BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

An OSGI-based testing and simulation framework for the study of reinforcement learning algorithms

Abstract

The goal of the dissertation is to create a dynamic testing and simulation environment for the evaluation of reinforcement learning algorithms on a remote server using large number of parallel running tests.

The simulation environment is based on the OSGI specification, which defines a dynamic, modularized component model for building complex applications. Previous attempts at creating such a testing environment for an example the "RL-GLUE" project had only partial success. Although it is capable to run and evaluate various tests written in different programming languages, it is already based in obsolete technology and the project was closed a few years ago.

The system I made is capable of evaluating RL algorithms written in JAVA through running various experiments. All this happens with the help of a simulation environment which runs on a remote access server. The client is initiating the test which are written based on a predefined standard, connects to the server which runs the test gets a proper feedback with the help of machine learning algorithms.

The server makes it possible to use a number of predefined simulation environments to perform tests that can be run on the server independently of each other.

The framework enables the monitoring of the experiments, based on a remote method invocation API and through a web interface.

This work is the result of my own activity. I have neither gave nor received unauthorized assistance on this work.

JULY 2015 GÁLL NORBERT

ADVISOR:
JAKAB HUNOR, PH.D, ASSISTANT PROFESSOR

BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

An OSGI-based testing and simulation framework for the study of reinforcement learning algorithms

SCIENTIFIC SUPERVISOR:
JAKAB HUNOR, PH.D, ASSISTANT PROFESSOR

STUDENT:
GÁLL NORBERT

JULY 2015

UNIVERSITATEA BABEŞ-BOLYAI, CLUJ–NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

-

CONDUCĂTOR ŞTIINŢIFIC:
LECTOR DR. TANÁR OKOS

ABSOLVENT:
GÁLL NORBERT

IULIE 2015

BABEŞ-BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Licensz-dolgozat

OSGI technológián alapuló tesztelési és szimulációs keretrendszer megerősítéses tanulási algoritmusok tanulmányozására

TÉMAVEZETŐ:

DR. JAKAB HUNOR, EGYETEMI ADJUNKTUS

SZERZŐ:

GÁLL NORBERT

2015 JÚLIUS

Tartalomjegyzék

1. fejezet

Bevezető

A dolgozat témája a megerősítéses tanulási algoritmusok futtatására és tesztelésére biztosítani egy egységes környezetet, mely egy távoli szerveren fut és bárhonnan elérhető.

A RoboRun projekt célja egy olyan dinamikus tesztelési és szimulációs környezet felépítése ahol megerősítéses tanulással kapcsolatos algoritmusok kipróbálására és tesztelésére van lehetőség egy távoli elérésű szerveren. E szimulációs környezet az OSGi[1] keretrendszerre épül, amely egy dinamikus, modularizált komponens modellt definiál komplex alkalmazások felépítésére. E környezet teljesen egységes és könnyen elérhető.

A tanulás egy nagyon fontos emberi tulajdonság, mely ott van az emberek mindennapjaiban. Hiszen az ember minden nap tanul valami újat, valami új tapasztalattal gazdagodik. A gépi tanulás is az emberi tanuláson alapszik, csak más jelentéssel bír. Mondhatjuk azt, hogy egy olyan folyamat, mely során a tanuló algoritmus paraméterei és belső állapotai változnak, amelyek később meghatároznak egy döntéshozatali stratégiát. Tehát bizonyos tapasztalat alapján, melyet a belső állapotok reprezentálnak, képes a számítógép döntéseket hozni. Ennek a legfőbb nehézsége abban rejlik, hogy véges számú lépés alatt meg kell tanítani a számítógépet arra, hogy egyre jobb döntéseket hozzon végtelen sok lépés közül.

A RoboRun projekt ötlete, nem számít újdonságnak a piacon. Számos hasonló projekt létezik, hasonló funkcionalitásokkal. A RoboRun projekt szempontjából az Rl-Glue[1] projektet érdemes kiemelni, hiszen ez szolgált a RoboRun projekt alapjául és számos funkcionalitását is felhasználtuk a projekt során. Az Rl-Glue projekt szerzői Brian Tanner és Adam White. E projekt eredetileg C++ ban íródott, viszont van teljesen Java- ban megírt változata is. Az Rl-Glue egy nyelv független környezet a megerősítéses tanulási algoritmusok tanulmányozására. Kétféle protokollt kínál, az úgymond külső- illetve belső módokat. A külső mód teljesen socketeken keresztül végzi a kommunikációt, míg a belső mód az teljesen lokálisan. Ez által a belső mód sokkal gyorsabb működést eredményez. A projekt 2010- ben lezárult, viszont teljesen nyílt forráskódú, mindenki számára elérhető és használható napjainkban is. Néhány hasonló projekt: RL Toolbox[2], ClSquare[3], PIQLE[4].

