

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

Diploma Thesis

License thesis title

Abstract

EZ AZ OLDAL NEM RÉSZE A DOLGOZATNAK!

Ezt az angol kivonatot külön lapra kell nyomtatni és alá kell írni!

A DOLGOZATTAL EGYÜTT KELL BEADNI!

Kötelező befejezés:

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

2018

GÁSPÁR ADALBERT

ADVISOR:
ASSIST PROF. DR. HUNOR JAKAB

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

Diploma Thesis

License thesis title



ADVISOR:

ASSIST PROF. DR. HUNOR JAKAB

STUDENT:

GÁSPÁR ADALBERT

2018

UNIVERSITATEA BABEȘ–BOLYAI, CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licență

Titlu lucrare licență



CONDUCĂTOR ȘTIINȚIFIC:
LECTOR DR. HUNOR JAKAB

ABSOLVENT:
GÁSPÁR ADALBERT

2018

BABEŞ–BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Szakdolgozat
Szakdolgozat cím



TÉMAVEZETŐ:

DR. OKTATÓ OTTÓ,
EGYETEMI ADJUNKTUS

SZERZŐ:

GÁSPÁR ADALBERT

2018

Tartalomjegyzék

1. Alapok	3
1.1. A gépi tanulás	3
1.2. Adatelemzés	3
1.3. Szerkesztés	3
2. Diszkrimináns-módszerek	7
2.1. Lineáris diszkrimináns	7
3. Matlab és Netlab ismertető	8
3.1. Rövid ismertető	8
3.2. Matlab műveletek	8
3.3. Matlab függvények	9
3.4. Netlab bevezető	10
4. Eredmények bemutatása és értékelése	12
4.1. Az utazóügynök feladata	12
4.2. Az utazóügynök feladatára vonatkozó heurisztikák	12
4.2.1. Beszúrási heurisztika	12
4.2.2. Körútjavító heurisztika	12
A. Fontosabb programkódok listája	14

1. fejezet

Alapok

Összefoglaló: A fejezetek elején egy rövid összefoglalót teszünk. Ez a rész opcionális.

1.1. A gépi tanulás

A gépi tanulás neve Mitchell [1997] azonos című – „*Machine Learning*” – könyvéből származtatható. A könyv alapján azt a kutatási területet nevezzük így, amelyben a cél olyan programok írása, amelyek futtatásuk során fejlődnek, vagyis valamilyen szempont szerint jobbak, okosabbak lesznek.¹ Itt az „okosság” metaforikus: a futási idő folyamán valamilyen mérhető jellemzőnek a javulását értjük alatta. Például a felhasználás kezdetén a szövegfelismerő még nem képes a szövegek azonosítására, azonban a használat – és a felhasználói utasítások – után úgy módosítja a működési paramétereit, hogy a karakterek egyre nagyobb hányadát tudja felismerni.

1.2. Adatelemzés

A mesterséges intelligencia azon módszereit, amelyeket numerikus vagy *enyhén strukturált*² adatokra tudunk alkalmazni, gépi tanulási módszereknek nevezzük [Mitchell, 1997]. A gépi tanulás e meghatározás alapján egy szerteágazó tudományág, amelynek keretén belül sok módszerről és ennek megfelelően sok alkalmazási területről beszélhetünk. A korábban említett neurális modellekkel ellentétben a központban itt az adatok vannak: azok típusától függően választunk például a binomiális modell és a normális eloszlás, a fő- vagy független-komponensek módszere vagy a k-közép és EM algoritmusok között. Egyre több adatunk van, azonban az „információt” megtalálni egyre nehezebb.³

1.3. Szerkesztés

A következőkben áttekintjük a \LaTeX dokumentumok szerkesztésének alapjait.

Átfogó referenciák a következők:

[Oetiker et al., 2008] – egy \LaTeX gyorstalpaló. A könyvben nagyon célirányosan mutatják be a szerkesztési szabályokat és a fontosabb parancsokat.

1. Részlet Mitchell [1997] könyvéből:

„The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience.”

2. **Enyhén** strukturált (nagyon felületesen): az adatok komponensei (dimenziók) közötti kapcsolat nem túl bonyolult.

3. David Donoho, a Stanford Egyetem professzora szerint a XXI. századot az adatok határozzák meg: azok gyűjtése, szállítása, tárolása, megjelenítése, illetve az adatok **felhasználása**.

