Vizualizace jako prostředek k získání znalostí o kvalitě řešení problémů pohybu po grafu*

Pavel Surynek[1] a Petr Koupý[2]

[1]Katedra teoretické informatiky a matematické logiky,
[2]Matematicko-fyzikální fakulta, Univerzita Karlova v Praze,
Malostranské náměstí 25, 118 00, Praha 1
pavel.surynek@mff.cuni.cz, petr.koupy@gmail.com

Abstrakt. Článek pojednává o problémech pohybu po grafu. Tyto problémy představují abstrakci pro úlohy, kdy je třeba naplánovat pohyby jistých mobilních entit, jež mají za úkol se v daném prostředí přesunout ze zadaného počátečního na zadané cílové místo. Přitom je třeba vyhýbat se vzájemným kolizím a překážkám. V této práci je pozornost upřena na získávání znalostí o řešeních produkovaných existujícími metodami. Byl vyvinut vizualizační nástroj pro komfortní pozorování časového průběhu vyprodukovaných řešení. Na základě získaných znalostí byla navržena opatření vedoucí k dalšímu zkvalitnění řešení.

Klíčová slova: plánování, pohyb po grafu, vizualizace, optimalita, redundance

1 Úvod a motivace

Problémy pohybu po grafu [2, 5] představují základní abstrakci k řadě teoretických i reálně řešených úloh. Klasická úloha, kterou lze abstrahovat jako pohyb po grafu, se odehrává v nějakém prostředí, kde se pohybují jisté mobilní *entity* (například roboty). Každá entita přitom má zadanou počáteční a cílovou pozici. Úkolem je naplánovat pohyby entit v čase tak, aby se každá entita dostala ze své počáteční do cílové pozice. Přitom se entity musejí vyhýbat překážkám v prostředí a nesmí spolu kolidovat.

Základní abstrakce respektive diskretizace takové úlohy vypadá tak, že prostředí je modelováno jako neorientovaný graf, kde vrcholy reprezentují místa v prostředí a hrany volnou cestu mezi místy, kterým odpovídají vrcholy hranou spojené. Rozložení entit v prostředí je potom abstrahováno jako prosté zobrazení množiny entit do množiny vrcholů grafu (tj. v jednom vrcholu se nachází nejvýše jedna entita), přitom alespoň jeden vrchol grafu zůstává volný, aby byl umožněn pohyb entit. Entita se může přesunout do volného vrcholu, tím je dána dynamicita abstrakce.

Jak již bylo naznačeno, mnoho reálných úloh lze interpretovat jako pohyb entit po grafu. Příkladem může být klasické plánování pohybů pro mobilní roboty v dvourozměrném prostředí. Obecně lze u těchto úloh říci, že pohyby robotů (entit) v prostředí jsou tím komplikovanější, čím méně volného prostoru prostředí obsazené entitami (roboty) obsahuje. Zatímco u prostředí s velkým volným prostorem lze použít jednoduché algoritmy pro plánování nejkratších cest v grafu [9], v prostředích, kde je obsazenost vysoká, je nutné pracovat pomocí pokročilejších metod.

* Podporováno z Výzkumného záměru MSM 0021620838 a z grantu GAČR GP201/09/P318.

Jako problém pohybu po grafu lze formulovat také řadu hlavolamů. Nejznámějším zástupcem je zřejmě tzv. *Lloydova patnáctka* (*15-puzzle*) [4, 10]. V praxi mohou být entity představovány i pasivními objekty – příkladem je přeuspořádání kontejnerů ve skladu. V této problémové oblasti vskutku bylo využito výše zmíněné grafové abstrakce (přesun kontejnerů v přístavu), avšak aplikovaný řešící algoritmus nebyl škálovatelný vzhledem k počtu pohybujících se entit [5]. Mnoho úloh z virtuálních prostředí lze také nahlížet jako pohyb po grafu - plánování přenosu dat, kde entita je reprezentována datovým paketem. V neposlední řadě je nutno zmínit virtuální prostředí strategických počítačových her, kde je nutné plánovat pohyby skupin jednotek.

Na vyřešení abstraktní podoby úlohy již dnes existuje několik metod. Tato práce se zaměřuje především na řešící metody popsané v článcích [6, 7]. Tyto metody v současnosti představují nejlepší řešící algoritmy pro třídu problémů, kde graf modelující prostředí je 2-souvislý a počet entit je lineární vzhledem k počtu vrcholů grafu. Přesto lze vyslovit hypotézu, že řešení generovaná těmito algoritmy jsou zatížena jistými redundancemi. Ověření této hypotézy, respektive **odhalení** a **náprava** zmiňovaných **redundancí** řešení jsou hlavními přínosy této práce.

Cílem tedy je prozkoumat řešení problémů relativně netriviální velikosti – graf obsahuje desítky vrcholů a řešení stovky pohybů. Taková řešení evidentně nelze zkoumat ručně. Navíc je třeba si uvědomit, že předem nebylo jasné jaké redundance v řešeních vůbec hledat (existovalo pouze podezření). Byl proto navržen komfortní nástroj *GraphRec* [3], který umožňuje řešení problémů pohybu po grafu zkoumat vizuálně. Úkolem nástroje *GraphRec* je automaticky nalézt vhodné nakreslení grafu modelujícího prostředí a v čase vizualizovat dané řešení problému nad tímto grafem. Přitom musí být výzkumníkovi umožněno komfortní pozorování průběhu řešení.

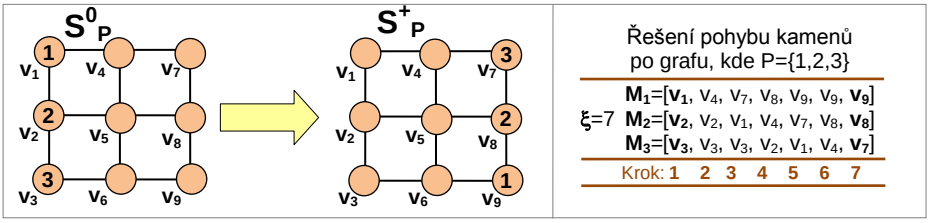
Užitím nástroje *GraphRec* se podařilo v řešeních generovaných metodami z [6, 7] odhalit několik druhů redundancí. V této práci jsou rovněž navrženy a vyhodnoceny postupy, jak nalezené redundance odstranit, což přináší další zvýšení kvality řešení.

