

Optimalisace záchranné služby

ZÁVĚREČNÁ ZPRÁVA

AUTOR: MICHAL FRDLÍK

1. Zadání

Cílem semestrální práce je optimalisovat efektivitu záchranné služby města. Město je reprezentováno jeho mapou, resp. ohodnoceným grafem, kde ohodnocení hran představuje „rychlost“ silničního spojení mezi dvěma posicemi, a vrcholy představují posice. Posice mohou být trojího typu – prázdná posice, pacient a nemocnice. Každý pacient je ohodnocen číslem, které představuje dobu, po kterou je schopen přežít bez pomoci lékařů v nemocnici. Nemocnice disponuje konstantním množstvím sanitek, které samy disponují konstantní kapacitou a konstantním počtem možných návratů do nemocnice.

Sanitka se pohybuje mezi vrcholy po orientovaných hranách, přičemž projede-li sanitka hranou, která je ohodnocena n , uplyne v systému čas n a o tento čas se všem pacientům sníží jejich životnost. Pokud jakýkoliv pacient dosáhne životnosti 0, zemře, což způsobí penalizaci řešení – nechceme, aby pacienti umírali.

Efektivita řešení je dobrá, pokud je jeho „fitness“ největší. Hodnota „fitness“ sestává z penalizací za zemřelé pacienty (se záporným smyslem) a bonusů plynoucích ze životnosti pacientů úspěšně dovezených do nemocnice.

2. Optimalizační problém

Optimalizační problém tohoto zadání spočívá v nalezení nejlepších cest městem pro sanitky tak, aby dovezly do nemocnice co nejvíce živých pacientů, pokud možno s co největší životností, přičemž status pacientů [živý/mrtvý] má podstatně větší prioritu.

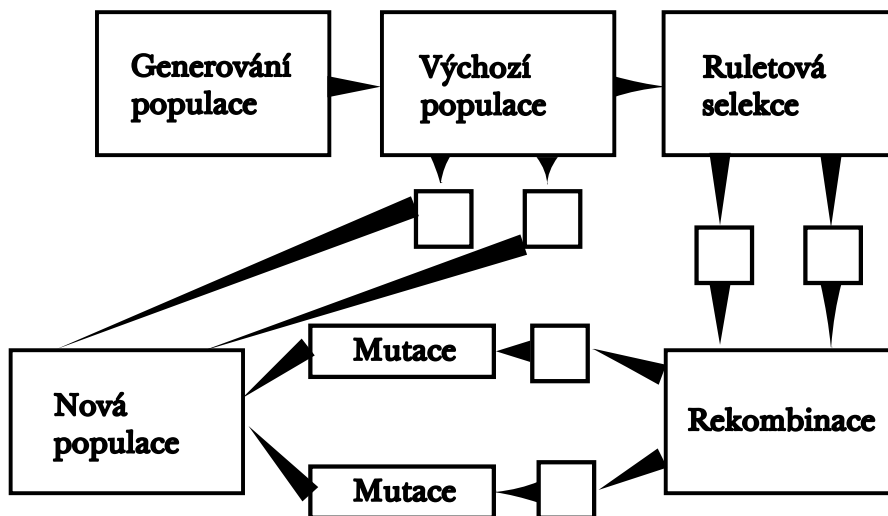
Vstupem necht je graf $G = (V, E, \Psi)$, zobrazení $\rho : V \rightarrow (X, t)$, které každému vrcholu přiřadí typ X a životnost t , a konstanty a , b a c , které určují počet sanitek a jejich kapacitu (v tomto pořadí) a počet možných návratů do nemocnice.

Omezením jsou životnosti pacientů a smysly jednotlivých hran (protože ulice mohou být i jednosměrné). Výstupem je výhodná cesta pro každou sanitku, uspořádání pacientů v sanitkách v čase a životnosti všech pacientů po jejich svozu.

3. Hlavní myšlenka řešení

Hlavní myšlenkou je genetický algoritmus, který je ilustrován [obrázkem 1](#). Genetický algoritmus nejdříve podle vstupní informace vygeneruje setříděnou množinu, která představuje počáteční populaci o n jedincích (viz [sekcí 3.1](#)). Následně provede ruletovou selekci (která je popsána v [sekcí 3.3](#)), tj. náhodně vybere dva jedince, přičemž každý má podle své fitness (fitness funkce je rozebrána v [sekcí 3.2](#)) přiřazenou poměrnou část intervalu $< 0; 100 >$. Tito jedinci poslouží jako vstupy procesu rekombinace, který je popsán v [sekcí 3.4](#). Tímto procesem vzniknou dva noví jedinci, z nich každý projde procesem mutace, který je popsán v [sekcí 3.5](#). Dva zmutovaní jedinci jsou potom umístěni do množiny, která představuje novou populaci. Tento proces se opakuje tak dlouho, dokud množina, která představuje novou populaci, nedosahuje velikosti staré populace minus 2. Poté se do nové populace přidají dva jedinci ze staré populace, kteří měli nejvyšší fitness (což vnáší do algoritmu prvek šlechtění a mírně snižuje jeho schopnost vyhnout se lokálním extrémům) a nová populace se ušta-

