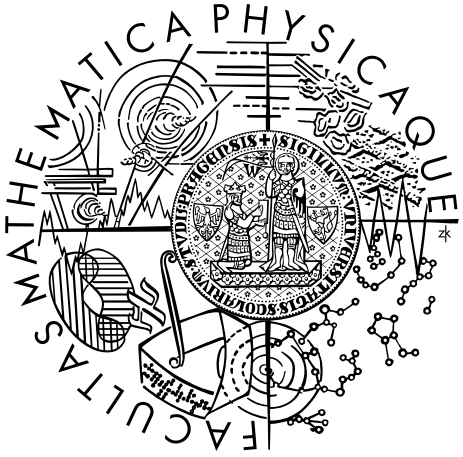
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Milan Ježek

Modelování kooperativního hledání cest

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. Pavel Surynek, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2015

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

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

V ..... dne ..... Podpis autora

Název práce: Modelování kooperativního hledání cest

Autor: Milan Ježek

Katedra: Katedra teoretické informatiky a matematické logiky

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

Abstrakt: V této práci jsou popsány nové modely pro řešení kooperativního hledání cest (CPF) s požadavkem na minimální makespan a je provedeno jejich experimentální porovnání se stávajícími modely. Nové modely uvedené v práci zkoumají možnosti kódování problému CPF pomocí celočíselného lineárního programování s binárními proměnnými (BIP) a jako problém splnitelnosti omezujících podmínek (CSP). Při testech se ukázaly hlavně poměrně dobré výsledky nového modelu IP active-edges při vyšším množství agentů, kdy jen mírně zaostával za nejlepším SAT modelem. Nový model pro CSP dosáhl nejrychlejších časů v testech s nízkým množstvím překážek a interakcí mezi agenty, zatímco v opačném případě se jeho výkon dramaticky snižoval.

Klíčová slova: cooperative path-finding, SAT, IP, CSP

Title: Modeling of Cooperative Path Finding

Author: Milan Ježek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Pavel Surynek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: In this thesis we describe new models for solving the cooperative path-finding (CPF) with the requirement of minimal makespan and experimental comparison with current models is performed. These new models investigate the possibilities of encoding the CPF problem into binary integer programming (BIP) or constraint satisfaction problem (CSP). Mainly the new active-edges IP model tests with high number of agents yielded good results, where it fell only slightly behind the best SAT model. A new CSP model reached the fastest times in tests with low number of obstacles and agent interactions while struggling heavily in the opposite cases.

Keywords: cooperative path-finding, SAT, IP, CSP

Rád bych poděkoval především svému vedoucímu bakalářské práce RNDr. Pavlu Surynkovi, Ph.D. za podporu a cenné rady při konzultacích v průběhu tvorby této práce.

Obsah

Úvod

V rámci této práce se zabýváme problémem kooperativního hledání cest na neorientovaných grafech (CPF) s dodatečným požadavkem na minimální makespan, tedy čas, během něhož se musí dostat všichni agenti ze své počáteční pozice do cílové pozice. Dále budeme tuto optimalizační variantu označovat jako (mCPF).

Problém CPF patří mezi aktuálně zkoumaná témata v oboru umělá inteligence s možným využitím při plánování pohybu jednotek v počítačových hrách (Botea et al., 2013; Silver, 2005), přesunů kontejnerů v námořních přístavech (Cho et al., 2008) nebo v mobilní robotice (Siméon, Leroy a Laumond, 2002). Další využití pak může nalézt při přepravě zboží v automatizovaných skladech nebo v budoucích komplexních systémech výtahů. Optimální varianta problému může mít výhodu především tam, kde se obdobné přesuny provádějí opakovaně a záleží celkovém času provedení úlohy.

Existuje několik různých definic CPF, které se vzájemně liší některými požadovanými podmínkami. V této práci se věnujeme především dvěma z nich, které se běžně používají. Přesné definice jsou popsány v následující kapitole 1.1.

Zatímco pro nalezení libovolného řešení jsou známy algoritmy běžící v kubickém čase vzhledem k počtu vrcholů grafu, ukazuje se, že problém, kdy je požadováno řešení s minimálním časem provedení, patří do třídy NP-úplných problémů (viz kap. 1.3). V současnosti stále není známo, zda lze pro problémy z této třídy složitosti sestrojit algoritmus schopný nalézt řešení v polynomiálním čase. Vzhledem k tomu, že jsou všechny NP-úplné problémy vzájemně polynomiálně redukovatelné, je možné je řešit tak, že je převedeme pomocí polynomiálního algoritmu na jiný NP-úplný problém, který řešíme namísto původního. To nám umožňuje využít k řešení mCPF specializované knihovny pro řešení klasických NP-úplných problémů. V práci porovnáváme již existující modely pro SAT s námi nově vytvořenými modely reprezentovanými jako CSP a BIP. U prvé ze dvou variant problému navíc provádíme srovnání s existujícím algoritmem OD+ID. Jelikož se algoritmy pro řešení úloh popsaných jako CSP, SAT a IP často používají k řešení složitých kombinatorických úloh, měly by být takto zakódované modely mCPF schopné poradit si i s větší hustotou agentů než jiné algoritmy pro CPF.

Cíle práce

- Hlavním cílem práce je porovnat vhodnost použití vybraných formalismů CSP, SAT a IP pro modelování problému mCPF. Za tímto účelem se snažíme pro každý formalismus zvolit vhodnou knihovnu, která patří mezi nejlepší ve své kategorii.

- Zatímco pro SAT existuje již několik různých modelů (viz kap. 4), pro celočíselné programování jsme nalezli pouze řešení, které reprezentuje mCPF jako multikomoditní toky v grafu a tuto reprezentaci modeluje pomocí celočíselného programování (Yu a LaValle, 2013a). V CSP existuje v současnosti model pro CPF, který využívá vlastností několika druhů podgrafů (Ryan, 2008), tento model však nemusí vždy vracet optimální řešení. Vidíme tak

prostor pro vytvoření nových modelů v CSP a v IP, které můžeme následně použít k porovnání.

- Jelikož existující algoritmy pracují často s jednou ze dvou různých definic CPF (viz kap. 1.1.), rozhodli jsme se vytvořit nové modely tak, aby zahrnovaly obě dvě a rovněž testy provádíme zvlášť pro každou z definic.

Očekávané výsledky

- Od modelů pro IP si slibujeme výsledky o něco slabší než u modelů pro SAT, k tomu nás vedou celkem dobré výsledky existujících modelů a zároveň se nám jeví popis CPF pomocí IP jako méně přirozený. Na druhou stranu stávající model založený na multikomoditních tocích se ukazuje jako poměrně úspěšný (Yu a LaValle, 2013a). Jelikož ale pracují nynější modely pro SAT a pro IP s odlišnými definicemi problému, nemáme zatím přímé srovnání.

- Výhodou CSP modelů oproti ostatním by měla být jednak možnost využít pro jejich popis některé globální omezující podmínky umožňující řešiči použít specializované algoritmy a dále pak možnost snadného přidání vlastní heuristiky pro volbu hodnot proměnných využívající znalost problému. Z toho důvodu předpokládáme, že by měly být tyto modely efektivnější především pro úlohy, kde se během své cesty nemusí agenti často vzájemně vyhýbat.

Struktura práce

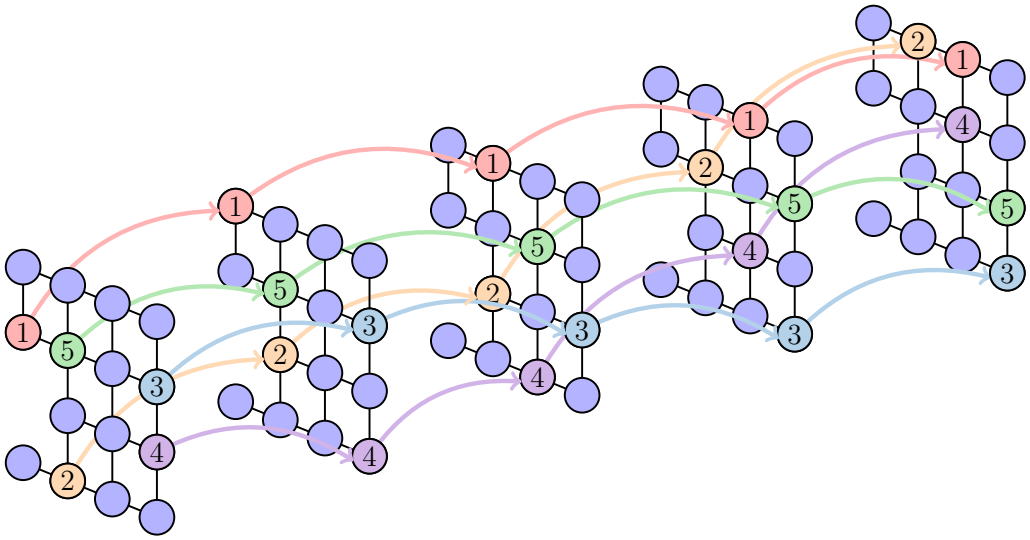
Na začátku jsou představeny definice jednotlivých variant problému a jsou zmíněny informace o některých existujících algoritmech. Zmíněny jsou rovněž známé informace o složitosti problému. Dále jsou stručně popsány použité formalismy pro modelování kombinatorických úloh.

V následujících kapitolách je uveden nejprve obecný algoritmus pro mCPF a určování dolních odhadů minimální délky řešení. Poté jsou stručně uvedeny použité SAT modely, po kterých je popsán nový CSP model společně s několika alternativními heuristikami a také dva nové BIP modely.

Následuje popis provedení experimentálního porovnání a jsou uvedeny výsledky jednotlivých testů. Nakonec provádíme závěrečné zhodnocení dosažených výsledků a uvažujeme možnosti, jak by se dalo na tuto práci navázat.

Přílohy obsahují postupně popis obsahu elektronické přílohy, návod pro uživatele vytvořeného programu a informace o jeho implementaci. Poslední přílohou je popis použitých formátů souborů.

1. Kooperativní hledání cest pro více robotů



Obrázek 1.1: Časově expandovaný graf s řešením úlohy CPF

Plánování cest patří mezi klasické problémy umělé inteligence. V této práci uvažujeme úlohu, kdy máme dáno prostředí a k němu zvolenou množinu agentů a množinu překážek. Každý z agentů má v prostředí pevně určenu svou počáteční a cílovou pozici. Úkolem je naplánovat pro každého agenta cestu z počáteční pozice do cílové tak, aby se agenti při simultánním pohybu vyhnuli všem překážkám a zároveň aby se nedostali do kolize s žádným jiným agentem.

Prostředí můžeme abstraktně reprezentovat pomocí navigačního grafu a případné překážky v něm pak jednoduše vynecháním příslušných vrcholů v grafu na místech, kde se překážky nacházejí. Příklad vyřešené úlohy je znázorněn na obrázku 1.1.

Případ kdy hledáme cestu pro jediného agenta lze obvykle řešit efektivně nalezením nejkratší cesty v navigačním grafu pomocí prohledávacího algoritmu jako je A* (Hart, Nilsson a Raphael, 1968).

Řešení úlohy hledání cest pro více agentů (MAPF) dělíme na kooperativní (CPF), kdy mezi sebou agenti sdílí veškerou informaci o svých plánech a na nekooperativní, kdy se hledá cesta pro každého agenta nezávisle. Dále budeme uvažovat jen kooperativní hledání cest.

Na řešení úlohy hledání cest pro více agentů můžeme rovněž použít algoritmus A*, kde uvažujeme každý stav jako společné rozmístění všech agentů. Ačkoliv je tento algoritmus kompletní a umožňuje nalézt optimální řešení, ukazuje se v praxi jako neefektivní i pro malé grafy, neboť se s rostoucím množstvím agentů zvětšuje stavový prostor i větvící faktor exponenciálně. Tento přístup patří mezi centralizované, protože hledá cesty pro všechny agenty najednou.

Necentralizované přístupy se oproti tomu snaží úlohu co nejvíce zjednodušit tak, že jí rozloží na více podúloh a snaží se nalést cestu postupně pro jednotlivé agenty, nebo jejich skupiny. Pro zamezení kolizí se využívají rezervační tabulky.

Tyto algoritmy také mohou dynamicky volit pořadí agentů nebo jejich skupin, pro které se postupně hledají cesty. Mezi takové algoritmy patří mimo jiné LRA*, CA*, HCA*, WHCA* (Silver, 2005), MAPP (Wang a Botea, 2011) nebo SDP (Wilt a Botea, 2014). Uvedené algoritmy jsou poměrně efektivní pro hledání cest s nižší hustotou agentů. Jejich nevýhodou ale je, že nejsou kompletní a negarantují nalezení optimálního řešení.

Mezi algoritmy, které dokážou nalézt vždy řešení pro graf alespoň se dvěma volnými vrcholy, patří například Push and Rotate (Wilde, Mors a Witteveen, 2013), nebo Push and Swap (Luna a Bekris, 2011). Jiný kompletní algoritmus využívá dekompozice grafu na různé speciální a často se vyskytující typy podgrafů jako jsou zásobník, hala, klika či kružnice a problém reprezentuje jako CSP model (Ryan, 2007).

Kromě algoritmů pro obecné neorientované grafy jsou i algoritmy omezující se jen na grafy splňující další požadované vlastnosti, z nich uvedeme jen algoritmus Bibox (Surynek, 2009). Ten je určen pro hledání cest na vrcholově 2-souvislých grafech s nejméně dvěma neobsazenými vrcholy, což je poměrně velká skupina v praxi použitelných grafů.

Algoritmy pro nalezení optimálního řešení uvádíme samostatně v části 1.2.

1.1 Různé varianty problému CPF

Varianta problému se již dříve zkoumala jako pohyb kamenů po grafu (PBO) (Kornhauser, 1984; Wilson, 1974), která vycházela ze známé hry 15 puzzle. Tato varianta umožňuje pohyb pouze jednoho kamene (agenta) na sousední volný vrchol současně.

Zde uvádíme dvě různé definice úlohy CPF. Definice 5 oproti pohybu kamenů po grafu umožňuje současný pohyb více agentů. Dále ji budeme označovat jako CPF-vacant-target.

Definice 6 navíc povoluje pohyb agentů na místo, které je sice před vykonáním pohybu obsazené jiným agentem, ale takovým, který se rovněž přesouvá do jiného vrcholu. Výjimkou je vzájemný pohyb dvou agentů po stejné hraně opačným směrem – ten z praktických důvodů zakazujeme. Platí, že pokud máme řešení úlohy splňující prvou definici, je to zároveň řešení úlohy dle druhé definice. Druhou definici budeme v textu označovat jako CPF-no-head-on.

Varianty obou definic, kde se požaduje navíc minimální makespan, označíme jako mCPF-vacant-target a mCPF-no-head-on.

V následujících definicích představuje $G = (V,E)$ neorientovaný graf a A množinu agentů.

Definice 1 (Rozmístění agentů)**.** *Rozmístění agentů v grafu G je prostá funkce $\alpha : A \to V$*

Definice 2 (Obsazení vrcholů)**.** *Obsazení vrcholů v grafu G je funkce $\alpha^{-1} : V \to A \cup \{\perp\}$, kde:*
(i) $\forall v \in V \quad (\exists a \in A \quad \alpha(a) = v) \Rightarrow \alpha^{-1}(v) = a$
(ii) $\forall v \in V \quad (\forall a \in A \quad \alpha(a) \neq v) \Rightarrow \alpha^{-1}(v) = \perp$ *(prázdný vrchol)*

Definice 3 (Instance kooperativního hledání cest (CPF)). *Instance CPF je čtveřice $\Sigma = (G, A, s, t)$, kde:*
$s : A \to V$ je rozmístění agentů určující počáteční stav
$t : A \to V$ je rozmístění agentů určující cílový stav

Definice 4 (Skupinové přemístění agentů pro makespan k). *Skupinové přemístění agentů pro makespan k v grafu G je soubor rozmístění agentů $\alpha_i : A \to V$, kde $i, k \in \mathbb{N},\ 0 \leq i \leq k$, splňující:*
$\forall i \in \mathbb{N},\ i < k, \forall a \in A \quad \alpha_{i+1}(a) = \alpha_i(a) \vee \{\alpha_i(a), \alpha_{i+1}(a)\} \in E$

Definice 5 (Úloha CPF-vacant-target). *Úlohou CPF-vacant-target je nalézt pro instanci CPF $\Sigma = (G, A, s, t)$ skupinové přemístění agentů pro nějaký makespan k $\alpha_i : A \to V$, kde $0 \leq i \leq k$, takové, že:*
(i) $\forall a \in A \quad \alpha_0(a) = s(a)$
(ii) $\forall a \in A \quad \alpha_k(a) = t(a)$
(iii) $\forall a \in A \quad \alpha_{i+1}(a) \neq \alpha_i(a) \Rightarrow \alpha_{i+1}^{-1}(a) = \perp$
Tato přemístění nazýváme řešení úlohy CPF-vacant-target s makespan k.

Definice 6 (Úloha CPF-no-head-on). *Úlohou CPF-no-head-on je nalézt pro instanci CPF $\Sigma = (G, A, s, t)$ skupinové přemístění agentů pro nějaký makespan k $\alpha_i : A \to V$, kde $0 \leq i \leq k$, takové, že:*
(i) $\forall a \in A \quad \alpha_0(a) = s(a)$
(ii) $\forall a \in A \quad \alpha_k(a) = t(a)$
(iii) $\forall a, b \in A, a \neq b \quad \alpha_{i+1}(a) = \alpha_i(b) \Rightarrow \alpha_i(a) \neq \alpha_{i+1}(b)$
Tato přemístění nazýváme řešení úlohy CPF-no-head-on s makespan k.

Definice 7 (Úloha mCPF-vacant-target). *Úloha mCPF-vacant-target je úloha CPF-vacant-target s dodatečnou podmmínkou:*
(i) Řešeními úlohy mCPF-vacant-target jsou právě ta řešení úlohy CPF-vacant-target s makespan k, kde k je minimální.

Definice 8 (Úloha mCPF-no-head-on). *Úloha mCPF-no-head-on je úloha CPF-vacant-target s dodatečnou podmmínkou:*
(i) Řešeními úlohy mCPF-no-head-on jsou právě ta řešení úlohy CPF-no-head-on s makespan k, kde k je minimální.

1.2 Existující algoritmy hledající optimální řešení problému CPF

Optimální řešení problému CPF je možno hledat vzhledem k různým cílovým funkcím. Náš přístup uvažuje jako optimální řešení to, které má minimální makespan. Tento požadavek nazveme min-makespan. Jiné algoritmy hledají často řešení s minimálním celkovým součtem počtů časových kroků, které jednotlivý agenti potřebují, aby se dostaly do své cílové pozice a v ní již setrvali. Nazvěme tento požadavek jako min-total-cost. Alternativně lze uvažovat podmínku, kdy hledáme řešení s nejmenším součtem všech přesunů jednotlivých agentů z aktuálního do jiného vrcholu.

- OD+ID (Standley a Korf, 2011) je kompletní aproximační algoritmus. Využívá rozdělování agentů do co nejmenších nezávislých skupin a oproti A* mění při prohledávání v prohledávacích uzlech vždy jen pozici jediného agenta oproti předchozímu uzlu. Umožňuje nalézt řešení s optimálním min-total-cost. Ve srovnání používáme upravenou implementaci hledající řešení úlohy mCPF-vacant-target.

- Algoritmus ICTS (Sharon, Stern, Goldenberg et al., 2011) dokáže nalézt optimální řešení úlohy CPF-no-head-on pro min-total-cost.

- CBS (Sharon et al., 2012a) hledá optimální řešení vhledem k min-total-cost.

- Algoritmy MA-CBS (Sharon et al., 2012b) a ICBS (Boyarski et al., 2015) vychází z algoritmu CBS, který dále vylepšují.

- Model založený na multikomoditních tocích v sítích (Yu a LaValle, 2013a) je kódován jako IP model a umožňuje nalézt řešení úlohy CPF-no-head-on jak pro min-makespan, tak i pro min-total-dist.

- Modely pro SAT (viz kap. 4) umožňují nalézt řešení úlohy mCPF-vacant-target.

1.3 Složitost kooperativního hledání cest

- Pro nalezení libovolného řešení úlohy CPF-vacant-target existují algoritmy se složitostí $O(|V|^3)$, kde $G = (V,E)$ je neorientovaný graf (Goraly a Hassin, 2010; Röger a Helmert, 2012). Stejný výsledek platí i pro nalezení libovolného řešení úlohy CPF-no-head-on (Yu a Rus, 2015).

- V případě kdy nás zajímá optimální řešení bylo ukázáno, že nalezení řešení úlohy mCPF-vacant-target a rovněž tak i nalezení řešení úlohy mCPF-no-head-on jsou NP-úplné problémy (Surynek, 2010; Yu a LaValle, 2013b)

- Poměrně nedávno bylo dokázáno, že pro ověření, zda má úloha CPF-vacant-target vůbec nějaké řešení, existuje algoritmus se složitostí $O(|V|+|E|)$ pro neorientovaný graf $G = (V,E)$ (Yu, 2013). Důkaz byl proveden pro problém PBO, je ale zřejmé, že pokud připustíme možnost paralelních pohybů agentů na neobsazené vrcholy, řešitelnost problému se nezmění. Algoritmus se stejnou časovou složitostí byl ukázán i pro variantu CPF-no-head-on (Yu a Rus, 2015).

2. Modelování kombinatorických úloh

Jelikož je optimalizační varianta kooperativního hledání cest NP-úplná, je možné k jejímu řešení využít polynomiální převoditelnost na jiný NP-úplný problém. V této práci jsme vybrali SAT, CSP a IP, které zde stručně popisujeme.

2.1 Splnitelnost výrokových formulí (SAT)

Problém splnitelnosti výrokových proměnných je jedním z prvních problémů, o kterém bylo dokázáno, že je NP-úplný. Vývoj algoritmů pro jeho řešení je dlouhodobý a intenzivní. Takto popsané modely mCPF by tak měly být pomocí stávajících SAT řešičů poměrně efektivně řešitelné oproti řešení v méně známých formalismech.

Výroková proměnná může nabývat buď hodnoty PRAVDA, nebo hodnoty NEPRAVDA. Výrokovou formuli můžeme posat induktivně. Každá výroková proměnná je výroková formule. Jsou-li α a β výrokové formule, pak i $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$ a $(\neg\alpha)$ jsou výrokové formule.

Výroková formule je splnitelná, právě když existuje takové ohodnocení jejích výrokových proměnných hodnotami PRAVDA, nebo NEPRAVDA tak, aby byla hodnota formule PRAVDA.

2.2 Problémy splňování podmínek (CSP)

Vývoji metod na řešení problémů splňování podmínek se rovněž aktivně věnuje výzkum v rámci umělé inteligence.

Problém zahrnuje konečnou množinu proměnných a konečnou množinu podmínek. Ke každé proměnné máme navíc dánu konečnou množinu hodnot nazývanou doména. Ta určuje, jakých hodnot může daná proměnná nabývat. Podmínkou rozumíme n-ární relaci na doménách n proměnných pro nějaké přirozené číslo n.

Jako přípustné řešení CSP označujeme ohodnocení proměnných hodnotami z jejich domén tak, aby byly splněny zároveň všechny podmínky.

2.3 Celočíselné programování (IP)

Celočíselné programování slouží k popisu optimalizačních úloh, jejichž proměnné mohou nabývat pouze celočíselných hodnot. V tomto textu pod tímto pojmem automaticky uvažujeme lineární celočíselné programování, kdy jsou k popisu přípustných řešení použity jen lineární rovnice.