Struktura článku je následující. Nejprve je formálně popsána hlavní varianta problému pohybu po grafu a řešící algoritmus s tím, že jsou diskutovány možné příčiny nižší kvality produkovaných řešení. Další sekce je věnována popisu vizualizačního nástroje *GraphRec*. Poté jsou algoritmicky popsány postupy na zkvalitnění řešení. Nakonec je experimentálně ukázán přínos navržených zlepšujících postupů.

2 Problém pohybu kamenů po grafu

Základní varianta problému pohybu po grafu je v literatuře známá jako *pohyb kamenů po grafu* (*pebble motion on a graph* [2, 10]). Roli entity zde hraje *kámen*. Úloha je zadána neorientovaným grafem, počátečním a koncovým rozložením kamenů ve vrcholech grafu. V každém vrcholu se vždy nachází nejvýše jeden kámen a alespoň jeden vrchol grafu je vždy volný. Cílem je nalézt pro každý kámen sekvenci pohybů, pomocí níž se kámen přesune z dané počáteční do dané cílové pozice. Dovolen je pohyb mezi vrcholy spojenými hranou, kde cílový vrchol je v okamžiku zahájení pohybu volný a žádný jiný pohyb ve stejném okamžiku nemá za cíl totožný vrchol.

Následující definice podává formální popis problému pohybu kamenů po grafu. Problém a jeho řešení je ilustrováno na obrázku 1.



Obr. 1. *Ilustrace problému* ***pohybu kamenů po grafu***. Úkol je přesunout kameny z počátečního rozmístění zadaného funkcí S_P^0 do cílového rozmístění zadaného funkcí S_P^+. Je ukázáno řešení délky 7.

Definice 1 ***(problém pohybu kamenů po grafu).*** Nechť je dán neorientovaný graf $G = (V, E)$ a množina kamenů $P = \{p_1, p_2, \dots, p_\mu\}$, kde $\mu < |V|$. **Počáteční** rozložení kamenů ve vrcholech grafu nechť je dáno pomocí prosté funkce $S_P^0: P \longrightarrow V$ (tj. $S_P^0(p_i) \neq S_P^0(p_j)$ pro $i, j = 1,2, \dots, \mu$, kde $i \neq j$), **cílové** rozložení kamenů ve vrcholech nechť je dáno pomocí prosté funkce $S_P^+: P \longrightarrow V$ (tj. $S_P^+(p_i) \neq S_P^+(p_j)$ pro $i, j = 1,2, \dots, \mu$, kde $i \neq j$). Problém ***pohybu kamenů po grafu*** je úloha najít číslo ξ a posloupnost pohybů reprezentovanou jako posloupnost vrcholů $M_p = [m_1^p, m_2^p, \dots, m_\xi^p]$ pro každý kámen $p \in P$, kde $m_i^p \in V$ pro $i = 1,2, \dots, \xi$, pro kterou platí: $m_1^p = S_P^0(p)$, $m_\xi^p = S_P^+(p)$ a buď $\{m_i^p, m_{i+1}^p\} \in E$, nebo $m_i^p = m_{i+1}^p$ pro $i = 1,2, \dots, \xi - 1$. Navíc je vyžadováno, aby posloupnosti vrcholů $M_p = [m_1^p, m_2^p, \dots, m_\xi^p]$ a $M_q = [m_1^q, m_2^q, \dots, m_\xi^q]$ pro každé dva kameny $p \in P$ a $q \in P$ takové, že $p \neq q$, splňovaly, že $m_{i+1}^p \neq m_i^q$ pro $i = 1,2, \dots, \xi - 1$ (cílový vrchol pohybu musí být volný) a $m_i^p \neq m_i^q$ pro $i = 1,2, \dots, \xi$ (žádné dva kameny nejsou současně přesouvány do stejného cílového vrcholu). □

Pro praktické aplikace je důležitá *kvalita řešení*. Typicky je vyžadováno, aby číslo ξ nebylo neúměrně velké. Požadovat **nejkratší** možné (*optimální*) řešení problému je *NP*-těžká úloha [4]. Proto je často nutné hledat kompromis mezi kvalitou řešení a časem na jeho zkonstruování. Nejsou-li na řešení kladeny žádné kvalitativní požadavky, patří úloha nalezení takového řešení do třídy *P* [10]. Ovšem řešení vygenerovaná postupy prokazující příslušnost úlohy do třídy *P* se vyznačují velkou délkou a pro praxi nejsou vhodná. Jiné metody založené na časově náročném prohledávání stavového prostoru [5] sice naleznou optimální řešení, ale jsou použitelné jen pro malé instance problému. Metody generující relativně kvalitní řešení problémů při zachování nízkých časových nároků jsou popsány v [6] a [7]. Řešení produkována právě těmito metodami budou podrobena navrženému vizualizačnímu nástroji za účelem jejich dalšího zkvalitnění.

3 Řešení problémů pohybu po grafu

Na tomto místě je třeba alespoň stručně vysvětlit základní postup řešení problémů pohybu po grafu, aby bylo lépe vidět, jakou strukturu produkovaná řešení mají. Nejvýznamnější třídou problémů pohybu kamenů po grafu pro praxi jsou problémy, kde graf modelující prostředí je *2-souvislý*. Následující definice pojem 2-souvislosti přesně vymezují.

Definice 2 ***(souvislost, 2-souvislost)*.** Neorientovaný graf $G = (V, E)$ je *souvislý*, jestliže $|V| \geq 2$ a pro každé dva různé vrcholy $u, v \in V$ existuje cesta mezi u a v v G. Neorientovaný graf $G = (V, E)$ je *2-souvislý*, jestliže $|V| \geq 3$ a pro každý vrchol $u \in V$ je graf $G' = (V - \{u\}, E \cap \{\{v, w\} | v, w \in V \wedge v \neq u \wedge w \neq u\})$ je souvislý. □

Důležitost této třídy problémů je dána hlavně tím, že problémy patřící do ní jsou téměř vždy řešitelné. Navíc je prostředí v reálných úlohách typicky abstrahováno jako graf, jehož nakreslení odpovídá mřížce, kde některé vrcholy jsou vynechané. Takový graf je ve většině případů 2-souvislý.

