VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

FORMOVÁNÍ MULTIAGENTNÍCH KOALIC POMOCÍ GENETICKÝCH ALGORITMŮ

COALITION FORMATION IN MULTIAGENT SYSTEMS USING GENETIC ALGORITHMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. TOMÁŠ KUČERA**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Kučera Tomáš, Bc.**
Program: Informační technologie Obor: Inteligentní systémy
Název: **Formování multiagentních koalic pomocí genetických algoritmů**
Coalition Formation in Multiagent Systems Using Genetic Algorithms
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou formování multiagentních koalic za účelem souběžného dosahování co nejvíce cílů.
2. Nalezněte vhodné způsoby, jak tento problém řešit s použitím genetických algoritmů
3. Pro vhodný problém, například zadaný pro soutěž MASSIM 2018, realizujte navržené řešení a vyhodnoťte jej z hlediska rycholsti nalézání přijatelného rozložení koalic pro daný problém.
4. Integrujte vytvořený algoritmus do multiagentního prostředí JADE a napište manuál pro jeho použití.

Literatura:

- Wooldridge, M.: An Introduction to MultiAgent Systems, 2nd Edition, Willey, 2009
- Shoham, Y., Leyton-Brown, K.: Multiagent Systems, Algorithmic, Game-Theoretic, and Logical Foundations, Cambridge University Press, 2009

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Táto práca pojednáva o základoch softvérových agentov a ich formovaní do multiagentových koalícií. Predstavené sú genetické algoritmy ako jedno z možných riešení formovania koalícií. Je predstavená súťaž MAPC 2018, vzhľadom na ktorú bolo navrhnuté a implementované výsledné riešenie a taktiež nástroje, ktoré k tomu boli použité. Vytvorený bol demo projekt, v ktorom agenti komunikujú so serverom MASSim a získané dáta sú použité ako vstup genetickému algoritmu. Ten zadaných agentov priradí na základe prijatých dát k úlohám tak, aby mohli byť riešené čo najefektívnejšie. Výsledky algoritmu sú zhodnotené v experimentoch, ktoré sa zameriavajú na kvalitu nájdených riešení a čas potrebný na výpočet.

Abstract

This thesis discusses the basics of software agents and the way they form the multiagent coalitions. Genetic algorithms are introduced as one of the methods of solving the coalition formation problem. MAPC 2018 competition is introduced, which inspired the final design and implementation of the solution by using the tools described. A demo project was created, in which agents communicate with the MASSim server and gather data which is then used as an input into the genetic algorithm. Its purpose is to assign the agents to the tasks based on the input data, so that the tasks can be accomplished in the most effective manner possible. The results of this algorithm are evaluated in experiments which are focused on the quality of the solutions found as well as the time required for the calculation.

Kľúčové slová

Genetický algoritmus, Agent, Multiagentový systém, Koalícia, Formovanie koalícií, MAS, JADE

Keywords

Genetic algorithm, Agent, Multiagent system, Coalition, Coalition formation, MAS, JADE

Citácia

KUČERA, Tomáš. *Formování multiagentních koalic pomocí genetických algoritmů*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. František Zbořil, Ph.D.

Formování multiagentních koalic pomocí genetických algoritmů

Prehlásenie

Prehlasujem, že som túto prácu vypracoval samostatne pod vedením Doc. Ing. Františka Zbořila, Ph.D. Uviedol som všetky literárne zdroje, z ktorých som čerpal.

.....
Tomáš Kučera
21. mája 2019

Poďakovanie

Týmto ďakujem vedúcemu práce Doc. Ing. Františkovi Zbořilovi, Ph.D. za odborné vedenie a pomoc pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Multiagentové systémy sú považované za dôležité a rýchlo sa rozširujúce odvetvie výskumu umelej inteligencie. Je to hlavne vďaka ich možnosti širokospektrálneho aplikovania na rôzne problémy reálneho sveta [14]. Za cieľom efektívnejšieho riešenia určitých problémov sa agenti v systémoch formujú do koalícií, v ktorých spolupracujú s ostatnými členmi s vidinou zvyšovania svojho blaha, ako aj blaha koalície, resp. celého systému. Riešenie problému tvorby koalícií je výpočtovo náročné, preto bolo vyvinutých veľa optimalizačných metód na jeho riešenie.

Táto práca sa zaoberá aplikovaním konkrétneho stochastického prístupu z odvetvia soft computing-u na formovanie koalícií agentov - genetickým algoritmom, ako jedným z najznámejších prístupov založených na evolúcii. Tieto algoritmy sa zvyknú využívať na riešenie problémov s enormným stavovým priestorom, u ktorých je neprijateľné pri hľadaní riešenia prechádzať všetky existujúce možnosti. Ich charakteristické črty, ako napríklad selekcia najlepších, kríženie za účelom prenesenia kombinácie vlastností rodičov na potomka, a iné, sú istou abstrakciou nad princípmi evolúcie fungujúcimi v prírode. S využitím tejto metodológie je možné optimalizovať zložité problémy bez toho, aby bol známy exaktný postup pre výpočet riešenia. Implementovaný algoritmus bude aplikovaný na konkrétny problém koaličného formovania ako jedného z kľúčových faktorov efektívneho riešenia zadania súťaže MAPC 2018.

V prvej časti práce sú položené teoretické základy, na ktorých je práca založená. Definované sú používané pojmy a techniky, ktoré sa v oblastiach umelej inteligencie, konkrétne multiagentových systémov a genetických algoritmov, používajú pre riešenie problému formovania koalícií. Analyzovaná je tiež aj súťaž MAPC 2018 z pohľadu možných typov agentov, úloh a zariadení, ktoré sa v scenári Agents in the City môžu vyskytovať. Teória ďalej zasahuje do oblasti softvérových nástrojov, ktoré môžu byť využité pri návrhu a implementácii výsledného riešenia. Tretia kapitola práce hovorí o existujúcich riešeniach podobných problémov, analyzované sú ich postupy, z ktorých je možno brať inšpiráciu pri návrhu vlastného riešenia.

Text ďalej obsahuje návrh a implementáciu vlastného riešenia, ktoré zahŕňa demo aplikáciu a samotný genetický algoritmus. Demo aplikáciu predstavujú agenti, ktorí komunikujú so serverom súťaže MASSim, a získavajú tak dáta potrebné pre výpočet algoritmu. Posledná časť popisuje priebeh a výsledky experimentov s výsledným systémom, ktoré sú zhodnotené z hľadiska časovej náročnosti a kvality výsledných riešení.

Kapitola 2

Teoretická analýza

Táto kapitola definuje teoretický základ nevyhnutný pre pochopenie riešenej úlohy a taktiež implementáciu jej riešenia. Najprv je čitateľ oboznámený s pojmom agent a ich formovaním do multiagentových systémov (MAS). Ďalej sú predstavené samotné genetické algoritmy, kde sú definované často používané pojmy, ako aj rôzne typy genetických operátorov, či samotných genetických algoritmov. V posledných častiach je popísaná súťaž MAPC, s ohľadom na ktorú bude riešenie genetického algoritmu (GA) vytvorené a takisto aj nástroje, ktoré na to budú použité.

2.1 Agent

V obore informačných technológií je agent definovaný ako umelý agent, ktorý bol vytvorený na plnenie istej úlohy, je založený na architektúre a riadi ho program. Jeho vlastnosť autonómne konať za účelom dosiahnutia svojho cieľa ho diferencuje od iných prvkov [15].

Všeobecne sa agent definuje ako hardvér, alebo častejšie softvérový systém (prípadne ich spojenie), ktorý disponuje nasledujúcimi vlastnosťami [21]:

- autonómnosť - jednanie bez priameho zásahu iných objektov a vlastníctvo kontroly nad vlastnými akciami a stavom,
- sociálnosť - prebieha interakcia s ostatnými agentmi, resp. ľuďmi,
- reaktívnosť - vnímanie svojho okolia a reakcia na jeho zmeny,
- proaktivita - vykonávanie krokov vedúcich k splneniu definovaných cieľov.

Agent na základe interakcie s prostredím zbiera informácie, ktoré spracúva a môže ich predať ďalším agentom v jeho okolí.

2.2 Multiagentové systémy

Systém je všeobecne množina prvkov a vzťahov medzi nimi. Formálna definícia ho popisuje ako dvojicu $S = (U, R)$, kde univerzum U je konečná množina prvkov systému, z ktorých každý má svoje vstupy a výstupy, a R je relácia vstupov a výstupov (definuje mapovanie výstupov na vstupy).

Jedným z prvkov multiagentového systému je agent. Zvyšné prvky tejto množiny tvoria okolie agenta, v ktorom vykonáva svoju činnosť, má isté vstupy (napr. vnemy z okolia) a

výstupy (napr. vykonávaná činnosť jeho akčnými členmi). Prostredia agenta sa delia na rôzne skupiny [25, 24]:

- statické/dynamické - pokiaľ agent nevykonáva žiadnu činnosť, statické prostredie sa na rozdiel od dynamického nemení,
- plne/čiastočne pozorovateľné - agent vníma celé/časť prostredia,
- deterministické/nedeterministické - v deterministickom prostredí je nasledujúci stav daný aktuálnym stavom, resp. činnosťou vykonávanou agentom,
- epizódne/neepizódne - v epizódnom prostrední je časová línia behu systému rozdelená na nezávislé časti,
- strategické - v jednom prostredí koná viac agentov súčasne.

Pojem multiagentové systémy, ako aj pojem agent sú v literatúre definované rôzne. Nakoľko sa systémy líšia, používa sa veľmi generalizovaná definícia: Multiagentové systémy sú systémy, ktoré zahŕňajú viacero samostatných jednotiek disponujúcich rôznymi informáciami alebo záujmami [17].

Podľa jednej z konkrétnejších definícií je multiagentový systém charakterizovaný ako kolekcia určitého počtu agentov, medzi ktorými prebieha istý druh interakcie (komunikujú). Sú schopní konať v prostredí, v ktorom sa nachádzajú, a každý z týchto agentov dokáže ovplyvňovať určitú časť svojho okolia. Nakoľko sa tieto okolia môžu prekrývať, dá sa hovoriť o nejakých závislostiach medzi agentmi (napríklad situácia, kedy roboti dokážu prechádzať rovnakými dvermi, avšak nie naraz). Závislosti môžu byť aj iného charakteru. Napríklad jeden agent môže mať vyššie práva - byť nadradený oproti ostatným. Návrh a implementácia týchto systémov sa zaoberá dvoma hlavnými problémami. Prvým problémom je ako vytvoriť agentov, ktorí nezávisle a autonómne podnikajú kroky k splneniu definovaných cieľov. Druhým je myšlienka, ako vytvoriť agentov schopných interakcie s ostatnými agentmi za účelom plnenia cieľov, hlavne ak rozdielni agenti majú rôzne úlohy [20].

2.3 Koalície agentov

V mnohých úlohách musia agenti vytvoriť tímy, aby boli schopní riešiť určité úlohy, resp. boli schopní efektívnejšie vykonávať činnosti za pomoci spoločenských dohôd, pomocou ktorých tiež predchádzajú konfliktom a majú spoločné metódy kooperácie. Je to akási abstrakcia, ktorá reprezentuje mentálny stav danej skupiny [23].

V teórii hier sa tento proces vytvárania družstiev nazýva *formovanie koalícií* (*coalition formation*). Tradičnou myšlienkou je, že požadovaným výsledkom tohto procesu je *grand koalícia* (obsahujúca množinu všetkých agentov) alebo *koaličná štruktúra* (*coalition structure* - *CS*), ktorá pozostáva z disjunktných koalícií agentov. Väčšina teórie v danej oblasti sa zaoberá *prenosným úžitkom* (*transferable utility*), v ktorých neexistujú obmedzenia, akým spôsobom si agenti získaný úžitok môžu rozdeliť - napr. agenti z jednej koalície môžu zaplatiť agentom inej koalície. Tento prístup môže byť aplikovateľný v istých situáciách, no existujú aj také, kde je to neprípustné. Vo viacerých scenároch je možné docieliť najlepší možný výsledok, len ak môže jeden agent patriť do viacerých koalícií súčasne. V tom prípade agent musí rozdeliť svoje zdroje medzi viacero koalícií, ktorých je členom [7]. Príkladom tejto situácie z reálneho života je riešenie školského zadania - nie je výhodné riešiť zadanie v jednom veľkom tíme (grand koalícia) po celý čas. Je efektívne istý čas diskutovať

o úlohách s ďalšími študentmi a k zaručeniu originality práce tiež v rôznych fázach riešiť projekt samostatne.

2.3.1 Klasické koalície

Nech $N = \{1, ..., n\}$ je množina agentov. Podmnožina $C \subseteq N$ je nazývaná *koalícia*. *Koaličná štruktúra* v klasických koalíciách (non-overlapping environment) je rozdelenie danej množiny agentov na podmnožiny. V prípade platnosti prenosného úžitku môže byť proces formovania koalícií interpretovaný veľmi jednoducho. Tento predpoklad je založený na existencii komodity, ktorá môže byť bez obmedzení prenášaná medzi hráčmi. Úlohou *charakteristickej funkcie* v koaličných hrách s prenesiteľným úžitkom (*TU-hra*) je špecifikovanie jedného čísla charakterizujúceho hodnotu koalície, formálne zapísané $v : 2^N \to \mathbb{R}$, a teda $v(C)$ definuje hodnotu koalície C. TU-hra je teda špecifikovaná množinou hráčov N a charakteristickou funkciou v, zapísané $G = (N, v)$ [7].

Zatiaľ čo charakteristická funkcia definuje odmenu pre koalíciu, nie je jasné rozdelenie odmeny medzi jej členov. Toto je dané teóriou *imputácie* definovanej nasledovne. *Alokácia* je vektor odmien $x = (x_1, ..., x_n)$ dávajúci nejakú odmenu každému $j \in N$. Alokácia je *efektívna* vzhľadom na koaličnú štruktúru CS, ak $\forall C \in CS : \sum_{j \in C} x_j = v(C)$. Alokácia $(x_1, ..., x_n)$ je nazývaná imputáciou, ak je efektívna a spĺňa individuálnu racionalitu $x_j \geq v(\{j\})$ pre $j = 1, ..., n$. $I(CS)$ značí množinu všetkých imputácií pre štruktúru CS.

Zmenou stavu systému sa tiež mení aj hodnotenie prospešnosti členstva v koalícii pre agentov. Pre zabránenie jednoduchého opustenia koalície agentom, ktorého prospech členstva v koalícii klesne natoľko, že v nej nemá racionálny dôvod ostať, sú zavedené normy a penalizácie [23]. *Stabilita* koaličnej štruktúry je založená na tom, aby agenti nemali racionálnu potrebu opustiť koalíciu s cieľom získania vyššej odmeny. Preto je nutné, aby výsledky dosiahnuté celou koalíciou a rozdelenie odmien pre jednotlivých agentov spĺňali individuálnu a skupinovú racionalitu. Existujú rôzne definície stability. Jednou z najlepšie zdokumentovaných je *jadro*. V teórii TU-hier je jadro chápané ako množina takých dôsledkov $(CS, x), x \in I(CS)$, že žiadna podmnožina agentov nie je motivovaná opustiť svoju koalíciu v CS.

Definícia 2.1. So štruktúrou koalícií CS a alokáciou odmien pre agentov $x \in \mathbb{R}^n$ je jadro hry (N, v) množina všetkých dvojíc (CS, x), kde $x \in I(CS)$ a $\forall C \subseteq N$ takých, že $\sum_{j \in C} x_j \geq v(C)$.

Z definície vyplýva, že žiadna koalícia by nemala odmietnuť návrh na vytvorenie alokácie jadra. Nakoľko je ale teória jadra veľmi obmedzujúca, v mnoho hrách je jadro prázdne. Ak uvažujeme vlastnosť superaditivity charakteristickej funkcie ($v(U \cup T) \geq v(U) + v(T)$ pre všetky disjunktné koalície U a T), potom z predošlej definície môžme brať do úvahy len taký prípad, že CS je grand koalícia $\sum_{j \in N} x_j = v(N)$.

2.3.2 Čiastkové koalície

V *prekrývajúcich sa* (*čiastkových, overlapping*) koalíciách ide v princípe o to, že agent vlastní určité zdroje, ktoré môže distribuovať medzi koalície, do ktorých patrí (tvorenie koalícií sa v tomto prípade nazýva *overlapping coalition formation, OCFP*). Nakoľko teória čiastkových koalícií nie je natoľko rozvinutá ako u klasických, bude popísaná jedna zo známejších teórií týkajúcich sa tejto oblasti [7]. Tá používa normalizáciu daného princípu v zmysle, že agent vlastní jeden zdroj a jeho príspevok koalícii bude teda zlomkom tohto zdroja.

Na rozdiel od klasických koalícií, tu je koalícia definovaná vektorom $r = (r_1, ..., r_n)$, kde r_j je zlomok zdroja agenta j, ktorým prispieva do tejto koalície. *Podpora* (*support*) čiastkovej koalície $supp(r)$ je definovaná nasledovne: $supp(r) = \{j \in N | r_j \neq 0\}$.

Definícia 2.2. OCFP hra G s množinou hráčov $N = \{1, ..., n\}$ je daná funkciou $v : [0, 1]^n \rightarrow \mathbb{R}$, kde $v(0^n) = 0$.

Funkcia v priraďuje čiastkovej koalícii r prislúchajúcu odmenu. Táto hra je označená $G = (N, v)$. V teórii klasických koalícií môže byť teda ekvivalentne koalícia S reprezentovaná ako vektor e^S, kde ak $j \in S$, tak $(e^S)_j = 1$, inak 0.

Definícia 2.3. Pre množinu agentov $T \subseteq N$, štruktúrou koalícií na T je konečný zoznam vektorov (čiastkových koalícií) $CS_T = (r^1, ..., r^k)$, pre ktorý platí:

- $r^i \in [0, 1]^n$,
- $supp(r^i) \subseteq T$ pre všetky $i = 1, ..., k$,
- $\sum_{i=1}^{k} r_j^i \leq 1$ pre všetky $j \in T$.

V predošlej definícii každé $r^i = (r_1^i, r_2^i, ..., r_n^i)$ odpovedá nejakej čiastkovej koalícii (r_j je časť zdrojov agenta j poskytovaných r^i). Je dôležité si všimnúť, že koaličná štruktúra je v tomto prípade zoznam, nie množina, a teda sa v nej môže nachádzať viac ako jedna rovnaká koalícia.

Na rozdiel od klasických koalícií, kde je počet rôznych štruktúr koalícií konečný, v danej definícií OCFP sa nachádza nekonečné množstvo koalícií, a teda aj nekonečné množstvo koaličných štruktúr. To znamená, že napriek tomu, že v modele klasického CF je hľadanie štruktúry maximalizujúcej celkové blaho možné nájsť porovnaním všetkých možných kombinácií (nie však prípustné vzhľadom na veľké množstvo kombinácií), v modele OCFP je to nereálne.

Charakteristická funkcia bude v tomto modele definovaná ako $v(CS) = \sum_{r \in CS} v(r)$. Rozsiahlejšia teória ohľadom agentových koalícií sa nachádza v zdroji [7], z ktorého bola tiež čerpaná väčšina informácií obsiahnutých v tejto sekcii.

2.3.3 Formovanie koalícií

Problém formovania optimálnej koaličnej štruktúry (CFP - Coalition Formation Problem), ktorým sa zaoberá jadro tejto práce je proces zahŕňajúci vytvorenie koherentných skupín zložených z odlišných, autonómnych agentov za účelom dosahovania individuálnych, resp. kolektívnych cieľov čo najefektívnejšie. Tento problém sa v posledných rokoch stal obrovskou výzvou oboru multiagentových systémov. Hlavnou myšlienkou problému je zistenie, koľko koalícií je potrebných vytvoriť pre splnenie definovaných cieľov. Preto je potrebné pre každú koalíciu vypočítať jej hodnotu indikujúcu prínos, ktorý by priniesla, ak by bola vytvorená. Akonáhle sú tieto hodnoty zistené, agenti zvyčajne musia nájsť kombináciu koalícií, kde každý agent patrí práve do jednej koalície (na rozdiel od problému OCFP), a zároveň je maximalizovaný celkový zisk systému [14].

Celkový počet kombinácií rastie exponenciálne s počtom agentov, a preto boli na riešenie použité rôzne optimalizované algoritmy: dynamické programovanie, celočíselné programovanie, stochastické prehľadávanie (medzi ktoré patrí aj genetický algoritmus), a i. Množina všetkých klasických koalícií (2.3.1) ktoré je možno vytvoriť z danej množiny agentov je 2^n, kde n je celkový počet agentov. Z toho plynie, že prehľadávanie celého stavového priestoru

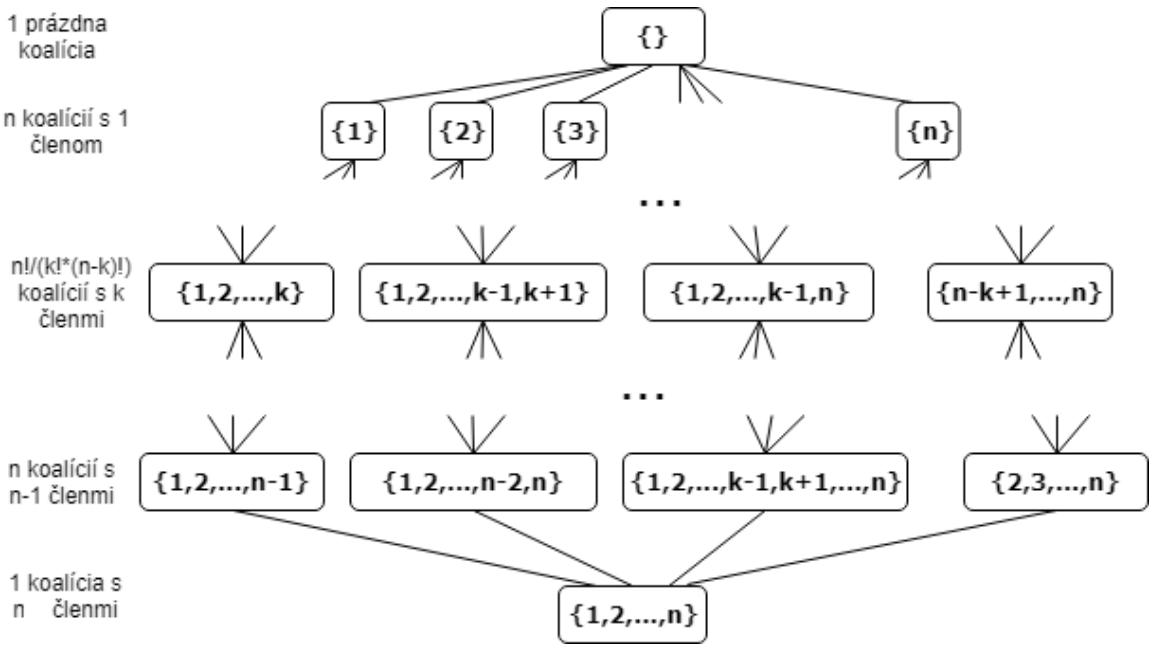
za účelom nájdenia vhodného rozdelenia agentov do koalícií nie je ideálne kvôli časovej náročnosti. Obrázok 2.1 znázorňuje rôzne koalície, ktoré je možné vytvoriť z n agentov. Všeobecný matematický výraz pre počet koalícií o k prvkoch z n agentov je $\binom{n}{k}$.

Všeobecne môže byť proces formovania koalícií rozdelený do troch krokov [14]:

1. Výpočet hodnoty koalície. Koaličná hodnota indikuje očakávaný zisk, ktorý je možné vyťažiť, ak by bola daná koalícia vytvorená. Hodnoty sú počítané vzhľadom na riešený problém. Ak sú vypočítané hodnoty všetkých koalícií, je možné rozhodnúť, ktoré koalície majú byť reálne vytvorené.

2. Generovanie štruktúry. Po získaní hodnôt koalícií je súčasťou problému CFP rozdelenie agentov do koalícií s cieľom čo najvyššieho verejného blaha. Nakoľko sa tento problém týka zvyšovania verejného blaha, nie je vylúčené, že daný princíp môže byť aplikovateľný aj na sebeckých agentov, ktorých prioritou je ich vlastný zisk. Návrh hlavne musí obsahovať posilňovací mechanizmus pre agentov, ktorý zvýši ich túžbu pridať sa do optimálnej štruktúry koalícií, čomu predchádza samotné zistenie, ako táto štruktúra vlastne vyzerá.

3. Rozdelenie odmeny. Po zistení, aké koalície majú byť vytvorené je nutné rozdelenie odmien pre agentov kvôli zaisteniu stabilnej koalície (agenti nemajú zámer opustiť koalície, v ktorých sa nachádzajú).



Obr. 2.1: Ilustrácia možných koalícií pre počet n agentov.

2.4 Genetické algoritmy

Evolučné algoritmy zahŕňajú metódy stochastického riešenia, ktoré za zvyknú využívať pri hľadaní riešení NP-úplných optimalizačných problémov. Základom týchto algoritmov je

prostredie, ktoré môže obsahovať len limitovaný počet indivíduí. Ich inštinktom je rozmnožovanie, a teda je nutné definovať určitý mechanizmus selekcie (výberu rodičov), aby veľkosť populácie nerástla exponenciálne. Selekcia uprednostňuje tých, ktorí sú schopní adaptovať sa k danému prostrediu najlepšie - silnejší prežije[1].

Genetické algoritmy (GA) sú optimalizačnou a prehľadávacou technikou využívajúcou princípy genetiky a prirodzeného výberu, a patria do skupiny najznámejších podtried evolučných algoritmov. Sú postavené na evolučných princípoch ako dedičnosť, mutácia, selekcia, kríženie a princíp silnejšieho. GA dovoľujú populácii indivíduí vyvíjať sa na základe rôznych princípov so spoločným cieľom - vylepšiť aktuálne riešenie [9]. Zvyšok kapitoly s pojmami a popisom metód sa opiera o zdroje [8, 13, 18, 9].

