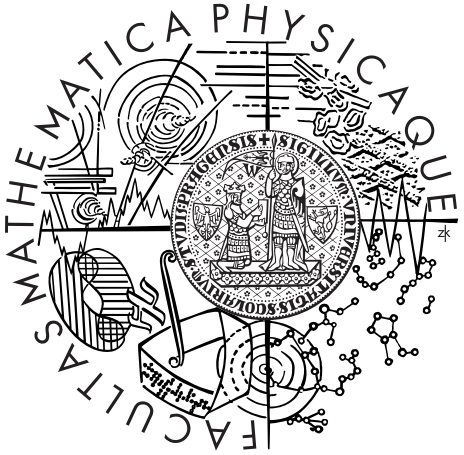
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Miloš Chromý

Vylepšení algoritmu BIBOX

Katedra teoretické informatiky a matematické logiky

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

Praha 2013

Rád bych poděkoval vedoucímu této práce, kterým byl RNDr. Pavel Surynek, Ph.D., za odborné vedení a za čas, který mi věnoval, rodičům a kamarádům, za toleranci, trpělivost, kterou mě neustále obdařují.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval 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: Vylepšení algoritmu BIBOX

Autor: Miloš Chromý

Katedra: Katedra teoretické informatiky a matematické logiky

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

Abstrakt: Algoritmus BIBOX je z rodiny algoritmů kooperativního hledání cest. Tento algoritmus je rychlý a suboptimální. I přes svou rychlost obsahuje několik nedeterministických rozhodnutí. Jeden z nedeterminismů je rozebrán v této práci a je jím předzpracování grafu na ucha. Cílem je pomocí analýzy algoritmu získat různá rozložení grafu, která budou mít různé vlivy na běh a výsledek algoritmu, experimentálně vyhodnotit výkon algoritmu na zvolených rozloženích a nalézt rozložení grafu na ucha, na kterém bude algoritmus nejvýkonnější.

Klíčová slova: kooperativní hledání cest, BIBOX

Title: Improvement of the BIBOX Algorithm

Author: Miloš Chromý

Department: Department of Theoretical Computer Science and Mathematical Logic

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

Abstract: The algorithm BIBOX is one from cooperative pathfinding's algorithms family. This algorithm is fast and suboptimal. Despite its speed, it has several non-deterministic decisions. One of the non-determinism is analyzed in this work and it's graph handle decomposition. The goal is to get different graph handle decomposition by analysis algorithm, which will have different effects on the running time and size of results of the algorithm, experimentally evaluate the performance of the algorithm on the selected handle decompositions and find handle decomposition, on which will be algorithm most efficient.

Keywords: cooperative path-finding, BIBOX

Obsah

Úvod

Budeme se zabývat kooperativním hledáním cest. Hledání cest je otevřený problém, který je řešen v umělé inteligenci, teoretické robotice, ale též u různých strategických her. Kooperativní hledání cest vytvoří posloupnost kroků pro každého robota z nějaké skupiny robotů, kde každý robot má nějaký cíl kam chce dojít. Navíc se žádní dva roboti nesmí srazit ani nacházet na jednom místě zároveň. U kooperativního hledání cest se předpokládá, že každý robot má znalost celého prostředí, ve kterém se pohybuje. Dále má též znalost o pozici všech ostatních robotů.

Existuje mnoho aplikací kooperativního hledání cest. Kupříkladu pohyb jednotek ve strategické hře, kde jednotky musí odhadovat pohyb ostatních jednotek. Tento problém je zajímavý hlavně u různých úzkých průchodů, popřípadě mostů. Dalším příkladem může být pohyb robotů a přesun zboží ve skladu, kde kvůli optimalizaci máme méně volného místa pro pohyb.

Hledání cest pro více agentů lze roztřídit buď podle úrovně spolupráce robotů na

- *Blokující hledání cest*, kde se roboti snaží dojít do svého cíle, ale zároveň se snaží ostatním robotům zabránit ve splnění jejich úkolu
- *Nekooperativní hledání cest*, kde se roboti snaží dojít do svého cíle a neznají své cíle navzájem, a tudíž musí tipovat pohyb ostatních robotů
- *Kooperativní hledání cest*, kde roboti znají pozici i plány všech ostatních robotů navzájem

nebo podle přístupu plánování na

- *centralizované*, kde cesty plánuje nějaký plánovač a může nalézt všechny možné plány,
- *distribuované*, kde se plánování rozdělí na nezávislé nebo slabě závislé podproblémy každého robota. Každý robot si pak může hledat cestu do cíle se znalostí pozice všech ostatních. robotů.

V této práci se zabýváme kooperativním hledáním cest. Tímto problémem se zabývá mnoho různých algoritmů s odlišnými přístupy a též s odlišnými výhodami a nedostatky.

Nyní si roztřídíme algoritmy kooperativního hledání cest do několika typů

1. Prohledávací algoritmy
2. Suboptimální algoritmy
3. Převod na problém jiný

Prohledávací algoritmy

Významnou třídou algoritmů, jsou algoritmy prohledávací. Následující algoritmy jsou algoritmy distribuované.

Hlavním zástupcem těchto algoritmů je algoritmus A*[6]. Protože tento algoritmus slouží pro vyhledání cesty jednoho robota do svého cíle, pro více robotů musíme použít rozšíření, které řeší kolize[9]. Tato modifikace a podobné jsou příkladem nekooperativního hledání, a mají nevýhodu v tom, že v obtížnějších a složitějších prostředích mohou nastat patové situace, či různá zacyklení. Tyto nedostatky se často projeví u dlouhých úzkých pasáží, kde hrozí riziko vytvoření přeplněného zúženého průchodu.

Některé nevýhody jsou řešeny dalšími vylepšeními algoritmu A*[6], které jsou narozdíl od předchozích algoritmů kooperativní.

Jsou to vylepšení

- Local Repair A*[7]
- Cooperative A*[7]
- Hierarchical Cooperative A*[7]
- Windowed Hierarchical Cooperative A*[7]

Local Repair A*[7]

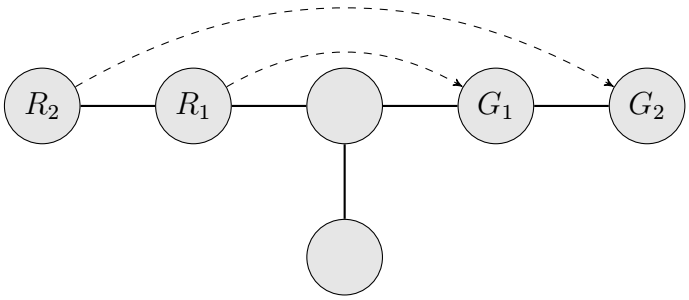
U algoritmu Local Repair A* si každý agent nalezne cestu do svého pomocí algoritmu A*[6] a ignoruje všechny roboty, kteří se právě nenachází v přilehlých pozicích. Poté se všichni roboti vydají po naplánovaných cestách, dokud se neocitnou v nebezpečné situaci(dva roboti se v dalším kroku budou nacházet na stejné pozici nebo se minou po jedné cestě). V ten moment si robot, který by se ocitl v kolizní pozici, přepočítá zbytek cesty do cíle, opět pomocí A*[6].

Pokud se vyskytne někde zúžený průchod v kombinaci s příliš mnoha roboty, dojde v každém tahu přepočítání trasy všech robotů, což může vést k tomu, že řešení bude zbytečně dlouhé. V nejhorším případě může nastat i zacyklení robotů.

Cooperative A*[7]

Problémy Local Repair A* se snaží vyřešit Coopertive A*. Tento přístup přidává robotovi možnost počkat na místě, tedy za časovou jednotku vytrvat na pozici, která nemusí být jeho cílem. Dále vytváří tří-dimenzionální tabulku rezerv, kde si každý robot zarezervuje cestu. Tato tabulka rezerv je sdílenou pamětí všech robotů. Vytvořená rezervace v tabulce je v daný časový úsek neprůchodná a roboti, kteří si nestihli naplánovat cestu nesmí rezervací projít.

Existují třídy problémů, které nelze vyřešit, jako třeba problém z obrázku 1. Robot R_1 se dostane do svého cíle G_1 a odtud už neuhne, tudíž robot R_2 se nemůže dostat na svou pozici G_1.



Obrázek 1: Problém Cooperative A*

Hierarchical Cooperative A*[7]

Hierarchií budeme chápat hierarchii z článku[8]. Budeme ji tedy chápat jako různé úrovně abstraktních stavů. Tyto stavy nejsou popsány pouze svou pozicí v prostředí, ve kterém hledáme cesty, ale může mít další aspekty. Hierarchie nemusí být příliš rozsáhlá. Dokonce menší hierarchie v původním Hierarchical A*[8] jsou vhodnější a rychlejší než rozsáhlé hierarchie.

Hierarchie u Hierarchical Cooperative A* používá jednu hierarchii obsahující pouze jednu oblast abstrakce. Tato abstrakce je pouze dvou-dimenzionální mapa prostředí bez robotů, tudíž ignoruje čas a rezervační tabulku. Tato mapa je ideální pro odhad vzdálenosti do cíle, bez ohledu na roboty. Pro znovupoužití již vypočtených vzdáleností z abstraktní mapy je v Hierarchical Cooperative A* použit Reverse Resumable A*[7].

Hierarchical Cooperative A* je velice podobný Cooperative A*, akorát používá chytřejší heuristiky, která používá Reverse Resumable A* na přepočtení abstraktních vzdáleností v mapě. Pokud je nejkratší cesta do cíle volná, pak je celá cesta obsažena v mapě. Pokud jsou na cestě do cíle nějací roboti, pak se musí přepočítat abstraktní tabulka pomocí Reverse Resumable A*.

Tento přístup vylepšuje Cooperative A*. Avšak třídy neřešitelných problémů jsou stejné.

Windowed Hierarchical Cooperative A*[7]

Tento přístup řeší některé nedostatky předchozích modifikacích A*.

Kupříkladu pokud robot dojde do svého cíle, který se nachází v úzkém koridoru, tak může zablokovat jedinou cestu k cíli jiným robotům. Dalším je upřednostňování robotů při výběru cesty a při samotném pohybu. Není úplně vhodné řešení upřednostňovat roboty globálně. Při pořadí robotů fixním a neměnitelném mohou být některé úlohy neřešitelné. Významným nedostatkem je i to, že robot musí svou cestu spočítat až do cílové pozice v komplexních tří-dimenzionálních tabulkách. Takto si naplánují cestu, která bývá často nevyužita, kvůli přepočtům při kolizi.

Přidáním okénka můžeme některé z těchto nedostatků zcela omezit, jiné alespoň zčásti. Toto okénko je nastaveno na nějaký rozměr $d \times d$. Uvnitř okénka se používá kooperativního přístupu(Hierarchical Cooperative A*, ale lze použít i jiný algoritmus). Aby robot šel správným směrem, má vypočtenou abstraktní cestu, což je nejkratší cesta do cíle. Směr je pak dán průnikem této cesty a okénka. Tím se šetří zdroje.

Poté co robot dojde do svého cíle, nezůstává stát, ale dává si za úkol dojít ke kraji okénka. Toto je zabezpečení proti tomu, aby nezablokoval cestu jiným robotům tím, že po nalezení svého cíle obsadí úzký koridor a nehne se z něj.

Suboptimální algoritmy

Suboptimální algoritmy mají výhodu v tom, že řešení je nalezeno velice rychle. Suboptimální algoritmy jsou kupříkladu

- Push and Swap[10]
- Push and Rotate[11]
- BIBOX[1]

Tyto algoritmy jsou schopny vyřešit prostředí, kde je až $V - 2$ robotů.

Push and Swap

Algoritmus Push and Swap[10] funguje na velice jednoduchém principu. Robot si najde cestu(nejlépe nejkratší) do své cílové pozice. Pokud na následující pozici není jiný robot, provede operaci *Push*, tedy pokročí o krok blíže k cíli. Pokud na dalším poli stojí robot, provede operaci *Swap*. Po dokončení této operace by měli všichni roboti, až na dva, kteří se vyměňují být na svých místech, na kterých stáli před operací. Funkce *Swap* najde nejbližší vhodný vrchol pro výměnu. Takový vrchol v musí splňovat následující podmínky

- musí být dosažitelný pro oba vrcholy, které chceme vyměnit operacemi *Push* a *Swap*
- stupně alespoň 3
- po přesunu obou vrcholů k vrcholu v musí být proveditelné funkce *Multipush*, respektive *Clear*, které zajistí uvolnění dvou vrcholů sousedících s vrcholem v

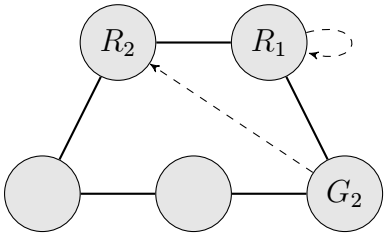
Operace *Multipush*, respektive *Clear*, zde nebudeme popisovat. Dále ještě existuje operace *ResolveOperation*, která zaručí navrácení vrcholu u na své původní místo, pokud provádíme operaci *Swap* s vrcholem u, který je již ve své konečné pozici.

Volba pořadí přesunů robotů není algoritmem specifikována.

Push and Rotate

Tento algoritmus vychází přímo z algoritmu Push and Swap[10], poněvadž algoritmus není kompletní. První problém u Push and Swap nastává u polygonu(cyklu) délky l. Příklad je uveden na obrázku 2. Robot R_1 je již ve svém cíli. Robot R_2 se snaží dostat do svého cíle G_2, ale nejkratši cesta vede, skrze vrchol obsazený robotem R_1, kterého ale nemůžeme posunout(vrchol R_1 má vyšší prioritu), ani se s ním vyměnit(nenachází se v grafu žádný vrchol, splňující podmínku pro výměnu).

Dále ještě doplňuje funkci *Clear* o jeden případ, který v algoritmu Push and Swap není popsán. Dále ještě řeší případ rekurzivního volání funkce *Reslove* a případ dvou dvasouvislých grafů spojených delším mostem. Tyto problémy řeší rozšířením stávajících funkcí a přidáním nové funkce *Rotate*.



Obrázek 2: Polygon délky pět

BIBOX

Algoritmus BIBOX[1] pracuje s grafem, který dva-souvislý. Takovému grafu existuje nějaké rozložení na ucha. Tento algoritmus postupně řeší jednotlivá ucha. Pokud je robot, který je právě zpracováván na právě zpracovávaném uchu, algoritmus přesune tohoto robota mimo zpracovávané ucho, aniž by zpřeházel již seřazené umístěné vrcholy na uchu. Takto uspořádá všechny ucha až na poslední, které tvoří cyklus. Na tomto uchu pak seřadí roboty do správného pořadí pomocí funkce na prohození sousedních vrcholů.