Pokud 2-souvislý graf obsahuje alespoň **dva** volné vrcholy a nejedná se o kružnici, pak je každé cílové rozložení kamenů dosažitelné z každého počátečního rozložení [6]. Pokud graf obsahuje pouze **jediný** volný vrchol, lze jej bez újmy na obecnosti fixovat (v souvislém grafu lze libovolný vrchol sekvencí tahů uvolnit). V takovém případě lze cílové rozložení kamenů vůči počátečnímu chápat jako *permutaci*. Permutace se nazývá *sudá*, pokud lze vyjádřit jako složení sudého počtu *výměn* prvků. Jinak se permutace nazývá *lichá*. Platí, že permutace je buď sudá, nebo lichá (nikoli obojí). Pokud je cílové rozložení kamenů sudou permutací vzhledem k počátečnímu rozložení, pak je problém vždy řešitelný. Pokud cílové rozložení kamenů vůči počátečnímu odpovídá liché permutaci, pak je problém řešitelný právě tehdy, když graf obsahuje kružnici liché délky. Tyto výsledky jsou detailně komentovány v [10]. Dá se říci, že případ s pouze jedním volným vrcholem představuje nejvíce restriktivní situaci. Řešící algoritmy se soustřeďují především na tuto situaci, případ s více volnými vrcholy je diskutován později.

Obecný případ problému pohybu kamenů po grafu, kdy graf modelující prostředí není nutně 2-souvislý, se řeší s využitím algoritmů pro 2-souvislý případ. Nejprve je zkonstruován rozklad grafu na strom 2-souvislých komponent. Poté jsou kameny přesunuty do svých cílových 2-souvislých komponent, což ovšem nemusí být vždy možné. Nakonec se v rámci jednotlivých 2-souvislých komponent použije algoritmus pro 2-souvislý případ.

Pro návrh řešících algoritmů je klíčová induktivní konstrukce 2-souvislých grafů pomocí přidávání *uch* ke grafu [8]. Nechť je dán graf $G = (V, E)$, ucho vzhledem ke grafu G je posloupnost vrcholů $L = [u, x_1, x_2, \ldots, x_l, v]$, kde $u, v \in V$ a $x_i \notin V$ pro $i = 1,2, \ldots, l$ (je dovoleno, aby $l = 0$). Výsledkem přidání ucha L ke grafu G je graf $G' = (V', E')$, kde $V' = V \cup \{x_1, x_2, \ldots, x_l\}$ a buď $E' = E \cup \{\{u, v\}\}$ pro $l = 0$, nebo $E' = E \cup \{\{u, x_1\}, \{x_1, x_2\}, \ldots, \{x_{l-1}, x_l\}, \{x_l, v\}\}$ pro $l \geq 1$. Každý 2-souvislý graf $G = (V, E)$ lze zkonstruovat z kružnice pomocí posloupnosti operací přidání ucha. Platí, že posloupnost uch ke konstrukci grafu lze efektivně určit v čase $O(|V| + |E|)$.

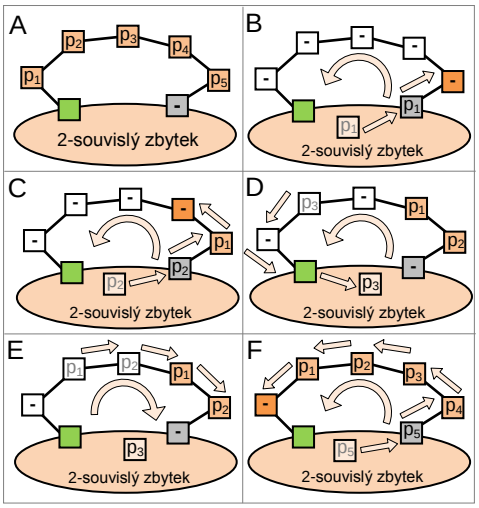
3.1 Řešící algoritmus *BIBOX-θ*

Řešící algoritmus *BIBOX-θ* [7], který je stručně připomenut v této sekci, řeší případ pohybu kamenů po grafu, kdy graf je **2-souvislý** a obsahuje právě **jeden volný** vrchol. Jak je ukázáno v [7], *BIBOX-θ* je pro popsanou třídu problémů v současnosti nejlepším algoritmem z hlediska kvality generovaných řešení a nároků na čas. Z tohoto důvodu jsou v této práci studována řešení produkovaná právě tímto algoritmem.

V první fázi práce algoritmu je nalezena kružnice a posloupnost uch, ze kterých lze daný graf zkonstruovat. Bez újmy na obecnosti lze požadovat, aby se v cílovém

rozložení kamenů volný vrchol nacházel v počáteční kružnici rozkladu. Algoritmus poté postupuje induktivně od posledního ucha rozkladu až k počáteční kružnici a prvnímu uchu.

Při umisťování kamenů v rámci ucha se uplatňují dvě základní vlastnosti: (i) v souvislém grafu, který obsahuje alespoň jeden volný vrchol, lze libovolný vrchol učinit volným; (ii) v 2-souvislém grafu s aspoň jedním volným vrcholem lze libovolný kámen přesunout do libovolného vrcholu [6] (rozmístění ostatních kamenů přitom ale zachováno není). Zvolí se orientace ucha. První vrchol ucha (tj. vrchol, kde je ucho připojeno ke zbytku grafu) je uvolněn. Dále jsou postupně zpracovávány kameny, jejichž cílové pozice jsou na druhém až předposledním vrcholu ucha.



Obr. 2. *Postup zásobníkového umisťování kamenů na cílové pozice v rámci ucha.* Část **A** ukazuje cílové rozložení kamenů. Části **B** a **C** ukazují postup zařazení nových kamenů do ucha v případě, že se nacházejí mimo ucho. Části **D** a **E** ukazují zařazení kamenu, jestliže tento se již v uchu nachází. Část **F** ukazuje závěrečné krok, kterým kameny dosáhnou svých cílových pozic. **Zelený** vrchol je volný.