Úlohou celočíselného programování je nalézt takové ohodnocení proměnných celými čísly, aby byly splněny všechny lineární podmínky a navíc aby byla při tomto ohodnocení hodnota zvolené lineární cílové funkce minimální, popřípadě maximální možná, dle toho, zda se jedná o minimalizační, či maximalizační úlohu.

Speciální variantou celočíselného programování je lineární celočíselné programování s binárními proměnnými (BIP).

3. Hledání optimálního řesení úlohy CPF

Uvádíme zde obecný metaalgoritmus pro hledání řešení s minimálním makespan, který lze použít společně s libovolným algoritmem určeným pro nalezení řešení s konkrétním makespan, pokud existuje. Metaalgoritmus je možné použít pro obě zde uvažované varianty problému CPF.

Algorithm 1 Hledání řešení úlohy CPF s minimálním makespan

```
Input: úloha                ▷ úloha mCPF-no-head-on, nebo mCPF-vacant-target

jeNeřešitelná ← ověřŘešitelnostÚlohy(úloha)
if jeNeřešitelná then
    return null
end if
makespan ← určiDolníOdhadMinimálníhoMakespan(úloha)
repeat
    řešení ← hledejŘešeníProDanýMakespan(úloha, makespan)
    makespan = makespan + 1
until řešení ≠ null
return řešení
```

Funkci ověřŘešitelnostÚlohy(úloha) lze naprogramovat tak, aby rozhodla v čase $O(|V| + |E|)$, kde (V,E) je neorientovaný graf, zda má daná úloha řešení (viz kap. 1.3). Naše implementace obsahuje pouze neúplnou variantu, která především ověřuje, zda existuje pro každého agenta cesta z jeho počáteční pozice do jeho cílové pozice (viz kap. 3.1). Toto zjednodušené ověřování nám stačilo pro odhalení naprosté většiny neřešitelných instancí, které jsme při testování generovali.

Funkce určiDolníOdhadMinimálníhoMakespan(úloha) slouží k rychlému určení co nejpřesnějšího dolního odhadu na velikost minimálního makespan řešení a díky čemuž se minimalizuje počet řešených úloh, které nemají pro daný makespan žádné řešení. Odhady, které jsme použili v naší implementaci, jsou popsány v části 3.2.

Jako hledejŘešeníProDanýMakespan(úloha, makespan) můžeme použít libovolný úplný algoritmus, který vrací správné řešení, pokud řešení dané délky existuje, nebo „null“, pokud neexistuje.

Pokud jsou funkce ověřŘešitelnostÚlohy a hledejŘešeníProDanýMakespan kompletní, vrací algoritmus optimální řešení, právě když je úloha řešitelná, jinak vrací „null“.

3.1 Kontrola řešitelnosti úlohy

V této práci používáme zjednodušenou kontrolu, zda je úloha CPF řešitelná.

- Kontrolujeme, zda si odpovídají počáteční a cílový počet robotů

- Pro každého agenta kontrolujeme dosažitelnost jeho cílové pozice z počáteční pozice

Pro makespan velikosti 0 a 1 řešíme problém přímo. Pokud je makespan = 0, pak kontrolujeme, zda se shoduje počáteční a koncové rozmístění. Pro makespan = 1 pak kontrolujeme, zda počáteční pozice agentů sousedí s jejich cílovými pozicemi, přitom pro definici CPF-vacant-target navíc ověřujeme, zda jsou cílové vrcholy volné a u definice CPF-no-head-on zjišťujeme, zda nedošlo k prohození agentů po stejné hraně.

3.2 Dolní odhady minimálního makespan

V této části rozebíráme možnosti jak rychle zdola odhadnout optimální makespan pro danou úlohu CPF. Pro zjednodušení zápisu definujeme několik značení.

Definice 9 (Délka nejkratší cesty pro agenta)**.** *Délkou nejkratší cesty pro agenta $a \in A$ v instanci CPF $\Sigma = (G, A, s, t)$ nazýváme délku nejkratší cesty z vrcholu $s(a)$ do vrcholu $t(a)$ v grafu G a označujeme ji $\delta_\Sigma(a)$.*

Definice 10 (Nejkratší vzdálenost k volnému místu pro agenta v počátku)**.** *Nejkratší vzdálenost k volnému místu pro agenta $a \in A$ v počátku v instanci CPF $\Sigma = (G, A, s, t)$ nazýváme délku nejkratší cesty z vrcholu $s(a)$ do nejbližšího vrcholu $v \in V$ v grafu G, pro který platí $s^{-1}(v) = \perp$, a označujeme ji $\delta^s_\Sigma(a)$.*

Definice 11 (Nejkratší vzdálenost k volnému místu pro agenta v cíli)**.** *Nejkratší vzdálenost k volnému místu pro agenta $a \in A$ v cíli v instanci CPF $\Sigma = (G, A, s, t)$ nazýváme délku nejkratší cesty z vrcholu $t(a)$ do nejbližšího vrcholu $v \in V$ v grafu G, pro který platí $t^{-1}(v) = \perp$, a označujeme ji $\delta^t_\Sigma(a)$.*

K dolnímu odhadu makespan používáme vždy ten větší z následujících dvou odhadů. Zatímco první odhad je určen pro obě varianty problému mCPF, druhý je určený pouze pro mCPF-vacant-target.

- $\max_{a \in A}(\delta^s_\Sigma(a) + \delta_\Sigma(a) + \delta^t_\Sigma(a))$
- $\dfrac{\sum_{a \in A} \delta_\Sigma(a)}{\min(|A|, |V| - |A|)}$

První odhad vyjadřuje, že minimální makespan musí být alespoň tak vysoký, aby se u každého agenta nejprve vyskytlo volné místo, poté musí agent urazit cestu alespoň o délce minimální vzdálenosti z jeho počáteční pozice do cílové a nakonec se musí kolem agenta seskupit ostatní agenti tak, aby jeho nejkratší vzdálenost k volnému místu v cílové pozici odpovídala dané instanci CPF.

Druhý odhad říká, že pokud se agenti mohou pohybovat pouze na volná místa, pak v každém kroku mohou udělat agenti současně maximálně $\min(|A|, |V| - |A|)$ kroků. Přitom minimální počet kroků, který musí agenti nutně urazit, je dán součtem délek jejich nejkratších cest do cíle.

4. Použité SAT Modely

Pro řešení kooperativního hledání cest jako SAT již existuje několik různých modelů. Zde jmenujeme modely, které jsme vybrali pro porovnání. Všechny tyto modely jsou vytvořené pouze pro definici CPF-vacant-target. Nejsou proto použity v testech pro CPF-no-head-on.

Pro testování jsme vybrali následující tři modely:

- sat_Hdifferential (Surynek, 2012a)
- sat_Hadvanced (Surynek, 2012b)
- sat_Hmatching (Surynek, 2014)

Uvedené modely jsou popsány podrobněji v odkazovaných článcích. Všechny výše zmíněné modely jsou společně s algoritmem OD+ID implementovány v knihovně reLOC_base, kterou využívá námi vytvořená aplikace.

5. Model CSP all-diff

V průběhu práce jsme vytvořili několik různých modelů pro CSP, z nichž se nejvíce osvědčil model CSP all-diff. Dále budeme popisovat jen tento model a heuristiky, které jsme pro něj připravili. U modelu lze parametrem zvolit, která definice problému CPF se má použít. Kromě definic CPF-vacant-target a CPF-no-head-on podporuje i definici, která navíc povoluje vzájemné prohození agentů po stejné hraně (nazýváme ji no-meet).

Model využívá globální podmínky „alldifferent“ (jinak také „distinct“), a „channel“, „⊆“. Významné je také omezení výsledného modelu tak, že se kontroluje, do jakých vrcholů se mohou jednotlivý agenti po různém počtu kroků dostat, a případné nedostupné situace se do modelu nezahrnou. Rovněž se kontroluje, ve kterých vrcholech se mohou v jednotlivých vrstvách setkat dvojice agentů – pak se omezí množství přidaných podmínek do modelu.

5.1 Popis modelu

Pro daný makespan m, konečnou množinu kroků K, agentů A, vrcholů V a pro konečné množiny sousedů vrcholů N_v, kde $|K| = m, |A| \leq |V|$ a $\forall v : N_v \subseteq V \backslash \{v\}$ rozhodujeme, zda existuje řešení délky $\leq m$.

Máme dány proměnné:

$$\text{pozice}[k,a] = v \Leftrightarrow \text{v kroku } k \text{ se agent } a \text{ nachází ve vrcholu } v \tag{5.1}$$

$$\text{jeNaPozici}[k,a,v] \Leftrightarrow \text{v kroku } k \text{ se agent } a \text{ nachází ve vrcholu } v \tag{5.2}$$

Podmínky:
Počáteční pozice:

$$\forall a \in A, \quad \text{pozice}[0,a] = \text{počátečníPozice}[a] \tag{5.3}$$

Cílové pozice:

$$\forall a \in A, \quad \text{pozice}[m,a] = \text{cílovéPozice}[a] \tag{5.4}$$

V každém kroku musí být různí agenti v různých vrcholech.

$$\forall k \in K, \quad \text{alldifferent}(\{\text{pozice}[k,a] | a \in \text{Agenti}\}) \tag{5.5}$$

channel - propojeni pozice a jeNaPozici

$$\forall k \in K, \forall a \in A, \quad \text{channel}(\{jeNaPozici(k,a,v) | v \in V\}, pozice(k,a)) \tag{5.6}$$

Nastavení možných přechodů pro každého agenta v každém kroku kromě toho posledního

$$\forall k \in K, \forall a \in A, \forall v \in V, \quad \text{jeNaPozici}[k,a,v] \Rightarrow \text{pozice}[k+1,a] \subseteq N_v \tag{5.7}$$

Zakazuje vstup agenta zpět do vrcholu, ze kterého se právě přesunul. (pouze pro CPF-vacant-target)

$$\begin{gathered}\forall v \in V, \forall a \in A, \forall k_1,k_2,k_3 \in K, \quad k_3 = k_2 + 1 \wedge k_2 = k_1 + 1 \rightarrow \\ ((\text{pozice}[k_3, a] \neq \text{pozice}[k_2, a]) \rightarrow (\text{pozice}[k_3, a] \neq \text{pozice}[k_1, a]))\end{gathered} \tag{5.8}$$

Požadavek na volné místo před agentem. (pouze pro CPF-vacant-target)

$$\forall a_1, a_2 \in A, a_1 \neq a_2, \forall k \in K, \quad \text{pozice}[k+1, a_1] \neq \text{pozice}[k, a_2] \tag{5.9}$$

Zákaz prohození agentů po stejné hraně. (pouze pro CPF-no-head-on)

$$\begin{gathered}\forall v \in V, \forall a_1, a_2 \in A, \forall k_1, k_2 \in K, \quad k_2 = k_1 + 1 \wedge a_1 \neq a_2 \rightarrow \\ ((\text{pozice}[k_2, a_1] = \text{pozice}[k_1, a_2]) \rightarrow (\text{pozice}[k_2, a_2] \neq \text{pozice}[k_1, a_1]))\end{gathered} \tag{5.10}$$

5.1.1 Heuristiky

Model CSP all-diff vybírá nejprve proměnné dle první podmínky, pokud je jich více, pak z nich pak volí ty, splňující druhou podmínku a z nich nakonec vybírá náhodně.

1. proměnné $\max\_v \in Vars$ stupeň(v)/velikost(v)
2. proměnné $\min\_v \in Vars$ velikost(v)
3. náhodná proměnná

Navíc umožňuje uživateli zvolit si pomocí paremetru jednu z následujících heuristik pro výběr hodnoty proměnných.

- max-range – předdefinovaná heuristika, vybírá co největší rozsah hodnot domény proměnné, pokud je jich více, jinak hodnotu větší než průměr z nejmenší a největší možné hodnoty.
- to-start – vybírá hodnoty, které odpovídají přesunům agentů co nejblíže směrem k jejich počáteční pozici.
- to-dest – vybírá hodnoty, které odpovídají přesunům agentů co nejblíže směrem k jejich cílové pozici.
- to-start-to-dest – pro proměnné určující pozice ve vrcholech v čase menším než makespan/2 vybírá hodnoty, které odpovídají přesunům agentů co nejblíže k jejich počáteční pozici, jinak co nejblíže k jejich cílové pozici.
- to-dest-to-start – pro proměnné určující pozice ve vrcholech v čase menším než makespan/2 vybírá hodnoty, které odpovídají přesunům agentů co nejblíže k jejich cílové pozici, jinak co nejblíže k jejich počáteční pozici.
- smooth – pro proměnné určující pozice ve vrcholech v čase k vybírá hodnoty, které odpovídají přesunům agentů co nejblíže k pozici, kde se poměr vzdálenost k počátku/(vzdálenost z počátku+vzdálenost k cíly) co nejvíce blíží poměru k/makespan/2.

6. IP Modely

Jedním z cílů práce bylo pokusit se vytvořit nějaké nové vhodné modely pro celočíselné programování (IP). Jako vhodný přístup se nakonec ukázalo popsat modely pomocí lineárního celočíselného programování s binárními proměnnými (BIP).

6.1 IP all-diff

Definice 12 (Kódování IP all-diff)**.** *Pro daný makespan m, konečnou množinu kroků K, agentů A, vrcholů V a pro konečné množiny sousedů vrcholů N_v, kde $|K| = m+1, |A| \leq |V|$ a $\forall v \quad N_v \subseteq V \backslash \{v\}$ rozhodujeme, zda existuje řešení délky $\leq m$.*
Máme dány binární proměnné:

$$^{k}V_v^a = 1 \Leftrightarrow v \text{ kroku } k \text{ je agent } a \text{ ve vrcholu } v \tag{6.1}$$

Cíl:

$$Maximalizace \quad \sum_{k \in K, a \in A, v \in V} {}^{k}V_v^a \tag{6.2}$$

Podmínky:
Počáteční rozmístění agentů

$$^{0}V_v^a = 1 \Leftrightarrow v \text{ počátečním rozmístění je agent } a \text{ ve vrcholu } v \tag{6.3}$$

Cílové rozmístění agentů

$$^{m}V_v^r = 1 \Leftrightarrow v \text{ cílovém rozmístění je agent } r \text{ ve vrcholu } v \tag{6.4}$$

Každý agent je v každém kroku umístěn v právě jednom vrcholu

$$\forall k \in K, \forall a \in A, \quad \sum_{v \in V} {}^{k}V_v^a = 1 \tag{6.5}$$

V každém vrcholu je v každém kroku nejvýše jeden agent

$$\forall k \in K, \forall v \in V, \quad \sum_{a \in A} {}^{k}V_v^a \leq 1 \tag{6.6}$$

Možná přemístění

$$\forall k \in K \backslash \{m\}, \forall v \in V, \forall a \in A, \quad {}^{k+1}V_v^a + \sum_{n \in N_v} {}^{k+1}V_n^a - {}^{k}V_v^a \geq 0 \tag{6.7}$$

Požadavek na volné místo před agentem (pouze pro CPF-vacant-target)

$$\forall k \in K \backslash \{m\}, \forall v \in V, \forall a_1 \in A, \quad {}^{k}V_v^{a_1} + \sum_{a_2 \in A \backslash \{a_1\}} {}^{k+1}V_v^{a_2} \leq 1 \tag{6.8}$$

Zákaz prohození agentů po hraně (pouze pro CPF-no-head-on)

$$\forall k \in K \backslash \{m\}, \forall v \in V, \forall n \in N_v, \forall a_1, a_2 \in A, a_1 \neq a_2, \\ {}^{k}V_v^{a_1} + {}^{k+1}V_n^{ar_1} + {}^{k}V_n^{a_2} + {}^{k+1}V_v^{a_2} \leq 3 \tag{6.9}$$

Podmínka 6.9 značně zvětšuje model instance CPF-no-head-on.

Do modelu se nezahrnují takové proměnné ${}^{k}V_v^a$, pro které se agent a nemůže dostat po k krocích do vrcholu v pohybem po nejkratší cestě z počátku, nebo pokud se po $m - k$ krocích nemůže dostat z vrcholu v do své cílové pozice.

6.2 IP active-edges

Definice 13 (Kódování IP active-edges)**.** *Pro daný makespan m, konečnou množinu kroků K, agentů A, vrcholů V a pro konečné množiny sousedů vrcholů N_v, kde $|K| = m, |A| \leq |V|$ a $\forall v : N_v \subseteq V \backslash \{v\}$ rozhodujeme, zda existuje řešení délky $\leq m$.*
Binární proměnné:

$${}^{k}_{v_i}E^a_{v_o} = 1 \Leftrightarrow \textit{v kroku } k \textit{ se agent } a \textit{ přesouvá z vrcholu } v_i \textit{ do vrcholu } v_o \tag{6.10}$$

Cíl:

$$Maximalizovat \sum_{k \in K \backslash \{0\}, a \in A, v \in V, n \in N_v} {}^{k}_{v}E^a_n \tag{6.11}$$

Podmínky:
Počáteční rozmístění agentů

$$\forall a \in A : {}^{0}_{v_S^a}E^a_{v_S^a} + \sum_{n \in N_{v_S^a}} {}^{0}_{v_S^a}E^a_n = 1 \tag{6.12}$$

Cílové rozmístění agentů

$$\forall a \in A : {}^{m}_{v_G^a}E^a_{v_G^a} + \sum_{n \in N_{v_G^a}} {}^{m}_{v_G^a}E^a_n = 1 \tag{6.13}$$

Každou hranou v každém kroku prochází maximálně jeden agent

$$\forall k \in K, v \in V : \sum_{a \in A} {}^{k}_{v}E^a_v \leq 1 \tag{6.14}$$

$$\forall k \in K, v \in V, n \in N_v : \sum_{a \in A} {}^{k}_{v}E^a_n \leq 1 \tag{6.15}$$

Každý agent využívá v každém kroku právě jednu hranu

$$\forall k \in K, a \in A : \sum_{v \in V} {}^{k}_{v}E^a_v + \sum_{v \in V, n \in N_v} {}^{k}_{v}E^a_n = 1 \tag{6.16}$$

Do každého vrcholu vstupuje v každém kroku nejvýše jeden agent

$$\forall k \in K, v \in V : \sum_{a \in A} {}^{k}_{v}E^{a}_{v} + \sum_{a \in A, n \in N_v} {}^{k}_{v}E^{a}_{n} = 1 \tag{6.17}$$

Z každého vrcholu vystupuje v každém kroku nejvýše jeden agent

$$\forall k \in K, v \in V : \sum_{a \in A} {}^{k}_{v}E^{a}_{v} + \sum_{a \in A, n \in N_v} {}^{k}_{n}E^{a}_{v} = 1 \tag{6.18}$$

Z každého vrcholu vystupuje v každém kroku stejný agent, jako do něj vstoupil, nebo jím žádný neprochází

$$\forall k \in K \backslash \{m\}, v \in V, a \in A : {}^{k+1}_{v}E^{a}_{v} + \sum_{n \in N_v} {}^{k+1}_{v}E^{a}_{n} - {}^{k}_{v}E^{a}_{v} - \sum_{n \in N_v} {}^{k}_{v}E^{a}_{n} = 0 \tag{6.19}$$

Požadavek na volné místo před agentem (pouze pro CPF-vacant-target)

$$\forall k \in K, v \in V : \sum_{a \in A} {}^{k}_{v}E^{a}_{v} + \sum_{a \in A, n \in N_v} ({}^{k}_{v}E^{a}_{n} + {}^{k}_{n}E^{a}_{v}) \leq 1 \tag{6.20}$$

Zákaz prohození agentů po hraně (pouze pro CPF-no-head-on)

$$\forall k \in K \backslash \{0\}, v \in V, n \in N_v : \sum_{a \in A} ({}^{k}_{v}E^{a}_{n} + {}^{k}_{n}E^{a}_{v}) \leq 1 \tag{6.21}$$

Do modelu se opět nezahrnují takové proměnné ${}^{k}_{v}E^{a}_{n}$, představující hrany, do kterých se nemůže agent v daném počtu kroků dostat ze svého počátku, nebo cíle. Díky tomu lze vynechat i podmínky 6.12 a 6.13, protože po jiných hranách, než těch co vedou z počáteční pozice se nemůže agent v prvním kroku pohnout, totéž platí pro cílové pozice.

7. Porovnání modelů a vyhodnocení

V této části popisujeme provedené testy. Nejprve zmíníme způsob, jakým jsme testy prováděli. Poté uvádíme postupně výsledky jednotlivých testů zobrazených v grafech společně s jejich zhodnocením.

7.1 Způsob provedení testů

K testování jsme použili počítač se čtyřjádrovým procesorem Intel Core 2 Quad Q9550 s frekvencí 2.83GHz pro každé jádro a se 4GB operační paměti. Na počítači byla nainstalována distribuce operačního systému Linux s názvem Gentoo.

K porovnání jsme si vybrali náhodně generované instance CPF na neorientovaných grafech ve tvaru mřížky o velikosti 8x8, jejíž vnitřní vrcholy jsou propojeny hranami se čtyřmi sousedními vrcholy. Testy jsme prováděli samostatně pro 0%, 10% a 20% náhodně rozmístěných překážek – ty byly reprezentovány vynecháním příslušných vrcholů v mřížce. Grafy tohoto tvaru jsme si vybrali proto, že se běžně používají k porovnání algoritmů pro řešení CPF ve článcích a zároveň jsou vhodné pro jednoduchou abstraktní reprezentaci 2D prostředí. Přestože zde ukazujeme testy na grafech ve tvaru mřížky, lze naše modely použít pro libovolné neorientované grafy.

Každý z testů jsme navíc spouštěli ve dvou variantách, jednou pro definici CPF-vacant-target a jednou pro definici CPF-no-head-on. Celkem jsme tedy provedli 6 samostatných porovnání.

Pro definici CPF-vacant-target jsme porovnali nové modely CSP all-diff, IP all-diff a IP active-edges s existujícími modely SAT Hmatching, SAT Hdifferential a SAT Hadvanced. Navíc jsme mezi ně zahrnuli i algoritmus OD+ID. U definice CPF-no-head-on jsme porovnávali pouze modely CSP all-diff, IP all-diff a IP active-edges. To proto, že ostatní nejsou pro tuto definici vytvořeny. Model CSP all-diff jsme v obou případech otestovali pro šest různých heuristik (viz kap. 5).

U jednotlivých porovnání je každý řešič testován na instancích s postupně se zvyšujícím počtem agentů. Pro každý počet agentů je provedeno deset různých testů, kde mají agenti náhodně určeny jejich počáteční i cílové pozice.

Všechny testy jsou omezeny časovým limitem jedné minuty, během které musí být nalezeno jejich řešení, jinak se považují za nevyřešené. Počet agentů se pro každý řešič zvyšuje tak dlouho, dokud se mu podaří vyřešit alespoň jednu instanci. Každý řešič byl navíc v každém porovnání zavolán pro daný počet robotů na stejných deseti instancích jako ostatní.

Pro každé porovnání jsme vytvořili samostatný bash skript. Jsou v něm uvedeny seedy pro generátor pseudonáhodných čísel, které se při testech používali ke generování náhodných instancí. Testy je tak možné provést opakovaně se stejnými instancemi jako v této práci.