Převedení na jiný problém

Kooperativní hledání cest lze převést do

- celočíselného lineárního programování
- SATu

Převod do celočíselného lineárního programování

Tento převod je proveden tak, že nejdříve převedeme problém kooperativního hledání cest na multitoky v sítích[13]. Pro převod je nutné, aby graf byl bezkolizní jednotkový graf. Bezkolizní jednotkový graf G je graf, který splňuje následující podmínky[13]

- G je spojitý
- G je neorientovaný
- každá hrana v G má jednotkovou délku
- libovolné dvě hrany (u_1, v_1) a (u_2, v_2) v G, kde $u_1 \neq u_2$,$v_1 \neq v_2$, dva roboti tvaru kola s průměrem menším než $\frac{\sqrt{2}}{4}$ putujícími jednotkovou rychlostí po těchto hranách(startujícími zároveň v u_1, respektive u_2) do sebe nenarazí

Problém multitoků v sítích lze přímo převést na celočíselené lineární programování[12].

Převod do SATu

Jedním přísupem je převést celý problém do SATu. Tento přístup je popsán v článku[16].

Jinak můžeme řešit převodem do SATu jen nějaké podproblémy celého plánování, jako je v článku[3]. Tento přístup začíná se suboptimálním základním řešením, které pak rozdělí na menší části, které řeší převodem na SAT. V článku[3] je jako základní řešení použito řešení z algoritmu BIBOX[1]. Převod pak optimalizuje řešení z BIBOXu[1].

Téma práce

Algoritmus BIBOX[1] je kompletní algoritmus ke kooperativnímu hledání cest. Zároveň patří mezi rychlé algoritmy, které hledají suboptimální řešení. Toto řešení lze zároveň optimalizovat pomocí převodu do SATu[3]. Proto se v této práci budeme zabývat zrychlením algoritmu BIBOX[1]. Toho docílíme tím, že se budeme zabývat podrobnou analýzou algoritmu BIBOX[1] a pokusíme se zjistit, jaký vliv má různé předzpracování dat na výkonnost algoritmu a na velikost jeho řešení. Nejprve představíme podrobně algoritmus Pak popíšeme zvolené způsoby předzpracování vstupních dat a jejich vliv na výkonnost algoritmu, tedy čas, potřebný k nalezení řešení a počet kroků, potřebných k vyřešení problému. Poté budeme prezentovat výsledky o experimentálním vyhodnocení. Nakonec ještě prozkoumáme samotné předzpracování grafu.

1. Teoretický model a testovací program

Nejdříve provedeme teoretickou analýzu algoritmu BIBOX[1]. Pseudokód algoritmu je převzat z článku[1]. V pseudokódu jsou udělány drobné úpravy, které neovlivňují jeho funkčnost.

1.1 Anazlýza algoritmu

Mějme množinu robotů R a 2-souvislý graf $G = (V, E)$ a jeho rozklad na ucha $C_0, L_1, L_2 \ldots, L_k$, kde C_0 je úvodní cyklus a $L_1, L_2 \ldots, L_k$ jsou jednotlivá ucha rozkladu taková, že $\forall j = 1, 2, \ldots, k : G \setminus \bigcup_{i=j}^{k} L_i$ je 2-souvislý. $C(L_j) = L_j + P_j$ je cyklus takový, že $C(L_j) \subseteq \bigcup_{i=0}^{j} L_i \wedge P_j \subseteq \bigcup_{i=0}^{j-1} L_i$. Každý vrchol $v \in V(G)$ má jednoznačnou příslušnost k některému uchu L_j. Tuto příslušnost vyjádříme funkcí $\Gamma : V(G) \to \{C_0, L_1, L_2, \ldots, L_k\}$.

Mějme dále funkce S, Φ. Funkce $S : R \to V$ popisuje pozice jednotlivých robotů a funkce $\Phi : V \to R \cup \perp$ je rozšířená inverzní funkce k funkci S o symbol $\perp$. Platí $\forall r \in R : \Phi(S(r))r; \Phi(v) = \perp$ pokud $\forall r' = R : S(r) \neq v$. Tyto funkce mají své varianty S_0, Φ_0 určující počáteční vztahy robotů a vrcholů a S_+, Φ_+ určující jejich vztahy cílové.

Poslední dvojici funkcí, které budeme potřebovat, jsou funkce $prev/S(C, r)$ a $next/S(C, r)$. Tyto funkce nám určují vrchol předcházejícího, či následujícího robota r v cyklu C.

Nyní můžeme přistoupit k analýze algoritmu BIBOX[1]. K tomu budeme potřebovat jeho zjednodušený pseudokód.

Algorithm 1.1 BIBOX

```
function BIBOX-solve
1: for c = k, k − 1, . . . , 1 do
2:    if |L_c| > 2 then
3:       SolveRegularCycle(c)
4:    end if
5: end for
6: SolveOriginalCycle
```

Vzhledem k tomu, že předpokládáme dva prázdné vrcholy na úvodní kružnici, nebudeme analyzovat funkci *SolveOriginalCycle*, poněvadž počet kroků i čas na vyřešení jsou zanedbatelné. Nyní bychom si měli rozebrat funkci *SolveRegularCycle*. Ale nejdříve si popíšeme funkce, které jsou použity v *SolveRegularCycle*. Funkci *SolveRegularCycle* podrobíme analýze jako poslední. Výsledný počet kroků bude součet kroků pro jednotlivá ucha. Tento součet vyhodnotíme též na konci.

Funkce *SwapRobotUnoccupied(u,v)* jen posune robota z vrcholu u na prázdný vrchol v, tedy provede konstantně kroků. U výpočtu počítáme se třemi kroky. Počet kroků funkce *FindShortesPath(u,v)* pro $u \in L_i$ a $v \in L_j$ označme $F_{max(i,j)}$. Hodnotu $F_{max(i,j)}$ a tedy i na způsob hlednání nejkratších cest v grafu si popíšeme později.

Funkce 1.2 Pootočení cyklu o jedna doprava(doleva)

function RotateCycle$^{+(-)}(C)$

1: **let** $x \in C : \Phi(x) = \perp \wedge x$ is not locked
2: **for** $i = 1, 2, \ldots, |C|$ **do**
3: SwapRobotUnoccupied($prev(next)/V(C, x), x$)
4: $x \leftarrow prev(next)/V(C, x)$
5: **end for**

Funkce *RotateCycle* zavolá $C(L_j)$-krát funkci *SwapRobotUnoccupied*, provede tedy $(L_j + P_j)$ kroků. V dalších výpočtech předpokládáme, že délka P_j je maximálně $\left(V - \sum_{i=j}^{k} L_i\right)$.

Funkce 1.3 Uvolnění vrcholu v

function MoveUnoccupied(v)

1: **let** $x \in V : \Phi(x) = \perp \wedge x$ is not locked
2: **let** $[x = p_1, p_2, \ldots, p_j = v]$ shortest path $\wedge p_i$ is not locked $\forall i$
3: **for** $i = 1, 2, \ldots, j - 1$ **do**
4: SwapRobotUnoccupied(p_{i+1},p_i)
5: **end for**

Pro cestu mezi vrcholem $v : v \in L_j$, pro nějaké j, a vrcholem $x : \Phi(x) = \perp$ předpokládáme cestu maximální délky $V \setminus \bigcup_{i=j}^{k} L_i$. Dále označme F_j počet kroků pro nalezení nejkratší cesty. Funkce *MoveUnoccupied* provede $|V \setminus \bigcup_{i=j}^{k} L_i|$-krát funkci *SwapRobotUnoccupied* a jednou funkci *FindShortestPath*, tedy provede $|V \setminus \bigcup_{i=j}^{k} L_i| + F_j$ kroků.

Funkce 1.4 Přesun robota r do vrcholu v

function MoveRobot(r,v)

1: **let** $[S(r) = p_1, p_2, \ldots, p_j = v]$ shortest path $\wedge p_i$ not locked
2: **for** $i = 1, 2, \ldots, j - 1$ **do**
3: lock($\{p_i\}$)
4: MoveUnoccupied(p_{i+1})
5: unlock($\{p_i\}$)
6: SwapRobotUnoccupied(p_{i+1},p_i)
7: **end for**

Nejkratší cestu mezi $S(r)$ a v budeme uvažovat stejnou jako u funkce *MoveUnoccupied*. Pak tedy funkce *MoveRobot* provede $|V \setminus \bigcup_{i=j}^{k} L_i|$-krát funkce *MoveUnoccupied* a *SwapRobotUnoccupied* a jednou provede funkci *FindShortestPath*. Výsledný počet kroků této funkce bude $|V \setminus \bigcup_{i=j}^{k} L_i|(|V \setminus \bigcup_{i=j}^{k} L_i| + F_j) + F_j$.

Funkce 1.5 Umístění všech robotů na uchu L_c

function SolveRegularCycle(c)

1: **let** $[u, x_1, x_2, \ldots, x_l, v] = L_c$
2: **for** $i = 1, 2, \ldots, l$ **do**
3: **if** $\Gamma(S(\Phi^+(x_i))) = L_c$ **then**
4: lock(L_c)
5: MoveUnoccupied(u)
6: ulock(L_c)
7: $\rho \leftarrow 0$
8: **while** $S(\Phi^+(x_i)) \neq v$ **do**
9: RotateCycle$^+$($C(L_c)$)
10: $\rho \leftarrow \rho + 1$
11: **end while**
12: lock(L_c)
13: **let** $o \in V \setminus \{\bigcup_{i=c}^{k} L_i \cup C(L_c)\}$
14: MoveRobot($\Phi^+(x_i, o)$)
15: lock($\{o\}$)
16: MoveUnoccupied(u)
17: unlock(L_c)
18: **while** $\rho > 0$ **do**
19: RotateCycle$^-$($C(L_c)$)
20: $\rho \leftarrow \rho - 1$
21: **end while**
22: unlock($\{o\}$)
23: **end if**
24: lock(L_c)
25: MoveRobot($\Phi^+(x_i), u$)
26: MoveUnoccupied(v)
27: unlock(L_c)
28: RotateCycle$^+$($C(L_c)$)
29: lock(L_c)
30: **end for**

U funkce *SolveRegularCycle* budeme předpokládat, že vrchol leží na uchu L_c. Dále předpokládáme, že po proběhnutí whilecyklu na řádkách 8–11 bude $\rho = |L_c|$, tedy $C(L_c)$ pootočíme právě $|L_c|$-krát. Počet kroků pro *lock* i *unlock* je stejný a počet kroků *lock*(A) je velikost množiny A.

Pro každého robota na uchu L_c provedeme sedmkrát (*un*)*lock*(L_c), dvakrát provedeme cyklus na otočení cyklu(8–11 a 18–21), dvakrát *MoveUnoccupied*, jednou *MoveRobot* a jednou pootočíme celý cyklus. Díky vhodné implementaci, je funkce *MoveRobot* na řádce 14 ekvivalentní funkci *MoveUnoccupied*. Ze všech částí nakonec sestavíme rovnici 1.1.

$$\sum_{j=1}^{k}|L_j|\Bigg(7|L_j|+3|F_j|+6\left(|V|-\sum_{i=j}^{k}|L_i|\right)+2|F_j|+2|L_j|(|L_j|+|P_j|)+ \\ |L_j|+|F_j|+\left(|V|-\sum_{i=j}^{k}|L_i|\right)\left(|F_j|+2\left(|V|-\sum_{i=j}^{k}|L_i|\right)\right)\Bigg) \tag{1.1}$$

Funkce 1.6 Nalezení cesty mezi u a v

function FindShortestPath(u,v)
1: **let** $L_k = [x_k, a_1, a_2, \ldots, a_{i-1}, u, a_{i+1} \ldots, a_n, y_k]$
2: **let** $L_l = [x_l, b_1, b_2, \ldots, b_{j-1}, v, b_{j+1} \ldots, b_n, y_l]$
3: **if** $k = l$ **let** $i < j$
4: **if** $l < k$ **then**
5: **return** $[u, a_{i+1}, \ldots, a_n$,FindShortestPath(y_k,v)]
6: **else if** $l > k$ **then**
7: **return** [FindShortestPath(u,x_l)$, b_1, \ldots, b_{j-1}, v]$
8: **else**
9: **return** $[u, a_{i+1}, \ldots, bi-1, v]$
10: **end if**

Tato funkce projde každé ucho maximálně jednou. Musíme ještě nalézt úsek cesty, který chceme vrátit. Proto je počet kroků $max(k,l)+\left(V-\sum_{i=j}^{k}L_i\right)$. Ještě si povšimneme, že takto nalezená cesta nebude nejkratší, s čímž ale počítáme. Tento předpoklad jsme zahrnuli v odhadu délky nejkratší cesty výše. Za využití tohoto předpokladu upravíme rovnici 1.1 a získáme rovnici 1.2.

$$\sum_{j=1}^{k}|L_j|\Bigg(\left(2\left(|V|-\sum_{i=j}^{k}|L_i|\right)+j\right)\left(|V|-\sum_{i=j}^{k}|L_i|\right)+6\left(|V|-\sum_{i=j}^{k}|L_i|\right)+ \\ 2|L_j|\left(|L_j|+|V|-\sum_{i=j}^{k}|L_i|\right)+8|L_j|+6j\Bigg) \tag{1.2}$$

Do této rovnice později budeme dosazovat konkrétní k a $|L_i|$.

2. Popis vlivu předzpracování grafu na výkonnost algoritmu

Nyní navrhneme rozložení uch, u kterého budeme později sledovat počet kroků řešení a časy experimentů. Pak si krátce popíšeme program, který použijeme pro testování a nakonec vytvoříme testovací data.