Další umisťovaný kámen je přesunut do koncového vrcholu ucha. Zde je třeba rozlišit případ, kdy se cílový kámen již někde v uchu nachází a případ, kdy nikoli. Je-li kámen mimo ucho, lze jej podle vlastnosti (ii) přesunout do koncového vrcholu ucha (přitom k pohybům dojde pouze v podgrafu bez ucha). Nachází-li se kámen v uchu, je nutné jej posloupností pohybů, které způsobí rotaci kamenů směrem k prvnímu vrcholu ucha, přesunout mimo ucho. Dále se postupuje jako v prvním případě. Po umístění kamene v posledním vrcholu ucha se provede posloupnost pohybů, která způsobí rotaci kamenů o jednu hranu směrem k prvnímu vrcholu ucha. Celý postup je ilustrován na obrázku 2.

Po zpracování všech kamenů pro dané ucho je třeba řešit úlohu stejného typu pro graf bez právě zpracovaného ucha. Tento graf je opět 2-souvislý, postupuje se tedy stejně. Pro počáteční kružnici a první ucho rozkladu však uvedený postup nelze aplikovat. Algoritmus proto používá *databázi vzorů*, která obsahuje **optimální** posloupnosti tahů pro *transpozice* dvojic a *rotace trojic* kamenů. Pomocí transpozicí a rotací trojic lze obdržet libovolnou respektive sudou permutaci kamenů v rámci počáteční kružnice a prvního ucha [10].

3.2 Vlastnosti a příčiny nedostatků produkovaných řešení

Platí, že délka řešení generovaného algoritmem *BIBOX-θ* pro problém s grafem $G = (V, E)$ je $O(|V|^4)$ [6, 7]. Jestliže je úkolem vyřešit problém pohybu kamenů po 2-souvislém grafu, kde je více než jeden vrchol volný či podstatná část vrcholů je

volných, postupuje se tak, že volné vrcholy až na jeden jsou obsazeny fiktivními kameny. Modifikovaný problém je poté vyřešen algoritmem *BIBOX-θ*. Z vyprodukovaného řešení jsou následně odfiltrovány pohyby fiktivních kamenů [6].

Při takovém postupu získávání řešení problémů s mnoha volnými vrcholy hrozí vznik redundantních pohybů, které mohou zbytečně prodlužovat řešení. Ovšem tento výrok je třeba chápat jako hypotézu, kterou je teprve nutno ověřit. Stejně tak pojem *redundantní pohyb* není zatím přesně vymezen a je předmětem dalšího zkoumání.

4 Nástroj pro vizualizaci řešení

Ověřování a zkoumání kvality řešení se ukázalo být obtížné bez jisté automatizace. Byl proto vytvořen softwarový nástroj *GraphRec* [3], který umožňuje pomocí animace vizualizovat vygenerovaná řešení pohybů entit po grafu a poskytuje komfortní funkce pro usnadnění pozorování průběhu řešení v čase. *GraphRec* je dostupný na adrese: http://www.koupy.net/graphrec.php. Podobný nástroj dosud nebyl k dispozici. Většina existujících vizualizačních nástrojů pro práci s grafy, jako je například *Graphviz* [1], problémy pohybu po grafu vůbec nereflektují.

4.1 Požadované vlastnosti nástroje

Důležitým předpokladem pro přehlednou vizualizaci řešení je *kvalitní* nakreslení grafu. Kvalitně nakreslený graf by měl obsahovat malý počet křížících se hran a vzdálenost jednotlivých vrcholů by měla být úměrná jejich *hranové vzdálenosti* (tj. délce cest mezi nimi).

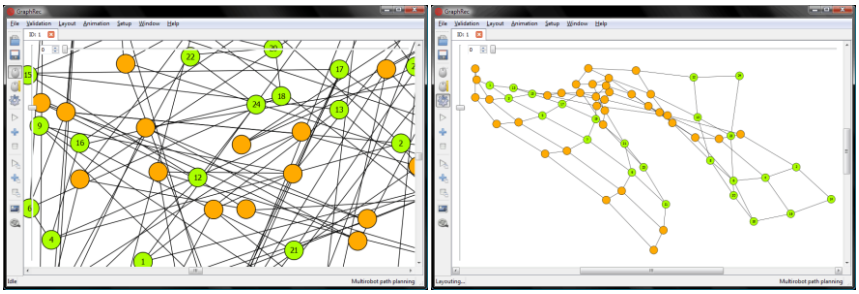
Hlavním úkolem nástroje je animovat pohyby entit v jednotlivých časových krocích řešení. Ovládání animace přitom musí být natolik vyspělé, aby výzkumníkovi umožňovalo pohodlné pozorování. To znamená, že musí nabízet krokování a rychlé přeskakování mezi libovolně vzdálenými časovými kroky. Při zkoumání konkrétního časového kroku může být také žádoucí zpomalení animace. Vysoké nároky jsou kladeny i na názornost animace. Pohyby entit je potřeba zvýrazňovat, aby nedošlo k jejich přehlédnutí. Měly by být rozlišitelné entity v cílových a necílových pozicích.

4.2 Popis funkcí vizualizačního nástroje

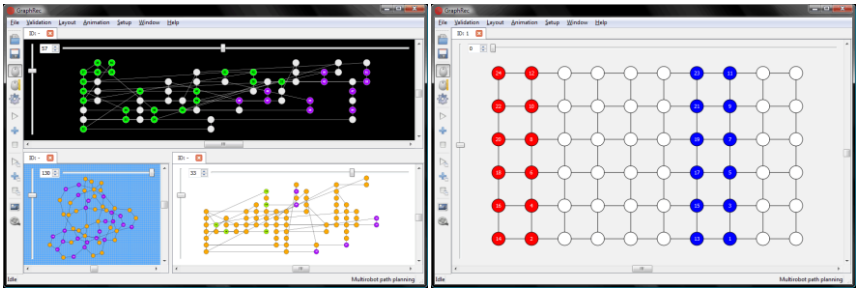
GraphRec pro **nakreslení grafu** používá modifikovanou verzi *silového algoritmu* [1], kde fyzikální interpretace grafu je taková, že vrcholy jsou považovány za odpuzující se tělesa a hrany za natažené pružiny mezi nimi. Algoritmus se díky své fyzikální povaze chová velmi intuitivně a umožňuje dostatečně rychlé interaktivní kreslení pro grafy obsahující desítky až stovky vrcholů. Průběh kreslení je ilustrován na obrázku 3. Protože automatické nakreslení nemusí ve všech případech stačit požadavkům uživatele, *GraphRec* umožňuje také pokročilou **manuální** úpravu pozic vrcholů.

