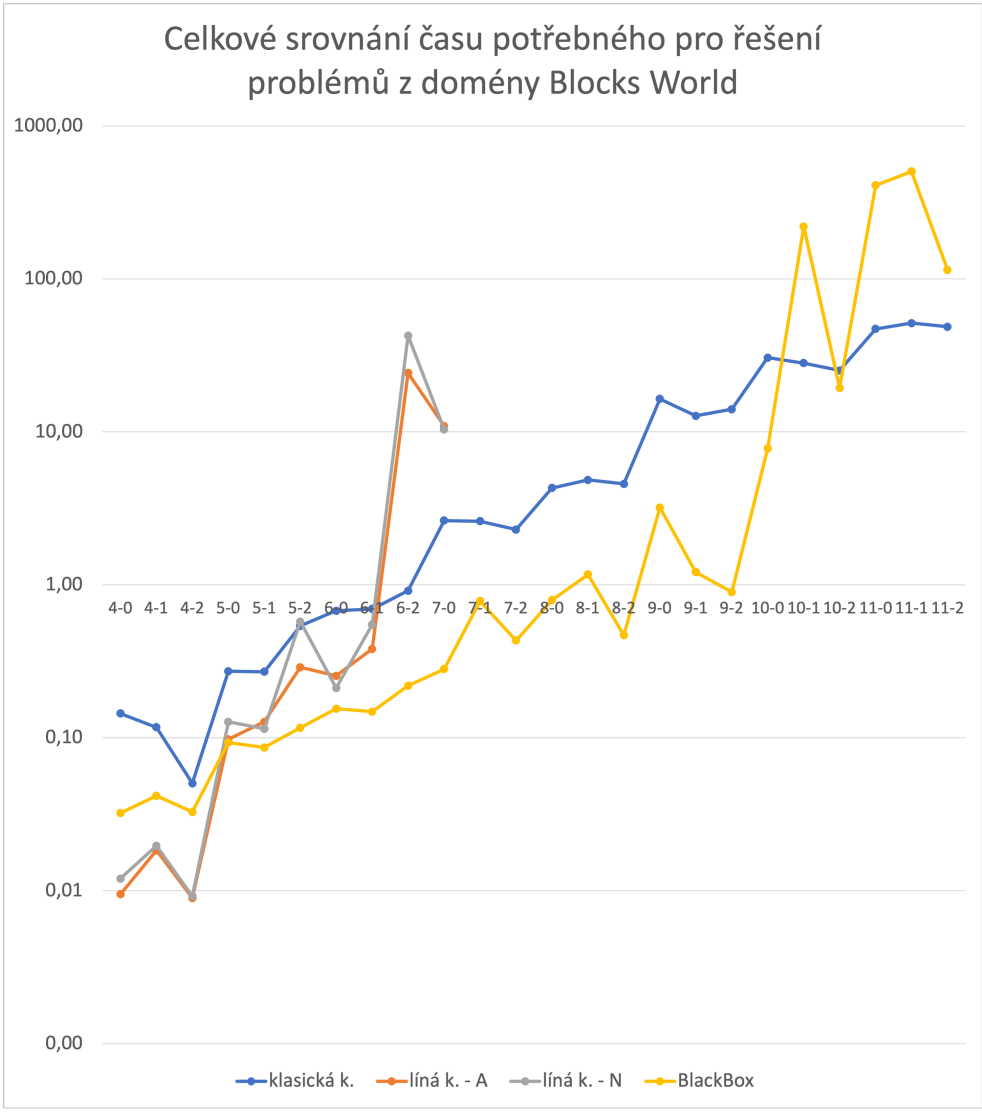
V problémech probBLOCKS-6-2, probBLOCKS-7-2 a probBLOCKS-8-2 potřebuje plánovač s línou kompilací mnohonásobně více času než plánovač s klasickou kompilací. Ve zbylých problémech čas líné kompilace ještě více narůstá a překračuje nastavený limit 15 minut. Skutečný čas řešení těchto problémů nebyl proto změřen. Výkon plánovače s línou kompilací varianta A a varianta N je srovnatelný.

Pokud porovnáme výkon našeho plánovače s hodnotami naměřenými pro plánovač BlackBox, v případě problémů 4-0 až 4-2 byl BlackBox lepší než náš plánovač s klasickou kompilací, ale zároveň pomalejší než náš plánovač s línou kompilací. Na středních problémech byl BlackBox lepší než náš plánovač s línou i klasickou kompilací. Na problémech od probBLOCKS-10-1 byl rychlejší náš plánovač s klasickou kompilací. (Jak již bylo řečeno, plánovač s línou kompilací překročil ve středně těžkých a těžkých problémech časový limit, proto nebyly jeho časy doměřeny.)

Do druhého kola soutěže IPC 2000 v doméně Blocks World postoupily pouze 3 plánovače ze 14, BlackBox nebyl mezi nimi. Výkon plánovače s klasickou kompilací nebyl již zkoumán na těžších problémech než probBLOCKS-11-2 z důvodu chybějícího srovnání s línou kompilací, která se ukázala jako neúspěšná již na lehčích problémech.



Obrázek 11.7: Celkové srovnání našeho plánovače s různými variantami kompilace a plánovače BlackBox. (Osa y znázorňuje čas v sekundách a osa x jednotlivé problémy z domény Blocks World.)



Obrázek 11.8: Celkové srovnání našeho plánovače s různými variantami kompilace a plánovače BlackBox. (Osa y znázorňuje čas v sekundách v logaritmickém měřítku a osa x jednotlivé problémy z domény Blocks World.)

BLOCKS	klasická k.	líná k. - A	líná k. - N	BlackBox
4-0	0,14	0,01	0,01	0,03
4-1	0,12	0,02	0,02	0,04
4-2	0,05	0,01	0,01	0,03
5-0	0,27	0,10	0,13	0,09
5-1	0,27	0,13	0,11	0,09
5-2	0,54	0,29	0,58	0,12
6-0	0,68	0,25	0,21	0,15
6-1	0,70	0,38	0,55	0,15
6-2	0,92	24,32	42,61	0,22
7-0	2,63	10,91	10,38	0,28
7-1	2,61	-	-	0,78
7-2	2,30	273	-	0,43
8-0	4,30	-	-	0,80
8-1	4,86	-	-	1,17
8-2	4,56	132,93	81	0,47
9-0	16,42	-	-	3,20
9-1	12,74	-	-	1,21
9-2	14,05	-	-	0,90
10-0	30,61	-	-	7,78
10-1	28,18	-	-	220,03
10-2	25,19	-	-	19,36
11-0	47,11	-	-	409,47
11-1	51,47	-	-	504,62
11-2	48,71	-	-	114,42

Tabulka 11.7: Čas v sekundách potřebný pro vyřešení problémů z domény Blocks World pro různé druhy kompilace. (varianta problému O)

11.3.3 Podrobnější měření u vybraných problémů

Následující sekce se zaměřuje na čas řešení SAT v jednotlivých vrstvách plánovacího grafu pro vybrané problémy. Pro toto zkoumání jsme zvolili problémy probBLOCKS6-1 a probBLOCKS6-2, protože právě mezi těmito problémy dochází k nárůstu času potřebného pro vyřešení problému u plánovače s línou kompilací. Pro doplnění je uvedeno ještě podrobnější měření u problému probBLOCKS8-2.

Při podrobnějším zkoumání těchto problémů bylo zjištěno, že špatný výkon plánovače s línou kompilací na středních a těžších problémech z domény Blocks World je způsobem prudce narůstajícím časem potřebným pro vyřešení výsledných formulí Φ. Například na poslední vrstvě plánovacího grafu problému BLOCKS8-2 potřeboval SAT řešič pouze 0,001 sekundy na vyřešení formule Φ v případě klasické kompilace, pro línou kompilaci trvalo řešení formulí Φ celkem 53 sekund (varianta N) a 120 sekund (varianta A). U těžších problémů čas řešení SAT při líné kompilaci ještě výrazněji narůstá, což způsobuje neefektivitu plánovače s touto variantou kompilace.