A RoboRun projekt az Rl-Glue projektet tovább gondolva és funkcionalitásait felhasználva, napjaink technológiáin alapszik. Egy dinamikus és egységes környezetet biztosít, mindezt úgy, hogy a rendszer teljesen moduláris, folyamatosan és könnyen változtatható, illetve bővíthető. Mindezek mellett nagyon

1. http://glue.rl-community.org/wiki/Main$_{P}age$
2. http://www.igi.tu-graz.ac.at/gerhard/ril-toolbox/general/overview.html
3. http://ml.informatik.uni-freiburg.de/research/clsquare
4. http://piqle.sourceforge.net/

jól megvalósítja a komponensek egymástól való elválasztását. A projekt teljesen Java alapokon nyugszik, felhasználva az OSGi platformot.

A fő változtatásokat főként a dinamikusság, a modularitás illetve a könnyed bővíthetőség foglalja magában. E mellett a RoboRun projekt egy nagy előnye, hogy nagyon kis erőforrásra van a felhasználónak szüksége még komolyabb tesztek futtatásánál is, hiszen a fő logikát, tehát az erőforrás igényes részeket mind a távoli elérésű szerver futtatja. Az eredmények összegezve megtekinthetőek minden teszt végén, illetve egy webes felületen keresztül is. A másik nagy előny, hogy a szerver bárhonnan elérhető internet kapcsolaton keresztül.

A dolgozat négy fejezetből áll. Az első fejezet röviden bemutatja a megerősítéses tanulást néhány világbeli példán keresztül, majd bemutatásra kerül néhány alap fogalom illetve ezek felhasználása a projekt során.

A második fejezet a RoboRun projekt alapjául szolgáló OSGi keretrendszert mutatja be, ismerteti ennek architektúráját, illetve azt, hogy miért esett e keretrendszerre a választás. Megemlít más eszközöket és technológiákat is melyek felhasználásra kerültek. E fejezet kitér arra is, hogy a projekt, hogyan használja a megerősítéses tanulási kísérletek lebonyolítására.

A harmadik fejezet részletezi a projekt által felhasznált technológiákat, majd tárgyalja a rendszer felépítését, ismerteti a szerver oldali architektúrát.

A negyedik fejezet a rendszer használatát és működését mutatja be egy példán keresztül. Részletesen kitér a rendszer által nyújtott funkcionalitásokra.

A RoboRun projekt sikeres elkészítéséért köszönet illeti a projektvezetőt.

2. fejezet

Alapfogalmak

Összefoglaló: *E fejezet célja bemutatni röviden a megerősítéses tanulást és ezen algoritmusok alap lépéseinek bemutatását, illetve a RoboRun projekttel kapcsolatos néhány alap fogalom bevezetése és ezek használata a projekt során.*

2.1. A megerősítéses tanulás

2.2. A megerősítéses tanulás algoritmusok kipróbálásának alap lépései

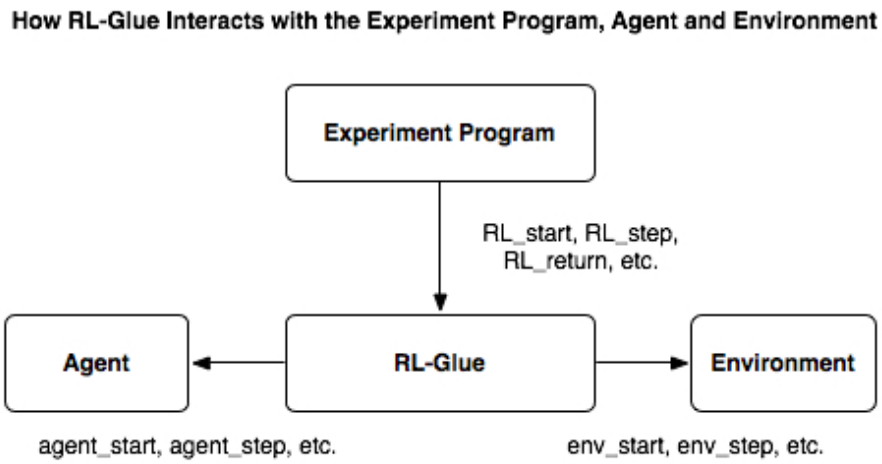
2.3. Alapfogalmak bevezetése

A projekt során a szerző az Rl-Glue projekt elveit követte, felhasználva annak funkcionalitásait. A három alap komponens mindkét projekt esetén az Agent[1], Environment[2] és az Experiment[3]. Ezen komponensek egymással való interakciója révén nyílik lehetőségünk futtatni illetve tesztelni a megerősítéses tanulási algoritmusokat.