2.1 Předzpracování grafu

Jak je patrné z odhadu počtu kroků(rovnice 1.2), běh algoritmu bude souviset s délkou jednotlivých uch $|L_i|$ a taktéž s počtem uch k. Uvědomíme si, že počet uch k, souvisí s průměrnou délkou ucha tak, že čím větší je počet uch k, tím menší je průměrná délka uch. Proto jsme zvolili počet uch jako jeden způsob rozložení grafu. V potaz budeme brát následující počty uch:

- *konstantní* počet uch
$$k \sim O(1) \tag{2.1}$$
- *lineární* počet uch
$$k \sim O(|V|) \tag{2.2}$$
- *logaritmický* počet uch
$$k \sim O(log|V|) \tag{2.3}$$
- *odmocninový* počet uch
$$k \sim O(\sqrt{|V|}) \tag{2.4}$$

Dalším faktorem je rozložení délek uch. Zvolili jsme následující rozložení délek uch:

- *rovnoměrné* rozložení velikostí uch s velikostí $i-$tého ucha
$$|L_i| = \frac{|V| - |C_0|}{k} \tag{2.5}$$
- *lineárně klesající* rozložení velikostí uch s velikostí $i-$tého ucha
$$|L_i| = 2(k - i + 1)\frac{|V| - |C_0|}{k(k+1)} \tag{2.6}$$
- *lineárně rostoucí* rozložení velikostí uch s velikostí $i-$tého ucha
$$|L_i| = 2i\frac{|V| - |C_0|}{k(k+1)} \tag{2.7}$$

Rozložení délek uch nesouvisí přímo s odhadu počtu kroků, ale předpokládáme, že algoritmus na rozložení grafu na ucha bude generovat tato rozložení. Přístupy a problémy se samotným rozložením grafu na ucha jsou zmíněny v kapitole 4.

2.2 Testovací program

K experimentálnímu vyhodnocení jsme použili autorovu implementaci algoritmu BIBOX[2].

K vytvoření testovacího grafu pro algoritmus slouží funkce

make_disasembled_graph(*graph size, ears count, ear size distribution function,start cycle size,graph,bots*)

s parametry

- *graph size* velikost generovaného grafu
- *ears count* počet uch generovaného grafu
- *ear size distribution function* ukazatel na funkci, která počítá velikost uch. Tato funkce musí mít čtyři parametry typu **int**. Funkce dostane k dispozici parametry: počet vrcholů grafu, velikost úvodního cyklu, celkový počet uch a právě zpracovávané ucho a musí vrátit velikost právě zpracovávaného ucha. K dispozici jsou čtyři různé funkce
  - *normal_distribution*
  - *linear_increasing_distribution*
  - *linear_decreasing_distribution*
  - *random_distribution*
- *start cycle size* velikost úvodního cyklu, v programu je zvolena hodnota čtyři
- *graph* ukazatel na graf, který je předán BIBOXu
- *bots* pole robotů, kde $i-$tá pozice je startovní pozice robota, který má cíl ve vrcholu i. Vrcholy 0 a 1 nejsou cílem žádného robota

Pro rozklad grafu na ucha uvažujeme úplný graf, na kterém jsme schopni se ke konkrétnímu počtu uch a jejich rozložení velikostí přiblížit nejvíce a tudíž jsme schopni dostat přesnější výsledky.

2.3 Testovací data

Rozložení uch je simulováno na úplném grafu. Počty uch jsou uvažovány následující:

- *konstantní* počet uch
$$k_{const} = 2 \tag{2.8}$$
- *lineární* počet uch
$$k_{lin} = \frac{|V|}{2} \tag{2.9}$$
- *logaritmický* počet uch
$$k_{log} = log|V| \tag{2.10}$$

- *odmocninový* počet uch

$$k_{sqrt} = \sqrt{|V|} \tag{2.11}$$

Ve všech testech je použit úvodní cyklus C_0 délky čtyři. Dvě pozice úvodního cyklu nejsou cílovou pozicí pro žádného robota, tedy $i = 0, 1 : \Phi^+(i) = \perp$, proto délka i-tého ucha pro rozložení je následující:

- *rovnoměrné* rozložení velikostí uch s velikostí $i-$tého ucha

$$L^{\rightarrow} = |L_i| = \frac{|V| - 4}{k} \tag{2.12}$$

- *lineárně klesající* rozložení velikostí uch s velikostí $i-$tého ucha

$$L^{\searrow} = |L_i| = 2(k - i + 1)\frac{|V| - 4}{k(k + 1)} \tag{2.13}$$

- *lineárně rostoucí* rozložení velikostí uch s velikostí $i-$tého ucha

$$L^{\nearrow} = |L_i| = 2i\frac{|V| - 4}{k(k + 1)} \tag{2.14}$$

Test je proveden na všech možných kombinacích počtu uch a jejich rozložení. Navíc ještě provedeme test na náhodném počtu uch a náhodném rozložení jejich délek, abychom měli porovnání i s obecnějším příkladem.

Pro náhodné testy jsme zvolili délky uch mezi 1 a $2\sqrt{V}$. Horní hranici délky ucha jsme zvolili $2\sqrt{V}$ kvůli tomu, že u vyšších limitů se generoval velice nízký počet uch, což způsobovalo chování zcela stejné jako u konstantního počtu uch. Tento limit na délku ucha by měl zajistit rozvržení uch mezi odmocninovým počtem a lineárním. Toto rozložení označíme $L^?$.

Pro každý graf s $|V|$ vrcholů provedeme 13 různých testů a počet provedení testu pro konkrétní $|V|$ je znázorněn v tabulce 2.1.

$\|V\|$	10	20	40	60	80	100	150	200
počet testů	100	100	100	100	100	100	90	80
$\|V\|$	250	300	350	400	500	600	700	800
počet testů	75	60	45	30	20	15	12	10

Tabulka 2.1: Rozložení testů

Poněvadž u náhodného počtu uch s jejich náhodným rozložením očekáváme větší rozptyl délky výpočtu, provedeme dvakrát více měření pro každé $|V|$.

3. Experimentální vyhodnocení

Řekneme, že *krok řešení* je množina $M_j = \{(u_i, v_i) \mid i \in I\}$ taková, že $\forall i \in I : \Phi(v_i) = \bot \wedge \Phi(u_i) \neq \bot$. Dále označme množinu všech kroků řešení $\mathcal{M}_s = \{M_j \mid j \in J\}$. Značení použité v této kapitole a v příloze:

- $\mathbf{E}_i^j$ je střední hodnota výsledků
- $\tilde{x}_i^j$ je medián výsledků
- $i \in \{const, lin, log, sqrt\}$ identifikátor počtu uch, ke kterému přísluší střední hodnota, respektive medián
- $i \in \{\rightarrow, \searrow, \nearrow\}$ rozložení délek uch(rovnoměrné, lineárně klesající, lineárně rostoucí), ke kterému přísluší střední hodnota respektive medián

Nejdříve porovnáme výsledky výpočtu, tedy počet kroků algoritmu, s časy experimentálních výsledků a s počtem kroků řešení. Poté porovnáme jednotlivé počty kroků algoritmu a časy experimentálních výsledků na jednotlivých rozloženích délek uch pro různé počty uch a obráceně. Dále si ukážeme výsledky z náhodného rozložení délek uch a s náhodným počtem uch a zjistíme, kterému rozložení grafu se podobá nejvíce. Nakonec zhodnotíme výsledky všech testů a porovnání.

3.1 Výsledky jednotlivých rozložení

Do rovnice 1.2 budeme postupně dosazovat různé kombinace počtů uch(rovnice 2.8–2.11). Rozložení délek uch má na výkonnost algoritmu malý vliv, což ukážeme v kapitole 3.2.2, a proto v této části popíšeme pouze normální rozložení délek uch, pro všechny počty uch. Popis všech ostatních rozložení délek uch, tedy lineárně rostoucí a lineárně klesající, je uveden v příloze 5.

Vzhledem k rozsáhlosti testovacích dat, jsou u studie jednotlivých rozložení pouze reprezentativní hodnoty. Kompletní tabulky středních hodnot a mediánů nalezneme v příloze 6.

3.1.1 Konstantní počet uch, rovnoměrné rozložení délek uch

Počet uch tohto případu je k_{const}(rovnice 2.8) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.12). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$V^3 + 6V^2 + 9V \tag{3.1}$$

Časy běhů programu jsou zachyceny v tabulce 3.1. Počty kroků řešení pak v tabulce 3.2. Vyšší rozdíl kvantilů $Q_{0.25}$ a $Q_{0.75}$ je dán tím, že v tomto rozložení záleží hodně na poloze robotů. Primárně záleží na tom, jestli je robot, kterého přesouváme na svůj cíl, v počátku pohyby na uchu, které je zpracováváno a pokud ano, tak jak je hluboko, tedy kolikrát budeme muset provést funkci *RotateCycle* 1.2.

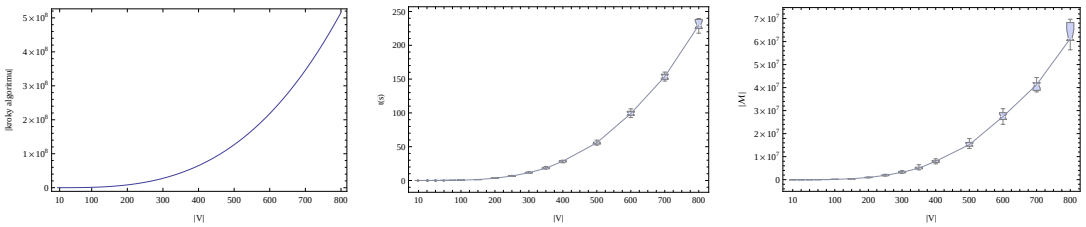
$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.07	0.39	3.36	11.7	28.3	99.1	230.7
$\tilde{x}$	0	0.08	0.39	3.36	11.6	28.6	99	230.9
$Q_{0.25}$	0	0.07	0.38	3.26	11.27	27.44	96.47	225.1
$Q_{0.75}$	0.01	0.08	0.41	3.46	12.06	28.96	102.2	238.2

Tabulka 3.1: Konstantní počet uch, rovnoměrné rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	8.89×10^2	1.16×10^5	9.55×10^5	$7.9\ \times10^6$	$2.7\ \times10^7$	$6.3\ \times10^7$
$\tilde{x}$	$9.2\ \times10^2$	1.16×10^5	9.65×10^5	$8\ \times10^6$	2.74×10^7	6.26×10^7
$Q_{0.25}$	7.78×10^2	1.06×10^5	8.89×10^5	7.44×10^6	2.61×10^7	6.05×10^7
$Q_{0.75}$	9.81×10^2	1.26×10^5	1.00×10^6	8.34×10^6	2.90×10^7	6.82×10^7

Tabulka 3.2: Konstantní počet uch, rovnoměrné rozložení délek uch: Počet kroků řešení

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999953 a mezi časem běhu programu a počtem kroků řešení je 0.999779. Korelace mezi časem běhu programu a počtem kroků řešení je dána tím, že při konstantním počtu uch stráví program nejvíce času na přesouvání robotů a minimum času hledáním cest, popřípadě nejkratších cest, po kterých se mají roboti posouvat. Grafy všech tří funkcí jsou zobrazeny na obrázku 3.1 (a)–(c).



(a) Počet kroků algoritmu (b) Čas běhu programu (c) Počet kroků řešení

Obrázek 3.1: Konstantní počet uch, rovnoměrné rozložení délek uch

3.1.2 Lineární počet uch, rovnoměrné rozložení délek uch

Počet uch tohoto případu je k_{lin}(rovnice 2.9) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.12). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{5}{6}V^3 + \frac{9}{2}V^2 + \frac{53}{3}V \tag{3.2}$$

Časy běhů programu jsou zachyceny v tabulce 3.3. Počty kroků řešení pak v tabulce 3.4. Malý rozdíl kvantilů $Q_{0.25}$ a $Q_{0.75}$ je dán velice krátkými uchy. Pohyb robota na své místo v cyklu, tedy řádky Funkce 1.5 24–29 nezáleží příliš na rozložení robotů.

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999961 a mezi časem běhu programu a počtem kroků řešení je 0.932322. Korelace mezi časem běhu programu a počtem kroků řešení je dána tím, že při lineárním počtu uch stráví program nejvíce času na hledání cest pro přesun robotů

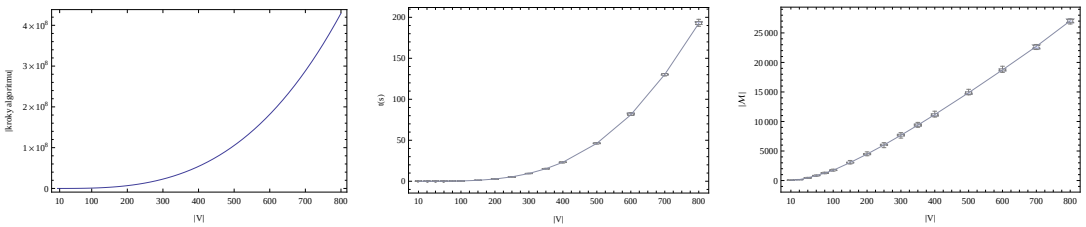
$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.08	0.34	2.71	9.45	23.1	81.7	193.3
$\tilde{x}$	0	0.08	0.34	2.71	9.46	23.1	81	192.9
$Q_{0.25}$	0	0.07	0.33	2.67	9.33	22.9	80.8	191.3
$Q_{0.75}$	0.01	0.08	0.34	2.76	9.57	23.37	83.29	194.7

Tabulka 3.3: Lineární počet uch, rovnoměrné rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	1.54×10^2	1.74×10^3	4.49×10^3	1.12×10^4	1.88×10^4	$2.7\ \times10^4$
$\tilde{x}$	1.55×10^2	1.74×10^3	4.48×10^3	1.12×10^4	1.88×10^4	2.71×10^4
$Q_{0.25}$	1.42×10^2	1.69×10^3	4.38×10^3	1.09×10^4	1.86×10^4	2.67×10^4
$Q_{0.75}$	1.62×10^2	1.78×10^3	4.57×10^3	1.12×10^4	1.89×10^4	2.73×10^4

Tabulka 3.4: Lineární počet uch, rovnoměrné rozložení délek uch: Počet kroků řešení

avšak díky vyššímu počtu hran a kratší délce uch roboti cestují kratší vzdálenost. Grafy všech tří funkcí jsou zobrazeny na obrázku 3.2 (a)–(c).



(a) Počet kroků algoritmu (b) Čas běhu programu (c) Počet kroků řešení

Obrázek 3.2: Lineární počet uch, rovnoměrné rozložení délek uch

3.1.3 Logaritmický počet uch, rovnoměrné rozložení délek uch

Počet uch tohoto případu je k_{log}(rovnice 2.10) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.12). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$V^3\left(\frac{2}{3}+\frac{4}{3\log^2 V}\right)+V^2\left(\frac{1}{3}\log V+3+\frac{14}{3\log V}\right)+V\left(3+3\log V\right) \qquad (3.3)$$

Časy běhů programu jsou zachyceny v tabulce 3.5. Počty kroků řešení pak v tabulce 3.6. Vyšší rozdíl kvantilů $Q_{0.25}$ a $Q_{0.75}$ je určen tím, že ucha jsou netriviální délky a tudíž už záleží na pozici robota, kterého chceme umístit. Záleží hlavně na tom, jestli je před tím, než ho začneme umisťovat na uchu, na které patří, a pokud ano, tak jak hluboko je na daném uchu. Tedy kolikrát budeme muset provést funkci *RotateCycle* 1.2.