7.2 Porovnávané hodnoty v testech

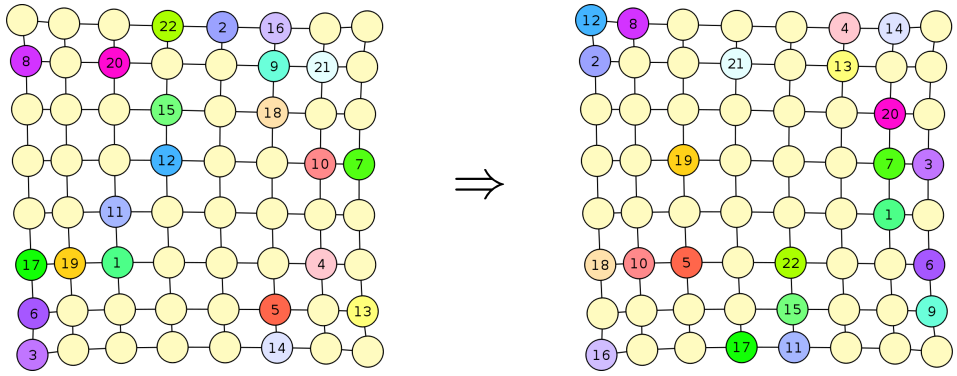
Pro každé porovnání jsme vytvořili grafy s následujícími údaji:

1. Počet vyřešených instancí z deseti
2. Průměrná doba trvání nalezení řešení
3. Průměrný procesorový čas využitý pro nalezení řešení
4. Celková doba strávená na řešení deseti instancí
5. Průměrný odhadovaný a skutečný makespan
6. Průměrná celková cesta agentů

V grafech 2, 3, 5, jsou uvedeny pro každý počet agentů průměrné naměřené hodnoty a směrodatné odchylky z deseti náhodných instancí. Přitom jsou v grafech 2 a 3 zobrazeny hodnoty jen tehdy, pokud bylo všech deset instancí vyřešeno. V grafu 4 jsou zahrnuty i časové limity jedné minuty pro nevyřešené instance. V grafu 6 jsou zahrnuty výsledky pro všechna nalezená řešení z deseti testů, i když se jich povedlo vyřešit méně než deset.

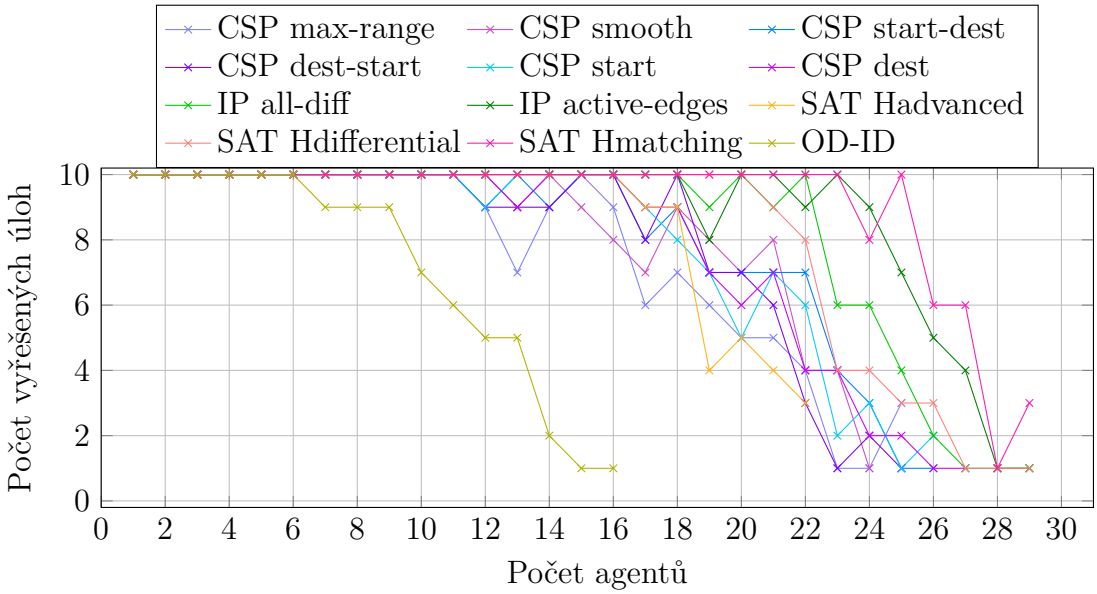
V grafech, kde je na svislé ose uvedena skutečná doba (grafy 2 a 4) je zobrazen jako horizontální červená čára časový limit. U grafu 2 je to jedna minuta, do které musí být instance vyřešena. V grafu 4 je to deset minut, což je součet maximálních časů pro jednotlivé úlohy z deseti.

7.3 Úloha CPF vacant-target bez překážek



Obrázek 7.1: Příklad řešené instance na mřížce 8x8 bez překážek

V této části popisujeme výsledky testů pro mřížku 8×8 bez překážek. Příklad takové instance je znázorněn na obrázku 7.1.



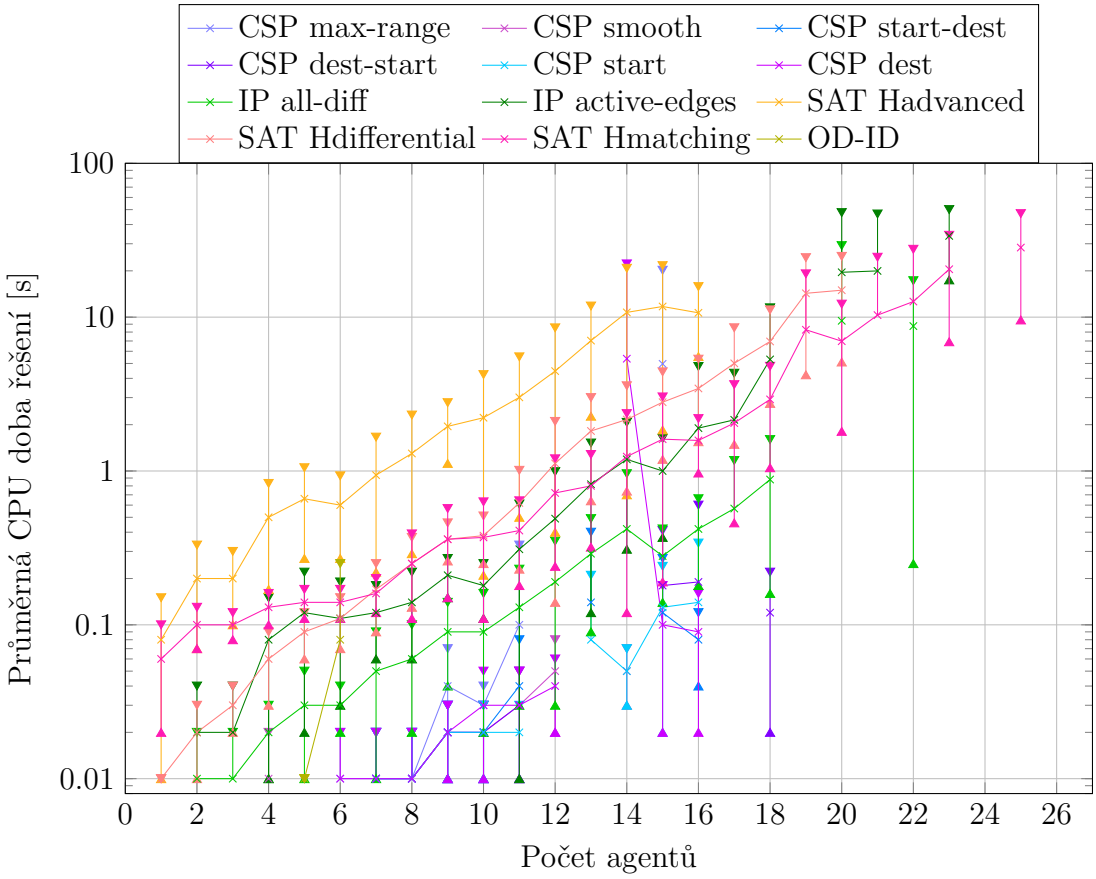
Obrázek 7.2: Množství vyřešených úloh do 60s

Podle grafu 7.2 se nejvíce instancí se podařilo vyřešit pro modely SAT Hmatching a IP active-edges, za nimi následovaly modely IP all-diff a SAT Hdifferential.

Model CSP all-diff vyřešil instancí podobné množství jako model SAT Hadvanced.

Algoritmus OD-ID v počtu vyřešených instancí výrazně zaostával za ostatními.

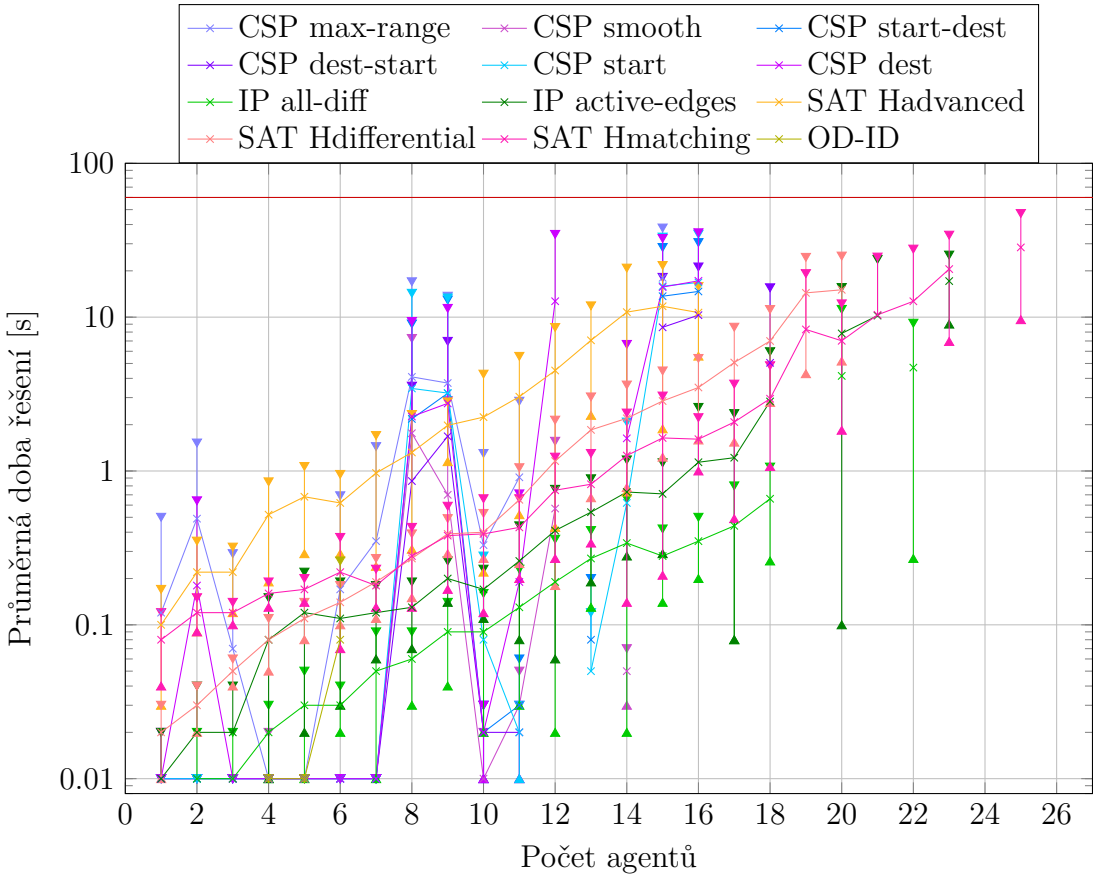
Z heuristik pro CSP modely dopadla asi nejlépe start-dest, nejhůře pak max-range, která nepoužívá informace o nejkratších cestách. Jinak dopadly heuristiky spíše vyrovnaně.



Obrázek 7.3: Průměrná CPU doba řešení 10 úloh

Nejméně procesorového času potřeboval CSP model. Nedokázal si však poradit s instancemi obsahujícími větší množství agentů. Dobré procesorové časy pro málo agentů měl i algoritmus OD+ID, tomu však dělalo problém řešit množství větší než pouhých 6 agentů.

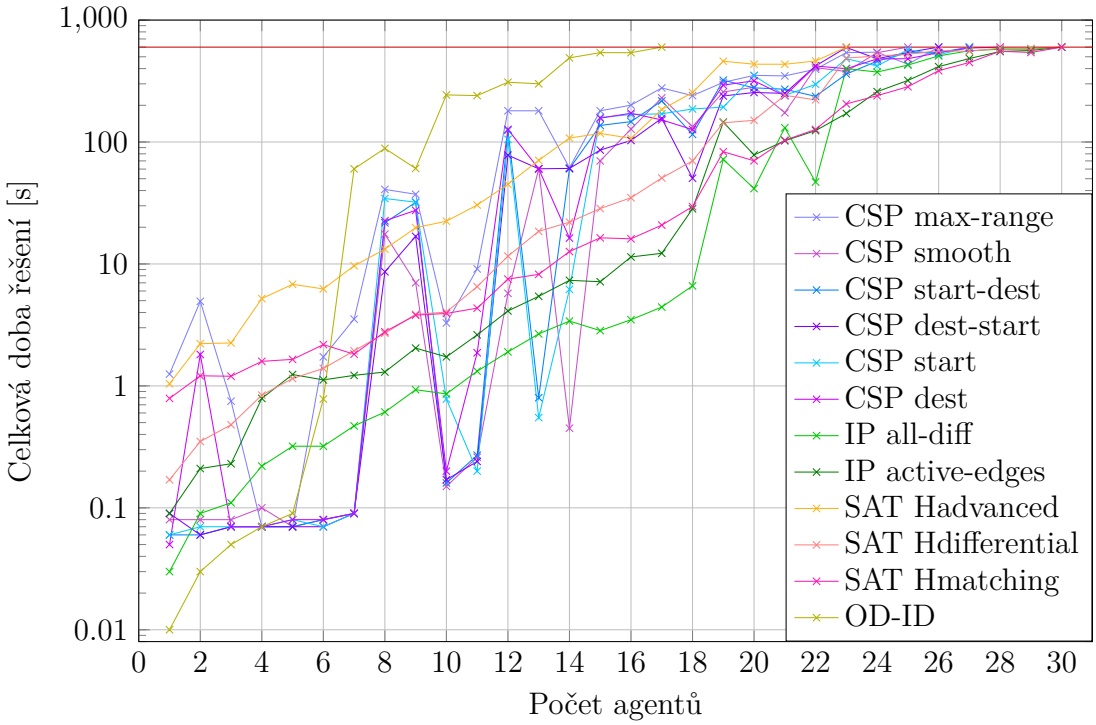
Poměrně dobré časy měly i IP modely, z nich lepší byl IP all-diff. Model IP active edges měl procesorový čas srovnatelný s modely SAT Hmatching a SAT Hdifferential. Nejvíce procesorového času využil model SAT Hadvanced. Dva ze SAT modelů dokázaly vyřešit nejvíce instancí.



Obrázek 7.4: Průměrná doba řešení 10 úloh

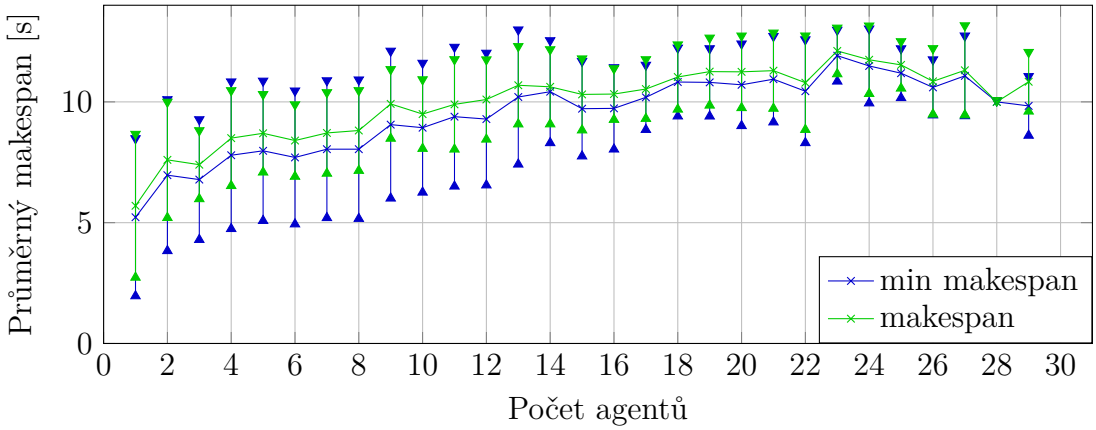
Zde je vidět velký propad v čase pro CSP model u všech heuristik pro 8 a 9 robotů, včetně modelu CSP max-range, který nevyužívá informace o nejkratších cestách. Přitom byl ale procesorový čas velmi nízký. CSP modely tedy při řešení zatěžovaly procesor minimálně. To jsme pozorovali i v programu Top, když jsme se v průběhu testu dívali na zatížení procesoru. Procesor přitom nezatěžovaly výrazněji žádné jiné spuštěné procesy. Navíc jsme zde vyhodnocovali postupně jeden model na všech instancích po druhém, nebylo to tedy pravděpodobně způsobeno náhlým zatížením počítače. Může to být způsobeno chováním knihovny Gecode na daných instancích CPF-vacant-target.

CSP mělo jinak dobré časy pro nízký počet agentů, stejně tak OD+ID. Při vyšším množství agentů měli oba problém instance vyřešit.
IP all-diff dokázal hledat řešení rychleji, než druhý IP model i než SAT modely. SAT modely SAT Hmatching a SAT Hdifferential dokázali řešit i vysoký instancí, přitom vykazovaly stabilní zpomalování.



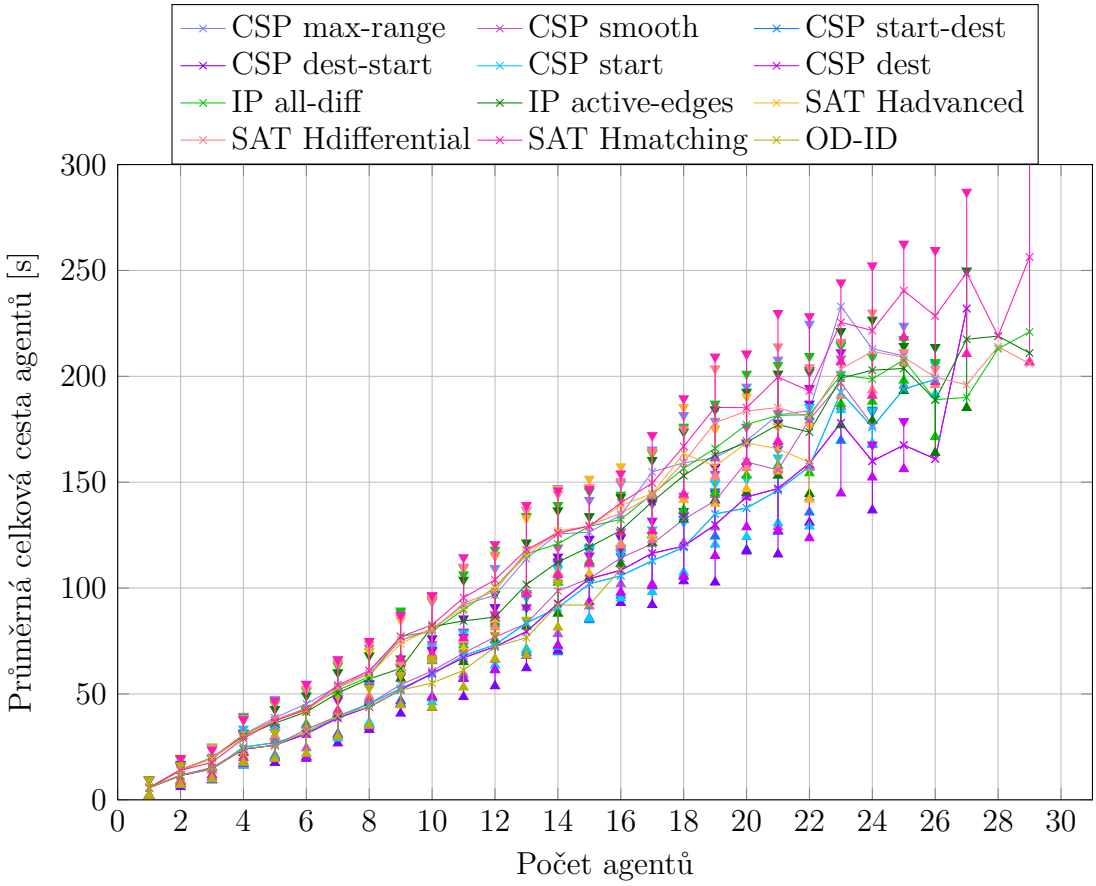
Obrázek 7.5: Celková doba řešení 10 úloh

Zde je vidět především nestabilita doby řešení u CSP modelů. Zároveň je vidět, že se i algoritmu OD+ID dařilo řešit alespoň některé instance s více agenty.



Obrázek 7.6: Odhadovaný a skutečný makespan pro 10 úloh

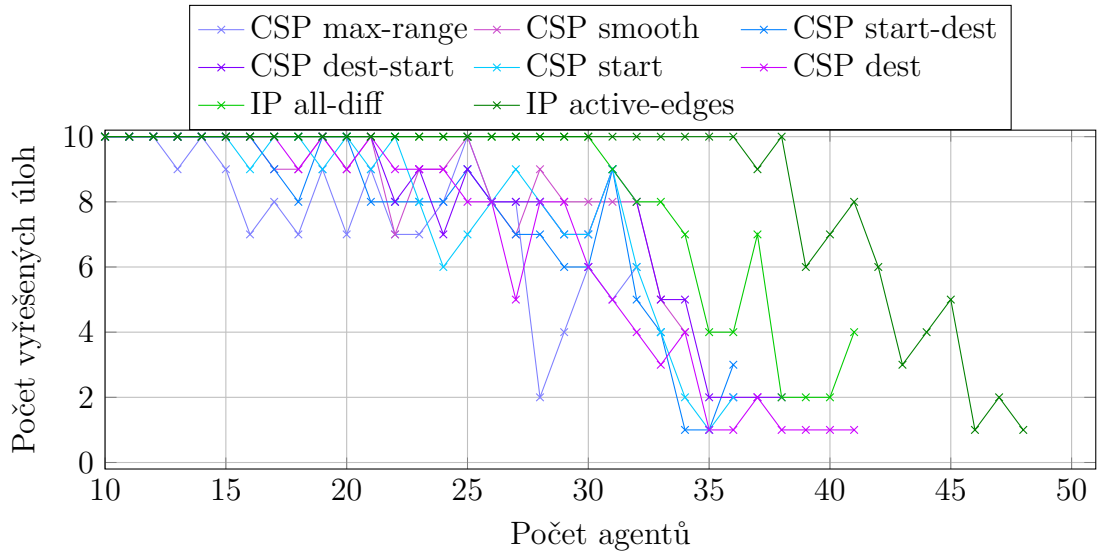
Z grafu je patrné, že se odhadovaný a skutečný makespan v některých případech mírně liší.



Obrázek 7.7: Průměrná celková cesta agentů pro 10 úloh

Na grafu 7.7 je vidět, že CSP modely využívající informaci o nejkratších cestách, stejně jako algoritmus OD+ID, nacházejí průměrně řešení s nižším celkovým počtem pohybů agentů, zatímco model CSP max-range nachází řešení s délkou cest srovnatelnou s ostatními modely.