Jen pro doplnění celkového obrazu uveďme, že při řešení plánovacích problémů probBLOCKS6-2 a probBLOCKS8-2 ve variantě problému NI s n zvoleným jako délka optimálního vrstveného plánu (známá z předchozích řešení problému ve variantě O) byl nárůst obtížnosti formulí Φ ještě významnější. V celkovém součtu trvalo řešení problému ve variantě NI (s jednorázovou expanzí grafu) se známou délkou plánu n déle než řešení toho samého problému ve variantě O (s postupnou expanzí grafu). V experimentální části této práce se však primárně zabýváme variantou problému O, proto zde konkrétní výsledky neuvádíme.

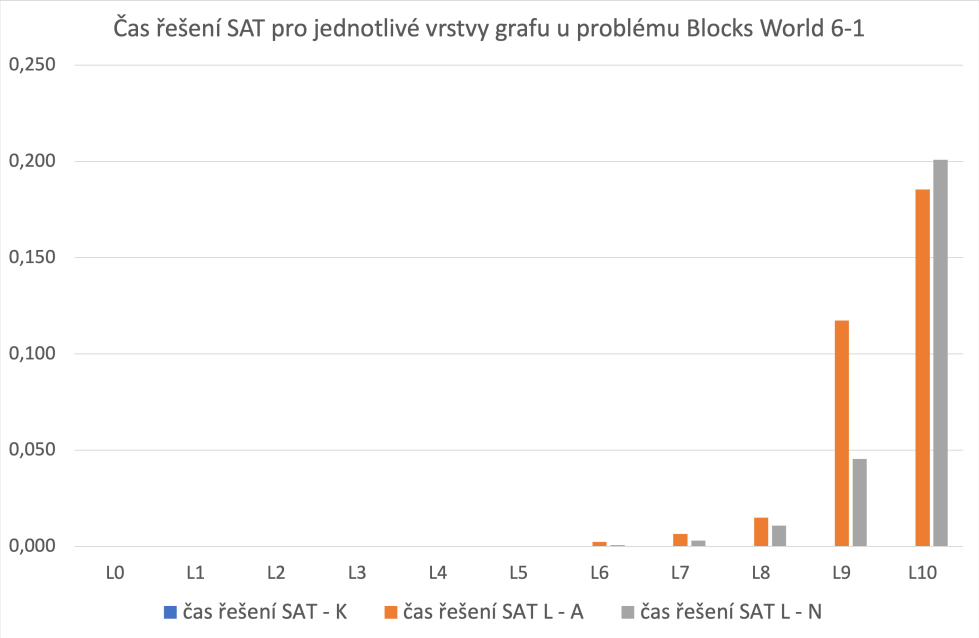
Můžeme se domnívat, že rozdíl v obtížnosti formulí mezi klasickou a línou kompilací je způsobený tím, že klauzule pro mutexy v tomto případě výrazně podporují jednotkovou propagaci. Při postupném doplňování některých klauzulí pro závislé akce nelze jednotkové propagace využít v takové míře a proto při líné kompilaci vznikají formule těžší.

Tento rozdíl se ze všech zkoumaných domén nejvýrazněji projevil právě v doméně Blocks World, která je specifická tím, že všechny akce jsou navzájem závislé. Plánovací graf pro problém z této domény tedy obsahuje na každé vrstvě akční mutexy pro všechny možné dvojice akcí.

Výsledky měření jsou uvedeny v tabulkách 11.8, 11.9, 11.10 a grafech 11.9, 11.10.

6-1	čas K	čas L - A	čas L - N	čas iterace L - A	čas iterace L - N	počet opak. L - A	počet opak. L - N
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	0	2,99E-05	2,93E-05	1,49E-05	1,46E-05	2	2
L4	0	8,19E-05	6,80E-05	2,73E-05	2,27E-05	3	3
L5	1,10E-04	1,68E-04	1,01E-04	5,61E-05	3,37E-05	3	3
L6	2,73E-05	2,29E-03	7,45E-04	2,29E-04	1,24E-04	10	6
L7	3,22E-05	6,47E-03	3,09E-03	2,31E-04	2,81E-04	28	11
L8	1,26E-04	1,50E-02	1,08E-02	3,34E-04	4,34E-04	45	25
L9	1,65E-04	1,17E-01	4,54E-02	1,35E-03	1,03E-03	87	44
L10	2,09E-04	1,85E-01	2,01E-01	2,77E-03	4,67E-03	67	43

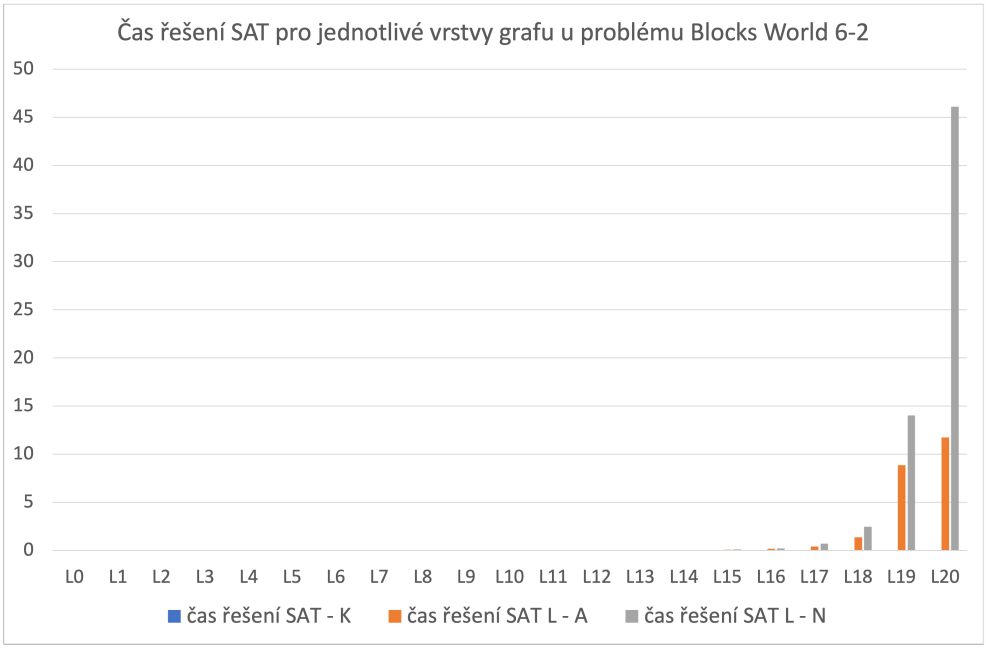
Tabulka 11.8: Problém probBLOCKS6-1: čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.



Obrázek 11.9: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém probBLOCKS6-1.

6-2	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	0	0	0	0	0	0	0
L4	0	0	0	0	0	0	0
L5	0	0	0	0	0	0	0
L6	0	0	0	0	0	0	0
L7	0	5,81E-05	1,19E-04	2,91E-05	5,97E-05	2	2
L8	0	2,59E-04	4,93E-04	6,47E-05	1,64E-04	4	3
L9	0	3,73E-04	2,06E-04	5,34E-05	5,16E-05	7	4
L10	1,90E-04	1,40E-03	5,93E-04	1,55E-04	9,89E-05	9	6
L11	6,40E-05	9,05E-04	7,37E-04	1,51E-04	1,05E-04	6	7
L12	3,87E-05	6,04E-03	2,66E-03	3,18E-04	2,66E-04	19	10
L13	8,03E-05	3,29E-02	8,04E-03	1,10E-03	4,73E-04	30	17
L14	9,30E-05	2,46E-02	2,63E-02	6,64E-04	1,25E-03	37	21
L15	3,43E-04	6,88E-02	1,15E-01	1,81E-03	3,97E-03	38	29
L16	3,93E-04	1,76E-01	2,18E-01	1,84E-03	6,62E-03	96	33
L17	9,70E-04	4,06E-01	7,07E-01	8,29E-03	1,57E-02	49	45
L18	7,28E-04	1,38E+00	2,45E+00	1,35E-02	5,21E-02	102	47
L19	5,86E-04	8,85E+00	1,40E+01	7,50E-02	2,65E-01	118	53
L20	7,36E-03	1,17E+01	4,61E+01	9,53E-02	6,32E-01	123	73

Tabulka 11.9: Problém probBLOCKS6-2: čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.



Obrázek 11.10: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém probBLOCKS6-2.