2.4.1 Terminológia

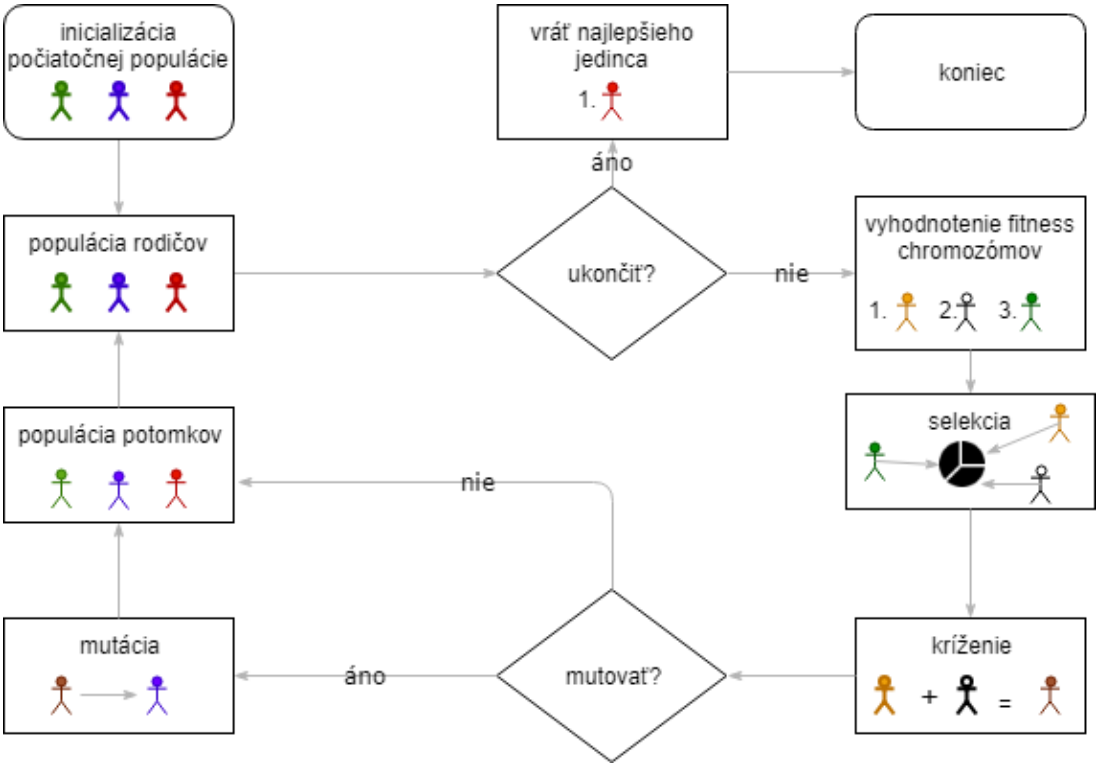
Táto sekcia slúži na definovanie základných pojmov, ktoré budú používané pri definovaní princípov a neskôr pri návrhu vlastného riešenia genetického algoritmu. Základnou jednotkou algoritmu je *chromozóm* (*jedinec*), ktorý je kódovaný vhodným spôsobom vzhľadom k definícii riešenej problematiky a reprezentuje jedno z možných riešení problému. Kód jedinca obsahuje atribúty, ktoré ho popisujú, a definujú tak jedno z riešení. Tieto atribúty sa nazývajú *gény* a hodnoty, ktoré môžu nadobúdať sú *alely*. Z chromozómu je nutné istým spôsobom získať informácie, čo reprezentuje (*fenotyp*) - napr. to môže byť rozdelenie služieb zdravotných sestier alebo rozsadenie hostí na oslave na základe ich podobných záujmov. Preto je nevyhnutná existencia metódy kódovania, alebo naopak interpretácie riešenia na základe chromozómu.

To, aké kvalitné je dané riešenie určuje *fitness* funkcia. Ovplyvňuje to, koľko genetického materiálu daného jedinca bude predané ďalšej generácii ako aj to, ako dlho bude daný jedinec existovať. Hodnota fitness funkcie pre kvalitného jedinca je obvykle vyššia ako u jedinca s horšími vlastnosťami. V prípade inverznej funkcie však platí presný opak, a preto sa tieto fakty uvažujú pri samotnom procese selekcie popísanej nižšie (druh fitness funkcie je zohľadnený vzhľadom na riešený problém).

2.4.2 Metodológia

Celý proces behu algoritmu je možné vidieť na obrázku 2.2. Prvým krokom je vytvorenie počiatočnej populácie - to môže byť náhodné alebo pomocou istej heuristiky, ktorá má zaručiť vyššiu fitness hodnotu už od začiatku. Algoritmus je zastavený, ak je splnená ukončovacia podmienka. Tá môže byť napríklad vo forme maximálneho počtu iterácií alebo dosiahnutia požadovanej hodnoty fitness u najkvalitnejšieho jedinca. Ak podmienka nie je splnená, vykoná sa vyhodnotenie fitness hodnoty chromozómov, na základe ktorej potom prebieha samotná selekcia. Po výbere vhodných rodičov nasleduje vytvorenie potomkov v procese kríženia. Potom sú s istou pravdepodobnosťou vybraní jedinci a gény, ktoré majú byť mutované. Ak je aplikovaný princíp *elitizmu* (najlepší jedinci sa dožívajú viac generácií), sú potomkovia pridaní k existujúcim elitám, a tým vzniká nová generácia rodičov, inak je celá nová generácia tvorená vzniknutými potomkami z aktuálneho cyklu.

[1] To však neznamená, že najsilnejší jedinec prežije vždy, nakoľko sú princípy evolučných algoritmov často založené na istej miere náhodnosti.



Obr. 2.2: Všeobecný priebeh genetického algoritmu.

2.4.3 Genetické operátory

Základnými genetickými operátormi sú kríženie, mutácia a selekcia. Každý z nich sa môže vyskytovať v rôznych variantoch adekvátnych k riešenej problematike.

Selekcia

Úlohou selekcie je výber jedincov, ktorí v procese kríženia vytvoria potomka. Väčšiu pravdepodobnosť výberu majú jedinci s vyšším fitness ohodnotením preto, aby bola časť ich génov predaná ďalšej generácii. Tento fakt však ale môže spôsobiť konvergenciu k lokálnemu optimu. V praxi je nutné *selekčným tlakom* ovplyvňovať pomer medzi lepšie a horšie ohodnotenými jedincami v procese selekcie, aby bola zaručená rôznorodosť riešení, a tým aj väčšia šanca nájdenia celkovo optimálneho riešenia. Jedným z druhov selekcie je princíp rulety, kde šanca na výber jedinca pre kríženie je proporcionálna k jeho hodnote fitness. To znázorňuje rovnica 2.1, v ktorej p_i je pravdepodobnosť výberu jedinca i a n je počet jedincov v populácii.

$$p_i = \frac{fitness_i}{\sum_{j=1}^{n} fitness_j} \tag{2.1}$$

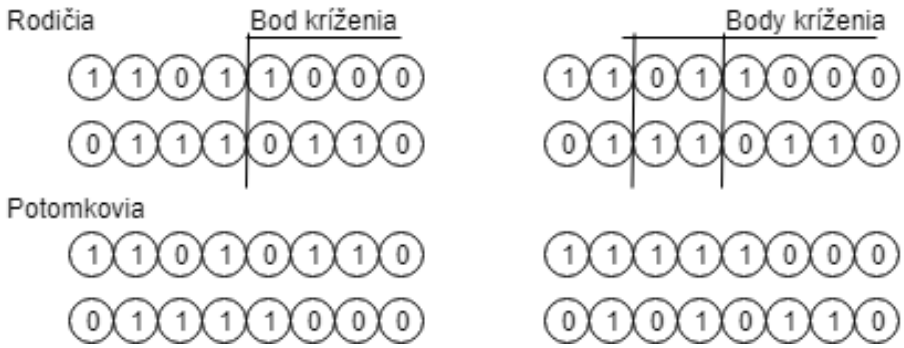
Ďalším spôsobom je výber na základe poradia - jedinci sú zoradení na základe fitness hodnoty a selekcia potom závisí na priradenom poradovom čísle. Výhodou tohto princípu na rozdiel od rulety je ten, že sú zanedbané rozdiely fitness medzi jednotlivými jedincami - uvažuje sa len o ich poradí, a teda najlepší jedinci neovládnu rýchlo celú populáciu, čím

je zaistená rôznorodosť riešení. Princíp turnaja je založený na výbere n jedincov, ktorých fitness hodnoty sú porovnané a víťaz postupuje ku kríženiu.

Kríženie

Kríženie (crossover) je proces vytvorenia nového chromozómu z tých existujúcich - vytvorenie nového jedinca, ktorý kombinuje pozitívne vlastnosti oboch rodičov. Nakoľko sa presne nevie, ktoré z vlastností sú kvalitné, je tento proces do istej miery náhodný. Úspešnosť vytvorenia nového, lepšieho riešenia závisí na reprezentácii problému, kódovaní, fitness funkcii, a ďalších parametroch algoritmu. Nakoľko nie je celkom známe prepojenie týchto vplyvov, neexistuje presný návod, ktoré z nich zvoliť, a teda sú často volené experimentálne.

Sú rôzne druhy kódovania jedinca, od ktorých sa ďalej odvíjajú spôsoby kríženia. Pri bitovom kódovaní je chromozóm zložený z génov, ktoré nadobúdajú binárne hodnoty (jedinec je teda reprezentovaný binárnym vektorom). Pri jednobodovom krížení sú 2 rodičovské chromozómy rozdelené v jednom bode na rovnakej pozícii a dvaja výslední potomkovia, ktorí môžu vzniknúť majú 1 časť od prvého rodiča a druhú časť od druhého. Na podobnom princípe je založené dvojbodové kódovanie, ktoré vektory rodičov rozdeľuje v 2 miestach. Oba tieto spôsoby sú naznačené v obrázku 2.3. Ďalším z mnoha spôsobov kríženia je uniformné (rovnomerné) kríženie - prechádzajú sa jednotlivé gény rodičov a s rovnakou pravdepodobnosťou (prípadne určenou parametrom) je vybraný výsledný gén od jedného alebo druhého rodiča.



Obr. 2.3: Príklad jedno a dvojbodového kríženia pri binárnom kódovaní.

Jedným z typov kódovaní je permutačná reprezentácia. Jej cieľom je zabránenie tomu, aby sa rovnaký gén objavil v chromozóme viac ako raz. Ak by taká situácia nastala, viedlo by to k nevalidnému riešeniu. Tento prístup sa využíva napríklad pri známom probléme obchodného cestujúceho, ktorý chce prejsť predom definované mestá tak, aby prešiel čo najkratšiu cestu. Chce sa však vyhnúť prípadu, kedy by jedno mesto navštívil viackrát. Jedným z možných druhov krížení pri tomto probléme je partially mapped crossover [8].

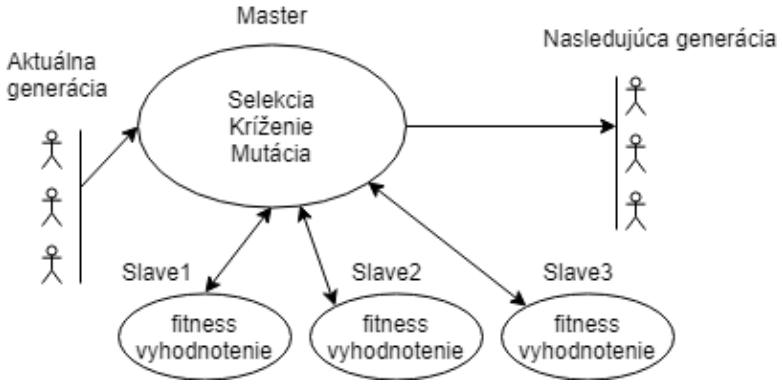
Mutácia

Na rozdiel od kríženia, mutácia je unárny operátor, ktorý náhodným spôsobom mení hodnotu génu. Tento prístup dovoľuje zachovať rôznorodosť populácie, a tým narušiť konvergenciu k lokálnemu suboptimu. Parameter mutácie, ktorý určuje pravdepodobnosť zmeny génu je dôležité nastaviť správne, nakoľko nízka hodnota môže viesť k príliš skorej lokálnej konvergencii, a naopak vysoká hodnota môže spôsobiť stratu dobrých riešení.

Druhy mutácie takisto, ako kríženie závisia na spôsobe kódovania jedincov. Pri binárnom chromozóme sa vyberie náhodný gén reprezentovaný binárnou hodnotou, ktorá je následne invertovaná.

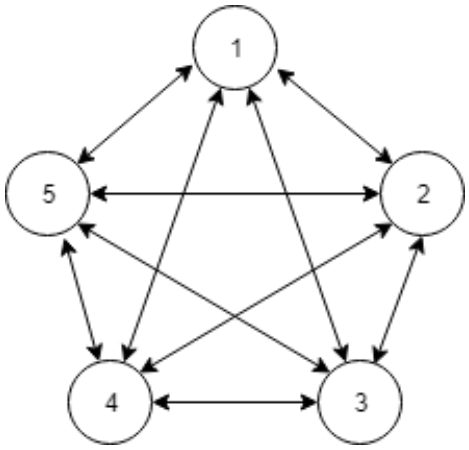
2.4.4 Paralelné GA

Paralelizmus je v GA, ako aj v ostatných odvetviach využívaný na minimalizáciu výpočtového času, resp. získanie lepšieho riešenia v rovnakom čase. Paralelné GA sú komplexné nelineárne algoritmy, ktorých výpočtová sila je ovplyvňovaná veľkým počtom parametrov, ako napríklad počet populácií - pri viacerých populáciách sa ďalej rieši otázka či budú izolované, alebo budú medzi sebou interagovať v podobe výmeny jedincov. Takáto interakcia ale prináša ďalšie komplikácie v podobe spomalenia výpočtu z dôvodu komunikácie jednotlivých výpočtových jednotiek, určenia topológie, určenia počtu vymieňaných indivíduí a frekvencie komunikácie. Podľa topológie a typu komunikácie sa delia na rôzne druhy [6].



Obr. 2.4: Príklad topológie paralelného GA master-slave.

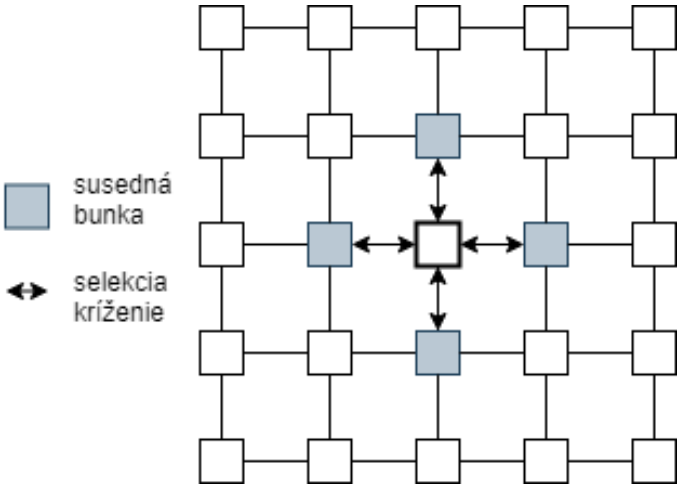
Master-slave s jednou populáciou ponecháva vyhodnocovanie fitness funkcie na slave procesoroch, master sa stará len o vykonanie vyššie spomenutých genetických operátorov. Je to najjednoduchšie riešenie paralelného výpočtu a prináša veľkú nevýhodu - častým prípadom je, že program čaká na výsledky od slave uzlov, a algoritmus je samozrejme tak rýchly, ako jeho najpomalšia časť.



Obr. 2.5: Cartwheel topológia multipopulačného paralelného GA.

Druhým príkladom je populácia rozčlenená na subpopulácie (ostrovy) distribuované medzi procesory, kde prebieha reprodukcia a výpočet fitness hodnoty nezávisle na zvyšku populácie. Ostrovy medzi sebou aplikujú proces výmeny indivíduí medzi sebou, čím zvyšujú rôznorodosť riešenia v danej oblasti - *migrácia*.

Topológia buniek (jemnozrnné paralelné GA) využíva jednu priestorovo rozdelenú populáciu, kde každý z procesorov uchováva len pár jedincov. Jedinci môžu interagovať len do určitej vzdialenosti v tejto topológii (pár susedných buniek), dôsledkom čoho je nutnosť prispôsobenia genetických operátorov.



Obr. 2.6: Topológia 2D jemnozrnného paralelného GA.

2.5 MAPC

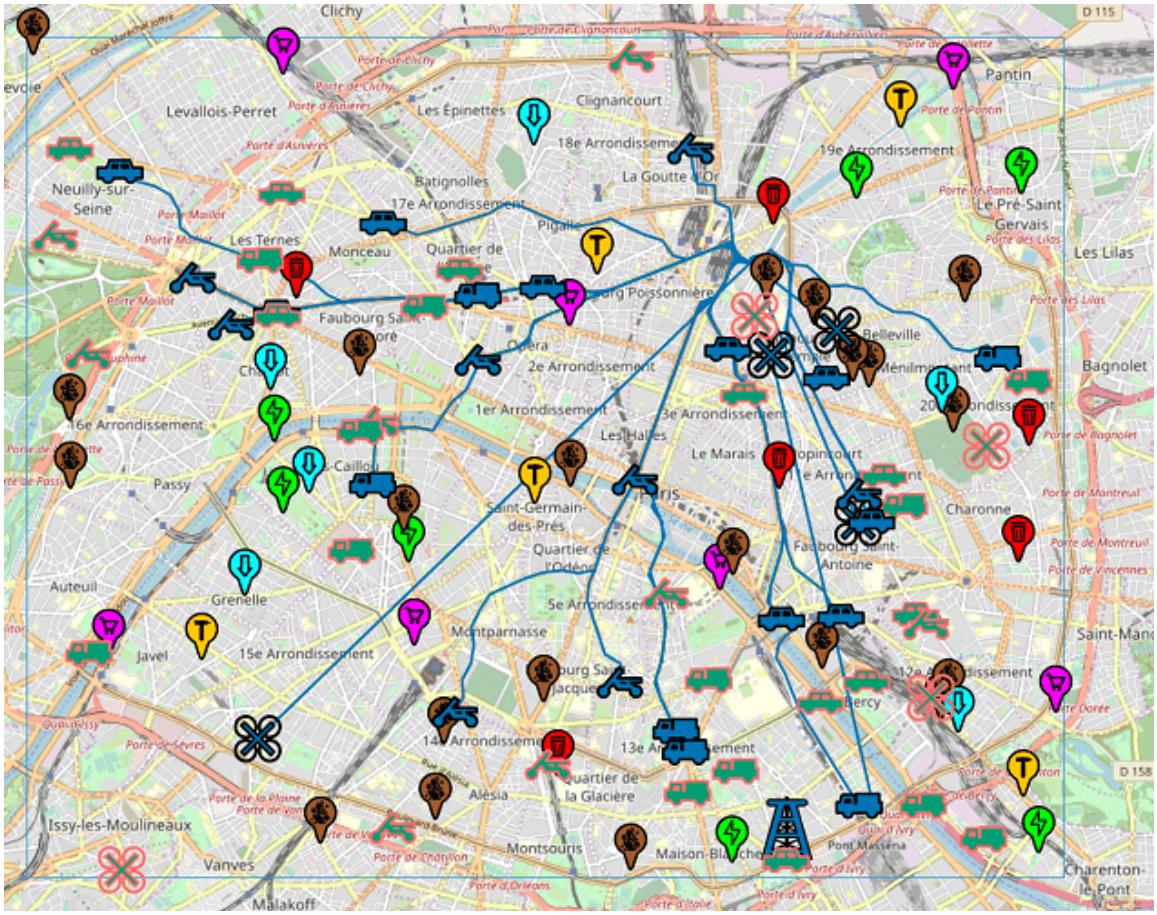
Multi-Agent Programming Contest je súťaž, ktorej zámerom je rozšíriť oblasť výskumu multiagentových systémov. Hlavnými prostriedkami pre splnenie tohto cieľa by malo byť identifikovanie kľúčových problémov, zbieranie vhodných benchmarkov a testovacích prípadov. Súťažiaci sú vedení k tomu, aby pomáhali odhaliť chyby existujúcich systémov a pracovali na ich vylepšení. Výkon systému je vyhodnotený sériou hier, kde dané systémy súťažia proti sebe [2].

Súťaž sa koná každoročne a téma sa zvykne meniť každých pár rokov. Posledné tri roky to bolo *Agents in the City*, pred tým *Agents on Mars* či *Cow Herding Scenario* (viac o minuloročných súťažiach je možné zistiť na stránke súťaže[2]).

Variant *Agents in the City (2018)* pozostáva z 2 a viac tímov agentov pohybujúcich sa v uliciach realistického mesta. Cieľom pre družstvá je postaviť čo najväčšie množstvo studní, čím tím získava body do celkového skóre hry. Hlavnou menou hry je *massium*, ktoré sa využíva na stavbu studní a je možné ho získať splnením určitých úloh alebo výmenou za poskladané predmety. Agent má zopár preddefinovaných vlastností, ktoré sú závislé na jednotlivých rolách priradených agentom. Mapa mesta je prevzatá z dát *OpenStreetMap* a smer cesty riadi *MASSim* server (2.6.1). V každom simulačnom kroku diskrétnej simulácie sa agent môže pohnúť o určitú pevne definovanú vzdialenosť. Podrobnejšie pravidlá je možno nájsť v zdroji [3]. Server tiež disponuje aj grafickým rozhraním, pomocou ktorého je možno

[2]dostupné online: https://multiagentcontest.org/

sledovať mapu s jej elementami a činnosti jednotlivých agentov, čo je možno vidieť na obrázku 2.7.



Obr. 2.7: Grafické rozhranie serveru MASSim s témou Agents in the City.

2.5.1 Agenti

Každý agent má priradenú určitú rolu, a každý tím obsahuje konzistentný počet agentov majúcich rovnakú rolu. Niektoré vlastnosti spojené s rolami majú definovanú základnú hodnotu, ktorú je však možné vylepšovať za istý poplatok, ale len do určitej maximálnej hodnoty: rýchlosť, kapacita, batéria, videnie a zručnosť. Posledná, špeciálna vlastnosť definuje pohyb agenta (pre drony je to vzduch - pohybuje sa vzdušnou čiarou, ostatní sa pohybujú len po cestách).

V čas každého simulačného kroku agenti odosielajú na server akciu, ktorú si želajú vykonať. V jednom kroku je možné spraviť len jednu akciu (ktorá môže viesť ku konfliktu - 2 agenti nakupujú rovnakú vec v ten istý čas a k dispozícii je len 1 inštancia, a teda vo výsledku agent, ktorého akcia bola vykonaná skôr dostane predmet, druhého akcia zlyháva kvôli neexistencii predmetu v danom obchode). Akcie očakávajú rôzne parametre viažuce sa na typ akcie [3].

2.5.2 Úlohy

Všeobecne je úloha definovaná začiatkom, koncom, odmenou za jej vykonanie a úložiskom, do ktorého má byť doručený daný predmet pre jej splnenie. Úlohy sú generované náhodne v priebehu simulácie a typicky môžu byť rôzneho druhu. Klasická úloha má preddefinovanú odmenu patriacu družstvu, ktoré úlohu dokončí ako prvý, ostatní na ňu nemajú nárok. Princípom aukčných úloh je ponúkanie množstva massia, za ktoré sú ochotní danú úlohu riešiť. Družstvo, ktoré si vypýtalo najnižšiu odmenu vyhráva a úlohu rieši tak, že ho nikto v riešení danej úlohy nemôže predbehnúť ako v predošlom prípade. Posledným typom sú misie priraďované náhodne, kde všetky tímy dostanú inštanciu rovnakej misie, ktorú musia splniť, inak sú sankcionovaní.

2.5.3 Zariadenia

Zariadenia sú rozmiestnené náhodne na mape a každé z nich má unikátne meno a lokalitu.

- **Obchody** sú miesta, kde je možné poskladané predmety vymieňať za predom nedefinovanú cenu. Tiež sa tu kupujú vylepšenia pre agentov.
- V **nabíjacej stanici** je možné nabiť batériu agenta, ktorá obsahuje energiu dôležitú pre pohyb. Sú definované množstvom energie, ktoré sú schopné dobiť v jednom simulačnom kroku.
- V **dielňach** agenti môžu skladať predmety z iných predmetov - pre zloženie predmetu je tiež nutné, aby pritom boli prítomné predom definované role agentov.
- Predmety môžu byť zničené v **skládkach** (pre zvýšenie aktuálnej nosnej kapacity agenta).
- Do **skladov** sú nosené predmety pre uskladnenie a taktiež sú cieľovou destináciou pre doručenie predmetu v rámci dokončenia úlohy.
- **Uzol surovín** je jediné miesto, kde môžu byť získané predmety najnižšej úrovne (predmety, z ktorých je možno vytvárať iné predmety). Sú skryté a musia byť nájdené agentmi počas behu simulácie. Každý z nich ponúka jeden druh suroviny, ktorej rýchlosť ťaženia spočíva v agentovej zručnosti (ak je o dosť vyššia ako požadovaná, je možné získať aj viac surovín v jednom kroku).
- **Studne** sú jediným zariadením, ktorý nie je generovaný serverom ale buduje ich agent. Majú definovanú cenu, za ktorú sú postavené, efektívnosť určujúcu koľko bodov vygenerujú za simulačný krok a integritu hovoriacu o výdrži. Tímy sa snažia zničiť súperove studne, ktoré však, podobne ako uzly zásob, nie sú obsiahnuté vo všeobecnej znalosti, a teda je nutné ich nájsť.

2.5.4 Vnemy

Vnemy pre agentov sú odosielané serverom a obsahujú informácie o aktuálnej simulácii. Existujú 2 druhy vnemov - počiatočný a krokový. Počiatočný je odoslaný na začiatku simulácie a obsahuje statické informácie, ktoré sa v čase nemenia - počet krokov simulácie, súradnice hraníc mapy, názov družstva agenta, kapacita predmetov a pod.