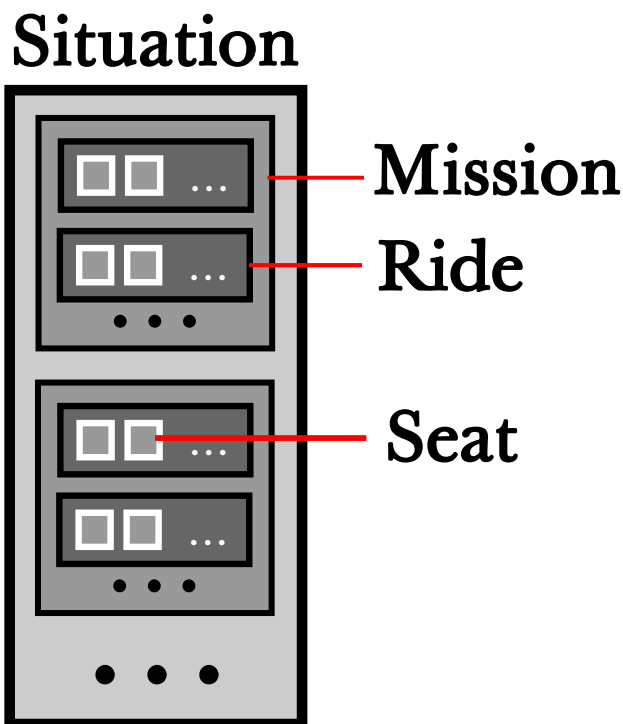
noví aktuální. Tento proces se opakuje po konstantní počet generací. Z poslední generace se potom vybere nejlepší jedinec, který je považován za výsledek.



Obr. 1. – Žjednodušené schéma genetického algoritmu

3.1. Jedinec

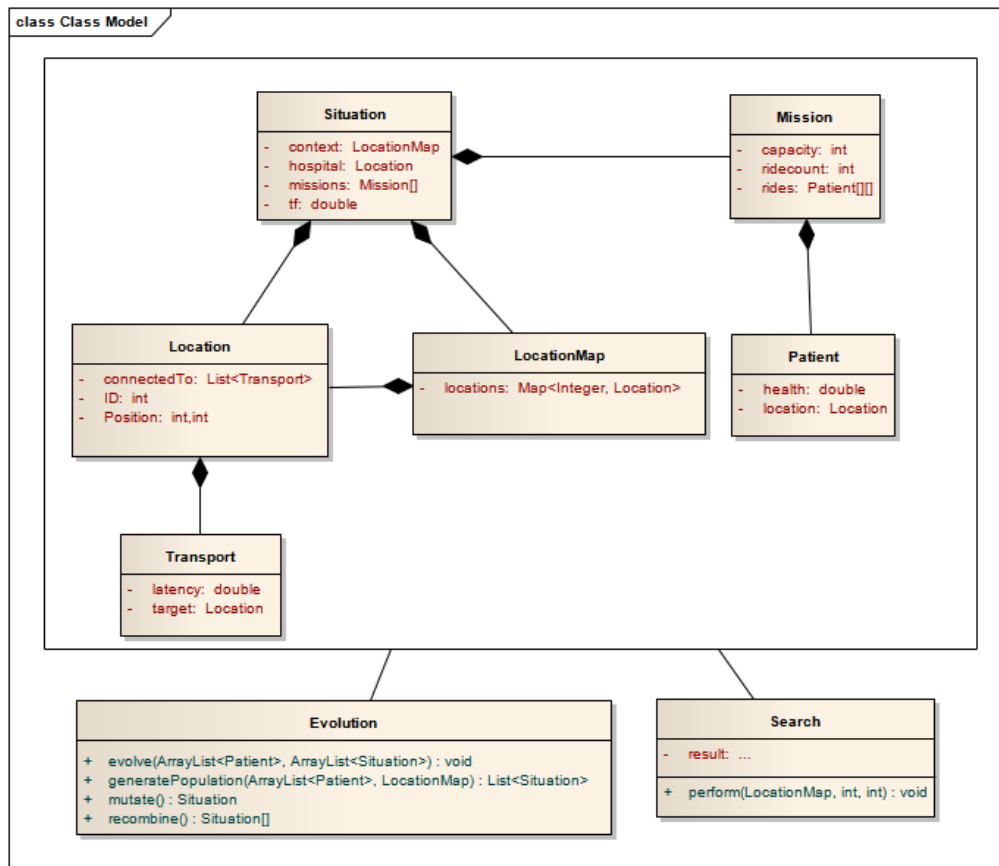
Správně navrhnout jedince bylo relativně složité a jedinec sám o sobě je relativně komplexní struktura, protože je zde požadavek na jeho dynamiku, tj. musí mít několik proměnných složek, které budou představovat jednotlivé sanitky, jízdy jednotlivých sanitek a pacienty v každé jízdě jednotlivé sanitky. Mé řešení je ilustrováno [obrázkem 2](#).