8-2	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	0	0	0	0	0	0	0
L4	0	0	0	0	0	0	0
L5	0	7,07E-05	8,53E-05	3,53E-05	4,26E-05	2	2
L6	0	3,29E-04	1,25E-04	5,49E-05	4,16E-05	6	3
L7	1,10E-02	1,51E-03	3,95E-04	1,16E-04	9,89E-05	13	4
L8	9,85E-05	4,48E-03	1,91E-03	2,04E-04	2,39E-04	22	8
L9	2,05E-04	2,59E-02	7,30E-03	4,31E-04	6,08E-04	60	12
L10	8,58E-05	4,82E-02	1,95E-02	7,65E-04	7,78E-04	63	25
L11	2,15E-04	7,32E-02	6,08E-02	8,93E-04	1,79E-03	82	34
L12	5,73E-04	3,32E-01	1,74E-01	1,93E-03	3,63E-03	172	48
L13	6,54E-04	1,19E+00	7,41E-01	6,42E-03	1,16E-02	186	64
L14	3,26E-04	2,15E+00	4,32E+00	1,23E-02	6,35E-02	174	68
L15	8,17E-04	1,88E+01	2,34E+01	4,96E-02	2,93E-01	379	80
L16	1,24E-03	1,21E+02	5,29E+01	2,68E-01	6,38E-01	452	83

Tabulka 11.10: Problém probBLOCKS8-2: čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.

11.4 Experimenty v doméně Mystery

Následující grafy a tabulky popisují výsledky experimentů prováděné v doméně Mystery. V prvním ročníku soutěže IPC, kde byla tato doména využita, byly problémy generovány strojově. Problémy měly být číslovány podle jejich předpokládané obtížnosti, ale vyšlo najevo, že u takto generovaných problémů je těžké obtížnost předem odhadnout. U některých z nich se v soutěži IPC stalo, že je nevyřešil v časovém limitu ani jeden se zúčastněných plánovačů. V experimentech proto využíváme z této domény pouze problémy p25, p01, p28, p11, p29, p03, p27, p26, p09, p30 a p02, které jsme seřadili podle počtu klauzulí ve výsledné formuli Φ při klasické kompilaci.

11.4.1 Výsledná formule Φ

V tabulce 11.11 jsou uvedeny počty proměnných a klauzulí výsledné formule Φ. Tato doména se oproti ostatním doménám vyznačuje velkým počtem klauzulí ve formuli Φ při klasické kompilaci. Největší řešený problém z této domény měl 11000 proměnných a 4,7 milionu klauzulí. (Oproti tomu největší řešený problém z domény Logistika měl cca 30000 proměnných a 2 miliony klauzulí, v doméně Blocks World cca 15000 proměnných a milion klauzulí.) Průměrný počet klauzulí při líné kompilaci byl ve variantě A 3,96 % a ve variantě N 5,19 % počtu klauzulí klasické kompilace. Toto číslo bylo naopak nejnižší ze všech zkoumaných domén. Procentuální srovnání počtu klauzulí je znázorněno v grafu 11.11.

MYST	proměnné	klauzule K	klauzule L - A	klauzule L - N
25	788	12631	1018	2129
01	1059	24596	2524	2529
28	2243	114595	7964	7900
11	2967	256287	11000	10948
29	2361	282067	3566	9949
03	2736	289876	3826	11237
27	2557	292578	11480	11480
26	6109	1136958	30665	30235
09	6152	1918960	34577	34553
30	11564	4142742	64010	64246
02	11281	4795637	69713	69571

Tabulka 11.11: Počty proměnných a klauzulí výsledné formule Φ pro různé druhy kompilace pro problémy z domény Mystery. (varianta problému O)

Obrázek 11.11: Procentuální srovnání počtu klauzulí výsledné formule Φ pro problémy z domény Mystery pro různé druhy kompilace.

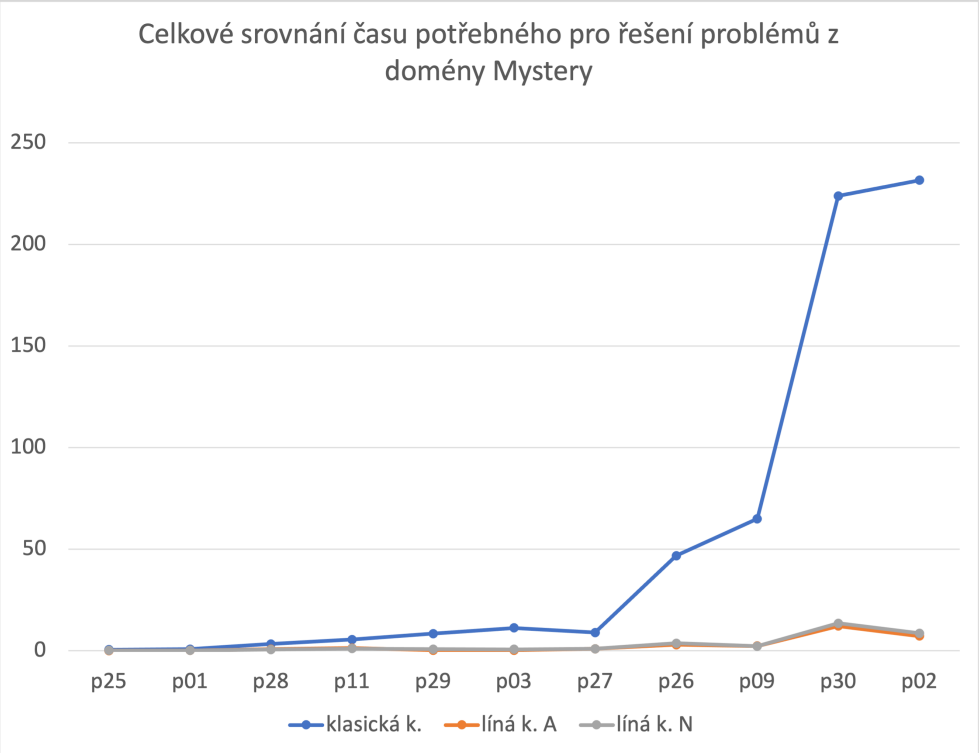
11.4.2 Čas potřebný pro vyřešení problémů

Na grafech 11.12 (normální měřítko) a 11.13 (logaritmické měřítko) a v tabulce 11.12 je vidět celkové srovnání časů řešení problémů z domény Mystery pro náš plánovač s klasickou a línou kompilací. Ve všech problémech byla líná kompilace (varianta A i N) rychlejší než kompilace klasická. Na vyřešení problémů stačilo v případě varianty A průměrně pouze 8,94 % času klasické kompilace, pro variantu N bylo toto číslo 11,12 %.

Pokud se zaměříme na detailnější srovnání dvou variant líné kompilace, všimneme si, že tyto dvě varianty jsou, co se týče potřebného času pro vyřešení problémů, srovnatelné. U problémů p29 a p03 byla rychlejší varianta A (0,20 a 0,17 oproti 0,74 a 0,65 sekundy).

11.4.3 Podrobnější analýza vybraných problémů

Následující sekce se zaměřuje na čas řešení SAT v jednotlivých vrstvách plánovacího grafu pro vybrané problémy z domény Mystery. Pro toto zkoumání jsme zvolili problém p26, protože od tohoto problému začíná narůstat čas plánovače s klasickou kompilací. Další zkoumaný je problém p02 s největším počtem klauzulí při klasické kompilaci.

