Report k Log úkolu

# Řešení

Zvolil jsem A\* algoritmus, který využívá iterative deepeningu, aby šetřil pamětí. Algoritmus je napsaný v C++ a je rozvržený tak, aby bylo možné ho použít i na jiné problémy. Poskytuje interface pro stav (IState), akci (IAction) a definici problému (IProblem).

IProblem vyžaduje implementaci:

* IState const\* GetInitialState() const
* bool IsGoalState(IState const\* state) const
* void EnumeratePossibleActions(IState const\* state, std::queue<std::pair<IAction\*, IState\*>>& possibleActions) const

Pro IState i IAction je potřeba funkce IState/IAction\* Clone() const.

Tyto tři třídy jsou implementovány třídami specifickými danému problému. Pro případ **Log** problému jsem vytvořil LogProblem pro IProblem, LogConfiguration pro IState a Action pro IAction. LogProblem je dále definován počátečním stavem a LogSetting třídou, která reprezentuje konstanty v instanci problému - zde jsou definována místa, města a letiště, zatímco LogConfiguration obsahuje informaci o dodávkách, letadlech a balíčcích.

Součástí LogConfiguration je funkce static int ComputeHeuristic(...), kde se počítá odhad zbývající vzdálenosti do cílového stavu. Tato funkce sestává ze tří částí:

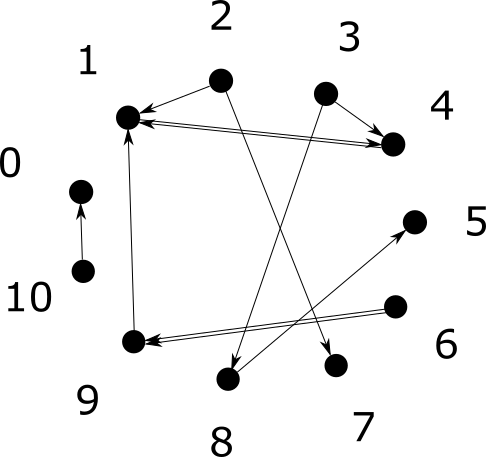
1. Přesný výpočet nákladů na přesun balíčků z a do vozidel. Tento výpočet může být naprosto přesný, protože víme, že každý balíček, který není na svém místě, bude nutné naložit do dodávky (pokud zdrojové i cílové místo není letiště), každý balíček, který je v dodávce, bude nutné vyložit, každý balíček, který je v jiném městě, než být má, bude nutné naložit a vyložit z letadla, a pokud musí balíček přeletět a není jeho destinace letiště, bude ho potřeba ještě naložit a vyložit z dodávky.
2. Odhad počtu jízd dodávek. Zde se nejprve vytvoří orientovaný graf (pro každé město zvlášť), kde každá hrana reprezentuje aspoň jeden balíček cestující přímo ze své pozice do své destinace. V tomto grafu jsou pak nalezeny smyčky, které zapříčíní, že některá místa budou muset být navštívena vícekrát. A nakonec se spočítají místa, která musí aspoň jednou být navštívena dodávkou (tedy buď odtud potřebuje balíček odjet, nebo sem potřebuje přijet). Dále jsou také započítány aktuální pozice dodávek. Aby se omezil počet nerozpoznaných případů, kdy nejkratší cesta vede přes stav, ve kterém je dodávka naplněna a musí se pro zbytek balíčků vrátit, orientovaný graf zbavený o cykly je ještě transformován na vrstvy, kde v první vrstvě jsou místa, do kterých nevede žádná hrana, ale zároveň z nich aspoň jedna hrana vede a každá další vrstva je množina míst, do nichž vedou jen hrany míst, ktereé u jsou v nějaké předchozí vrstvě. Idea je taková, že pokud z jedné vrstvy do druhé vede více hran než je kapacita dodávky, pak se bude muset dodávka vrátit. Tato poslední část opravdu zlepšila odhady o hodně, ale ideální také není. V některých případech je možné, že pokud pokud se části vrstvového grafu posunou do jiných vrstev (samozřejmě bez toho, aby porušily, že hrany vedou jen z dřívějších do pozdějších vrstev), je možné tento vrstevnatý graf zoptimalizovat tak, že “úzkých hrdel” bude méně. Naneštěstí existuje velmi mnoho možností, jak rozumně vybrat takovou část grafu a jak ji posunout – ještě více možností, jak více částí grafu posunout naráz, aby vyšla co nejlevnější kombinace. Bohužel mě nenapadlo, jak toto nějak elegantně vyřešit.
3. Odhad počtu letů letadel. Postup je analogický k postupu v bodu 2.