Krokový vnem nesie informácie o aktuálnom priebehu simulácie na začiatku každého simulačného kroku - aktuálny stav vlastností agenta, jeho pozícia, úspech poslednej akcie, úlohy, objavené uzly so zdrojmi atď.

2.6 Nástroje

Nasledujúce časti popisujú nástroje, ktoré budú použité pri implementácii finálneho riešenia. Je popísaný server MASSim ako hlavná stavebná jednotka reprezentujúca prostredie pre beh agentov. Ďalej sa pojednáva o spôsobe komunikácie so serverom a tiež nástroji JADE, do ktorého má byť výsledok tejto práce integrovaný.

2.6.1 MASSim

Multi-Agent Systems Simulation Platform je simulačný server napísaný v jazyku Java. Používaný je v súťaži Multi-Agent Programming Contest opísanej v kapitole 2.5, kde naprogramovaní agenti jednotlivými tímami súťažiacich súperia v preddefinovanej hre. Simulácia beží v diskrétnych krokoch po tom, ako sú agenti pripojení na server, s ktorým komunikujú prijímaním vnemov a odosielaním akcií určených k vykonaniu týmto serverom. Tento softvér je licencovaný pod AGPLv3+ [3].

2.6.2 EISMASSim

Mapovanie komunikácie medzi MASSim serverom a samotnými agentmi (preposielanie XML-správ) sprostredkuje EISMASSim, ktorý dané správy mapuje na volanie Java metód. Po vytvorení inštancie serveru je nutné ho spustiť, zaregistrovať agentov, ktorí sa zúčastnia komunikácie, namapovať entity (hmotné indivíduá vnímajúce okolie a vykonávajúce potrebné činnosti, ktoré sú špecifikované v špeciálnom súbore) na samotných implementovaných a spustených agentov, následne je možné prijímať a odosielať správy. Celý proces je založený na Enviroment Interface Standard (EIS), ktorý je štandardom pre interakciu v agentovom prostredí [1].

2.6.3 EIS

EIS je nástroj, ktorý slúži ako medzikód uľahčujúci prácu pri spojení agentovej platformy s prostredím, v ktorom má existovať. Každý agent, ktorý podporuje toto rozhranie je schopný sa pripojiť do prostredia implementujúceho dané rozhranie. Výhoda takého prístupu spočíva teda v tom, že kód, ktorý sa na strane agentovej platformy pripája k prostrediu je potrebné napísať raz (to platí aj o kóde na strane prostredia) a akonáhle ho platforma podporuje, agent je schopný sa pripojiť a komunikovať. Spadá pod licenciu GNU GPLv3. [10]

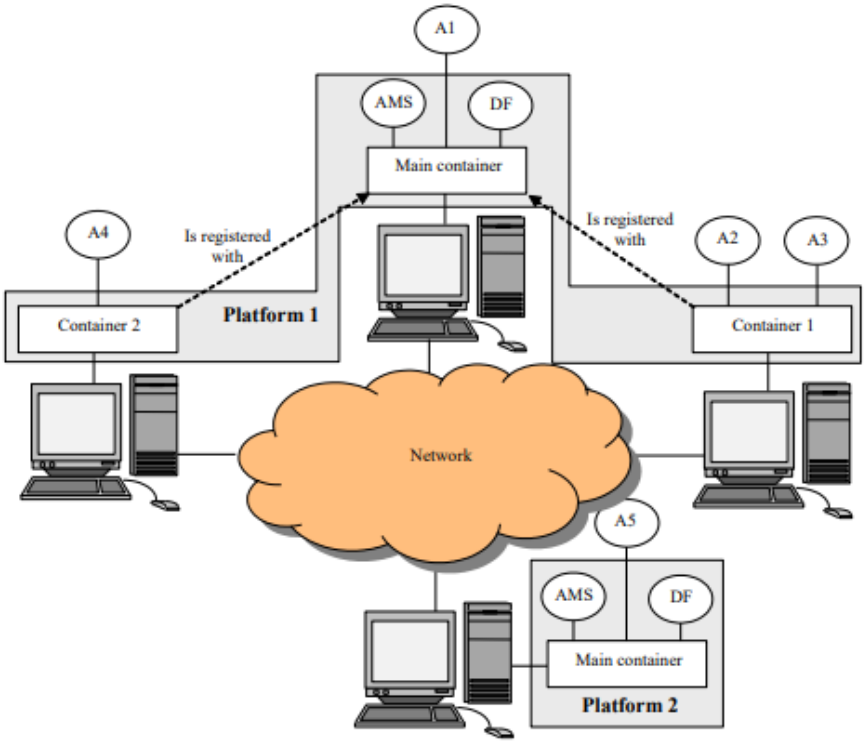
2.6.4 JADE

Java Agent DEvelopment Framework (JADE) je softvérový framework, ktorého cieľom je zjednodušiť implementáciu multiagentových systémov. Odpovedá špecifikácii FIPA, a teda zaisťuje interoperabilitu medzi ostatnými technológiami, ktoré podporujú tento agentný IEEE štandard [11]. Obsahuje tiež grafické nástroje pre podporu ladenia a nasadenia. Systém postavený pomocou JADE môže byť tiež distribuovaný medzi rôznymi strojmi s odlišnými operačnými systémami a riadený pomocou vzdialeného GUI. Konfigurácia môže byť modifikovaná v run-time pohybom agentov z jedného stroja na druhý akokoľvek a kedykoľvek. Tento systém je celý naprogramovaný v jazyku Java s požiadavkou na minimálnu verziu JRE 5 [4].

Každá inštancia JADE runtime enviroment je nazývaná kontajner a môže obsahovať viacero agentov. Skupina aktívnych kontajnerov sa nazýva platforma. V jednej platforme

musí byť aktívny minimálne jeden hlavný kontajner, čo je znázornené na príkladovej architektúre v obrázku 2.8. Hlavný kontajner na rozdiel od ostatných obsahuje 2 špeciálnych agentov, ktorí sú vytvorení implicitne pri vytvorení tohto kontajneru [5].

- AMS (Agent Management System) - priraďuje mená agentom (s ohľadom na ich jedinečnosť v platforme) a reprezentuje autoritu platformy (napr. možnosť vytvorenia/zrušenia agenta vo vzdialených kontajneroch).
- DF (Directory Facilitator) - poskytuje službu Yellow Pages (žlté stránky), pomocou ktorých sú agenti schopní nájsť iných agentov poskytujúcich službu potrebnú pre dosiahnutie ich cieľa.



Obr. 2.8: Príklad JADE architektúry s 2 platformami. Prevzaté z [5].

Kapitola 3

Súčasný stav

Táto kapitola popisuje súčasný stav, v akom sa problematika formovania koalícií za použitia genetických algoritmov nachádza. Všetky nižšie popísané princípy sú založené na probléme CFP (2.3.3), nakoľko nebol nájdený zdroj s týmto evolučným prístupom berúci do úvahy čiastkové koalície.

Na stránke súťaže MAPC[1] sú k dispozícii zdrojové kódy a dokumentácie súťažiacich z minulých rokov. Zatiaľ na konkrétny problém MAPC 2018 neboli použité metódy na formovanie koalícií pomocou genetických algoritmov, a teda boli skúmané riešenia iných problémov riešených použitým genetiky. Popísané budú ich spôsoby reprezentácie riešení spolu s genetickými operátormi. Vynechané sú fitness funkcie a konkrétne podrobnosti o riešených problémoch, nakoľko sa netýkajú problematiky spomenutej súťaže.

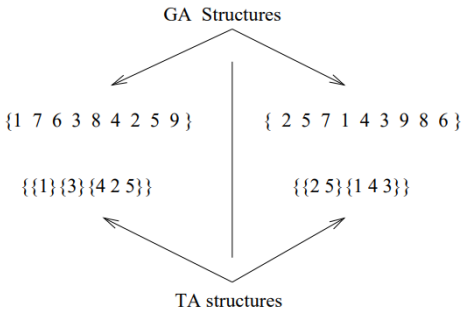
3.1 Edge recombination

Sandip Sen a Partha Sarathi Dutta v článku [16] navrhli a implementovali genetický algoritmus na riešenie CFP, v ktorom využili operátor kríženia Edge Recombination Operator. Jeho podstatou je uprednostnenie zachovania informácie o susedstve pred ich pozíciou, resp. poradím. Informácie o susedstve uchováva v pamäti štruktúra *edge table*, ktorá obsahuje spoje do a z jednotlivých hrán (edges). Tieto hrany sú potom použité na vytvorenie potomstva, v ktorom je zabránené izolovanie susedných hrán [19].

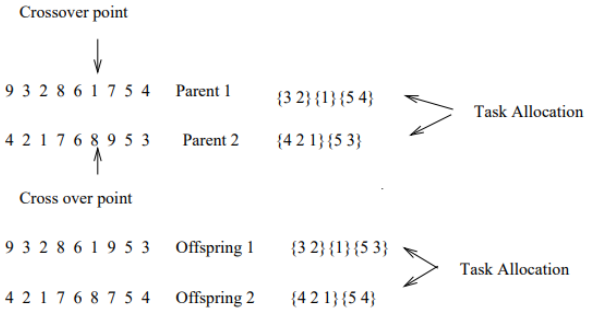
V tomto riešení sú agenti očíslovaní celými číslami od 0 do $n-1$ a tiež sú použité značky pre vytvorené koalície, ktoré hovoria o tom, akí agenti patria do ktorých koalícií. Štruktúry genetického algoritmu a odpovedajúce koaličné štruktúry (CS), z ktorých každej koalícii prislúcha určitá úloha (task allocation - TA), sú zobrazené na obrázku 3.1.

Dôsledkom tejto reprezentácie riešenia je nutnosť použiť operátor kríženia, ktorý garantuje práve jeden výskyt agenta v celej CS - Edge recombination operator. Ten je schopný splniť dané kritérium, ktoré požaduje obsiahnuť v riešení všetkých agentov bez duplicít. V prípade použitia klasických operátorov kríženia je pripustený vznik nesprávneho riešenia, ktorý je zobrazený na obrázku 3.2. Samotný typ riešeného problému a od neho odvíjajúca sa fitness funkcia nekorelujú s problémom riešeným v tejto práci, preto nie sú v tejto práci uvedené.

[1] https://multiagentcontest.org/



Obr. 3.1: Členovia populácie a príslušná CS. Prevzaté z [16].



Obr. 3.2: Problém kríženia - 2. potomok obsahujúci duplicitného agenta č. 4. Prevzaté z [16].

3.2 Subpopulácie

Alan Mehlenbacher v článku [12] používa genetický algoritmus pre riešenie problému vytvorenia optimálnej koaličnej štruktúry a aplikuje ho na problém tvorby kriminálnych gangov.

Reprezentácia riešenia je podobná ako v predchádzajúcom prípade, a teda koaličné štruktúry sú očíslované sekvenciou celých čísel, kde pozícia i reprezentuje agenta i. Číslo $j \in [1, k]$ na pozícii i reprezentuje priradenie agenta i do koalície j. Ďalej sa autor algoritmu inšpiroval technikou používanou v paralelných genetických algoritmoch - rozdelením populácie na ostrovy (2.4.4). Na každom ostrove sa nachádzajú indivíduá rozdelené do j koalícií. Rozdelenie populácie do ostrovov je podľa článku kritickým faktorom na ceste k úspechu, nakoľko izolácia ostrovov zaručuje rôznorodosť riešení, a teda aj vyhnutie sa problému konvergencie k lokálnemu optimu.

Výmena chromozómov z rôznych ostrovov je takisto, ako aj v klasickom paralelnom prevedení riešená pomocou procesu migrácie. Indivíduá sú rozdelené podľa koaličnej veľkosti do ostrovov, v rámci ktorých sa vykonávajú operácie selekcie a kríženia. Selekcia spočíva vo vybraní rodiča spárovaného s partnerom z rovnakého ostrova, ktorý do jeho chromozómu vloží nejakú z vlastných koalícií. Tento náhodný proces vkladania je opakovaný, pokým nie je vytvorený potomok s vyššou hodnotou fitness ako obaja jeho rodičia, alebo pokým nie je dosiahnutý maximálny počet iterácií. Následne je aplikovaný operátor opravy (repairing operator) zaručujúci výskyt všetkých koalícií (ktorých počet je parametrom algoritmu). Ak určitá koalícia z riešenia zmizne, je vybraná koalícia s najnižšou fitness hodnotou a polovica jej populácie je migrovaná do vyhynutej koalície.

Po krížení nasleduje mutácia, ktorá z jej podstaty nemôže vytvoriť nevalidné riešenie (neobsahujúce všetky definované koalície). Proces vykonáva určitý počet výmien jedincov medzi koalíciami a iteruje dovtedy, pokým rozdiel medzi rodičom a potomkom presahuje definovaný hranicu.

Migrácia spočíva v odoslaní zlomku každej subpopulácie do inej subpopulácie (na iný ostrov). Počas tejto operácie môže zase dôjsť k vytvoreniu neprípustného riešenia, na ktorého korekciu sú použité dva operátory - expanzia a kontrakcia. Expanzia koalície vyberá koalíciu s najnižšou hodnotou o veľkosti aspoň 1 a polovica z jej členov je presunutá do susednej koalície s vyššou hodnotou. Kontrakcia spočíva vo vybraní koalície s poradím $n+1$ pri zoradení podľa hodnoty a presunu jej členov do koalície s poradím n.

3.3 Binárne kódovanie

V článku [22] je predstavené dvojdimenzionálne binárne kódovanie pomocou matice $m \times n$, v ktorom každý stĺpec matice prislúcha jednému agentovi a každý riadok matice reprezentuje jednu úlohu (resp. koalíciu). Hodnota bunky $a_{ij} = 1$ vyjadruje, že agentovi j bola priradená úloha i, resp. bol zaradený do koalície C_i, ktorej náleží úloha i, a naopak $a_{ij} = 0$ hovorí, že agent j do koalície C_i nepatrí. Treba tiež poznamenať, jednej koalícii prislúcha práve jedna úloha. Kódovacia matica je zobrazená na obrázku 3.3.

Kvôli tejto reprezentácii bolo nutné vymyslieť nový typ operátora kríženia - dvojdimenzionálny OR operátor, ktorý pre všetky bunky matice vykoná binárnu operáciu OR, zapísané $\forall 0 \leq i < m, 0 \leq j < n, o_{ij} = a_{ij} \vee b_{ij}$, kde o_{ij} je bunka matice potomka a a_{ij}, b_{ij} sú bunky matíc rodičov. V mnohých prípadoch potomok nemusí spĺňať podmienku klasických koalícií, a teda že jeden agent môže patriť maximálne do jednej koalície. V tom prípade sú aplikované operátory opravy, aby bola podmienka splnená a v každom stĺpci sa vyskytovala hodnota 1 maximálna jedenkrát.

Fitness funkcia je definovaná ako efektivita celého systému. Ak riešenie nie je schopné splniť všetky definované úlohy, jeho fitness hodnota je 0, inak je fitness indivídua súčtom efektivity všetkých koalícií.

Úloha \ Agent	0	1	...	n-1
0	0	0	...	1
1	1	0	...	0
⋮	⋮	⋮	⋮	⋮
m-1	0	1	...	0

Obr. 3.3: Dvojdimenzionálne binárne kódovanie chromozómu.

Kapitola 4

Návrh

V prvej časti budú riešené vstupy genetického algoritmu, ktoré budú získané agentmi komunikujúcimi so serverom v scenári Agents in the City. Je popísané, akým spôsobom budú agenti so serverom komunikovať, aká bude ich úloha pri pohybe na hracej ploche a akými úpravami budú musieť spracované vnemy prejsť tak, aby boli kompatibilné so vstupmi evolučného algoritmu.

V druhej časti budú predstavené navrhnuté časti genetického algoritmu akým je kódovanie jedinca, genetické operátory, fitness funkcia a vlastnosti populácie. Cieľom tejto časti je ukázať, akým spôsobom sa dá problém priradenia agentov k úlohám v súťaži MAPC 2018 riešiť pomocou genetického algoritmu, aké úskalia riešenie tohto problému prináša a ako sa im vyhnúť. Predstavené sú rôzne varianty výberu jedincov, kríženia a tiež parametre vo fitness funkcii, ktoré budú neskôr predmetom experimentov.

4.1 Návrh hry

Prepojenie existujúcej aplikácie komunikujúcej so serverom a výpočtom genetického algoritmu bude predvedené pomocou demo aplikácie. Tá bude obsahovať agentov, ktorí sú schopní prijímať a odosielať dáta na MASSim server, prijaté dáta si budú agenti schopní medzi sebou vymieňať a ukladať. Tieto informácie budú slúžiť ako vstup genetického algoritmu, ktorý sa podľa zadaných agentov, úloh a informácií o prostredí pokúsi nájsť optimálne riešenie priradenia agentov k úlohám.

4.1.1 Server

MASSim server bude slúžiť na získavanie údajov o stave hry, ktoré budú ďalej spracúvať agenti. Dáta je potrebné dať do súladu s rozhraním genetického algoritmu, ktorý potrebné vnemy zo servera použije na svoj výpočet. Samotný server vo formáte *JAR* je dostupný na stránkach GitHub-u MASSim [3].

Server pre svoje fungovanie bude potrebovať *JSON* súbor, ktorý je nutné mať uložený v pracovnom adresári v zložke *conf*, alebo ho predať pomocou parametra príkazového riadku. Konfigurácia sa skladá zo 4 hlavných blokov: server, manual-mode, match a teams.

Z bloku server je pre účely získania zopár vnemov zo servera dôležitý parameter *teamSize*, ktorý hovorí o počte agentov v tíme, ostatné parametre budú mať hodnoty prevzaté z ukážkových konfiguračných súborov. Jedným z takých parametrov je aj *agentTimeout*, ktorý určuje čas, za ktorý musí agent od prijatia vnemov odoslať svoju akciu pre nasledujúci simulačných krok. Blok teams obsahuje názvy jednotlivých agentov spolu s heslami pre

prihlásenie na server. Manual-mode je pre samotné získanie malého počtu vnemov nepotrebný, nakoľko vyjadruje spôsob, akým budú družstvá párované do jednotlivých zápasov, ak je v bloku `server` nastavený *tournamentMode* na hodnotu *manual*.

Najpodstatnejšie informácie, z ktorých je väčšina závislá na scenári hry sa nachádzajú v bloku `match`. Nachádza sa tu údaj o počte simulačných krokov, ktorý bude musieť byť zvolený na dostatočne vysokú hodnotu, aby sa za ten čas dokázalo nazbierať dostatočne veľa informácií (napríklad polohy uzlov so surovinami) pre výpočet algoritmu. Ďalej sa tu nachádzajú hodnoty hraníc hracej plochy - minimálna a maximálna zemepisná šírka a dĺžka. Pravdepodobne najdôležitejším údajom v tejto sekcii sú údaje o agentových rolách (ich názvy, atribúty - rýchlosť, zručnosť...) a počet agentov s jednotlivými rolami. Ďalšou nezanedbateľnou informáciou sú údaje o generovaní zariadení na určité miesta mapy (sklady, nabíjacie stanice, uzly atď.), maximálny a minimálny počet surovín, hĺbka grafu určujúceho predmety, ktoré sú potrebné na skladanie iných predmetov a podobne. Tieto informácie je okrem ďalších možno uviesť aj v samostatných súboroch. Všetky údaje, okrem veľkosti tímu a početnosti druhov rolí v tíme (tieto údaje budú korigované vzhľadom na účely experimentov) budú zachované z ukážkových konfiguračných súborov súťaže.

4.1.2 Agenti

Agenti budú po úspešnom prihlásení komunikovať so serverom (bližšie popísané v časti 4.1.3), a tým budú získavať dôležité informácie o svojom stave, ako aj stave prostredia. Agenti si budú dáta zo servera ukladať do svojich premenných, ktoré budú môcť následne predať na vstup genetického algoritmu.

Je dôležité rozlišovať informácie, ktoré sú špecifické pre konkrétneho agenta a informácie, ktoré sú súčasťou hry (mal by o nich vedieť každý agent). Špecifické informácie budú uložené priamo u agenta - aktuálna pozícia, rola, náklad, zručnosť, rýchlosť, stav batérie a stav poslednej akcie.

Informácie o stave hry, ktoré sú spoločné pre agentov, a teda sú reprezentáciou stavu prostredia, v ktorom sa agenti nachádzajú budú uložené v štruktúre reprezentujúcej stav hry - úlohy na vykonanie, vlastnosti predmetov ako objem, zložky, role, informácie o nabíjacích staniciach a súradnice hracej plochy. Aby nevznikali konflikty pri zapisovaní do tejto štruktúry, bude vytvorená jedným agentom, ktorý doň bude môcť ako jediný zapisovať, ostatní naň ale budú vlastniť odkaz. Rozhodnutie o tom, ktorý z agentov to bude prebehne na začiatku simulácie, kde prvý agent, ktorý ako prvý dostane možnosť prijímať prvotnú správu zo servera sa stáva lídrom. Tento princíp musí byť zachovaný, keďže prvý agent vykonávajúci akciu bude pri svojej prvej akcii potrebovať referenciu na už existujúci objekt hry.

Ďalšou vecou, ktorá stojí za zmienku je objavovanie uzlov s predmetmi. Keďže poloha uzlov je na začiatku simulácie neznáma, agenti o nich získavajú informácie až po ich nájdení. Ak však ale nejaký agent nájde uzol, musí o tom informovať všetkých svojich spoluhráčov. Keďže poloha uzlov je informácia týkajúca sa prostredia, tak agent, ktorý uzol našiel predáva informáciu o jeho názve, polohe a type predmetu lídrovi tímu. Líder následne vytvorí záznam v štruktúre hry o tomto uzle.

Agent bude disponovať len funkciami špecifickými pre účel projektu - zbieranie informácií o uzloch so surovinami, a v prípade nutnosti dobíjanie batérie. Pre využitie služieb agenta, ktoré sú už k dispozícii vďaka ukážkovým kódom zverejneným organizátormi súťaže (komunikácia server-agent, agent-agent), bude trieda reprezentujúca agenta rozširovať abstraktnú triedu `massim.javaagents.agents.Agent`, ktorá implementuje potrebné metódy.

4.1.3 Rozhrania

Ako bolo vyššie popísané, agent sa musí na serveri autentizovať, musí vedieť na server odoslať správy a tiež ich aj prijať. Agenti tiež zdieľajú informácie, ako sú napríklad zvolenie lídra, stav hry a podobne, preto je potreba existencie rozhraní, ktoré túto komunikáciu umožnia.

Agent-server

Na komunikáciu so serverom existuje MASSim protokol, ktorý definuje správy v *XML* formáte odosielané smerom zo servera a na server. Agenti sú schopní komunikovať pomocou týchto správ používaním štandardných TCP soketov. Navrhnutí agenti musia byť schopní iniciovať komunikáciu so serverom zaslaním správy `AUTH-REQUEST`, ktorá obsahuje názov agenta a jeho prihlasovacie heslo. Ak bude agent úspešne autentifikovaný (informáciu o stave prihlásenia obsahuje správa `AUTH-RESPONSE`), agent musí byť schopný prijať nasledujúce typy správ (server reálne odosiela väčšie množstvo typov správ, ale pre jednoduché účely zbierania informácií o agentovom prostredí budú stačiť nasledujúce):

- `SIM-START` - agent z tejto správy musí byť schopný spracovať informácie o veľkosti hracej plochy, svoje atribúty (rýchlosť, zručnosť, kapacita, rola) a informácie o predmetoch (ich objem, predmety potrebné na ich zloženie a role, ktorých účasť je pri tomto úkone vyžadovaná),
- `REQUEST-ACTION` - na začiatku každého simulačného kroku je táto správa odoslaná serverom a obsahuje informácie o aktuálnom stave simulácie - pozícia agenta, stav batérie, výsledok poslednej akcie, vlastnené predmety, zadané úlohy a informácie o zariadeniach (polohy uzlov s predmetmi sú známe až potom, čo ich agent objaví).

Agent po obdržaní správy `REQUEST-ACTION` bude schopný odoslať správu typu `ACTION`, ktorá musí obsahovať plánovanú akciu v ďalšom kroku a jej prípadné parametre. V tomto modele bude agent na server odosielať len 2 typy akcie. Prvou je `goto`, ktorá má 2 parametre - zemepisnú šírku a dĺžku miesta, na ktoré sa má agent presunúť. Druhý typ akcie, ktorá je tentokrát bez parametrov je nabíjanie batérie `charge` a používa sa v prípade, že agent nemá dostatok energie na to, aby ďalej skúmal mapu a hľadal uzly so surovinami.

Pre sprostredkovanie komunikácie medzi serverom a agentmi bude použitý existujúci nástroj EISMASSim, ktorý bude mapovať spomenuté správy v *XML* formáte na volania metód jazyka Java. Princíp jeho používania, ako aj niektoré triedy budú v implementačnej časti prevzaté a upravené opäť z projektu servera MASSim ([3]). Po vytvorení inštancie rozhrania ho bude nutné uviesť do chodu. Ďalej budú agenti zaregistrovaní k tomuto rozhraniu, aby mohli byť pripojení k serveru (preto je potreba rozhraniu predať konfiguračný súbor, v ktorom sú okrem iných informácií aj mená a heslá jednotlivých agentov). Taktiež je nutná asociácia agenta (trieda komunikujúca so serverom cez rozhranie) s entitou (vozidlo, ktoré sa pohybuje po mape), pričom prebehne samotná autentifikácia.

Po úspešnom overení prístupových údajov serverom budú správy (vnemy) zo servera získavané a ukladané do agentovej bázy znalostí v každom simulačnom kroku. Znalosti budú v každom kroku vymenené za aktuálne, preto je potreba uchovávať ich históriu a ukladať ich do iných pomocných štruktúr kompatibilných so vstupmi genetického algoritmu. Po predaní vnemov budú agenti vyzvaní k odoslaniu akcie, ktorú chcú v danom kroku vykonať na základe poskytnutých informácií o prostredí. Následne budú tieto akcie pomocou EISMASSim rozhrania odoslané na server.