Zvýšení přehlednosti je docíleno **obarvením** grafu a entit. Různými barvami lze zvýraznit sledované entity nebo rozlišit skupiny entit. Barevně jsou také odlišeny

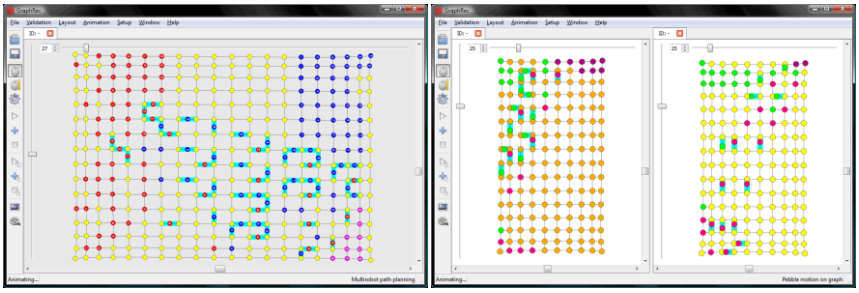
entity, které dosáhly své cílové pozice a již se dále nepohybují. Možnosti barevného zvýrazňování entit a grafu ilustruje obrázek 4.



Obr. 3. *Automatické rozvrhování pozic vrcholů (kreslení grafu)*. Levý obrázek ukazuje počáteční náhodné uspořádání. Na pravém obrázku je již graf znatelně více uspořádaný a začíná připomínat pravidelnou mřížku, ve které se nakonec ustálí.



Obr. 4. *Barvení entit a grafu*. Oba snímky ukazují míru, do jaké lze ovlivňovat obarvení jednotlivých částí grafu. Fialové entity na levém obrázku se nachází ve svých cílových pozicích, zelené entity se oproti tomu budou ještě dále pohybovat. Na pravém obrázku je zase vidět barevné odlišení dvou skupin entit, které si v průběhu řešení prohodí své pozice.



Obr. 5. *Animace pohybů entit*. Levý obrázek ukazuje, jak zvýrazňování pohybů přispívá ke zvýšení přehlednosti animace. Na pravém obrázku jsou srovnávány průběhy dvou různých řešení pro tentýž problém.

Vizualizace pohybu je řešena podobně jako přehrávání filmu ve video přehrávači. Uživatel nejprve specifikuje animační krok na časové ose a nastaví rychlost přehrávání. Animaci lze potom spouštět, pozastavovat nebo krokovat. Trajektorie entit je zdůrazněna podkladovou barvou. *GraphRec* dovoluje animovat pohyb na několika grafech zároveň, což umožňuje porovnání kvality dvou různých řešení téhož problému či zkoumání různé míry paralelismu. Průběh animace je ilustrován na obrázku 5.

4.3 Odhalení redundancí v produkovaných řešeních

Vizualizace řešení se ukázala jako účinný způsob, jak hledat v řešeních *redundance* **předem neznámé povahy**. Pokud totiž není známo, jak redundance vypadají, je těžké navrhnout postupy na jejich automatickou detekci a eliminaci. V prvotní fázi je tedy potřebná analýza člověkem. Pro lidské vnímání je nejvhodnější způsob prezentace řešení právě prostřednictvím vizualizace. Znalost vizuální podoby problému pomáhá výzkumníkovi pochopit kontext pohybů a rozhodnout o jejich přínosu či naopak kontraproduktivitě vzhledem k řešení jako celku.

4.4 Další využití vizualizačního nástroje

GraphRec dále nabízí funkci *validace*, která umožňuje ověřovat správnost řešení problému (viz. definice 1). Ověřování správnosti řešení je primárně určeno pro zajištění konzistence animace, jestliže vstupní řešení obsahuje pohyby, které odporují jeho definici. Funkce validace ale také představuje účinný prostředek pro **ladění** algoritmů pro řešení problémů pohybu po grafu.

GraphRec lze chápat i jako **prezentační** nástroj. Během animace je umožněno vytvářet kvalitní rastrové nebo vektorové snímky. Program také podporuje zachytávání animace do nejpoužívanějších video formátů. Obrázky a videa mohou doplňovat dokumentaci a prezentaci problémů pohybu po grafu například při výuce.

5 Opatření k odstranění redundancí v řešeních

Pomocí nástroje *GraphRec* bylo odhaleno několik druhů redundancí v rámci generovaných řešení. V následujících sekcích budou tyto redundance formálně popsány a budou navržena opatření na jejich odstranění. Pro uvažování o redundancích je vhodné pracovat s tvarem řešení, kde mezi dvojicí po sobě následujících časových kroků dochází právě k jednomu pohybu. Popsaný algoritmus *BIBOX-θ* produkuje řešení právě v této formě. Řešení tohoto tvaru se dá nahlížet jako posloupnost pohybů, kde pořadí pohybů odpovídá časovým krokům jejich zahájení.

Zápis $k_i: u_i \rightarrow v_i$ bude označovat pohyb kamenu k_i z vrcholu u_i do vrcholu v_i zahájený v časovém kroku i. Pohyb je netriviální, když $u_i \neq v_i$. Formálně je řešení posloupnost netriviálních pohybů $\Phi = [k_i: u_i \rightarrow v_i | i = 1,2,\dots,\xi - 1]$ (předpokládá se také soulad s definicí 1). Z prostorových důvodů budou tvrzení o korektnosti odstranění dále definovaných redundancí ponechána bez důkazu. Stejně tak bude bez důkazu konstatována výpočetní složitost algoritmů na eliminaci redundancí.

5.1 Odstranění inverzních pohybů

Definice 3 ***(inverzní pohyby)***. Pohyby $k_i: u_i \rightarrow v_i$ a $k_{i+1}: u_{i+1} \rightarrow v_{i+1}$ pro $i \in \{1,2,\dots,\xi - 2\}$ se nazývají *inverzní*, jestliže $k_i = k_{i+1}$, $u_i = v_{i+1}$ a $v_i = u_{i+1}$. □

Algoritmus 1. Odstraňování inverzních pohybů.

function *EraseInverseMoves* (Φ): **sequence**
do
  $\eta \leftarrow \emptyset$
  let $[k_1: u_1 \rightarrow v_1, k_2: u_2 \rightarrow v_2, \dots, k_{\xi-1}: u_{\xi-1} \rightarrow v_{\xi-1}] = \Phi$
  for $i = 1,2, \dots, \xi - 1$ **do**
    if $k_i: u_i \rightarrow v_i$ **a** $k_{i+1}: u_{i+1} \rightarrow v_{i+1}$ jsou inverzní **then**
      $\eta \leftarrow \eta \cup \{k_i: u_i \rightarrow v_i, k_{i+1}: u_{i+1} \rightarrow v_{i+1}\}$
  $\Phi \leftarrow \Phi - \eta$
while $\eta \neq \emptyset$
return Φ

Lze pozorovat, že dvojici inverzních pohybů je možné z řešení vynechat, aniž by toto vynechání mělo dopad na platnost řešení. Je však třeba si uvědomit, že odstranění dvojice inverzních pohybů z řešení může mít za následek vznik nové dvojice inverzních pohybů. Je tedy nutno inverzní pohyby z řešení odstraňovat až do okamžiku, kdy řešení žádné takové neobsahuje.

Formálně je postup vyjádřen algoritmem 1. Časová složitost algoritmu je $O(|\Phi|^2)$, prostorová složitost je $O(|\Phi|)$.

5.2 Vynechání redundantních pohybů

Definice 4 ***(redundantní pohyby)***. Posloupnost pohybů $[k_{i_j}: u_{i_j} \rightarrow v_{i_j} | j = 1,2, \dots, l]$, kde $I = [i_j \in \{1,2, \dots, \xi - 2\} | j = 1,2, \dots, l]$ je rostoucí posloupnost indexů, se nazývá *redundantní*, jestliže $|\{k_{i_j} | j = 1,2, \dots, l\}| = 1$, $u_{i_1} = v_{i_l}$ a pro všechny pohyby $k_\iota: u_\iota \rightarrow v_\iota$ takové, že $i_1 < \iota < i_l \wedge \iota \notin I$ platí $k_\iota \neq k_{i_1} \Rightarrow u_{i_1} \notin \{u_\iota, v_\iota\}$. □

Algoritmus 2. Odstraňování redundantních pohybů.

function *EraseRedundantMoves* (Φ): **sequence**
do
  $\eta \leftarrow$FindRedundantMoves(Φ)
  $\Phi \leftarrow \Phi - \eta$
while $\eta \neq \emptyset$
return Φ

function *FindRedundantMoves* (Φ): **sequence**
let $[k_1: u_1 \rightarrow v_1, \dots, k_{\xi-1}: u_{\xi-1} \rightarrow v_{\xi-1}] = \Phi$
for $i = 1,2, \dots, \xi - 2$ **do** {začátek redundantní}
  for $j = \xi - 1, \xi - 2, \dots, i + 1$ **do** {konec redund.}
    if $k_i = k_j \wedge u_i = v_j$ **then**
      $\eta \leftarrow \emptyset$ {redundantní posl.}
      for $\tau = i, i + 1, \dots, j$ **do**
        if $k_i = k_\tau$ **then** $\eta \leftarrow \eta \cup \{k_\tau: u_\tau \rightarrow v_\tau\}$
      if *CheckRedundantMoves*(Φ, i, j) **then return** η
return $\emptyset$

function *CheckRedundantMoves* (Φ, i, j): **boolean**
let $[k_1: u_1 \rightarrow v_1, \dots, k_{\xi-1}: u_{\xi-1} \rightarrow v_{\xi-1}] = \Phi$
for $\iota = i + 1, i + 2, \dots, j - 1$ **do**
  if $k_\iota \neq k_i \wedge u_i \in \{u_\iota, v_\iota\}$ **then return** $False$
return $True$

Redundantní pohyby představují jakési zobecnění inverzních pohybů (inverzní pohyby splňují definici redundantních pohybů). Jedná se o posloupnost pohybů, které daný kámen opětovně přesunou do stejného vrcholu, přitom ostatní pohyby řešení mezi prvním a posledním pohybem v posloupnosti nesmí do daného vrcholu zasahovat.

Vynechání redundantní posloupnosti pohybů z řešení toto řešení zachová. Podobně jako pro inverzní pohyby je nutno redundantní posloupnosti odstraňo-

vat opakovaně, neboť odstranění redundantní posloupnosti může mít za následek vznik jiné.

Formálně je postup vyjádřen jako algoritmus 2. Časová složitost algoritmu je $O(|\Phi|^4)$, prostorová složitost je $O(|\Phi|)$.

5.3 Nahrazování posloupností pohybů zkratkou

Definice 5 ***(dlouhá posloupnost)***. Posloupnost pohybů $[k_{i_j}: u_{i_j} \to v_{i_j} | j = 1,2,\dots,l]$, kde $I = \left[i_j \in \{1,2,\dots,\xi-2\} \middle| j = 1,2,\dots,l\right]$ je rostoucí posloupnost indexů, se nazývá *dlouhá*, jestliže $|\{k_{i_j} | j = 1,2,\dots,l\}| = 1$ a v grafu G existuje cesta $C = [c_1 = u_{i_1}, c_2, \dots, c_n = v_{i_l}]$ taková, že $n < l$ a pro všechny pohyby $k_\iota: u_\iota \to v_\iota$ takové, že $i_1 < \iota < i_l \wedge \iota \notin I$ platí $k_\iota \neq k_{i_1} \Rightarrow \{u_\iota, v_\iota\} \cap C = \emptyset$. □