7.4 Úloha CPF no-head-on bez překážek



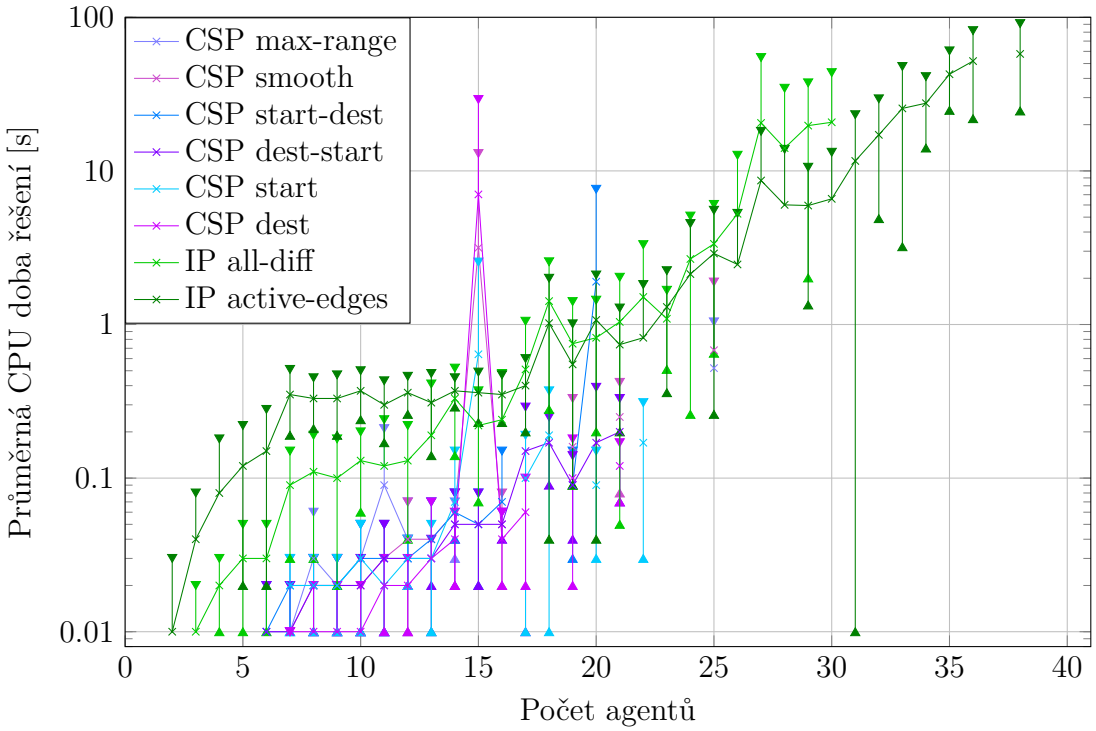
Obrázek 7.8: Množství vyřešených úloh do 60s

IP active-edges má výrazně nejvíce vyřešených instancí.

Následuje IP all-diff a pak ostatní CSP modely.

IP all-diff obsahuje pro modely dle definice CPF-no-head-on oproti definici CPF-vacant-target v kódování velké množství dodatečných podmínek pro zabránění kolizím po stejné hraně. To se patrně projevilo i na jeho slabším výsledku pro tuto definici.

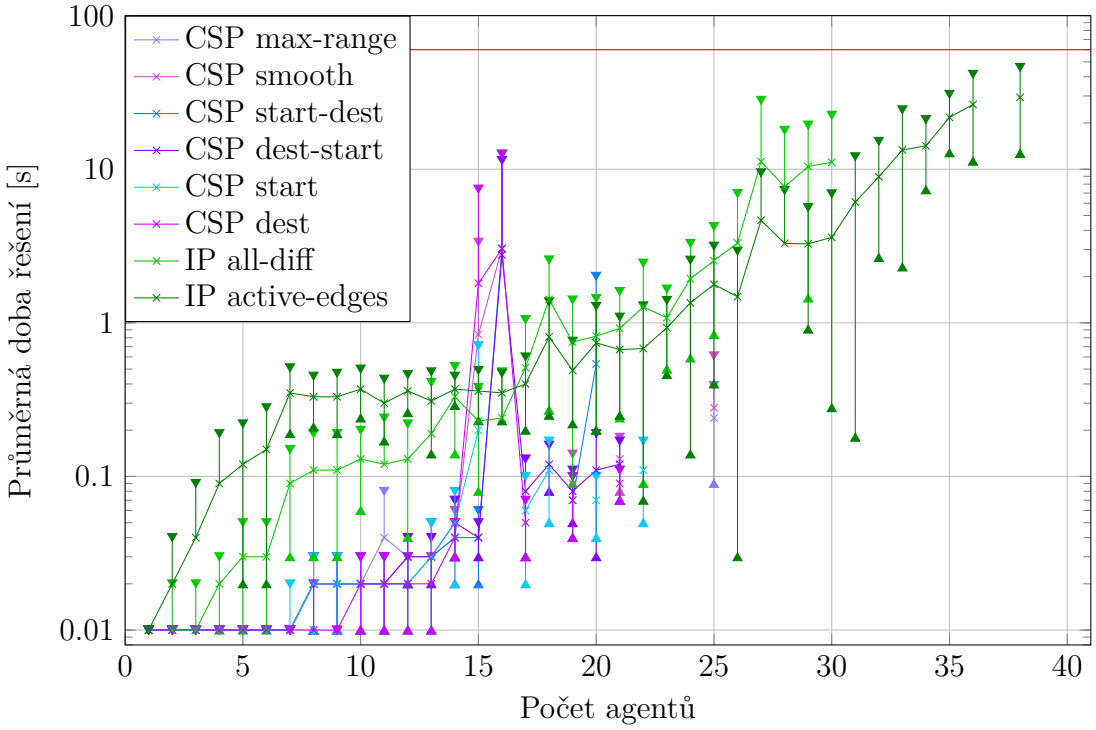
Heuristiky pro CSP model mají vzájemně přibližně podobné výsledky a za IP modely zaostávají.



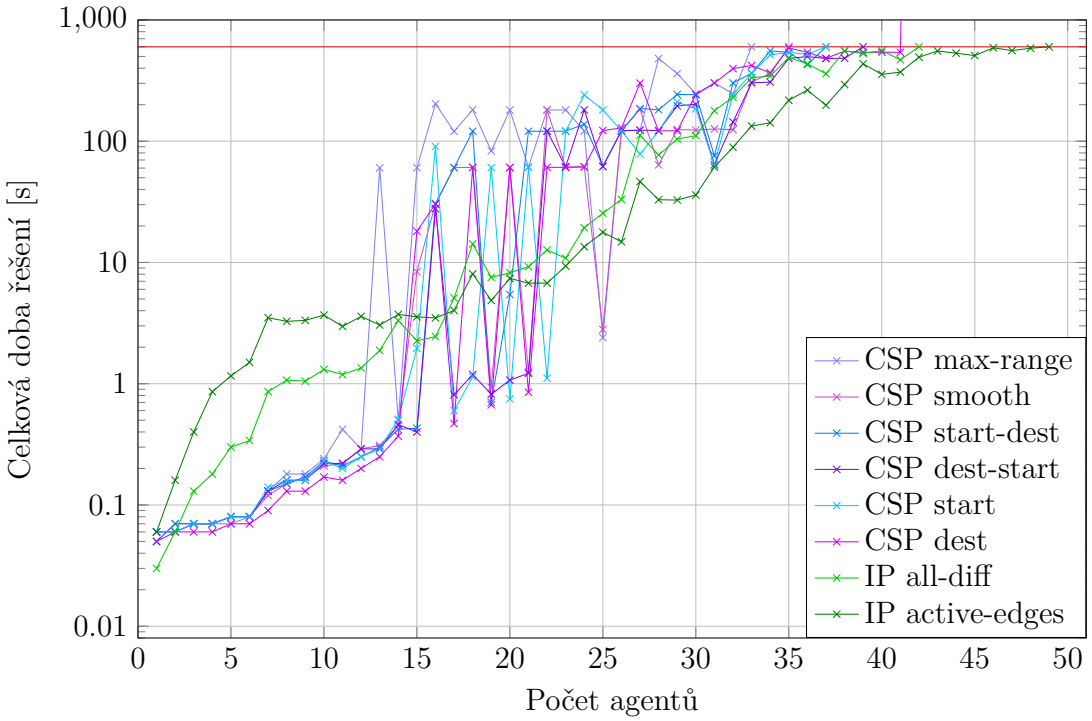
Obrázek 7.9: Průměrná CPU doba řešení 10 úloh

Nejlepší časy zde vykazuje model CSP all-diff, ale nezvládne instance s více agenty. Navíc se ukazuje u několika heuristik skokové zhoršení u některých instancí i u procesorového času, zatímco u skutečného času se projevilo u všech.

Pro nižší počet agentů má průměrně lepší časy IP all-diff, než IP active-edges, ale IP active-edges zvládá vyřešit instance s více agenty.

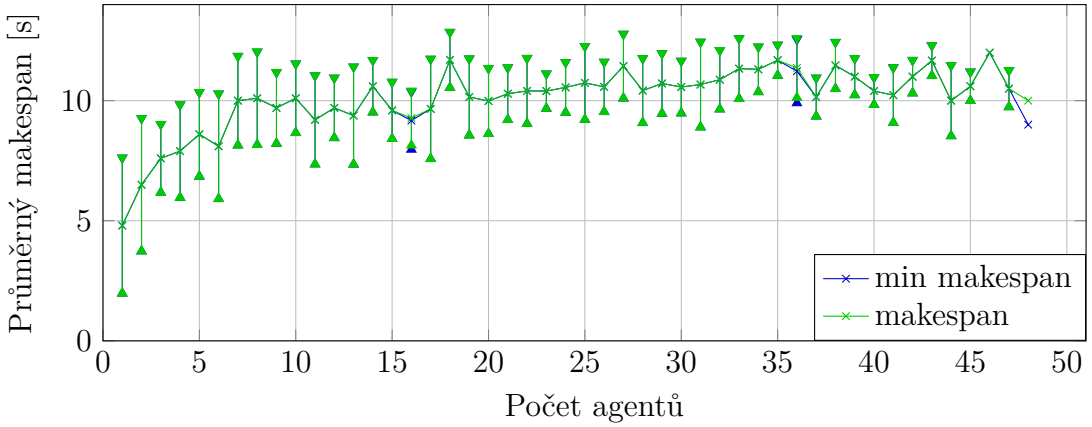


Obrázek 7.10: Průměrná doba řešení 10 úloh



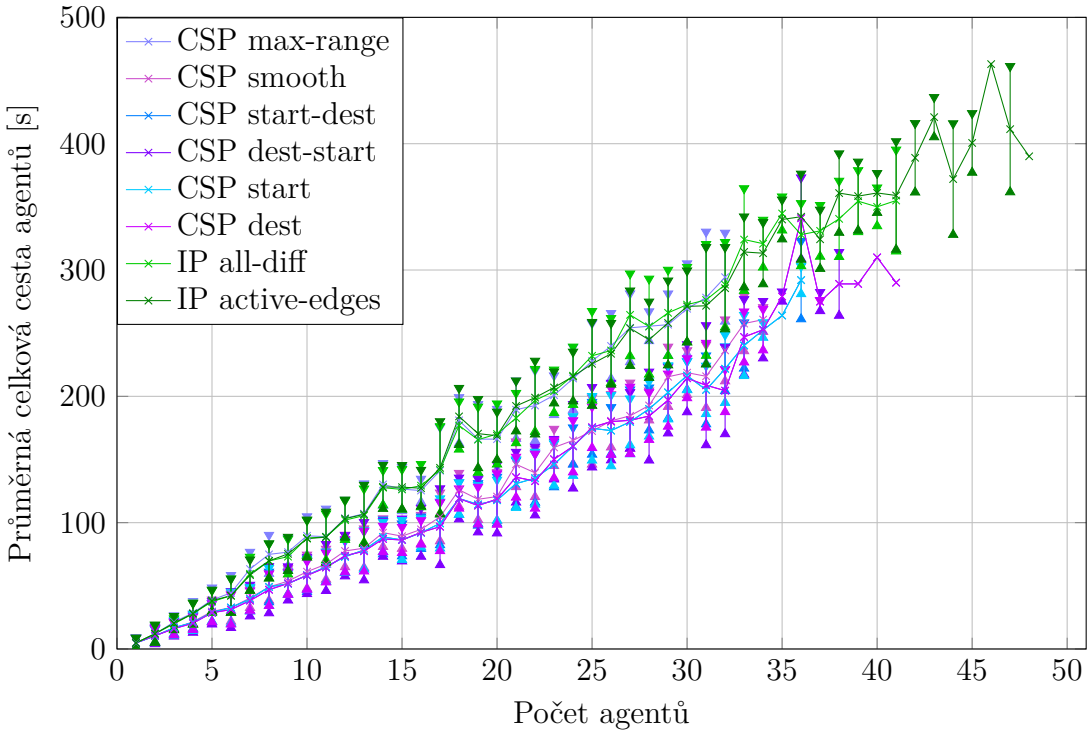
Obrázek 7.11: Celková doba řešení 10 úloh

Na grafu jsou vidět nestabilní výsledky CSP modelu oproti IP modelům.



Obrázek 7.12: Odhadovaný a skutečný makespan pro 10 úloh

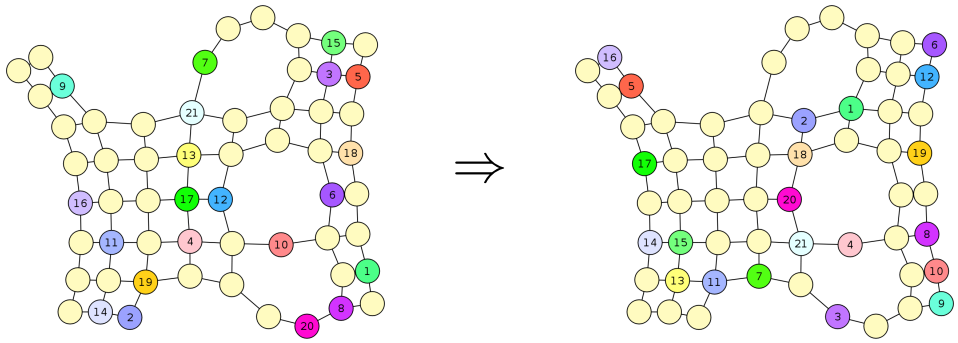
Na tomto grafu je patrné, že pro úlohu CPF-no-head-on vyšli odhadované velikosti makespan při počtu agentů do 75% většinou správné.



Obrázek 7.13: Průměrná celková cesta agentů pro 10 úloh

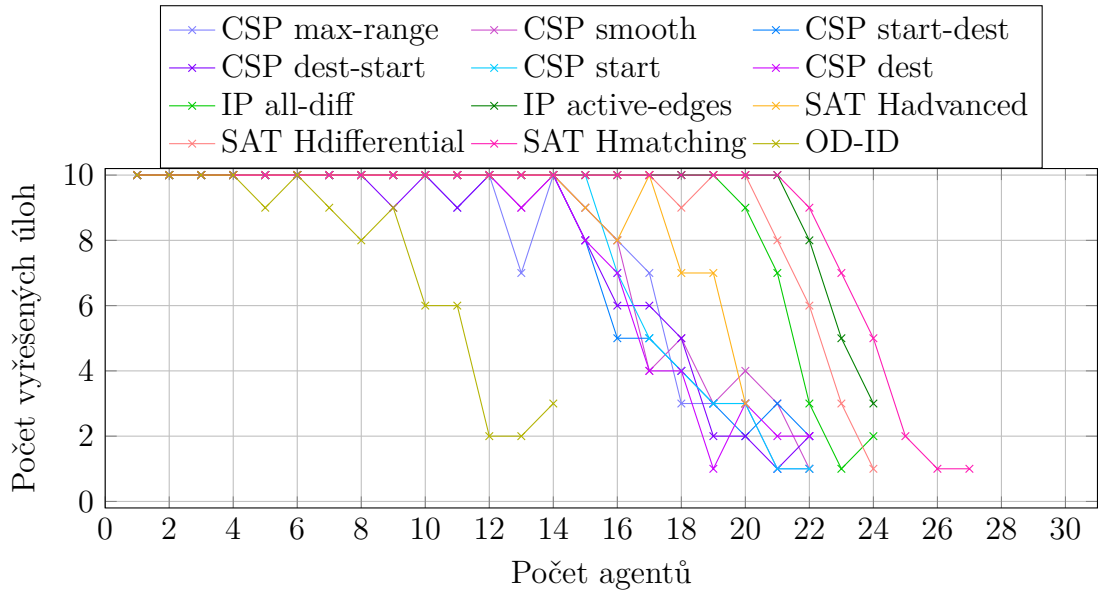
Zde se ukazuje, že CSP model s heuristikami využívajícími znalosti nejkratší cesty mívá nalezená řešení s menším celkovým počtem pohybů agentů.

7.5 Úloha CPF vacant-target s 10% překážek



Obrázek 7.14: Příklad řešené instance na mřížce 8x8 s 10% překážek

V následujících dvou porovnáních ukazujeme výsledky na mřížce 8x8 s 10% překážek.

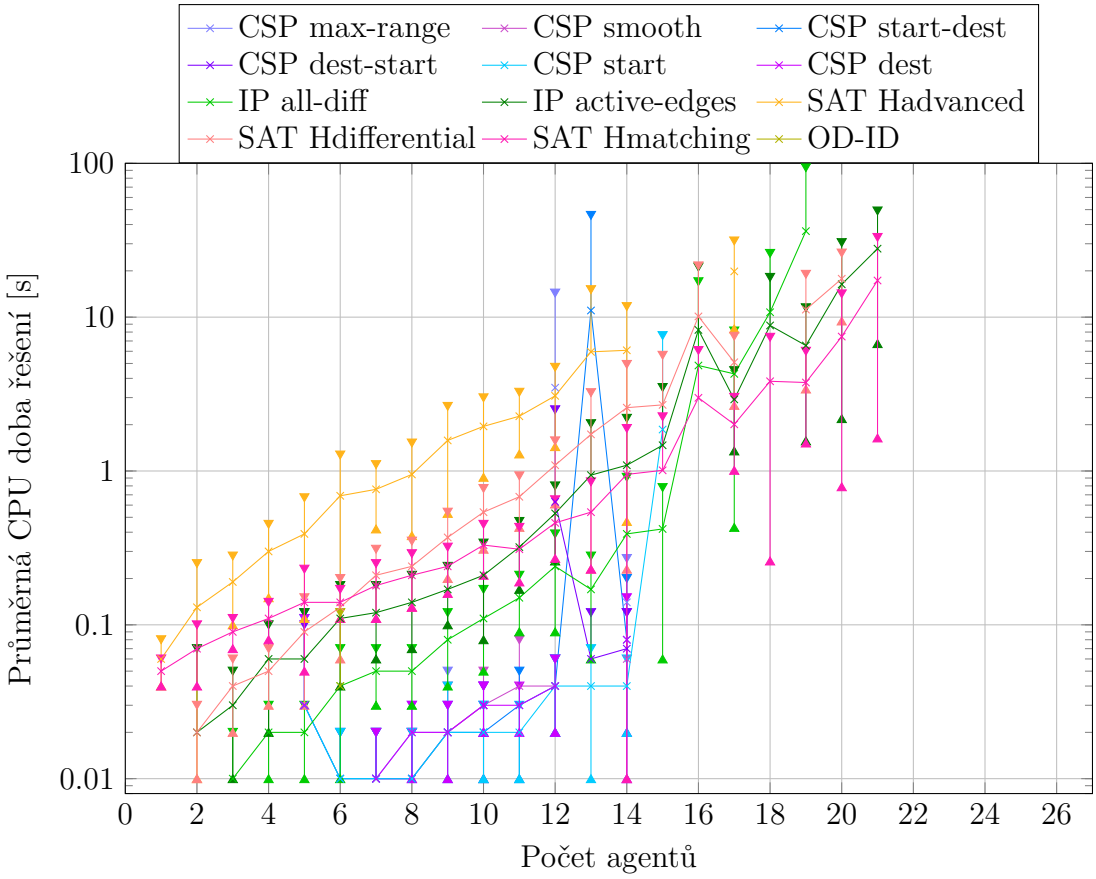


Obrázek 7.15: Množství vyřešených úloh do 60s

Nejvíce instancí vyřešil model SAT Hmatching, následovaný modely IP all-diff, SAT Hdifferential a IP active edges.

S odstupem následují modely SAT Hadvanced a CSP all-diff pro různé heuristiky.

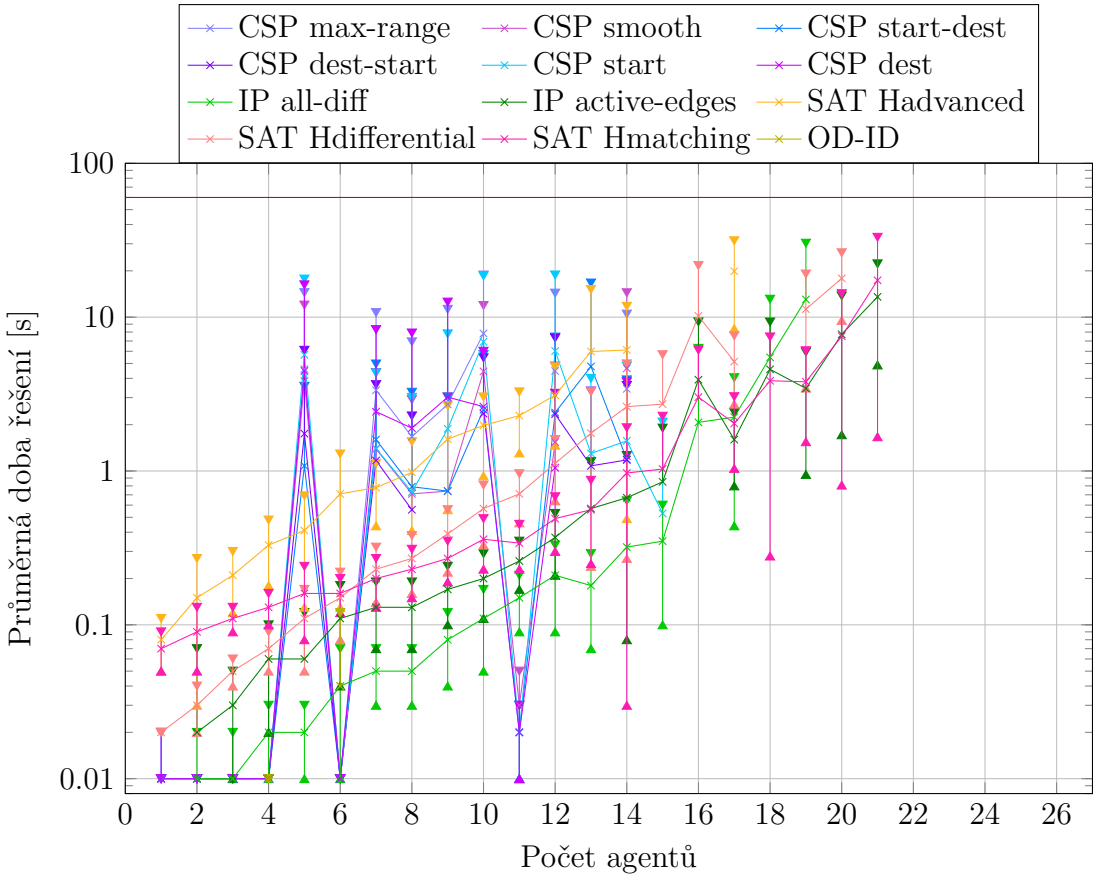
Nejméně instancí vyřešil model OD-ID.