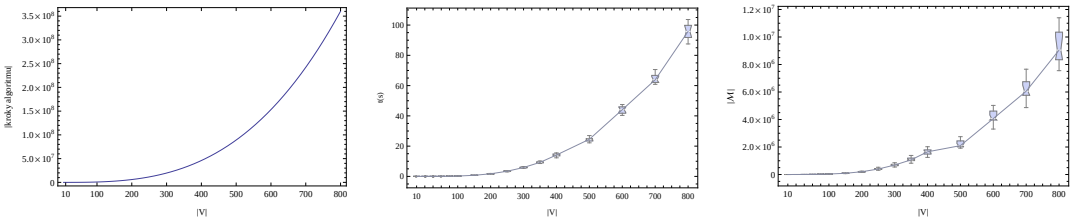
Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999597 a mezi časem běhu programu a počtem kroků řešení je 0.999007. Korelace mezi počtem kroků řešení a časem běhu programu souvisí s délkou uch obdobným způsobem jako rozdíl kvantilů. Grafy všech tří funkcí jsou zobrazeny na obrázku 3.2 (a)–(c).

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.05	0.24	1.73	5.89	14.1	43.8	96.2
$\tilde{x}$	0	0.05	0.24	1.72	5.89	14.1	44.1	96.7
$Q_{0.25}$	0	0.05	0.23	1.66	5.64	13.6	41.7	91.8
$Q_{0.75}$	0.01	0.06	0.25	1.79	6.11	14.8	45.9	99.8

Tabulka 3.5: Logaritmický počet uch, rovnoměrné rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	8.18×10^2	3.62×10^4	2.03×10^5	1.65×10^6	4.17×10^6	9.35×10^6
$\tilde{x}$	8.14×10^2	3.55×10^4	2.02×10^5	1.65×10^6	4.07×10^6	9.29×10^6
$Q_{0.25}$	7.02×10^2	3.34×10^4	1.89×10^5	1.49×10^6	3.96×10^6	8.35×10^6
$Q_{0.75}$	8.84×10^2	3.87×10^4	2.20×10^5	1.79×10^6	4.61×10^6	1.04×10^7

Tabulka 3.6: Logaritmický počet uch, rovnoměrné rozložení délek uch: Počet kroků řešení



(a) Počet kroků algoritmu (b) Čas běhu programu (c) Počet kroků řešení

Obrázek 3.3: Logaritmický počet uch, rovnoměrné rozložení délek uch

3.1.4 Odmocninový počet uch, rovnoměrné rozložení délek uch

Počet uch tohoto případu je k_{sqrt}(rovnice 2.11) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.12). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{2}{3}V^3 + \frac{1}{3}V^{5/2} + \frac{13}{3}V^2 + \frac{23}{3}V^{3/2} + 3V \tag{3.4}$$

Časy běhů programu jsou zachyceny v tabulce 3.7. Počty kroků řešení pak v tabulce 3.8. Nízký rozdíl kvantilů $Q_{0.25}$ a $Q_{0.75}$ je určen tím, že délky uch jsou blíže k délkám uch u lineárního rozložení, tudíž se chovají podobným způsobem.

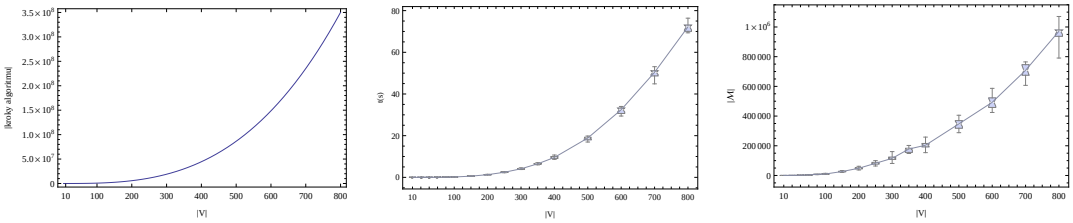
$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.05	0.18	1.28	4.11	9.63	31.9	72.1
$\tilde{x}$	0	0.05	0.18	1.29	4.11	9.57	32.4	72.4
$Q_{0.25}$	0	0.04	0.18	1.24	3.99	9.14	30.7	70.1
$Q_{0.75}$	0.01	0.05	0.19	1.33	4.19	10.01	33.3	73.1

Tabulka 3.7: Odmocninový počet uch, rovnoměrné rozložení délek uch: Čas běhu programu

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999793 a mezi časem běhu programu a počtem kroků řešení je 0.993302. Korelaci mezi počtem řešení časem běhu programu lze vysvětlit obdobným argumentem jako rozdíl kvantilů. Grafy všech tří funkcí jsou zobrazeny na obrázku 3.4 (a)–(c).

$\|V\|$	20	100	200	400	600	800
E	3.42×10^2	1.02×10^4	4.89×10^4	2.06×10^5	4.94×10^5	9.51×10^5
$\tilde{x}$	3.46×10^2	1.01×10^4	4.87×10^4	2.04×10^5	4.93×10^5	9.72×10^5
$Q_{0.25}$	3.16×10^2	9.43×10^3	4.47×10^4	1.92×10^5	4.58×10^5	9.37×10^5
$Q_{0.75}$	3.77×10^2	1.08×10^4	5.19×10^4	2.15×10^5	5.23×10^5	9.79×10^5

Tabulka 3.8: Odmocninový počet uch, rovnoměrné rozložení délek uch: Počet kroků řešení



(a) Počet kroků algoritmu (b) Čas běhu programu (c) Počet kroků řešení

Obrázek 3.4: Odmocninový počet uch, rovnoměrné rozložení délek uch

3.2 Porovnání podle rozložení

Předpokládejme, že máme rozložení délek uch lineárně rostoucí. Celý algoritmus BIBOX je složen z umisťování jednotlivých robotů. Ukážeme si tedy složitost umisťování jednoho robota. Umisťování robota má tři části, které může ovlivnit počet uch a těmi jsou:

1. Vyrotování robota z právě zpracovávaného ucha (Funkce 1.5, řádky 3–23)
2. Nalezení cesty délky l na počátek v právě zpracovávaného ucha (Funkce 1.5, řádek 25)
3. Přesun robota po cestě délky l na vrchol v (Funkce 1.5, řádky 5)

Bod číslo jedna přispívá jak k časové složitosti programu, tak k počtu kroků řešení. Zbylé dva body přispívají primárně k časové složitosti. K počtu kroků řešení přispívají body dva a tři hlavně délkou cesty l.

Nejdříve si musíme uvědomit, že při rostoucím počtu uch, klesá délka jednotlivých uch.

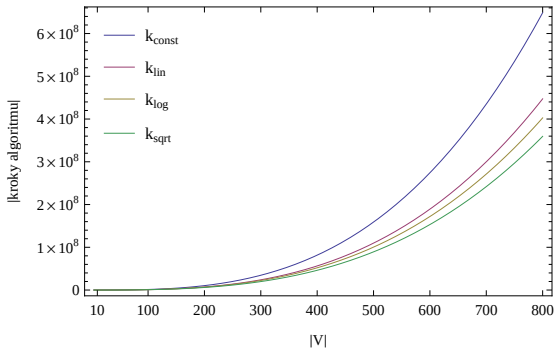
Pro delší ucha se provede vícekrát bod číslo jedna, což výrazně přispívá k počtu kroků řešení i k časové složitosti. Naopak body dva a tři, tedy hledání cesty se zjednodušuje(při konstantním počtu uch je nalezení cesty $O(1)$), poněvadž máme řidší graf.

Pro kratší ucha se sice bod jedna provede méně často, ale pro nalezení cesty spotřebujeme mnohem více času (při lineárním počtu uch je časová složitost nalezení $O(V)$). Délku cesty l můžeme shora omezit v obou případech $V - \sum_{i=j}^{k} |L_i|$.

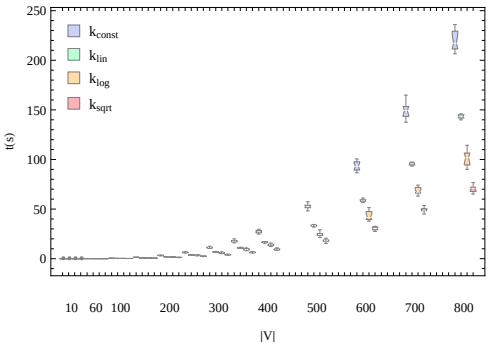
Cílem je tedy nalézt rozložení takové, kde poměr délek uch bude ne příliš malý, kvůli bodu jedna, a ani ne příliš vysoký, kvůli bodům dva a tři. Z toho důvodu jsme zvolili počty uch konstantí, lineární, logaritmický a odmocninový. Délka příslušných uch je vyjádřena v rovnici 2.14, kde za k dosadíme hodnoty vypočteny v rovnicích 2.8–2.11. Nyní ukážeme, že pro čas běhu programu je nejlepší rozložení s počtem uch odmocninovým, zatímco pro počet kroků řešení je nejlepší rozložení s počtem uch lineárním.

3.2.1 Porovnání podle počtu uch

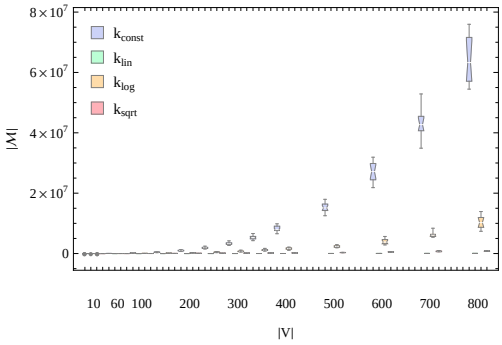
Porovnáním rovnic kroků algoritmu pro různé délky uch (rovnice 5.5, 5.6, 5.7 a 5.8) zjistíme, že nejrychlejší by mělo být rozložení grafu s počtem uch odmocninovým. O něco pomalejší je rozložení grafu s počtem uch logaritmickým a ještě o něco vyšší čas bude mít rozložení grafu s počtem uch lineárním. Nejpomalejší a výrazně pomalejší by mělo být rozložení s počtem uch konstantním. Porovnání těchto výsledků je zobrazeno v grafu 3.5a.



(a) Počet kroků algoritmu



(b) Čas běhu programu



(c) Počet kroků řešení

Obrázek 3.5: Různý počet uch, lineárně rostoucí rozložení délek uch

Podle našeho předpokladu odpovídají výpočtům naměřené testy na čas běhu programu, jejichž výsledky jsou v tabulce 3.9. Tyto výsledky jsou též vyobrazeny v grafu 3.5b. Avšak jak si můžeme povšimnout v tabulce 3.10 a jí příslušném grafu 3.5c, počet kroků řešení má značně odlišné chování než zbylé dvě hodnoty. Rozdílnost kroků byla ovšem též předpokládána a důvody jsme k tomu zmínili výše.

$\|V\|$	20	60	100	200	300	400	600	800
$\mathbf{E}_{const}$	0.0028	0.078	0.38	3.25	11.2	27.2	93.2	220.2
$\mathbf{E}_{lin}$	0.0031	0.059	0.25	1.95	6.79	16.6	58.6	143.4
$\mathbf{E}_{log}$	0.0029	0.052	0.24	1.72	5.89	13.9	43.3	101.8
$\mathbf{E}_{sqrt}$	0.0026	0.045	0.18	1.25	4.08	9.4	30.3	70.7

Tabulka 3.9: Lineárně rostoucí rozložení délek uch: Čas běhu programu

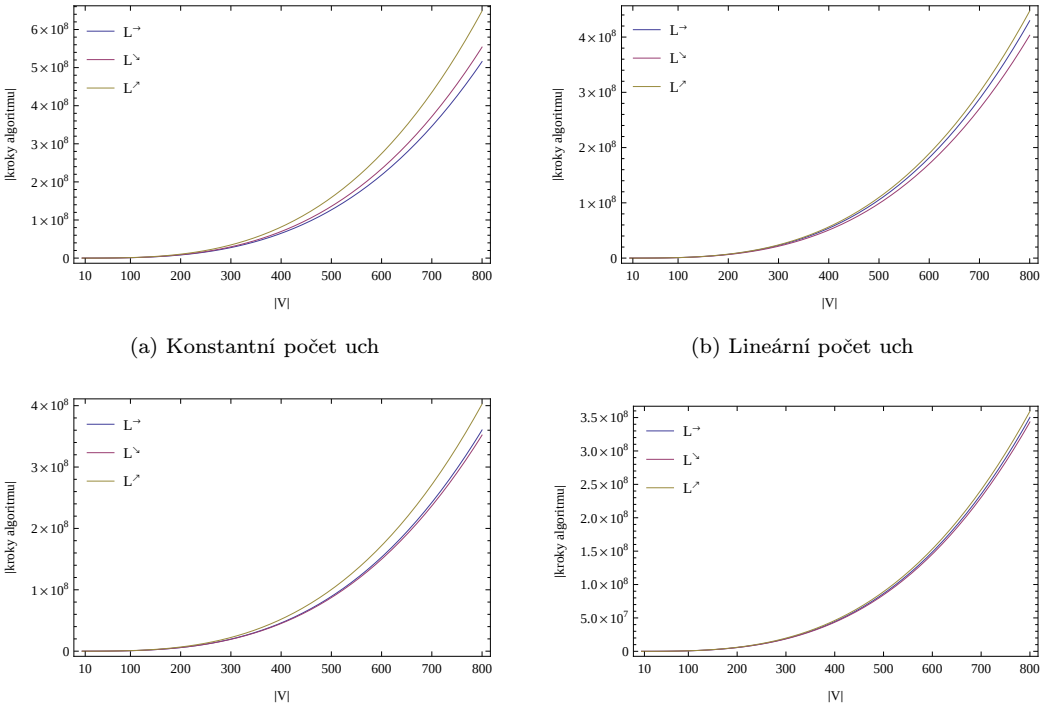
$\|V\|$	20	100	200	400	600	800
$\mathbf{E}_{const}$	6.60×10^2	1.18×10^5	9.94×10^5	8.17×10^6	2.70×10^7	6.45×10^7
$\mathbf{E}_{lin}$	1.82×10^2	2.17×10^3	5.43×10^3	1.37×10^4	2.25×10^4	3.21×10^4
$\mathbf{E}_{log}$	7.46×10^2	3.80×10^4	2.16×10^5	1.67×10^5	4.02×10^6	1.03×10^7
$\mathbf{E}_{sqrt}$	2.96×10^2	1.10×10^4	4.77×10^4	2.12×10^5	5.01×10^5	8.74×10^5

Tabulka 3.10: Lineárně rostoucí rozložení délek uch: Počet kroků řešení

3.2.2 Porovnání podle rozdělení délek uch

Nyní prozkoumáme rozdíly mezi rozložením délek uch.