Agent-agent

Existujúca schránka v podobe triedy `massim.javaagents.MailService` bude použitá na komunikáciu medzi agentmi. Tá je tiež dostupná na GitHub-e ([3]) a deklaruje metódy umožňujúce odosielanie správ, ktorých obsahom sú vnemy prostredia agentov. Toto rozhranie bude použité v projekte na odoslanie správy všetkým agentom o zvolení lídra tímu a tiež oznámení atribútov nájdeného uzla lídrovi, ktorý o ňom vytvorí záznam v spoločnom objekte.

Pre účely projektu riešeného v tejto práci bude rozhranie doplnené o možnosť zdieľania štruktúry hry, aby všetci agenti mali konzistentné informácie o prostredí hracej plochy.

4.1.4 Predspracovanie dát

Po tom, čo bude odsimulovaný dostatočný počet krokov na to, aby agenti získali potrebné informácie o simulácii a našli určitý počet uzlov s predmetmi, budú agenti serializovaní - všetky podstatné údaje, ktoré si uložili do pamäte spolu so stavom hry budú uložené do súboru. Táto akcia je užitočná z toho dôvodu, že server zvyčajne generuje náhodné dáta, t.j. pri každom spustení by boli informácie o prostredí iné a nebolo by možné parametre genetického algoritmu správne odladiť - na to je potrebná konzistencia dát. Ďalšou výhodou toho prístupu je ten, že zbieranie dát zo servera počas simulácie je časovo náročná záležitosť, preto ich stačí raz uložiť do súboru, a pri opätovnom spustení len načítať do pamäte.

Server generuje predmety a role takým spôsobom, že všetky úlohy vyžadujú z každého existujúceho základného predmetu (t.j. predmetu, ktorý sa neskladá zo žiadnych čiastkových predmetov) istý počet kusov. Tiež úlohy vyžadujú účasť všetkých rolí na tvorbe potrebných predmetov. Preto musia byť deserializované dáta (načítané zo súboru) pred výpočtom genetického algoritmu upravované. Zadaným úlohám budú odstránené isté vyžadované predmety a role. Naopak agentom budú pridané predmety tak, aby sa úlohy líšili čo najviac a aby agenti mali istý počet rôznych predmetov, podľa ktorých ich bude možno genetický algoritmus priradiť k týmto úlohám. Bez tejto úpravy by problém rozdelenia agentov k úlohám nebol tak zaujímavý, nakoľko by každý agent vlastnil predmet (resp. reprezentoval takú rolu), ktorý potrebuje každá úloha.

4.2 Genetika

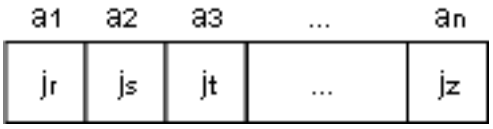
Pri návrhu genetického algoritmu bolo v každom kroku nutné zohľadniť podstatu problému, na ktorý bude aplikovaný. Nebol nájdený žiaden literárny zdroj, v ktorom by bol genetický algoritmus použitý na riešenie tak komplexného problému, akým je rozdelenie agentov do koalícií v hre MAPC 2018, preto bolo možné čerpať inšpiráciu z existujúcich riešení len do určitej hĺbky. V ďalších častiach bude predstavené, kódovanie chromozómu, genetické operátory a taktiež navrhnutá fitness funkcia.

4.2.1 Reprezentácia riešenia

Po zvážení možných reprezentácií chromozómu pre daný problém bol zvolený prístup celočíselného jednodimenzionálneho kódovania, v ktorom je každému agentovi priradená práve jedna úloha. Nebude aplikovaná teória čiastkových koalícií (2.3.2), keďže by bolo nutné využiť minimálne dvojdimenzionálnu reprezentáciu, s ktorou sú spojené problémy pri krížení. Vo výsledku bude stratégia závisieť na hernej logike, ktorá bude navrhnutá a implementovaná mimo tejto práce.

V nasledujúcich odsekoch budú úlohy označené celými číslami v intervale $[0, m-1]$, kde m je počet úloh, a agenti sú reprezentovaní celými číslami v intervale $[0, n-1]$, kde n je počet agentov.

V celočíselnom kódovaní je chromozóm reprezentovaný vektorom celých čísel o veľkosti n v intervale $[0, m-1]$. Zložky v tomto prípade udávajú čísla úloh a indexy vektora sú čísla agentov. Sémantika chromozómu sa dá interpretovať ako vektor úloh, v ktorom úloha $0 \leq j \leq m-1$ na indexe $0 \leq a \leq n-1$ udáva, že agent a je viazaný k riešeniu úlohy j. Princíp je graficky znázornený na obrázku 4.1.



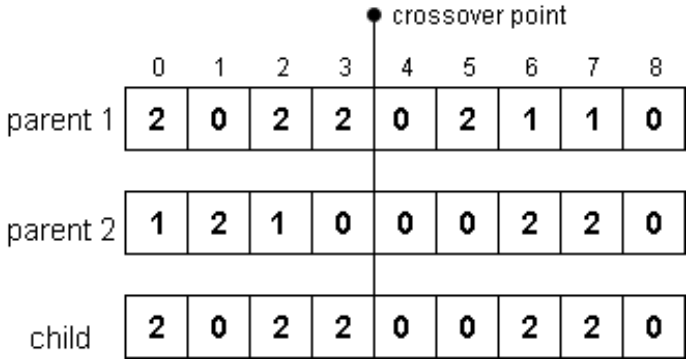
Obr. 4.1: Celočíselné jedno-dimenzionálne kódovanie.

Výhodou tohto kódovania oproti dvojdimenzionálnemu je, že v prípade klasických koalícií odpadá nutnosť použitia korekčného operátora na zaistenie prípustnosti riešenia, ako bolo popísané v sekcii 3.2. Vzhľadom na tento druh kódovania môže však nastať komplikácia, kedy sa z riešenia pri krížení vytratí niektorá z úloh. Tento problém možno riešiť vhodným typom kríženia, alebo fitness funkciou, ktorá chromozómy s týmto deficitom náležite penalizuje.

4.2.2 Genetické operátory

Po zvolení vhodnej reprezentácie riešenia boli navrhnuté genetické operátory (kríženie a mutácia sú priamo závislé na druhu kódovania). Navrhnutých bude viacero druhov operátorov selekcie a kríženia, pričom ich výsledky budú porovnané v kapitole experimentov 6

- **Selekcia** - Úlohou selekcie bude vyberať tých jedincov, u ktorých sa predpokladá najvyššia pravdepodobnosť produkcie kvalitného potomstva (kvalita je určená fitness funkciou 4.2.3). Prvým druhom navrhnutej selekcie je ruletový výber, kde šanca na výber jedinca pre účely kríženia je daná pomerom jeho fitness hodnoty s celkovou hodnotou populácie. Ďalším typom selekcie plánovanej na implementáciu je algoritmus turnaja, kde bude predom vybraný určitý počet jedincov, z ktorých bude ku kríženiu pripustený ten s najvyššou hodnotou fitness. Oba algoritmy by mali na rozdiel od princípu výberu elity zamedziť rýchlu konvergenciu k lokálnemu optimu.
- **Kríženie** - Viaceré druhy kríženia budú opäť predmetom pokusov. Implementovanými budú jednobodové a viacbodové kríženia. Pre možnosť uprednostnenia génov od kvalitnejšieho jedinca je tiež navrhnuté parametrizované kríženie rovnomerného rozloženia, v ktorom pre každý gén nezávisle je daná pravdepodobnosť (závislá na parametri) výberu od jedného alebo druhého rodiča. Keďže ani jeden z týchto druhov kríženia nezaručuje zachovanie všetkých úloh v potomkovom riešení (4.2), bude tento efekt potláčať fitness funkcia.
- **Mutácia** - Pri mutácii bude s istou pravdepodobnosťou náhodne zmenená hodnota génu práve vzniknutého potomka, tým pádom bude agentovi na danom indexe priradená iná úloha. Výška pravdepodobnosti vykonania mutácie bude tiež experimentálne upravovaná.



Obr. 4.2: Chýbajúca úloha č. 1 v potomkovi po procese jednobodového kríženia.

Na obrázku 4.2 môžeme vidieť problém s jednobodovým krížením (problém sa tiež týka viacbodového, ako aj kríženia rovnomerného rozloženia) pri jednodimenzionálnom celočíselnom kódovaní tohto problému. Pri kríženi jedincov na obrázku je po zvolení indexu kríženia $i = 4$ vytvorený nový jedinec, ktorého žiaden gén neobsahuje hodnotu 1 čo znamená, že úloha s týmto číslom nie je priradená žiadnemu agentovi, a teda nebude môcť byť riešená. Nakoľko tento fenomén nie je možné eliminovať s pomocou zvolenej reprezentácie riešenia, ani genetických algoritmov, bude ho nutné hendikepovať pomocou fitness funkcie, ktorá takto vytvorený chromozóm ohodnotí nižším počtom fitness bodov.

4.2.3 Fitness funkcia

V poslednej časti návrhu genetického algoritmu bolo nutné navrhnúť fitness funkciu, pomocou ktorej je možné objektívne ohodnotiť kvalitu jednotlivých nájdených riešení. Správna fitness funkcia musí kvôli zvolenému kódovaniu zaručiť, aby samotný genetický algoritmus preferoval riešenia, v ktorých je ku každej zadanej úlohe priradené také množstvo agentov, aby bolo možné danú úlohu riešiť čo najefektívnejšie, a v čo najkratšom čase vzhľadom na zadaných agentov. Na výšku kvality chromozómu vplýva viac faktorov, ktoré vyplývajú priamo z podstaty zadania MAPC 2018. Vo výslednom riešení boli zohľadnené vlastnosti úloh, ktoré najviac interferujú s vlastnosťami agentov.

- Predmety. Predmety vlastnené určitými agentmi sú porovnané s predmetmi, ktoré úloha (ku ktorej sú daní agenti priradení) potrebuje pre svoje splnenie - v celom kontexte fitness funkcie sú uvažované len tzv. základné predmety (suroviny), t.j. predmety, ktoré neobsahujú žiadne časti - predmety úloh a agentov na nelistovej úrovni je nutné rekurzívne prepočítať až na listovú úroveň.
- Role. Je zohľadnený počet jednotlivých rolí agentov priradených k určitej úlohe vzhľadom na počet konkrétnych agentových rolí požadovaných danou úlohou.
- Uzly. Počíta sa pomer trasy medzi agentmi priradenými k úlohe a uzlami zásob (z ktorých je možno vyťažiť suroviny potrebné pre dohotovenie úlohy) a rýchlosti týchto agentov. Výsledkom je teda predpokladaný čas cesty.

Každý z vyššie spomenutých bodov je algoritmicky vyhodnotený a prevedený na reálne číslo vyjadrujúce závažnosť jeho vplyvu na kvalitu riešenia. Nakoľko neexistuje mierka, podľa ktorej by bolo možné exaktne porovnať body ako také, a teda určiť pomer definujúci, ktorý bod je o koľko fitness čísel nutné uprednostniť pred inými bodom v aktuálnej

situácii (napríklad obetovať agenta, ktorý je síce bližšie k požadovanému uzlu zásob, ako agent priradený k inej úlohe, ale na rozdiel od neho nemá požadovanú rolu na vyskladanie iného predmetu pre danú úlohu), bolo nutné pri výpočtoch zaviesť parametre, ktoré budú korigované vzhľadom na výsledky riešení dosiahnutých vo fáze experimentov.

Pokračovať bude podrobný popis troch segmentov funkcie fitness, z ktorých každý je kalkulovaný v kontexte jednej úlohy patriacej celkovému riešeniu. Každý z týchto krokov fitness funkcie ohodnocuje riešenie tzv. penalizáciami za isté priestupky. Výsledná fitness funkcia udávajúca kvalitu chromozómu bude daná pomerom sumy odmien a súčtu penalizácií fitness funkcie pre všetky úlohy v probléme. Cieľom vyvíjaného genetického algoritmu je minimalizovať výsledok fitness funkcie, ktorá penalizuje horšie riešenia nižším počtom bodov.

Predmety

Prvá časť fitness funkcie pracuje s informáciami o predmetoch, ktoré vyžaduje daná úloha a predmetmi vlastnenými agentami priradenými k danej úlohe. Tiež je započítaná zručnosť agentov vplývajúca na rýchlosť ťaženia surovín, ako aj potrebná a vlastnená kapacita - v prípade prekročenia kapacity je započítaná aj rýchlosť agentov pôsobiaca na zbavovanie sa predmetov v obchodoch, dielňach alebo skladoch vzhľadom na konkrétny prípad použiteľnosti predmetu.

V nasledujúcich odsekoch bude I označovať množinu všetkých základných predmetov, J je množina plánovaných úloh, A množina agentov a množina $A_j, j \in J$ značí, množinu agentov $a \in A$ priradených k úlohe j. Vstupmi daného segmentu funkcie sú teda prvky:

- $carried_a^i$ - počet predmetov druhu i vlastnených agentom a
- $needed_j^i$ - počet predmetov druhu i potrebných pre splnenie úlohy j
- $volume_i$ - objem predmetu i
- $tskill_j$ (total skill) - suma zručnosti agentov priradených k úlohe j
- $tspeed_j$ (total speed) - suma rýchlosti agentov priradených k úlohe j
- $tload_j$ (total load) - suma zaťaženia agentov priradených k úlohe j
- $tmload_j$ (total max load) - suma maximálnej kapacity agentov priradených k úlohe j

Najprv je nutné vytvoriť štruktúru uchovávajúcu informáciu o tom, koľko predmetov rôznych druhov je ešte potrebné získať (remaining), a ktoré predmety vlastnené agentmi sú zbytočné (nepotrebné pre úlohu, ku ktorej sú agenti priradení). Pre každú úlohu je teda nutné odpočítať pre každý druh predmetu počet predmetov daného druhu vlastneného agentmi priradenými k danej úlohe od počtu predmetov toho istého druhu vyžadovaných touto úlohou. Výsledkom tejto operácie bude teda štruktúra, v ktorej kladné číslo pri type predmetu určuje, koľko ešte predmetov je nutné získať, záporné číslo hovorí o tom, koľko predmetov daného typu je zbytočných a nulová hodnota indikuje vlastníctvo presného počtu predmetov pre danú úlohu:

$$\forall j \in J, \forall i \in I : remaining_j^i = needed_j^i - \sum_{a \in A_j} carried_a^i \,. \tag{4.1}$$

Následne je vypočítaný požadovaný úložný priestor (needed load space) na uchovanie zostávajúcich predmetov v danej úlohe j s využitím vstupného vektora objemov predmetov v, pričom v_i určuje objem predmetu i:

$$\forall j \in J : nls_j = \sum_{i \in I, remaining_j^i > 0} remaining_j^i * v_i \,. \tag{4.2}$$

Pred aplikovaním vzorca pre výslednú penalizáciu je nutné najprv počítanie potrebného miesta pre predmety (needed load space penalty), ktoré je nevyhnutné vyťažiť s ohľadom na zručnosť agentov:

$$\forall j \in J : nlsp_j = \begin{cases} nls_j, & tskill_j = 0 \\ \frac{nls_j}{tskill_j}, & inak \end{cases} \tag{4.3}$$

Druhý pomocný vzorec vyjadruje postih za nedostatok miesta pre potrebné predmety (lack of space penalty). Nedostatok miesta pre zvyšné potrebné predmety nastáva v prípade, ak platí podmienka $nls > tmload - tload$:

$$\forall j \in J : lsp_j = \begin{cases} \frac{nls_j - (tmload_j - tload_j)}{tspeed_j}, & tspeed_j \neq 0 \wedge nls_j > tmload_j - tload_j \\ nls_j - (tmload_j - tload_j), & tspeed_j = 0 \wedge nls_j > tmload_j - tload_j \\ 0, & inak \end{cases} \tag{4.4}$$

Výsledný vzorec pre vypočítanie penalizácie za požadované predmety (item penalty) pre úlohu j je definovaný súčtom výsledkov dvoch predošlých pomocných medzi výpočtov

$$\forall j \in J : ip_j = nlsp_j + lsp_j \,. \tag{4.5}$$

Role

Role sú spolu s predmetmi dva najvplyvnejšie faktory pri určovaní kvality riešenia problému. Existuje veľa spôsobov, ako zohľadniť prítomnosť vyžadovaných rolí v kontexte funkcie fitness, spomedzi ktorých bol vybraný variant s najlepšími výsledkami z pohľadu experimentov, a ten je popísaný v tejto časti. Ako už bolo naznačené, je potreba získať numerický výsledok porovnania agentových rolí priradených k úlohám a rolí, ktoré daná úloha reálne vyžaduje.

Vstupmi časti fitness funkcie, ktorá pracuje s agentovými rolami sú:

- R - označuje množinu typov rolí, ktorými disponujú agenti určení na výpočet GA
- ar^r (agents by role) - označuje počet všetkých agentov s rolou r, ktorí sú súčasťou riešenia GA
- $assigned_j^r$ - označuje počet rolí typu r priradených k úlohe j
- $duties^r$ - počet predmetov v rámci všetkých úloh, na ktorých vytvorení sa musí podieľať rola r
- $duties_j^r$ - počet predmetov v kontexte úlohy j, na ktorých vytvorení sa musí podieľať rola r

Po kalkulácii vyššie spomenutých štruktúr je zahájený proces výpočtu penalizácie za nedostatok požadovaných agentových rolí, ktorý prebieha v niekoľkých krokoch. Najprv je pre každú rolu zistený ideálny počet agentov s touto rolou priradených k určitej úlohe (required agent roles):

$$\forall j \in J, \forall r \in R : duties^r > 0 : rar_j^r = \frac{ar^r * duties_j^r}{duties^r} \,. \tag{4.6}$$

S informáciami o počte priradených agentov s istou rolou a požadovanom počte priradených agentov s touto rolou je vypočítaný rozdiel týchto dvoch hodnôt, a tým je zistená odchýlka požadovanej a skutočnej hodnoty:

$$\forall j \in J, \forall r \in R : diff_j^r = rar_j^r - assigned_j^r \,. \tag{4.7}$$

V prípade, že je odchýlka nekladná, úloha má buď presný, alebo väčší požadovaný počet rolí daného typu ako potrebuje, tým pádom sa zvyšujú šance na jej úspešné dokončenie. Ak je odchýlka príliš nízka, neudeľuje sa penalizácia za plytvanie rolami. Táto skutočnosť je však zohľadnená pri inej úlohe, v ktorej dané role budú chýbať - odchýlka v tejto úlohe pre danú rolu bude kladná. Pre tieto úlohy je počítaná hodnota určujúca počet predmetov, ktoré budú kvôli nedostatku danej role dokončené neskôr, alebo nebudú dokončené vôbec. Je teda nutné vypočítať, koľko predmetov vychádza na jedného agenta s aktuálnou počítanou rolou (duty per role):

$$\forall j \in J, \forall r \in R : rar_j^r \neq 0 : dpr_j^r = \frac{duties_j^r}{rar_j^r} \,, \tag{4.8}$$

a následne je hodnota násobená počtom rolí, ktoré v danej úlohe chýbajú. Pre korigovanie miery dopadu na výsledok fitness funkcie medzi penalizáciou za predmety a role, boli do pomocných vzorcov pre penalizáciu rolí pridané parametre ϵ pre aplikovanie vyššej penalizácie za úplnú neprítomnosť vyžadovanej role (role penalty complete):

$$\forall j \in J : rpc_j = \sum_{r \in R, diff_j^r = rar_j^r} diff_j^r * dpr_j^r * \epsilon \,, \tag{4.9}$$

a η len za jej čiastočný nedostatok (role penalty partial):

$$\forall j \in J : rpp_j = \sum_{r \in R, diff_j^r \neq rar_j^r} diff_j^r * dpr_j^r * \eta \,. \tag{4.10}$$

Pred vykonaním experimentov je možné tvrdiť, že pre elimináciu najkritickejšieho javu, keď potrebná rola je z úlohy úplne vynechaná bude nutné parameter ϵ nastaviť na vyššiu hodnotu ako parameter η. Tiež je jasné, že parametre budú musieť byť po vytvorení logiky pre hru MAPC 2018 testované, a prípadne dynamicky menené pri každom zahájení výpočtu genetického algoritmu vzhľadom na pomer počtu agentov, ktorí sú k dispozícii, a predmetov, ktoré je potreba vytvoriť (rôzne pomery týchto dvoch veličín by mali za následok rôzne pomery výsledkov týchto dvoch častí fitness funkcie, ktorá by mohla dávať skreslené výsledky). Výsledná penalizácia rolí (rp - role penalty) je daná súčtom:

$$\forall j \in J : rp_j = rpc_j + rpp_j \,. \tag{4.11}$$

Uzly

Odhadovaný čas príchodu agentov k jednotlivým uzlom so zásobami je pravdepodobne položka s najmenším dopadom na výslednú kvalitu riešenia. Intuitívne je najdôležitejšie priradiť k úlohám agentov s požadovanými rolami, aby vedeli plniť požiadavky daných úloh a tiež je vhodné, aby daní agenti vlastnili určité množstvo potrebných surovín pre splnenie úloh. Pozícia agentov sa však neustále mení, preto bude reálny význam tejto časti fitness funkcie odhalený až pri nasadení do reálneho scenára.

Ďalšou z nevýhod okrem predpokladaného nízkeho vplyvu na kvalitu je réžia spojená so samotným výpočtom štruktúr potrebných v tejto sekcii. Pre všetkých agentov je nutné zistiť ich predpokladaný čas príchodu k práve jednému uzlu pre každý predmet, ku ktorému je najbližšie (napr. predmet item0 má známe 3 uzly, takže pre každého agenta je nutné uložiť identifikátor toho v najkratšej vzdialenosti).

Vstupmi sekcie sú:

- $reward_j$ - odmena za splnenie úlohy j, pre $\forall j \in J$
- $speed^a$ - rýchlosť agenta a, pre $\forall a \in A$
- RN_a^i - množina obsahujúca vzdialenosti agenta a od všetkých známych uzlov pre predmet i, pre $\forall i \in I, \forall a \in A$
- $remaining_j^i$ (sekcia 4.2.3)- počet predmetov typu i, ktorý je ešte potrebný nazbierať pre splnenie úlohy j, pre $\forall j \in J, \forall i \in I$

Prvým krokom je výpočet predpokladaného času príchodu agenta a k najbližšiemu zdrojovému uzlu (closest resource node) pre predmet i (funkcia $min(A)$ vyberá najmenší prvok množiny A):

$$\forall a \in A, \forall i \in I : clrn_a^i = \frac{min(RN_a^i)}{speed^a} . \tag{4.12}$$

Po výpočte najbližších uzlov je pre jednotlivé úlohy j spočítaný priemerný predpokladaný čas príchodov agentov k uzlom (average resource node) pre predmet typu i, ak je hodnota $remaining_j^i > 0$, t.j. daný z daného predmetu je stále nutné vyťažiť aspoň jeden kus pre splnenie úlohy:

$$\forall a \in A, \forall i \in I : avgrn_j^i = \frac{\sum_{a \in A_j} clrn_a^i}{|A_j|} . \tag{4.13}$$

Výsledná penalizácia za predpokladaný čas dorazenia (time penalty) k uzlom pre úlohu j teda predstavuje suma priemeru času jazdy k najbližšiemu uzlu každého predmetu všetkých priradených agentov k tejto úlohe, pričom je do vzorca pridaný koeficient γ pre nastavenie váhy medzi jednotlivými penalizáciami podľa potreby riešeného problému:

$$\forall j \in J : dp_j = \sum_{i \in I} avgrn_j^i * \gamma . \tag{4.14}$$

Výpočty v tejto sekcii boli zamerané len v rámci kontextu jednej úlohy, v ktorej je výsledná penalizácia definovaná podielom súčtu odmien za splnenie daných úloh a súčtu 3 vyššie popísaných penalizácií, a teda:

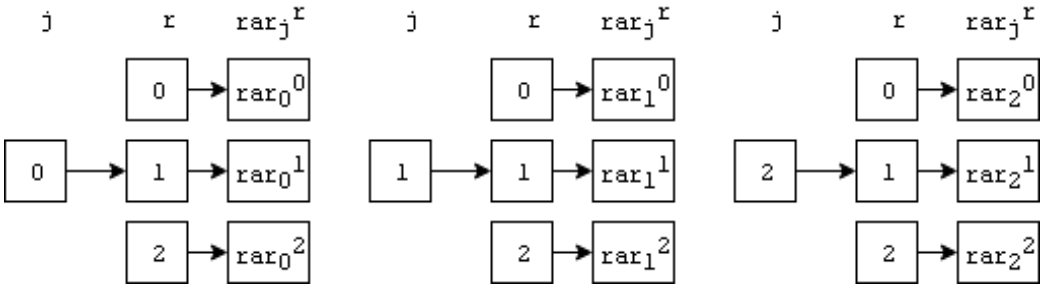
$$fp = \frac{\sum_{j \in J} reward_j}{\sum_{j \in J} ip_j + rp_j + dp_j} . \tag{4.15}$$

4.2.4 Populácia

Vlastnosti populácie, ako je jej počet jedincov alebo počet generácií, budú určené experimentálne na základe dosiahnutých výsledkov. Pri procese tvorby novej populácie je aplikovaný tzv. generačný model s prístupom elitizmu, t.j. určitý počet elitných jedincov (opäť určený experimentálne) je automaticky presunutý do ďalšej generácie. Tým sa zaistí zachovanie vysoko kvalitných riešení, ktorých gény budú ďalej predávané na potomkov. Zvyšní členovia budúcej generácie sú doplnení potomkami rodičov z predošlej generácie.

Navrhnuté boli 2 spôsoby tvorby počiatočnej populácie, ktoré budú opäť testované a porovnané v experimentoch. Prvým typom tvorby iniciálnej populácie je aplikácia náhodného procesu. Vektor reprezentujúci riešenie je postupne plnený náhodnými číslami v intervale $[0, m-1]$ pri počte úloh m, a tento postup je aplikovaný na každého jedinca v populácii. Pri takto vytvorenej populácií sa očakáva vysoký nárast priemernej hodnoty fitness v prvých cykloch algoritmu.