Obr. 2. – Representace jedince

Zde Mission představuje sanitku, která jezdí paralelně se všemi ostatními. Ride obsahuje informaci o jedné jízdě pro pacienty a zpět, přičemž tyto cesty probíhají sériově. Seat představuje jedno místo v jedné jízdě jedné sanitky a může být buď volné nebo obsazené. Pokud je místo volné, sanitka jede pro následujícího pacienta. Pokud jsou všechna místa volná, sanitka zůstává v nemocnici a pokud je poslední místo volné, sanitka se vrací zpět do nemocnice.

Implementačně je jedinec řešen strukturou s kontejnery pro další struktury s dalšími kontejnery dalších struktur. Implementaci jedince je možné si představit po zhlédnutí [obrázku 3](#), který je class diagramem programu.



Obr. 3. – Class diagram

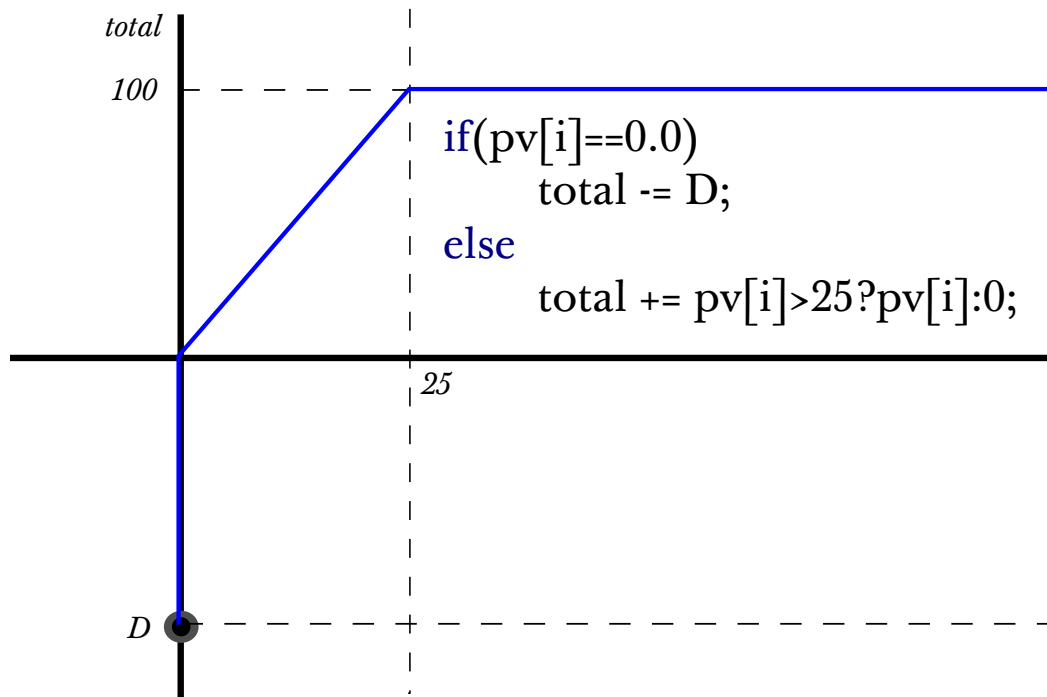
Třída Location představuje vrchol grafu, seznam vrcholů, které jsou s tímto vrcholem propojené, identifikační číslo a posici (to může být například zeměpisná šířka a délka). Informace o posici se uplatní v heuristice algoritmu A* pro hledání nejkratší cesty. Třída Transport určuje přechod mezi vrcholy, kterážto obsahuje i informaci o ohodnocení hrany (field latency) – průstupnost nebo také „rychlost“ cesty. Třída LocationMap obsahuje Mapu, která přiřazuje číslům ID jejich odpovídající lokaci. Třída Patient představuje pacienta, který se vyskytuje na určitém vrcholu s určitou životností. Třídy Mission a Situation potom odpovídají popisu z předchozí podsekcce.

3.2. Funkce fitness

Funkce fitness přiřazuje jedincům hodnoty od 0 do 100 podle toho, jak moc splňují naše požadavky, kde 0 představuje nejhorší možný scénář, tj. ten, kdy všichni pacienti zemřou, a kde 100 představuje nejlepší možný scénář, tj. ten, kdy se všichni pacienti dostanou do nemocnice s takovou životností, jakou měli na začátku. Každý jedinec má dvě hodnoty fitness

– normální fitness a relativní fitness, kde normální fitness představuje fitness výše popsanou a relativní fitness číselně popisuje, jakou výše kotoučového grafu zabírá daný jedinec poměrně vzhledem k ostatním jedincům v aktuální populaci.