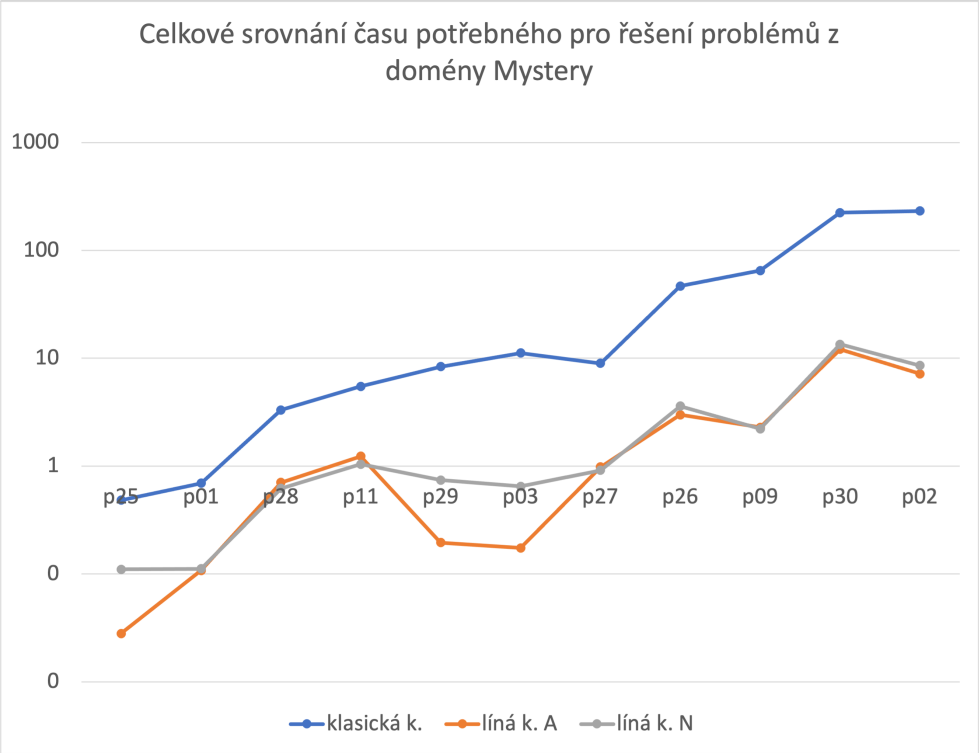


Obrázek 11.12: Celkové srovnání času potřebného pro vyřešení problémů z domény Mystery pro různé druhy kompilace. (Osa y znázorňuje čas v sekundách a osa x jednotlivé problémy z domény Mystery.)

Podrobnější měření ukázalo, že vzrůstající čas řešení problémů plánovačem s klasickou kompilací není způsoben vznikem těžkých formulí Φ. Čas řešení SAT na jednotlivých vrstvách je srovnatelný s línou kompilací. Časově náročná je v tomto případě konstrukce mutexů v plánovacím grafu, kterých je velké množství. (Například u problému p26 stačí k získání řešení pouze šest akčních vrstev grafu, ale výsledná formule Φ pro klasickou kompilaci obsahuje přes milion klauzulí. Při líné kompilaci má formule pouze kolem 30000 klauzulí, což znamená, že formule Φ při klasické kompilaci obsahuje více než milion klauzulí pro mutexy navíc, které nejsou k vyřešení problému potřeba.)

Pokud se podíváme na poslední vrstvu grafu, stojí za zmínku, že v obou případech je čas řešení SAT pro línou kompilaci varianta A znatelně menší než pro variantu N. Ve variantě N dochází na poslední vrstvě k většímu počtu opakování řešení formule s novými klauzulemi (8 a 7 oproti 52 a 101).

Výsledky měření jsou uvedeny v tabulce 11.13 a grafech 11.14, 11.14.



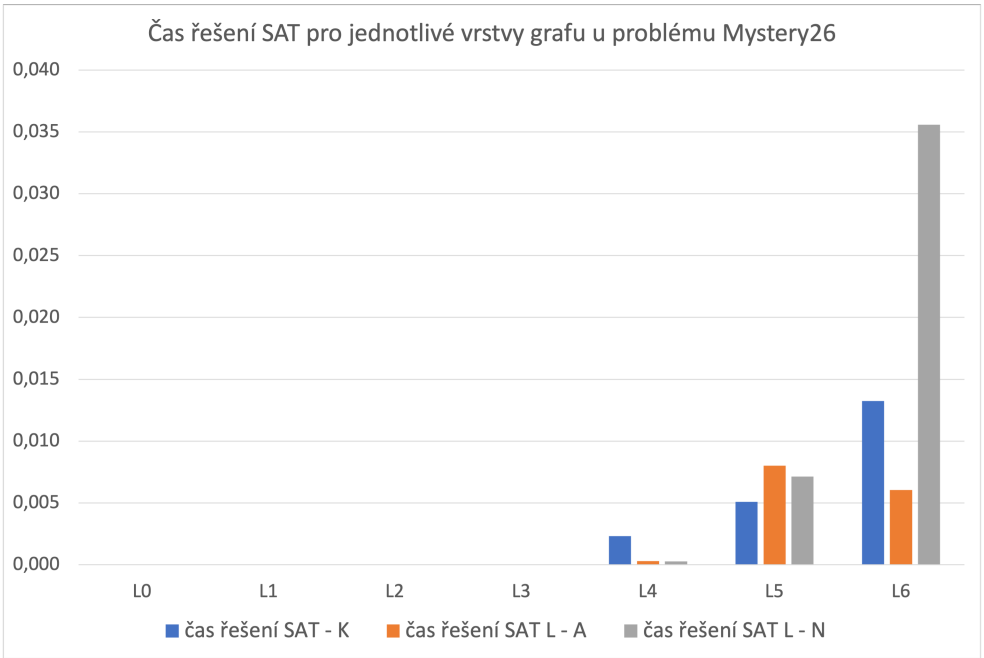
Obrázek 11.13: Celkové srovnání času potřebného pro vyřešení problémů z domény Mystery pro různé druhy kompilace. (Osa y znázorňuje čas v sekundách v logaritmickém měřítku a osa x jednotlivé problémy z domény Mystery.)

MYST	klasická k.	líná k. A	líná k. N
p25	0,48	0,03	0,11
p01	0,69	0,11	0,11
p28	3,30	0,70	0,62
p11	5,49	1,23	1,04
p29	8,34	0,20	0,74
p03	11,19	0,17	0,65
p27	8,96	0,98	0,91
p26	46,70	2,98	3,59
p09	64,94	2,28	2,22
p30	223,77	12,08	13,47
p02	231,56	7,17	8,54

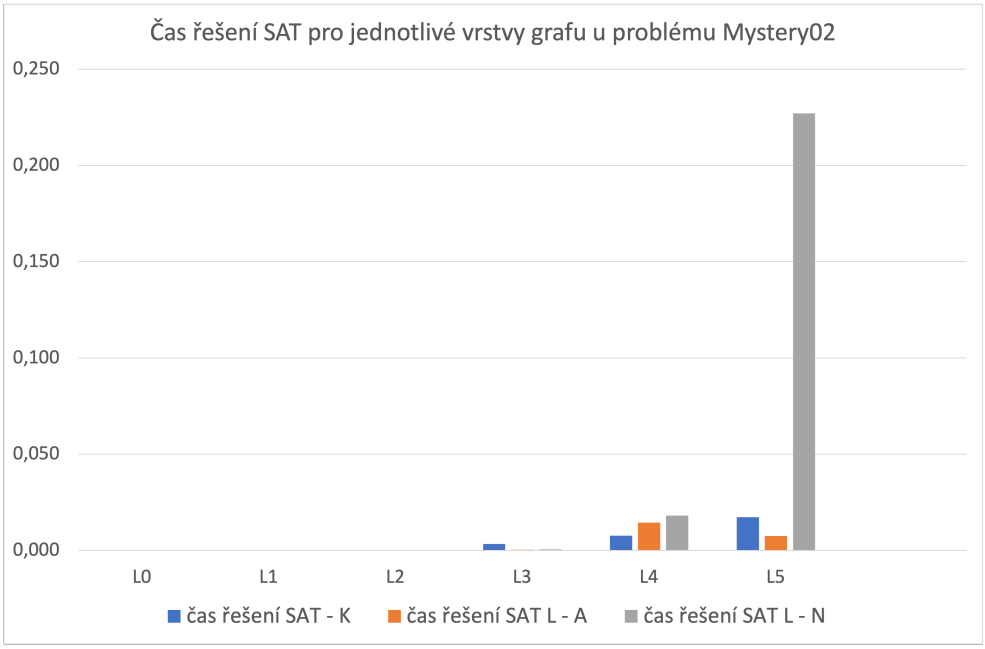
Tabulka 11.12: Čas v sekundách potřebný pro vyřešení problémů z domény Mystery pro různé druhy kompilace. (varianta problému O)

M26	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	0	0	0	0	0	0	0
L4	2,32E-03	3,08E-04	2,83E-04	1,54E-04	1,42E-04	2	2
L5	5,09E-03	8,02E-03	7,12E-03	3,34E-04	2,97E-04	24	24
L6	1,32E-02	6,05E-03	3,56E-02	7,56E-04	6,84E-04	8	52
M02	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	3,31E-03	4,00E-04	5,20E-04	1,33E-04	1,73E-04	3	3
L4	7,73E-03	1,44E-02	1,80E-02	6,54E-04	7,51E-04	22	24
L5	1,73E-02	7,57E-03	2,27E-01	1,08E-03	2,25E-03	7	101

Tabulka 11.13: Čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.