Z grafů 3.6a–3.6d můžeme usuzovat, že lineárně klesající rozložení délek uch by mělo být nejrychlejší.



(a) Konstantní počet uch

(b) Lineární počet uch

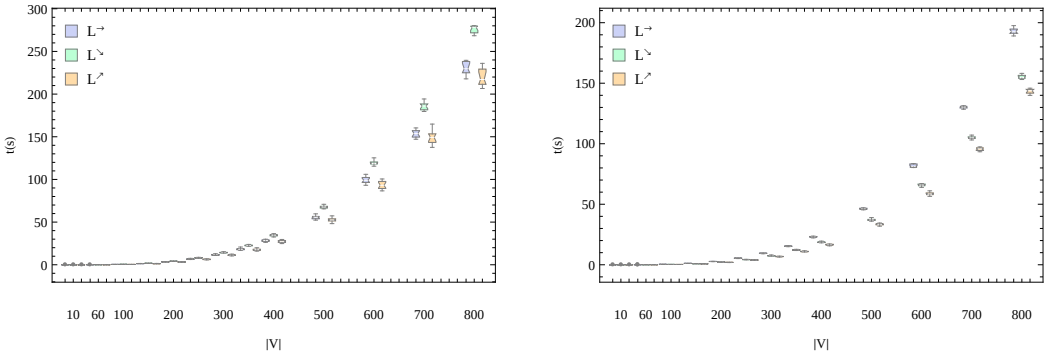
(c) Logartimický počet uch

(d) Odmocninový počet uch

Obrázek 3.6: Počet kroků algoritmu

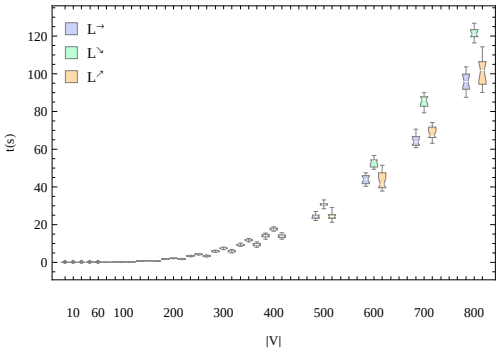
Jak si ale můžeme všimnout u grafů 3.7a–3.7d, výsledky testů se zcela neshodují s analýzou.

Tato neshoda je způsobena tím, že při odhadu počtu kroků algoritmu počítáme s nejhorším možným případem, tedy že právě nalezený robot je na právě zpracovávaném uchu, tudíž ho musíme vyrotovat. Tento případ však zjevně nenastane tak často, což se projeví u neshody, poněvadž rozdíly mezi rozděleními uch jsou příliš malé.

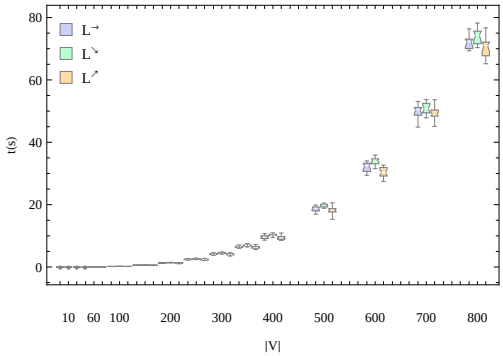


(a) Konstantní počet uch

(b) Lineární počet uch



(c) Logartimický počet uch

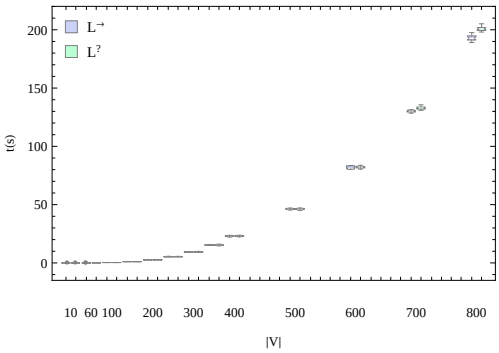


(d) Odmocninový počet uch

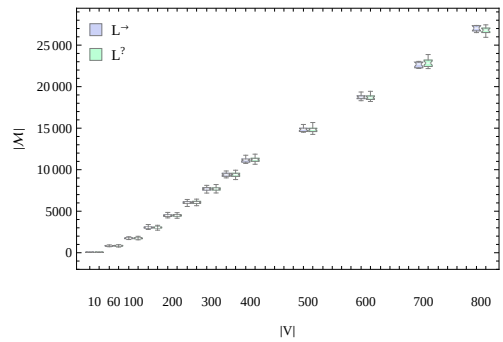
Obrázek 3.7: Čas běhu programu

3.2.3 Náhodné měření

Při porovnání experimentálních výsledků náhodného rozložení zjistíme, že toto rozložení je velice podobné rozložení s délkami uch rovnoměrně rozloženými a počtem uch lineárním. Toto je vidět jak v tabulce 3.12, tak v grafu 3.8.



(a) Čas běhu programu



(b) Počet kroků řešení

Obrázek 3.8: Porovnání náhodného testu a rovnoměrného rozložení délek lineárního počtu uch

$\|V\|$	20	60	100	200	300	400	600	800
$\mathbf{E}^{\rightarrow}_{lin}$	0.0033	0.076	0.338	2.72	9.45	23.1	81.717	193.4
$\mathbf{E}_{rand}$	0.0033	0.072	0.336	2.72	9.47	23.1	82.195	201.2

Tabulka 3.11: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
$\mathbf{E}^{\rightarrow}_{lin}$	1.54×10^2	1.74×10^3	4.49×10^3	1.12×10^4	1.88×10^4	2.7×10^4
$\mathbf{E}_{rand}$	1.56×10^2	1.74×10^3	$4.5\ \times10^3$	1.12×10^4	1.87×10^4	2.68×10^4

Tabulka 3.12: Počet kroků řešení

3.3 Vyhodnocení výsledků

Zjistili jsme tedy, že při rostoucím počtu uch klesá počet kroků řešení(příloha 6). Také jsme zjistili, že při odmocninovém počtu uch jsou časy běhů programu nejnižší, čemuž odpovídá i počet kroků algoritmu(kapitola 3.2.1).

Dále jsme si ukázali, že rozložení délek uch je nejvýhodnější lineární, avšak tento výsledek se zcela neshoduje s výsledky analýzy programu. Rozdíly mezi rozloženími délek uch jsou u odmocninového počtu uch zanedbatelné. Čím pomalejší řešení je pro různé počty uch, tím se zvyšují rozdíly mezi rozloženími délek uch, tudíž u konstantního počtu uch je rozdíl nejvíce znatelný(kapitola 3.2.2).

4. Diskuse a otevřené otázky

Obecně k problému rozložení grafu na ucha

Nyní bychom potřebovali nějaký algoritmus s časovou složitostí $O(V^3)$, který by parametricky rozebral graf na k uch.

Tento problém se dá zjednodušit na problém hledání cesty nějaké délky l. Pro $l = V$ je tento problém problém hamiltonovské kružnice. Pro $l < V$ lze tento problém vyřešit v čase O*(2^l)[4] (O* potlačuje polynomiální části v O), ale to jen za předpokladu, že takováto cesta existuje. Jak je zjevné, tento přístup má smysl jen pro $l \sim O(\log V)$, což splňuje jen lineární počet uch. U lineárního počtu uch, je zřejmě vhodnější využít nějakého algoritmu na hledání nejkratších cest, poněvadž na cestu klademe ještě ten požadavek, že její koncové vrcholy musí být v množině vrcholů obsažených v uchách s nižším indexem.

Jistým zjednodušením problému parametrizace uch bychom mohli uvažovat cestou délky l_{int} v nějakém rozmezí l_{min} a l_{max}. Stačí dosadit za délku cesty $l = V$. Problém cesty alespoň délky l_{min} lze vyřešit v časové složitosti O*$(4^{l_{min}})$[4][5] i na neválených grafech. Problém je tedy stejný, jako u cest konkrétní délky l.

Parametrický rozklad na ucha na obecných grafech je tedy příliš složitý. Pokud bychom ale uvážili nějaké speciální případy, mohli bychom získat jednodušší rozklad.

Některé speciální případy

Úplný graf

Kupříkladu na úplných grafech je možné provést rozklad v lineárním čase. Rozložení je provedeno ve dvou jednoduchých krocích.

Funkce 4.1 Rozložení úplného grafu na ucha

1: **let** $L_0 = [0, 1, 2, 3]$
2: $firstUnused \leftarrow 4$
3: $i \leftarrow 1$
4: **while** $firstUnused \neq |V|$ **do**
5: $\quad l \leftarrow$délka aktuálního ucha
6: $\quad L_i \leftarrow (firstUnused, \ldots, firstUnused + l)$
7: $\quad firstUnused \leftarrow firstUnused + l + 1$
8: **end while**

Délku aktuálního ucha si volíme podle nějakých předpokladů a délky ucha přímo určí jejich počet. Tento přístup je využit i v této práci k provedení experimentu. Délky uch jsou zmíneny v kapitole 2.3 a funkce, které jsou použity v programu pak v kapitole 2.2.

Další speciální případy

U řidších grafů, kupříkladu rovinných, by se dalo využít struktur, kterými se dají reprezentovat. U rovinných grafů je jedna ze struktur popisující graf SPQR

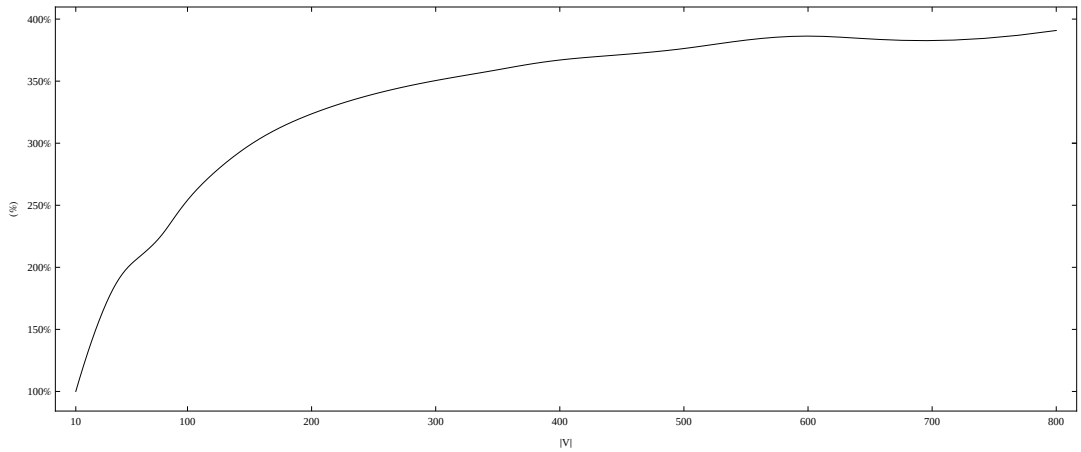
strom[14], který lze implementovat v čase O(V)[15]. SPQR strom je strom pro rovinný graf, kde každý syn nějakého vrcholu stromu odpovídá podgrafu, který je spojen se zbytkem grafu pomocí nějakých dvou vrcholů, tudíž těmito vrcholy může vést právě jedno ucho. Pokud bychom tedy udržovali v každém vrcholu hodnotu délky cest v synech, mohli bychom tím vytvořit rozložení grafu na ucha. U tohot rozložení bychom byli schopni manipulovat s délkami uch. Abychom zjistili, zdali příliš nezkomplikuje SPQR strom a zanechá mu lineární časovou složitost a nejlépe i velikost, museli bychom tento problém zanalyzovat značně podrobněji.

Pokud bychom uvážili nějaký specifičtější rovinný graf, jako je kupříkladu graf mřížky bez děr, pak bychom k rozložení zřejmě nepotřebovali ani SPQR strom.

Pokud budeme předpokládat na vstupu nějaké specifické grafy, pak bychom mohli navrhnout nějaký algoritmus rozkladu na ucha. Taktéž pokud bychom si mohli vybrat s více rozkladů, pak se budeme snažit vybrat si rozložení výhodnější. Výhodnost rozložení popisují výsledky prezentované v kapitole 3.3.

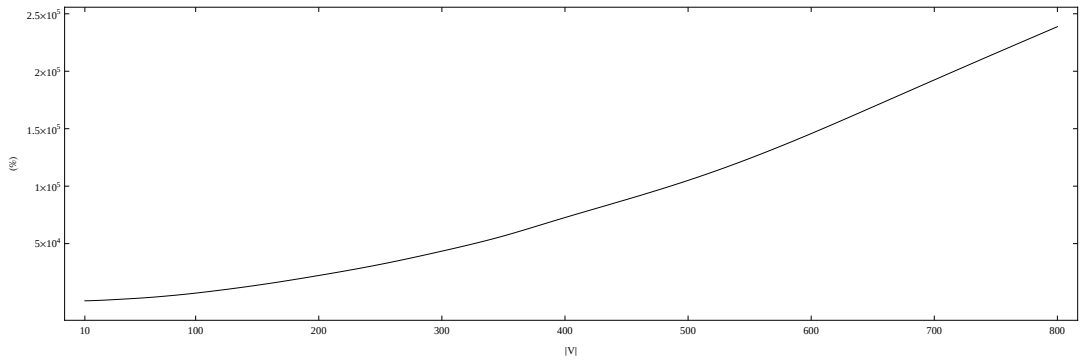
Závěr

Z analýzy algoritmu Bibox[1] a provedených experimentů na programu BIBOX[2] plyne, že při předzpracování grafu pro algoritmus BIBOX[1] záleží hlavně na počtu uch k, respektive na jejich průměrné délce l. Pro optimální rychlost algoritmu je vhodné volit počet uch $k \sim O(\sqrt{V})$. Vyvarovat bychom se měli počtu uch $k = O(1)$, pro který je časová složitost značně vyšší než pro ostatní testované k. Náhodné rozložení grafu je velmi podobné počtu uch $k = O(V)$. Volbou rozložení grafu zrychlíme téměř čtyřikrát programu pro graf na osmi stech vrcholech. Násobnost zrychlení pro různé počty vrcholů je zobrazeno v grafu 4.1. Jak lze předpokládat z grafu 4.1, tak zrychlení programu roste logaritmicky, vzhledem k počtu vrcholů.