Normální fitness jsem zpočátku navrhl lineární, ale byl jsem upozorněn na z toho plynoucí nevýhodu (řešení takové, že budeme mít v nemocnici jednoho pacienta s enormně vysokou životností a ostatní na pokraji smrti), takže jsem funkci změnil následujícím způsobem (obrázek 4).



Obr. 4. – Funkce fitness

Takto se fitness zvyšuje pro pacienty do čtvrtinové životnosti o jejich životnost a pro pacienty s vyšší životností je konstantní. Následek této změny popisuje druhý příklad později v textu.

3.3. Ruletová selekce

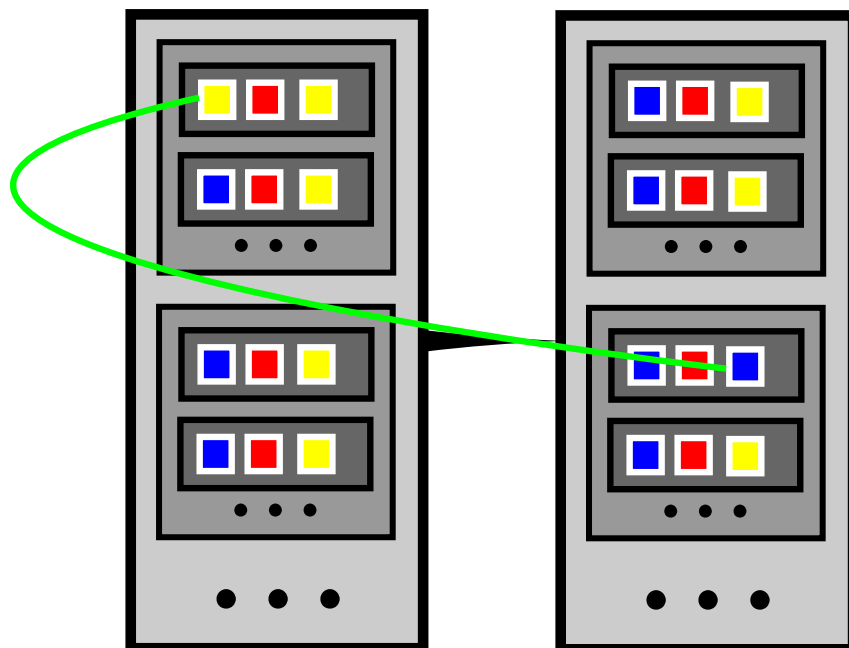
V ruletové selekci se přidělí každému jedinci poměrná část od 0 % do 100 % vzhledem k normální fitness ostatních jedinců, čili pro každého jedince existuje nenulová pravděpodobnost, že bude vybrán. Tato pravděpodobnost bude pochopitelně záviset na fitness jedinců a jejich počtu, čili pokud bude v populaci jeden jedinec s výbornou fitness a desítky jedinců s velmi špatnou fitness tak, že nejlepší jedinec bude mít relativní fitness 50 %, bude v jednom se dvěma případy vybrán některý z velmi špatných jedinců k rekombinaci. Volba fitness funkce je stěžejní pro každý adaptivní systém.

3.4. Rekombinace

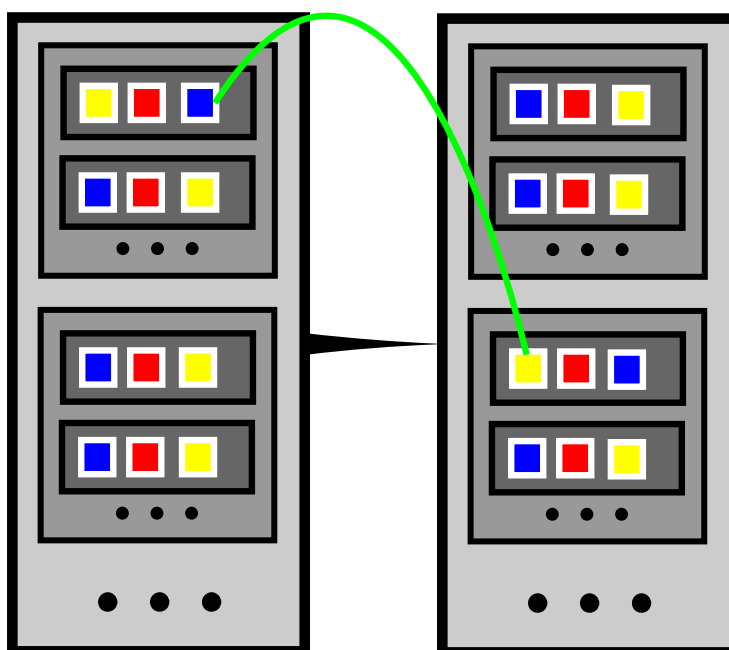
V procesu rekombinace (také známém jako „křížení“) se ruletovou selekcí vyberou dva jedinci a jejich genotypy se vzájemně modifikují způsobem, který je ilustrován [obrázkem 5](#) a [obrázkem 6](#).