Obrázek 11.14: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém Mystery26.



Obrázek 11.15: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém Mystery02.

ZENO	proměnné	klauzule K	klauzule L - A	klauzule L - N
1	78	54	28	28
2	697	19492	1883	2895
3	1348	52700	2948	6857
4	1181	40634	5256	5552
5	1507	91883	8502	8780
6	2042	147761	12876	13025
7	2266	138415	12326	11880
8	3514	385582	22648	24765
9	4719	548189	24691	36439
10	5484	684230	41317	45411
11	7403	1389961	73084	78423
12	7293	1302345	63258	65882
13	9416	1790809	72397	119535

Tabulka 11.14: Počty proměnných a klauzulí výsledné formule Φ pro různé druhy kompilace pro problémy z domény ZenoTravel. (varianta problému O)

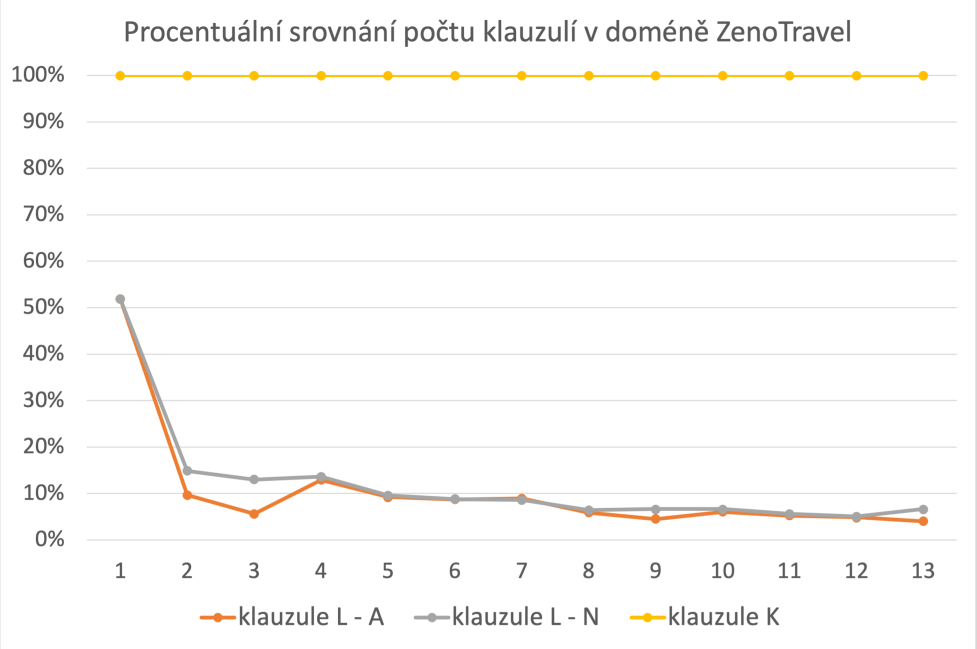
11.5 Experimenty v doméně ZenoTravel

Následující grafy a tabulky popisují výsledky experimentů prováděné v doméně ZenoTravel. Experimenty byly prováděny na problémech 1 až 13. Všechny tyto problémy byly využity v soutěži IPC v roce 2002 a jsou seřazené podle (předpokládané) obtížnosti. Všechny použité problémy mají řešení.

11.5.1 Výsledná formule Φ

Tabulka 11.14 zobrazuje počet proměnných a klauzulí ve výsledné formuli Φ pro různé druhy kompilace. Problém číslo 1 je oproti ostatním problémům velmi malý, výsledná formule Φ měla pouze 78 proměnných a 54 (klasická kompilace) či 28 klauzulí (líná kompilace). Formule pro ostatní řešené problémy obsahovaly od 679 do 9416 proměnných. (U varianty A líné kompilace pro problémy 2, 3, 9 a 13 obsahovala formule méně proměnných než ostatní druhy kompilace, protože se podařilo řešení nalézt na dřívější vrstvě grafu, v tabulce jsou uvedeny počty proměnných pouze pro ostatní varianty.)

Pro problémy 2 až 13 obsahovala formule Φ pro klasickou kompilaci od cca 19000 až do necelých 2 milionů klauzulí. Ve všech případech to bylo znatelně více klauzulí než při využití líné kompilace (varianta A i N). Průměrný počet klauzulí při líné kompilaci byl ve variantě A 7,14 % a ve variantě N 8,80 % počtu klauzulí klasické kompilace. Procentuální srovnání je znázorněno na grafu 11.16.

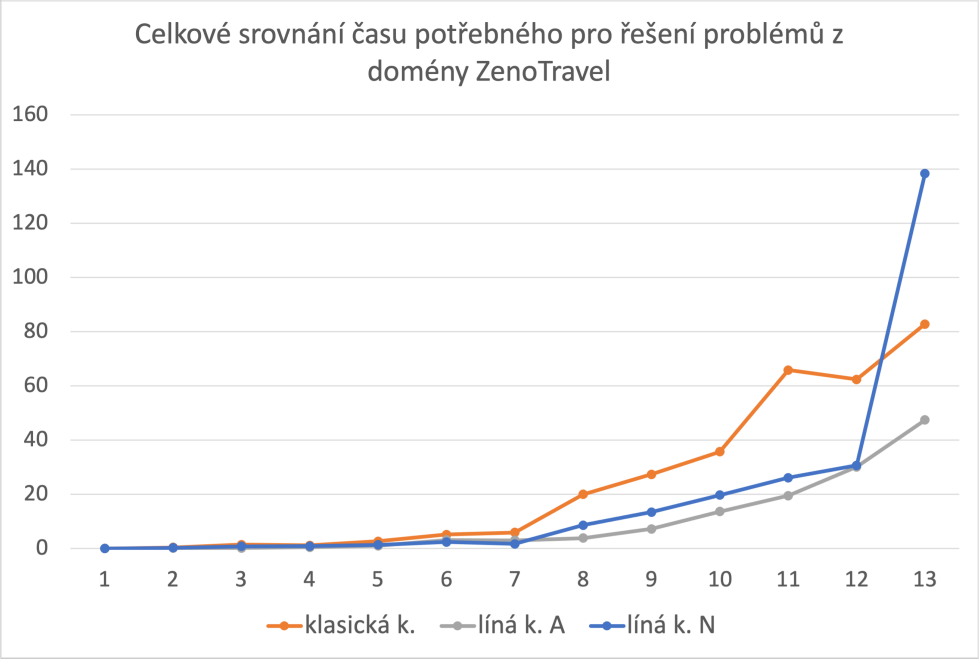


Obrázek 11.16: Procentuální srovnání počtu klauzulí výsledné formule Φ pro problémy z domény ZenoTravel pro různé druhy kompilace.

11.5.2 Čas potřebný pro vyřešení problémů

Na obrázku 11.17 a v tabulce 11.15 je vidět celkové srovnání časů řešení problémů z domény ZenoTravel pro náš plánovač s klasickou a línou kompilací. Líná kompilace byla rychlejší než kompilace klasická ve všech problémech kromě problému 13 ve variantě N. Na vyřešení problémů stačilo v případě varianty A průměrně 38 % času klasické kompilace, pro variantu N bylo toto číslo 58 %. Ušetření času použitím líné kompilace bylo v této doméně menší než v doméně Logistika a Mystery.

Pokud se zaměříme na detailnější srovnání dvou variant líné kompilace, varianta A dosahovala v této doméně o něco lepších výsledků než varianta N. Rozdíl byl výrazný hlavně u problému číslo 13. U některých problémů (2, 3, 9 a 13) se podařilo při využití varianty A nalézt řešení s menším počtem vrstev plánovacího grafu.