Algoritmus 3. Nahrazování dlouhých posloupností.

```
function ReplaceLongMoves (Φ, G): sequence
1: do
2:     (η, π) ←FindLongMoves(Φ, G)
3:     Φ ← Φ − η; Φ ← Φ ∪ π
4: while (η, π) ≠ (∅, [])
5: return Φ

function FindLongMoves (Φ, G): pair
6: let [k_1: u_1 → v_1, ..., k_{ξ−1}: u_{ξ−1} → v_{ξ−1}] = Φ
7: for i = 1,2, ..., ξ − 2 do
8:     for j = ξ − 1, ξ − 2, ..., i + 1 do
9:         if k_i = k_j then
10:            η ← ∅
11:            for τ = i, i + 1, ..., j do
12:                if k_i = k_τ then η ← η ∪ {k_τ: u_τ → v_τ}
13:            C ←CheckLongMoves(Φ, i, j, |η|, G)
14:            if C ≠ [] then
15:                let [c_1, c_2, ..., c_n] = C
16:                π ← [k_i: c_1 → c_2, ..., k_i: c_{n−1} → c_n]
17:                return (η, π)
18: return (∅, [])

function CheckLongMoves (Φ, i, j, l, G = (V, E)): sequence
19: let [k_1: u_1 → v_1, ..., k_{ξ−1}: u_{ξ−1} → v_{ξ−1}] = Φ
20: (V', E') ← G
21: for ι = i + 1, i + 2, ..., j − 1 do
22:     if k_ι ≠ k_i then
23:         V' ← V' − {u_ι, v_ι}
24:         E' ← E' ∩ {{u, v} | u, v ∈ V'}
25: let C je nejkratší cesta z u_i do v_j v G' = (V', E')
26: if C je definováno and |C| < l then return C
27: return []
```

Dlouhá posloupnost pohybů je opět jakýmsi dalším zobecněním posloupnosti redundantních pohybů (redundantní posloupnost pohybů splňuje definici dlouhé posloupnosti s tím, že za cestu C je volena prázdná posloupnost vrcholů). Intuitivně je dlouhá posloupnost pohybů taková, že ji lze nahradit posloupností pohybů podél kratší cesty (zkratky), do které ale nesmí zasahovat ostatní pohyby mezi začátkem a koncem dlouhé posloupnosti.

Platí, že posloupnost pohybů vzniklá nahrazením dlouhé posloupnosti posloupností pohybů podél zkracovací cesty C je opět řešením daného problému pohybu kamenů po grafu. Z algoritmického hlediska je třeba větší pozornost věnovat hledání cesty C v definici. Je-li fixován začátek a konec posloupnosti, která je ověřována podle definice, lze postupovat tak, že vrcholy z ostatních pohybů mezi začátkem a

koncem posloupnosti jsou z grafu G dočasně vypuštěny a hledá se nejkratší cesta v takto modifikovaném grafu. Stejně jako v předchozích případech je nutno mít na zřeteli, že nahrazením dlouhé posloupnosti může v řešení vzniknout jiná dlouhá posloupnost, která se předtím nevyskytovala. Proces náhrady je tedy nutné provádět opakovaně.

Formálně je postup nahrazování dlouhých posloupností pohybů vyjádřen jako algoritmus 3. Časová složitost algoritmu je $O(|\Phi|^4 + |\Phi|^3|\mathrm{V}|^2)$, prostorová složitost je $O(|\Phi| + |\mathrm{V}| + |\mathrm{E}|)$.

5.4 Shrnutí metod na odstraňování redundancí

Uvedené formalizace redundancí, jež byly objeveny za pomoci vizualizačního nástroje *GraphRec*, představují postupná **zobecnění**. Bylo by tedy možné hovořit rovnou o nahrazování dlouhých posloupností, které zahrnuje obě předchozí opatření. Ovšem vzhledem k rostoucí výpočetní složitosti algoritmů realizující odstranění obecnějších redundancí toto není vhodné. Výhodnější je řešení vyprodukované řešícím algoritmem nejprve podrobit analýze na inverzní pohyby, což může řešení zkrátit, poté v modifikovaném řešení hledat redundantní posloupnosti pohybů a až nakonec se výpočetně nejnáročnějším algoritmem pokoušet o nahrazování dlouhých posloupností v již potenciálně zkráceném řešení.

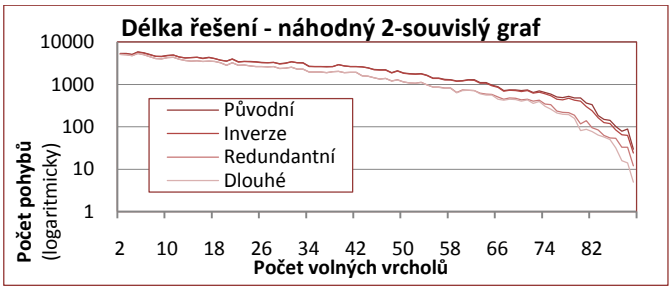
6 Experimentální výsledky

Navržená opatření na odstranění redundancí byla experimentálně vyhodnocena. Algoritmy 1, 2 a 3 byly implementovány v C++ a byla navržena testovací sada problémů pohybu kamenů po grafu. Řešení problémů nalezená pomocí algoritmu *BIBOX-θ* byla poté podrobena popsaným opatřením na odstranění redundancí. Sledovanou veličinou přitom bylo zkrácení řešení oproti původnímu, pro orientaci byl sledován i čas běhu algoritmů. Opatření na zkrácení řešení byly aplikovány tak, jak je popsáno v sekci 5.4. Z prostorových důvodů bude uveden jen fragment experimentálních výsledků.

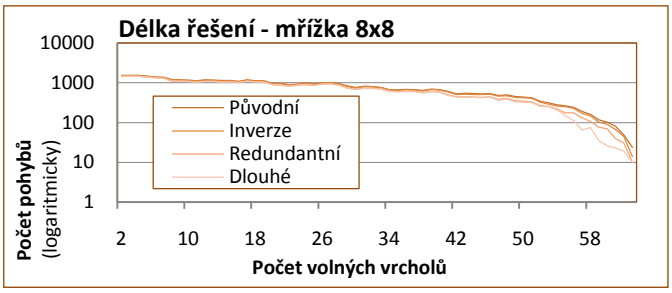
První prezentovaná sada problémů používala náhodně vygenerovaný 2-souvislý graf o 90 vrcholech. Náhodný 2-souvislý graf byl zkonstruován postupným přidáváním uch náhodné velikosti z rovnoměrného rozdělení $2..10$ ke kružnici velikosti 7. Počáteční a cílové rozložení kamenů bylo zvoleno jako náhodná permutace.