Náhodně se vybere mise, jízda a pacient v obou sanitkách a prohodí se. Protože tímto může vzniknout situace, kdy budou v některé misi někteří pacienti obsaženi dvakrát, vyhledají se v obou misích příslušní duplicitní pacienti a taktéž se prohodí. Tímto je zajištěno, že proces

rekombinace vrací „validní“ jedince. Druhá část způsobu rekombinace, který jsem použil, se často vyskytuje externě (mimo samotný proces rekombinace) jako proces opravy.



Obr. 5. – Proces rekombinace (část 1)



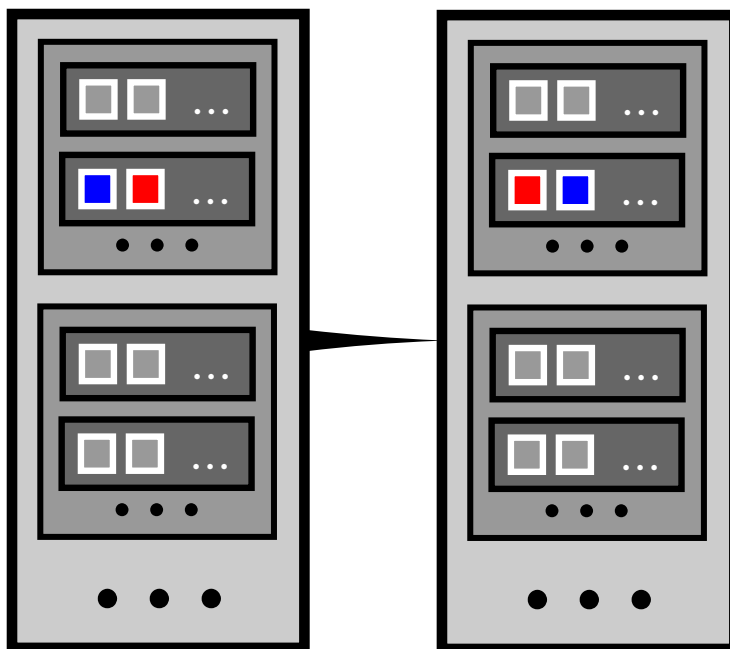
Obr. 6. – Proces rekombinace (část 2)

Původně jsem chtěl udělat rekombinační proces více ovlivňující genomy (v genetickém programování se například rekombinací vyměňují celé podstromy syntaktických stromů dvou programů), ale narazil jsem na problém, při kterém vznikaly jen velmi těžce odstranitelné duplicitní pacienti na různých místech. Oprava by potom byla podstatně složitější než samotný

proces rekombinace, a tedy jsem se rozhodl použít tuto variantu, která je méně efektivní, ale snadněji produkuje „validní“ jedince.

3.5. Mutace

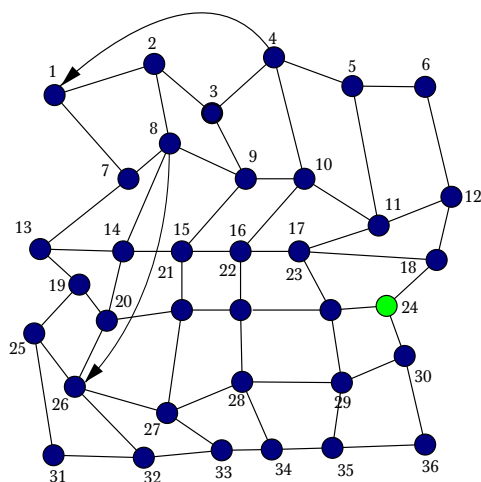
Proces mutace je relativně triviální. V jedinci se náhodně vyberou dva pacienti a prohodí se, jak je ilustrováno [obrázkem 7](#).



Obr. 7. – Proces mutace

4. Experimenty

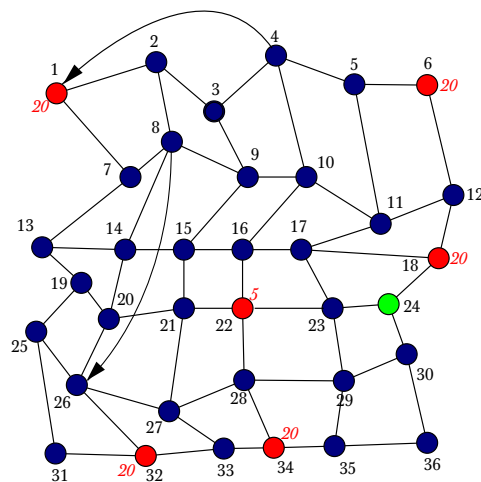
Pro ověření funkčnosti svého programu jsem navrhl dva příklady, oba nad jednou [mapou](#) a s následujícími parametry: 2 sanitky, 2 povolené jízdy, kapacita sanitky je 2, 6 pacientů.