Obrázek 11.17: Celkové srovnání času potřebného pro vyřešení problémů z domény ZenoTravel pro různé druhy kompilace. (Osa y znázorňuje čas v sekundách a osa x jednotlivé problémy z domény ZenoTravel.)

ZenoTravel	**klasická k.**	**líná k. - A**	**líná k. - N**
p1	0,02	0,01	0,01
p2	0,39	0,13	0,25
p3	1,45	0,21	0,83
p4	1,13	0,51	0,90
p5	2,68	1,03	1,35
p6	5,17	3,18	2,38
p7	5,90	3,00	1,72
p8	20,00	3,86	8,59
p9	27,39	7,21	13,46
p10	35,68	13,65	19,71
p11	65,86	19,51	26,16
p12	62,38	30,08	30,70
p13	82,78	47,42	138,33

Tabulka 11.15: Čas v sekundách potřebný pro vyřešení problémů z domény ZenoTravel pro různé druhy kompilace. (varianta problému O)

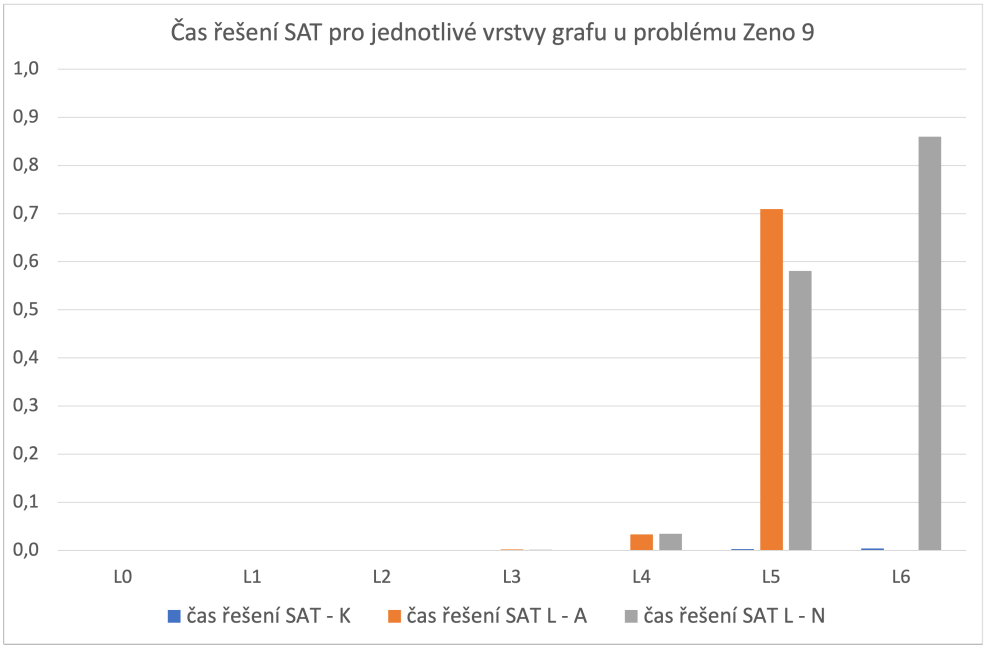
11.5.3 Podrobnější analýza vybraných problémů

Následující sekce se zaměřuje na čas řešení SAT v jednotlivých vrstvách plánovacího grafu pro vybrané problémy z domény ZenoTravel. Pro toto zkoumání jsme zvolili problémy číslo 9 a 13, protože u těchto problémů došlo při líné kompilaci ve variantě A k nalezení řešení s menším počtem vrstev plánovacího grafu než u varianty N. U problému číslo 13 byla také varianta A výrazně rychlejší než varianta N.

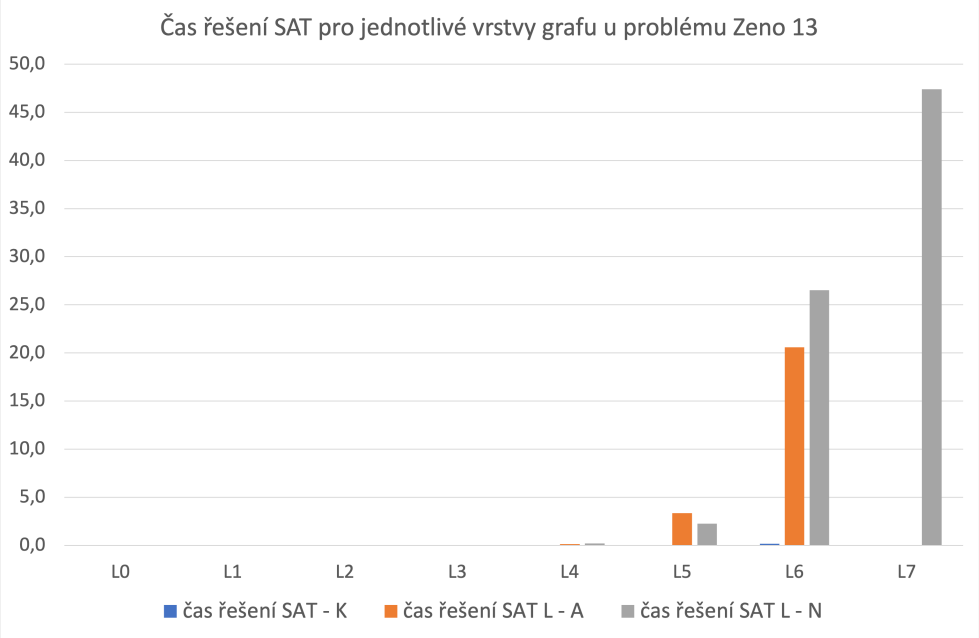
Výsledky měření jsou uvedeny v tabulkách 11.16, 11.17 a grafech 11.18, 11.19.

Na problému číslo 13 se ukazuje potenciální výhoda varianty A líné kompilace oproti variantě N. Čas řešení SAT na jednotlivých vrstvách narůstá a nalezení řešení na dřívější vrstvě může způsobit významné ušetření času. (Variantě A trvalo celkové vyřešení problému 47 sekund. U varianty N zabralo 47 sekund pouze řešení SAT na poslední vrstvě grafu, celkový čas řešení byl poté 138 sekund.)

Oproti ostatním doménám zde obecně stojí za zmínku velký počet opakování řešení formule s novými klauzulemi při líné kompilaci, pro některé vrstvy grafu u problému číslo 13 byl řešen SAT i více než tisíckrát.



Obrázek 11.18: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém ZenoTravel 9.



Obrázek 11.19: Celkový čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace pro problém ZenoTravel 13.

Z9	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	5,12E-04	1,86E-03	1,38E-03	1,33E-04	1,15E-04	14	12
L4	3,93E-04	3,31E-02	3,46E-02	3,18E-04	4,21E-04	104	82
L5	2,90E-03	7,10E-01	5,81E-01	1,85E-03	1,50E-03	384	386
L6	3,88E-03	-	8,60E-01	-	1,85E-03	-	464

Tabulka 11.16: Problém ZenoTravel 9: čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.

Z13	**čas K**	**čas L - A**	**čas L - N**	**čas iterace L - A**	**čas iterace L - N**	**počet opak. L - A**	**počet opak. L - N**
L0	0	0	0	0	0	0	0
L1	0	0	0	0	0	0	0
L2	0	0	0	0	0	0	0
L3	2,52E-03	5,69E-03	6,89E-03	1,78E-04	2,76E-04	32	25
L4	4,54E-03	1,28E-01	2,00E-01	5,30E-04	1,13E-03	241	177
L5	7,99E-03	3,35E+00	2,27E+00	4,09E-03	4,10E-03	818	553
L6	1,69E-01	2,06E+01	2,65E+01	1,42E-02	2,42E-02	1453	1098
L7	2,25E-02	-	4,74E+01	-	3,30E-02	-	1437

Tabulka 11.17: Problém ZenoTravel 13: čas řešení SAT na jednotlivých vrstvách plánovacího grafu pro různé druhy kompilace. Pro línou kompilaci je uveden také průměrný čas jedné iterace řešení SAT na dané vrstvě a počet opakování řešení formule s novými klauzulemi.