Druhým navrhovaným spôsobom je použitie heuristiky, ktorá minimalizuje penalizáciu chromozómov s neadekvátnym počtom rolí. V prvom kroku je nutné vytvoriť štruktúru, ktorá bude uchovávať optimálny počet jednotlivých agentových rolí v každej úlohe (obrázok 4.3) - jej definícia sa nachádza vo vzorci 4.6. Následne bude pre každý index a vektora z intervalu $[0, n-1]$ vybrané náhodné číslo úlohy j z intervalu $[0, m-1]$. Ak $rar_j^r > 0$ (úloha j stále potrebuje na svoje splnenie rolu r agenta a), je indexu a vektoru riešenia priradené číslo j a hodnota rar_j^r je znížená o hodnotu 1. Ak je naopak $rar_j^r \leq 0$, potom je vyhľadané nové číslo k z intervalu $[0, m-1]$ také, že $k \neq j$, a celý proces sa opakuje. Po vyčerpaní všetkých čísel úloh bez úspešného priradenia je priradená náhodná úloha. Tento algoritmus je opakovaný pre všetkých členov počiatočnej populácie. Pre takto vytvorené riešenie je očakávaná minimálna penalizácia za deficit rolí v jednotlivých úlohách, avšak krivka zlepšovania hodnoty fitness bude pravdepodobne stúpať pomalšie.



Obr. 4.3: Štruktúra uchovávajúca chýbajúci počet jednotlivých rolí pre každú úlohu.

Kapitola 5

Implementácia

Programová realizácia projektu je rozdelená na dve časti. Prvou je demo aplikácia, ktorá simuluje pohyb agentov po hracej ploche a zbieranie informácií o prostredí. To je zaistené komunikovaním naprogramovaných agentov so serverom MASSim cez využité rozhranie.

Druhá časť zozbierané informácie prijíma na vstupe a pomocou výpočtu genetického algoritmu hľadá optimálne riešenie priradenia agentov k úlohám vygenerovaným serverom s využitím informácií zbieraných počas simulácie.

Obe časti budú skúmané na programovej úrovni jazyka Java. Ten bol zvolený z dôvodu kompatibility s nástrojom Jade, do ktorého má byť výsledné riešenie integrovateľné. Na popis komponentov programu sú použité diagramy tried zobrazujúce vzťahy medzi balíkmi, triedami a rozhraniami systému.

5.1 Demo aplikácia

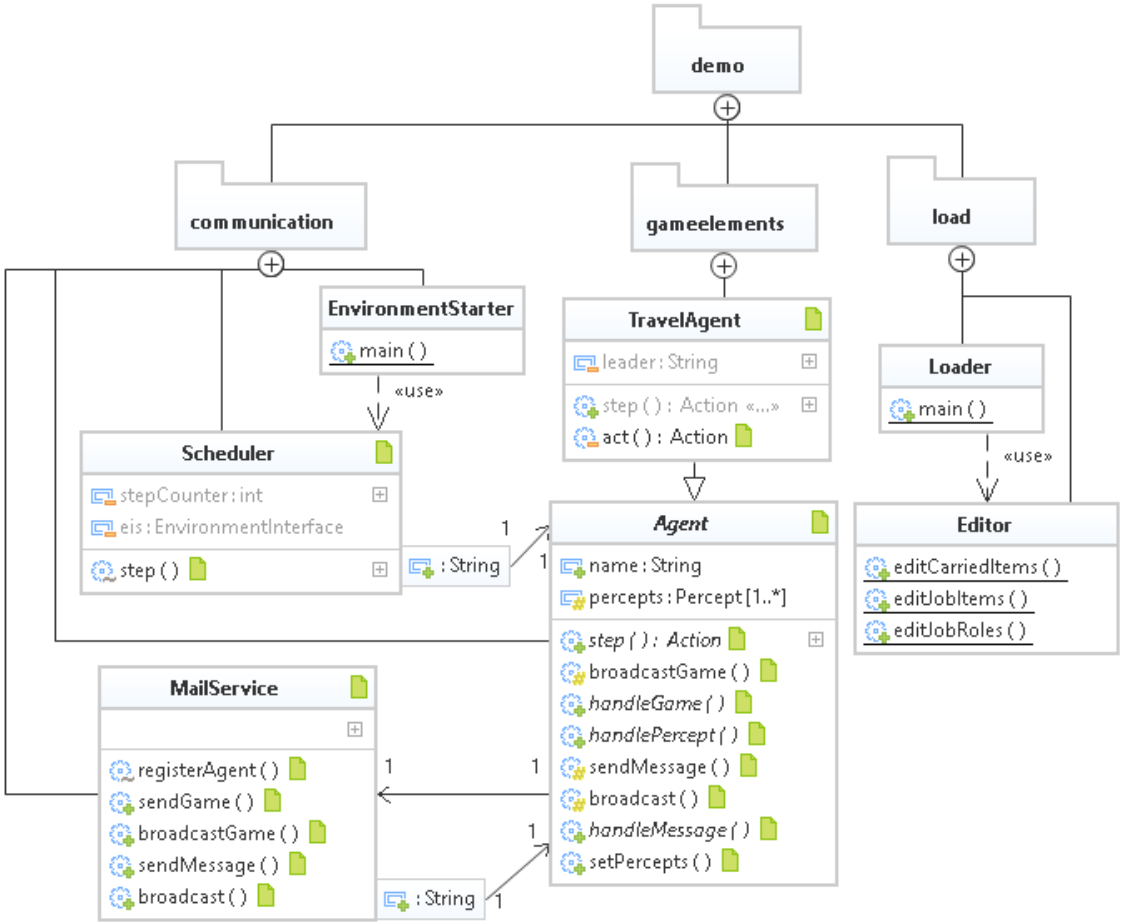
Aplikácia sa skladá z dvoch častí. Prvou je hra, ktorá iniciuje komunikáciu so serverom a sprostredkuje výmenu informácií medzi agentmi a serverom MASSim. Agenti su riadení na hernej ploche a zbierajú informácie potrebné pre výpočet genetického algoritmu. V druhej časti dema sú zbierané informácie predspracované a transformované do formy, ktorú požaduje rozhranie genetického algoritmu. Po tejto transformácii je zahájený samotný výpočet GA, ktorého popis je predmetom druhej časti tejto kapitoly.

5.1.1 Zber dát z MASSim

Celá komunikácia so serverom MASSim je zabezpečená balíkom `demo.communication`. Samotný proces komunikácie (zberania dát z prostredia) zahajuje vstupný bod triedy `EnvironmentStarter`, ktorý vytvorí objekt typu `EnvironmentInterface` reprezentujúci rozhranie komunikácie agentov a serveru MASSim. Registrovanie agentov k rozhraniu a asociáciu entít s agentmi vykonáva po registrácii plánovač `Scheduler`. Ten je po vykonaní potrebných úkonov pripojenia k rozhraniu riadený opäť vstupnými bodom, ktorý ho používa na cyklickú komunikáciu agentov a servera cez vytvorené rozhranie. V každom kroku sú zo servera pomocou metódy `getAllPercepts()` získané informácie o prostredí a prijímajú ich konkrétni agenti, pre ktorých boli určené. Plánovač po predaní vnemov agentom čaká na ich odpoveď vo forme akcie (bližšie popísané v 5.1.2), ktorú chcú vykonať. Táto akcia je po prijatí odoslaná na server. Celý proces je od prijatia vnemov zo servera opakovaný po určitý počet krokov. Kód zabezpečujúci komunikáciu so serverom bol prevzatý zo zdroja [3] a upravený podľa potreby tohto projektu.

Ak bol vykonaný potrebný počet simulačných krokov (počet krokov sa líši od parametrov servera, v experimentoch bolo použitých 15 a 20 krokov - za tento čas agenti stihnú objaviť dostatočný počet uzlov v daných scenároch hry) agenti sú spolu s ich informáciami o prostredí uložení do súboru.

Druhá časť dema z balíka `demo.load` začína načítaním uložených dát zo súboru do pamäte objektom triedy `Loader`. V pamäti sú upravené predmety a role potrebné pre vykonanie úloh (aby boli vlastnosti úloh rôznorodé, a teda aby algoritmus generoval zaujímavejšie riešenia). Tiež sú priradené predmety agentom, nakoľko funkcia ťaženia predmetov z uzlov nebola programovaná. Tieto úpravy vykonáva `Editor`, ktorý dané úpravy vykonáva vopred známym spôsobom. Odstránené/pridané predmety a role nie sú náhodne vybrané, aby bol scenár hry stále konzistentný. To uľahčuje ladenie programu a porovnávanie výsledkov z rôznych behov programu. Takto predspracované dáta sú predané metóde výpočtu GA, ktorý má na starosti trieda Evolution (5.2.3).



Obr. 5.1: Diagram tried zobrazujúci elementy zodpovedné za komunikáciu a predspracovanie vstupov GA.

5.1.2 Činnosť agentov

Komunikačná činnosť agentov je definovaná v triede `Agent`, ktorá využíva `MailService` ako schránku na preposielanie správ medzi agentmi. Obe triedy sú prevzaté z GitHub-u MASSim ([3]). V oboch triedach boli vykonané zmeny, ktoré rozšírili funkcionalitu o možnosť odoslania objektu typu `Game` lídrom ku všetkým agentom tímu pomocou metódy `broadcastGame()`. Agenti túto správu prijímajú metódou `handleGame()`, ktorá dokáže uložiť lídrovu referenciu hry lokálne do svojho priestoru.

Agenti pohybujúci sa po hernej ploche sú reprezentovaní triedou `TravelAgent`, ktorá pre možnosť komunikácie medzi agentmi rozširuje vyššie spomenutú triedu `Agent`. Plánovač riadi agentov pomocou metódy `step()`. V nej je pri jej počiatočnej invokácii u prvého agenta odoslaná správa cez broadcast o tom, že je lídrom skupiny a vytvára teda štruktúru s informáciami o hre.

V každom simulačnom kroku agent po prijatí vnemov zo serveru cez plánovač v rovnakej metóde pracuje na spracovaní a uložení vnemov do vnútorných štruktúr. Vnemy sú objekty triedy `Percept` a skladajú sa z názvu vnemu (job - úloha, charge - stav batérie atď.) a parametrov. Pokiaľ sa jedná o vnem definujúci stav prostredia, ako informácie o predmetoch v hre (`Item`), zadaných úlohách(`Job`), rozmeroch hracej plochy a nabíjacích staniciach, reaguje na ne len líder skupiny, ktorý ich zapisuje do objektu hry. Ak sa jedná o vnemy týkajúce sa individuálnych agentov (zručnosť, stav batérie...), sú spracované a uložené agentmi, pre ktorých sú určené. Pokiaľ agent našiel uzol, odosiela lídrovi cez schránku správu, ktorá obsahuje údaje o surovine, ktorú je možno v uzle ťažiť a tiež jeho polohu. Líder záznam o uzle vo forme inštancie `ResourceNode` uloží do štruktúry hry. Všetky triedy zastupujúce prvok hry (agent, úloha, predmet, hra, uzol) sa nachádzajú v balíku `demo.gameelements` a implementujú rozhrania z časti 5.2.1, ktoré definujú ich povinné metódy využívané pri výpočte GA.

Po spracovaní vnemov je potrebné previesť predmety vyžadované úlohami na suroviny, z ktorých sú zložené, keďže genetický algoritmus pracuje len na úrovni surovín. Predmety vyšších úrovní sú len rekurzívne prechádzané smerom nadol, kým sa rekurzia nedostane k predmetom na najnižšej úrovni. Pri rekurzívnom zostupe stromovej štruktúry predmetov sú tiež pre každý predmet vyššej úrovne zaznamenávané role potrebné pre zloženie týchto predmetov. Suroviny a role sú následne uložené ako potrebné zložky pre úspešné splnenie danej úlohy.

Keď sú uložené všetky potrebné dáta, metóda `step()` musí predať plánovačovi akciu `Action`, ktorú má poslať na server. V prvom simulačnom kroku sú vygenerované náhodné súradnice miesta na mape, na ktoré bude agent cieliť, kým naň nepríde. Odosielaná akcia má v tomto prípade parametre `goto` a súradnice daného miesta. Ak stav jeho batérie ale klesne pod 30% (stav batérie a jej kapacita bola zistená z vnemov zo servera), agent príde na najbližšiu nabíjaciu stanicu a batériu bude dobíjať akciou s parametrom `charge`. Pri stave úplného nabitia sa opäť začnú generovať náhodné pozície na mape, ktoré bude agent nasledovať. Dôležité komunikačné elementy a ich vlastnosti sú zobrazené v diagrame tried 5.1.

5.2 Genetický algoritmus

Nasleduje popis implementačných detailov genetického algoritmu. Predstavené sú vstupy algoritmu, ich formát a význam. Ďalej je popísaná forma reprezentácie riešenia a počítania fitness funkcie. V posledných častiach sú analyzované témy postupu výpočtu, štruktúry,

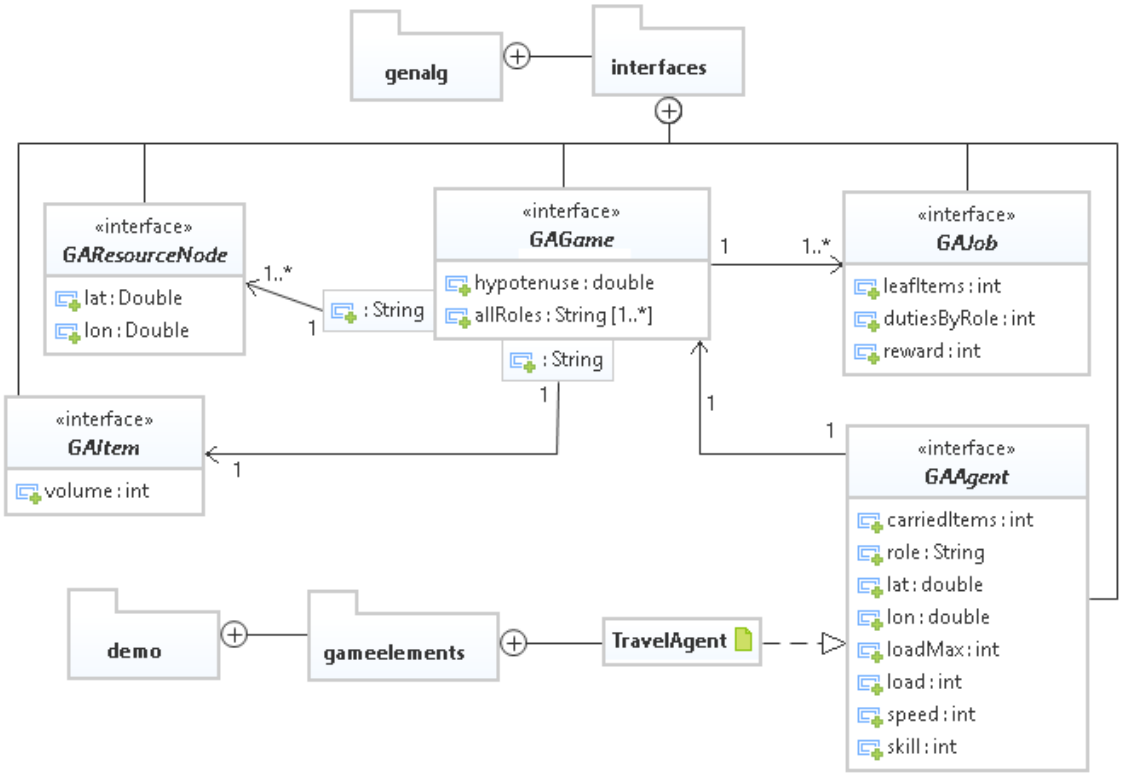
ktoré sú na výpočet využívané, diagram tried zobrazujúci vzťahy medzi elementami programu a spôsob integrovania knižnice do JADE projektu.

5.2.1 Formát vstupov

Pred započatím výpočtu je nutné prispôsobiť dáta forme kompatibilnej so vstupom GA. Pre tento účel boli v balíku `genalg.interfaces` vytvorené a implementované rozhrania. Všetky obsiahnuté rozhrania nejakým spôsobom reprezentujú prvok hry MAPC 2018 a väčšina z nich definuje formát informácií, ktoré budú dôležité aj pri programovaní hernej logiky. Žiadne kalkulácie navyše kvôli zaisteniu potrebných dát pre výpočet GA nebudú nutné. Rozhrania v podstate definujú, ktoré informácie a v akom formáte majú byť, aby boli v súlade so vstupom algoritmu.

Rozhranie `GAAgent` definuje statické aj dynamické vlastnosti agenta. Z dynamických informácií musí agent držať referenciu na hru, informácie o jeho predmetoch a ich počte, jeho aktuálnej polohe, objeme nákladu, rýchlosti a zručnosti. Tieto vlastnosti sú dynamickými, lebo sa v čase menia (rýchlosť a zručnosť sa menia vylepšovaním v hre). Medzi statické vlastnosti patrí rola agenta, ktorá bude potrebná pri počítaní penalizácie za priradené role k úlohe a maximálna kapacita, ktorá je taktiež v čase nemenná.

GAGame vyjadruje všeobecné informácie o hre, nezávislé na agentoch. Sú tu definované úlohy, ku ktorým je plánované priradiť agentov pri ďalšom spustení evolúcie, údaje o nájdených uzloch zásob od všetkých agentov daného tímu, role a predmety, ktoré v hre vystupujú a tiež diagonála hracej plochy pre účely výpočtu vzdialeností agentov od uzlov.



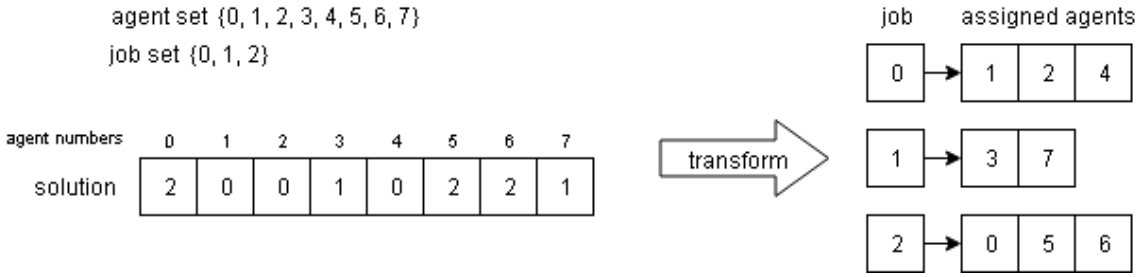
Obr. 5.2: Diagram tried zobrazujúci nutne implementovateľné rozhrania vstupov GA.

Posledné 3 rozhrania `GAItem`, `GAJob` a `GAResourceNode` reprezentujú podľa poradia predmet hry, zadanú úlohu a uzol s predmetmi. Predmet poskytuje len informáciu o svojom objeme pre výpočet potrebnej kapacity agentov pre úlohu. Úloha musí vedieť poskytnúť výšku odmeny za jej splnenie, typ a počet vyžadovaných základných predmetov a pre každý predmet definuje, ktorý musí každá rola agenta vyrobiť. Vzťahy medzi rozhraniami a časť ich implementácie do dema zobrazuje diagram tried 5.2.

5.2.2 Chromozóm

Jedinec je reprezentovaný triedou `Chromosome`, ktorej hlavnými vlastnosťami sú hodnota fitness funkcie daného chromozómu a jeho riešenie. Riešenie je dané polom typu `Integer`, kde index poľa určuje číslo agenta (kvôli tomu musia byť agenti usporiadaní v štruktúre zachovávajúcej poradie). Hodnota na tomto indexe určuje číslo úlohy, ku ktorej je agent priradený.

Metódou `calculateFitness()` je po vytvorení objektu vypočítaná hodnota fitness funkcie tohto jedinca. Najprv je riešenie prevedené z poľa do štruktúry `HashMap`, ktorá zaručí konštantnú časovú zložitosť `O(1)` namiesto lineárnej `O(n)` pri prístupe k agentom priradeným k určitej úlohe (obrázok 5.3).



Obr. 5.3: Transformácia riešenia z jednorozmerného poľa do reprezentácie `HashMap`.

Pre každú úlohu samostatne sú následne vypočítané penalizácie presne ako bolo popísané v sekcii návrhu fitness funkcie 4.2.3. Prejdením poľa agentov priradených k aktuálne spracúvanej úlohe sú prepočítané dáta, ktoré sú špecifické pre dané riešenie - súčet zručnosti, nákladu, rýchlosti a kapacity. Tiež sú od predmetov vyžadovaných touto úlohou odpočítané predmety vlastnené priradenými agentmi. Tým je zistený počet stále potrebných, ale aj prebytočných predmetov. Na základe týchto informácií sú vypočítané čiastkové penalizácie, a na konci spravený podiel sumy odmien a sumy penalizácií pre všetky úlohy.

5.2.3 Výpočet riešenia

Celá kódová časť evolučného algoritmu (triedy reprezentujúce chromozóm, kríženie atď.) sa nachádza v balíku `genalg`. Pred začatím výpočtu je možné pomocou statických setterov polí v `Evolution` nastaviť parametre výpočtu (veľkosť populácie, počet elít, typ kríženia, a pod.). Prednastavené hodnoty boli určené na základe experimentov.

Výpočet je možné spustiť invokovaním statickej metódy `calculate()`, ktorá ako parameter prijíma zoznam agentov. Tento zoznam musí byť definovaný typom `List<GAAgent>` kvôli zachovaniu poradia agentov, keďže na výstupe algoritmu budú agenti číslovaní v poradí, v akom sú na vstupe.

Pred zahájením samotného evolučného cyklu sú vypočítané dáta, ktoré pravdepodobne nebude nutné počítať v každom simulačnom kroku hernej logiky, a teda budú zistené len pre účely genetického algoritmu. Medzi tieto dáta patria vzdialenosti agentov k všetkým známym uzlom, zoradenie agentov podľa rolí a pre každú rolu samostatne sa zistí informácia, pri koľkých procesoch tvorby predmetov musí byť prítomná v rámci všetkých plánovaných úloh.

Nasleduje vytvorenie počiatočnej postupnosti, ktorej typ je prednastavený na generátor heuristickej populácie `HeuristicSolutionGenerator`, avšak setterom evolučnej triedy môže byť zmenený na náhodné generovanie `RandomSolutionGenerator`. Pri náhodnom generovaní je pseudonáhodným generátorom vygenerované číslo indikujúce číslo úlohy pre každý index vektora riešenia. Tento proces je vykonaný pre každého člena počiatočnej populácie.

Heuristické generovanie je tiež založené na procesoch náhodných dejov, avšak pokiaľ je to možné, priradí určitú úlohu k agentovi, iba pokiaľ rolu daného agenta daná úloha stále potrebuje. Najprv je kalkulovaná mapa, ktorá priradí každej úlohe j pre každú rolu r číslo. Toto číslo indikuje ideálny počet agentov s rolou r priradených k úlohe j (bližší popis tejto štruktúry sa nachádza v návrhu populácie 4.2.4. Následne je pre každý index vektora riešenia vygenerované náhodné číslo úlohy. Ak agent určený indexom vektora vlastní rolu, ktorú náhodne vygenerovaná úloha stále potrebuje, je táto úloha priradená na index tohto agenta. Počítadlo tejto role pre danú úlohu je znížené o hodnotu 1. Ak je rola nepotrebná, je generovaná iná úloha a proces sa opakuje. Ak v zozname ešte negenerovaných úloh zostala už len jedna položka, je priradená na tento index a pokračuje sa nasledujúcim indexom.

Z prvotnej populácie je vybraný taký objekt jedinca, ktorý má najvyššiu fitness hodnotu. Ten je uložený ako doteraz najlepšie nájdené riešenie, ktorého majú postupne v priebehu výpočtu nahradzovať objekty s vyššími fitness hodnotami.

V každom cykle tvorby novej generácie je na začiatku počet elitných jedincov (implicitný alebo určený parametrom) priamo presunutý do novej populácie. Zo starej populácie sú vybraní rodičia k procesu kríženia metódou select() buď triedy `TournametSelection`, v prípade výberu na princípe turnaja, alebo `RouletteWheelSelection` pre výber na základe rulety. Táto metóda vráti jeden objekt typu `Chromosome`, ktorý reprezentuje jedného z rodičov pripusteného k procesu kríženia.