Druhá sada problémů byla založena na grafu, jehož nakreslení odpovídá mřížce o velikosti 8×8. Počáteční a cílové rozložení kamenů v mřížce bylo opět zvoleno jako náhodná permutace. V obou případech byly náhodné permutace zkonstruovány prováděním kvadratického počtu transpozic.

Zkrácení řešení bylo zkoumáno v závislosti na počtu volných vrcholů. Výsledky experimentů jsou ukázány na obrázcích 6 a 7. Dá se říci, že pro náhodný 2-souvislý graf lze pomocí odstranění redundancí zkrátit řešení až na polovinu. Pro mřížku bylo získáno zkrácení zhruba o 10%. Výsledky ukazují, že největší přínos má odstraňování redundantních posloupností pohybů. Naopak odstraňování inverzních pohybů a provádění nahrazování dlouhých posloupností se ukazuje jako nepříliš přínosné.



Obr. 6. *Srovnání účinnosti opatření na odstranění redundancí na náhodných 2-souvislých grafech*. Pro více volných vrcholů lze dosáhnout zkrácení řešení až o polovinu.



Obr. 7. *Srovnání účinnosti opatření na odstranění redundancí na mřížce 8x8*. Dosahuje se zkrácení řešení zhruba o 10%.

Ačkoli efektivní implementace algoritmů pro odstraňování redundancí a jejich časové srovnání není náplní této práce, byl v rámci experimentů orientačně měřen i čas. Na testovaných problémech byl čas na odstranění inverzních pohybů neměřitelně malý, odstraňování redundantních posloupností trvalo řádově vteřiny (což je srovnatelné s tím, jak dlouho běží samotný algoritmus BIBOX-θ) a nahrazování dlouhých posloupností trvalo řádově minuty (měření probíhalo na počítači s procesorem Pentium 4, 2Ghz).

Společně s výsledky ohledně zkrácení řešení se tedy jako prakticky nejpoužitelnější metoda jeví eliminace redundantních posloupností pohybů. Dále také výsledky ukazují, že většího zkrácení řešení je dosahováno v situaci, kdy graf obsahuje více volných vrcholů (konkrétně na mřížkovém grafu lze patrné zlepšení pozorovat, až když je alespoň polovina vrcholů neobsazených). To však není překvapivé, neboť definice odstranitelných redundancí se podstatně opírá o vzájemnou neinterferenci pohybů kamenů, která je u problémů s omezeným neobsazeným prostorem malá.

7 Shrnutí a závěr

Předmětem studia v této práci byla kvalita (délka) řešení problémů pohybu po grafu. Konkrétně bylo zkoumáno, zda řešení vygenerovaná pomocí existujících algoritmů neobsahují jisté redundance a jaké povahy tyto redundance jsou. Pro odhalení těchto redundancí byl navržen a implementován vizualizační nástroj *GraphRec*.

Pomocí nástroje *GraphRec* byly odhaleny redundance několika druhů. V této práci byly redundance formálně popsány a byly navrženy algoritmy na jejich odstranění. Experimentálně pak byl prokázán přínos odstranění zejména jednoho typu redundance. Byla rovněž objevena charakterizace, že ke značnému zkrácení řešení odstraněním redundancí dochází zejména v případě, kdy graf problému obsahuje více volných vrcholů. Provedené experimenty ukázaly zkrácení původního řešení až o polovinu.

Reference

1. A. Bilgin, J. Ellson, E. Gansner, Y. Hu, Y. Koren a S. North. *Graphviz - Graph Visualization Software.* Webová stránka projektu, http://www.graphviz.org, (září 2009).
2. D. Kornhauser, G. L. Miller a P. G. Spirakis. *Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications.* Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS 1984), 241-250, IEEE Press, 1984, ISBN 0-8186-0591-X.
3. P. Koupý. *GraphRec - a visualization tool for entity movement on graph.* Webová stránka studentského projektu, http://www.koupy.net/graphrec.php, (září 2009).
4. D. Ratner a M. K. Warmuth. *Finding a Shortest Solution for the N×N Extension of the 15-PUZZLE Is Intractable.* Proceedings of the 5th National Conference on Artificial Intelligence (AAAI 1986), 168-172, Morgan Kaufmann Publishers, 1986.
5. M. R. K. Ryan. *Graph Decomposition for Efficient Multi-Robot Path Planning*. Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), 2003-2008, IJCAI Conference, 2007.
6. P. Surynek. *A Novel Approach to Path Planning for Multiple Robots in Bi-connected Graphs*. Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009), 3613-3619, IEEE Press, 2009, ISBN 978-1-4244-2789-5.
7. P. Surynek. *An Application of Pebble Motion on Graphs to Abstract Multi-robot Path Planning.* Proceedings of the 21st International Conference on Tools with Artificial Intelligence (ICTAI 2009), IEEE Press, 2009, přijato k tisku.
8. R. E. Tarjan. *Depth-First Search and Linear Graph Algorithms.* SIAM Journal on Computing, Volume 1 (2), 146-160, Society for Industrial and Applied Mathematics, 1972.
9. K. C. Wang, A. Botea. *Tractable Multi-Agent Path Planning on Grid Maps.* Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), 1870-1875, IJCAI Conference, 2009.
10. R. M. Wilson. *Graph Puzzles, Homotopy, and the Alternating Group.* Journal of Combinatorial Theory, Ser. B 16, 86-96, Elsevier, 1974.

Annotation:

Visualization as a tool for acquiring knowledge about the quality of solutions of problems of motion on graphs

The quality (length) of solutions of problems of motion on graphs is addressed in this paper. Existing state-of-the-art algorithms for generating solutions of these problems are suspected of producing solutions containing redundancies of a priori unknown nature. A visualization tool has been developed to discover such redundancies. Knowledge about solutions acquired by the tool served as basis for the formal description of redundancies and for the development of methods how to detect and eliminate them.