11.6 Shrnutí výsledků

Provedené experimenty se zaměřovaly na zjišťování parametrů výsledné formule Φ, měření celkového času potřebného pro vyřešení problémů a podrobnější měření času řešení SAT na jednotlivých vrstvách plánovacího grafu u vybraných problémů. V experimentech byl porovnáván plánovač s klasickou kompilací a s dvěma variantami líné kompilace A a N. Experimenty probíhaly dohromady na 79 problémech rozdílné obtížnosti ze čtyř domén. Podrobnější měření bylo provedeno celkem na devíti problémech ze všech čtyř domén. Tyto problémy byly vybrány podle předchozích měření tak, aby u nich podrobnější měření poskytlo bližší vysvětlení výsledků.

Experimenty ukázaly, že námi implementovaný plánovač dokáže řešit v rozumném čase plánovací problémy použité v soutěži IPC. V experimentech se dále podařilo porovnat výkon plánovače s klasickou a línou kompilací a také srovnat dvě navržené varianty líné kompilace A a N. Níže uvádíme některé poznatky zjištěné při provádění experimentů.

11.6.1 Srovnání klasické a líné kompilace

Chování plánovače s různými variantami kompilace se v jedné doméně (od jednodušších k těžším problémům) dalo většinou označit za konzistentní. (V tom smyslu, že klasická nebo líná kompilace se dala označit jako vhodnější přístup obecně pro danou doménu.) V průběhu provádění experimentů jsme vypozorovali dva aspekty - jeden pozitivně působící na rychlejší řešení problémů pomocí plánovače s línou kompilací a jeden aspekt působící naopak proti.

Výhoda plánovače s línou kompilací se projevovala v části vytváření plánovacího grafu, kdy nebylo potřeba konstruovat mutexy, což přineslo značnou úsporu času. V některých doménách byla část konstrukce mutexů časově náročná a mutexů vznikalo velké množství. Časové ztráty při líné kompilaci naopak způsobovala část řešení SAT pomocí SAT řešiče. Z podrobnějšího měření u vybraných problémů bylo zjištěno, že vezmeme-li čistý čas řešení SAT na jednotlivých vrstvách grafu pomocí SAT řešiče, líná kompilace potřebuje na vyřešení rovnic Φ zpravidla o něco více času než kompilace klasická. (Ať už z důvodu velkého počtu opakování řešení SAT nebo z důvodu vzniku těžších formulí.) V některých doménách byl tento rozdíl vzhledem k celkovému času potřebného pro vyřešení problému zanedbatelný. V doméně Blocks World vznikaly při líné kompilaci příliš těžké formule a větší problémy se proto nepodařilo vyřešit v časovém limitu 15 minut. Formule Φ vznikající při klasické kompilaci byla u stejných problémů snadno řešitelná. Domníváme se, že u těchto problémů se při klasické kompilaci uplatňovala ve velké míře jednotková propagace.

Ve většině případů v našich experimentech převažoval aspekt ušetření času při tvorbě plánovacího grafu a líná kompilace byla v celkovém součtu u většiny problémů rychlejší. (Plánovač s línou kompilací byl rychlejší u 63 problémů ze

79, naproti tomu plánovač s klasickou kompilací jen u 16 problémů – všechny byly z domény Blocks World.)

Výsledný čas řešení plánovacích problémů a to zda bude rychlejší líná či klasická kompilace ovlivňuje mnoho dalších faktorů jako je efektivita jednotlivých částí implementace (expanze plánovacího grafu, konstrukce mutexů, …) a v neposlední řadě taky výkon použitého SAT řešiče.

Obecně línou kompilaci označujeme za koncept, který má (minimálně v některých doménách) potenciál k vylepšení výkonu plánovače.

11.6.2 Srovnání varianty A a varianty N líné kompilace

V rámci experimentů jsme zkoumali dvě mírně rozdílné varianty líné kompilace A a N. Ve většině případů byly výsledky těchto dvou variant přibližně srovnatelné. U několika málo problémů byl rozdíl mezi těmito dvěma variantami líné kompilace významnější, zpravidla ve prospěch varianty A.

Ve variantě A dochází k postupnějšímu (línějšímu) doplňování klauzulí do formule a výsledná formule Φ má obvykle o něco méně klauzulí než ve variantě N. Varianta A se jevila jako mírně lepší ve více paralelních doménách, kde na poslední vrstvě grafu docházelo k méně opakování řešení SAT či bylo případně řešení nalezeno dokonce o vrstvu dříve než ve variantě N. Toto se u některých problémů ukázalo jako velmi časově výhodné (příkladem může problém ZenoTravel 13).

Ve variantě N se klauzule doplňují do formule rychleji (po více klauzulích najednou) a na některých vrstvách grafu díky tomu dochází k menšímu počtu opakování řešení SAT. V některých případech díky tomu dosahuje varianta N lepších výsledků. Někdy to však způsobuje naopak zbytečné opakování řešení SAT, zejména na poslední vrstvě grafu. Řešení se opakuje, dokud nejsou všechny akce na dané vrstvě nezávislé, zatímco varianta A skončí úspěšně dříve.

Závěr

Tématem této diplomové práce byla líná kompilace v klasickém plánování. V teoretické části práce byly nejprve shrnuty nejdůležitější informace o klasickém plánování a definovány důležité pojmy klasické reprezentace plánovacích problémů (plánovací doména, plánovací problém, plán, operátory, akce, …). Teoretická část obsahovala také stručný úvod do historie automatického plánování a byly zde představeny základní plánovací algoritmy, zejména prohledávání plánovacího stavového prostoru a techniky využívající plánovací graf. Poslední sekce se věnovala převodu (kompilaci) plánování na problém výrokové splnitelnosti (SAT).

Cílem praktické části práce byl návrh líného způsobu kompilace plánovacích problémů do SAT. Tento cíl se podařilo naplnit. Na základě studia teorie klasického plánování byl navržen líný způsob kompilace, který modifikuje běžně používanou klasickou kompilaci. Námi navržená metoda pracuje s vytvořením výrokové formule popisující neúplný plánovací graf pro daný problém (bez akčních a predikátových mutexů). Líný způsob poté spočívá v postupné identifikaci akcí, které ve vzniklém plánu způsobují chyby, a postupném přidávání nových klauzulí reprezentující mutexy těchto akcí do výsledné výrokové formule. V rámci praktické části práce byl dále implementován a otestován plánovač využívající dvě varianty kompilace – navrženou metodu pro línou kompilaci a kompilaci klasickou.

Experimenty v závěrečné části práce se zaměřovaly na vyhodnocení úspěšnosti plánovače s různými variantami kompilace. Sledován byl zejména čas, který plánovač potřebuje pro úspěšné vyřešení plánovacích problémů. Při experimentech byly využity plánovací úlohy ze soutěže IPC. Celkem bylo využito 79 problémů různé obtížnosti ze čtyř domén, 63 z nich dokázal plánovač s línou kompilací vyřešit rychleji než plánovač s klasickou kompilací. Provedené experimenty poukázaly na výhody a možné nevýhody líné kompilace. Výsledky experimentů naznačují, že využití líné kompilace má potenciál ke zlepšení výkonu plánovače.

Existuje mnoho způsobů, jak navázat na tuto práci. Jako velmi zajímavou

možnost rozšíření a pokračování práce vidíme návrh a vytvoření plánovače, který podle definice domény a plánovacího problému automaticky rozhodne, který způsob kompilace je pro daný případ nejvhodnější použít.