1. FEJEZET: ALAPOK



1.1. ábra. Példa képek beszúrására: a képen rev. Thomas Bayes látható. A képek után *kötelezően* szerepelnie kell a forrásnak:

http://en.wikipedia.org/wiki/Thomas_Bayes

[Oetiker et al., 1998] – a gyorstalpaló magyar változata egy BME-s csapat jóvoltából.

[Doob, 1995; Mittelbach et al., 2004] – az angol nyelvű alapkönyvek. Kérésre az utóbbi – [Mittelbach et al., 2004] – elérhetővé tehető.

Természetesen a Kari könyvtárban is találtak könyvészetet. A fentebb említetteken kívül nagyon sok internetes oldal tartalmaz \LaTeX szerkesztésről bemutatókat:

Képeket beszúrni a

```
\pgfimage[width=0.4\linewidth]{images/bayes}
```

paranccsal lehet, ahol a

```
width=0.4\linewidth
```

a kép szélességét jelenti. Amennyiben a magasság nincs megadva – mintjelen esetben – akkor azt automatikusan számítja ki a rendszer, az eredeti kép arányait figyelembe véve. Egy másik jellegzetesség az, hogy a képek kiterjesztését nem adjuk meg – a \LaTeX megkeresi a számára elfogadható kiterjesztéseket, azok listájából az elsőt használja. A .JPG, .PNG, .TIFF, valamint a .PDF kiterjesztések is használhatók.

Két kép egymás mellé tétele a `tabular` környezet-mintával lehetséges, amint a 1.2 ábrán látjuk. Amennyiben grafikonunk van, általában ajánlott VEKTOROSAN menteni – ezt általában a PDF driverrel tesszük – az eredmény a 1.3 ábrán látható.

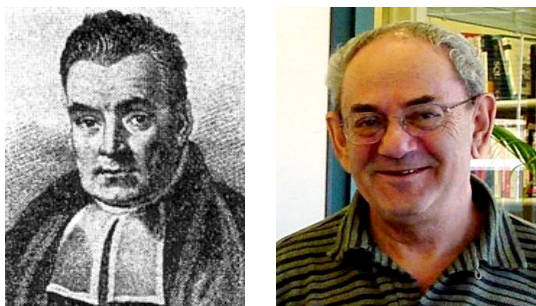
A szerkesztés folyamata során ajánlott a:

- különböző strukturális elemek használata: a `\chapter`, `\section`, `\subsection`, `\subsubsection` parancsok.
- a listák használata felsorolásoknál;
- a tétel típusú environmentek és a bizonyítás-environment használata (alább bemutatunk néhányat, az összes értelmezett tétel típust megtalálod a `definitions.sty` fájlban).

1.1. Értelmezés. Az n pozitív egész számot *négyzetmentesnek* nevezzük, ha az n prímtényező felbontásában minden prím legfentebb az első hatványon szerepel.

1.2. Példa. Az 1, 7 és 33 természetes számok négyzetmentesek, míg a 9 és a 45 nem.

1. FEJEZET: ALAPOK



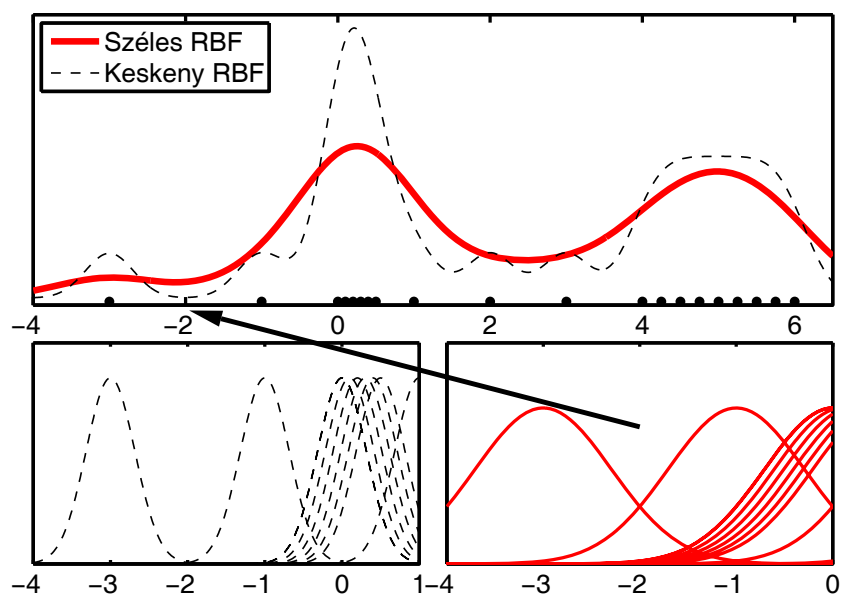
1.2. ábra. Példa képek beszúrására: a bal oldalon rev. Thomas Bayes, a jobb oldalon egy jelenkori matematikus, Vladimir Vapnik látható.