Obr. 8. – Příklad mapy města

4.1. Příklad 1.

V příkladě 1 máme (podle **obrázku**) 5 pacientů s životností 20 a jednoho pacienta s životností 5. Chceme, aby GA upřednostnil pacienta s malou životností (nenechal ho zemřít).



Obr. 9. – Příklad 1. (1)

Zelený vrchol je v tomto případě nemocnice, červené vrcholy jsou pacienti, červená čísla jsou životnosti. Necháme program, aby problém zpracoval a po několika málo vteřinách dostáváme následující **výstup**:

```

0: 78.88198757763976
1: 77.39130434782608
2: 66.70807453416148
3: 64.22360248447205
4: 55.15527950310559
5: 54.285714285714285
6: 53.66459627329192
7: 41.73913043478261
8: 40.24844720496895
9: 39.75155279503105
MISSION 0:
RIDE 0:
PATIENT ID: 22 1.0.
PATIENT ID: -1 null.
RIDE 1:
PATIENT ID: 6 2.0.
PATIENT ID: 1 2.0.
MISSION 1:
RIDE 0:
PATIENT ID: 18 18.0.
PATIENT ID: -1 null.
RIDE 1:
PATIENT ID: 34 6.0.
PATIENT ID: 32 6.0.

```

N Fit:78.88198757763976

% PIE: 14.517604023776862

Ambulance 0:

24->23->22->(_) ->22->23->24->24->18->12->6->6->5->4->1->1->2->3->
4->10->11->17->18->24->

Ambulance 1:

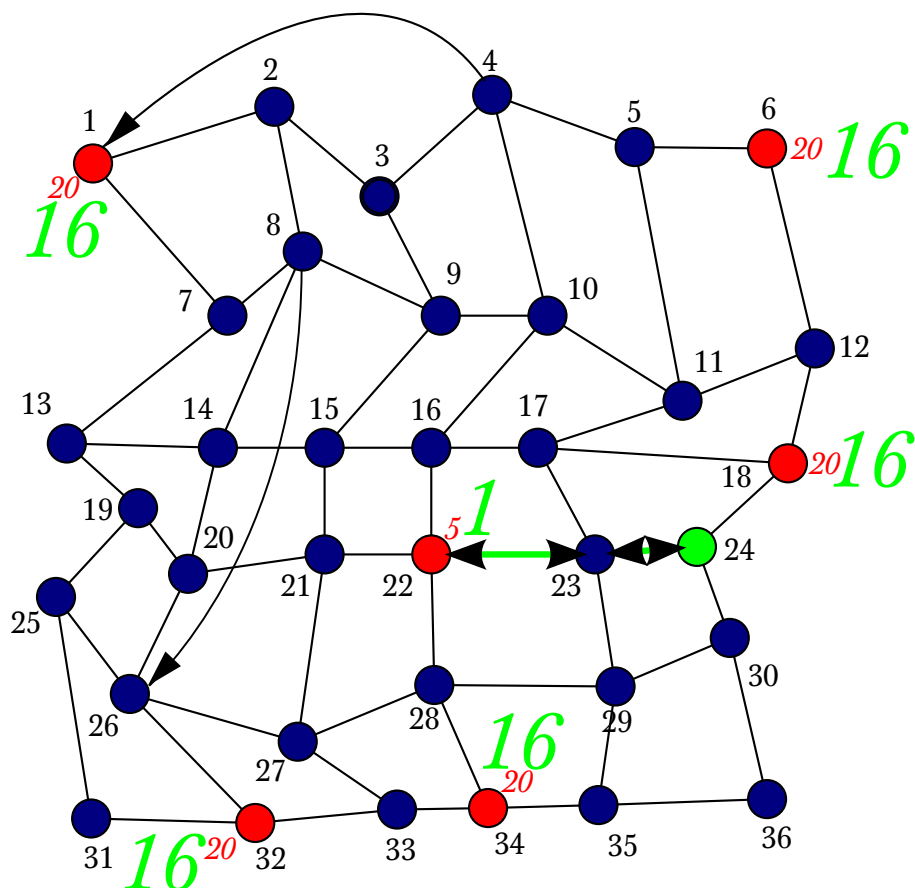
24->18->(_) ->18->24->24->23->29->28->34->34->33->32->32->33->34->
35->29->30->24->

Výpis 1. – Výstup programu, příklad 1

Jako první dostáváme pro informaci seznam fitness vyskytujících se v konečné populaci. Následuje výpis nejlepšího jedince, resp. nejprve obsazení sanitek pacienty a jejich konečné životnosti, následně oba dva druhy fitness a konečně cesta, kterou se mají obě sanitky paralelně vydat.

Řádek PATIENT ID: -1 null. znamená, že toto místo není obsazeno. Ve výpisu cesty sekvence ([podtržítka]) znamená, že algoritmus narazil na neobsazené místo, a tedy pokračuje dále. Pokud se ve výpisu cesty vyskytuje nějaký vrchol dvakrát, znamená to, že na tom místě byl vyzvednut pacient.