Obrázek 4.1: Čas běhu programu

Jak ukazují experimentální výsledky, počet kroků řešení je pro větší k menší. Vhodnou volbou rozložení grafu docílíme 2500 násobnému zmenšení velikosti řešení pro graf na osmi stech vrcholech. Násobnost zmenšení pro různé počty vrcholů je zobrazeno v grafu 4.2. Jak lze předpokládat z grafu 4.2, zmenšení řešení se chová polynomiálně vzhledem k počtu vrcholů.



Obrázek 4.2: Počet kroků řešení

Rozložení délek uch nemá příliš velký vliv na výkonnost algoritmu ani na velikost řešení. Výsledky jsou trochu podrobněji popsány v kapitole 3.3.

Parametrizace rozkladu grafu na ucha je obecně těžká[4][5]. Na některých specifických příkladech je implementace parametrického rozkladu na ucha triviální, což je ukázáno v programu na úplných grafech. Rozebíraní grafu je více popsáno v kapitole 4.

Seznam použité literatury

[1] Surynek, Pavel. *A novel approach to path planning for multiple robots in bi-connected graphs*. IEEE International Conference on Robotics and Automation. ICRA'09. 2009. 3613–3619.

[2] Surynek, Pavel. *Program Bibox*. Univerzita Karlova v Praze. 23 července 2013. `http://ktiml.mff.cuni.cz/~surynek/research/icra2009/experiments/source/multirobot-0.1.24-loiterer.tgz`

[3] Surynek, Pavel. *A SAT-Based Approach to Cooperative Path-Finding Using All-Different Constraints*. Proceedings of the 5th Annual Symposium on Combinatorial Search. SoCS 2012. 2012. 191–192.

[4] Williams, Ryan. *Finding paths of length k in $O^*(2^k)$ time*. Information Processing Letters. 109(6). 2009. 315–318. 23 července 2013. `http://arxiv.org/abs/0807.3026`.

[5] Kneis, Joachim and Mölle, Daniel and Richter, Stefan and Rossmanith, Peter. *Divide-and-color*. Proceedings of the 32nd international conference on Graph-Theoretic Concepts in Computer Science. WG'06. 2006. 58–67. Springer-Verlag. 23 července 2013. `http://dx.doi.org/10.1007/11917496_6`.

[6] Hart, P.E. and Nilsson, N.J. and Raphael, B.. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics. SSC4 4(2). 1968. 100–107.

[7] Silver, David. *Cooperative Pathfinding*. In 1st Conference on Artificial Intelligence and Interactive Digital Entertainment. AIIDE'05. 2005. 117-123.

[8] Holte, R. C. and Perez, M. B. and Zimmer, R. M. and MacDonald, A. J.. *Hierarchical A*: Searching Abstraction Hierarchies Efficiently*. In Proceedings of the National Conference on Artificial Intelligence. 1996. 530–535.

[9] Erdmann, M. and Lozano-Perez, Tomás. *On Multiple Moving Objects* Algorithmica. Vol. 2. 1987. 477 – 521. Proceedings of IEEE International Conference on Robotics and Automation. San Francisco, CA. April 1986. 1419-1424.

[10] Luna, Ryan and Bekris, Kostas E.. *Push and Swap: Fast Cooperative Path-Finding with Completeness Guarantees*. Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume One. IJCAI'11. 2011. 294–300.

[11] de Wilde, Boris and ter Mors, Adriaan W. and Witteveen, Cees. *Push and rotate: cooperative multi-agent path planning*. Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems. AAMAS '13. 2013. 87–94.

[12] Yu, Jingjin and LaValle, Steven M.. *Time Optimal Multi-agent Path Planning on Graphs*. The First AAAI Workshop on Multiagent Pathfinding. WoMP'12. 2012.

[13] Yu, Jingjin and LaValle, Steven M.. *Multi-agent path planning and network flow*. Workshop on the Algorithmic Foundations of Robotics. 2012.

[14] Di Battista, Giuseppe and Tamassia, Roberto *On-Line Planarity Testing* SIAM Journal on Computing vol. 25(5). 1996. 956–997.

[15] Gutwenger, Carsten and Mutzel, Petra. *A linear time implementation of SPQR-trees*. Lecture Notes in Computer Science. Vol. 1984 Proceedings 17th International Colloquium on Automata, Languages and Programming. GD 2000.2001 77–90. Springer Berlin Heidelberg.

[16] Huang, Ruoyun and Chen, Yixin and Zhang, Weixiong. *A Novel Transition Based Encoding Scheme for Planning as Satisfiability*. Proceedings AAAI 2010. 2010 AAAI Press.

Seznam tabulek

Seznam použitých zkratek

SAT - satisfiability (splnitelnost logické formule)

Přílohy

5. Další rozložení uch

5.1 Konstantní počet uch, lineárně klesající rozložení délek uch

Počet uch tohoto případu je k_{const}(rovnice 2.8) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.13). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{29}{27}V^3 + \frac{56}{9}V^2 + 8V \tag{5.1}$$

Časy běhů programu jsou zachyceny v tabulce 5.1. Počty kroků řešení pak v tabulce 5.2.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.09	0.47	4.08	14.3	34.6	119.5	276.2
$\tilde{x}$	0	0.09	0.47	4.07	14.29	34.8	118.7	278.6
$Q_{0.25}$	0	0.09	0.46	3.99	14.02	33.8	118.2	271.8
$Q_{0.75}$	0.01	0.10	0.49	4.17	14.54	35.2	120.4	279.6

Tabulka 5.1: Konstantní počet uch, lineárně klesající rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	8.95×10^2	1.72×10^5	1.44×10^6	1.21×10^7	4.06×10^7	9.61×10^7
$\tilde{x}$	8.58×10^2	$1.7\ \times10^5$	1.45×10^6	1.22×10^7	4.04×10^7	9.61×10^7
$Q_{0.25}$	7.67×10^2	1.61×10^5	1.38×10^6	1.16×10^7	3.94×10^7	$9.5\ \times10^7$
$Q_{0.75}$	9.79×10^2	1.83×10^5	1.49×10^6	1.25×10^7	4.19×10^7	9.74×10^7

Tabulka 5.2: Konstantní počet uch, lineárně klesající rozložení délek uch: Počet kroků řešení

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999962 a mezi časem běhu programu a počtem kroků řešení je 0.999954.

5.2 Lineární počet uch, lineárně klesající rozložení délek uch

Počet uch tohoto případu je k_{lin}(rovnice 2.9) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.13). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{47}{60}V^3 + \frac{98}{15}V^2 + \frac{243}{5}V + \frac{464}{5} + \frac{176}{5V} \tag{5.2}$$

Časy běhů programu jsou zachyceny v tabulce 5.3. Počty kroků řešení pak v tabulce 5.4.

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999944 a mezi časem běhu programu a počtem kroků řešení je 0.938674.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.06	0.27	2.16	7.52	18.6	65.8	155.2
$\tilde{x}$	0	0.06	0.27	2.17	7.53	18.6	65.8	154.9
$Q_{0.25}$	0	0.06	0.27	2.13	7.39	18.4	65.3	154.2
$Q_{0.75}$	0.01	0.07	0.28	2.21	7.59	18.8	66.6	156.5

Tabulka 5.3: Lineární počet uch, lineárně klesající rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	2.14×10^2	2.89×10^3	7.86×10^3	2.09×10^4	3.55×10^4	5.23×10^4
$\tilde{x}$	2.12×10^2	2.87×10^3	7.88×10^3	$2.1\ \times10^4$	3.55×10^4	5.22×10^4
$Q_{0.25}$	1.96×10^2	2.78×10^3	7.65×10^3	2.05×10^4	3.50×10^4	5.21×10^4
$Q_{0.75}$	2.27×10^2	$3\ \times10^3$	8.07×10^3	2.13×10^4	3.64×10^4	5.28×10^4

Tabulka 5.4: Lineární počet uch, lineárně klesající rozložení délek uch: Počet kroků řešení

5.3 Logaritmický počet uch, lineárně klesající rozložení délek uch

Počet uch tohoto případu je k_{log}(rovnice 2.10) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.13). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\begin{aligned} &V^3\left(\frac{2}{3}+\frac{14}{15\log V}+\frac{56}{15\log^2 V}+\frac{44}{15\log^3 V}-\frac{4}{15\log^4 V}\right)+\\ &+V^2\left(\frac{7}{30}\log V+\frac{11}{3}+\frac{77}{6\log V}+\frac{37}{3\log^2 V}+\frac{44}{15\log^3 V}\right)+\\ &+V\left(2\log V+8+\frac{10}{\log V}+\frac{4}{\log^2 V}\right) \end{aligned} \tag{5.3}$$

Časy běhů programu jsou zachyceny v tabulce 5.5. Počty kroků řešení pak v tabulce 5.6.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.062	0.29	2.15	7.45	17.6	53.4	121.8
$\tilde{x}$	0	0.06	0.29	2.17	7.46	17.6	53.9	122.1
$Q_{0.25}$	0	0.06	0.28	2.09	7.28	17.2	50.6	119.7
$Q_{0.75}$	0.01	0.07	0.3	2.2	7.6	18.2	54.3	123.4

Tabulka 5.5: Logaritmický počet uch, lineárně klesající rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	8.83×10^2	6.15×10^4	3.77×10^5	2.98×10^6	7.38×10^6	1.92×10^7
$\tilde{x}$	8.73×10^2	6.01×10^4	3.74×10^5	2.93×10^6	7.38×10^6	1.95×10^7
$Q_{0.25}$	7.85×10^2	5.71×10^4	3.49×10^5	2.82×10^6	6.97×10^6	1.74×10^7
$Q_{0.75}$	9.51×10^2	6.68×10^4	4.01×10^5	3.14×10^6	8.11×10^6	2.07×10^7

Tabulka 5.6: Logaritmický počet uch, lineárně klesající rozložení délek uch: Počet kroků řešení

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999657 a mezi časem běhu programu a počtem kroků řešení je 0.998289.

5.4 Odmocninový počet uch, lineárně klesající rozložení délek uch

Počet uch tohoto případu je k_{sqrt}(rovnice 2.11) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.13). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{2}{3}V^3 + \frac{7}{6}V^{5/2} + \frac{37}{5}V^2 + \frac{1253}{30}V^{3/2} + \frac{1141}{15}V + \frac{794}{15}\sqrt{V} + 12 \tag{5.4}$$

Časy běhů programu jsou zachyceny v tabulce 5.7. Počty kroků řešení pak v tabulce 5.8.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.05	0.21	1.43	4.48	10.3	33.9	73.9
$\tilde{x}$	0	0.05	0.21	1.42	4.48	10.3	34.3	74.6
$Q_{0.25}$	0	0.05	0.2	1.39	4.37	10.1	33.1	71.6
$Q_{0.75}$	0.01	0.05	0.21	1.46	4.55	10.5	34.7	75.6

Tabulka 5.7: Odmocninový počet uch, lineárně klesající rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	4.39×10^2	1.73×10^4	8.77×10^4	4.06×10^5	1.05×10^6	1.94×10^6
$\tilde{x}$	4.27×10^2	1.73×10^4	8.82×10^4	4.09×10^5	1.06×10^6	1.92×10^6
$Q_{0.25}$	3.85×10^2	1.58×10^4	8.18×10^4	3.71×10^5	1.01×10^6	1.89×10^6
$Q_{0.75}$	4.81×10^2	1.85×10^4	9.33×10^4	4.42×10^5	1.11×10^6	1.96×10^6

Tabulka 5.8: Odmocninový počet uch, lineárně klesající rozložení délek uch: Počet kroků řešení

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.99947 a mezi časem běhu programu a počtem kroků řešení je 0.99387.

5.5 Konstantní počet uch, lineárně rostoucí rozložení délek uch

Počet uch tohoto případu je k_{const}(rovnice 2.8) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.14). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{34}{27}V^3 + \frac{56}{9}V^2 + 10V \tag{5.5}$$

Časy běhů programu jsou zachyceny v tabulce 5.9. Počty kroků řešení pak v tabulce 5.10.

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999939 a mezi časem běhu programu a počtem kroků řešení je 0.999987.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.078	0.37	3.25	11.2	27.2	93.2	220.2
$\tilde{x}$	0	0.08	0.38	3.25	11.2	27.1	93.2	218.4
$Q_{0.25}$	0	0.07	0.35	3.11	10.8	26.1	89.4	211.4
$Q_{0.75}$	0.01	0.08	0.4	3.38	11.5	28.3	97.7	229.3

Tabulka 5.9: Konstantní počet uch, lineárně rostoucí rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	6.60×10^2	1.18×10^5	9.94×10^5	8.17×10^6	2.71×10^7	6.45×10^7
$\tilde{x}$	6.71×10^2	1.15×10^5	9.85×10^5	7.94×10^6	2.72×10^7	6.44×10^7
$Q_{0.25}$	5.34×10^2	1.01×10^5	8.82×10^5	7.62×10^6	2.44×10^7	$5.7\ \times10^7$
$Q_{0.75}$	7.80×10^2	1.35×10^5	1.09×10^6	9.07×10^6	2.98×10^7	7.16×10^7

Tabulka 5.10: Konstantní počet uch, lineárně rostoucí rozložení délek uch: Počet kroků řešení

5.6 Lineární počet uch, lineárně rostoucí rozložení délek uch

Počet uch tohoto případu je k_{lin}(rovnice 2.9) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.14). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{13}{15}V^3 + \frac{269}{30}V^2 + \frac{776}{15}V + \frac{426}{5} + \frac{256}{15V} \tag{5.6}$$

Časy běhů programu jsou zachyceny v tabulce 5.11. Počty kroků řešení pak v tabulce 5.12.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.06	0.25	1.95	6.79	16.6	58.6	143.4
$\tilde{x}$	0	0.06	0.25	1.96	6.81	16.7	58.4	143.8
$Q_{0.25}$	0	0.06	0.24	1.9	6.67	16.3	57.9	141.8
$Q_{0.75}$	0.01	0.06	0.25	2	6.93	16.9	59.4	144.9

Tabulka 5.11: Lineární počet uch, lineárně rostoucí rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	1.82×10^2	2.17×10^3	5.43×10^3	1.37×10^4	2.26×10^4	3.21×10^4
$\tilde{x}$	1.82×10^2	2.15×10^3	5.44×10^3	1.37×10^4	2.29×10^4	$3.2\ \times10^4$
$Q_{0.25}$	1.63×10^2	2.07×10^3	5.22×10^3	1.33×10^4	2.19×10^4	3.17×10^4
$Q_{0.75}$	2.00×10^2	2.26×10^3	5.63×10^3	1.41×10^4	2.31×10^4	3.24×10^4

Tabulka 5.12: Lineární počet uch, lineárně rostoucí rozložení délek uch: Počet kroků řešení

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999862 a mezi časem běhu programu a počtem kroků řešení je 0.925597.