Obrázek 7.16: Průměrná CPU doba řešení 10 úloh

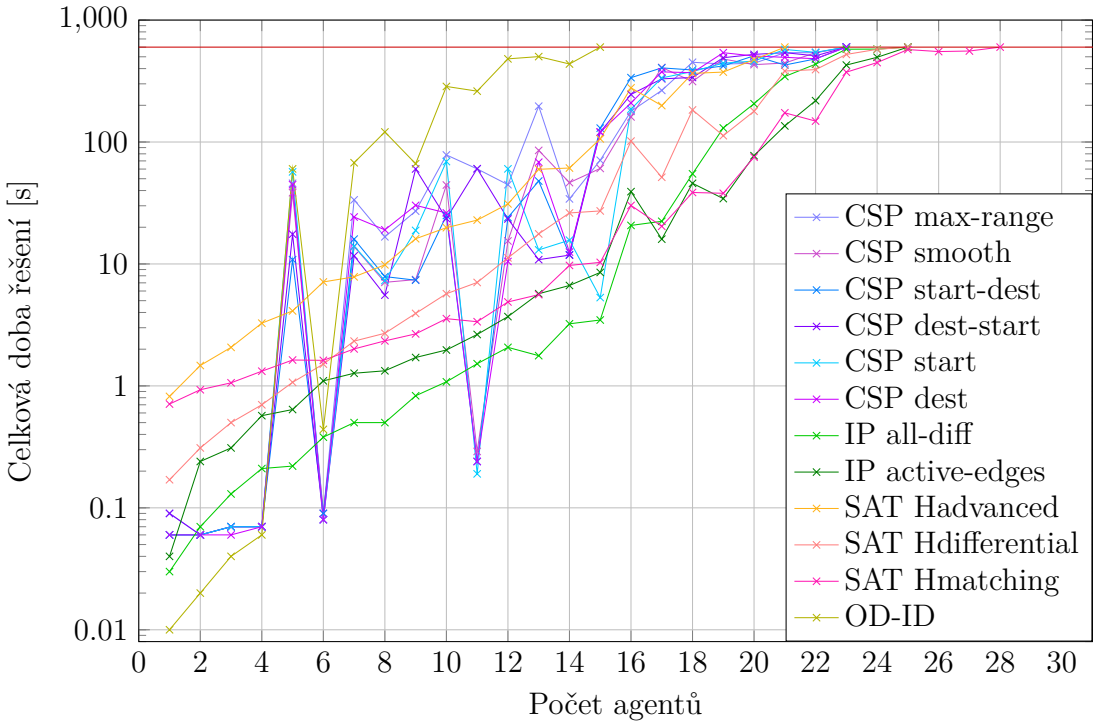
Nejlepší časy má model pro CSP. Pro dostatečně nízký počet agentů je model IP all-diff rychlejší, než model IP active-edges a SAT modely, ale s rostoucím počtem agentů ale tento model oproti nim naopak zaostává.

Nejhorší časy zde měl model SAT Hadvanced.



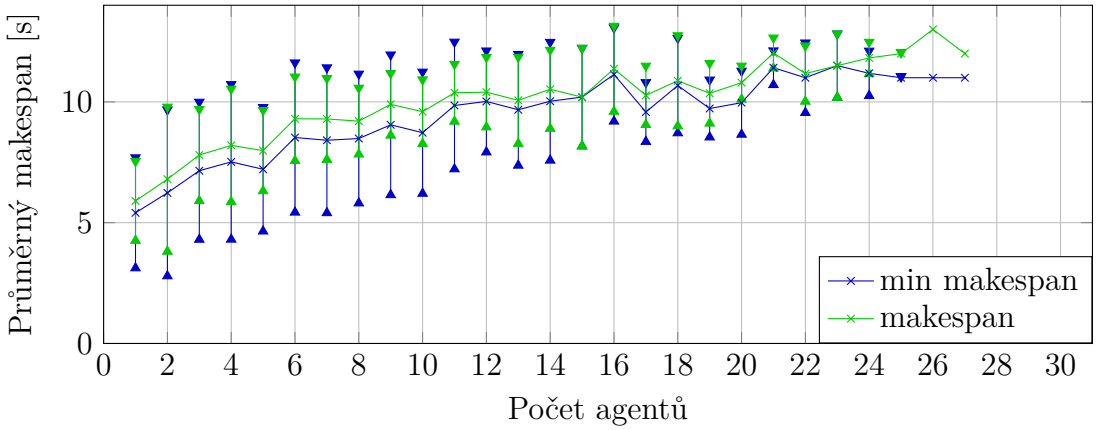
Obrázek 7.17: Průměrná doba řešení 10 úloh

Výsledky na grafu 7.17 přibližně odpovídají výsledkům zobrazeným u grafu s procesorovým časem kromě CSP modelu. Ten opět vykazuje velmi nestabilní výsledky, kdy řešení některých instancí zabere dlouhou skutečnou dobu, během které se drží vytížení procesoru na nízké úrovni. Zdá se, že knihovna Gecode nedokáže tyto podmínky řešit s efektivním využitím procesoru.



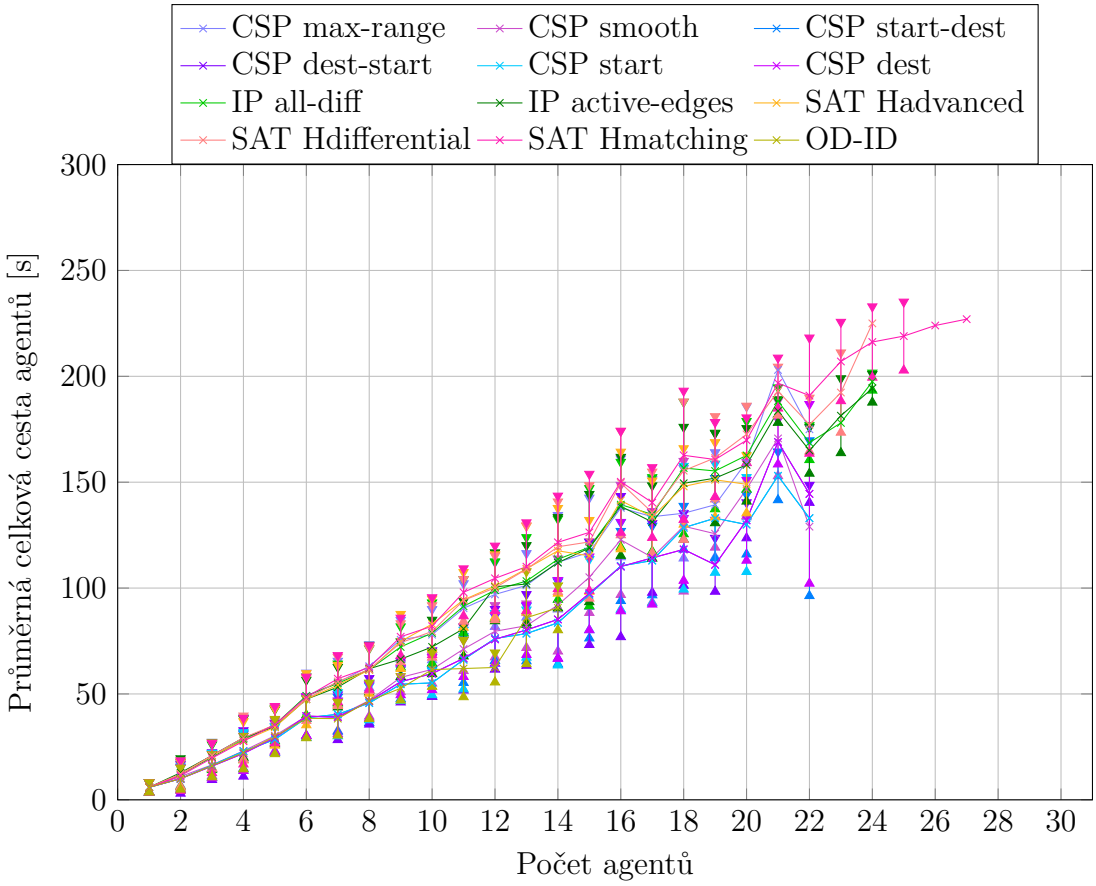
Obrázek 7.18: Celková doba řešení 10 úloh

V tomto grafu jsou oproti předešlému vidět i výsledky algoritmu OD+ID, u kterého s rostoucím počtem agentů brzy přibývá počet nevyřešených instancí.



Obrázek 7.19: Odhadovaný a skutečný makespan pro 10 úloh

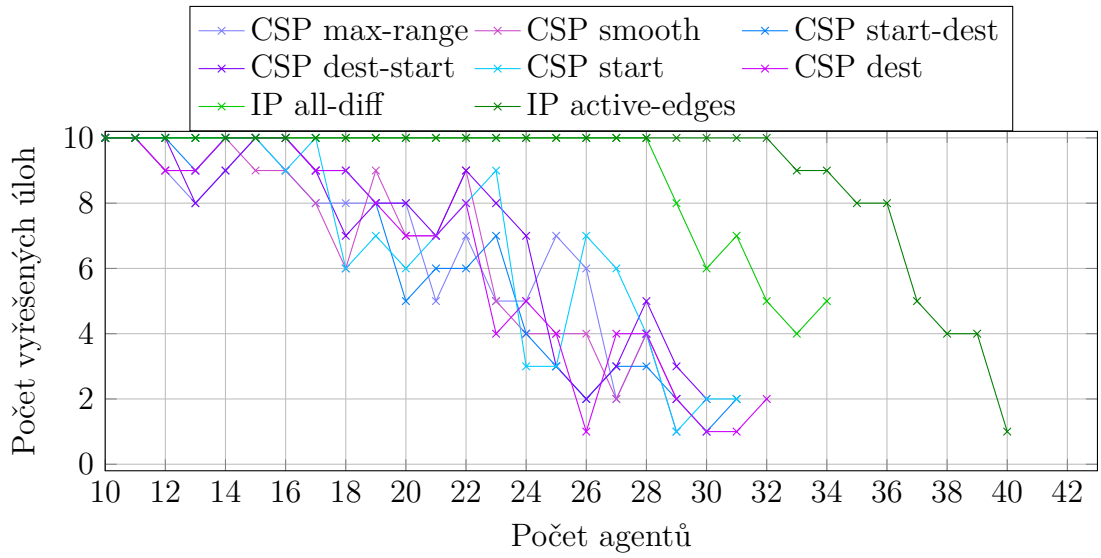
Je vidět, že se průměrný odhadovaný makespan a průměrný skutečný makespan při definici CPF-vacant-target mírně liší.



Obrázek 7.20: Průměrná celková cesta agentů pro 10 úloh

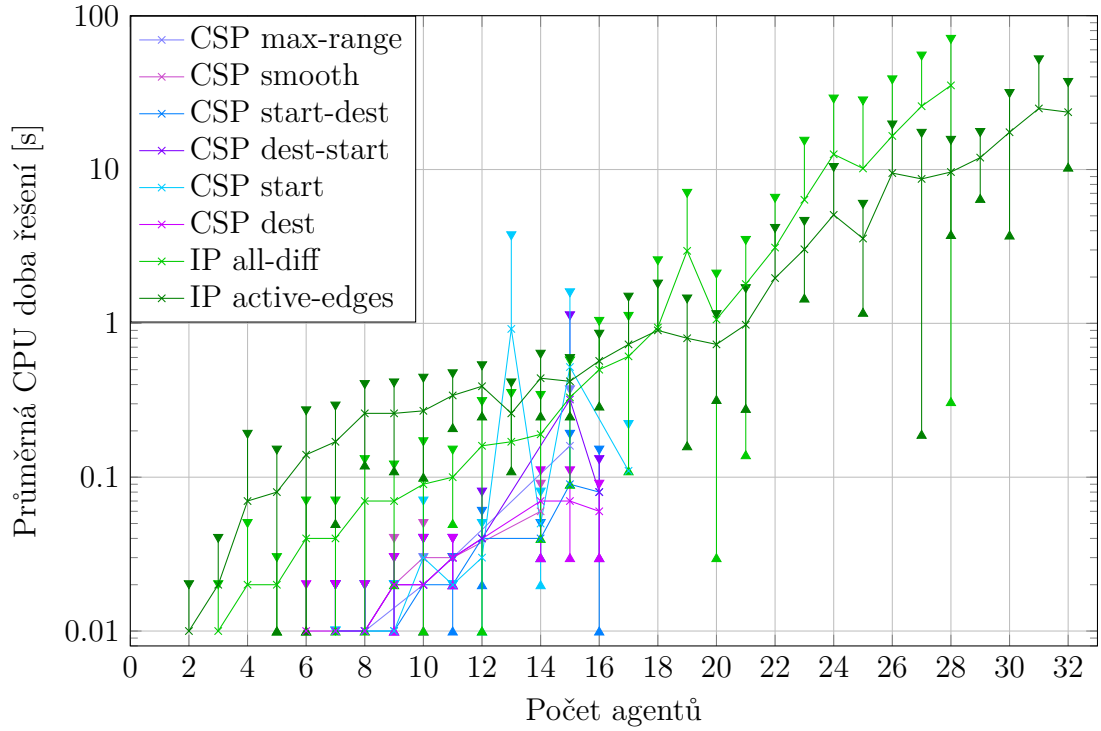
Graf 7.20 potvrzuje, že algoritmus OD+ID a model CSP all-diff s heuristikami založenými na znalosti nejkratších cest nacházejí průměrně řešení s nižším počtem přesunů agentů oproti ostatním modelům.

7.6 Úloha CPF no-head-on s 10% překážek



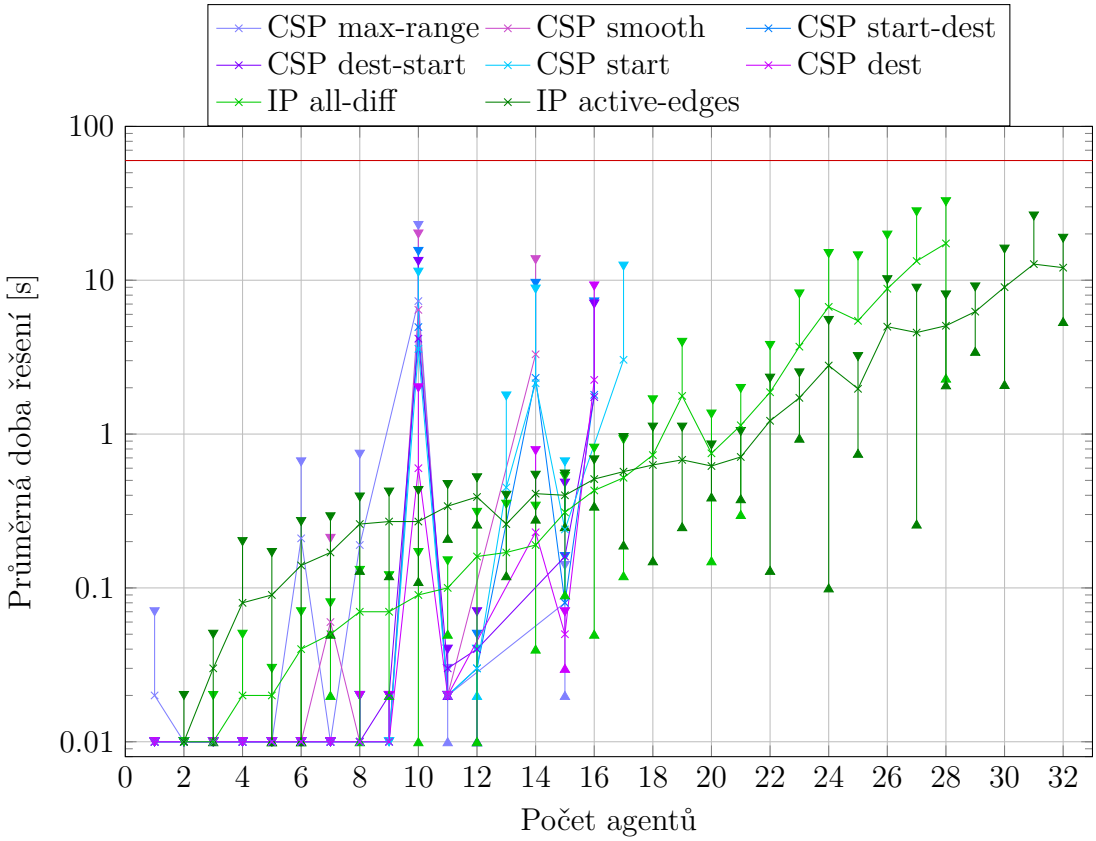
Obrázek 7.21: Množství vyřešených úloh do 60s

Výrazně nejvíce instancí vyřešil model IP active-edges, podstatně méně pak model IP all-diff. CSP model pak vyřešil ještě výrazně méně instancí, než oba IP modely.



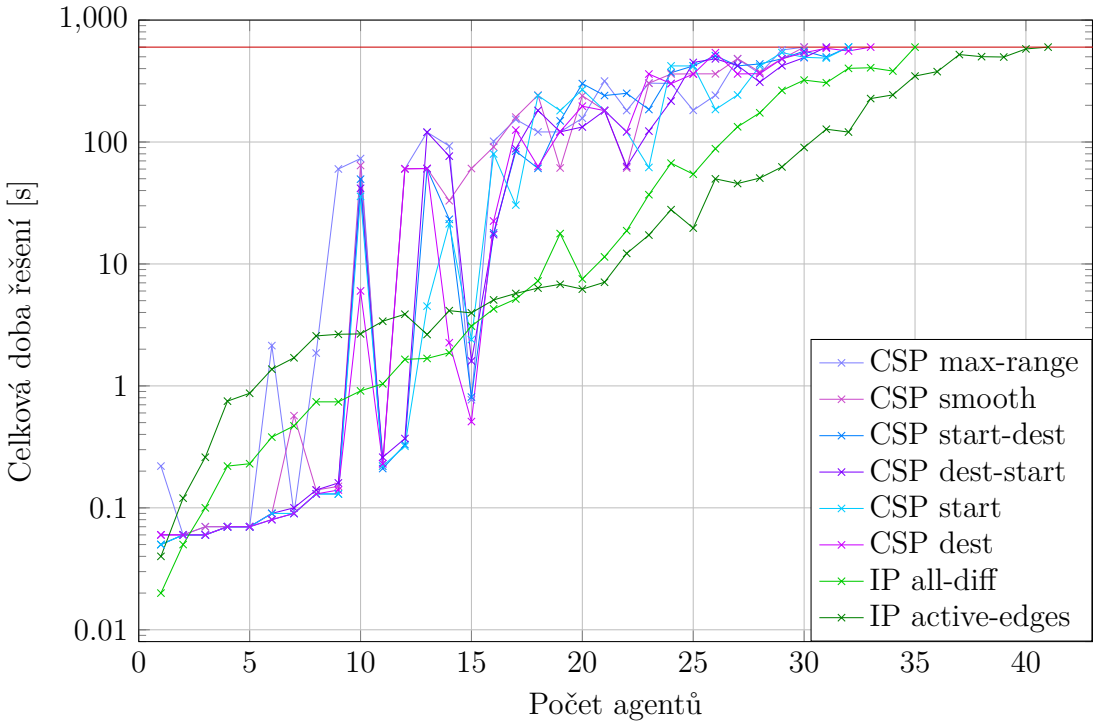
Obrázek 7.22: Průměrná CPU doba řešení 10 úloh

Model CSP all-diff nacházel pro instance s nižší hustotou agentů řešení v podstatně kratším čase, než oba IP modely. Oproti nim se ovšem nedokázal vypořádat s vysokou hustotou agentů.



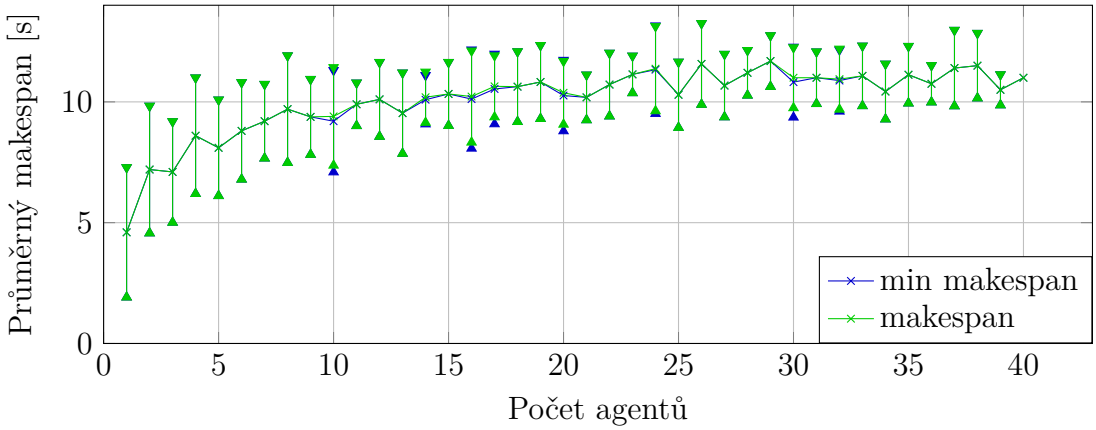
Obrázek 7.23: Průměrná doba řešení 10 úloh

Ačkoliv pro nízkou hustotu agentů vracel CSP model řešení podstatně rychleji, než IP modely, i zde nastala situace, kdy některé instance byly řešeny CSP modely výrazně dlouhou dobu, přitom s málo vytíženým procesorem. Pro vysoké množství agentů pak mají IP modely výrazně lepší čas. Přitom model IP all-diff má lepší časy pro nižší hustotu agentů oproti druhému IP modelu, pro vysokou hustotu agentů je tomu naopak.



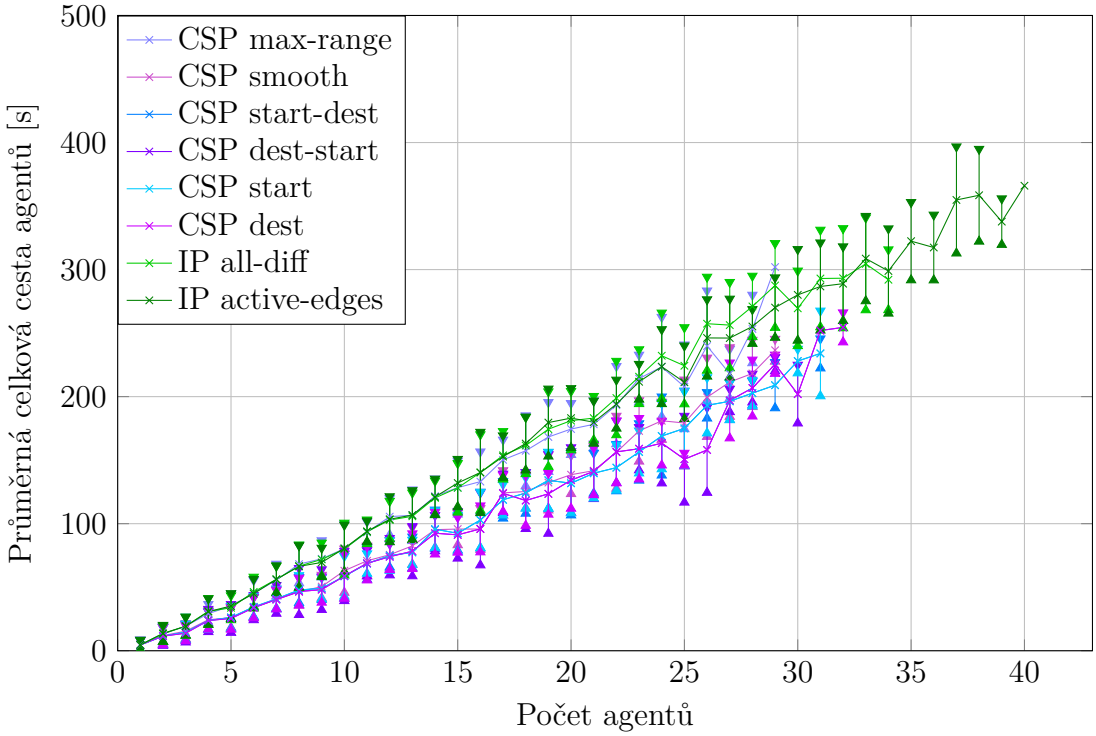
Obrázek 7.24: Celková doba řešení 10 úloh

Zde je vidět, že CSP model dokáže pro některé instance nalézt řešení, i když mají poměrně vysoké množství agentů.