# Měření

Zkusil jsem zvyšovat počet míst a zároveň počet balíčků. V testech bylo pouze jedno město a letadla tedy nebyla vůbec nutná. Testy byly automaticky generovány, tedy počáteční pozice a destinace balíčků jsou náhodné.

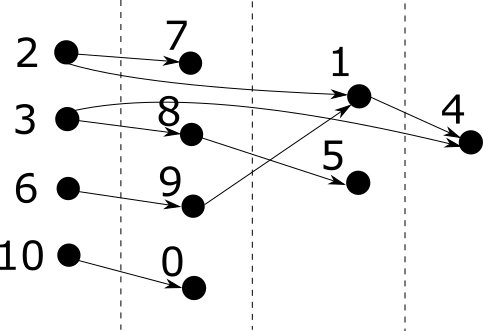
V grafu je vidět, že časová náročnost algoritmu postupně stoupá s přibývajícími místy/balíčky. Jsou ale také občas vidět anomálie jako třeba ta na počtu míst 11 – tato konkrétní konfigurace je příkladem takové, jejíž vrstvový graf by bylo možné lépe uspořádat. Se zvětšujícím se počtem míst se i zvyšuje šance na delší výpočet a stává se i, že dříve dojde paměť než algoritmus skončí.

Příklad 11 vypadá takto:



1->4 a 4->1 tvoří cyklus a 6->9 je dvojitá hrana.

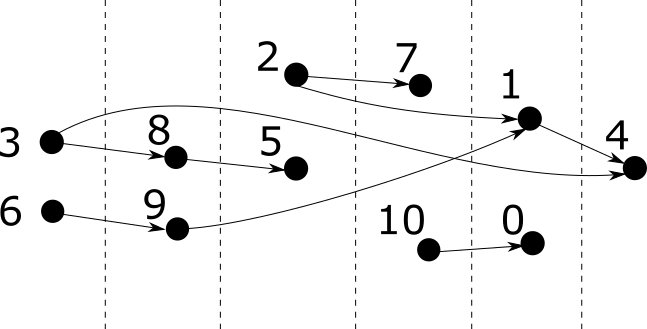
V tomto grafu je nutné odstranit cyklus pro další krok – odstraníme tedy třeba hranu 4->1 a ponecháme hranu 1->4. Poté vytvoříme vrstevnatý graf.



V přechodech vrstev vidíme, že z první do druhé je 7 hran (6->9 je dvojitá), podle tohoto grafu bychom tedy měli potřebovat, aby se dodávka do této vrstvy jednou vrátila. Z druhé do třetí vrstvy jsou 4 hrany, a ze třetí do čtvrté jen 2 – není tedy zapotřebí žádných dalších návratů.

Je ale vidět, že graf lze přeorganizovat rozšířením o nové vrstvy bez porušení dopřednosti hran tak, aby počet hran pojící jakékoliv dvě po sobě jdoucí vrstvy byl nejvýše 4.

Zde je takový graf:



Je vidět, že není porušena ani dopřednost hran, ani kapacita přenášená do dalších vrstev. Podle tohoto diagramu je možné navštívit všechna místa pouze jednou a doručit všechny balíčky do jejich destinací (s výjimkou cyklu, který jsme přerušili dříve). Bohužel původní graf, se kterým moje heuristika pracuje, tvrdí, že alespoň jeden návrat bude nutný, proto bohužel není tato heuristika přípustná – nemusí najít optimální řešení. Pokud tuto část heuristiky (vrstevnatý graf) vynecháme, dojdeme k velmi vysokým vypočetním časům mnohem dříve, protože se realtivně často stává, že se dodávka naplní a algoritmus pak musí prohledávat nemálo špatně odhadnutých stavů.

Měření s fixním počtem míst v každém městě a postupně se zvyšujícím počtem měst bude analogické k měrení s jedním městem a zvyšujícím se počtem míst, ale výkyvy kvůli plnému nákladu budou nastávat méně často, protože letadla mohou nést větší náklad.

# Závěr

Řešení různých konfigurací dle mého názoru vyplývají rychle, až na případy, kdy nějaké vozidlo naplní svoji kapacitu. Pak je nutné spoléhat na heuristiku, která není přípustná a tedy nejen, že nemusí nalézt optimální řešení, je i možné, že hledání výsledku bude trvat také příliš dlouho. Bohužel se mi nepodařilo přijít na to, jak efektivně zoptimalizovat vrstevnatý graf.

V dřívějších verzích mého programu také byly velkým problémem cykly v orientovaném grafu potřebných přeprav, čehož jsem byl schopen se zbavit pomocí klasické reprezentace orientovaného grafu v programu a DFS, nalezení těchto cyklů a jejich připočítáním k odhadovanému potřebnému počtu jízd/letů.