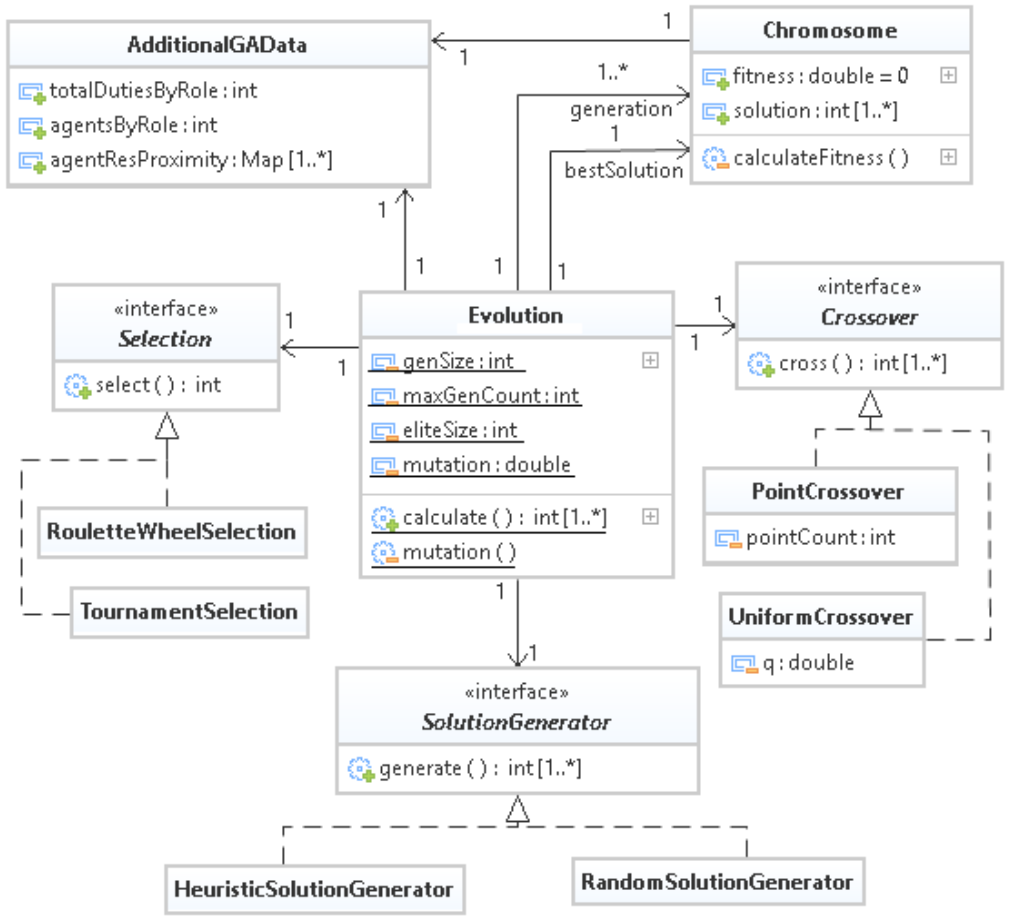
Rovnomerné kríženie je zastúpené triedou `UniformCrossover`, ktorého konštruktor berie ako parameter desatinné číslo q v intervale $[0, 1]$ určujúce, s akou pravdepodobnosťou bude pre potomka vybraný gén od rodiča s vyššou hodnotou fitness. Tento proces sa opakuje pre všetky indexy vektoru potomka.

Ak bolo zvolené 1 alebo viacbodové kríženie, metóda `cross()` v triede PointCrossover vygeneruje počet bodov kríženia určených parametrom konštruktora triedy. Tieto body označujú indexy, v ktorých sa rodičovské vektory rozdelia na súvislé bloky čísel, ktoré budú striedavo priradené k vektoru potomka na rovnaké indexy.

Po každom vytvorení takéhoto vektora je s istou pravdepodobnosťou vektor pripustený k procesu mutácie, v ktorom metóda `mutation()` vyberie náhodný index daného vektora a náhodné číslo úlohy. Ak sa úloha líši od aktuálnej úlohy na danom indexe, je hodnota nahradená práve vygenerovaným číslom úlohy.

Po vytvorení dostatočného počtu objektov `Chromosome` reprezentujúcich potomkov sú títo potomkovia pridaní do novej populácie, do ktorej boli v prvom kroku presunutí elitní jedinci z predošlej generácie. Z novej generácie je opäť vybraný jedinec s najvyššou fitness hodnotou. Ak je jeho hodnota vyššia ako hodnota doteraz nájdeného najkvalitnejšieho je-

dinca, tento chromozóm je považovaný za aktuálne najlepšie riešenie. Vzťahy medzi triedami algoritmu zobrazuje diagram tried 5.4.



Obr. 5.4: Diagram tried genetického algoritmu.

5.2.4 JADE integrácia

Pre jednoduché integrovanie do systému založeného na framework-u JADE bol GA programovaný v jazyku Java s JDK 8. Priložené pamäťové médium obsahuje vytvorenú *JAR* knižnicu, ktorú je nutné vložiť do cesty `classpath` projektu. Následne je možné využívať funkcie algoritmu spôsobom popísaným v sekcii 5.2.3. Návody na opätovné preloženie a spustenie demo aplikácie a knižnice s genetickým algoritmom sú obsiahnuté v súbore *README.txt* priloženom k projektu.

Kapitola 6

Experimenty

Podstatou experimentov bude zistiť vlastnosti algoritmu na základe zvolených parametrov GA. Medzi vlastnosti algoritmu patrí časová náročnosť a kvalita nájdených riešení. Testované budú rôzne druhy navrhnutých genetických operátorov ako aj parametre populácie a fitness funkcie. Testovanie prebehne vo viacerých experimentoch, pričom každý z experimentov pozostáva z 20 behov programu. Boli zaznamenávané priemerné a najlepšie hodnoty každého z behov.

Pokusy budú realizované na 2 scenároch. Úlohou scenárov je zaručiť rôznorodosť dát, na ktorých budú testované genetické operátory a parametre genetického algoritmu. Scenáre sú zložené z dát potrebných pre výpočet algoritmu, ktoré sú agentmi zozbierané zo servera MASSim, prípadne ďalej upravené podľa potreby (napríklad predmety a role potrebné pre splnenie úlohy). Experimenty budú teda vykonané na viacerých testovacích dátach, a teda budú môcť byť skúmané výsledky pre rôzne množiny agentov, úloh a predmetov.

6.1 Scenár 1

Dáta pre tento scenár boli vygenerované serverom, ktorý bol nakonfigurovaný referenčným súborom *SampleConfig-DivJobs.json* prebraným z projektu MASSim ([3]). Následne boli dáta upravené podľa popisu v sekcii 5.1.1. Charakteristikami tohto scenára je vyšší počet agentov oproti nízkemu počtu úloh. Bez nasadenia výsledkov genetického algoritmu do prostredia hry a jej vyhodnotenia je ťažké povedať či je pomer agentov, resp. ich rolí dostatočný na vykonanie daných úloh v rozumnom čase. Tieto závery budú vyhodnotené až pri programovaní samotnej logiky agentov.

Prvá množina údajov o scenári je definovaná v tabuľke A.1 v nasledujúcom poradí: identifikátory agentov, ktorí sa účastnia výpočtu genetického algoritmu, ich role, zručnosť, objem neseného nákladu, maximálna kapacitou, zemepisná šírka a dĺžka, predmety, ktoré daný agent nesie spolu s ich počtom v tvare predmet=počet.

V tabuľke A.2 je možné vidieť predmety (suroviny), pre ktoré boli po 20 simulačných krokoch komunikácie medzi serverom a agentmi títo agenti schopní nájsť uzly, a teda ich je možné ťažiť. Tabuľka v stĺpcoch podľa poradia obsahuje hodnoty vyjadrujúce názov predmetu, jeho objem, poradie uzlu zásob vzhľadom k jednému predmetu, zemepisná šírka a dĺžka uzlu pre daný predmet.

Scenár ďalej reprezentujú údaje o úlohách v tabuľke A.3, ku ktorým budú agenti priradení. Úloha je identifikovaná menom a obsahuje parametre ako odmena za jej vykonanie,

druh predmetu a jeho počet potrebný pre splnenie v tvare druh=počet a ďalej role spolu s počtom predmetov, na ktorých vytváraní sa má daná rola podieľať v tvare rola=počet.

Po preskúmaní tejto sady údajov je možné vyvodiť zopár dodatočných záverov o scenári. Predmetov i2 je ešte potreba vyťažiť v počte 72, predmetov i0 je navyše o 24 a i3 o 10. Predmety, ktorých všetci agenti dohromady vlastnia viac, ako dané úlohy vyžadujú, budú predstavovať výzvu pre genetický algoritmus (keďže prebytok predmetov nie je zas tak výrazný). Ten sa agentov s danými predmetmi musí pokúsiť priradiť k úlohám, ktoré ich potrebujú tak, aby ich všetky úlohy mali dostatok.

Experiment 1 Pri experimentovaní s fitness funkciou a jej parametrami ϵ, η, γ (4.9) nie je možné porovnávať jednotlivé nájdené riešenia pomocou výslednej fitness hodnoty, nakoľko tá bude v priebehu týchto testov závislá na týchto parametroch. Výsledky v experimentoch č. 1 a 2 budú teda vyhodnocované z pohľadu samotných predmetov a rolí agentov, ktorí sú k úlohám priradení a ich vzdialenosti k potrebným uzlom so zdrojmi.

V prvom kole testovania budú všetky parametre fitness funkcie nastavené na hodnotu 1, prehľad všetkých parametrov genetického algoritmu v tomto kole je možné vidieť v tabuľke 6.1. Vyjadrené sú: počet indivíduí v populácii, maximálny počet generácií, počet elít, pravdepodobnosť mutácie, princíp výberu rodičov, druh kríženia, princíp generovania počiatočnej populácie a parametre ϵ, η, γ (ich význam vysvetľuje podsekcia 4.2.3).

indiv.	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.1	tourn(5)	single point	random	1	1	1

Tabuľka 6.1: Parametre genetického algoritmu pre experiment č. 1.

Algoritmus bol s týmito parametrami spustený 20-krát, pričom najlepší výsledok z pohľadu fitness funkcie dosiahlo rozdelenie:

- **job0** ← A20, A23, A17, A3, A27
- **job2** ← A1, A10, A12, A26
- **job4** ← A21, A25, A24, A16, A18, A19, A7, A5, A8, A9, A28
- **job5** ← A22, A15, A2, A6, A4, A11, A14, A13

Priemerná fitness hodnota najlepších jedincov nájdených v každej z 20 pokusov bola 144.873 a väčšina z nich bola nájdená medzi 30.-100. iteráciou algoritmu, čo znamená príliš rýchlu konvergenciu. Hodnoty najlepšieho jedinca je možno vidieť v tabuľke 6.2. Záporná hodnota v bunke hovorí o počte prebytočných kusov konkrétnych predmetov alebo rolí. Kladná hodnota určuje naopak počet potrebných kusov, pričom u predmetov je pridaná informácia o priemernom čase príchodu k uzlom určených pre ťaženie konkrétnej suroviny v tvare počet/čas. Tabuľka tiež obsahuje informácie o jednotlivých penalizáciách a ich súčte v rámci jedného druhu penalizácie (role - rp, predmety - ip, čas - dp), resp. v rámci jednej úlohy. Vzorce pre výpočet penalizácií rp, ip a dp sa nachádzajú v popise návrhu fitness funkcie 4.2.3.

Pri sledovaní vývoja populácie bolo pozorované, že jedinci, ktorí k nejakej úlohe nemajú priradeného žiadneho agenta sa neobjavili ani v jednej generácii. Preto možno dedukovať, že pomer parametrov ϵ, η je nepodstatným faktorom, a teda bude zachovaný.

	job0	job2	job4	job5	
i0	-18	10/0.420	-9	-7	
i2	10/0.314	-1	42/0.331	21/0.398	
i1	22	14	0	34	
i4	26	14	39	0	
i3	-5	3/0.412	-11	3/0.436	
motorcycle	0	-0.420	0.049	0.370	
car	0.385	0	0.587	-0.972	
truck	-0.174	0.304	-0.130	0	
drone	0.385	-0.231	0	-0.154	**total pen.**
ip	8.919	10.043	6.663	10.261	35.886
rp	6.7	3.5	6.9	3.75	20.85
dp	0.314	0.832	0.331	0.834	2.311
total pen.	15.933	14.375	13.894	14.845	59.047
total fitness 155.401, iteration 30					

Tabuľka 6.2: Vlastnosti najlepšieho riešenia 1. kola experimentov.

Z výsledného riešenia možno pozorovať, že typom penalizácie s najvyšším dopadom na hodnotu fitness je penalizácia predmetov. Predmety i0 sú pomerne nevyvážené, nakoľko job0 žiadne nepotrebuje. Aj napriek tomu má navyše 18 kusov, na rozdiel od job2, ktorá stále potrebuje 10 kusov. Pri i1 a i4 je predmetov celkový nedostatok, a teda situácia je pozitívna, keďže žiadna z úloh ich nemá prevyšujúci počet. O vyváženej situácii sa dá hovoriť aj pri predmetoch i3.

Čo sa týka rolí, situácia je pomerne priaznivá. Je to z toho dôvodu, že žiadnej úlohe nechýba viac ako 1 kus z každej role. Nakoľko agentov nie je možné deliť na časti, desatinné čísla môžu byť mätúce, avšak keby sú výsledky penalizácie za role zaokrúhlené na celé čísla, nebolo by si možné všimnúť pár zaujímavých anomálií. V tomto prípade napríklad presunutie agenta s rolou car z job5 do job4 by rapídne znížilo penalizácia za role, keďže hodnota pre car v job4 by sa zmenila na -0.413, a v job5 na 0.028. Tým by rozdiel nedostatkov tejto role klesol z 0.587 na 0.028.

Experiment 2 Nasledovať bude pokus s upravenými parametrami ϵ a η, ktorého úlohou je docieliť nižší efekt rozloženia rolí medzi úlohy na fitness funkciu a naopak, dať prednosť riešeniam, ktorých vlastnené predmety sa najviac približujú predmetom potrebným pre splnenie úloh. Upravenie parametrov fitness funkcie je možno vidieť na v tabuľke 6.3.

indiv.	max g.	elite	mut	selection	crossover	initial	ϵ	η	σ
30	1000	3	0.1	tourn(5)	single point	random	0.5	0.5	1

Tabuľka 6.3: Parametre genetického algoritmu pre experiment č. 2.

Najlepšie ohodnotenie v 50 behoch algoritmu dosiahlo riešenie:

- **job0** ← A20, A18, A3, A26, A28
- **job2** ← A23, A22, A1, A8, A11
- **job4** ← A21, A24, A16, A15, A7, A5, A9, A12, A13, A27

- **job5** ← A25, A17, A2, A19, A6, A4, A10, A14

Charakteristiky tohto jedinca zobrazuje tabuľka 6.4. Porovnanie riešení z hľadiska výsledku fitness funkcie nie je validné, nakoľko boli v tomto pokuse zmenené parametre s priamym vplyvom na spôsob jej počítania.

Zlepšenie rozloženia predmetov je evidentné, nakoľko ip (item penalty) klesla z 35.886 na 29.623 (parametre segmentu fitness funkcie, ktorá počíta ip zmenené neboli, takže výsledky možno porovnať). Jedným z dôvodov takéhoto efektu je rovnomerné rozloženie predmetu i0. Na rozdiel od predošlého experimentu, pri takomto rozdelení už nie je vyžadovaný žiadnou úlohou. Situácia je o niečo horšia u i2, avšak rozdiel nie je tak citeľný ako u predošlého predmetu. Zdroj i4 zostáva nezmenený a mierne zlepšenie nastalo aj u i3, kde momentálne v job0 chýba už len 1 kus, oproti 6 spoločným kusom pre úlohy job2 a job5 z predošlého pokusu. Zlepšenie tiež nastalo vo vzdialenosti k uzlom pre ťaženie potrebných predmetov na 1.583. To je pochopiteľné, nakoľko potrebných typov predmetov je v jednotlivých úlohách menší, tým pádom nie je potrebné pripočítavať priemerný čas príchodu k uzlom pre nepotrebné predmety.

Pri rolách sa situácia očakávane zhoršila. Odlišne od predošlého experimentu, v tomto môžeme vidieť nedostatky hneď na viacerých miestach. Presun car z job2 do job4 by eliminoval penalizáciu znížením potrebných kusov z 0.587 na -0.174, truck z job5 do job4 tiež z 0.87 na -0.13.

Momentálne nie je možné povedať, ktoré z nastavení parametrov je lepšie a ktoré horšie, pretože každé z nich môže byť vhodné na iný typ nastavení samotnej hry. Konečné rozhodnutie o správnosti parametrov bude možno uzniesť až po naprogramovaní hernej logiky agentov a experimentoch v prostredí samotnej hry.

	job0	**job2**	**job4**	**job5**	
i0	-4	-7	-4	-9	
i2	12/0.384	-6	43/0.298	23/0.383	
i1	22	14	0	34	
i4	26	14	39	0	
i3	1/0.518	-5	-6	0	
motorcycle	0	-0.42	0.049	0.37	
car	0.385	-1	0.587	0.028	
truck	-0.174	0.304	0.87	-1	
drone	0.385	-0.231	0	-0.154	**total pen.**
ip	9.595	4.516	7.658	7.854	29.623
rp	3.35	1.75	8.45	2.025	15.575
dp	0.902	0	0.298	0.383	1.583
total pen.	13.847	6.266	16.406	10.262	46.781
total fitness 196.147, iteration 38					

Tabuľka 6.4: Vlastnosti najlepšieho riešenia 2. kola experimentov.

Experiment 3 V Nasledujúcich experimentoch bude použitý ruletový výber, ktorý bude následne porovnaný s turnajovým výberom s rôznymi veľkosťami účastníkov. Experimenty s číselnými parametrami fitness funkcie boli ukončené, preto je možné výsledné riešenia porovnávať na základe fitness hodnôt.

Prvým zo série pokusov bude algoritmus s turnajovým výberom, ktorý na rozdiel od predošlých experimentov bude obsahovať len 2 účastníkov turnaja. Z toho plynie menšia

šanca, že sa do súboja zapoja jedinci s vyššími hodnotami fitness, ale diverzita populácie bude stúpať.

indiv.	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.1	tourn(2)	single point	random	1	1	1

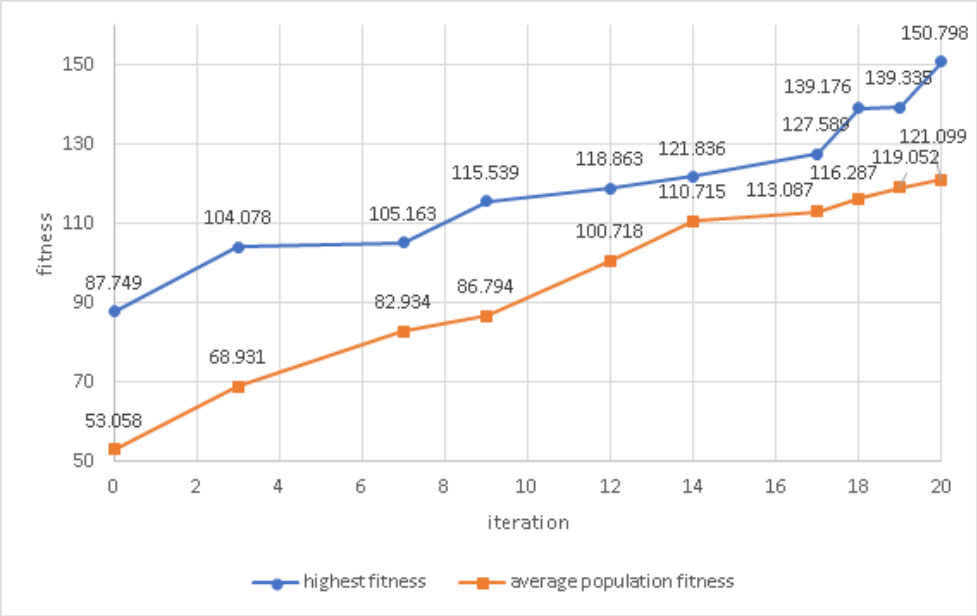
Tabuľka 6.5: Parametre genetického algoritmu pre experiment č. 3.

Najlepší jedinec predstavuje rozdelenie:

- **job0** ← A25, A2, A19, A14, A28
- **job2** ← A6, A7, A4, A26
- **job4** ← A21, A23, A22, A24, A16, A5, A8, A9, A12, A13, A27
- **job5** ← A20, A15, A18, A17, A3, A1, A10, A11

Najkvalitnejší jedinec z 20 behov mal fitness hodnotu 150.798, čo je skoro o 5 čísel nižšia hodnota ako v experimente 6.1 pri použití 5 účastníkov turnaja. Priemerná hodnota fitness najlepšieho riešenia každého behu bola taktiež o niečo nižšia a jej hodnota bola 142.8064. Všetky potrebné údaje o riešení sú zobrazené na obrázku 6.1.

Čo sa týka konvergencie k lokálnemu optimu, tá bola podstatne rýchlejšia, nakoľko najlepší nájdený jedinec bol známy už v 20. iterácii. Od počiatočného náhodného riešenia bol najlepší jedinec vylepšený celkovo 9-krát, pričom zvyšujúca sa fitness hodnota najelitnejšieho jedinca je odzrkadlená aj priemerom fitness hodnoty celej populácie, keďže tieto hodnoty stúpajú spoločne s postupom iterácií.



Obr. 6.1: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií s nastavením tourn(2).

Experiment 4 V nasledujúcom experimente bude ukázaný opäť turnajový výber s počtom súťažiacich 14, čo je pri populácii s 30 členmi takmer polovica. Keďže medzi sebou bude zápasiť viac jedincov, pravdepodobnosť, že medzi nimi budú elitní jedinci, rastie. Očakáva sa, že diverzita populácie bude teda menšia (gény od elitných jedincov budú súčasťou viacerých chromozómov), avšak elitné gény ostanú zachované vo väčšej časti populácie.

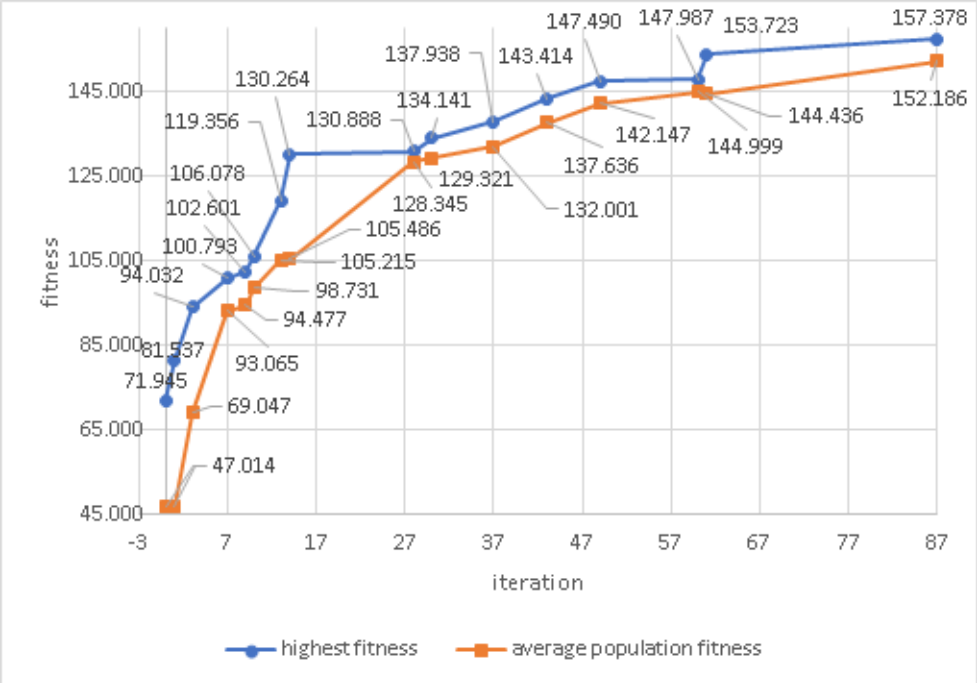
indiv.	max g.	elite	mut	selection	crossover	initial	ϵ	η	σ
30	1000	3	0.1	tourn(14)	single point	random	1	1	1

Tabuľka 6.6: Parametre genetického algoritmu pre experiment č. 4.

Najlepšie ohodnoteným riešením je rozdelenie:

- **job0** ← A20, A24, A16, A3, A27
- **job2** ← A7, A4, A5, A26
- **job4** ← A21, A23, A22, A25, A18, A19, A6, A8, A9, A12, A28
- **job5** ← A15, A17, A2, A1, A10, A11, A14, A13

S týmto nastavením bol prekonaný výsledok z predošlého experimentu, a dokonca aj z experimentu 6.1, a teda rekord najlepšieho jedinca je určený fitness hodnotou 157.378. Toto riešenie bolo nájdené po 87 iteráciách, v ktorých bol najlepší jedinec vylepšený celkovo 15-krát. Priemerná hodnota fitness najlepších jedincov z každého behu bola taktiež vylepšená na hodnotu 145.103 (obrázok 6.2).



Obr. 6.2: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií s nastavením tourn(14).

Výsledok je o to zaujímavejší, že algoritmus začínal s horším elitným jedincom počiatočnej populácie ako v predošlom prípade, keďže najvyššia fitness hodnota bola na úrovni 71.945. K tomuto faktu mohla prispieť vyššia priemerná fitness hodnota počiatočnej populácie, ktorá je vyššia približne o 6 bodov.

Experiment 5 V tomto experimente bude použitý ruletový výber rodičov (6.7), v ktorom bude šanca na výber rodiča k procesu kríženia závisieť na jeho hodnote fitness vzhľadom na súčet fitness hodnôt všetkých členov populácie. Predpokladá sa teda, že kvalitnejší jedinci budú mať vyššiu šancu predať svoje gény na viac indivíduí ako pri použití princípu turnaja, kde je proces výberu jedincov k turnaju čisto náhodný.

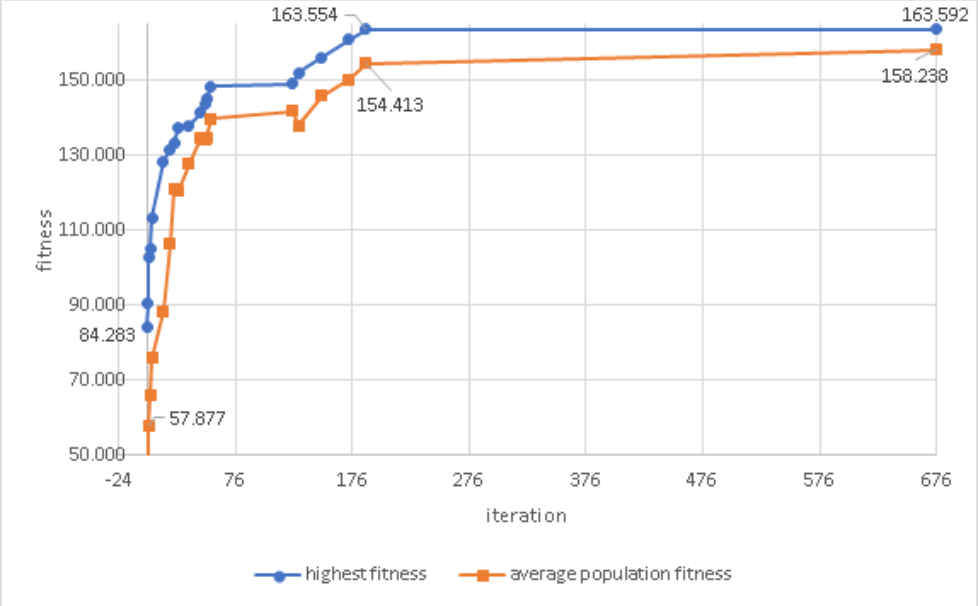
indiv.	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.1	roulette	single point	random	1	1	1

Tabuľka 6.7: Parametre genetického algoritmu pre experiment č. 5.