Obrázek 7.25: Odhadovaný a skutečný makespan pro 10 úloh

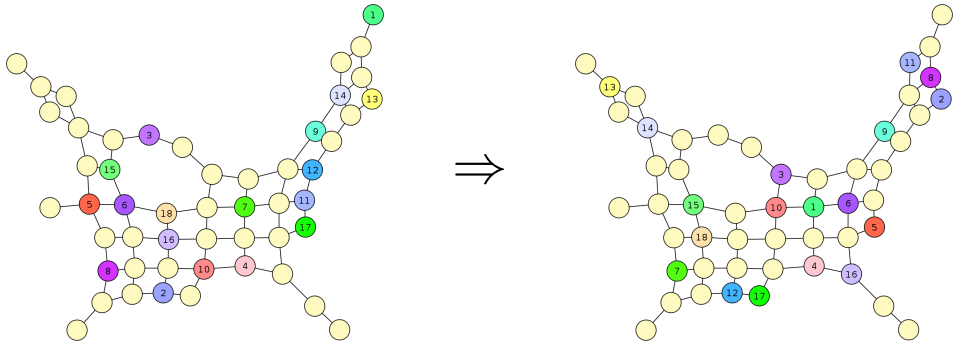
Dolními odhady se nám pro tento problém podařilo téměř ve všech případech trefit skutečný minimální makespan.



Obrázek 7.26: Průměrná celková cesta agentů pro 10 úloh

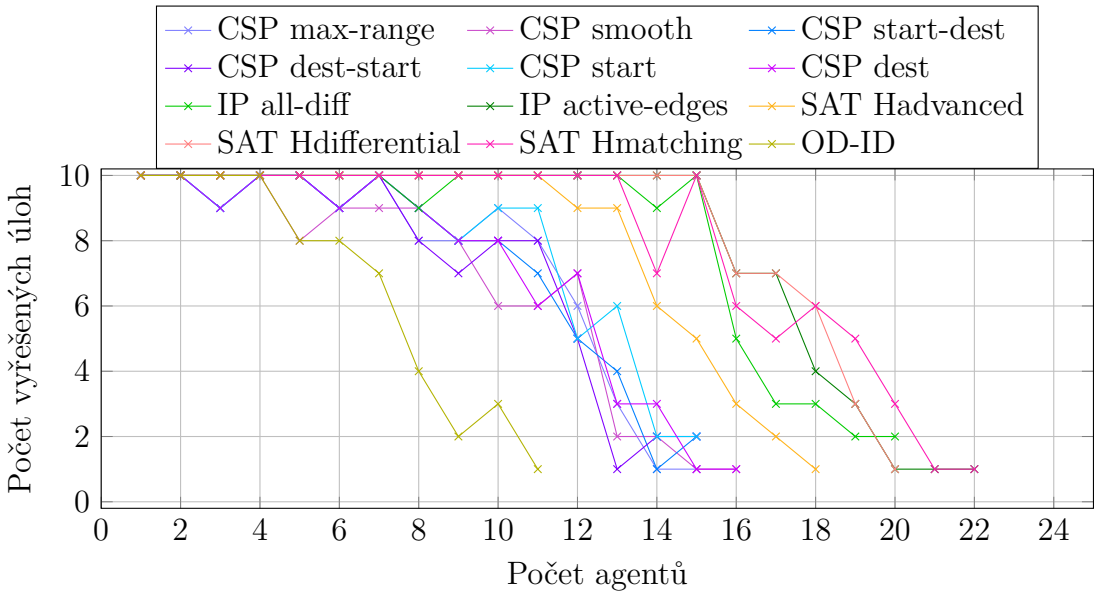
CSP model volaný s heuristikami využívající informaci o nejkratších cestách v grafu nacházel průměrně kratší řešení, než ostatní.

7.7 Úloha CPF vacant-target s 20% překážek



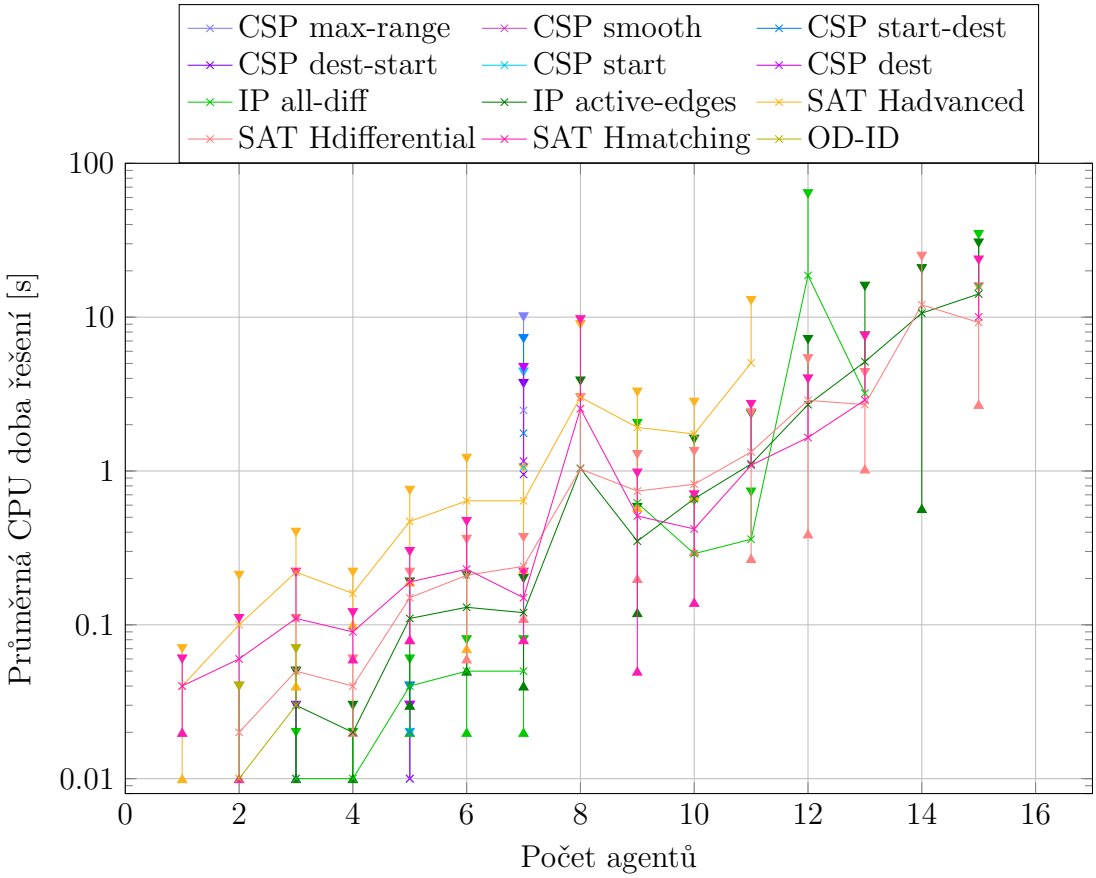
Obrázek 7.27: Příklad řešené instance na mřížce 8x8 s 20% překážek

Poslední dvě porovnání provádíme na mřížce o velikosti 8x8, která má 20% překážek.



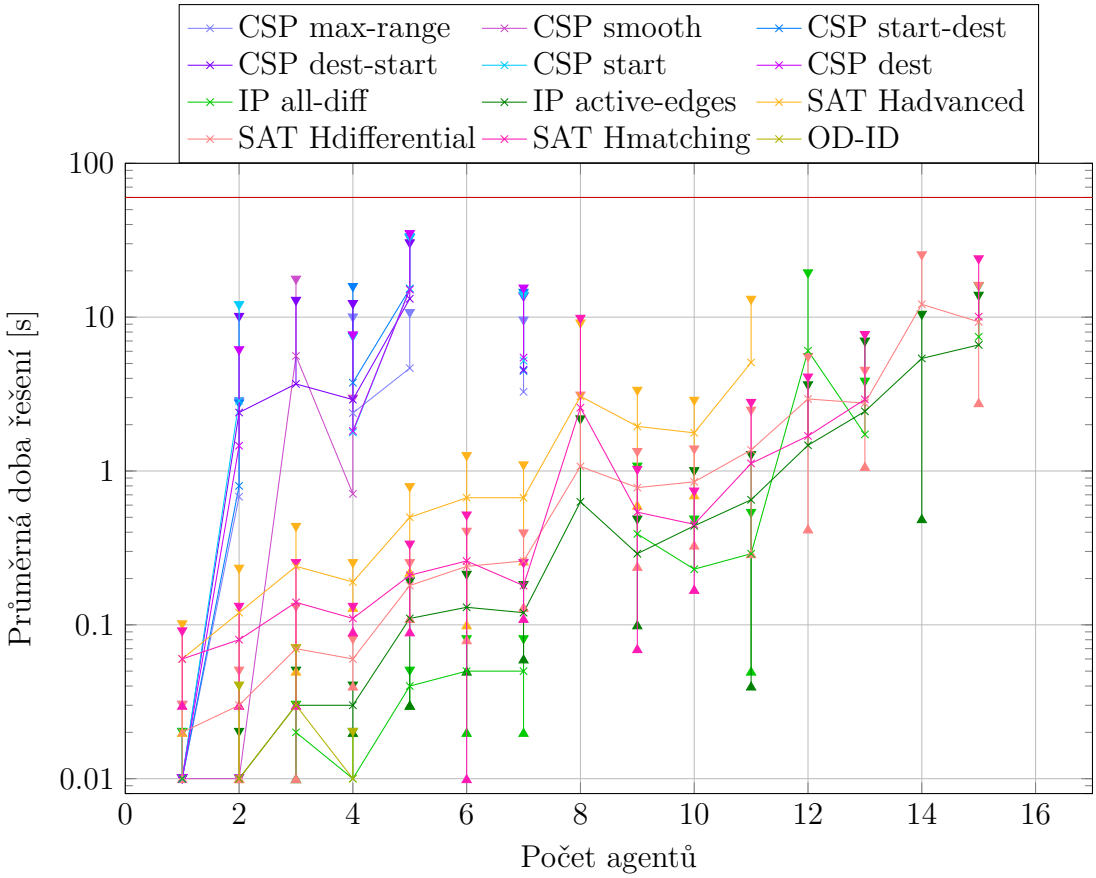
Obrázek 7.28: Množství vyřešených úloh do 60s

Nejvíce instancí se podařilo vyřešit modelům SAT Hmatching, SAT Hdifferential a IP active-edges, za nimi následoval model IP all-diff a SAT Hadvanced. Výrazně méně instancí se podařilo vyřešit CSP modelu, který měl problém vyřešit i jednu instanci se třemi roboty. Nejméně instancí se podařilo vyřešit algoritmu OD+ID. Je také vidět, že model IP all-diff s jinak poměrně stabilními výsledky měl problém vyřešit jednu instanci s osmi agenty.



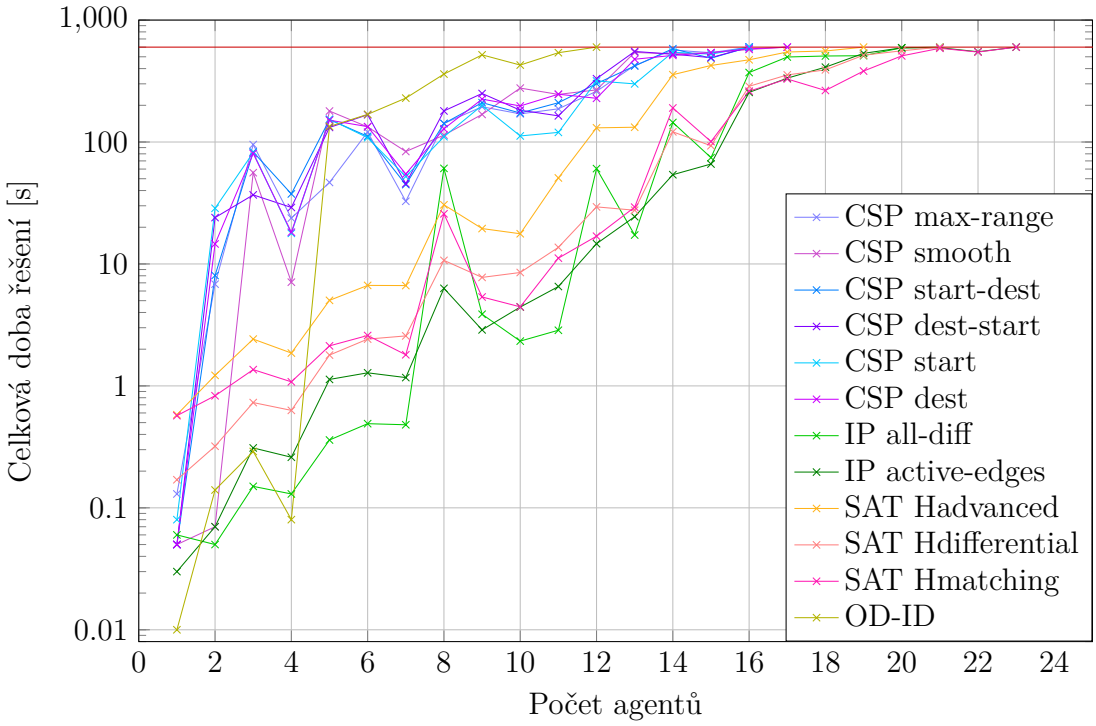
Obrázek 7.29: Průměrná CPU doba řešení 10 úloh

Ačkoliv to není v grafu vidět, podařilo se CSP modelům pro instance s pěti a méně agenty nalézt řešení většinou během několika milisekund procesorového času, pro vyšší počet agentů se doba výrazně zhoršila. Modely SAT Hdifferential, SAT Hmatching a IP active-edges mají poměrně vyrovnané výsledky. Model IP all-diff v porovnání se SAT modely a druhým IP modelem zvládl nalézt řešení rychleji pro nižší počet agentů.



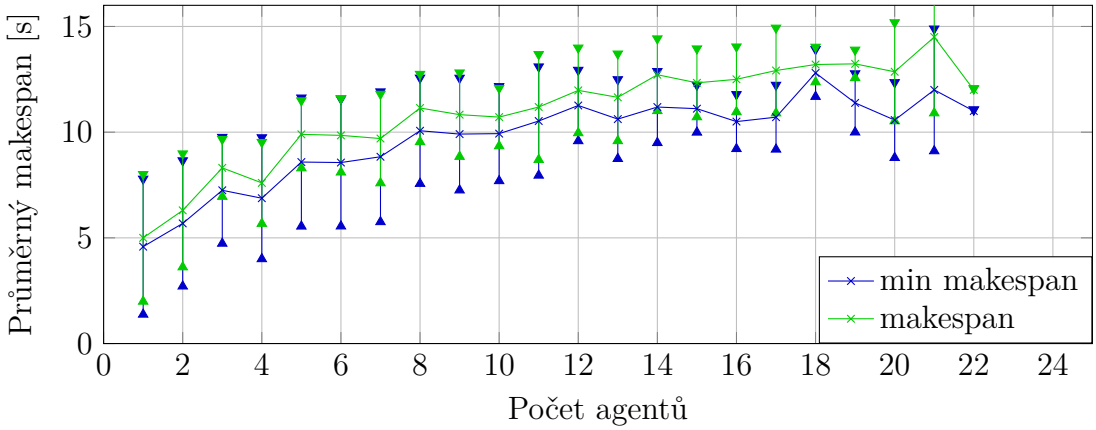
Obrázek 7.30: Průměrná doba řešení 10 úloh

CSP modely opět potřebovaly na vyřešení průměrně výrazně vyšší dobu, než byl využitý procesorový čas. U ostatních modelů výsledné časy řešení přibližně odpovídají využitému procesorovému času.



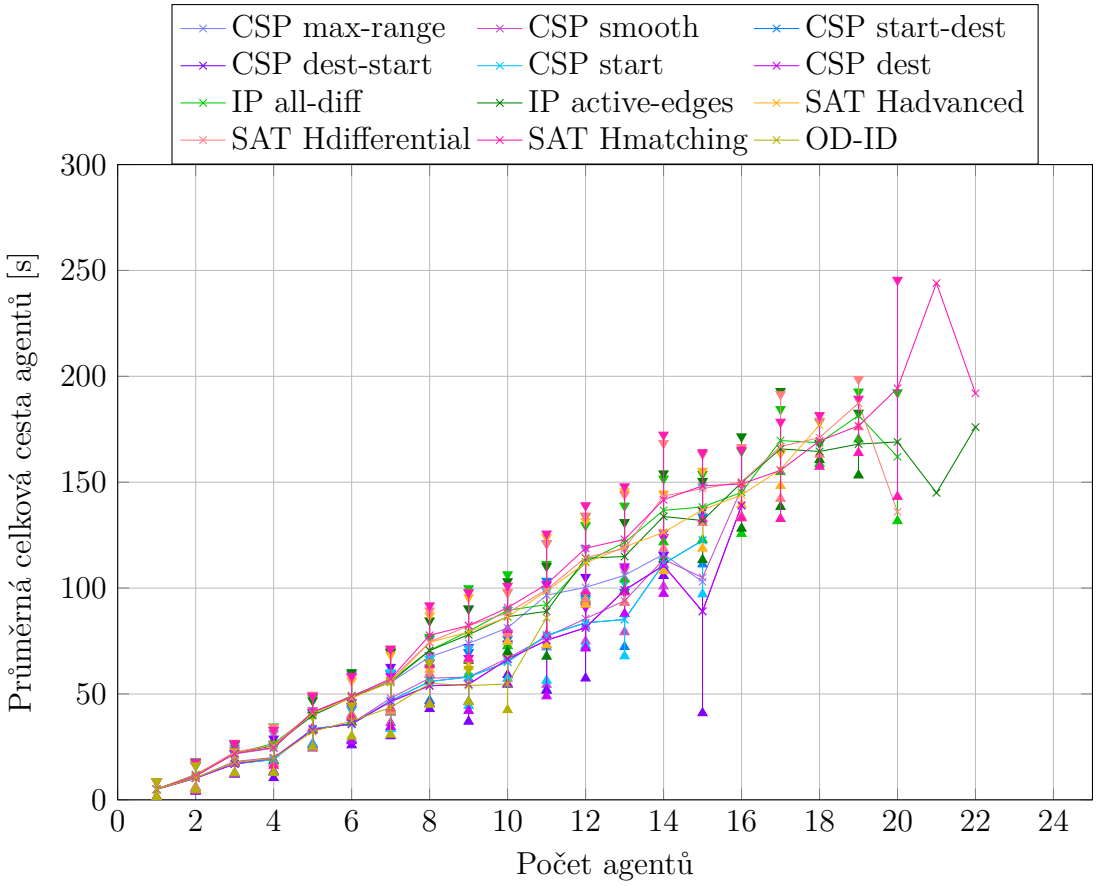
Obrázek 7.31: Celková doba řešení 10 úloh

Na tomto grafu je mimo jiné vidět, že si algoritmus OD-ID poradil s některými instancemi až do zaplnění jedenácti agenty, počet vyřešených instancí byl však nízký. CSP model dopadl o něco lépe. V porovnání IP a SAT modelů se ukazují IP modely jako rychlejší pro nižší počet agentů, jinak jsou přibližně srovnatelné.



Obrázek 7.32: Odhadovaný a skutečný makespan pro 10 úloh

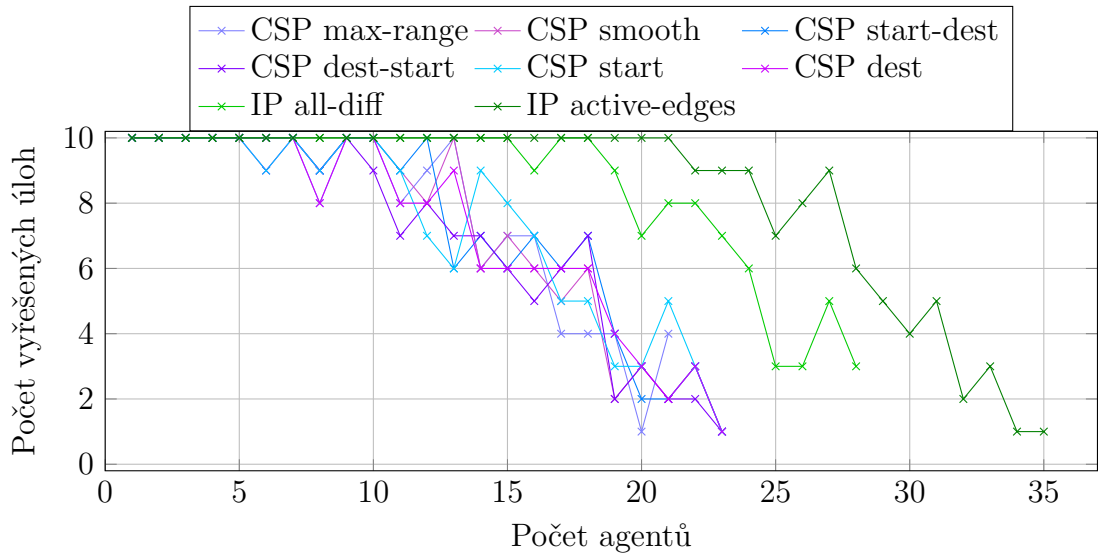
Zdá se, že pro vyšší množství překážek se průměrně zhoršují dolní odhady na minimální makespan.



Obrázek 7.33: Průměrná celková cesta agentů pro 10 úloh

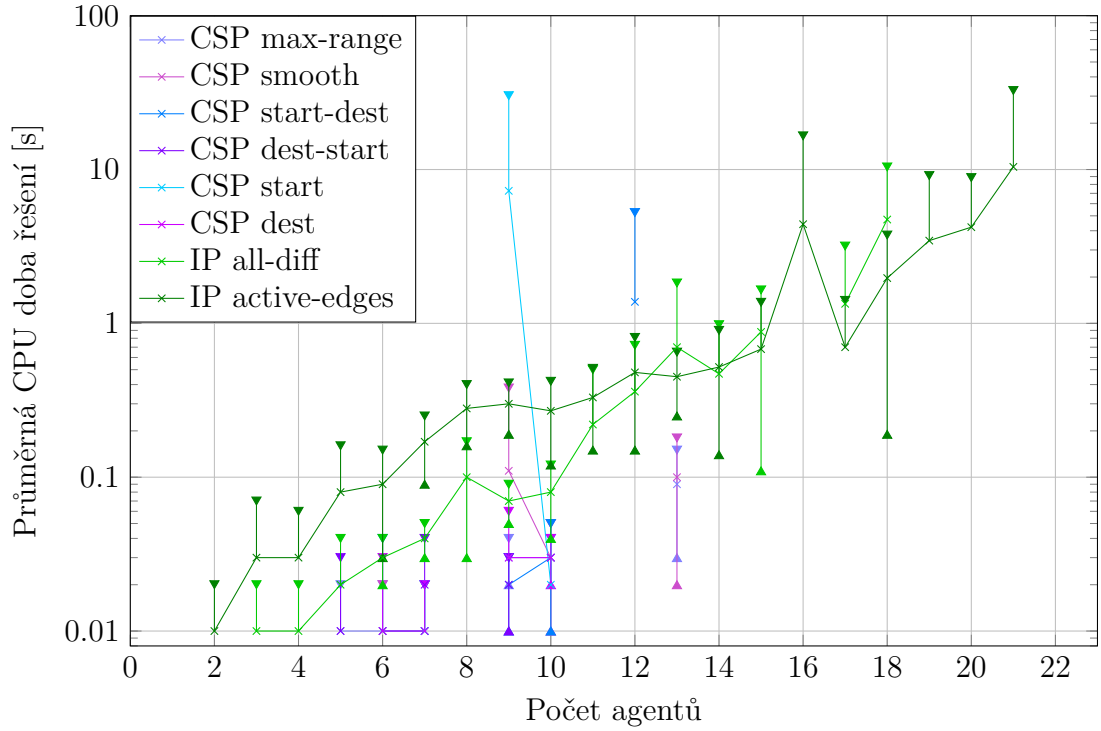
Zde se opět ukazuje, že model CSP s heuristikami využívajícími znalost nejkratších cest a algoritmus OD-ID nacházejí řešení, která průměrně obsahují nižší množství přesunů agentů.

7.8 Úloha CPF no-head-on s 20% překážek



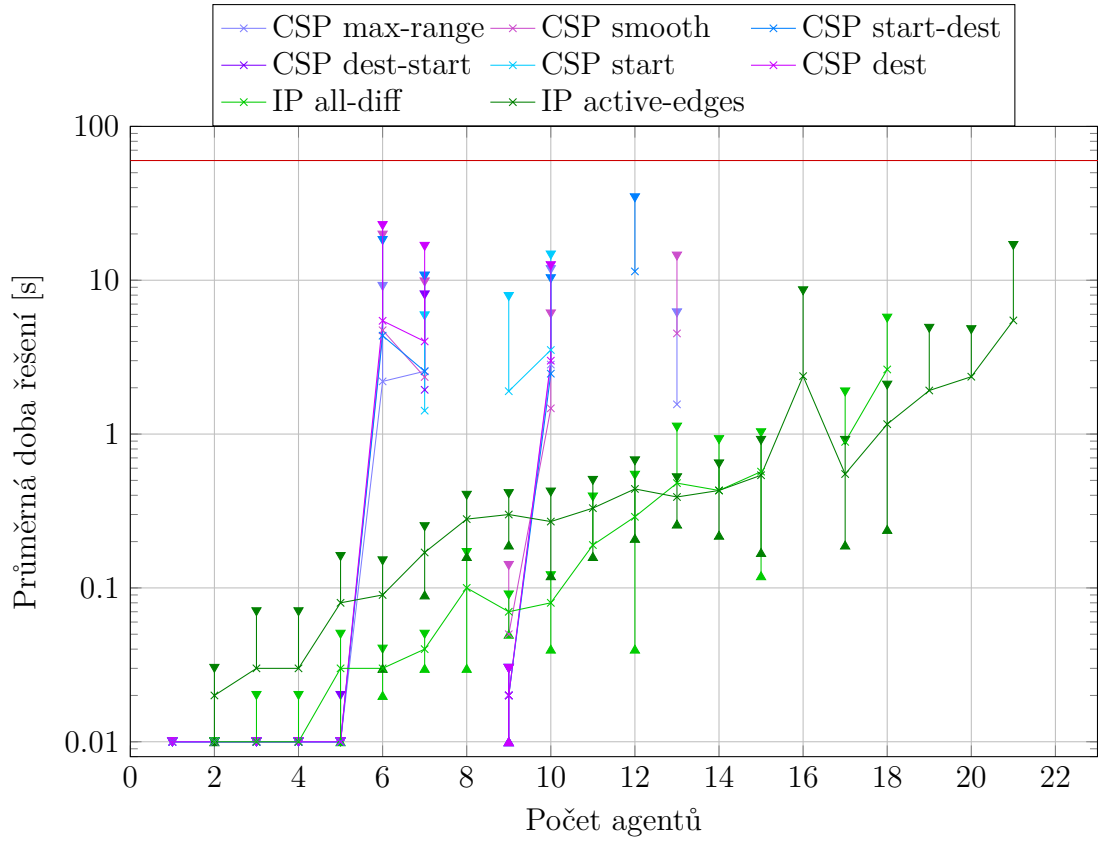
Obrázek 7.34: Množství vyřešených úloh do 60s

Výrazně nejvíce instancí se podařilo vyřešit modelu IP active-edges, za nímž zaostával model IP all-diff. Model CSP all-diff vyřešil podstatně méně instancí, než oba IP modely a to nezávisle na zvolené heuristice.



Obrázek 7.35: Průměrná CPU doba řešení 10 úloh

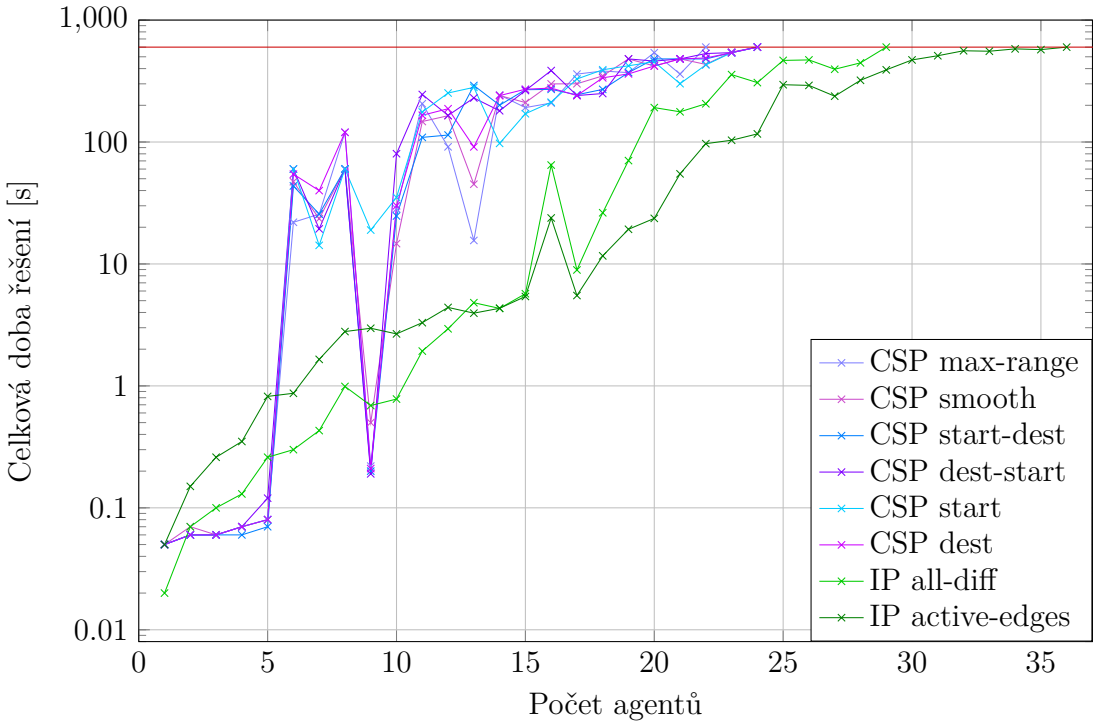
Pro nižší počet agentů se CSP modelu podařilo nalézt řešení během podstatně nižšího procesorového času, než IP modely. Pro vyšší množství agentů se u něj objevují ve využitém procesorovém času velké výkyvy.



Obrázek 7.36: Průměrná doba řešení 10 úloh

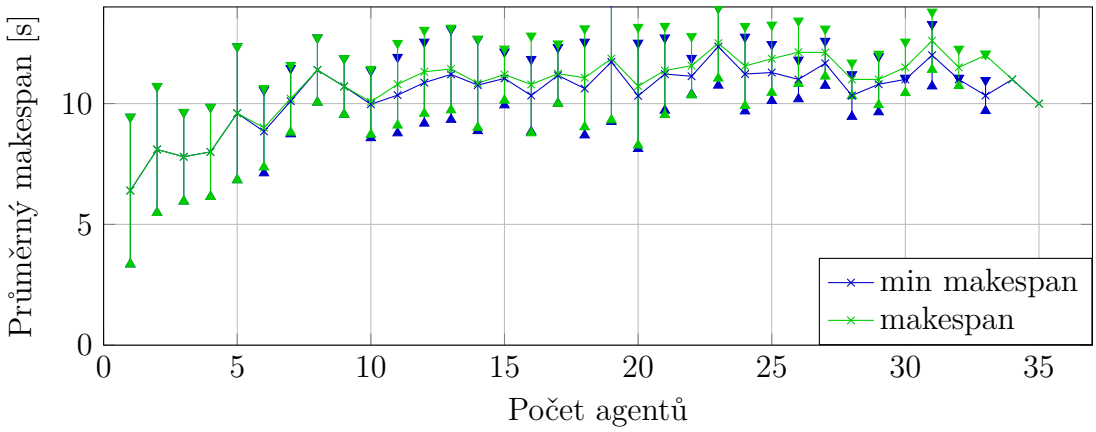
Zde lze znovu vidět místy dlouhou skutečnou dobu, kterou potřeboval CSP model k nalezení řešení i v případech s jen nízkým využitím procesoru.

Model IP all-diff vykazuje lepší výsledky pro nižší hustotu agentů, pro vyšší hustotu agentů naopak nedokázal vyřešit tolik instancí jako model IP active-edges.



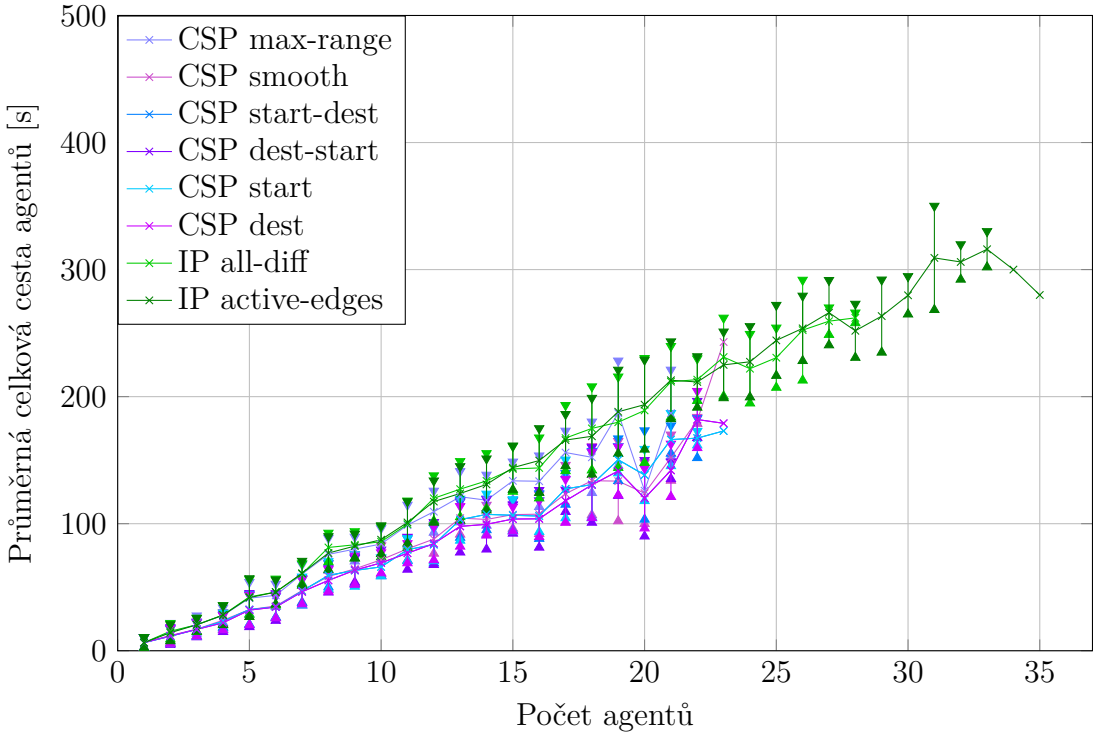
Obrázek 7.37: Celková doba řešení 10 úloh

V grafu 7.37 je vidět, že stabilní růst doby, kterou při řešení instancí s rostoucím množstvím agentů potřebovaly IP modely. U CSP modelu jsou pro více než 5 agentů vidět výrazné výkyvy, kdy pro některé instance nezvládne nalézt řešení v časovém limitu, zatímco jiné zvládne vyřešit relativně rychle. Pro devět agentů je v grafu vidět, že CSP model s heuristikou start potřeboval výrazně delší dobu k řešení, než ostatní heuristiky. V některých případech může tedy zvolená heuristika výrazně ovlivnit dobu hledání řešení.



Obrázek 7.38: Odhadovaný a skutečný makespan pro 10 úloh

Znovu se ukazuje, že pro problém CPF-no-head-on se daří skutečný minimální makespan poměrně dobře odhadnout. Na druhou stranu se množství špatných odhadů na mřížce s 20% překážek nepatrně zvýšilo.



Obrázek 7.39: Průměrná celková cesta agentů pro 10 úloh

IP modely spolu s modelem CSP all-diff, který využívá heuristiku max-range nacházejí opakovaně řešení s větším množství přesunů agentů po grafu, než u CSP modelu s heuristikami znajícími nejkratší cesty mezi vrcholy.

Závěr

Vytvořili jsme aplikaci použitelnou pro problému CPF s požadavkem na minimální makespan. K řešení poskytuje možnost zvolit si nejen různými SAT modely a algoritmem OD+ID, ale nabízí i nový CSP model s několika různými heuristikami pro výběr hodnoty proměnných a dva nové modely kódované pomocí celočíselného programování s binárními proměnnými. Všechny nové modely je možné použít pro řešení instancí tří různých variant problému CPF. Aplikace navíc umožňuje generovat náhodné instance těchto problémů, a to na několika různých typech grafů.

Pomocí vytvořené aplikace a několika bash skriptů jsme provedli srovnávací testy vybraných modelů popsaných v CSP, BIP a SAT společně s algoritmem OD+ID na náhodných instancích problému CPF na mřížce o velikosti 8x8 s různým množstvím překážek.

7.9 Zhodnocení výsledků testování

Testování jsme prováděli pouze na grafech ve tvaru mřížky, která se běžně používají pro srovnávání algoritmů pro řešení CPF.

Pozorovali jsme, že modely popsané jako lineární celočíselné programování s binárními proměnnými dokážou konkurovat modelům pro SAT. Přestože knihovna Guroby použitá pro BIP modely dokáže využít více procesorů, na rozdíl od použitého SAT řešiče Glucose 3.0, bylo na řešení BIP modelů překvapivě využíváno spíše méně procesorového času. Model IP all-diff vykazoval vyšší rychlost pro nižší počet agentů, pro vyšší množství se situace obrátila ve prospěch modelu IP active-edges.

CSP má výhodu v možnosti přidat vlastní heuristiku pro volbu proměnných jejich hodnot, díky tomu může využít znalosti nejkratších vzdáleností. Jeho výhodou je i možnost využití globální podmínky all-different. Potvrdilo se sice, že pro náhodná zadání s nižší hustotou agentů bývá řešení CSP modelu nalezeno velmi rychle. Překvapili nás však horší výsledky CSP modelu pro vyšší množství agentů. Navíc se ukázalo, že má tento model problémy i s některými instancemi s nižším množstvím agentů. To snižuje jeho možnosti praktického použití, protože tak nelze dobře odhadnout čas potřebný k nalezení řešení pro daný počet agentů. Ukázalo se tedy, že vyzkoušet řešit problém mCPF ve více různých formalismech mělo smysl, neboť algoritmy pro IP předčili naše očekávání, naopak od CSP modelu jsme očekávali lepší vlastnosti i pro více agentů, díky jeho možnosti použít globální podmínku all-different.

V testu jsme dále porovnávali současně několik různých heuristik pro CSP model. Rozdíly mezi jejich časovými výsledky se však neprokázali jako výrazné. Všechny heuristiky, kromě heuristiky max-range, využívaly znalost nejkratších cest v grafu. Jejich výhoda se ukázala především v tom, že dokázali nacházet řešení s průměrně menším množstvím přesunů agentů. Porovnávaný algoritmus OD+ID zvládal řešit rychle pouze instance s velmi nízkým množstvím agentů, při jejich rostoucím množství se pro něj stávaly úlohy rychle neřešitelné.

Z grafů je vidět, že varianta CPF-no-head-on je řešitelná výrazně rychleji oproti té, která povoluje pouze přesun do prázdných vrcholů. Navíc se dle našich

pozorování minimální makespan u problémů CPF-no-head-on liší jen minimálně od použitých dolních odhadů.

7.10 Možná rozšíření

Možná rozšíření o další testy.

- Jednou z možností je vyzkoušet modelovat problém CPF pomocí dalších formalismů, jako je například ASP.
- Provést porovnání s dalšími algoritmy na hledání optimálního makespan
- Vyzkoušet program pro řešení SAT, který umí využít více jader procesoru
- Provést testy na instancích s jiným typem grafů, k tomu je možné využít generátory problému CPF zahrnuté v aplikaci cpf.

Možná budoucí rozšíření aplikace.

- Doplnit do programu algoritmus ověřující kompletně řešitelnost dané úlohy v lineárním čase.
- Přidat možnost generovat pouze řešitelné instance CPF
- Vytvořit heuristiku pro výběr hodnot proměnných v CSP modelu kombinující hledání znalost nejkratších cest a znalost velikosti domény proměnných
- Upravit SAT modely tak, aby podporovaly obě definice problému CPF
- Umožnit použití více různých řešičů pro použité formalismy

Problém CPF je možné dále rozšířit mnoha způsoby, z nichž některé se již zkoumají.

- Orientovaný graf
- Skupiny nerozlišitelných agentů, u nichž nezáleží na tom, který dorazí do kterého z jejich cílů
- Více alternativních cílů/počátků pro každého agenta
- Kapacity vrcholů a hran
- Délky hran

Model CSP all-diff lze díky možnosti využití algoritmu branch-and-bound knihovny Gecode snadno rozšířit tak, aby mezi nalezenými řešeními s optimálním makespan, bylo řešení optimální vzhledem k dalšímu vhodnému kritériu. Příkladem může být minimalizace součtu časových kroků, během niž se dostanou jednotlivý agenti do cíle.

Seznam použité literatury

Botea, Adi et al. (2013). „Pathfinding in Games“. In: *Artificial and Computational Intelligence in Games*. Ed. Simon M. Lucas et al. Sv. 6. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, s. 21–31. ISBN: 978-3-939897-62-0. DOI: http://dx.doi.org/10.4230/DFU.Vol6.12191.21.

Boyarski, Eli et al. (2015). „ICBS: Improved Conflict-Based Search Algorithm for Multi-Agent Pathfinding“. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, s. 740–746.

Goraly, Gilad a Refael Hassin (2010). „Multi-Color Pebble Motion on Graphs“. English. In: *Algorithmica* 58.3, s. 610–636. ISSN: 0178-4617. DOI: 10.1007/s00453-009-9290-7.

Hart, P.E., N.J. Nilsson a B. Raphael (1968). „A Formal Basis for the Heuristic Determination of Minimum Cost Paths“. In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2, s. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

Cho, Hyun Cheol et al. (2008). „Routing of Linear Motor based Shuttle Cars in the Agile Port Terminal with Constrained Dynamic Programming“. In: *International Journal of Control Automation and Systems* 6, s. 278–281.

Kornhauser, D. M. (1984). *COORDINATING PEBBLE MOTION ON GRAPHS, THE DIAMETER OF PERMUTATION GROUPS, AND APPLICATIONS*. Tech. zpr. Cambridge, MA, USA.

Koupý, Petr (2013). *GraphRec*. URL: http://www.koupy.net/graphrec.php.

Luna, Ryan a Kostas E. Bekris (2011). „Push and Swap: Fast Cooperative Pathfinding with Completeness Guarantees“. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume One*. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, s. 294–300. ISBN: 978-1-57735-513-7. DOI: 10.5591/978-1-57735-516-8/IJCAI11-059.

Röger, Gabriele a Malte Helmert (2012). „Non-Optimal Multi-Agent Pathfinding is Solved (Since 1984)“. In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*.

Ryan, Malcolm (2007). „Graph Decomposition for Efficient Multi-robot Path Planning“. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., s. 2003–2008.

— (2008). „Constraint-Based Multi-agent Path Planning“. English. In: *AI 2008: Advances in Artificial Intelligence*. Ed. Wayne Wobcke a Mengjie Zhang. Sv. 5360. Lecture Notes in Computer Science. Springer Berlin Heidelberg, s. 116–127. ISBN: 978-3-540-89377-6. DOI: 10.1007/978-3-540-89378-3_12.

Sharon, Guni, Roni Stern, Meir Goldenberg et al. (2011). „The Increasing Cost Tree Search for Optimal Multi-Agent Pathfinding“. In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Ed. Toby Walsh. IJCAI/AAAI, s. 662–667. ISBN: 978-1-57735-516-8.

Sharon, Guni et al. (2012a). „Conflict-Based Search for Optimal Multi-Agent Path Finding“. In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*.

— (2012b). „Meta-Agent Conflict-Based Search For Optimal Multi-Agent Path Finding“. In: *Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS 2012, Niagara Falls, Ontario, Canada, July 19-21, 2012*.

Silver, David (2005). „Cooperative Pathfinding.“ In: *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2005)*. AAAI Press, s. 117–122.

Siméon, T., S. Leroy a J.P. Laumond (2002). „Path coordination for multiple mobile robots : geometric algorithms“. In: *IEEE Transaction on Robotics and Automation* 18.1.

Standley, Trevor Scott a Richard E. Korf (2011). „Complete Algorithms for Cooperative Pathfinding Problems“. In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Ed. Toby Walsh. IJCAI/AAAI, s. 668–673. ISBN: 978-1-57735-516-8.

Surynek, Pavel (2009). „A novel approach to path planning for multiple robots in bi-connected graphs“. In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, s. 3613–3619.

— (2010). „An Optimization Variant of Multi-Robot Path Planning Is Intractable.“ In: *AAAI*. Ed. Maria Fox a David Poole. AAAI Press.

— (2012a). „A SAT-Based Approach to Cooperative Path-Finding Using All-Different Constraints.“ In: *SOCS*.

— (2012b). „Towards optimal cooperative path planning in hard setups through satisfiability solving“. In: *PRICAI 2012: Trends in Artificial Intelligence*. Springer, s. 564–576.

— (2014). „Compact Representations of Cooperative Path-Finding as SAT Based on Matchings in Bipartite Graphs“. In: *Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on*, s. 875–882. DOI: 10.1109/ICTAI.2014.134.

Walsh, Toby, ed. (2011). *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. IJCAI/AAAI. ISBN: 978-1-57735-516-8.

Wang, Ko-Hsin Cindy a Adi Botea (2011). „MAPP: A Scalable Multi-agent Path Planning Algorithm with Tractability and Completeness Guarantees“. In: *J. Artif. Int. Res.* 42.1, s. 55–90. ISSN: 1076-9757.

Wilde, Boris de, Adriaan W. ter Mors a Cees Witteveen (2013). „Push and Rotate: Cooperative Multi-agent Path Planning“. In: *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*. AAMAS '13. St. Paul, MN, USA: International Foundation for Autonomous Agents a Multiagent Systems, s. 87–94. ISBN: 978-1-4503-1993-5.

Wilson, Richard M (1974). „Graph puzzles, homotopy, and the alternating group“. In: *Journal of Combinatorial Theory, Series B* 16.1, s. 86–96. ISSN: 0095-8956. DOI: http://dx.doi.org/10.1016/0095-8956(74)90098-7.

Wilt, Christopher Makoto a Adi Botea (2014). „Spatially Distributed Multiagent Path Planning“. In: *Proceedings of the Twenty-Fourth International Confe-*

rence on Automated Planning and Scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21-26, 2014.