5.7 Logaritmický počet uch, lineárně rostoucí rozložení délek uch

Počet uch tohoto případu je k_{log}(rovnice 2.10) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.14). Dosazením těchto hodnot do rovnice 1.2 získáme

počet kroků algoritmu:

$$V^3\left(\frac{2}{3}+\frac{26}{15\log V}+\frac{64}{15\log^2 V}+\frac{12}{5\log^3 V}-\frac{16}{15\log^4 V}\right)+$$
$$+V^2\left(\frac{2}{5}\log V+\frac{7}{2}+\frac{37}{3\log V}+\frac{25}{2\log^2 V}+\frac{49}{15\log^3 V}\right)+ \quad (5.7)$$
$$+V\left(4\log V+10+\frac{8}{\log V}+\frac{2}{\log^2 V}\right)$$

Časy běhů programu jsou zachyceny v tabulce 5.13. Počty kroků řešení pak v tabulce 5.14.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.052	0.24	1.72	5.89	13.9	43.3	101.8
$\tilde{x}$	0	0.05	0.24	1.72	5.91	13.9	41.5	103.1
$Q_{0.25}$	0	0.05	0.23	1.64	5.54	13.2	39.6	94.5
$Q_{0.75}$	0.01	0.06	0.25	1.81	6.13	14.5	47.5	106.5

Tabulka 5.13: Logaritmický počet uch, lineárně rostoucí rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	7.46×10^2	3.80×10^4	2.16×10^5	1.67×10^6	4.02×10^6	1.03×10^7
$\tilde{x}$	7.17×10^2	3.79×10^4	2.08×10^5	1.72×10^6	$3.8\ \times10^6$	1.03×10^7
$Q_{0.25}$	6.25×10^2	3.38×10^4	1.85×10^5	1.43×10^6	3.24×10^6	8.69×10^6
$Q_{0.75}$	8.54×10^2	4.10×10^4	2.39×10^5	1.83×10^6	4.62×10^6	1.19×10^7

Tabulka 5.14: Logaritmický počet uch, lineárně rostoucí rozložení délek uch: Počet kroků řešení

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999912 a mezi časem běhu programu a počtem kroků řešení je 0.998215.

5.8 Odmocninový počet uch, lineárně rostoucí rozložení délek uch

Počet uch tohoto případu je k_{sqrt}(rovnice 2.11) a rozložení jejich délek uvažme rovnoměrné rozložení(rovnice 2.14). Dosazením těchto hodnot do rovnice 1.2 získáme počet kroků algoritmu:

$$\frac{2}{3}V^3+\frac{32}{15}V^{5/2}+\frac{233}{30}V^2+\frac{281}{15}V^{3/2}+\frac{643}{30}V+\frac{169}{15}\sqrt{V}+2 \quad (5.8)$$

Časy běhů programu jsou zachyceny v tabulce 5.15. Počty kroků řešení pak v tabulce 5.16.

$\|V\|$	20	60	100	200	300	400	600	800
E	0.003	0.045	0.18	1.26	4.08	9.38	30.3	70.7
$\tilde{x}$	0	0.04	0.18	1.26	4.06	9.4	30.3	71.4
$Q_{0.25}$	0	0.04	0.17	1.2	3.9	8.9	29.2	67.8
$Q_{0.75}$	0.01	0.05	0.2	1.31	4.2	9.7	31.9	72.1

Tabulka 5.15: Odmocninový počet uch, lineárně rostoucí rozložení délek uch: Čas běhu programu

$\|V\|$	20	100	200	400	600	800
E	2.96×10^2	1.10×10^4	4.77×10^4	2.12×10^5	5.01×10^5	8.74×10^5
$\tilde{x}$	2.90×10^2	1.06×10^4	4.84×10^4	2.11×10^5	4.82×10^5	9.06×10^5
$Q_{0.25}$	2.51×10^2	8.96×10^3	4.17×10^4	1.76×10^5	4.47×10^5	7.51×10^5
$Q_{0.75}$	3.32×10^2	1.26×10^4	5.28×10^4	2.35×10^5	5.67×10^5	9.82×10^5

Tabulka 5.16: Odmocninový počet uch, lineárně rostoucí rozložení délek uch: Počet kroků řešení

Korelační koeficient mezi počtem kroků algoritmu a časem běhu programu je 0.999812 a mezi časem běhu programu a počtem kroků řešení je 0.986371.

6. Střední hodnoty a mediány

$\lvert V\rvert$	10	20	40	60	80	100	150	200
$\mathbf{E}^{\rightarrow}_{const}$	0.0005	0.0033	0.0229	0.0783	0.1937	0.3932	1.3836	3.3658
$\mathbf{E}^{\rightarrow}_{lin}$	0.0006	0.0033	0.0228	0.0765	0.1754	0.3376	1.1304	2.7166
$\mathbf{E}^{\rightarrow}_{log}$	0.0005	0.0031	0.0172	0.0545	0.1231	0.2402	0.7296	1.729
$\mathbf{E}^{\rightarrow}_{sqrt}$	0.0005	0.0027	0.0144	0.0477	0.1035	0.1845	0.5684	1.2851
$\mathbf{E}^{\searrow}_{const}$	0.0005	0.0035	0.0274	0.0936	0.2325	0.4745	1.683	4.0793
$\mathbf{E}^{\searrow}_{lin}$	0.0004	0.003	0.021	0.0622	0.146	0.2717	0.904	2.1636
$\mathbf{E}^{\searrow}_{log}$	0.0005	0.0034	0.0221	0.0624	0.1513	0.2932	0.8881	2.1533
$\mathbf{E}^{\searrow}_{sqrt}$	0.0005	0.003	0.0173	0.0507	0.1147	0.2076	0.6352	1.425
$\mathbf{E}^{\nearrow}_{const}$	0.0006	0.0028	0.0232	0.0776	0.19	0.3773	1.3233	3.2453
$\mathbf{E}^{\nearrow}_{lin}$	0.0003	0.0031	0.0181	0.0591	0.1331	0.2475	0.8172	1.9468
$\mathbf{E}^{\nearrow}_{log}$	0.0004	0.0029	0.0176	0.0523	0.1226	0.2425	0.7278	1.7239
$\mathbf{E}^{\nearrow}_{sqrt}$	0.0005	0.0026	0.0144	0.0451	0.1038	0.1836	0.5577	1.2555
$\mathbf{E}_{rand}$	0.0005	0.0033	0.0227	0.0719	0.1743	0.3364	1.1283	2.7166
$\lvert V\rvert$	250	300	350	400	500	600	700	800
$\mathbf{E}^{\rightarrow}_{const}$	6.6348	11.686	18.295	28.295	55.653	99.14	153.56	230.74
$\mathbf{E}^{\rightarrow}_{lin}$	5.3665	9.449	15.271	23.107	46.306	81.717	130.318	193.3
$\mathbf{E}^{\rightarrow}_{log}$	3.39	5.895	9.293	14.115	24.446	43.773	64.978	96.21
$\mathbf{E}^{\rightarrow}_{sqrt}$	2.4463	4.118	6.518	9.627	18.644	31.95	50.034	72.17
$\mathbf{E}^{\searrow}_{const}$	8.1724	14.313	22.667	34.604	67.758	119.517	185.923	276.2
$\mathbf{E}^{\searrow}_{lin}$	4.3005	7.518	12.192	18.593	37.12	65.865	105.205	155.25
$\mathbf{E}^{\searrow}_{log}$	4.2727	7.452	11.784	17.642	30.948	53.373	85.693	121.84
$\mathbf{E}^{\searrow}_{sqrt}$	2.6699	4.479	6.922	10.271	19.753	33.974	51.061	73.9
$\mathbf{E}^{\nearrow}_{const}$	6.3091	11.227	17.571	27.208	52.72	93.195	150.117	220.2
$\mathbf{E}^{\nearrow}_{lin}$	3.8924	6.794	10.962	16.569	33.375	58.612	95.651	143.39
$\mathbf{E}^{\nearrow}_{log}$	3.3873	5.886	9.361	13.903	24.461	43.303	68.448	101.82
$\mathbf{E}^{\nearrow}_{sqrt}$	2.4013	4.078	6.317	9.384	18.13	30.339	49.39	70.67
$\mathbf{E}_{rand}$	5.3769	9.47	15.269	23.083	46.266	82.195	133.054	201.15

Tabulka 6.1: Střední hodnoty časů běhu programu

$\lvert V\rvert$	10	20	40	60	80	100
$\mathbf{E}^{\rightarrow}_{const}$	8.3×10^{1}	8.89×10^{2}	7.11×10^{3}	2.4×10^{4}	5.87×10^{4}	1.17×10^{5}
$\mathbf{E}^{\rightarrow}_{lin}$	5.1×10^{1}	1.54×10^{2}	4.60×10^{2}	8.43×10^{2}	1.27×10^{3}	1.74×10^{3}
$\mathbf{E}^{\rightarrow}_{log}$	9×10^{1}	8.18×10^{2}	3.48×10^{3}	8.69×10^{3}	1.84×10^{4}	3.63×10^{4}
$\mathbf{E}^{\rightarrow}_{sqrt}$	9.4×10^{1}	3.42×10^{2}	1.38×10^{3}	4.2×10^{3}	7.22×10^{3}	1.02×10^{4}
$\mathbf{E}^{\searrow}_{const}$	1.25×10^{2}	8.95×10^{2}	1×10^{4}	3.44×10^{4}	8.53×10^{4}	1.72×10^{5}
$\mathbf{E}^{\searrow}_{lin}$	5.6×10^{1}	2.14×10^{2}	6.99×10^{2}	1.32×10^{3}	2.11×10^{3}	2.89×10^{3}
$\mathbf{E}^{\searrow}_{log}$	1.24×10^{2}	8.83×10^{2}	5.91×10^{3}	1.28×10^{4}	3.17×10^{4}	6.16×10^{4}
$\mathbf{E}^{\searrow}_{sqrt}$	8.7×10^{1}	4.39×10^{2}	2.11×10^{3}	6.56×10^{3}	1.19×10^{4}	1.74×10^{4}
$\mathbf{E}^{\nearrow}_{const}$	1.2×10^{2}	6.6×10^{2}	7.61×10^{3}	2.49×10^{4}	6×10^{4}	1.18×10^{5}
$\mathbf{E}^{\nearrow}_{lin}$	4.4×10^{1}	1.82×10^{2}	5.22×10^{2}	1.04×10^{3}	1.58×10^{3}	2.17×10^{3}
$\mathbf{E}^{\nearrow}_{log}$	9.2×10^{1}	7.46×10^{2}	3.43×10^{3}	7.85×10^{3}	1.9×10^{4}	3.8×10^{4}
$\mathbf{E}^{\nearrow}_{sqrt}$	6.4×10^{1}	2.96×10^{2}	1.22×10^{3}	4.2×10^{3}	8.03×10^{3}	1.11×10^{4}
$\mathbf{E}_{rand}$	5.1×10^{1}	1.56×10^{2}	4.51×10^{2}	8.47×10^{2}	1.27×10^{3}	1.74×10^{3}

$\lvert V\rvert$	150	200	250	300	350
$\mathbf{E}^{\rightarrow}_{const}$	3.98×10^{5}	9.56×10^{5}	1.86×10^{6}	3.26×10^{6}	5.06×10^{6}
$\mathbf{E}^{\rightarrow}_{lin}$	3.04×10^{3}	4.49×10^{3}	6.03×10^{3}	7.68×10^{3}	9.41×10^{3}
$\mathbf{E}^{\rightarrow}_{log}$	8.86×10^{4}	2.03×10^{5}	4×10^{5}	6.89×10^{5}	1.09×10^{6}
$\mathbf{E}^{\rightarrow}_{sqrt}$	2.67×10^{4}	4.9×10^{4}	8.2×10^{4}	1.17×10^{5}	1.73×10^{5}
$\mathbf{E}^{\searrow}_{const}$	6.11×10^{5}	1.44×10^{6}	2.87×10^{6}	5.02×10^{6}	7.94×10^{6}
$\mathbf{E}^{\searrow}_{lin}$	5.24×10^{3}	7.86×10^{3}	1.08×10^{4}	1.42×10^{4}	1.72×10^{4}
$\mathbf{E}^{\searrow}_{log}$	1.55×10^{5}	3.77×10^{5}	7.37×10^{5}	1.26×10^{6}	2.05×10^{6}
$\mathbf{E}^{\searrow}_{sqrt}$	4.42×10^{4}	8.78×10^{4}	1.52×10^{5}	2.18×10^{5}	3.21×10^{5}
$\mathbf{E}^{\nearrow}_{const}$	4.09×10^{5}	9.95×10^{5}	1.9×10^{6}	3.33×10^{6}	5.25×10^{6}
$\mathbf{E}^{\nearrow}_{lin}$	3.77×10^{3}	5.43×10^{3}	7.38×10^{3}	9.4×10^{3}	1.14×10^{4}
$\mathbf{E}^{\nearrow}_{log}$	9.03×10^{4}	2.16×10^{5}	4.15×10^{5}	7.22×10^{5}	1.16×10^{6}
$\mathbf{E}^{\nearrow}_{sqrt}$	2.57×10^{4}	4.77×10^{4}	8.22×10^{4}	1.19×10^{5}	1.64×10^{5}
$\mathbf{E}_{rand}$	3.04×10^{3}	4.5×10^{3}	6.07×10^{3}	7.69×10^{3}	9.38×10^{3}