Rozdelenie agentov u najlepšieho jedinca:

- **job0** ← A20, A23, A25, A15, A1
- **job2** ← A3, A6, A12, A26
- **job4** ← A24, A16, A17, A19, A5, A8, A9, A11, A13, A27, A28
- **job5** ← A21, A22, A18, A2, A7, A4, A10, A14

Ruletový výber prekonal výsledky všetkých doterajších parametrov algoritmu. Priemer fitness hodnôt elitných jedincov z 20 behov algoritmu bol 155.507, čo je prekonanie doterajšieho rekordu o vyše 10 bodov. Najelitnejší chromozóm dosiahol hodnotu 163.592 v 676. iterácii (obrázok 6.3). Aj napriek tomu, že sa riešenie od iterácie 188 dlho nezlepšovalo, tento spôsob výberu rodičov teda eliminoval tvrdé stagnovanie riešenia po prvých pár iteráciách.



Obr. 6.3: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií s ruletovým výberom.

Treba brať do úvahy, že náhodné generovanie úvodnej populácie dokázalo vytvoriť elitného jedinca s vyššou fitness hodnotou ako v predošlom experimente a taktiež priemerná kvalita riešení v iniciálnej populácii je vyššia. Tým pádom algoritmus pracoval s istou výhodou oproti predošlému prípadu, to však nemení nič na fakte, že toto riešenie bol schopný výrazne vylepšiť.

Experiment 6 V ďalších pokusoch budú porovnávané výsledky algoritmu dosiahnuté rôznymi druhmi kríženia. V predošlých experimentoch bol použitý 1-bodový algoritmus, ktorý rozdeľoval rodičovské chromozómy na 2 časti. Skúšané budú viacbodové a rovnomerné kríženie.

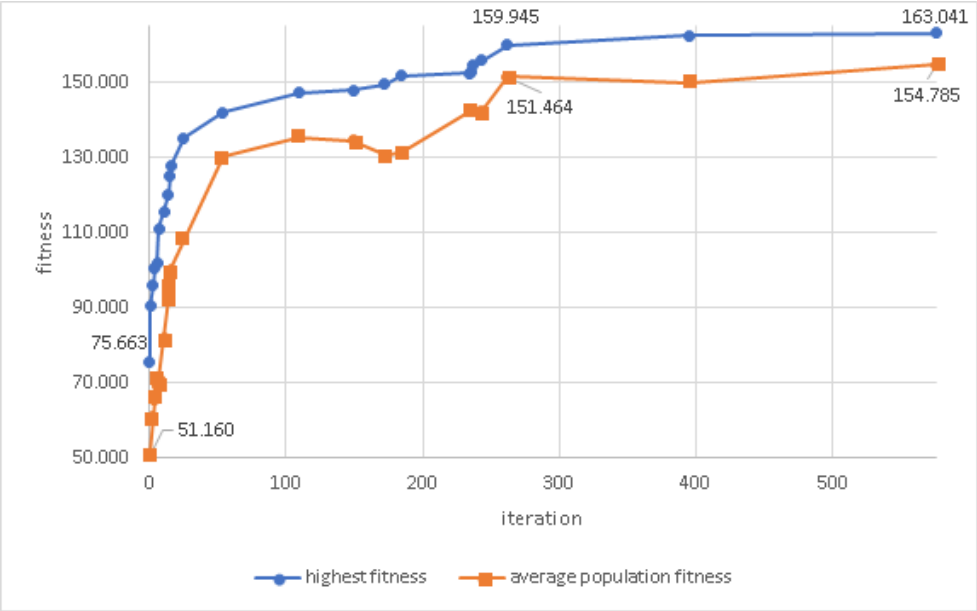
Je veľa miest, v ktorých možno chromozóm deliť. Preto boli testované viaceré počty bodov kríženia, z ktorých najlepšie výsledky vykazovalo 4-bodové kríženie. Výsledky sú zobrazené nižšie.

indiv.	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.1	roulette	4-point	random	1	1	1

Tabuľka 6.8: Parametre genetického algoritmu pre experiment č. 6.

Rozdelenie agentov v najlepšom chromozóme:

- **job0** ← A20, A25, A3, A14, A27
- **job2** ← A23, A2, A7, A12
- **job4** ← A21, A24, A16, A17, A19, A8, A9, A10, A11, A26, A28
- **job5** ← A22, A15, A18, A1, A6, A4, A5, A13



Obr. 6.4: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií so 4-bodovým krížením.

Ako je možno vidieť z obrázka 6.4, hodnoty nie sú až natoľko odlišné od výsledkov 1-bodového kríženia z predošlého pokusu. Top jedinec z počiatočnej populácie je síce nižšie ohodnotený, spolu s priemerom celej populácie, čiže algoritmus mal v tomto prípade horšie začiatočné predispozície, ale dokázal vytvoriť jedinca ohodnoteného 163.041 bodmi. To je približne rovnaká úroveň ako pri 1-bodovom variante kríženia. Podobný výsledok dosiahol aj v priemernej fitness hodnote najlepších jedincov zo všetkých 20 behoch programu, ktorá je na hodnote 156.423. V istom bode je možné vidieť aj istý pokles priemernej hodnoty celej populácie. To je ale pri generačnom modele tvorby novej populácie očakávaný efekt.

Dlhodobé vylepšovanie riešenia počas iterácií ostalo zachované vďaka ruletovému výberu, aj keď po 200 generáciách nastáva jasný úpad. Počiatočné najlepšie riešenie riešenie bolo vylepšené celkovo 22-krát.

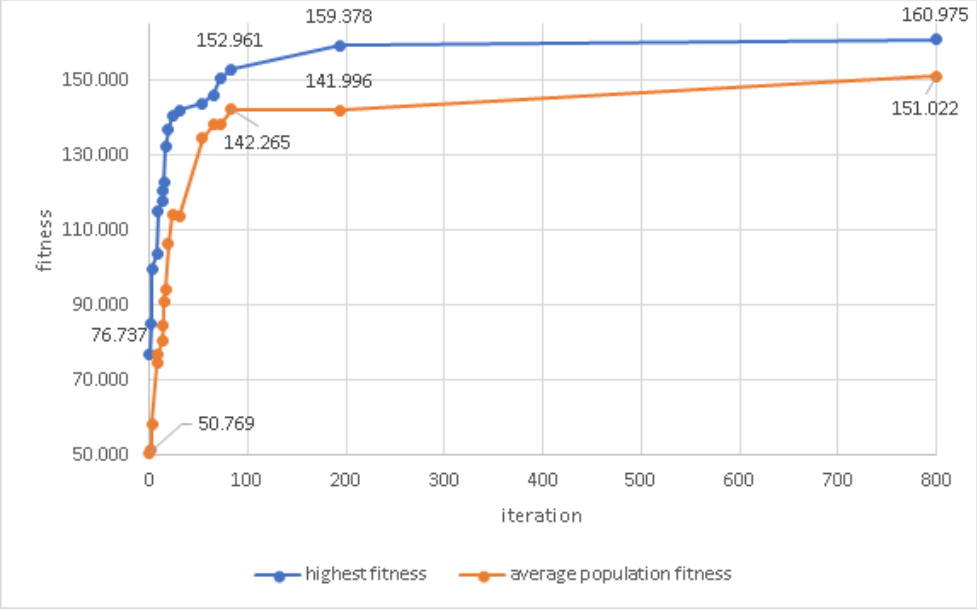
Experiment 7 Ďalším druhom kríženia, ktorý sa podrobí experimentom bude uniformné (rovnomerné). V tomto experimente bude parameter kríženia ponechaný na hodnote 0.5, čiže pravdepodobnosť získania génu od oboch rodičov pre potomka bude rovnaká.

indiv	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.1	roulette	uniform(0.5)	random	1	1	1

Tabuľka 6.9: Parametre genetického algoritmu pre experiment č. 7.

Rozdelenie agentov v najlepšom chromozóme:

- **job0** ← A20, A23, A25, A18, A1
- **job2** ← A6, A4, A12, A26
- **job4** ← A22, A24, A16, A17, A7, A8, A9, A10, A13, A27, A28
- **job5** ← A21, A15, A2, A3, A19, A5, A11, A14



Obr. 6.5: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií s rovnomerným križením s parametrom 0.5.

Z obrázku 6.5 možno vidieť, že pri približne rovnakej hodnote elitného jedinca z náhodnej počiatočnej populácie ako v predošlom experimente algoritmus nedokázal dosiahnuť lepšie výsledky. Elitný jedinec bol vylepšený 18-krát, pričom jeho poslednou hodnotou je 160.975. Pred finálnym zlepšením v generácii 800 nebolo riešenie zlepšené vyše 600 generácií. Priemer najkvalitnejších riešení zo všetkých behov bol 154.09, čo je tiež o niečo nižšia hodnota oproti minulému výsledku.

Experiment 8 Posledným experimentom s krížením v tejto sekcii bude upravený parameter rovnomerného kríženia z predošlého pokusu na hodnotu 0.7 (tento parameter dosiahol najlepšie hodnoty zo všetkých testovaných okrem parametru 0.5). To znamená, že pre každý rodičovský gén je pravdepodobnosť výberu tohto génu z lepšie ohodnoteného rodiča 70%.

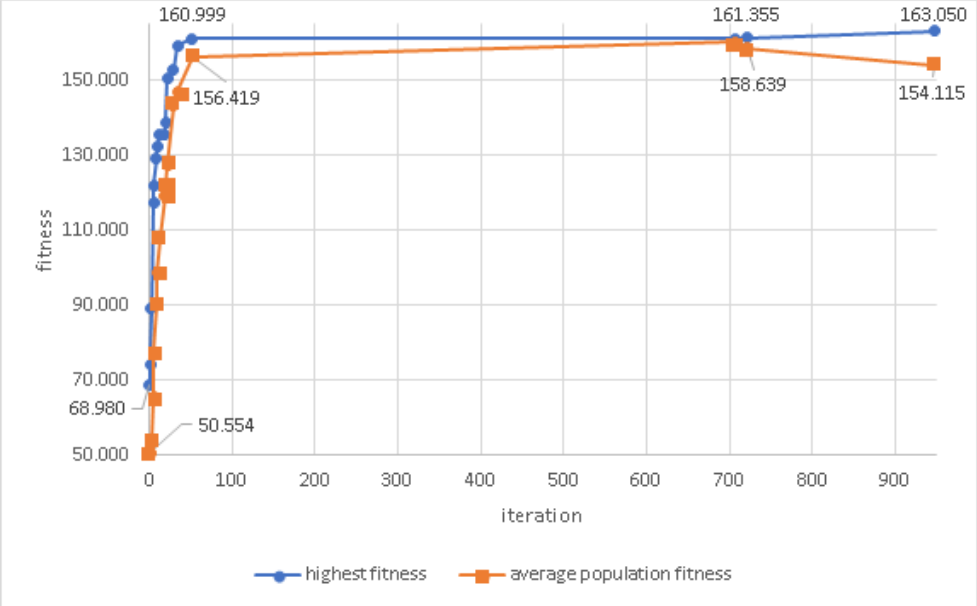
indiv.	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.1	roulette	uniform(0.7)	random	1	1	1

Tabuľka 6.10: Parametre genetického algoritmu pre experiment č. 8.

Rozdelenie agentov v najlepšom chromozóme:

- **job0** ← A20, A25, A24, A17, A3
- **job2** ← A23, A2, A7, A12
- **job4** ← A21, A22, A16, A15, A5, A8, A9, A10, A27, A26, A28
- **job5** ← A18, A19, A1, A6, A4, A11, A14, A13

Graf 6.6 zobrazuje najlepšieho jedinca spomedzi všetkých behov s hodnotou 163.050, nájdeného v generácii 948. Vylepšenie top jedinca prebehlo celkovo v 16. generáciách, pričom žiadna z nich nebola v intervale 53-705, v ktorom opäť nastalo veľké stagnovanie.



Obr. 6.6: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií s rovnomerným krížením s parametrom 0.7.

Počiatočná generácia mala takmer rovnaké vlastnosti ako v predošlom experimente. Priemer najlepších fitness hodnôt spomedzi behov programu bol oproti pôvodnému nastaveniu parametra približne o 4 čísla nižší s hodnotou 150.813.

Experiment 9 Posledná sada experimentov sa bude zaoberať parametrami priamo súvisiacimi s populáciou, ako sú tvorba počiatočnej populácie, pravdepodobnosť mutácie, veľkosť populácie, maximálny počet generácií a počet elitných jedincov.

Teraz bude algoritmus testovaný s doteraz najlepšie ohodnotenými parametrami s tým, že pri tvorbe počiatočnej populácie bude použitá heuristika popísaná v návrhu 4.2.4. Sledovaný bude opäť vývoj najlepšieho jedinca a vlastnosti jednotlivých generácií.

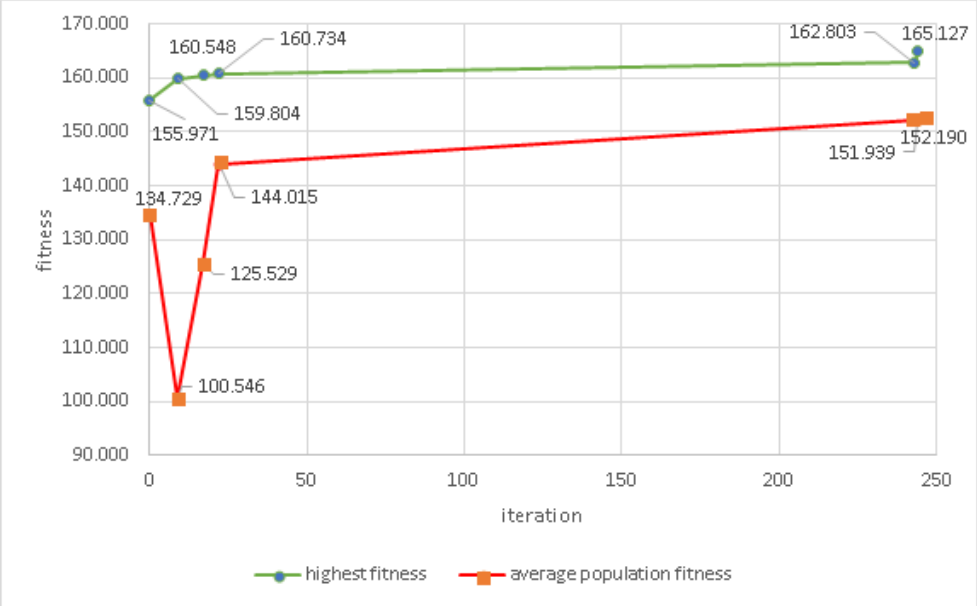
indiv.	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.1	roulette	4-point	heuristic	1	1	1

Tabuľka 6.11: Parametre genetického algoritmu pre experiment č. 9.

Rozdelenie agentov v najkvalitnejšom nájdenom chromozóme:

- **job0** ← A25, A15, A18, A3, A26
- **job2** ← A23, A2, A7, A12
- **job4** ← A21, A24, A16, A17, A19, A6, A8, A9, A10, A13, A27, A28
- **job5** ← A20, A22, A1, A4, A5, A11, A14

Z grafu 6.7 je možné vidieť, že použitie heuristiky pri tvorbe počiatočnej populácie výrazne znížilo počet vylepšení top jedinca, v tomto konkrétnom riešení na 5. Dôvodom je hlavne vysoká fitness hodnota najlepšieho potomka hneď v 1. generácii, ktorá dosahuje úroveň 155.971 čo je vyššie, ako finálne riešenia niektorých predošlých experimentov.



Obr. 6.7: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií s vytvorením počiatočnej populácie za použitia heuristiky.

Algoritmus v tomto prípade dokázal nájsť nové najvyššie riešenie už v 244 cykloch so stagnovaním od 22. generácie. Jeho fitness hodnota je na úrovni 165.127 bodov. Priemer fitness elitných jedincov z 20 behov algoritmu je 160.893, čo je v porovnaní s minulými pokusmi veľký skok vpred.

Keďže všetci jedinci počiatočnej generácie majú nízku penalizáciu za nedostatok rolí, v prvých pár generáciách je očakávaný pokles priemernej fitness hodnoty jedincov v populácii. To je spôsobené tým, že pri väčšine krížení rodičov z počiatočnej populácie bude porušené heuristické rozloženie týchto rolí medzi úlohy. Princíp je podobný ako pri vytvorení neprípustného riešenia znázorneného na obrázku 4.2.

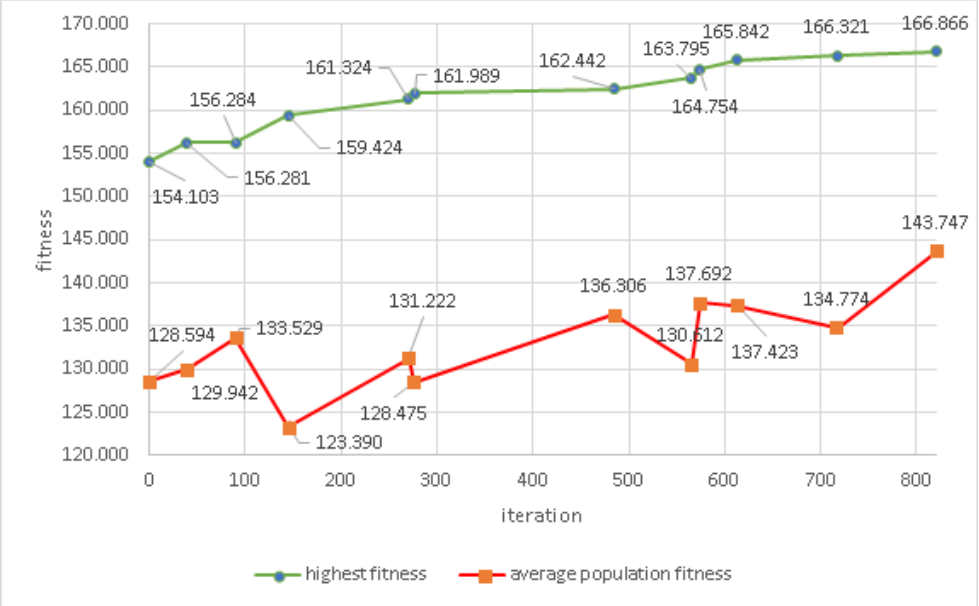
Experiment 10 Na rozdiel od obvyklých hodnôt pravdepodobnosti mutácie (1/n pri počte n chromozómov v populácii, resp. 0.001 pri n < 1000), bola podľa predbežných výsledkov nastavená štandardná hodnota 0.1 výrazne vyššia. Najlepšie výsledky však boli nájdené pri pravdepodobnosti mutácie až 50%, zvyšné hodnoty v tomto experimente boli ponechané tie, ktoré dosiahli najlepšie výsledky (6.12).

indiv.	**max g.**	**elite**	**mut**	**selection**	**crossover**	**initial**	ϵ	η	σ
30	1000	3	0.5	roulette	4-point	heuristic	1	1	1

Tabuľka 6.12: Parametre genetického algoritmu pre experiment č. 10.

Rozdelenie agentov v top nájdenom chromozóme:

- **job0** ← A20, A25, A18, A3, A26
- **job2** ← A23, A7, A4, A12
- **job4** ← A21, A24, A16, A17, A19, A6, A8, A9, A10, A27, A28
- **job5** ← A22, A15, A2, A1, A5, A11, A14, A13



Obr. 6.8: Vylepšovanie elitného jedinca a sumy fitness celej populácie v priebehu iterácií s pravdepodobnosťou mutácie 50%.

Pomocou spomenutej vysokej pravdepodobnosti mutácie bola nájdená nová rekordná hodnota fitness jedinca 166.866. Veľmi blízku hodnotu k optimu opäť ponúkla počiatočná heuristika, ktorej riešenie bolo bez väčšieho stagnovania upravované po 821. generáciu. Paradoxne je priemerná hodnota fitness generácií výrazne rozdielna od fitness hodnoty najlepšieho jedinca v prislúchajúcej populácii na rozdiel od predošlých experimentov. Vysoká pravdepodobnosť mutácie tiež dosiahla novú najvyššiu hodnotu priemernej fitness hodnoty najlepších jedincov z 20 behov programu. Tá má teraz hodnotu 164.698.

Experiment 11 V tomto experimente budú testované rôzne kombinácie parametrov populácie ako je veľkosť populácie, počet generácií a elít. Zvyšné parametre genetického algoritmu sú nastavené podľa najlepších výsledkov z predošlých experimentov.

mut	**selection**	**crossover**	**initial**	ϵ	η	σ
0.5	roulette	4-point	heuristic	1	1	1

Tabuľka 6.13: Parametre genetického algoritmu pre experiment č. 11.

Tabuľka 6.14 zobrazuje podľa poradia údaje: počet indivíduí v populácii, maximálny počet generácií, počet elitných jedincov, najkvalitnejšieho jedinca nájdeného spomedzi 20 behov, generáciu nájdeného top jedinca, priemernú fitness hodnotu najlepších jedincov všetkých behov a priemerný čas behu programu.

n	**indiv.**	**max g.**	**elite**	**fittest**	**fittest g.**	**fittest avg.**	**avg. time[ms]**
1	10	100	0	152.819	1	144.274	94
2	10	100	2	159.288	29	153.328	84
3	10	100	3	164.714	43	154.015	79
4	10	100	4	162.134	87	153.448	75
5	20	100	2	162.89	59	156.435	133
6	20	100	6	163.249	77	158.15	116
7	20	100	10	160.826	33	156.576	98
8	20	300	6	165.013	167	161.225	212
9	20	500	6	166.167	498	161.925	310
10	20	1000	6	166.866	916	163.64	561
11	30	500	10	164.982	423	162.319	422
12	30	1000	10	166.333	940	163.749	749
13	30	5000	10	166.866	3019	164.889	3316
14	50	1000	16	166.116	928	164.001	1296

Tabuľka 6.14: Výsledky testov viacerých parametrov genetického algoritmu v experimente č. 11.

V prvých 4 riadkoch bolo testované, ako vplýva počet elitných jedincov na výsledky algoritmu. Z nich je možno odvodiť, že ideálny počet elitných jedincov sa pohybuje na hranici 33% veľkosti populácie, nakoľko pri 3 elitách z 10 jedincov sú výsledky najlepšieho jedinca a priemeru týchto jedincov najlepšie. Pri 4 elitách sa výsledky začínajú zhoršovať. Pozitívum veľkého počtu elít je nižší čas behu algoritmu, avšak rozdiel je takmer zanedbateľný. Teória počtu elít bola potvrdená aj v testoch s 20 jedincami v populácii (5.-10. riadok), kde boli výsledky najlepšie pri 6 elitách. Po tomto 2-násobnom zvýšení jedincov populácie nebolo možné nájsť lepšieho jedinca, avšak priemerná fitness hodnota top chromozómov sa

pohybuje na podstatne vyššej úrovni. Táto výhoda však prichádza za cenu času, ktorá je takmer o 47% vyššia. S 20 jedincami v populácii a 1000 iteráciách začal algoritmus dávať konštantne výborné výsledky za cenu vyše pol sekundového výpočtu.

Po zvýšení veľkosti populácie na 30-50 neboli výsledky výpočtu až tak rozdielne s ohľadom na fitness. Čas výpočtu je však vysoko závislý na počte indivíduí a generácií. Preto je potreba otestovať algoritmus v reálnom scenári a vyhodnotiť výsledky dosiahnutého výkonu v prostredí hry.

Pre zaujímavosť je zobrazené rozdelenie agentov v najlepšom dosiahnutom riešení z týchto testov:

- **job0** ← A20, A25, A18, A3, A26
- **job2** ← A23, A7, A4, A12
- **job4** ← A21, A24, A16, A17, A19, A6, A8, A9, A10, A27, A28
- **job5** ← A22, A15, A2, A1, A5, A11, A14, A13

Počet prebytočných/potrebných rolí, predmetov, priemerných čas príchodu k ich uzlom a penalizácie z toho plynuje sú zobrazené v tabuľke 6.15.

	job0	**job2**	**job4**	**job5**	
i0	0	3/0.448	-20	-7	
i2	7/0.324	-1	45/0.350	21/0.377	
i1	22	14	0	34	
i4	26	14	39	0	
i3	1/0.482	-5	-6	0	
motorcycle	0	-0.42	0.049	0.37	
car	0.385	0	0.587	-0.972	
truck	-0.174	0.304	-0.13	0	
drone	0.385	-0.231	0	-0.154	**total pen.**
ip	8.378	7	6.977	9.804	32.159
rp	6.7	3.5	6.9	3.75	20.85
dp	0.806	0.448	0.35	0.377	1.981
total pen.	15.884	10.948	14.227	13.931	54.99
total fitness 166.866, iteration 916					

Tabuľka 6.15: Vlastnosti najlepšieho riešenia 11. kola experimentov.

6.2 Scenár 2

Dáta z aktuálneho scenára boli vygenerované serverom, ktorého konfigurácia sa nachádza v súbore SampleConfig-ManyItems.json prevzatom z GitHub-u projektu MASSim ([3]). V tomto scenári bude testovaný menší počet agentov aj úloh. Základných predmetov je oveľa viac. Preto bude zaujímavé sledovať, ako si s nimi genetický algoritmus poradí. Agenti tiež môžu niesť ľubovoľné množstvo druhov predmetov na rozdiel od scenára č. 1, kde bol počet obmedzený na 1.