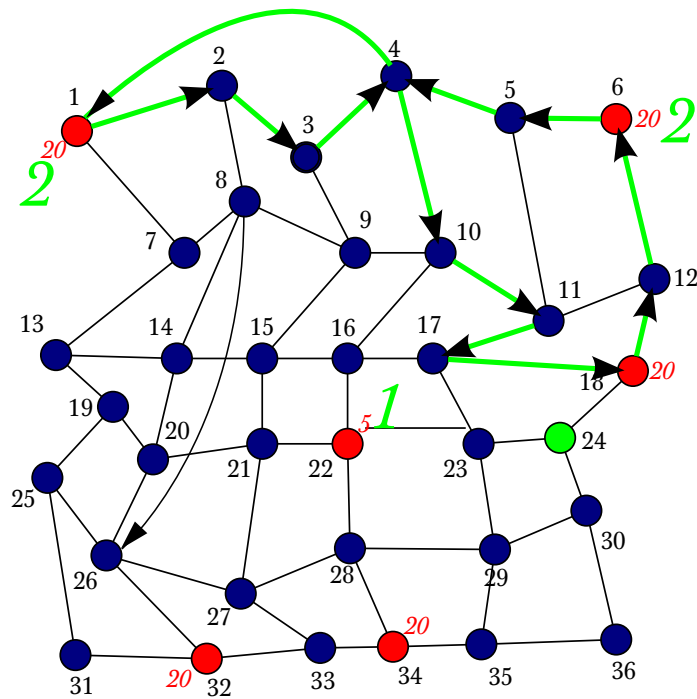
Představíme-li si tedy výslednou cestu sanitek, vypadá pro sanitku 2 přesně jako na následujícím obrázku.



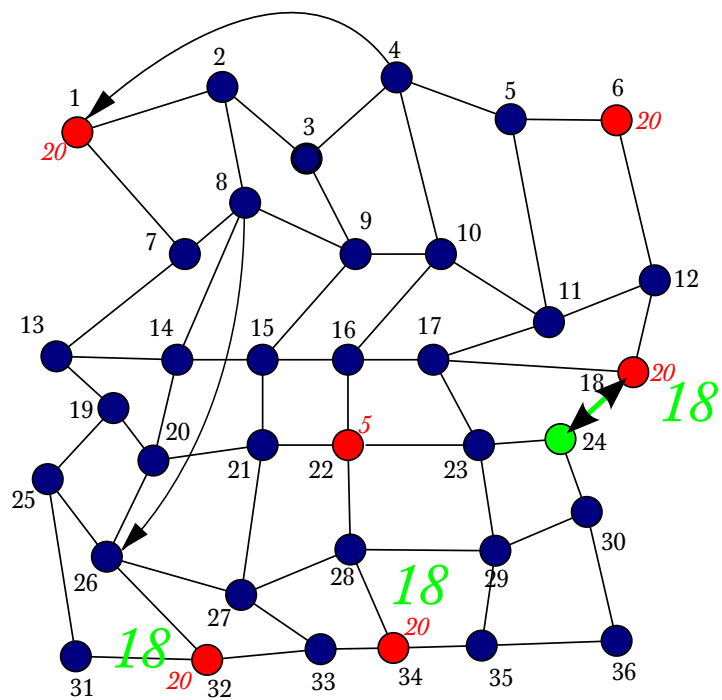
Obr. 10. – Příklad 1. (první sanitka 1)

Vidíme, že první sanitka jela nejprve pro nejkritičtějšího pacienta a okamžitě se s ním vrátila do nemocnice (tedy nebrala žádného dalšího, protože by mezitím zemřel). Pro úplnost uvedu

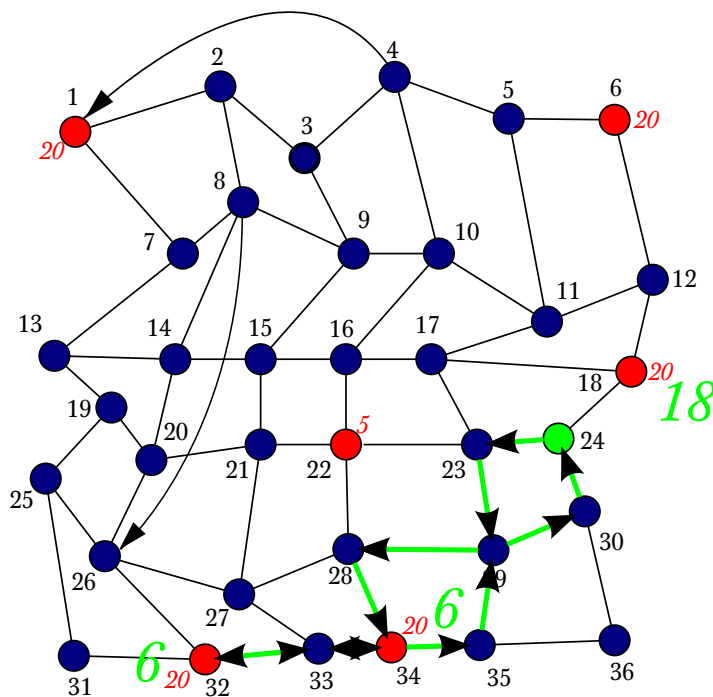
v této grafické formě zbytek cesty pro první sanitku a dále paralelní cestu druhé sanitky.