$\lvert V\rvert$	400	500	600	700	800
$\mathbf{E}^{\rightarrow}_{const}$	7.91×10^{6}	1.54×10^{7}	2.74×10^{7}	4.09×10^{7}	6.33×10^{7}
$\mathbf{E}^{\rightarrow}_{lin}$	1.12×10^{4}	1.49×10^{4}	1.88×10^{4}	2.27×10^{4}	2.7×10^{4}
$\mathbf{E}^{\rightarrow}_{log}$	1.65×10^{6}	2.24×10^{6}	4.17×10^{6}	6.27×10^{6}	9.35×10^{6}
$\mathbf{E}^{\rightarrow}_{sqrt}$	2.06×10^{5}	3.51×10^{5}	4.94×10^{5}	7.14×10^{5}	9.52×10^{5}
$\mathbf{E}^{\searrow}_{const}$	1.21×10^{7}	2.33×10^{7}	4.06×10^{7}	6.39×10^{7}	9.61×10^{7}
$\mathbf{E}^{\searrow}_{lin}$	2.09×10^{4}	2.79×10^{4}	3.56×10^{4}	4.36×10^{4}	5.23×10^{4}
$\mathbf{E}^{\searrow}_{log}$	2.99×10^{6}	4.4×10^{6}	7.39×10^{6}	1.25×10^{7}	1.92×10^{7}
$\mathbf{E}^{\searrow}_{sqrt}$	4.07×10^{5}	7.06×10^{5}	1.06×10^{6}	1.39×10^{6}	1.95×10^{6}
$\mathbf{E}^{\nearrow}_{const}$	8.17×10^{6}	1.54×10^{7}	2.71×10^{7}	4.38×10^{7}	6.45×10^{7}
$\mathbf{E}^{\nearrow}_{lin}$	1.37×10^{4}	1.81×10^{4}	2.26×10^{4}	2.78×10^{4}	3.22×10^{4}
$\mathbf{E}^{\nearrow}_{log}$	1.68×10^{6}	2.35×10^{6}	4.03×10^{6}	6.38×10^{6}	1.03×10^{7}
$\mathbf{E}^{\nearrow}_{sqrt}$	2.13×10^{5}	3.34×10^{5}	5.02×10^{5}	7.34×10^{5}	8.75×10^{5}
$\mathbf{E}_{rand}$	1.12×10^{4}	1.49×10^{4}	1.87×10^{4}	2.27×10^{4}	2.68×10^{4}

Tabulka 6.2: Střední hodnoty počtu kroků řešení

$\lvert V\rvert$	10	20	40	60	80	100	150	200
$\tilde{x}^{\rightarrow}_{const}$	0	0	0.02	0.08	0.19	0.39	1.38	3.355
$\tilde{x}^{\rightarrow}_{lin}$	0	0	0.02	0.08	0.18	0.34	1.13	2.71
$\tilde{E}^{\rightarrow}_{log}$	0	0	0.02	0.05	0.12	0.24	0.73	1.72
$\tilde{x}^{\rightarrow}_{sqrt}$	0	0	0.01	0.05	0.1	0.18	0.57	1.29
$\tilde{x}^{\searrow}_{const}$	0	0	0.03	0.09	0.23	0.47	1.68	4.07
$\tilde{x}^{\searrow}_{lin}$	0	0	0.02	0.06	0.15	0.27	0.91	2.165
$\tilde{x}^{\searrow}_{log}$	0	0	0.02	0.06	0.15	0.29	0.89	2.165
$\tilde{x}^{\searrow}_{sqrt}$	0	0	0.02	0.05	0.11	0.21	0.63	1.42
$\tilde{x}^{\nearrow}_{const}$	0	0	0.02	0.08	0.19	0.375	1.325	3.25
$\tilde{x}^{\nearrow}_{lin}$	0	0	0.02	0.06	0.13	0.25	0.82	1.955
$\tilde{x}^{\nearrow}_{log}$	0	0	0.02	0.05	0.12	0.24	0.73	1.72
$\tilde{x}^{\nearrow}_{sqrt}$	0	0	0.01	0.04	0.1	0.18	0.555	1.26
$\tilde{x}_{rand}$	0	0	0.02	0.07	0.17	0.34	1.13	2.715

$\lvert V\rvert$	250	300	350	400	500	600	700	800
$\tilde{x}^{\rightarrow}_{const}$	6.61	11.595	18.09	28.545	55.795	98.96	154.075	230.935
$\tilde{x}^{\rightarrow}_{lin}$	5.36	9.455	15.27	23.14	46.31	81.01	130.54	192.855
$\tilde{x}^{\rightarrow}_{log}$	3.4	5.89	9.26	14.13	24.56	44.07	64.455	96.655
$\tilde{x}^{\rightarrow}_{sqrt}$	2.44	4.11	6.5	9.57	18.885	32.44	50.63	72.375
$\tilde{x}^{\searrow}_{const}$	8.14	14.285	22.6	34.755	67.85	118.66	186.005	278.59
$\tilde{x}^{\searrow}_{lin}$	4.31	7.525	12.21	18.56	36.96	65.84	105.17	154.94
$\tilde{x}^{\searrow}_{log}$	4.31	7.46	11.85	17.62	30.885	53.85	86.295	122.095
$\tilde{x}^{\searrow}_{sqrt}$	2.67	4.475	6.92	10.245	19.895	34.32	51.065	74.55
$\tilde{x}^{\nearrow}_{const}$	6.26	11.18	17.59	27.07	52.645	93.16	150.4	218.375
$\tilde{x}^{\nearrow}_{lin}$	3.88	6.805	10.9	16.66	33.485	58.35	96	143.81
$\tilde{x}^{\nearrow}_{log}$	3.35	5.905	9.27	13.93	24.375	41.51	67.035	103.11
$\tilde{x}^{\nearrow}_{sqrt}$	2.4	4.06	6.32	9.395	18.355	30.31	49.03	71.43
$\tilde{x}_{rand}$	5.38	9.465	15.285	23.025	46.24	82.36	133.065	201.045

Tabulka 6.3: Mediány časů běhu programu

$\lvert V\rvert$	10	20	40	60	80	100
$\tilde{x}^{\rightarrow}_{const}$	$8.6\ \times10^{1}$	$9.2\ \times10^{2}$	6.92×10^{3}	2.37×10^{4}	5.78×10^{4}	1.19×10^{5}
$\tilde{x}^{\rightarrow}_{lin}$	$5\ \times10^{1}$	1.55×10^{2}	$4.6\ \times10^{2}$	8.47×10^{2}	1.27×10^{3}	1.74×10^{3}
$\tilde{x}^{\rightarrow}_{log}$	$8.8\ \times10^{1}$	8.14×10^{2}	3.43×10^{3}	8.65×10^{3}	1.84×10^{4}	3.55×10^{4}
$\tilde{x}^{\rightarrow}_{sqrt}$	$9.9\ \times10^{1}$	3.46×10^{2}	$1.4\ \times10^{3}$	4.22×10^{3}	7.19×10^{3}	1.01×10^{4}
$\tilde{x}^{\searrow}_{const}$	1.19×10^{2}	8.58×10^{2}	9.96×10^{3}	3.43×10^{4}	8.44×10^{4}	$1.7\ \times10^{5}$
$\tilde{x}^{\searrow}_{lin}$	$5.4\ \times10^{1}$	2.12×10^{2}	6.96×10^{2}	1.31×10^{3}	2.11×10^{3}	2.87×10^{3}
$\tilde{x}^{\searrow}_{log}$	1.27×10^{2}	8.73×10^{2}	5.76×10^{3}	1.28×10^{4}	3.17×10^{4}	6.09×10^{4}
$\tilde{x}^{\searrow}_{sqrt}$	$9\ \times10^{1}$	4.27×10^{2}	$2.1\ \times10^{3}$	6.51×10^{3}	1.18×10^{4}	1.73×10^{4}
$\tilde{x}^{\nearrow}_{const}$	$1.2\ \times10^{2}$	6.71×10^{2}	7.53×10^{3}	2.38×10^{4}	5.93×10^{4}	1.15×10^{5}
$\tilde{x}^{\nearrow}_{lin}$	$4.2\ \times10^{1}$	1.82×10^{2}	5.21×10^{2}	1.04×10^{3}	1.57×10^{3}	2.15×10^{3}
$\tilde{x}^{\nearrow}_{log}$	$9.8\ \times10^{1}$	7.17×10^{2}	3.39×10^{3}	7.72×10^{3}	$1.9\ \times10^{4}$	3.79×10^{4}
$\tilde{x}^{\nearrow}_{sqrt}$	$6.6\ \times10^{1}$	$2.9\ \times10^{2}$	$1.2\ \times10^{3}$	4.22×10^{3}	8.05×10^{3}	1.06×10^{4}
$\tilde{x}_{rand}$	$5\ \times10^{1}$	1.55×10^{2}	4.52×10^{2}	8.52×10^{2}	1.27×10^{3}	1.74×10^{3}

$\lvert V\rvert$	150	200	250	300	350
$\tilde{x}^{\rightarrow}_{const}$	3.96×10^{5}	9.64×10^{5}	1.85×10^{6}	3.22×10^{6}	4.97×10^{6}
$\tilde{x}^{\rightarrow}_{lin}$	3.05×10^{3}	4.48×10^{3}	6.03×10^{3}	7.67×10^{3}	9.38×10^{3}
$\tilde{x}^{\rightarrow}_{log}$	8.86×10^{4}	2.02×10^{5}	3.98×10^{5}	6.91×10^{5}	1.07×10^{6}
$\tilde{x}^{\rightarrow}_{sqrt}$	2.66×10^{4}	4.87×10^{4}	8.19×10^{4}	1.15×10^{5}	1.78×10^{5}
$\tilde{x}^{\searrow}_{const}$	6.07×10^{5}	1.45×10^{6}	2.86×10^{6}	4.99×10^{6}	7.97×10^{6}
$\tilde{x}^{\searrow}_{lin}$	5.22×10^{3}	7.88×10^{3}	1.09×10^{4}	1.42×10^{4}	1.73×10^{4}
$\tilde{x}^{\searrow}_{log}$	1.55×10^{5}	3.74×10^{5}	7.37×10^{5}	1.26×10^{6}	2.03×10^{6}
$\tilde{x}^{\searrow}_{sqrt}$	4.39×10^{4}	8.82×10^{4}	1.52×10^{5}	2.21×10^{5}	3.25×10^{5}
$\tilde{x}^{\nearrow}_{const}$	4.04×10^{5}	9.82×10^{5}	1.87×10^{6}	3.28×10^{6}	5.26×10^{6}
$\tilde{x}^{\nearrow}_{lin}$	3.77×10^{3}	5.44×10^{3}	7.37×10^{3}	9.35×10^{3}	1.14×10^{4}
$\tilde{x}^{\nearrow}_{log}$	9.12×10^{4}	2.08×10^{5}	4.09×10^{5}	7.01×10^{5}	$1.1\ \times10^{6}$
$\tilde{x}^{\nearrow}_{sqrt}$	$2.5\ \times10^{4}$	4.84×10^{4}	8.18×10^{4}	1.19×10^{5}	1.61×10^{5}
$\tilde{x}_{rand}$	3.04×10^{3}	$4.5\ \times10^{3}$	6.05×10^{3}	7.69×10^{3}	$9.4\ \times10^{3}$

$\lvert V\rvert$	400	500	600	700	800
$\tilde{x}^{\rightarrow}_{const}$	$8\ \times10^{6}$	1.53×10^{7}	2.74×10^{7}	4.13×10^{7}	6.26×10^{7}
$\tilde{x}^{\rightarrow}_{lin}$	1.12×10^{4}	1.49×10^{4}	1.87×10^{4}	2.27×10^{4}	2.71×10^{4}
$\tilde{x}^{\rightarrow}_{log}$	1.65×10^{6}	2.11×10^{6}	4.07×10^{6}	6.07×10^{6}	9.29×10^{6}
$\tilde{x}^{\rightarrow}_{sqrt}$	2.04×10^{5}	3.48×10^{5}	4.93×10^{5}	7.24×10^{5}	9.72×10^{5}
$\tilde{x}^{\searrow}_{const}$	1.22×10^{7}	2.33×10^{7}	4.04×10^{7}	$6.4\ \times10^{7}$	9.61×10^{7}
$\tilde{x}^{\searrow}_{lin}$	$2.1\ \times10^{4}$	2.78×10^{4}	3.55×10^{4}	4.33×10^{4}	5.22×10^{4}
$\tilde{x}^{\searrow}_{log}$	2.93×10^{6}	4.36×10^{6}	7.38×10^{6}	1.24×10^{7}	1.95×10^{7}
$\tilde{x}^{\searrow}_{sqrt}$	4.09×10^{5}	7.19×10^{5}	1.06×10^{6}	1.36×10^{6}	1.92×10^{6}
$\tilde{x}^{\nearrow}_{const}$	7.94×10^{6}	1.51×10^{7}	2.71×10^{7}	4.34×10^{7}	6.44×10^{7}
$\tilde{x}^{\nearrow}_{lin}$	1.37×10^{4}	1.81×10^{4}	2.28×10^{4}	2.76×10^{4}	$3.2\ \times10^{4}$
$\tilde{x}^{\nearrow}_{log}$	1.72×10^{6}	2.31×10^{6}	$3.8\ \times10^{6}$	6.11×10^{6}	1.03×10^{7}
$\tilde{x}^{\nearrow}_{sqrt}$	2.11×10^{5}	$3.4\ \times10^{5}$	4.82×10^{5}	7.22×10^{5}	9.07×10^{5}
$\tilde{x}_{rand}$	1.12×10^{4}	1.48×10^{4}	1.87×10^{4}	2.29×10^{4}	2.69×10^{4}

Tabulka 6.4: Mediány počtu kroků řešení

7. Použitý hardware a software

Experimentální výsledky byly pořízeny na hardwaru

Procesor: Intel® i5-2500k 3.3GHz

Paměť: Kingston® 2×4GB 1600Hz Cl9

K vypracování této práce jsem použil následující software

- Wolfram Mathematica® 9 Student edition
- gcc 4.7.2
- TexMaker 3.5.2
- Linux 3.9.2.fc18.x86_64

Experimentální vyhodnocení trvalo přibližně dva dny.

8. Obsah CD

Text V této složce je pdf soubor s textem bakalářské práce.

Program V této složce jsou zdrojové soubory a spustitelná verze programu, kterým bylo provedeno experimentální vyhodnocení. Program je možno zkompilovat v Unixovém prostředí pomocí příkazu MAKE.

Data V této složce jsou všechny experimentální výsledky, ze kterých byly sestaveny grafy a tabulky v práci. Hodnoty časů a kroků jsou pro každé rozložení uch ve svém vlastním souboru.