Popis agentov, surovín a úloh obsahujú tabuľky podľa poradia B.1, B.2 a B.3, ktorých formát je rovnaký ako v predošlom scenári. Dáta boli získané po 15 simulačných krokoch hry a následne boli upravené.

Experiment 12 V tomto experimente bude vykonaný test výkonu algoritmu. Keďže sa v scenári č. 2 nachádza 2-krát menší počet agentov a o jednu úlohu menej, skúmané budú dosahované časy s rovnakými nastaveniami parametrov algoritmu ako v predošlom experimente (6.13).

V porovnaní so scenárom č. 1 sú časové hodnoty približne rovnaké (6.16). Pri istých hodnotách parametrov je nameraný čas v prospech druhého scenára, pri iných zase je o pár milisekúnd lepší algoritmus pre predošlý scenár. O badateľnejšom časovom rozdiele sa dá hovoriť pri väčšej veľkosti populácie a vyššom počte generácií. Napríklad v riadkoch 13 a 14 sú aktuálne väčšie o približne 700 a 500 milisekúnd, čo je paradoxné vzhľadom na menší počet agentov a úloh tohto scenára. Svoju úlohu pri výpočtoch zohráva spracovanie predmetov, ktorých je v tomto nastavení podstatne viac.

n	indiv.	max g.	elite	fittest	fittest g.	fittest avg.	avg. time[ms]
1	10	100	0	68.05	0	66.034	105
2	10	100	2	68.05	16	67.491	93
3	10	100	3	68.05	1	68.743	87
4	10	100	4	68.05	1	68.342	81
5	20	100	2	68.05	30	67.822	143
6	20	100	6	68.05	15	67.9	130
7	20	100	10	68.05	0	67.776	102
8	20	300	6	68.05	12	68.003	242
9	20	500	6	68.05	140	68.023	361
10	20	1000	6	68.05	183	68.044	653
11	30	500	10	68.05	43	68.037	490
12	30	1000	10	68.05	8	68.417	935
13	30	5000	10	68.05	9	68.05	4147
14	50	1000	15	68.05	12	68.05	1777

Tabuľka 6.16: Výsledky testov viacerých parametrov genetického algoritmu v experimente č. 12.

Počet možných kombinácií v tomto scenári je $3^{14} = 4,782,969$. To znamená, že je možné v rozumnom čase dostať optimálny výsledok použitím brute force algoritmu, ktorý ukázal, že najlepšie možné priradenie agentov má hodnotu fitness 68.05. Čo sa týka výsledkov fitness funkcie, aspoň jeden z dvadsiatich behov programu dokázalo pre každé nastavenie z tabuľky 6.16 nájsť optimálne riešenie, ktoré reprezentuje nasledujúce rozdelenie:

- **job1** ← A25, A2, A20, A16
- **job4** ← A24, A28, A3, A1, A4, A11, A12
- **job3** ← A23, A21, A22

Vlastnosti uvedeného rozdelenia, sú zobrazené v tabuľke 6.17. Všetkých predmetov je celkový nedostatok, s výnimkou i8 a i11, ktoré však majú nepriaznivé rozdelenie medzi agentov (jeden agent ich má príliš mnoho, preto ich nie je možné distribuovať medzi viaceré úlohy). Čo sa týka rolí je jasné, že akýkoľvek presun agentov s inými rolami medzi úlohami by spôsobil vyššiu penalizáciu rp. Vzdialenosť k uzlom tentokrát zohráva vyššiu úroveň ako v predošlom scenári, keďže jeho dp penalizácia tvorí vyše polovicu hodnoty penalizácie rp.

	job0	job4	job3	
i0	4/0.462	20/0.388	0	
i2	11/0.240	18/0.209	4/0.211	
i1	11	24	8	
i12	4	11	4	
i11	-8	-1	7/0.422	
i10	6/0.461	18/0.313	8/0.508	
i8	0	10/0.219	-30	
i7	5	0	5	
i9	3/0.298	-3	5/0.284	
i4	7/0.200	4/0.244	0	
i3	0	0	2/0.298	
i6	6/0.196	23/0.184	5/0.226	
i5	5/0.596	21/0.480	3/0.653	
motorcycle	0	-0.143	0.143	
car	0.378	0	-0.378	
truck	-0.094	0.415	-0.321	
drone	0.185	-0.185	0	**total pen.**
ip	16.259	46.571	12.654	75.484
rp	6	5.5	2	13.5
dp	2.452	2.038	2.601	7.091
total pen.	24.711	54.109	17.255	96.075

Tabuľka 6.17: Vlastnosti najlepšieho riešenia 13. kola experimentov.

Experiment 13 V poslednom experimente porovnáme výsledky algoritmov s princípom turnaja a princípom rulety z hľadiska výpočtového času. Do turnaja bude zapojených vždy 7 hráčov, nakoľko algoritmus pre tento pomer účastníkov a veľkosti populácie dosiahol najlepšie výsledky v predošlých pokusoch.

Obrázok 6.9 zobrazuje rozdiely v časovej náročnosti týchto dvoch princípov výberu. Jasné rozdiely nastávajú až pri vyššom počte generácií a väčšej veľkosti populácie. Turnajový výber mal v predošlých experimentoch horšie výsledky z podhľadu dosiahnutej priemernej fitness hodnoty. Preto je potreba zvážiť či ušetrené milisekundy stoja za marginálne horšie dosiahnuté riešenie.

6.3 Zhodnotenie

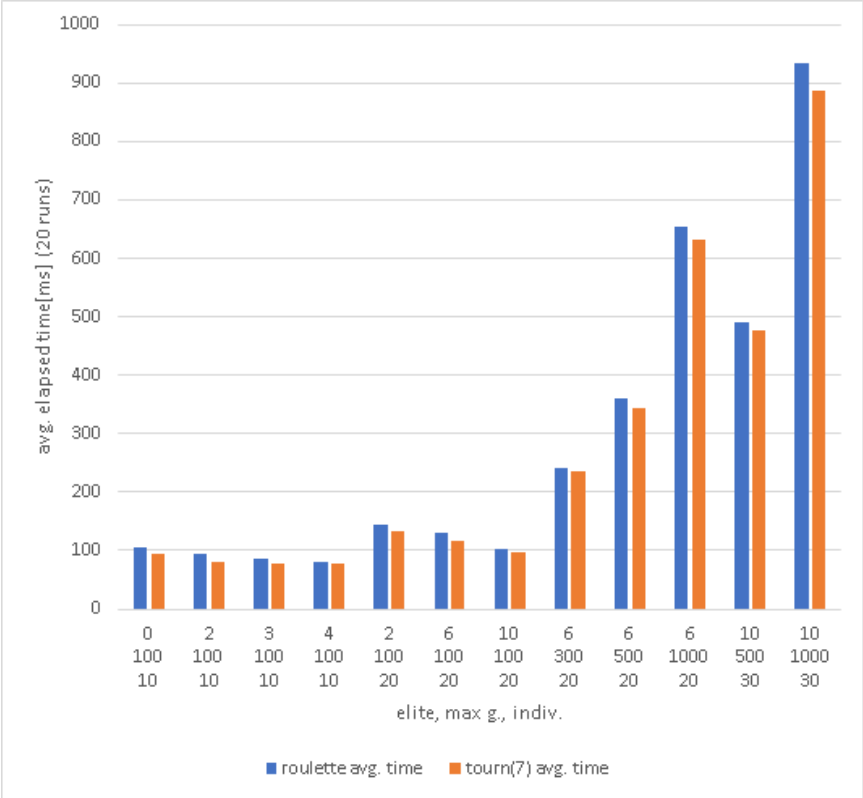
V celkovo 13-tich experimentoch bol GA testovaný na 2 scenároch, ktoré boli reprezentované dátami zo simulačného prostredia hry. Každý z experimentov bol vykonaný 20-krát, pričom boli sledované priemerné aj najlepšie hodnoty.

Prvý zo scenárov obsahoval 3 druhy surovín, 28 agentov a 4 úlohy, ku ktorým mal algoritmus na základe dostupných dát týchto agentov priradiť s ohľadom na efektivitu ich riešenia. Pre príliš veľké množstvo existujúcich možností riešenia je ťažké povedať, ako blízko bol algoritmus k nájdeniu optimálneho riešenia. Top jedinec z celého experimentálneho setu pre tento scenár dosiahol hodnotu fitness 166.866 bodov v 1 z 20 behov programu v experimente číslo 6.1 s parametrami zobrazenými v tabuľke 6.12. Na tomto výsledku,

ktorý výrazne vyčnieval z ostatných mala veľký podiel heuristika aplikovaná pri tvorbe počiatočnej populácie, ako aj vysoká pravdepodobnosť mutácie.

Čo sa týka času výpočtu, ten najviac závisí na číselných parametroch genetického algoritmu ako veľkosť populácie, počet generácií a elitných jedincov. Spolu s hodnotou týchto parametrov ale stúpala aj priemerná kvalita nájdených riešení. V referenčných nastaveniach servera je priestor agenta na odoslanie akcie na server 4 sekundy, čo je hodnota vyššia takmer o 700ms, ako priemerná hodnota nameraná pri 50 jedincoch v populácii, 15 elitách a 1000 generáciách. Pri týchto nastaveniach dával algoritmus konštantne solídne výsledky.

Druhý scenár disponoval 11 typmi predmetov, 14 agentmi a 3 úlohami. Vďaka menšiemu počtu existujúcich kombinácií bolo možné brute force algoritmom zistiť, že najlepšie riešenie má 68.05 fitness bodov. Tohto jedinca sa podarilo vytvoriť minimálne v 1 z 20 behov programu pri všetkých kombináciách 3 parametrov GA v experimente 6.2.



Obr. 6.9: Porovnanie priemerného času výpočtu s použitím rôznych gen. parametrov medzi ruletovým a turnajovým výberom so 7 účastníkmi v scenári č. 2.

Časovo algoritmus v tomto prípade zaostáva za scenárom č. 1. Toto je odôvodnené vysokým nárastom počtu predmetov, aj napriek 2-krát menšiemu počtu agentov. Pri menších hodnotách parametrov GA boli rozdiely marginálne, s ich nárastom sa však rozdiel výpočtových časov zvyšoval. Pri 30 členoch 5000 generácií s 15 elitami čas prevyšoval spomenuté 4 sekundy, v tomto prípade to však nehrá veľkú rolu, nakoľko pri iných parametroch algoritmus dáva akceptovateľné výsledky s oveľa nižšou časovou náročnosťou. Vplyv typu selekcie na čas kalkulovania má zanedbateľný efekt.

Kapitola 7

Záver

Táto práca sa zaoberá implementáciou genetického algoritmu a jeho aplikáciou na problém tvorby multiagentových koalícií. Algoritmus bol aplikovaný na konkrétny problém tvorby koalícií v hre MAPC 2018, v ktorej je základom úspechu spolupráca agentov v rámci tímu. Implementácia je rozdelená do dvoch častí - demo aplikácia a samotný GA. Demo aplikácia bola vytvorená na účely zbierania dát z prostredia hry, ktoré je simulované na serveri MASSim. Súčasťou dema sú ďalej agenti, ktorí so serverom komunikujú a ukladajú si informácie o sebe a svojom prostredí. Tieto informácie sú odoslané na vstup genetického algoritmu, ktorý sa na ich základe pokúša nájsť optimálne riešenie problému priradenia agentov do koalícií.

Hlavným bodom práce bol návrh fitness funkcie hodnotiacej kvalitu nájdených riešení. Zadané úlohy vyžadujú pre ich splnenie predmety, ktoré sú schopné vyrobiť agenti s určitými rolami. Na výrobu týchto predmetov sú vyžadované suroviny ťažené v zdrojových uzloch. Preto bola hodnotiaca funkcia rozdelená na 3 časti: role agentov priradených ku konkrétnej úlohe, suroviny vlastnené týmito agentmi a ich vzdialenosť od uzlov. Nakoľko nie je jasné, v akom pomere majú vplývať tieto 3 časti na výsledok fitness funkcie, boli k nim pridané nastaviteľné parametre ϵ, η a σ, ktoré možno korigovať vzhľadom na výsledky dosiahnuté v hrách po naprogramovaní hernej logiky. Na tvorbu počiatočnej populácie bola vytvorená heuristika, ktorá minimalizuje penalizáciu fitness funkcie za nedostatok určitých rolí priradených agentov. Zvyšné časti genetického algoritmu boli navrhnuté podľa známej teórie a sú taktiež predmetom experimentov.

V experimentoch sú testované 2 scenáre s rôznymi vlastnosťami a počtami agentov, úloh a existujúcich predmetov. Cieľom prvého scenára je otestovať vlastnosti algoritmu na veľkom počte agentov a úloh. Overená bola funkcionalita parametrov ϵ a η, keďže ich zníženie podporilo preferenciu zohľadnenia vlastnených predmetov oproti rolám. Čo sa týka selekcie, ruletový výber dosiahol výrazne lepšie výsledky oproti turnajovému s viacerými veľkosťami účastníkov turnaja. Medzi rôznymi druhmi kríženia nebolo možné vo výsledkoch badať veľký rozdiel, ale naopak kľúčovou bola vyššia pravdepodobnosť mutácie.

Druhý scenár bol navrhnutý tak, aby bolo jeho optimálne riešenie možno nájsť v relatívne krátkom čase brute force algoritmom, nakoľko obsahuje 2-krát menší počet agentov oproti predošlému scenáru a len 3 úlohy. GA v tomto prípade našiel top riešenie v každej z testovacích sád, ktoré pozostávali z 20 behov. Čas potrebný na výpočet bol v tomto scenári o niečo vyšší kvôli vyššiemu počtu existujúcich surovín, avšak stále v rámci tolerancie vzhľadom na zvolenú veľkosť populácie, počet generácií a elitných jedincov.

Čo sa týka prínosu práce, bola v nej ukázaná možnosť riešenia rozsiahleho problému ako je formovanie koalícií v hre MAPC 2018. Komplexnosť v tomto prípade spočívala vo veľkom

počte faktorov, ktoré môžu vplývať na kvalitu riešenia, a kvôli ktorým bola fitness funkcia rozdelená na viac segmentov. Priestor na ďalšiu prácu sa nachádza hlavne v oblasti hernej logiky, ktorá by implementovala činnosť agentov potrebnú pre úspech v kompetitívnom prostredí. Je potreba, aby si agenti zvládali deliť čiastkové procesy potrebné na splnenie úloh, stavali zariadenia na generovanie bodov do celkového skóre a využívali celé spektrum prostriedkov, ktoré hra ponúka.

Vo veci vylepšenia genetického algoritmu by zaujímavou témou na spracovanie bolo porovnanie aktuálneho a paralelného riešenia, v ktorom by agenti spolupracovali na vytvorení požadovanej koaličnej štruktúry. Predpokladá sa, že po výkonnostnej stránke by paralelné riešenie nebolo ideálne kvôli réžii spojenej s komunikáciou agentov, avšak pri rozdelení populácie na ostrovy by mohli byť zmiernené účinky konvergencie riešenia k lokálnemu optimu.

Literatúra

[1] Ahlbrecht, T.; Dix, J.; Fiekas, N.: EISMASSim. [Online; cit. 11.1.2019].
URL
https://github.com/agentcontest/massim/blob/master/docs/eismassim.md

[2] Ahlbrecht, T.; Dix, J.; Fiekas, N.: Multi-Agent Programming Contest. [Online; cit. 11.1.2019].
URL https://multiagentcontest.org/

[3] Ahlbrecht, T.; Dix, J.; Fiekas, N.: Multi-Agent Systems Simulation Platform. [Online; cit. 11.1.2019].
URL https://github.com/agentcontest/massim

[4] Bellifemine, F. L.; Caire, G.; Greenwood, D.: *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[5] Caire, G.: JADE TUTORIAL Jade Programming For Beginners. 2009, [Online; cit. 11.1.2019].
URL http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf

[6] Cantu-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 2000, ISBN 0792372212.

[7] Chalkiadakis, G.; Elkind, E.; Markakis, E.; aj.: Overlapping Coalition Formation. In *Internet and Network Economics*, editácia C. Papadimitriou; S. Zhang, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 307–321.

[8] Eiben, A. E.; Smith, J. E.: *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, druhé vydanie, 2015, ISBN 3662448734, 9783662448731.

[9] Haupt, R. L.; Haupt, S. E.: *Practical Genetic Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 1998, ISBN 047-1188735.

[10] Hindriks, K.: The Environment Interface Standard. [Online; cit. 11.1.2019].
URL https://github.com/eishub/eis

[11] Huget, M.-P.: The Foundation for Intelligent Physical Agents. [Online; cit. 11.1.2019].
URL http://www.fipa.org/

[12] Mehlenbacher, A.: Genetic Algorithm for Coalition formation. Department of Economics, University of Victoria, 2010, [Online; cit. 11.1.2019].

URL
http://web.uvic.ca/~amehlen/publications/CoalitionGeneticAlgorithms.pdf

[13] Mitchell, M.: *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998, ISBN 0262631857.

[14] Rahwan, T.; Ramchurn, S. D.; Jennings, N. R.; aj.: An Anytime Algorithm for Optimal Coalition Structure Generation. *J. Artif. Int. Res.*, ročník 34, č. 1, Apríl 2009: s. 521–567, ISSN 1076-9757.
URL http://dl.acm.org/citation.cfm?id=1622716.1622730

[15] Russel, S. J.; Norvig, P.: *Artificial Intelligence: a Modern Approach*. Upper Saddle River : Prentice Hall, tretie vydanie, 2010, ISBN 978-0-13-207148-2, 34-59 s.

[16] Sen, S.; Dutta, P. S.: Searching for Optimal Coalition Structures. In *4th International Conference on Multi-Agent Systems, ICMAS 2000, Boston, MA, USA, July 10-12, 2000*, 2000, s. 287–292, doi:10.1109/ICMAS.2000.858465.
URL https://doi.org/10.1109/ICMAS.2000.858465

[17] Shoham, Y.; Leyton-Brown, K.: *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. New York, NY, USA: Cambridge University Press, 2008, ISBN 0521899435.

[18] Simon, D.: *Evolutionary Optimization Algorithms*. John Wiley & Sons, Inc., prvé vydanie, 2013, ISBN 978-0-470-93741-9.

[19] Starkweather, T.; Mcdaniel, S.; Whitley, D.; aj.: A Comparison of Genetic Sequencing Operators. In *Proceedings of the fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991, s. 69–76.

[20] Wooldridge, M.: *An Introduction to MultiAgent Systems*. Wiley Publishing, druhé vydanie, 2009, ISBN 0470519460, 9780470519462.

[21] Wooldridge, M.; Jennings, N. R.: Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, ročník 10, 1995: s. 115–152.

[22] Yang, J.; Luo, Z.: Coalition Formation Mechanism in Multi-agent Systems Based on Genetic Algorithms. *Appl. Soft Comput.*, ročník 7, č. 2, Marec 2007: s. 561–568, ISSN 1568-4946, doi:10.1016/j.asoc.2006.04.004.
URL http://dx.doi.org/10.1016/j.asoc.2006.04.004

[23] Zbořil, F.: Slajdy k predmetu AGS - Koalice v MAS. FIT VUT v Brně, [Online; cit. 11.1.2019].
URL https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FAGS-IT%2Flectures%2FAGS17_KOALICE_f.pdf&cid=12046

[24] Zbořil, F.: Slajdy k predmetu AGS - Úvod do agentních a multiagentních systémů. FIT VUT v Brně, [Online; cit. 11.1.2019].
URL https://wis.fit.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FAGS-IT%2Flectures%2FAGS_17_01.pdf&cid=12046

[25] Zbořil, F.: *Plánování a komunikace v multiagentních systémech*. dissertation, Fakulta informačních technologií VUT v Brně, Fakulta informačních technologií VUT v Brně, october 2004.

Príloha A

Dáta pre scenár experimentov č. 1

n	name	role	skill	load	mLoad	lat	lon	carried
1	A21	car	8	7	150	48.85559	2.36993	i0=1
2	A20	car	8	126	150	48.85063	2.36718	i2=14
3	A23	truck	10	168	300	48.84728	2.36579	i3=24
4	A22	car	8	140	150	48.83727	2.37083	i0=20
5	A25	truck	10	45	300	48.86413	2.37057	i2=5
6	A24	truck	10	266	300	48.86734	2.36809	i3=38
7	A16	car	8	133	150	48.83261	2.37489	i0=19
8	A15	car	8	18	150	48.82664	2.38244	i2=2
9	A18	car	8	35	150	48.82842	2.37884	i3=5
10	A17	car	8	126	150	48.82827	2.37252	i0=18
11	A2	drone	1	9	25	48.83816	2.32175	i2=1
12	A3	drone	1	14	25	48.82682	2.33135	i3=2
13	A19	car	8	119	150	48.82827	2.37252	i0=17
14	A1	drone	1	9	25	48.83816	2.32175	i2=1
15	A6	motorcycle	6	42	70	48.82742	2.33517	i3=6
16	A7	motorcycle	6	49	70	48.84211	2.3676	i0=7
17	A4	drone	1	9	25	48.83816	2.32175	i2=1
18	A5	motorcycle	6	63	70	48.84211	2.3676	i3=9
19	A8	motorcycle	6	7	70	48.82664	2.38244	i0=1
20	A9	motorcycle	6	18	70	48.82664	2.38244	i2=2
21	A10	motorcycle	6	21	70	48.82586	2.34582	i3=3
22	A12	motorcycle	6	28	70	48.82715	2.36745	i0=4
23	A11	motorcycle	6	45	70	48.82616	2.35668	i2=5
24	A14	car	8	140	150	48.82616	2.35668	i3=20
25	A13	car	8	91	150	48.83727	2.37083	i0=13
26	A27	truck	10	18	300	48.87409	2.35692	i2=2
27	A26	truck	10	91	300	48.85063	2.36718	i3=13
28	A28	truck	10	28	300	48.84211	2.3676	i0=4

Tabuľka A.1: Identifikátory agentov spolu s ich parametrami danými scenárom č. 1.

resource	volume	n	lat	lon
i0	7	1	48.84182	2.30902
i2	9	1	48.87517	2.36395
i3	7	1	48.87721	2.337222
		2	48.89594	2.38247

Tabuľka A.2: Informácie o predmetoch a uzloch zásob, z ktorých je ich možné ťažiť pre scenár č. 1.

	job0	job2	job4	job5
reward	2061	1321	3412	2382
needed items	i2=26	i0=14	i0=40	i0=26
	i1=22	i1=14	i2=49	i2=30
	i4=26	i4=14	i4=39	i1=34
	i3=21	i3=19	i3=41	i3=29
needed roles	car=26	truck=15	car=50	car=33
	truck=21	motorcycle=16	truck=33	motorcycle=24
	drone=9	drone=5	motorcycle=41	drone=12

Tabuľka A.3: Úlohy, odmeny za ich splnenie, potrebné predmety a role pre scenár č. 1.

Príloha B

Dáta pre scenár experimentov č. 2

n	name	role	skill	load	mload	lat	lon	carried	
1	A23	truck	10	148	300	48.8695	2.35431	i10=2	i8=23
2	A25	truck	10	112	300	48.87776	2.35115	i5=4	i11=16
3	A24	truck	10	230	300	48.87374	2.3631	i11=18	i9=14
4	A28	truck	10	124	300	48.87374	2.3631	i3=12	i4=8
5	A2	drone	1	5	25	48.82567	2.35516	i6=1	-
6	A3	drone	1	5	25	48.82372	2.3006	i10=1	-
7	A1	drone	1	6	25	48.85226	2.37364	i8=1	-
8	A4	drone	1	0	25	48.82567	2.35516	-	-
9	A21	car	8	12	150	48.87383	2.35837	i6=1	i0=1
10	A20	car	8	74	150	48.86574	2.36933	i0=2	i10=12
11	A22	car	8	82	150	48.84581	2.3673	i8=11	i5=2
12	A16	car	8	38	150	48.86574	2.36933	i9=1	i3=4
13	A11	motorc.	6	43	70	48.84211	2.3676	i10=5	i8=3
14	A12	motorc.	6	34	70	48.84211	2.3676	i0=2	i10=4

Tabuľka B.1: Identifikátory agentov spolu s ich parametrami danými scenárom č. 2.

resource	volume	n	lat	lon
i6	5	1	48.86724	2.37933
		2	48.83989	2.37488
i0	7	1	48.83496	2.39215
i10	5	1	48.83634	2.39324
		2	48.82318	2.30232
i8	6	1	48.87498	2.3669
i5	8	1	48.82545	2.31892
i11	5	1	48.86243	2.39391
i9	10	1	48.83268	2.32016
		2	48.85724	2.33848
i3	7	1	48.86671	2.38171
i4	5	1	48.87256	2.37387
i2	9	1	48.84882	2.36206

Tabuľka B.2: Informácie o predmetoch a uzloch zásob, z ktorých je ich možné ťažiť pre scenár č. 2.

	job1	job4	job3
reward	1621	3646	1271
needed roles	car=22	truck=32	car=15
	truck=12	drone=38	truck=9
	drone=16	motorcycle=26	motorcycle=2
needed items	i7=5	i1=24	i8=4
	i4=7	i6=23	i7=5
	i10=18	i5=21	i10=10
	i6=7	i0=22	i6=6
	i2=11	i12=11	i2=4
	i1=11	i2=18	i1=8
	i9=4	i9=11	i9=5
	i3=4	i3=12	i3=2
	i11=8	i8=14	i11=7
	i12=4	i11=17	i12=4
	i5=9	i4=12	i5=5
	i0=6	i10=28	i0=1

Tabuľka B.3: Úlohy, odmeny za ich splnenie, potrebné predmety a role pre scenár č. 2.

Príloha C

Obsah CD

- *dp.pdf* - text diplomovej práce
- *tex/* - zdrojové kódy pre text DP
- *src/* - programová časť DP
- *src/demoproject* - zdrojové, konfiguračné a spustiteľné súbory demo aplikácie
- *src/gaproject* - zdrojové súbory a knižnica genetického algoritmu
- *lib/* - knižnice potrebné pre preklad a spustenie demo aplikácie
- *README.txt* - súbor s návodom na preklad a spustenie