Yu, Jingjin (2013). „A Linear Time Algorithm for the Feasibility of Pebble Motion on Graphs“. In: *ArXiv e-prints*. arXiv: 1301.2342 [cs.DS].

Yu, Jingjin a Steven M LaValle (2013a). „Planning Optimal Paths for Multiple Robots on Graphs“. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, to–appear.

— (2013b). „Structure and Intractability of Optimal Multi-robot Path Planning on Graphs“. In: *The Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI-13)*, s. 1444–1449.

Yu, Jingjin a Daniela Rus (2015). „Pebble Motion on Graphs with Rotations: Efficient Feasibility Tests and Planning Algorithms“. English. In: *Algorithmic Foundations of Robotics XI*. Ed. H. Levent Akin et al. Sv. 107. Springer Tracts in Advanced Robotics. Springer International Publishing, s. 729–746. ISBN: 978-3-319-16594-3. DOI: 10.1007/978-3-319-16595-0_42.

Seznam obrázků

Seznam použitých zkratek

ASP Answer Set Programming — programování množinou dotazů. 48

BIP Binary Integer Programming — celočíselné programování s binárními proměnnými. ii, 3, 4, 9, 15, 47, 58

CA* Cooperative A* — algoritmus na řešení CPF. 6

CBS Conflict-Based Search — algoritmus na řešení CPF. 8

CPF Cooperative Pathfinding — kooperativní hledání cest. ii, 3–7, 10, 11, 13, 18, 47, 48, 52–54, 56–58, 60–63

CSP Constraint Satisfaction Problems — problémy splňování podmínek. ii, 3, 4, 6, 9, 13, 21–25, 27, 28, 30, 31, 34–48, 57

HCA* Hierarchical Cooperative A* — algoritmus na řešení CPF. 6

ICBS Improved Conflict-Based Search — algoritmus na řešení CPF. 8

ICTS Increasing Cost Tree Search — algoritmus na řešení CPF. 8

IP Integer Programming — celočíselné programování. ii, 3, 4, 8, 9, 15, 21, 22, 25, 27, 34, 35, 39, 41, 43, 45–47, 57

LRA* Local Repair A* — algoritmus na řešení CPF. 6

MA-CBS Meta-Agent Conflict-Based Search — algoritmus na řešení CPF. 8

MAPF Multi-Agent Pathfinding — hledání cest pro více agentů. 5

MAPP Multi-Agent Path Planning — algoritmus na řešení CPF. 6

mCPF Makespan Optimal Cooperative Pathfinding — kooperativní hledání cest s minimálním makespan. 3, 4, 9, 11, 47

OD+ID Operator Decomposition + Independence Detection — algoritmus na řešení CPF. 3, 8, 12, 18, 21–24, 32, 33, 38, 47, 57

PBO Pebble Motion on Graph — pohyb kamenů po grafu. 6, 8

PDDL Planning Domain Definition Language — formát pro reprezentaci plánovacích problémů. 63

SAT Boolean Satisfiability — problémy splnitelnosti výrokových proměnných. ii, 3, 4, 8, 9, 12, 21, 22, 30, 39, 41, 47, 48, 57

SDP Spatially Distributed Multiagent Planner — algoritmus na řešení CPF. 6

WHCA* Windowed Hierarchical Cooperative A* — algoritmus na řešení CPF. 6

Příloha A – Obsah elektronické přílohy

- Elektronická verze textu bakalářské práce
- Adresář „bin“ obsahuje spustitelný soubor aplikace „cpf“, soubor s postupem instalace „INSTALL.txt“ a vzorový konfigurační soubor.
- Adresář „src“ obsahuje zdrojové kódy aplikace
- V adresáři „tests“ jsou v podadresáře pro jednotlivá provedená porovnání zahrnující bash skripty „solve.sh“ pro provedení testů, soubory s vyřešenými instancemi a soubory „statistics.txt“ se získanými hodnotami testů. V podadřesáři „result_data“ jsou pak soubory s odvozenými daty, které byly použity pro vytvoření výsledných grafů.
- Adresář „examples“ zahrnuje příklady vyřešených instancí pro různé typy generovaných grafů v souborech zobrazitelných v programu „GraphRec“. Podadresář „cuboid“ obsahuje příklady různých formátů instance CPF a jejího řešení.

Příloha B – Návod pro uživatele

Aplikace je určena pro operační systémy s jádrem Linux. Zahrnuje jediný samostatně spustitelný soubor s názvem cpf. Ke svému běhu vyžaduje několik knihoven, jejichž instalace je popsána v části Instalace. Aplikace má konzolové rozhraní a lze ji spustit se jednoduše z konzole spolu se zadanými parametry. Po spuštění jednoduše vykoná zadanou úlohu, jejíž průběh se zobrazuje na konzoli, a skončí. P Je také možno použít konfigurační soubor, obsahující defaultní parametry. Seznam všech parametrů je možno zobrazit spuštěním aplikace s parametrem -h nebo –help. Příklad

```
> cpf -x=8 -y=8 -z=2 -o=5 -r=10 -S=ip-all-diff -t=60
```

B.1 Instalace

Aplikace ke svému běhu vyžaduje mít nainstalované následující knihovny. Více informací o nich a o tom, jak je správně nainstalovat naleznete na jejich oficiálních webových stránkách. Součástí elektronické přílohy je soubor INSTALL.txt obsahující informace o tom, jak nastavit systémové proměnné. Důležité je především přidat cesty k použitým dynamickým knihovnám do $LD_LIBRARY_PATH.

- Gurobi Optimizer – http://www.gurobi.com – zde je nutné mít pro daný počítač registrovanou dostatečnou licenci, pro akademické účely je možné využívat knihovnu zdarma
- Gecode – http://www.gecode.org
- Glucose – http://www.labri.fr/perso/lsimon/glucose/
- LEDA – http://www.algorithmic-solutions.com/leda/index.htm

B.2 Návratové hodnoty

Návratová hodnota programu je užitečná při dávkovém zpracování, kdy podle ní může skript určit, jak má pokračovat. Aplikace cpf vrací vždy jednu z následujících hodnot:

- 0 – řešení byloúspěšně nalezeno
- 1 – řešení nebylo nalezeno – tato hodnota je vyhrazena, protože se používá interně při mezivýpočtech, jinak by neměla být programem vracena
- 2 – zadaná úloha je neřešitelná
- 3 – v zadaném rozsahu možných makespanů nebylo nalezeno žádné řešení
- 4 – výpočet překročil nastavený maximální časový limit
- 5 – chyba

B.3 Zobrazení řešení v programu GraphRec

Nalezená řešení uložená v souboru ve formátu Multirobot je možné zobrazit pomocí programu Graphrec (Koupý, 2013). Ukázkové instance problému CPF uvedené jako obrázky u vyhodnocení testů byly zobrazeny právě v tomto programu.

Příloha C – Implementace

Pro implementaci aplikace jsme zvolili programovací jazyk C++ ve verzi 11. Hlavním důvodem byla možnost jednoduchého použití některých částí aplikace reLOC, která je v C++ naprogramována. Velkou výhodou C++ je také to, že je možné v něm lze psát vysoce optimalizovaný software a proto v něm také existuje naprogramováno mnoho specializovaných knihoven pro řešení různých výpočetně náročných úloh.

C.1 Použitý software

Pro testování jsme využily následující software.

- Aplikace na řešení CPF pomocí SAT modelů – reLOC - využité zdrojové kódy jsme zabalili do statické knihovny reLOC_base (viz kap. C.2)
- Aplikace pro řešení SAT – Glucose 3.0
- Knihovna pro modelování a řešení CSP – Gecode 4.4.0
- Knihovna pro modelování a řešení IP – Gurobi Optimizer 6.0
- Generování různých typů grafů – LEDA 6.3
- Aplikace pro vizualizaci řešení CPF – GraphRec (Koupý, 2013)

C.2 Rozvržení programu

Program se skládá ze spustitelného souboru „cpf“, ke kterému jsou přilinkovány statické knihovny.

- cpf – Obsahuje konzolové rozhraní aplikace (main.h), zpracování vstupních a výstupních souborů (io.h) a rozhraní pro použitý algoritmus OD+ID (od_id_solver.h) a SAT modely (sat_solvers.h) z knihovny reLOC_base
- cpf_base – Statická knihovna obsahující abstraktní třídu MultirobotOptimalSolver (cpf_solvers.h), náhodné generátory problému CPF (cpf_generators.h) a funkce pro měření času (meassuring.h)
- reLOC_base – Statická knihovna zabalující základní třídy použité z existujícího programu reLOC. Především to jsou reprezentace instancí CPF, SAT modely, implementace algoritmu OD+ID, výpočet nejkratších cest mezi všemi vrcholy, načítání instancí ve formátu Multirobot a ukládání řešení ve formátu Multirobot
- cpf_csp – Statická knihovna obsahující především nový model CSP alldiff a k němu volitelné heuristiky (csp_all_diff_heuristic_solver.h), dále obsahuje několik dalších pokusných CSP modelů, u nichž se však neprokázal vysoký výkon. Nebyly proto již dále doplňovány o všechny definice a ořezání modelu.

- cpf_ip – Statická knihovna obsahující dva nové modely kódované jako BIP (ip_all_diff_solverG.h, ip_active_edges_solverG.h).

C.2.1 Třída MultirobotOptimalSolver

Jsou od ní odvozené ostatní řešiče. Obsahuje výchozí částečné ověření řešitelnosti problému CPF, výpočet dolního odhadu minimálního makespan a nalezení řešení pro triviální případy s minimálním makespan 0 nebo 1. Především pak obsahuje algoritmus, který postupně od zvolené dolní hranice po zvolenou horní hranici pro hledání makespan volá abstraktní funkci solve_Instance(sMultirobotSolution& solution, unsigned makespan), dokud není řešení nalezeno, nebo nevyprší časový limit. V případě, že není nalezeno ani po dosažení horní hranice pro hledání, je vrácena hodnota UNSOLVABLE_FOR_MAKESPANS signalizující, že nebylo v daném rozsahu nalezeno žádné řešení. To může znamenat buď to, že má úloha řešení s větším makespan, nebo, v případě kdy nebyla provedena úplná kontrola řešitelnosti, může být rovněž úloha neřešitelná. Stejná hodnota je vrácena i pokud byla zadána vyšší dolní, než horní hranice.

C.3 Generování instancí CPF

Pro generování všech grafů, kromě mřížek a náhodných grafů s daným počtem vrcholů a hran, využíváme knihovnu LEDA. Pro generování problému CPF na různých grafech slouží následující funkce:

- generateObstaclesPositions(x, y, početPřekážek, seed, indikátory, vrcholy)
- createGridWithObstacles(graf, x, y, početPřekážek, const vrcholy)
- createCuboidWithObstacles(graf, x, y, z, početPřekážek, const vrcholy)
- generateGridInstance(instance, x, y, početAgentů, početPřekážek, seed, vrcholy)
- generateGridInstanceP(instance, x, y, početAgentů, double obstacleProbability, seed)
- generateCuboidInstance(instance, x, y, z, početAgentů, početPřekážek, seed, vrcholy)
- generateMaximalPlanarInstance(instance, početVrcholů, početAgentů, seed)
- generatePlanarInstance(instance, početVrcholů, početAgentů, seed)
- generatePlanarInstance(instance, početVrcholů, početHran, početAgentů, seed)
- generateTriangularInstance(instance, početVrcholů, početAgentů, seed)
- generateSeriesParallelInstance(instance, početVrcholů, početHran, početAgentů, seed)

- generateCompleteGraphInstance(instance, početVrcholů, početAgentů, seed)
- generateGraphInstance(instance, početVrcholů, početHran, početAgentů, seed)

C.4 Bash skripty pro spuštění sérií testů

Jsou součástí elektronické přílohy. Pro každé porovnání je vytvořen jeden skript. Obsahuje seznam řešičů a případných heuristik, pro které se má test provádět. Obsahuje seznam dostatečného množství seedů, kterými se inicializuje generátor pseudonáhodných čísel v programu pro jednotlivé alternativní instance. Kontroluje se, zda byla úloha řešitelná a pokud ne, vybere se následující seed. Vzhledem k tomu, že to, zda je úloha řešitelná nezávisí na volbě solveru, je zajištěno, že se všechny solvery spustí nad stejnou skupinou seeds. Dále obsahuje kontrolu, zda bylo řešeno všech 10 řešitelných instancí, pokud ano, pak buď byla alespoň jedna z nich vyřešena v časovém limitu a pak se pokračuje v řešení pro počet robotů o jedna vyšší, jinak se začne řešit pro následující řešič.

C.5 Překlad zdrojových kódů

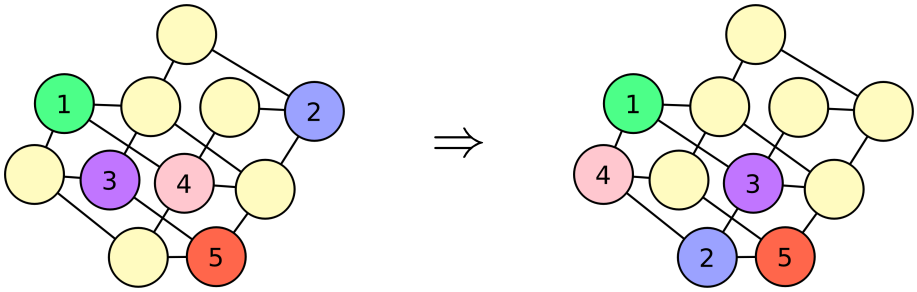
Program byl psán ve vývojovém prostředí Eclipse, pro přeložení je možné využít automaticky vygenerované makefile. Ve složkách s názvem „lib“ jsou přeložené statické knihovny a ve složkách s názvem „include“ hlavičkové soubory určené pro linkování.

C.6 Měření času

Pro měření času a procesorového času jsme vytvořili funkce getWallClockTime() a getCpuTime(). Ty obsahují volání funkcí operačního systému a jsou vytvořeny ve verzích pro Linux a pro Windows.

Příloha D – Formáty souborů

V této příloze jsou popsány formáty souborů, které používá aplikace cpf pro načítání instancí, nebo pro ukládání nalezených řešení problému CPF. Příklady formátů odpovídají problému CPF na následujícím obrázku D.40.



Obrázek D.40: Kvádr $2 \times 3 \times 2$ s překážkou a pěti agenty

D.1 Konfigurační soubor

Do textového souboru s názvem „default.conf“ umístěného ve stejném adresáři, jako je aplikace cpf, lze uložit seznam předdefinovaných parametrů, které se mají přidávat ke každému volání aplikace cpf. Pokud je aplikace cpf volána s některými parametry se stejnými názvy jako v souboru „default.conf“, mají parametry zadané společně s voláním programu přednost.

Seznam všech použitelných parametrů je možné zjistit zobrazením nápovědy aplikace voláním „cpf -h“. Speciálním případem jsou parametry, na jejichž začátku je -%. Takové parametry jsou ignorovány.

```
-x=4 -y=3 -z=2 -o=4 -r=10 -s=789 -O=solution.txt
-%--out-instance=instanceMultirobot.txt
-%--out-instance-grid=instanceGrid.txt
-%--out-instance-pddl=instancePDDL.txt
```

D.2 Soubor se společnou statistikou

Společná statistika pro jednotlivé testy se ukládá do souboru uvedeného za parametrem „–out-global-stat=“. Pokud soubor neexistuje, vytvoří se, a vloží se do něj první řádek s názvy sloupců a další řádek s výsledky pro aktuální řešení. Existuje-li soubor, pak se jen přidá nový řádek s výsledky na jeho konec.

Tato statistika se ukládá do souboru ve formátu se středníkem oddělenými sloupci hodnot na řádcích. První řádek obsahuje názvy sloupců hodnot pod ním. Názvy jsou následující:

```
Seed;Definition;Graph;X;Y;Z;Vertices;Edges;Obstacles;Robots;Solver;
MinMakespan;Makespan;Moves;Solved;ProcessorTime[ms];Duration[ms]
```

Příklad řádku s příslušnými hodnotami:

```
573358282;vacant-target;grid;8;8;1;51;69;13;17;ip-active-edges;
11;12;159;solved;5080;2693
```

D.3 Formát Multirobot

Formát Multirobot je určen k načítání i ukládání instancí CPF a také k ukládání nalezených řešení. Pro načítání a ukládání v tomto formátu využíváme knihovnu reLOC_base.

D.3.1 Instance problému CPF

Instance problému CPF je zde popsána jako neorientovaný graf (V,E), na němž jsou určeny počáteční a cílové pozice robotů. Zatímco vrcholy indexujeme od nuly, agenti jsou indexování od jedné, protože index nula uvedený místo nich značí prázdný vrchol.

Formát začíná řádkem s nápisem „V =“, za nímž následují řádky popisující rozmístění agentů ve vrcholech. Sekce končí řádkem s nápisem „E =“, za kterým následuje sekce popisující hrany v grafu.

Řádky v první sekci mají v kulaté závorce zapsán index vrcholu. Za indexem je uveden nápis „:-1“. Po zakončení kulaté závorky jsou zapsána v hranaté závorce tři čísla oddělená dvojtečkou. První značí index agenta, který má tento vrchol jako svou počáteční pozici, další dvě jsou vzájemně identická a značí index agenta s cílovou pozicí v tomto vrcholu.

Ve druhé sekci jsou na řádkách popsány hrany grafu ve složených závorkách s čárkou oddělenými indexy dvou propojených vrcholů. Za koncem složené závorky je uvedeno „(-1)“.

Soubor je zobrazen ve z úsporných důvodů zobrazen do tří sloupců.

```
V =                     (9:-1)[0:2:2]           {7,1} (-1)
(0:-1)[0:0:0]           (10:-1)[5:5:5]          {7,5} (-1)
(1:-1)[1:1:1]           E =                     {8,2} (-1)
(2:-1)[0:0:0]           {2,0} (-1)              {8,6} (-1)
(3:-1)[0:4:4]           {2,1} (-1)              {8,7} (-1)
(4:-1)[3:0:0]           {3,1} (-1)              {9,3} (-1)
(5:-1)[0:0:0]           {4,2} (-1)              {9,7} (-1)
(6:-1)[2:0:0]           {4,3} (-1)              {10,4} (-1)
(7:-1)[4:3:3]           {6,0} (-1)              {10,8} (-1)
(8:-1)[0:0:0]           {6,5} (-1)              {10,9} (-1)
```

D.3.2 Řešení problému CPF

Soubor s řešením ve formátu Multirobot je možno otevřít v programu GraphRec.

Uložené řešení obsahuje na začátku popis instance CPF, Za ním následuje řádek s nápisem „solution“. Poté jsou uvedeny jednotlivé řádky popisující přesun agentů v časovém kroku zapsaném v závorce na konci řádku z vrcholu s indexem uvedeným před šipkou „—>“ do vrcholu s indexem uvedeným za šipkou. Na posledním řádku je pak nápis „Length:“ následován počtem provedených přesunů.

Příklad je zobrazen do tří sloupců.

```
V =
(0:-1)[0:0:0]
(1:-1)[1:1:1]
(2:-1)[0:0:0]
(3:-1)[0:4:4]
(4:-1)[3:0:0]
(5:-1)[0:0:0]
(6:-1)[2:0:0]
(7:-1)[4:3:3]
(8:-1)[0:0:0]
(9:-1)[0:2:2]
(10:-1)[5:5:5]
E =
{2,0} (-1)
{2,1} (-1)
{3,1} (-1)
{4,2} (-1)
{4,3} (-1)
{6,0} (-1)
{6,5} (-1)
{7,1} (-1)
{7,5} (-1)
{8,2} (-1)
{8,6} (-1)
{8,7} (-1)
{9,3} (-1)
{9,7} (-1)
{10,4} (-1)
{10,8} (-1)
{10,9} (-1)
Solution
6 ---> 5 (0)
4 ---> 2 (0)
7 ---> 9 (0)
5 ---> 7 (1)
2 ---> 8 (1)
9 ---> 3 (1)
7 ---> 9 (2)
8 ---> 7 (3)
Length:8
```

D.4 Formát Grid

Tento formát byl vytvořen za účelem snadného a přehledného zadávání popisu problému CPF na obdélníkových a kvádrových mřížkách s překážkami.

Formát je rozdělen do sekcí uvozených slovem s dvojtečkou na konci. Na množství bílých znaků použitých k oddělování ve formátu nezáleží.

Na prvním řádku jsou uvedeny za nápisem „size: “ velikosti tří souřadnic „zyx“. Druhý řádek obsahuje za nápisem „robots: “ počet agentů. Nakonec jsou uvedeny dvě po sobě jdoucí sekce začínající řádky „start:“ a „goal:“. V nich jsou uvedeny počáteční a cílová rozmístění agentů na kvádrovém grafu. Rozmístění jsou popsána následovně.

Na řádkách pod sebou je uvedeno z matic majících y řádků a x sloupců. Ty mohou obsahovat čísla značící indexy robotů číslovaných od jedné, posloupnost podtržítek _ značících neobsazené vrcholy, případně posloupnost znaků # značících překážky.

Příklad je zobrazen do dvou sloupců.

```
size: 2 3 2
robots: 5
start:
## __
 1 __
__  3

__  2
 4 __
__  5

goal:
## __
 1 __
 4 __

__ __
 3 __
 2  5
```

D.5 Formát PDDL

Mezi formáty pro uložení instance problému CPF patří také známý formát PDDL. K hledání řešení pak lze použít software pro řešení obecných plánovacích problémů. Tento formát v současnosti není možno použít k načítání instancí. K ukládání instancí v tomto formátu rovněž využíváme knihovnu reLOC_base.

Příklad je zobrazen do dvou sloupců.

```
(define (problem multirobot_instance)
   (:domain multirobot)
   (:objects
       v0 v1 v2 v3 v4 v5
       v6 v7 v8 v9 v10
       r1 r2 r3 r4 r5
    )(:init
       (adjacent v0 v2)
       (adjacent v2 v0)
       (adjacent v1 v2)
       (adjacent v2 v1)
       (adjacent v1 v3)
       (adjacent v3 v1)
       (adjacent v2 v4)
       (adjacent v4 v2)
       (adjacent v3 v4)
       (adjacent v4 v3)
       (adjacent v0 v6)
       (adjacent v6 v0)
       (adjacent v5 v6)
       (adjacent v6 v5)
       (adjacent v1 v7)
       (adjacent v7 v1)
       (adjacent v5 v7)
       (adjacent v7 v5)
       (adjacent v2 v8)
       (adjacent v8 v2)
       (adjacent v6 v8)
       (adjacent v8 v6)
       (adjacent v7 v8)
       (adjacent v8 v7)
       (adjacent v3 v9)
       (adjacent v9 v3)
       (adjacent v7 v9)
       (adjacent v9 v7)
       (adjacent v4 v10)
       (adjacent v10 v4)
       (adjacent v8 v10)
       (adjacent v10 v8)
       (adjacent v9 v10)
       (adjacent v10 v9)
       (robot_location r1 v1)
       (robot_location r2 v6)
       (robot_location r3 v4)
       (robot_location r4 v7)
       (robot_location r5 v10)
       (no_robot v0)
       (no_robot v2)
       (no_robot v3)
       (no_robot v5)
       (no_robot v8)
       (no_robot v9)
   )(:goal (and
       (robot_location r1 v1)
       (robot_location r2 v9)
       (robot_location r3 v7)
       (robot_location r4 v3)
       (robot_location r5 v10)
       (no_robot v0)
       (no_robot v2)
       (no_robot v4)
       (no_robot v5)
       (no_robot v6)
       (no_robot v8)
   )))
```