Az Agent komponens valójában a tanulási algoritmus, amely kiszabja a feladatokat és az ezekre vonatkozó megszorításokat egy adott iterációra vonatkozóan. Az Agent jutalmat(reward) kap minden egyes iteráció után arra vonatkozóan, hogy a probléma megoldásának szempontjából mennyire volt hatékony a kiszabott feladat, illetve az erre vonatkozó megszorítás. Mivel nem tudhatja az algoritmus, hogy melyik a helyes módszer a probléma megoldására, ezért találgatnia kell. Időnként új cselekvéseket is kell próbálnia, majd az ezekből megszerzett tudást, ami esetünkben a jutalom, optimális módón felhasználnia a következő cselekvés meghatározására.

Az Environment komponens feladata végrehajtani az Agent komponens által meghatározott feladatokat és az ezekre vonatkozó megszorításokat az adott problémára. A végrehajtás során következtetéseket(observation) von le minden egyes állapotról. Majd ezen következtetések alapján jutalmakat(reward) határoz meg. Mivel bizonytalan a környezet, valami becslést kell alkalmaznia a jövőre nézve, így kezdetben, lehet, hogy egy jó lépésért nem kapjuk meg a megfelelő jutalmat. Viszont minél jobban megismerjük a környezetet, annál pontosabb lesz egy lépésért vagy lépés sorozatért járó jutalom. A jutalom egy számban fejezhető ki, amely egy adott intervallumban mozog. Ha az intervallum felső határához közelít a szám akkor pozitív visszajelzést kaptunk az adott lépés vagy lépes sorozat után, amennyiben az intervallum alsó határához közelít a szám, negatív a visszajelzés.

1. Agent - magyarul ügynök
2. Environment - magyarul környzet
3. Experiment - magyarul kísérlet



2.1. ábra. Rl-Glue projekt komponensek közti kommunikációja:
http://rl-glue.googlecode.com/svn/trunk/docs/html/index.html

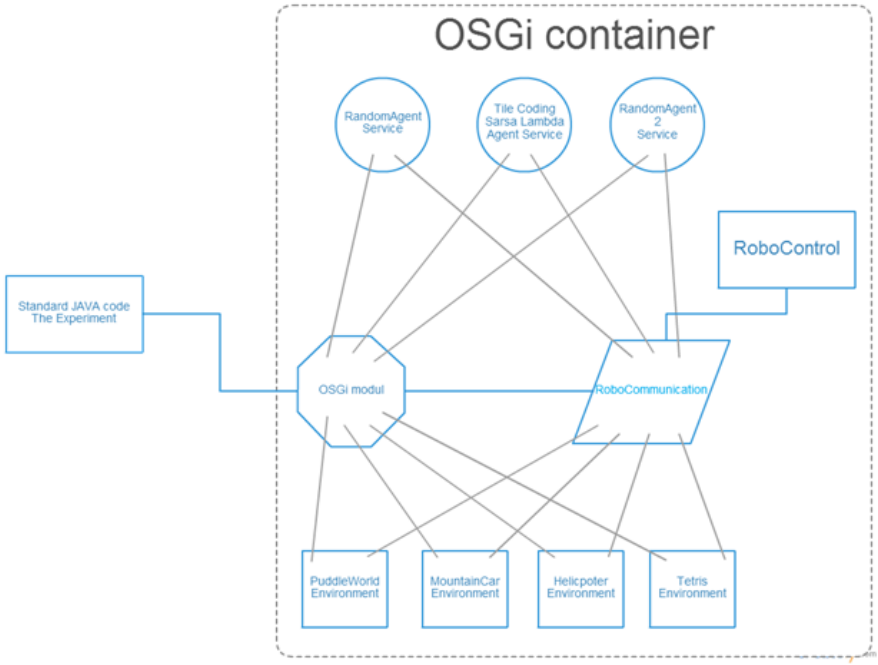
Az Experiment komponens irányítja a teljes kísérlet végrehajtását. E komponens nincs direkt kapcsolatban az Agent és az Environment komponensekkel. Van köztük egy köztes réteg, amely végzi a kommunikációt e három komponens között. Az Experiment komponensben van meghatározva a lépések száma egy adott iterációban, illetve az iterációk száma is. Fontos azon szerepe is az Experiment komponensnek, hogy a végső eredményeket ő kapja meg az Agent illetve az Environment komponensektől a köztes rétegen keresztül. A fent említett köztes réteg az Rl-Glue projekt esetén az úgynevezett RL-Glue mely a teljes kommunikáció lebonyolítását végzi a komponensek között illetve létrehozza a hálózati kommunikációhoz szükséges objektumokat.

A RoboRun projekt esetén az RL-Glue komponens helyét felváltja a RoboControl komponens, viszont a funkcionalitások nagy része megmarad vagy csak részben változik. Például a RoboRun projekt esetén nincs szükség ezen a szinten beállítani a hálózati kommunikációt.

Az Agent, Environment, Experiment és a RoboControl hasonló módon működnek mint az Rl- Glue projekt esetén, viszont a RoboRun projektben az Experiment kivételével, minden komponens egy szolgáltatás az OSGi konténerben, amely egy távoli GlassFish[2] szerveren fut. Az Experiment komponens és az OSGi konténer közötti kapcsolat megteremtéséért egy OSGi modul a felelős. E modul ismeri az összes konténerbe telepített Agent illetve Environment komponenst.

A konténerben egyszerre több előre definiált Agent és Environment lehet telepítve, ezek száma nincs korlátozva. Bármikor módosítható, törölhető vagy teljesen új Agent és Environment is hozzáadható a konténerhez anélkül, hogy a szervert meg kellene állítani vagy újra kellene indítani.

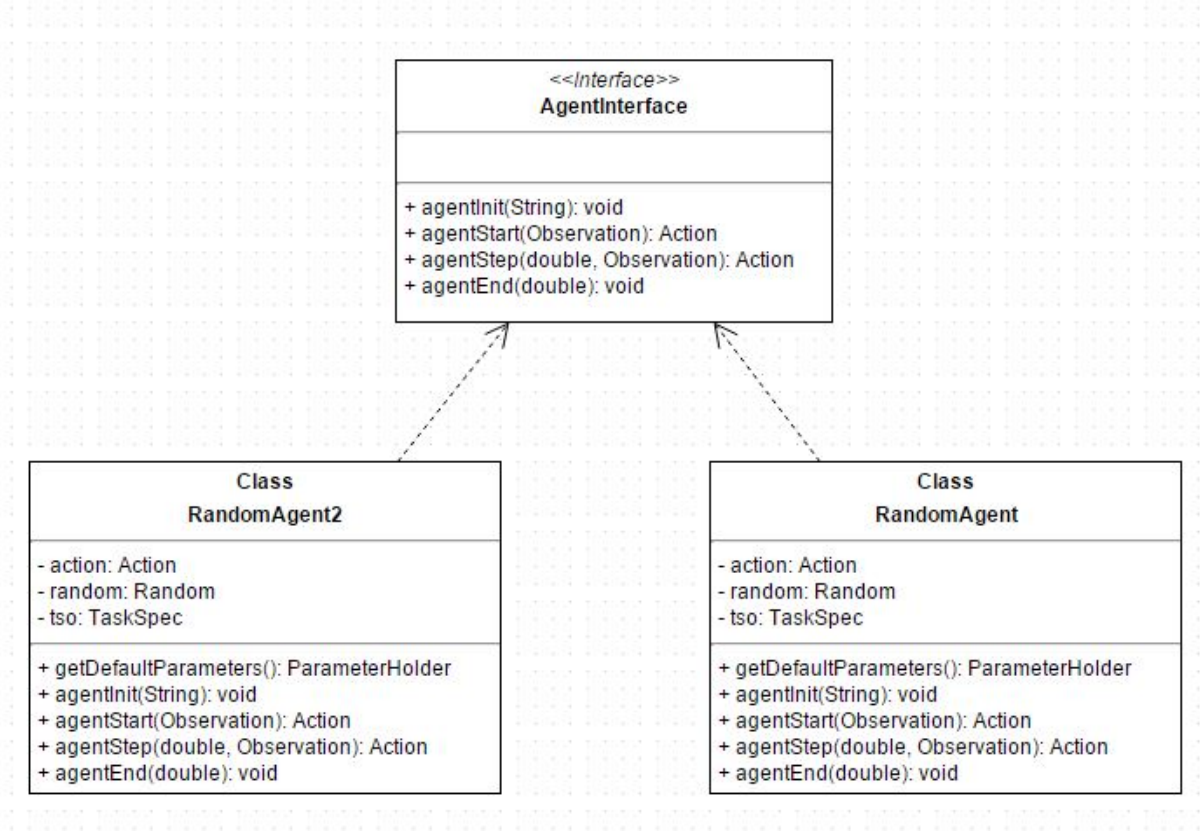
Egy kísérlet futtatása során kliens oldalon az Experiment komponens meghatározza a szükséges lépések számát iterációnként és az iterációk számát, majd kapcsolatba lép az OSGi modullal. Az OSGi modul a fent említett módon, ismer minden olyan Agent - et és Environment - et, amely telepítve van a konténerbe. Ezek közül választhat az Experiment komponens, hogy melyiket szeretné használni. Tehát kiválaszthatja azt, hogy melyik Environment - hez milyen Agent -t szeretne használni a kísérlet során.



2.2. ábra. A RoboRun projekt alap komponensek közti kommunikáció:

Ezek az információk mind az OSGi modulon keresztül jutnak el a szervertől a kliensig. Amint az Experiment komponens meghatározta a kívánt Agent -t és Environment -t, az OSGi modul közvetíti ezt a RoboCommunication komponens fele, mely lekérdezi a kiválasztott Agent és Environment szolgáltatásokat. Ezek folyamatos interakcióba kerülnek egymással melyet a RoboControl komponens irányít. Amint véget ér a kísérlet az Experment komponens az OSGi modul segítségével megkapja a kísérlet eredményét.

Minden Agent implementálja az AgentInterface -t és minden Environment implementálja az EnvironmentInterface -t.



2.3. ábra. Az Agent osztály UML diagramja

3. fejezet

Felhasznált módszerek és eszközök

Összefoglaló: *Minden nagyobb projekt esetén a fejlesztőknek szükségük van arra, hogy a megfelelő felkészültségük és találékonyságuk mellett, figyelmet fordítsanak a hatékony munkára is. Ehhez szükségük lehet különböző verziókövető rendszerek használatára, projektmenedzsment eszközökre és build rendszerekre.*

3.1. Verziókövetés

Napjainkban egyre nagyobb szükség van arra, hogy egy projekt esetén a munka könnyedén megosztható és hordozható legyen a fejlesztők közt. E mellett nagyon fontos a fejlesztési folyamat monitorizálása.

E célra fejlesztették ki a verziókövető rendszereket és a projektmenedzsment eszközöket melyek által könnyedén megoszthatóvá válik a fejlesztői munka és folyamatosan ellenőrizhető a fejlesztés folyamata.

A verziókövető rendszer által folyamatosan nyomon követhető a projekt fejlődése. Könnyed visszaállítási lehetőséget biztosit arra az esetre, ha valami történne a lokális gépünkön tárolt forrás állományokkal vagy ha bármi hiba történne a fejlesztés során ami visszaállítást igényel. Legnagyobb haszna a verzió követő rendszereknek, az olyan projekteknél van, amelyet több fejlesztő fejleszt egyszerre. Így nagyon egyeszűen követhető, hogy melyik fejlesztő milyen fázisban tart.

A RoboRun projekt fejlesztése során a forrásállományok tárolására és a fejlesztés nyomkövetésére felhasznált verziókövető rendszer a Git[3], amely nyílt forráskódú és teljesen ingyenes. A projekt szerzője által használt kliensalkalmazás a TortoiseGit[4]. A TortoiseGit rendelkezik grafikus felülettel, így használta egyszerű.

3.2. Projektmenedzsment

"A projektmenedzsment az erőforrások szervezésével és azok irányításával foglalkozó szakterület, melynek célja, hogy az erőforrások által végzett munka eredményeként egy adott idő- és költségkereten belül sikeresen teljesüljenek a projekt céljai." (forrás: http://hu.wikipedia.org/wiki/Projektmenedzsment)

A projektmenedzsment eszközök fő célja tehát, hogy a fejlesztők a specifikáció által meghatározott feladatot adott idő - és költségkereten belül sikeresen tudják teljesíteni. Ennek érdekében számos hasznos funkcionalitást biztosítanak. Ilyen funkcionalitások például, különböző feladatkörök kiosztása, különböző feladatok kiosztása, egy adott folyamatra szánt idő meghatározása, a fejlesztő által eltöltött munkaidő egy adott rész megvalósításával. Mindezek mellett kommunikációs lehetőséget biztosít a fejlesztők között. Lehetőség ad feltölteni dokumentumokat, diagramokat, segédanyagokat a projekthez, ez

által megkönnyítve a fejlesztők munkáját.

A RoboRun projekt fejlesztése során alkalmazott projektmenedzsment eszközként a Redmine[5] webes menedzsment eszköz szolgált. Amely nyílt forráskódú és platform független. Egyszerű és letisztult felülete révén könnyen kezelhető. Néhány Redmine által nyújtott funkcionalitás: naptár, e-mail értesítés, szerepkör szerinti hozzáférés, wiki, stb.

3.3. Build rendszer

A build rendszereket többnyire projektek menedzselésére és a build folyamat automatizálására alkalmazzák.

A RoboRun projekt fejlesztése során alkalmazott build rendszer a Maven[6], amelyet Jason van Zyl készített 2002-ben. A Maven egy nyílt forráskódú, platform független eszköz. Leggyakoribb felhasználása a Java nyelvben írt projektek esetében történik. A Maven konfigurációs modellje XML alapú, e mellett bevezetésre került a POM(Project Object Model). A POM az adott projekt szerkezeti vázának teljes leírását tartalmazza és a modulokat azonosítókkal látja el. Ez a pom.xml állomány által valósul meg, amely tartalmazza ezeket az információkat.

A RoboRun projekt esetén a Maven build eszköz legfontosabb szerepe a függőségek, célok és pluginok kielégítése a build folyamat során, hiszen a Maven saját függőség kezelő rendszerrel rendelkezik, amely a build - elés során letölti a központi repositoryból(gyűjteményből) az előre megadott függőségeket és elhelyezi a lokális tárolóban, ahonnan a jövőben használni fogja.

4. fejezet

Az OSGi keretrendszer

Összefoglaló: *E fejezet célja bemutatni az OSGi keretrendszert, illetve annak architektúráját. Bemutatja, hogy a RoboRun projekt miért használja az OSGi keretrendszert. Végül egy általános leírást ad arról, hogy a RoboRun projekt, hogyan használja az OSGi keretrendszert a megerősítéses tanulási kísérletek futtatására és tesztelésére.*

4.1. Az OSGi keretrendszer

Az OSGi -t eredetileg arra fejlesztették ki, hogy home gateway - ként működjön. Ez azt jelenti, hogy a home gateway kapcsolatban áll egy szolgáltatóval és a felhasználók által kifizetett szolgáltatásokhoz biztosít elérést. Tehát a szolgáltató kezében van a teljes menedzselés joga, a felhasználó csak használja az adott szolgáltatásokat.

Az OSGi keretrendszer egy olyan keretrendszer, mely a Java nyelv fölött fut. Az OSGi jelentése, Open Service Gateway Initiative. E keretrendszer célja bővíthető Java alkalmazások fejlesztésének a támogatása. Teljesen dinamikus környezetet biztosít, hiszen képes kezelni a csomagok futás idejű megjelenését és eltűnését a nélkül, hogy a felhasználó bármit is észrevenne ebből. Lehetőséget nyújt különböző szolgáltatások definiálására, amelyek folyamatosan bővíthetőek, változtathatóak, szintén futás időben. Mindezek mellett nagyon jól megvalósítja a komponensek egymástól való elkülönítését.

Az OSGi biztosít néhány nagyon fontos és nélkülözhetetlen eszközt, amelyek segítségével különböző szolgáltatásokat lehet építeni.

4.2. Az OSGi architektúra

5. fejezet

Eredmények bemutatása és értékelése

5.1. Az utazóügynök feladata

A standard utazóügynök feladat a következõ:

Adott egy súlyozott gráf G = (V,E) a cij súly az i és j csomópontokat összekötõ élre vontakozik, és a cij érteke egy pozitiv szám. Találd meg azt a körutat, amelynek minimális a költsége.

5.2. Az utazóügynök feladatára vonatkozó heurisztikák

Az utazóügynök feladata egy np-teljes feladat. A megoldás megtalálására túl sok idõ szükséges, ezért heurisztikákat használunk.

5.2.1. Beszúrási herusztika

A beszúrási heurisztikát akkor használjuk, amikor egy új csomópontot akarunk beszúrni a körútba. úgy szúrunk be egy új csomópontot , hogy a körút hossza minimális legyen. A csomópontot minden pozícióba megpróbáljuk beszúrni, és mindig kiszámoljuk a költséget. Az új pozícioja a csomópontnak a körútban az a pozício lesz, ahol a költség minimális.

5.2.2. Körútjavító heurisztika

A körút javito heurisztikakat arra használjuk, hogy meglévõ megoldásokat javitsunk. A legismertebb heurisztikak a 2-opt és a 3-opt heurisztikak.

A 2-opt herurisztika

A 2 opt heurisztika megprobal talalni 2 elet, amelyet el lehet tavolitani, és 2 elet, amelyet be lehet szúri, ugy, hogy egy körútat kapjunk, amelynek a költsége kisebb mint az eredetié. Euklideszi távolságoknál a 2-opt csere, kiegyenesíti a körútat, amely saját magát keresztezi.

A 2-opt algoritmus tulajdonképpen kivesz két élet a körútból, és újra összeköti a két keletkezett útat. Ezt 2-opt lépésként szokták emlegetni. Csak egyetlen módon lehet a két keletkezett útat összekötni úgy, hogy körútat kapjunk. Ezt csak akkor tesszük meg, ha a keletkezendõ körút rövidebb lesz. Addig

veszünk ki éleket, és kötjük össze a keletkezett utakat, ameddig már nem lehet javítani az úton. Igy a körút 2-optimális lesz. A következõ kép egy gráfot mutat amelyen végrehajtunk egy 2-opt mozdulatot.

A genetikus algoritmus

A gentikus algoritmusok biologia alapú algoritmusok. Az evolúciót utánozzák, és ez által fejlesztenek ki megoldásokat, sokszor nagyon nehéz feladatokra. Az általános gentikus algoritmusnak a következõ lépéseit különböztethetjük meg:

1. Létrehoz egy véletlenszerû kezdeti állapotot

Egy kezdeti populációt hozunk létre, véletlenszerûen a lehetséges megoldásokból. Ezeket kromoszómáknak nevezzük. Ez különbözik a szimbolikus MI rendszerektõl, ahol a kezdeti állapot egy feladatra nézve adott.

2. Kiszámítja a megoldások alkalmasságát

Minden kromoszómához egy alkalmasságot rendelünk, attól függõen, hogy mennyire jó megoldást nyújt a feladatra.

3. Keresztezés

Az alkalmasságok alapján kivalásztunk néhány kromoszómát, és ezeket keresztezzük. Eredményûl két kromoszómát kapunk, amelyek az apa és az anya kromoszóma génjeinek a kombinációjából állnak. Ezt a folyamatot keresztezésnek nevezzük. Keresztezéskor tulajdonképpen két részmegoldást vonunk össze, és azt reméljük, hogy egy jobb megoldás fog keletkezni.

4. A következõ generáció létrehozása

Ha valamelyik a kromoszómák közül tartalmaz egy megoldást, amely elég közel van, vagy egyenlõ a keresett megoldással, akkor azt mondhatjuk , hogy megtaláltuk a megoldást a feladatra. Ha ez a feltétel nem teljesûl, akkor a következõ generáció is át fog menni az **a.-c**. lépéseken. Ez addig folytatódik, ameddig egy megoldást találunk.

A. függelék

Fontosabb programkódok listája

Itt van valamennyi Prolog kód, megfelelõen magyarázva (komment-elve). A programok beszúrása az `\lstinputlisting[multicols=2]{progfiles/lolepes.pl}` paranccsal történik, és látjuk, hogy a példában a `progfiles` könyvtárba tettük a file-okat.

Az alábbi kód Prolog nyelvbõl példa. Az `\lstset{language=Prolog}` paranccsal a programnyelvet változtathatjuk meg, ezt a **`listings`** csomag teszi lehetõvé [Heinz és Moses, 2007], amely nagyon jól dokumentált.

```
% lolepes(N,Lista) - az NxN-es sakktáblán lép a lóval
lolepes(N,Lista):-
    N2 is ceiling(N / 2), numlist(1,N2,NLista),
    member(Kx,NLista), member(Ky,[1,2]),
    egeszit(N,[Kx/Ky],Lista).

% egeszit(N,Lis1,Lis2).
% megáll, ha N*N mezon már voltunk.
egeszit(N,Lis1,Lis2) :-
    N2 is N * N,
    length(Lis1,N2),
    reverse(Lis1,Lis2),!.

% keresünk következő lépést
egeszit(N,[Fej|Mar],Valasz) :-
    egylepes(N,Fej,Utan,Mar),
    egeszit(N,[Utan,Fej|Mar],Valasz).

% egylepes(Ex/Ey,Ux/Uy,Tiltott) - Ex/Ey
% mezorol lép úgy, hogy a Tiltott
% elemeket kerüli.
egylepes(N,Ex/Ey,Ux/Uy,Tiltott) :-
    LL=[1/2, 2/1, 1/(-2),
        (-2)/1,(-1)/2, 2/(-1),
        (-1)/(-2),(-2)/(-1)],
    member(Ix/Iy,LL),
    Ux is Ex + Ix, Ux > 0, Ux =<N,
    Uy is Ey + Iy, Uy > 0, Uy =<N,
    \+ member(Ux/Uy,Tiltott).

korLepes(N,Lista) :-
    lolepes(N,Lista),
    [Fej|_] = Lista,
    last(Lista,Veg),
    egylepes(N,Fej,Veg,[]).

%%%%%%%%%%%%% MEGJELENITES %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% osszlepes(N) - kiírja az összes
% lehetséges bejárást, visszalépéssel
osszlepes(N) :-
    lolepes(N,Lista),
    tikzKiir(Lista),
    fail.

% tikzKorKiir(Lista) - a listaban szereplő teljes
% lólépés-sort kiirja a Latex-hez - felté telezi,
% hogy a LISTA kör.
tikzKorKiir([Fej|Mar]) :-
    % meret megallapítása
    length([Fej|Mar],N2),N is ceiling(sqrt(N2)),
    writeln('\%'),
    writePre(N),
    drawkezd(Fej),
    writeDraw(Mar),
    writeln('    \\draw (start) -- (stop);'),
    writePost,!.

% tikzKiir(Lista) - a listaban szereplő teljes lólépés-sort
% kiirja a Latex-hez kompilálásra.
tikzKiir([Fej|Mar]) :-
    % meret megallapítása
    length([Fej|Mar],N2),N is ceiling(sqrt(N2)),
    writeln('\%'),
    writePre(N),
    drawkezd(Fej),
    writeDraw(Mar),
    writeln('    \\node[draw,circle] at (start)  {$\\cdot$};'),
    writeln('    \\node[draw,rectangle] at (stop)  {$\\cdot$};'),
    writePost,!.

% Preambulum a TIKZ képhez
writePre(N) :-
    write('\\begin{tikzpicture}[line width=1.5pt,scale='),
    Sc is min(1.5,3.6/N),
    write(Sc),writeln(']'),
    write('  \\draw[step=1cm,gray!25!red!25!, thick] (-0.1,-0.1) grid ('),
    write(N),write('.1,'),write(N),writeln('.1);'),
    writeln('  \\begin{scope}[color=blue!35!green!,minimum size=0.2cm,\%'),
    writeln('      xshift=-0.5cm,yshift=-0.5cm,inner sep=0pt,outer sep=0pt]').

% drawkezd(Fej) - kiirja a kezdopozíciót és kezdi vonalat.
drawkezd(Kx/Ky) :-
    write('    \\coordinate (start) at ('),
```

```
    write(Kx),write(','),write(Ky),writeln(');
        '),
    write('␣␣␣␣\\draw[rounded␣corners=1pt]␣(
        start)').

% writeDraw(Lista) - befejezi a vonalkiírást.
writeDraw([Kx/Ky]) :-
    write('␣--␣('),write(Kx),write(','),write(
        Ky),writeln(');'),
    write('␣␣␣␣\\coordinate␣(stop)␣at␣('),
    write(Kx),write(','),write(Ky),writeln(');
        ').
writeDraw([Kx/Ky|Marad]) :-
    write('␣--␣('),write(Kx),write(','),write(
        Ky),write(')'),
    writeDraw(Marad).

writePost :-  % vége - nincs paraméter
    writeln('␣␣\\end{scope}\n\\end{tikzpicture
        }').
```

Irodalomjegyzék

Baldi, P. és Brunak, S. *Bioinformatics: The Machine Learning Approach*. The MIT Press, 1998.

Boyd, S. P. és Vandenberghe, L. *Convex Optimization*. Cambridge University Press, Cambridge, UK, 2004. URL http://www.stanford.edu/~boyd/cvxbook.html.

Doob, M. *TeX könnyedén (A gentle introduction to TeX)*. POLYGON, 1995. URL http://www.inf.unideb.hu/~matex/konyvek/texkonyvek.html.

Heinz, C. és Moses, B. The listings package. Technical report, CTAN TEX Archive, 2007. URL http://www.ctan.org/tex-archive/macros/latex/contrib/listings.

Mitchell, T. M. *Machine Learning*. Computer Science Series. McGraw-Hill, New York, 1997.

Mittelbach, F., Goossens, M., Braams, J., Carlisle, D., és Rowley, C. *The Latex Companion*. Addison Wesley, 2004.

Nabney, I. T. *NETLAB - Algorithms for Pattern Recognition*. Springer, 2002.

Oetiker, T., Partl, H., Hyna, I., és Schlegl, E. Egy nem túl rövid bevezető a LaTex használatába. Technical report, BME math Latex, 1998. URL http://www.math.bme.hu/latex.

Oetiker, T., Partl, H., Hyna, I., és Schlegl, E. The not so short introduction to LaTex. Technical report, CTAN TEX Archive, 2008. URL http://www.ctan.org/tex-archive/info/lshort/english.

Rabiner, L. és Juang, B.-H. *Fundamentals of Speech Recognition*. Prentice-Hall, 1993.

Vapnik, V. N. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.