Obr. 11. – Příklad 1. (první sanitka 2)



Obr. 12. – Příklad 1. (druhá sanitka 1)



Obr. 12. – Příklad 1. (druhá sanitka 2)

4.2. Příklad 2.

V příkladě 2 nechť pacient, který měl v příkladě 1 životnost 5, má nyní životnost 100, a tedy si budeme přát, aby byl vzat až jako poslední (i když pokud by byl vzat jako první, ostatní pacienti by přežili také, ale s nižší průměrnou životností). Nebudu se nyní již zdržovat pracným vykreslováním cesty v editoru Inkscape, ale analyzuji rovnou výstup z programu.

```

0: 77.94285714285715
1: 68.11428571428571
2: 66.05714285714286
3: 65.82857142857142
4: 55.08571428571428
5: 54.400000000000006
6: 45.02857142857143
7: 44.57142857142857
8: 43.65714285714286
9: 42.285714285714285
10: 33.6
MISSION 0:
RIDE 0:
PATIENT ID: -1 null.
PATIENT ID: 18 13.0.
RIDE 1:
PATIENT ID: 34 1.0.
PATIENT ID: 32 1.0.
MISSION 1:
RIDE 0:
PATIENT ID: 6 1.0.
PATIENT ID: 1 1.0.
RIDE 1:
    
```

```

PATIENT ID: -1 null.
PATIENT ID: 22 82.0.

N Fit:77.94285714285715
% PIE: 14.114238410596025
Ambulance 0:
( )->24->18->18->24->24->23->29->28->34->34->33->32->32->33->34->
35->29->30->24->
Ambulance 1:
24->18->12->6->6->5->4->1->1->2->3->4->10->11->17->18->24->( )->
24->23->22->22->23->24->

```

Výpis 2. – Výstup programu, příklad 2

Je zřejmé, že program splnil náš požadavek výborně, protože druhá sanitka nejdříve vezme dva naléhavé případy najednou a potom jede již prázdná k pacientovi s velkou životností (který, mimo jiné, bydlí hned těsně vedle nemocnice).

5. Závěr

Když jsem si tuto úlohu vybíral, vybíral jsem si ji proto, že jsem si chtěl vyzkoušet napsat vlastní genetický algoritmus (tímto tématem se ve školních předmětech zabýváme jen velmi povrchně). Skoro v zápětí jsem ale pracoval na seminární práci z jiného předmětu, jejíž náplní bylo genetické programování v jazyce Scheme. Výsledkem byl jeden z nejsložitějších a nejzajímavějších projektů, jaký jsem kdy psal. Můj entusiasmus z této optimalizační úlohy tedy poněkud opadl, ale stále jsem se při jejím psaní velice bavil.

Když člověku, který jinak výpočetní techniku nemusí, vyprávíte o genetických algoritmech, pokaždé ho dokáže nadchnout představa, že program vytváří jiné programy, že programy replikuje a vyvíjí, že přejímá evoluční teorii. Zřejmě si pokaždé vybaví situace z populárních vědeckofantastických děl, kdy byla u strojů tato vlastnost dovedena k dokonalosti. Právem jsou tedy GA a GP jednou z nejzajímavějších oblastí computer science.

Při práci na této úloze jsem si mj. uvědomil, že optimální řešení je často zbytečný luxus a že hodně přijatelných řešení není optimální.

6. Použitá literatura

- CHAIYARATANA, N. Recent developments in evolutionary and genetic algorithms: theory and applications. Genetic Algorithms in Engineering Systems: Innovations and Applications [online]. 1997, 446, [cit. 2011-04-27]. ISSN 0537-9989.
- *Web katedry řídicí techniky* [online]. 2011 [cit. 2011-04-27]. Slidy k přednáškám předmětu Optimalisace v inteligentních systémech. Došupné z WWW: <<https://moodle.dce.fel.cvut.cz/course/view.php?id=28>>.
- *Web katedry kybernetiky* [online]. 2011 [cit. 2011-04-27]. Slidy k přednáškám předmětu Teoretická informatika. Došupné z WWW: <<http://cw.felk.cvut.cz/doku.php/courses/a7b33tin/start>>.
- *Web katedry kybernetiky* [online]. 2011 [cit. 2011-04-27]. Slidy k přednáškám předmětu Systémy s umělou inteligencí. Došupné z WWW: <<http://cw.felk.cvut.cz/doku.php/courses/a7b33sui/prednasky>>.