http://en.wikipedia.org/wiki/Vladimir_Vapnik

1.3. Tétel. *Az n pozitív egész szám akkor és csak akkor négyzetmentes, ha minden n elemű Abel csoport ciklikus.*

Bizonyítás. Túl hosszú! ■

1.4. *Megjegyzés.* Az 1.3 tétel a végesen generált Abel csoportok jellemzési tételének a következménye.



1.3. ábra. Példa grafika beszűrésére.

2. fejezet

Diszkriminációs-módszerek

Összefoglaló: Példafejezet. Nem releváns a szöveg.

A fejezetben a matematikai elemeket illusztráltuk.

2.1. Lineáris diszkrimináció

Legyen ismert az x_1, \dots, x_n gyakorló minták sorozata és a minták osztályozása. A minták száma N , az Ω_x mintatér d dimenziója sokkal kisebb, mint N .

Célunk egy olyan függvény meghatározása, mely *diszkriminál* az adatok terében, azaz a pozitív példákra pozitív értékkel, a negatívakra pedig negatív értékkel tér vissza:

$$f : \Omega_x \rightarrow \mathbb{R} \quad \text{úgy, hogy} \quad f(x) \begin{cases} < 0 & \forall x \in Neg \\ \geq 0 & \forall x \in Poz \end{cases} \quad (2.1)$$

ahol a negatív doméniumot *Neg*-gel, a pozitívet meg *Poz*-zal jelöltük.

Figyeljük meg a \LaTeX kódban a `\label` használatát: a szövegben kijelentünk egy *címkét*, melyet az `\eqref{eq:diszkr:fugg}` vagy a `\ref{eq:diszkr:fugg}` parancsokkal tudunk később beszúrni a szövegbe. A kompilálás során a \LaTeX mindig aktualizálja a mutató értékét, nem kell tehát újraszámolni kézzel az objektumokat. Ez természetesen érvényes más számozott egységre is, mint fejezetek, alfejezetek, bekezdések, ábrák, stb. – lásd a jelen fejezet \TeX kódját.

Lineáris diszkriminációs függvény

A diszkriminációs függvények legegyszerűbb változata a lineáris. A lineáris diszkriminációs függvényeket a következőképp definiáljuk:

$$D_k(x) = x_1 \alpha_{1k} + \dots + x_N \alpha_{Nk} + \alpha_{N+1,k} \quad k = 1, 2, \dots, K,$$

ahol K az osztályok száma, x_1, \dots, x_N az x mintavektor N komponense, az α számok a súlyozó együtthatók. Vektoros formában felírva:

$$D_k(x) = \tilde{x}^T \alpha_k = \alpha_k^T \tilde{x},$$

ahol $\tilde{x}^T = [x^T, 1]$ a transzponáltja \tilde{x} -nak, a megnövelt mintavektornak, és α_k a k -adik súlyozó vektor, amely tartalmazza az $N + 1$ súlyozó együtthatót.

3. fejezet

Matlab és Netlab ismertető

Összefoglaló: A következőkben a jegyzet során használt Matlab nyelvet mutatjuk be és definiáljuk a használt függvényeket. A Matlab programnyelven írták meg a NETLAB csomagot, mellyel nagyon könnyen lehet mintafelismerő algoritmusokat elemezni.

3.1. Rövid ismertető

A Matlab nyelv egy *interpreter*. A változókat létrehozzuk, nincs szükség azok deklarálására. A változókat megfeleltetésekkel hozzuk létre. A változók lehetnek:

valós típusúak – például **a=ones (5, 1)** ; , amelyekről a rendszer megjegyzi, hogy mekkorák és a megfelelő mennyiségű memóriát lefoglalja. A változók alapértelmezetten mátrixok, azonban lehetőség van magasabb fokú tenzorok definíciójára is, például az **b=ones (5, 5, 2)** ; egy $5 \times 5 \times 2$ -es méretű tenzort hoz létre, mely két darab 5×5 -ös mátrixot tárol és melyekre a **b (:, :, 1)** , valamint a **b (:, :, 2)** parancsokkal hivatkozunk. Egy mátrixban egy egész sort *kettősponttal* választunk ki. A Matlab-ban nincsenek egész vagy logikai típusú változók.

sztring típusúak – például **s='a1b2c3d4'** egy sztringet hoz létre. A sztringek karakter típusú vektorok, melyekkel az összes mátrix művelet is végezhető.

cella típusúak – például **c={'sty', [1;2;3;4;5;6], 2}**, mindegyik elem lehet különböző típusú és méretű. A cellák elemeire a **c{3}** jelöléssel hivatkozunk és nem tudunk a vektorokra illetve a mátrixokra jellemző műveletek alkalmazni azokon.

Mint általában, a Matlab rendszerben is segít a **help** parancs, mely egy adott parancshoz ad magyarázatot: a **help <függvény>** a függvényhez tartozó magyarázatot jeleníti meg. A rendszerbe be van építve egy további segítség, a **demo** parancs, mely példákon keresztül mutatja be a Matlab működését és az interpreter jelleg által nyújtott lehetőségeket.

3.2. Matlab műveletek

A Matlab nyelvben a vektorokra jellemző műveletek jelölése intuitív: a mátrix transzponáltját a **bt=b'** művelet, a mátrix-szorzatot a **c=b (:, :, 1) *a** jelöli. Amennyiben a műveletek operandusai nem megfelelő méretűek, a rendszer hibaüzenetet ír ki. A szokásos aritmetikai műveleteken kívül ismeri a rendszer a hatványozást is, a \wedge jelöléssel, melyek mind érvényesek a mátrixokra is. Az osztáshoz például két művelet is tartozik, melyek az $A \cdot X = B \Leftrightarrow X = A \backslash B$, valamint a $X \cdot A = B \Leftrightarrow X = A / B$ lineáris egyenleteket oldják meg.

Sokszor szeretnénk, ha elemenként végezne a rendszer műveleteket a mátrixokon, ezt a műveleteknek pontokkal való prepozíciójával tesszük. Például a **C=(b (:, :, 2)) ^ 2** a mátrix önmagával való

szorzásának az eredményét, a $\mathbf{C} = (\mathbf{b}(:, :, 2)) .^2$ a mátrix elemeinek a négyzeteit tartalmazó mátrixot adja eredményül.

3.3. Matlab függvények

A jegyzet során a programokban gyakran alkalmaztuk a következő függvényeket:

rand – egy véletlen számot térít vissza a $[0, 1]$ intervallumból az intervallumon egyenletes eloszlást feltetelezve. Argumentum nélkül a függvény egy nulla és egy közötti véletlen számot, egy argumentummal egy $k \times k$ méretű véletlen mátrixot, egy vektorra pedig egy tenzort térít vissza, melynek méreteit a vektor elemei tartalmazzák. A **randn** hasonlóan véletlen változókat visszatérítő függvény, azonban azok nulla átlagú és egy szórású normális eloszlást követnek.

ones – a fenti esethez hasonlóan egy vektort, mátrixot vagy tenzort térít vissza, azonban az elemeket 1-gyel tölti fel. A **zeros** az elemeket lenullázza.

linspace – a bemenő argumentumok skalárisak és a függvény visszatéríti az első két argumentum mint intervallum N részre való felosztásának a vektorát; az N a harmadik bemenő változó.

randperm – egy argumentummal a k hosszúságú $[1, \dots, k]$ vektor egy véletlen permutációját téríti vissza.

union – a lista elemeit halmazként használva egyesíti a két bemenő halmazt.

setdiff – a lista elemeit halmazként használva visszatéríti az argumentumok metszetét. Használhatjuk még a **unique** és az **ismember** parancsokat a halmazzal való műveletekre.

find – indexeket térít vissza. Egy vektor azon elemeinek indexét, mely egy bizonyos feltételnek eleget tesz. Használják a vektor elemeinek szelekciójára.

repmat – első argumentuma egy mátrix, amit adott sokszorossággal bemásol az eredménybe. A sokszorosságot a második argumentum adja meg.

reshape – argumentumai egy vektor vagy mátrix illetve egy méret paraméter. A függvény az első argumentum elemeit a második argumentumban található méretek szerint formázza át. Pl. a \mathbf{v} 256 hosszú vektort az \mathbf{m} 16×16 -os vektorba a **m=reshape(v, [16, 16])** paranccsal alakítjuk át.

meshgrid – az első argumentum elemeit X -tengelyként, a második argumentumét Y -tengelyként használva két $m \times n$ -es mátrixot térít vissza, ahol az elemek a négyzetháló X , illetve Y koordinátái oszlop-szerinti bejárásban. Hasznos amikor egy felületet szeretnénk kirajzolni.

diag – ha a bemenő argumentum egy mátrix, akkor az átlón levő elemek vektora az eredmény, ha pedig egy vektor, akkor az az átlós mátrix, mely nulla a főátló elemeit kivéve, ott meg a bemenő vektor elemei találhatóak. Igaz a **b = diag(diag(b))** állítás.

inline – egy függvény, mely segít rövid függvényeket – általában egysorosakat – definiálni, a függvényekben az alapértelmezett argumentum az x , több argumentum esetén a sorrendet is lehet specifikálni.

load – egy korábban elmentett állapottól változóit állítja vissza. A változók elmentéséhez használjuk a **save** parancsot.

fprintf – egy vektor eleit formázva kiírja, a 'C'-hez hasonló szintakszisban. Hasonlóan működnek a **disp**, a **sprintf**, valamint a **num2str** parancsok.

acosd – az argumentumra alkalmazott inverz koszinusz függvény, fokokban kifejezve. Más trigonometriai függvények a szokásosak: **sin**, **cos**, **tan**, **atan**, melyeket lehet elemenként és egyben is alkalmazni, ekkor az eredmény a vektorok elemeire alkalmazott műveletek vektora.

3. FEJEZET: MATLAB ÉS NETLAB ISMERTETŐ

- chol** – egy négyzetes pozitív definit mátrix Cholesky-felbontása. Az A mátrix Cholesky-alakja egy olyan C mátrix, mely csak a főátlón és az alatt tartalmaz nullától különböző elemeket és fennáll az $A = C \cdot C^T$ egyenlőség. Figyeljük meg, hogy a Cholesky-alakot a négyzetgyök általánosításának lehet tekinteni.
- plot** – két vektort megadva kirajzolja az $(x_1(i), x_2(i))$ pontokat és azokat összeköti egy vonallal. Egy argumentum esetén $x_1 = [1, \dots, N]$ és x_2 a bemenő paraméter. Opcionálisan lehet stílusparamétereket is megadni. A **plot3** paranccsal háromdimenzióban lehet pontokat, vonalakat megjeleníteni.
- hist** – egy adott adatvektor elemeinek a gyakoriságát rajzolja ki. Alapértelmezetten a vektor legkisebb és legnagyobb eleme közötti intervallumot osztja fel 10 részre és számolja az egyes szakaszokba eső pontok számát. Úgy az intervallum mérete, mint a részintervallumok számossága illetve mérete változhat.
- contour** – egy Z mátrix által definiált felület kontúrjait rajzolja ki. A felület generálásánál általában használjuk a **meshgrid** parancsot. A felületek rajzolására használhatjuk a **surf** parancsot is.
- figure** – létrehoz egy új ábrát illetve, amennyiben létezik a kívánt ábra, akkor aktívvá teszi azt.
- subplot m n k** – létrehoz az ábrán egy részábrát úgy, hogy az eredeti ábra terét $m \times n$ részre osztja, majd annak a k -adik komponensét teszi aktívvá.
- xlim** – az aktuális rajzon beállítja az X tengely alsó és felső határát. Ugyanígy működnek az **ylim** és **zlim** parancsok az Y illetve a Z tengelyekre.
- quadprog** – az $x^T H x + b^T x + c$ másodfokú egyenlet minimumát határozza meg, ahol feltételezzük, hogy a megoldásokat az $Ax > 0$ konvex doméniumra szűkítjük. Bővebb információk Boyd és Vandenberghe [2004] könyvében.

3.4. Netlab bevezető

A Netlab neurális modellek hatékony implementációit tartalmazza. A programcsomag egy egységes felületet nyújt a létező algoritmusok gyors teszteléséhez és az új algoritmusok írásához. A különböző módszerek közös jellemzője, hogy egy változóba – általában ennek neven **net** és egy struktúra – gyűjti össze a modell paramétereit. Ehhez általában először specifikáljuk a modellt; ennek a függvénynek a neve ugyanaz, mint a modell neve. Amennyiben szükséges, akkor a **<modellnév>init** paranccsal lehet más paramétereket is beállítani. A modelleket a **<modellnév>train** paranccsal lehet tanítani, ahol általában paraméterként kerül a tanuló adathalmaz illetve az optimalizálási folyamatot jellemző más konstansok. Amikor megvan az eredmény, akkor a tanult – becsült – modell paramétereit használjuk a **<modellnév>fwd** paranccsal. Ahol nem lehetséges az új adatokra a tesztelés, ott a paraméterek terében tudunk mintát vételezni a **<modellnév>sample** függvény segítségével.

Az általunk használt modellek a következők:

- mlp** – a többrétegű neurális háló;
rbf – az RBF típusú háló;
kmeans – a k-közép algoritmus;
som – a SOM vagy Kohonen-háló;

3. FEJEZET: MATLAB ÉS NETLAB ISMERTETŐ

A **net** struktúrának van egy azonosítója, a **net.type** mező és a többi paraméter ennek az azonosítónak is a függvénye. További mezők a bemenő illetve kimeneti adatok dimenzióit, az aktivációs függvények típusait, valamint a különböző kapcsolatokhoz rendelt súlymátrixokat tárolják.

Az optimalizálás szintén egységesen történik, minden modellnek van hibafüggvénye, ezt a **<modellnév>err** függvény tartalmazza. Az optimalizálási rutin a **netopt**, mely a struktúrát, az adatokat, valamint egy **options** vektort kap paraméterként és a visszaadott struktúra tartalmazza az optimalizált modellt. Ahhoz, hogy heterogén struktúrájú modelleket lehessen használni, minden modellhez kell írjunk egy **<modellnév>pak**, illetve egy **<modellnév>unpak** függvényt, mely a paramétereket a struktúrából egy vektorba, illetve visszaalakítja. Az **options** vektor a **netopt** függvényt paraméterezi. Egy 14 hosszúságú vektor, melynek főbb értékei:

options(1) – a hibafüggvény értékeinek a kiírása. +1-re minden lépésben kiírja a hibát, nullára csak a végén, negatív értéknél nem jelenít meg semmit;

options(2) – a megállási feltétel abszolút pontossága: amennyiben két egymásutáni lépésben az θ paraméterek kevesebbet változnak, akkor az algoritmus leáll;

options(3) – az **options(2)**-höz hasonló küszöbérték, azonban ez a hibafüggvény értékeit vizsgálja;

options(10) – tárolja és visszaadja a hibafüggvény kiértékelésének a számát;

options(11) – tárolja és visszaadja a hibafüggvény gradiense hívásának a számát;

options(14) – a lépések maximális száma, alapértelmezetten 100.

A Netlab csomagban implementálva van sok hasznos lineáris és nemlineáris modell, mint például a PCA módszer, valamint annak valószínűségi kiterjesztése, a **ppca** módszer. Megtalálható az általánosított lineáris modell – **glm** –, számos konjugált gradiens módszer és sok más. Jelen felsorolásban említettünk néhányat a használt illetve a további feladatok során használható programok közül, ezt a teljesség igénye nélkül tettük, az érdeklődő hallgatónak ajánljuk a Netlab hivatalos honlapját a <http://www.ncrg.aston.ac.uk/netlab> oldalt és Nabney [2002] Netlab könyvét.

4. fejezet

Eredmények bemutatása és értékelése

4.1. Az utazóügynök feladata

A standard utazóügynök feladat a következő:

Adott egy súlyozott gráf $G = (V, E)$ a c_{ij} súly az i és j csomópontokat összekötő élre vonatkozik, és a c_{ij} értéke egy pozitív szám. Találd meg azt a körutat, amelynek minimális a költsége.

4.2. Az utazóügynök feladatára vonatkozó heurisztikák

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

4.2.1. Beszúrási heurisztika

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

4.2.2. Körútjavító heurisztika

A körút javító heurisztikákat arra használjuk, hogy meglévő megoldásokat javítsunk. A legismertebb heurisztikák a 2-opt és a 3-opt heurisztikák.

A 2-opt heurisztika

A 2 opt heurisztika megpróbál találni 2 élet, amelyet el lehet távolítani, és 2 élet, amelyet be lehet szűrni, úgy, hogy egy körutat kapjunk, amelynek a költsége kisebb mint az eredetié. Euklideszi távolságoknál a 2-opt csere, kiegyenesíti a körutat, amely saját magát keresztezi.

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

4. FEJEZET: EREDMÉNYEK BEMUTATÁSA ÉS ÉRTÉKELÉSE

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

A genetikus algoritmus

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

1. Létrehoz egy véletlenszerű kezdeti állapotot

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

2. Kiszámítja a megoldások alkalmasságát

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

3. Keresztezés

Az alkalmasságok alapján kiválasztunk néhány kromoszómát, és ezeket keresztezzük. Eredményül 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 következő generáció létrehozása

Ha valamelyik a kromoszómák közül tartalmaz egy megoldást, amely elég közel van, vagy egyenlő a keresett megoldással, akkor azt mondhatjuk, hogy megtaláltuk a megoldást a feladatra. Ha ez a feltétel nem teljesül, akkor a következő 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, megfelelően magyarázva (komment-elve). A programok beszúrása az `\lstinputlisting[multicol=2]{progfiles/lolepes.pl}` paranccsal történik, és látjuk, hogy a példában a `progfiles` könyvtárba tettük a file-okat.

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

```
1 % 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).
6
% egeszit(N,Lis1,Lis2).
% megáll, ha N*N mezon már voltunk.
egeszit(N,Lis1,Lis2):-
  N2 is N * N,
11  length(Lis1,N2),
  reverse(Lis1,Lis2),!.

% keresünk következő lépést
egeszit(N,[Fej|Mar],Valasz):-
16  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.
21 egylepes(N,Ex/Ey,Ux/Uy,Tiltott):-
  LL=[1/2, 2/1, 1/(-2),
    (-2)/1,(-1)/2, 2/(-1),
    (-1)/(-2),(-2)/(-1)],
26  member(Ix/Iy,LL),
  Ux is Ex + Ix, Ux > 0, Ux =<N,
  Uy is Ey + Iy, Uy > 0, Uy =<N,
  \+ member(Ux/Uy,Tiltott).

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

36 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MEGJELENITES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% osszlepes(N) – kiírja az összes
41 % lehetséges bejárást, visszalépéssel
```

A. FÜGGELÉK: FONTOSABB PROGRAMKÓDOK LISTÁJA

```

osszlepes(N) :-
    lolepes(N, Lista),
    tikzKiir(Lista),
    fail.
46
% tikzKorKiir(Lista) – a listaban szereplő teljes
% lólépés-sort kiírja a Latex-hez – feltételezi,
% hogy a LISTA kör.
tikzKorKiir([Fej|Mar]) :-
51     % meret megállapítása
    length([Fej|Mar], N2), N is ceiling(sqrt(N2)),
    writeln('\%'),
    writePre(N),
    drawkezd(Fej),
56     writeDraw(Mar),
    writeln('\draw (start) — (stop);'),
    writePost, !.

% tikzKiir(Lista) – a listaban szereplő teljes lólépés-sort
61 % kiírja a Latex-hez kompilálásra.
tikzKiir([Fej|Mar]) :-
    % meret megállapítása
    length([Fej|Mar], N2), N is ceiling(sqrt(N2)),
    writeln('\%'),
66     writePre(N),
    drawkezd(Fej),
    writeDraw(Mar),
    writeln('\node [draw, circle] at (start) { $\cdot$ };'),
    writeln('\node [draw, rectangle] at (stop) { $\cdot$ };'),
71     writePost, !.

% Preambulum a TIKZ képhez
writePre(N) :-
    write('\begin{tikzpicture}[line width=1.5pt, scale='),
76     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, %'),
81     writeln('xshift=-0.5cm, yshift=-0.5cm, inner sep=0pt, outer sep=0pt]'),
    writeln(')').

% drawkezd(Fej) – kiírja a kezdopozíciót és kezdi vonalat.
drawkezd(Kx/Ky) :-
    write('\coordinate (start) at ('),
86     write(Kx), write(','), write(Ky), writeln(');'),
    write('\draw[rounded corners=1pt] (start)').

% writeDraw(Lista) – befejezi a vonalkiírást.
writeDraw([Kx/Ky]) :-
91     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(')'),
96     writeDraw(Marad).

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

```

Irodalomjegyzék

- 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/tekkonyvek.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>.