Literatura

1. SURYNEK, Pavel. Unifying Search-based and Compilation-based Approaches to Multi-agent Path Finding through Satisfiability Modulo Theories. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 2019, s. 1177–1183. Dostupné z DOI: 10.24963/ijcai.2019/164.
2. MAŘÍK, V.; ŠTĚPÁNKOVÁ, O.; LAŽANSKÝ, J. *Umělá inteligence*. Academia, 1993. Umělá inteligence, č. sv. 1. ISBN 9788020004963.
3. GHALLAB, Malik; NAU, Dana; TRAVERSO, Paolo. *Automated Planning: Theory and Practice*. Elsevier Science, 2004. The Morgan Kaufmann Series in Artificial Intelligence. ISBN 9781558608566.
4. SLANEY, John; THIÉBAUX, Sylvie. Blocks world revisited. *Artificial Intelligence*. 2001, roč. 125, č. 1-2, s. 119–153.
5. KAUTZ, Henry. The Role of Domain-Specific Knowledge in the Planning as SatisfiabUity Framework. In: *Proc. of AIPS*. 1998, sv. 98.
6. CHENOWETH, Stephen V. On the NP-Hardness of Blocks World. In: *AAAI*. 1991, s. 623–628.
7. GHALLAB, Malik; HOWE, Adele; KNOBLOCK, Craig; MCDERMOTT, Drew; RAM, Ashwin; VELOSO, Manuela; WELD, Daniel; WILKINS, David. *PDDL - The Planning Domain Definition Language*. 1998. Tech. zpr. Technical Report.
8. YOUNES, Håkan LS; LITTMAN, Michael L. PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*. 2004, roč. 2, s. 99.
9. FOX, Maria; LONG, Derek. PDDL+: Modeling continuous time dependent effects. In: *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*. 2002, sv. 4, s. 34.

10. *1st International Planning Competition* [online]. ICAPS [cit. 2021-10-23]. Dostupné z: `https://ipc98.icaps-conference.org`.

11. RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2016. ISBN 9781292153971. Dostupné také z: `https://cs.calvin.edu/courses/cs/344/kvlinden/resources/AIMA-3rd-edition.pdf`.

12. BONET, Blai; GEFFNER, Héctor. Planning as heuristic search. *Artificial Intelligence*. 2001, roč. 129, č. 1-2, s. 5–33.

13. HOFFMANN, Jörg. FF: The fast-forward planning system. *AI magazine*. 2001, roč. 22, č. 3, s. 57–57.

14. HELMERT, Malte. The fast downward planning system. *Journal of Artificial Intelligence Research*. 2006, roč. 26, s. 191–246.

15. FIKES, Richard E; NILSSON, Nils J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*. 1971, roč. 2, č. 3-4, s. 189–208.

16. BLUM, Avrim L.; FURST, Merrick L. Fast Planning Through Planning Graph Analysis. *ARTIFICIAL INTELLIGENCE*. 1995, roč. 90, č. 1, s. 1636–1642.

17. KOEHLER, Jana. Handling of conditional effects and negative goals in IPP. *Technical Report: report00128*. 1999.

18. ANDERSON, Corin R; SMITH, David E; WELD, Daniel S. Conditional Effects in Graphplan. In: *AIPS*. 1998, sv. 98.

19. LONG, Derek; FOX, Maria. Efficient implementation of the plan graph in STAN. *Journal of Artificial Intelligence Research*. 1999, roč. 10, s. 87–115.

20. BONNET, Blai; GEFFNER, Héctor. HSP: Heuristic search planner. 1998.

21. KAUTZ, Henry; SELMAN, Bart. BLACKBOX: A new approach to the application of theorem proving to problem solving. In: *AIPS98 workshop on planning as combinatorial search*. 1998, sv. 58260, s. 58–60.

22. *The International Planning Competition Series* [online]. ICAPS [cit. 2021-10-23]. Dostupné z: `https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node2.html`.

23. COLES, Amanda; COLES, Andrew; OLAYA, Angel García; JIMÉNEZ, Sergio; LÓPEZ, Carlos Linares; SANNER, Scott; YOON, Sungwook. A survey of the seventh international planning competition. *AI Magazine*. 2012, roč. 33, č. 1, s. 83–88.

24. SPECK, David; GEIßER, Florian; MATTMÜLLER, Robert. Symbolic planning with edge-valued multi-valued decision diagrams. In: *Twenty-Eighth International Conference on Automated Planning and Scheduling*. 2018.

25. *An Overview of the International Planning Competition* [online]. King's College London [cit. 2021-10-23]. Dostupné z: `https://www.nms.kcl.ac.uk/andrew.coles/PlanningCompetitionAAAISlides.pdf`.

26. CENAMOR, Isabel; DE LA ROSA, Tomás; FERNÁNDEZ, Fernando. The IBaCoP planning system: Instance-based configured portfolios. *Journal of Artificial Intelligence Research*. 2016, roč. 56, s. 657–691.

27. HELMERT, Malte; RÖGER, Gabriele; KARPAS, Erez. Fast downward stone soup: A baseline for building planner portfolios. In: *ICAPS 2011 Workshop on Planning and Learning*. 2011, s. 28–35.

28. KATZ, Michael; SOHRABI, Shirin; SAMULOWITZ, Horst; SIEVERS, Silvan. Delfi: Online planner selection for cost-optimal planning. *IPC-9 planner abstracts*. 2018, s. 57–64.

29. CENAMOR, Isabel; DE LA ROSA, Tomás; FERNÁNDEZ, Fernando. Learning predictive models to configure planning portfolios. In: *Proceedings of the 4th workshop on Planning and Learning (ICAPS-PAL 2013)*. 2013, s. 14–22.

30. KAUTZ, Henry A; SELMAN, Bart et al. Planning as Satisfiability. In: *ECAI*. 1992, sv. 92, s. 359–363.

31. KAUTZ, Henry A. Deconstructing planning as satisfiability. In: *Proceedings of the National Conference on Artificial Intelligence*. 2006, sv. 21, s. 1524. Č. 2.

32. BIERE, Armin; HEULE, Marijn; MAAREN, Hans van; WALSH, Toby (ed.). *Handbook of Satisfiability*. IOS Press, 2009. Frontiers in Artificial Intelligence and Applications. ISBN 978-1-58603-929-5.

33. KAUTZ, Henry; SELMAN, Bart; HOFFMANN, Joerg. SatPlan: Planning as satisfiability. In: *5th international planning competition*. 2006, sv. 20, s. 156. Č. 49.

34. RINTANEN, Jussi. Madagascar: Scalable planning with SAT. *Proceedings of the 8th International Planning Competition (IPC-2014)*. 2014, roč. 21.

35. TRLIFAJOVÁ, Kateřina; VAŠATA, Daniel. *Matematická logika*. České vysoké učení technické v Praze, 2013. ISBN 9788001053423.

36. KAUTZ, Henry; MCALLESTER, David; SELMAN, Bart. Encoding plans in propositional logic. *KR*. 1996, roč. 96, s. 374–384.

37. KAUTZ, Henry; SELMAN, Bart. SATPLAN04: Planning as satisfiability. *Working Notes on the Fifth International Planning Competition (IPC-2006)*. 2006, s. 45–46.

38. ROBINSON, Nathan; GRETTON, Charles; PHAM, Duc Nghia; SATTAR, Abdul. SAT-based parallel planning using a split representation of actions. In: *Nineteenth International Conference on Automated Planning and Scheduling*. 2009.

39. KAUTZ, Henry; SELMAN, Bart. Unifying SAT-based and graph-based planning. In: *IJCAI*. 1999, sv. 99, s. 318–325.

40. BUENO, Thiago P. *Pddlparser-pp* [soft.]. 2017 [cit. 2021-11-10]. Dostupné z: `https://github.com/thiagopbueno/pddlparser-pp`.

41. ESTES, Will. *Flex* [soft.]. 2017 [cit. 2021-11-10]. Dostupné z: `https://github.com/westes/flex`.

42. CORBETT, Robert. *Bison* [soft.]. 2021 [cit. 2021-11-10]. Dostupné z: `https://www.gnu.org/software/bison/`.

43. EÉN, Niklas; SÖRENSSON, Niklas. *MiniSat* [soft.]. 2005 [cit. 2021-11-10]. Dostupné z: `http://minisat.se/MiniSat.html`.

44. EÉN, Niklas; SÖRENSSON, Niklas. An extensible SAT-solver. In: *International conference on theory and applications of satisfiability testing*. 2003, s. 502–518.

45. *International Planning Competition 2002* [online]. ICAPS [cit. 2021-12-16]. Dostupné z: `https://ipc02.icaps-conference.org/domains.html`.

Příloha **A**

Seznam použitých zkratek

SAT satisfiability, problém splnitelnosti

IPC International Planning Competition

PDDL The Planning Domain Definition Language

ICAPS International Conference on Automated Planning and Scheduling

STRIPS The Stanford Research Institute Problem Solver

DWR doména Dock Worker Robots

CNF konjunktivní normální forma

CDCL Conflict-Driven Clause Learning

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF