УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Милош З. Јовановић

АУТОМАТСКО ГЕНЕРИСАЊЕ АЛГОРИТАМА СТАБАЛА ОДЛУЧИВАЊА ЗА КЛАСИФИКАЦИЈУ

докторска дисертација

Београд, 2016

UNIVERSITY OF BELGRADE

FACULTY OF ORGANIZATIONAL SCIENCES

Miloš Z. Jovanović

AUTOMATIC GENERATION OF DECISION TREE ALGORITHMS FOR CLASSIFICATION

doctoral dissertation

Belgrade, 2016

Ментор:

др Милија Сукновић, редовни професор

Факултет организационих наука, Универзитет у Београду

Чланови комисије:

др Борис Делибашић, ванредни професор

Факултет организационих наука, Универзитет у Београду

др Драган Радојевић, научни саветник

Институт „Михајло Пупин”, Београд

Датум одбране: _____ 2016. године

Аутоматско генерисање алгоритама стабала одлучивања за класификацију

Резиме:

Стабла одлучивања су врло распрострањен модел за класификацију, којим се описује начин како се може предвиђати класа неког објекта (нпр. добар/лош клијент, болестан/здрав пацијент, позитиван/негативан текст, итд), на основу доступних атрибута тих објеката. Постоји више алгоритама у литератури за прављење стабла одлучивања из података, а на различитим проблемима су други алгоритми оптимални.

Развојем компонентних алгоритама уочава се заједничка структура ових алгоритама, где се различити алгоритми описују различитим избором компоненти за одређених потпроблема, док је генерални ток извршавања исти. Оваквим приступом се омогућава и размена делова решења из више алгоритама и лако прављење хибридних алгоритама који комбинују добре аспекте из више оригиналних алгоритама, и тиме омогућавају боље прилагођавање алгоритма проблему који се решава, што експерименти потврђују. Такође се омогућава и боље разумевање функционисања алгоритма, насупрот имплементацијама у виду црних кутија.

Компонентни алгоритми отварају комбинаторни простор могућих алгоритама, чијом претрагом би се аутоматски саставио алгоритам из постојећих делова, који најбоље решава дати проблем предвиђања (класификације). У овом раду се истражује могућност претраживања простора алгоритама метахеуристиком еволуционих алгоритама. Експерименталном провером су испитани ефективност и ефикасност овог начина претраге, спровођеном на више јавно доступних података, и поређењем са познатим бенчмарк алгоритмима. Експерименти показују да су генерисани алгортими бољи од постојећих познатих алгоритама, једнаки или близу оптималним хибридним алгоритмимаиз из целог простора, као и да често имају боље перформансе у односу на раније пријављене резултате.

Анализом популације алгоритама која еволуира се такође може стећи увид у то који делови алгоритма су одговорни за добре перформансе. Поред експеримената на јавно доступним подацима, приказује се и примена на решавање проблема предвиђања успеха студената.

Цео приступ је имплементиран као модул за популарну платформу за откривање законитости у подацима (*RapidMiner*), чиме се омогућава лако коришћење овог генератора алгоритама, уз могућност сужавања простора претраге и подешавање додатаних параметара претраге.

Кључне речи: стабла одлучивања, алгоритми, компоненте, еволуциони алгоритам, метахеуристика, простор алгоритама, класификација

Научна област: Организационе науке

Ужа научна област: Моделирање пословних система и пословно одлучивање

УДК број: 519.816

Automatic generation of decision tree algorithms for classification

Abstract:

Decision trees are very popular classification model, which describes a way to predict classes of an object (e.g. good/bad client, sick/healthy patient, positive/negative text), based on available attributes of the given object. There are several algorithms in the literature, for creation of decision trees from data, and for different problems different algorithms are optimal.

With the development of component-based algorithms, a common structure of these algorithms emerges, where different algorithms are described by different choice of components that solve specific sub-problems, while the general execution flow is the same. This approach enables sharing of parts of solutions from different algorithms, and easy creation of hybrid algorithms, that combine good aspects of several original algorithms, and thus enable better adaption of algorithms to the problem at hand, which is supported by experiments. Also, this approach enables better understanding of the inner functioning of the algorithm, in contrast to implementations as black-boxes.

Component-based algorithms open a combinatorial space of possible algorithms, where search over this space can automatically construct an algorithm from existing parts, which best solved the given prediction (classification) problem. This thesis explores the possibility to search the algorithm space, using the metaheuristic of evolutionary algorithms. Experiments are conducted to examine the efficiency and the effectiveness of this search method, on multiple publicly available datasets, and comparing to well known benchmark algorithms. Experiments show that generated algorithms are more accurate than well known ones, equal or close to optimal hybrid algorithms from the whole space, and often give better performance than previously reported results.

The analysis of the population of algorithms that evolves can also give insights to which parts of algorithms are more responsible for good performance of the overall algorithm. In addition to experiments on publicly available data, an application is presented that solves the problem of predicting success of students.

The approach is implemented as a module for a popular data mining platform (*RapidMiner*), which enables easy usage of this algorithm generator, with the optional restriction of search space and setting of additional search parameters.

Keywords: decision trees, algorithm, components, evolutionary algorithm, metaheuristic, algorithm space, classification

Scinetific field: Organizational Sciences

Field of scientific expertise: Business system modeling and business decision making

UDK code: 519.816

Садржај

1. Увод

Откривање законитости у подацима - ОЗП (енг. *data mining*) је дисциплина одлучивања која има за циљ да развије моделе реалних система на основу велике количине доступних података из тих система. Такви модели описују комплексне зависности између измерених сигнала у неком систему и помогли да се те зависности боље разумеју. Такође, модели изграђени из историјских података се често користе и за предвиђање будућих стања система, у циљу подршке процесу одлучивања.

Један од основних задатака области ОЗП је **класификација**, као проблем разврставања објеката у предодређене класе. Класификација се врши на основу атрибута, који представљају опис објеката који треба класификовати. Алгоритми за класификацију треба да открију модел ("правило") за класификовање које ће правити минималну грешку приликом класификације, а на основу примера добрих класификација. Грешка може да се мери као проценат лоше разврстаних објеката. Овакви модели су корисни јер за нове случајеве могу предвидети класу неког објекта (нпр. добар/лош клијент, болестан/здрав пацијент, позитиван/негативан текст, итд.)

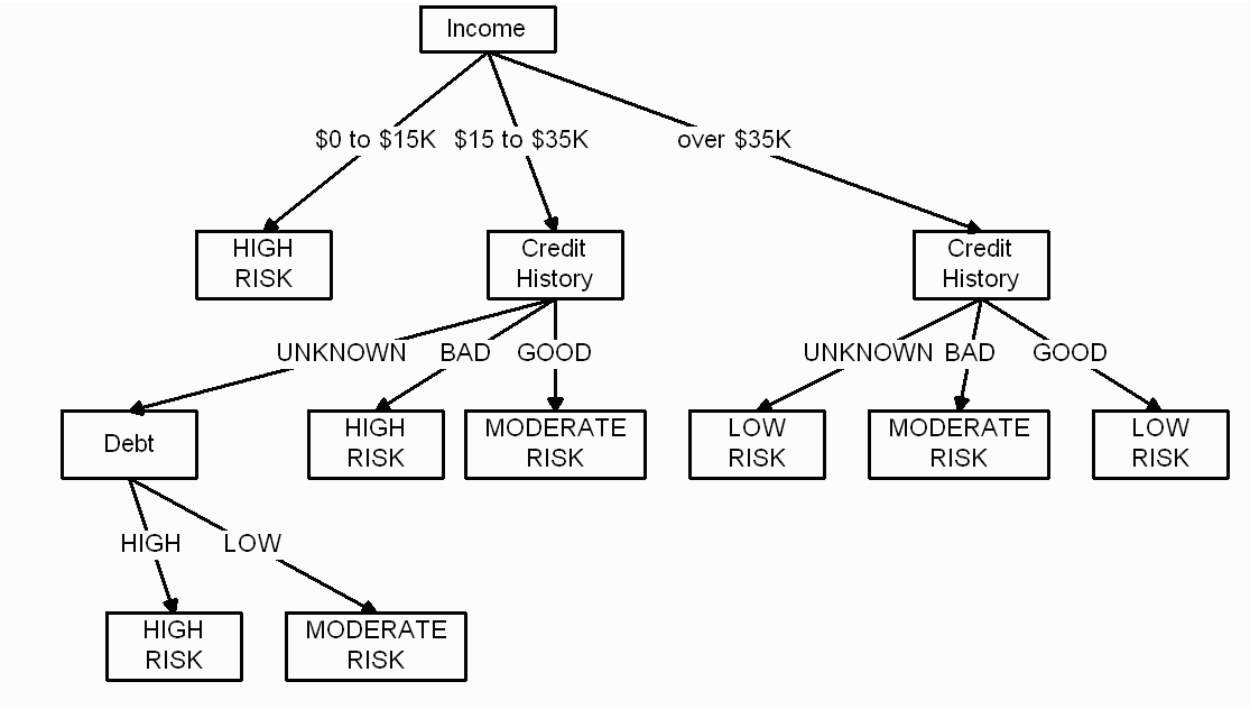
Модели ("правила") класификовања, који се генеришу оваквим алгоритмима, могу бити статистички, логички, алгоритамски, или другог облика, а базирани су на предходним (историјским) случајевима, из којих алгоритми "уче" те правилности исказане у моделу.

Једни од најраспрострањенијих модела класификације су и **стабла одлучивања**. Они хијерархијском структуром дефинишу правила када неки објекат треба да припадне некој класи. Пример стабла одлучивања за класификацију клијената банке по ризичности је дат на Слици 1. Понекад се називају и "индуктивна стабла одлучивања", да би се нагласило да се стабла креирају на основу историјских случајева, који представљају примере добре класификације. Изграђена стабла треба да открију правила којима се ти историјски примери најбоље репродукују, тј. да индуктивно дођу од примера

до "знања" (модела). (Tsiptis и Chorianopoulos, 2009) описују предности стабла одлучивања и разлоге због којих су често коришћени у пракси:

- генеришу једноставна, разумљива правила, лака за тумачење;
- алгоритми су брзи и скалабилни и ефекасно могу да раде са великим бројем случајева и атрибута;
- интегришу процес избора атрибута, који им омогућава да сузе скуп разматраних атрибута, избацујући (не разматрајући) небитне атрибуте;
- нису спутана корелацијама атрибута у подацима, тј.присуством мултиколинеарности.

У литератури постоји прегршт извештаја о успешној примени стабла одлучивања. Добар преглед разноликости и количине примена је дат у (Murthy, 1998), где се цитирају радови примена у агрикултури, астрономији, биомедицији, финансијама, процесирању слика и звука, правима, медицини, производњи, молекуларној биологији, фармацији, физици, развоју софтвера, итд. Нешто скорије студије наводе и додатне примере употребе стабла одлучивања који, поред тога што су широки у броју области у којима се појављују, су и врло бројни (Almuallim et al. 2001) (Rokach, 2008).



Слика 1: Пример стабла одлучивања за класификацију клијената банке по ризичности

Постоји више **алгоритама** за креирање стабла одлучивања, од којих су можда најпознатији алгоритми CART (Breiman et al., 1984), ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993) и CHAID (Kass, 1980). Ови алгоритми, иако доста стари, су тренутно најкоришћенији у пракси, што сведочи њихова присутност у готово сваком софтверу за ОЗП. Такође су се два алгоритма пронашла у истакнутој студији о *10 најкоришћенијих алгоритама за ОПЗ* (Wu et al. 2008).

Сваки од ових алгоритама дефинише и скуп **параметара**, чијом изменом се мења и добијено стабло одлучивања из података. На тај начин се може контролисати величина и комплексност добијеног стабла. Превише једноставно стабло најчешће не може довољно добро да опише правила сврставања објеката у класе, док превише комплексно стабло може бити нерепрезентативно, тј. да важи само на подскупу података из којих је стабло генерисано (такозвани проблем *претренирања*). Иако је познат начин на који многи параметри алгоритма утичу на комплексност добијеног стабла, није јасно какав ће бити ефекат на укупну прецизност стабла, тј. могућност да прецизно предвиђа класе нових објеката.

Параметри алгоритма су изложени корисницима и очекује се да они знају да подесе те параметре како би добили најбоље резултате. Како не постоји јасан начин како се то изводи, аналитичари најчешће испробавају разне комбинације параметара док не добију специфично стабло које најпрецизније предвиђа класе. У скорије време се јављају и различите оптимизационе процедуре, којима се аутоматски одређују добре вредности параметера алгоритама за изградњу стабла одлучивања. Генерално важи да што је већи простор параметара (тј. што више комбинација могућих вредности има), то је могуће добити бољи модел за предвиђање. У исто време, проблем подешавања ових параметара постаје све комплекснији. Стратегија је због тога најчешће да се ограничи простор параметара да би се смањила комплексност проблема, мада постоји и други пут, а то је да се аутоматизује избор вредности параметара.

У литератури се могу наћи на десетине **варијација алгоритама** за стабла одлучивања. Да је толики број различитих алгоритама потребан потврђују и такозване теорије да "нема бесплатног ручка" (eng. *No free lunch theories*) (Wolpert, 1996), које математички доказују да се за сваки изабрани проблем може наћи алгоритам који даје најбоље резултате (је оптималан) на том проблему, док на другим проблемима тај алгоритам не мора бити уопште добар. Овакви докази су учинили да истраживачи одустану од потраге за једним најбољим алгоритмом. То заправо указује да, уместо да се тражи најбољи алгоритам уопште, потребно је да постоји велики број алгоритама који ће бити добри у појединим случајевима.

Један начин да се повећа број могућих алгоритама, и тако омогући прављење специјализованих алгоритама за неки проблем, је приступ **алгоритама заснованих на компонентама** (енг. *component-based algorithms*). У том приступу је могуће, кобиновањем делова алгоритама (тј. компоненти), добити знатно већи број алгоритама. Овај приступ је већ успешно примењен на алгоритме стабла одлучивања (Suknović et al, 2012) (Delibašić et al, 2011a). Поред тога, овакве алгоритме је могуће боље разумети, јер су видљиви и разумљивији унутрашњи делови алгоритма, због чега је овај приступ назван

прицнип "белих кутија", као контраст традиционалном начину прављења алгоритама као "црних кутија" (Delibašić et al, 2012a) (Delibašić et al, 2013). Такође је могуће одредити који делови алгоритама су најзаслужнији за добре перформансе неког алгоритма. Овај приступ је објашњен у више детаља у Поглављу 3.

Велики проблем остаје како наћи прави избор компоненти алгоритма, који ће сачинити добар алгоритам за проблем који се решава. Ово посебно постаје тешко када број компонената расте, јер укупан простор алгоритама тада убрзано расте. Решавањем овог проблема би се искористио знатно већи простор могућих алгоритама, што би резултовало бољим резултатима у сваком специфичном случају примене. Један од могућих начина за решавање овог проблема је аутоматска претрага простора алгоритама, постојећим оптимизационим приступима, који је главна тема овог рада.

1.1. Предмет истраживања

Предмет истраживања су **алгоритми** за изградњу **стабла одлучивања**, као модела за **класификацију**, као и **аутоматско генерисање** варијација основних алгоритама, оптимизацијом комбинације компонената тих алгоритама. Како све могуће варијације алгоритама дефинишу „простор алгоритама“, предмет ове тезе се може дефинисати и као **претрага простора алгоритама** који расте супер-линеарно са бројем могућих компоненти.

Алгоритми се оцењују преко своје могућности да креирају стабла која прецизно предвиђају класе објеката од разматрања, што се може егзактно мерити (Pang-Ning et al, 2006), а биће коришћено за поређење различитих алгоритама и дискусију о утицају појединих фактора на тај квалитет.

Не изучава се конкретан домен примене, већ ће се на подацима из доста специфичних домена илустровати употребљивост овог приступа и поредити са постојећим приступима.

Овакав проблем још није директно решаван у литератури, али постоје слични и аналогни проблеми који јесу били предмет истраживања, и дискутују се у засебној секцији овог рада.

1.2. Циљеви истраживања

Циљ је дакле аутоматизација избора најбоље комбинације делова алгоритма за стабла одлучивања. Тиме се добије својеврстан систем за подршку одлучивању, који омогућава:

- да се генерише бољи алгоритам који одговара неком конкретном проблему примене;
- да се боље разуме који делови алгоритма су одговорни за добре перформансе, као и које компоненте добро раде заједно;
- да се даљим развојем компоненти за алгоритме стабла одлучивања омогући испитивање употребљивости нових компоненти;
- да се олакша и значајно убрза рад аналитичара приликом примене стабла одлучивања.

(Wolpert, 1996) је већ показао да већим избором алгоритама постоји и већи потенцијал за специфичним решењима која раде боље од генералних решења. (Suknović et al, 2012) и (Delibašić et al, 2011) су дали начин како се може знатно проширити простор могућих алгоритама, додавањем нових компоненти. Наредни корак треба да омогући кретање кроз такав велики простор алгоритама, у потрази за најприхватљивијим у сваком појединачном случају.

Да би се испунили ти циљеви, у раду ће као доприноси бити приказани:

- преглед литературе о алгоритама за изградњу стабла одлучивања за класификацију;

- детаљан преглед концепта компонентних алгоритама за стабла одлучивања, имплементације и експерименталне подршке том приступу;
- дефинисање простора алгоритама и проблема који он представља;
- изучавање могућности за аутоматско генерисање алгоритама за стабла одлучивања, као и параметризацију компоненти, коришћењем еволутивних алгоритама за претрагу;
- експериментална провера ефективности и ефикасности претраге простора алгоритама;
- анализа конструисаних алгоритама на специфичним применама стабла одлучивања, како би се извршила карактеризација и опис "добрих" компоненти;
- изградња софтверског алата који би служио као подршка одлучивању, аналитичарима који примењују алгоритме за стабла одлучивања, што би значајно олакшало и убрзало примену.

1.3. Хипотезе

На основу задатих циљева, проучавања предходних истраживања из релевантне литературе и њихових закључака, могу се дефинисати хипотезе, као претпоставке које се могу експериментално верификовати.

Основна хипотеза је да се **простор алгоритама креиран разменом компоненти (делова) алгоритама може ефективно и ефикасно претраживати**. Верификовањем ове хипотезе би се отворио простор за аутоматско генерисање и других алгоритама (осим стабла одлучивања) представљених компонентама.

Даље, претпоставља се и да **пронађени оптимални (или субоптимални) алгоритми могу бити значајно бољи од постојећих алгоритама**. Верификовањем ове хипотезе ће бити дат и јак практичан мотив за коришћење овог приступа.

Коначно, верује се да перформансе алгоритма не зависе подједнако од свих делова (компоненти). **Анализом аутоматски конструисаних алгоритама на неком проблему, могуће је открити који делови алгоритма су заслужнији за његове перформансе**.

1.4. Методе истраживања

Наведене хипотезе ће бити верификоване строгом научном методологијом, заснованом на емпиријској (експерименталној) потврди.

Прво, прегледом и **проучавањем литературе, критичким освртом на њу и систематизацијом** треба да се стекне јасна слика о проблемима и потенцијалним решењима за реализацију задатих циљева. Изучаваће се алгоритми за стабла одлучивања, структура тих алгоритама и постојећа унапређења, са фокусом на компонентни дизајн за стабла одлучивања. Посебно ће се изучавати оптимизациона техника за глобалну оптимизацију путем еволуционих алгоритама, као мета-хеуристички метод за оптимизацију. Поред литературе, проучаваће се алат за изградњу компонентних стабала одлучивања и могућност надградне овог алата са аутоматским генерисањем алгоритама.

Треба јасно **дефинисати егзактне мере квалитета** алгоритма, на начин који би омогућио **експериментално поређење са другим приступима** у литератури. Процена колико добијено стабло одлучивања добро предвиђа класе ће се вршити искључиво на примерима који нису коришћени приликом изградње стабла, како би се мерила генерализација добијеног модела класификације.

Квалитет алгоритама ће се проверавати и на добро познатим (**бенчмарк**) проблемима, како би се резултати упоредили са постојећим приступима. Ово такође треба да омогући **транспарентност и поновљивост** свих експеримената. Алгоритми ће се **проверавати на што већем броју примера**, како закључци не би били појединачни, већ генерални.

Експерименти ће се спроводити на истој, јасно дефинисаној, хардверској инфраструктури, како би поређења била **фер и коректна**.

Оптимизационе методе ће се оцењивати са тиме колико је решење близу оптималном, као и колико времена је потребно за извршавање.

Подаци од спроведених експеримената ће бити анализирани и биће извучени **закључци**, **објашњења**, као и дискутована **нова истраживачка питања** која ће се тиме отворити.

2. Стабла одлучивања за класификацију

Стабла одлучивања представљају модел процеса доношења одлука, који у корацима, кроз ***гране*** стабла, описује на који начин се користе информације да би се дошло до неке одлуке, представљене у ***листу*** стабла одлучивања.

Стабла одлучивања могу ручно дефинисати експерти из личног искуства, или се могу изградити из података без присуства експерта (такозвана "индуктивна стабла одлучивања"). У овом раду се изучава приступ изградње стабла одлучивања из података, што припада области машинског учења и откривања законитости у подацима (енг. *data mining*).

Ради илустрације проблема и процеса решавања, послужићемо се примером из (Tan et al, 2005), који описује десет ***инстанци*** клијената банке кроз неколико особина или ***атрибута*** (Табела 1).

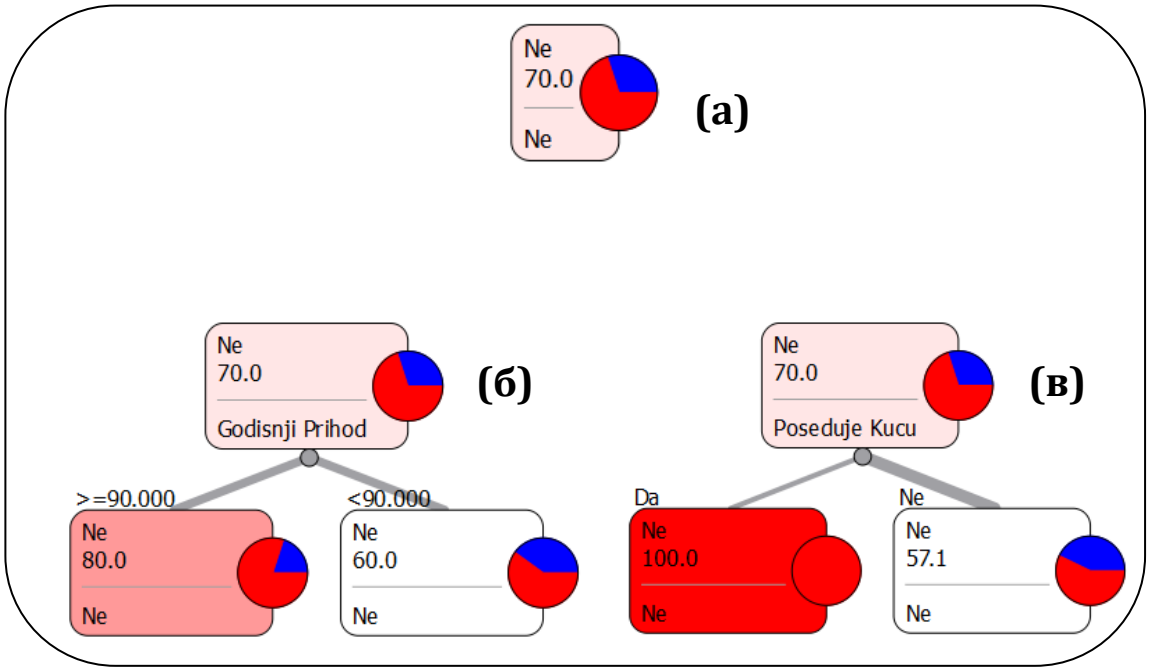
Табела 1. Илустративни пример: класификација клијената банке

РБр	Поседује кућу	Брачни статус	Годишњи приход	Неизвршене обавезе
1	Да	Сам	125К	**Не**
2	Не	У браку	100К	**Не**
3	Не	Сам	70К	**Не**
4	Да	У браку	120К	**Не**
5	Не	Разведен	95К	**Да**
6	Не	У браку	60К	**Не**
7	Да	Разведен	220К	**Не**
8	Не	Сам	85К	**Да**
9	Не	У браку	75К	**Не**
10	Не	Сам	90К	**Да**

Последњи атрибут је специјалан јер представља информацију коју немамо за нове клијенте (не знамо да ли ће имати проблема са извршавањем обавеза). Циљ је да изградимо модел којим можемо предвидети вредност овог атрибута за нове клијенте, а на основу информације о осталим атрибутима.

Тај атрибут зваћемо излазни атрибут или ***класа***, пошто на основу њега можемо разликовати две (у овом случају) класе клијената од интереса. Стога се и проблем предвиђања класе објеката назива ***класификација***.

Уколико би класа свих инстанци била иста, проблем класификације би био тривијалан, јер тривијалан модел који увек сврстава објекте у ту класу никада не би погрешио. Иако такву идеалну ситуацију не можемо очекивати у реалним проблемима, циљ изградње стабла одлучивања је да се подаци некако поделе на подскупове, у којима ће у сваком подскупу бити тривијално рећи којој класи припадају објекти. У нашем примеру имамо случај да седам објеката (клијената) припада класи клијената који успешно враћају кредит, а три објекта класи која има проблеме са подмиривањем дугова. Иницијална дистрибуција класа дата је на Слици 2а.



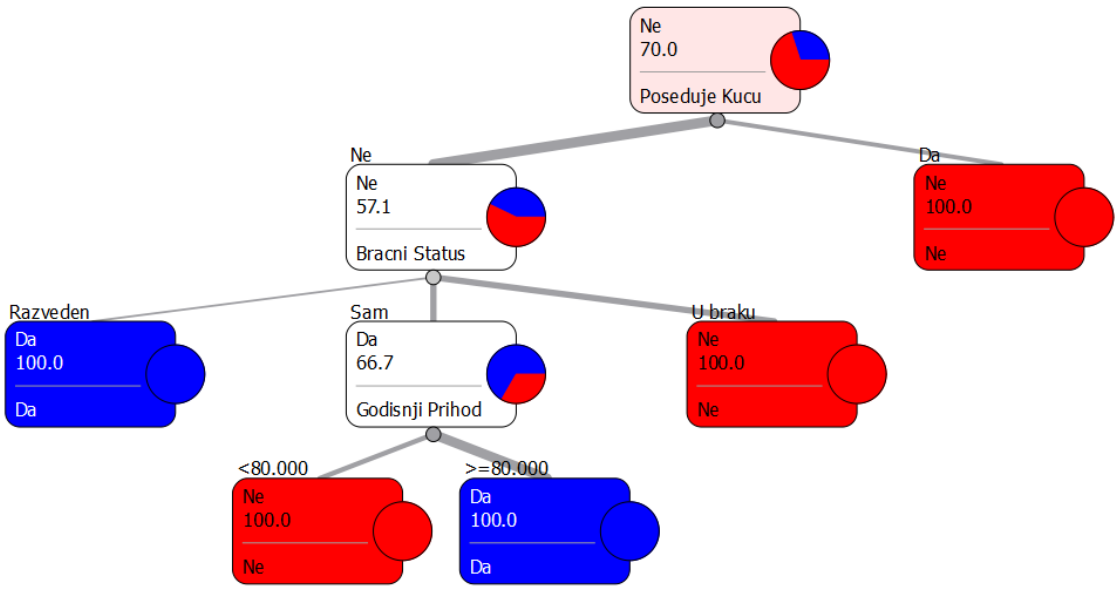
Слика 2. Дистрибуција класа за пар могућности гранања стабла

Циљ је дакле да некако поделимо клијенте да у сваком делу буде заступљена само једна класа, или што ближе том идеалу. На пример, људе можемо поделити на основу прихода, где би прва група били клијенти који зарађују мање од 90К долара годишње, а друга група они који зарађују више од тога. Волели бисмо да у првој групи буду сви клијенти који (на пример) не враћају

кредит, а да у другој групи сви враћају кредит. Онда би на основу информације о примањима могли да одлучимо коме да одобримо кредит. Нажалост, информација о примањима није потпуна да одреди да ли је неко способан да врати кредит, што видимо из условних вероватноћа, датих на Слици 2б.

Ако бисмо тестирали све атрибуте, испоставља се да највише информација носи атрибут "поседује кућу", јер ***гранањем*** (дељењем) клијената на те две групе ("поседује" и "не поседује") се добија најчистија дистрибуција класа у свакој групи (Слика 2в).

Тачан начин ***евалуације (мерења) квалитета*** сваког атрибута ће бити детаљно обрађиван у наредним поглављима. С обзиром да у свакој од грана немамо чисту ситуацију расподеле класа, овај поступак можемо рекурзивно поновити за сваку новонасталу грану, све док не добијемо чисту ситуацију, тј. подгрупу клијената за које увек можемо рећи да враћају или не враћају кредит. Након целог поступка гранања са расположивим атрибутима, добија се следеће резултујуће стабло, дато на Слици 3.



Слика 3. Пример коначног стабло за класификацију ризичности клијнта.

Произведено стабло можемо користити као знање које је проистекло из доступних података, на основу ког можемо **класификовати нове клијенте** на оне који враћају или не враћају кредит.

Основна претпоставка је да су дати **подаци репрезентативни за популацију** клијената. Иако се у овом примеру то не може тврдити због малог узорка, у реалним случајевима најчешће имамо на стотине, хиљаде, па и милионе случајева (инстанци) доступних за анализу. Оваква анализе стога спадају у домен откривања законитости у подацима што је у контрасту са класичним статистичким обрадама код којих је мали узорак типична појава и основни проблем анализе.

У приказаном примеру треба приметити и чињеницу да су коришћени **атрибути различитог типа**, наиме појављује се један бинарни атрибут ("Поседује кућу"), један категорички ("Брачни статус"), као и један нумерички атрибут ("Годишњи приход"). То илуструје робустност стабала одлучивања у вези врсте улазних атрибута, коју многи статистички модели немају, јер захтевају конверзију у један од типова података.

Резултујући модел стабла се такође може тумачити као статистички модел, јер моделује условне вероватноће класа, када су дате вредности улазних атрибута.

Додатно, уз процену условних вероватноћа, овај модел уједно и одређује структуру зависности променљивих, као и **селекцију атрибута** који носе важне информације за класификацију. Без тог важног аспекта, једноставно рачунање свих услових вероватноћа не би било изводљиво, што због времена срачунавања, што због недовољно података за процену вероватноћа када је дат већи број атрибута.

Постоје бројни алгоритми за изградњу стабала одлучивања који су обрађивани у литератури. Ови алгоритми се разликују у многим детаљима, о чему ће бити речи у наредним поглављима, али су такође и сличности велике, што ће омогућити генерализацију алгоритама приказану у Поглављу

3. Треба само нагласити да се у овом раду обрађују стабла одлучивања за класификацију. Слични модели се изучавају и ради прављења модела за регресију (тј. предвиђање нумеричких излазних вредности), или анализу ризика, али ти модели неће бити обрађивани у овом раду.

У наставку овог поглавља ће бити приказани постојећи алгоритми за изградњу стабала одлучивања за класификацију. Након приказа дискутоваће се поређење алгоритама по више битних аспеката, уз осврт на имплементационе разлике и доступност у софтверима. Најзад, биће приказане и дискутоване актуелне модификације постојећих алгоритама из литературе.

2.1. Алгоритми стабала одлучивања из литературе

Стабла одлучивања су почела да се изучавају седамдесетих година прошлог века, упоредо са развојем рачунарства. Заправо концепт стабла одлучивања је и старији, и обухвата моделе који су људи ручно састављали, како би визуализовали процесе доношења одлука. Таква ручно изграђена стабла имају велику примену у анализи ризика, где се моделују неизвесни исходи и избори алтернативних путева за решавање неког проблема. О таквим стаблима се више може прочитати у (Lee et al., 1981) (Čupić, Suknović, 2008), али она нису предмет овог истраживања, а овде се помињу због тога да би терминолошки били једнозначно одређени. Врсте стабла одлучивања која се изучавају у овом раду су „индуктивна“ стабла одлучивања, тј. она која се генеришу из података, а не записом експертских знања. Надаље ће се термин *стабла одлучивања* односити на индуктивна стабла одлучивања.

Међу запаженијим алгоритмима у литератури су сигурно и ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), CART (Breiman et al, 1984) и CHAID (Kass, 1980), и они су присутни у великом броју софтверских алата за откривање законитости у подацима (ОЗП). Такође су препознати у студији о 10 најпопуларнијих алгоритмима за ОЗП (Wu et al, 2008). Детаљнији опис ових

алгоритама, са примерима рачунања, се такође може наћи и у (Delibašić et al, 2009a).

2.1.1. *ID3* алгоритам

Принцип који прате сви алгоритми за изградњу стабла је да итеративно деле скуп података на подскупове (на основу изабраних атрибута), тако да је у тим подскуповима што лакше одредити класу објеката. Тај принцип се може видети у ID3 алгоритму (Quinlan, 1986):

1. Одредити почетну ентропију (неуређеност) система на основу излазног атрибута (класе)
2. За сваки улазни (не-класни) атрибут:
    a. поделити скуп на делове који одговарају различитим вредностима изабраног атрибута (гранање)
    b. израчунати очекивану ентропију у подељеном скупу, као отежани просек ентропија сваког подскупа
    c. разлика између почетне ентропије и очекиване ентропије представља информациону добит, тј. количину информација коју носи атрибут за класификацију
3. За гранање стабла изабрати онај атрибут који максимизира информациону добит
4. За сваку новодобијену грану, а уколико грана има преостале ентропије, вратити се на корак 1 (рекурзија). У супротном, зауставити алготитам.

У овом алгоритму се користи концепт ентропије, из информационе теорије (Shannon, 1948), као мера колико је тешко сврстати елементе скупа у класе, а рачуна се као:

$$H(S) = \sum_{i}^{K} p_i \log_2 p_i$$

где је $H(S)$ ознака за ентропију скупа S, p_i вероватноћа да елемент припада i-тој класи, а К укупан број класа.

Јасно је да ће ентропија износити 0 једино када је скуп „чист“, тј. када сви објекти припадају истој класи. Ово је идеална ситуација, јер је сврставање у класе тривијално, и за будуће објекте са истим особнама можемо тврдити да припадају тој класи. Ентропија је максимална када су све класе подједнако вероватне, што одговара најнеповољнијој ситуацији за предвиђање класе објекта.

Дакле, концепт ентропије се у овом алгоритму користи као ***Мера евалуације*** предложеног гранања (дељења) скупа података. Касније ћемо видети да различити алгоритми користе различите мере да оцене по ком атрибуту треба гранати стабло у сваком кораку гранања. Да би оценио најбоље гранање, алгоритам прво рачуна укупну ентропију излазног атрибута (класе), што бројчано мери колико је тешко одредити класу без додатних информација. Затим се рачуна условна ентропија, тј. очекивана ентропија ако би се скуп поделио у подскупове (гране) на основу неког атрибута:

$$H(X,S) = \sum_{j}^{|X|} P\big(X = x_j\big)\, H\big(X = x_j\big)$$

где је $|X|$ број различитих вредности атрибута X, а $H\big(X = x_j\big)$ ентропија подскупа података где је атрибут X има вредност x_j.

Разлика почетне и ове условне ентропије представља **информациону добит** (енг. *information gain*), тј. колико ентропије је смањено од излазног атрибута, ако знамо вредности неког улазног атрибута X:

$$IG(X,S) = H(S) - H(X,S)$$

Атрибут који има највећу информациону добит се бира за гранање стабла, као одраз похлепне (енг. *greedy*) хеуристике смањивања ентропије, тј. могућности за лакше предвиђање класе.

Вредност информационе добити је изражена јединицом мере **бит**, због коришћења основе 2 у израчунавању логаритама у формули за ентропију. Уколико би се користиле друге основе, избор гранања би остао исти, али би јединице мере биле другачије (*нат* за природни логаритам, *бан* за основу 10, итд.) Ова мера евалуације потиче из информационе теорије, а измерена ентропија се такође може протумачити као очекивани број битова потребан за кодирање поруке о вредности класе. Уколико је енропија 0, онда је порука о класи непотребна, јер се увек појављује иста класа.

Једна од карактеристика овог алгоритма је да грана стабло на онолико грана колико има вредности атрибута *X*, што се назива вишеструко (енг. *multiway*) гранање. Ово ствара и малу контраст са називом алгоритма, где *ID3* је акроним од *iterative dihotomizer t(h)ree*, јер овакав избор вишеструког гранања више прави стабла која су полихотомна него дихотомна. Други алгоритми који ће бити приказани понекад више заслужују такав назив.

Алгоритам рекурзивно понавља дељење подграна са атрибутом који је у том скупу најбољи за разликовање класа. Алгоритам се зауставља када је у грани ентропија снижена на 0 (где је предвиђање тривијално) или када за даље гранање не постоји више атрибута, јер су сви искоришћени у предходним гранањима на путу ка корену. Додатно се може ограничити максимална дубина стабла, како би класификација радила на мањем броју упита о вредностима атрибута.

2.1.2. *C4.5* алгоритам

Овај алгоритам представља унапређену верзију ID3 алгоритма и дело је истог аутора (Quinlan, 1993). Основни алгоритам је промењен на више начина:

Нумерички атрибути

Прво унапређење односи се на могућност да алгоритам **грана стабло по атрибуту нумеричког типа**. За атрибуте који могу узети бесконачно много различитих вредности на реалној оси, није јасно како треба поделити стабло

на коначно грана. Прво, овај алгоритам се ограничава на бинарно гранање $X \leq m$ и $X > m$, за изабрану границу гранања m. Остаје да се нађе оптимална граница, што се ради на следећи начин. Све вредности нумеричког атрибута које су виђене у подацима се сортирају у неопадајући низ, и као потенцијалне тачке гранања се узму вредности између сваке две вредности из низа. Тиме се добија $n-1$ потенцијалних тачака гранања, где је n величина података (број инстанци), што је и даље превелики број за испитивање. Да би се овај број смањио, задржавају се само оне тачке гранања између којих је дошло до промене у класи у подацима за тренирање. На овај начин се знатно редукује број потенцијалних тачака гранања, без губитка тачности, јер ентропија (и друге мере нечистоће чворова) монотоно опада када год се пар тачака исте класе нађу у истој грани. На тај начин се ефикасно проналази оптимална тачка гранања m за изабрани нумерички атрибут.

Мера евалуације – Рацио добити

Даље, овај алгоритам уводи **додатну меру евалуације квалитета гранања**. Информациона добит, коју користи *ID3* алгоритам, има недостатак да даје предност гранањима која гранају стабло у више (у односу на мање) грана. Тако атрибути са више могућих вредности имају вештачку предност. Из погледа информационе теорије, ово је оправдано јер атрибути са више вредности носе више информација, јер сужавају на мању пропорцију могућих стања. Ово се може видети и математички из формуле ентропије (Поглавље 2.1.1), где ентропија расте са бројем вредности K, нпр. максимална ентропија за 2 вредности је 1, за 4 вредности износи 2, док за атрибут са 8 вредности износи 3. Иако је то оправдано са информационе теорије, када правимо стабло одлучивања, ми желимо да дамо подједнаку шансу и атрибутима са природно мање категорија (попут пола особе). У C4.5 алгоритму ово се решава увођењем мере Рацио добити (енг. *Gain ratio*), уместо информационе добити:

$$GR(X, S) = \frac{IG(X, S)}{H(X)}$$

Делилац у овој формули је ентропија атрибута који треба да грана стабло и он је већи са већим бројем могућих вредности, па с тиме ублажава квалитет гранања атрибутом са већим бројем вредности. Ово се испоставља и веома битним у присусву атрибута који су идентификатори (тј. јединствени за сваку инстанцу) или врло блиски идентификаторима, јер би информациона добит такве атрибуте прво преферирала, а они не носе корисне информације за класификацију, док рацио добити не упада у ту замку.

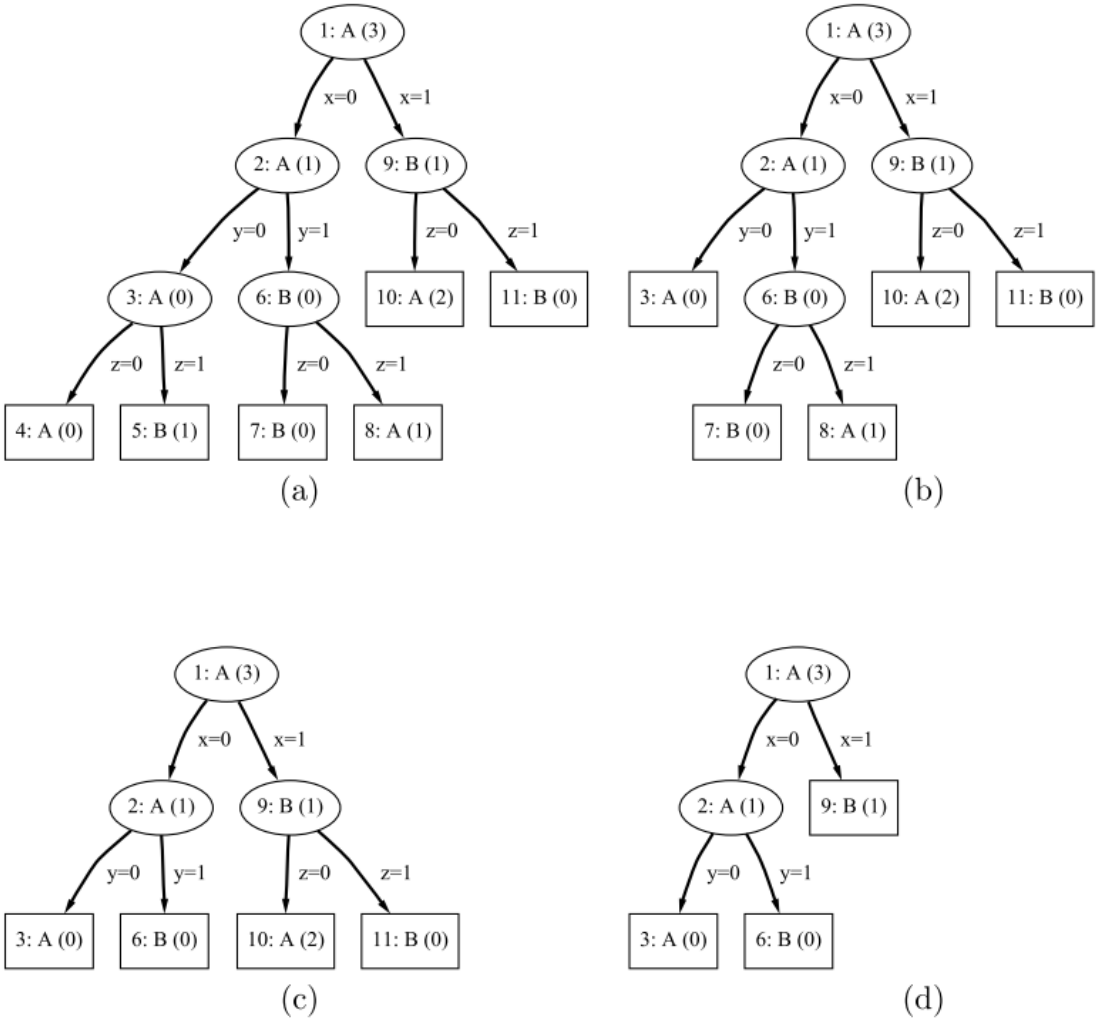
Поткресивање (орезивање) стабла

Најзад, ID3 алгоритам врши гранање док не смањи ентропију на минималну могућу меру, како би најбоље разликовао класе. Ово може бити опасно јер са превише разгранавања добијамо стабло чији су листови садрже врло мало инстанци, па могу бити нерепрезентативни за будуће случајеве које желимо да класификујемо. Зато треба наћи добар однос између ниске ентропије и мање комплексности (разгранатости) стабла, како би правила које стабло индукује била подржана са више примера (инстанци). Интуиција је да желимо најједноставније стабло које што боље препознаје класе објеката. Једноставност се добија **поткресивањем (или орезивање) стабла** (енг. *pruning*), које се спроводи након изграђеног стабла, и које треба да елиминише сувишна гранања која не производе значајна побољшања а уводе комплексност у модел.

C4.5 користи такозвано *песимистичко поткресивање*. Очекивана вероватноћа грешке у сваком чвору се процењује са $P_e = \frac{e}{N}$, при чему је e број начињених грешака, а N број инстанци у том листу. Због тога што скуп за тренирање може бити нерепрезентативан (посебно мањи подскуп у једном листу стабла), за процену грешке се узима песимистичка процена P_{e^*} за коју важи да је вероватноћа да $P_e > P_{e^*}$ буде мала (5%, 10% или 25%). Под претспоставком да су грешке независне, број грешака e се расподељује по биномној расподели, па је горњу границу грешке P_{e^*} је лако одредити на основу дефинисане мере поверења (типично 25%).

Алгоритам онда проверава песимистичку грешку за неко подстабло и такође мери колика би била песимистичка грешка ако би уместо тог подстабла био само лист (тј. ако би се предвиђала једноставно већинска класа у том подстаблу). Уколико је песимистичка грешка листа мања, подстабло се мења за лист. Ефекат је да уколико подстабло је дефинисано на врло мало инстанци, песимистичка грешка ће бити већа (због варијансе биномне расподеле), па неће бити оправдано да се задржи цело подстабло, већ се оно мења листом.

Пример оригиналног, као и стабла поткресаног са неколико интензитета дати су на Слици 4.



Слика 4. Визуелизација (a) оригиналног и (b-d) поткресаних стабала (Frank, 2000)

Овај алгоритам има и других предности, попут трансформације стабла у низ „ако-онда“ правила, рада са недостајућим вредностима и друго.

Алгоритам

Алгоритам се може описати следећим корацима:

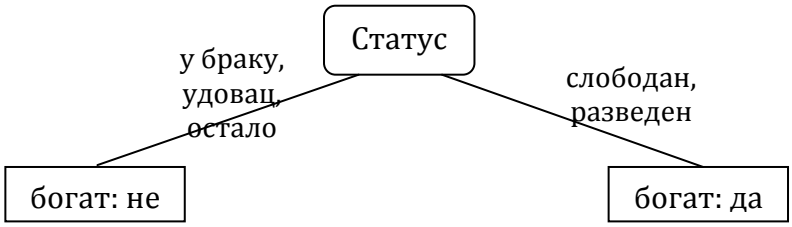
1. Одредити почетну ентропију (неуређеност) система на основу излазног атрибута (класе)
2. За сваки улазни (не-класни) атрибут:
    a. уколико је атрибут нумеричког типа, одредити потенцијалне тачке гранања
    b. поделити скуп на делове који одговарају различитим вредностима изабраног атрибута (гранање). За нумеричке атрибуте скуп се дели на основу правила $X \leq m$ і $X > m$, где је m потенцијална тачка гранања
    c. израчунати очекивану ентропију у подељеном скупу, као отежани просек ентропија сваког подскупа
    d. израчунати информациону добит и Рацио добити за тренутно гранање
3. За гранање стабла изабрати онај атрибут који максимизира Рацио добити
4. Уколико нису задовољени услови заустављања, за сваку новодобијену грану поновити поступак рекурзивно од корака 1. У супротном, зауставити алготитам.
5. Након завршеног гранања комплетног стабла, проћи од листова ка корену сваки чвор стабла и:
    a. измерити песимистичку грешку за тренутно подстабло и за лист који би могао да замени подстабло
    b. уколико је песимистичка грешка листа мања, заменити подстабло са листом

2.1.3. *CART* алгоритам

Овај алгоритам је дело познатих статистичара Лео Брајмана и Џером Фридмена, а акроним је од *Classification and Regression Trees* (Breiman et al, 1984). Разлике, у односу на предходно описане алгоритме, могу се сврстати у три категорије.

Бинарно гранање

Када је потребно гранати стабло на основу атрибута који има више категорија (вредности), предходни алгоритми су гранали тако да свака вредност атрибута добије посебну грану стабла, такозвано вишеструко (енг. *multiway*) гранање. У *CART* алгоритму се у сваком кораку стабло грана на само две гране (бинарно гранање). Стога је потребно одредити које вредности ће припадати левој, а које десној грани. За мањи број вредности, могуће је евалуирати све могуће комбинације бинарног гранања (груписања вредности у леву и десну грану), док се за атрибуте са више могућих вредности најчешће прибегава хеуристикама, попут гранања „један-против-осталих“, где у једну грану иде само једна вредност, а у другу све остале. На Слици 5 се може видети пример бинарног гранања.



Слика 5. Бинарно гранање стабла

Мера евалуације – Џини (енг. *Gini*)

CART користи статистичку меру за процену квалитета изабраног гранања, која има доста сличности са информационом добити, али је утемељена у

статистици уместо информационој теорији. Ђини коефицијент мери очекивану *нечистоћу* чвора, и у сваком чвору се рачуна нечистоћа као:

$$Gini(S) = \sum_{i}^{K} p_i(1 - p_i)$$

где је, као и раније, p_i вероватноћа класе i, од K могућих класа у том чвору. Јасно је да је ова мере нечистоће највећа када су све вероватноће подједнаке, а најмања када је вероватноћа једне класе 1, а свих осталих класа 0. Пошто се део израза $(1 - p_i)$ може протумачити као грешка предвиђања ако би за чвор предвидели класу i, онда цео израз за Ђини коефицијент се може протумачити као математичко очекивање грешке класификације, или $\mathrm{E}(1 - p_i)$.

Укупан квалитет неког гранања се, као и раније, рачуна као очекивани ђини коефицијент свих грана:

$$Gini(X, S) = \sum_{j}^{|X|} P\big(X = x_j\big)\, Gini\big(X = x_j\big)$$

Орезивање на основу цене комплексности

CART алгоритам користи специфичну верзију орезивања стабла, која се зове *орезивање на основу минималне цене комплексности* (енг. *minimal cost complexity pruning*). Идеја је да се пенализује број већи број чворова стабла, па свако гранање (које уводи нове чворове) треба бити оправдано са становишта повећане тачности предвиђања, како би оправдало додатну комплексност коју ствара. Укупна грешка класификације стабла се онда процењује као:

$$\text{Грешка} = \text{Грешка} + \alpha \cdot \text{број\_чворова}$$

где је α параметар који говори колика је казна на повећану комплексност стабла. Варирањем овог параметра се добија низ стабала, оптималних (по

израчунатој грешци) за сваку вредност α, а број тих стабала је коначан. Свако од тих стабала (за различиту вредност овог параметра) се процењује колико је заиста добро, на изолованом тест скупу података.

Алгоритам

Алгоритам се може описати следећим корацима:

1. За сваки улазни (не-класни) атрибут:
    a. уколико је атрибут нумеричког типа, одредити потенцијалне тачке гранања m, које ће гранати на гране: $X \leq m$ і $X > m$
    b. уколико атрибут има више категорија, генерисати могућа бинарна гранања.
    c. за свако потенцијално гранање израчунати очекивану нечистоћу коришћењем ђини коефицијента и изабрати најбоље гранање тренутног атрибута
2. За гранање стабла изабрати онај атрибут који минимизира Ђини коефицијент
3. Уколико нису задовољени услови заустављања, за сваку новодобијену грану поновити поступак рекурзивно од корака 1. У супротном, зауставити алготитам.
4. Након завршеног гранања комплетног стабла, за различите вредности параметра цене комплексности:
    a. за различите вредности параметра цене комплексности, направити низ орезаних стабала
    b. изабрати подстабло које има најбољу грешку мерену на издвојеном скупу података.

2.1.4. CHAID алгоритам

CHAID алгоритам (Kass, 1980) је такође дело статистичара Гордон Каса, и акроним је од *CHiSquare Automatic Interaction Detection*. Он користи

статистичку меру хи-квадрат, која се користи за оцену корелисаности категоричких варијабли, како би оценио повезаност гранања са класама које желимо да предвиђамо.

Мера евалуације

Хи квадрат тест је стандардни тест за утврђивање повезаности категоричких атрибута, а базира се на хи-квадрат статистици, која мери квадратну удаљеност очекиваних и измерених фреквенција појављивања вредности атрибута, а распоређује се хи-квадрат расподелом.

$$\chi^2(X, C) = \sum_{i}^{|X|} \sum_{j}^{|C|} \left(\frac{O_{ij} - E_{ij}}{E_{ij}} \right)^2$$

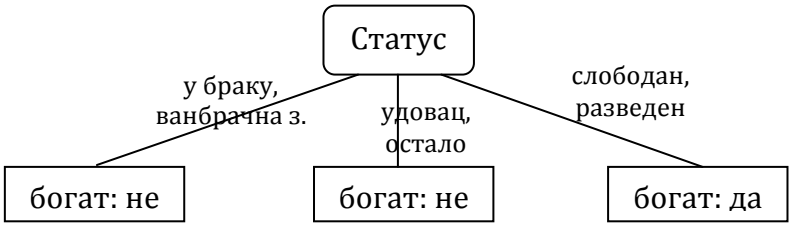
Где је X атрибут, а C класа, између којих се процењује повезаност, O_{ij} је измерен број случајева који имају атрибут i и класу j, док је E_{ij} очекивана вредност на основу маргиналних фреквенција сваке вредности.

Критеријум заустављања из хи-квадрат теста

Уколико неко гранање произведе „чисте“ чворове, тада ће хи квадрат статистика бити јако велика. Добра страна ове мере је што се може измерити *П*-вредност, која говори о веродостојности да је измерена хи-квадрат статистика дело случајности. На тај начин се лако утврђује и критеријум заустављања гранања, када изабрано гранање статистички није значајно са одређеним нивоом поверења (нпр. 5%).

Груписање вредности атрибута

Овај алгоритам такође има начин за груписање вредности атрибута, насупрот гранању по свакој вредности (као у ID3 и C4.5) или бинарном гранању (као у CART). Критеријум да се две вредности неког атрибута уједине у једну вредност се такође спроводи хи-квадрат тестом, па ако не постоји статистички значајан допринос сваке категорије, оне се сједињују. Тиме се могу добити гранања попут ове дате на Слици 6.



Слика 6. Гранање са груписаним вредностима атрибута.

Алгоритам

1. За сваки улазни (не-класни) атрибут, који мора бити категоричког типа:
    a. уколико атрибут има више категорија, за сваки пар вредности атрибута, хи-квадрат тестом испитати статистичку значајност да вредности атрибута имају различите дистрибуције класа.
    b. Уколико је значајност мања од предефинисаног прага (нп. 5%), спојити пар вредности (категорија) у једну.
2. За гранање стабла изабрати онај атрибут који максимизира хи-квадрат статистику
3. Уколико гранање по изабраном атрибуту није статистички значајно на изабраном нивоу (нпр. 5%), зауставити гранање. У супротном, гранати по изабраном атрибуту и рекурзивно поновити поступак за сваку грану од корака 1.

2.1.5. QUEST алгоритам

QUEST алгоритам је описан је у (Loh, Shih, 1997), и нуди неколико новитета у изградњи стабала.

Преселекција атрибута

Пре него што се приступи избору гранања у одређеном подстаблу, врши се преселекција атрибута коришћењем Ф-теста за нумеричке атрибуте или хи-квадрат теста за категоричке. Сваки атрибут који је испод изабраног нивоа статистичке значајности (нпр. 5%) бива избачен за разматрање (само у том

чвору стабла). Пошто се више атрибута тестира на значајност, користи се Бонферони корекција п-вредности, како би се избегла статистичка грешка првог типа. Атрибут са најмањом п-вредности (а која је значајна) се узима ко атрибут за гранање. Уколико нема атрибута који су статистички значајни, зауставља се гранање тог подстабла.

Транформација атрибута у нумерички

Уколико је изабрани атрибут категоричког типа, врши се трансофмација у нумерички атрибут, путем CRIMCOORD трансформације (Gnanadesikan, 1977). Ова трансформација прво представи категорички атрибут путем више индикатор варијабли (атрибута), сваки за посебну вредност атрибута, са вредности 1 уколико случај има ту вредност атрибута, и 0 у супротном. Након тога се врши дискриминациона анализа, тако да се ти атрибути трансформишу у један атрибут нумеричког типа, који представља дискриминациону функцију. Ова процедура је врло слична методи главних компоненти, са разликом што се пројекција на једну димензију ради коришћењем информације о класама, дакле нагледаним учењем.

Бинарно гранање

За добијени нумерички атрибут се одређује оптимална тачка гранања као и у ранијим алгоритмима. На тај начин је свако гранање овог алгоритма заправо бинарно, без обзира на број вредности категоричког атрибута. Процедура се понавља рекурзивно за сваку новодобијену грану.

Више информација о алгоритму, са примерима рачунања, се може наћи у (Delibašić et al, 2009a)

2.2. Поређење алгоритама

Сви наведени алгоритми су вођени истим циљем, да генеришу стабло које на будућим подацима прави најмању грешку класификације. Ипак, различити детаљи у томе како се прилази решавању тог циља доводе до тога да стабла

изграђена различитим алгоритмима могу изгледати сасвим другачије. У зависности од изабраног алгоритма разликоваће се:

- **дубина стабла** (нпр. бинарна гранања теже да направе дубља стабла)
- **ширина стабла** (нпр. информациона добит тежи да добије широка стабла)
- **укупан број чворова**
- **вредности атрибута које одређују** гране (појединачне или спојене категорије)
- **брзина изградње** (зависно колико потенцијалних гранања се евалуира и како се зауставља гранање)
- **тачност на новим подацима** (зависно од начина да се контролише комплексност модела и избегне претренирање)

Такође треба приметити да стабла изграђена различитим алгоритмима могу изгледати врло различито. То укључује избор атрибута који ће се наћи у чворовима стабла и сама правила које стабло дефинише. Штавише, у Поглављу 3 ће бити показано да и врло мале варијације у алгоритмима могу понекад резултовати врло различитим стаблима, а понекад немати уопште ефекта. Али свакако је најбитнија разлика између алгоритама колико су стабла која они креирају тачна у предвиђању класа.

У литератури су бројне студије које пореде перформансе постојећих алгоритама. Да би извукли општије закључке, студије врше поређења на низу различитих проблема (скупова података).

На пример, у студији (Lim et al, 2000) пореде се 22 алгоритма за стабла одлучивања, као и друге фамилије алгоритама, на 16 различитих скупова података из различитих домена.

Анализе показују да постоје алгоритми који су у просеку бољи и које аутори препоручују, али они нису најбољи на сваком скупу података. Стога генерални закључци о квалитету алгоритма могу бити контрапродуктивни када се бира алгоритам за неку специфичну област примене. Проблем је што

(у литератури и пракси) не постоје јасне одреднице када користити који алгоритам.

Такође, мерећи статистичку значајност разлика у тачности предвиђања, у овој студији показује се да су неки алгоритми статистички значајно различити, док се између неких алгоритама не може уочити значајна статистичка разлика, тј. разлике су можда дело случајности.

Аутори у (Lim et al, 2000) анализирали су и брзину алгоритма и поделили алгоритме у четири кластера, оне чије се извршавање мери секундама, минутима, десетинама минута и сатима. Интересантно је да алгоритми који су брзи имају статистички једнаке перформансе предвиђања.

Аутори такође у закључку предлажу алгоритме који су укупно дали добре резултате, и дају интерпретацију због чега мисле да су се ти алгоритми истакли. У тој интерпретацији се тумаче делови алгоритама за које аутори мисле да су одиграли кључну улогу у побољшању перформанси.

Када је реч о специфичним студијама које испитују успешност алгоритама за изградњу стабала одлучивања у одређеној области примене, добар пример је и студија (Zhao, Zhang, 2007). У овој студији се спроводи експериментално поређење алгоритама на проблему класификовања свемирских објеката на типове (попут квазара, активних галаксија, звезда, итд.), а на основу особина са визуелних и инфрацрвених мерења. Аутори издвајају алгоритме који су показали најбоље резултате по тачности и брзини извршавања, али наглашавају и битну карактеристику да су модели стабла разумљиви за тумачење, у облику правила, и да то даје добру повратну информацију астрономима у разумевању битних карактеристика објеката.

Најзад, постоје и напори да се експериментисање са алгоритмима на различитим подацима обједини у такозване базе експеримената (Blockeel, 2006). У тим базама би се акумулирало искуство у примени алгоритама, што би створило основу за мета-анализе и препоруке када који алгоритам треба користити. Прегледом ове базе (која је доступна за упите путем веба), такође

се могу видети перформансе постојећих алгоритама за изградњу стабала одлучивања, а први увид је да, као и у другим студијама, на различитим подацима различити алгоритми су најбољи, док није најјасније због чега.

Имплементације у софтверима

Када су у питању имплементације у софтверима, алгоритми за изградњу стабала одлучивања су врло зрео концепт и готово да не постоји софтвер (или библиотека) за откривање законитости, који не садржи барем неки алгоритам за изградњу стабла. То укључује популарне софтвере отвореног кода, попут софтвера: Rapid Miner, Weka, Orange, Knime, Scikit-Learn, R, Java-ML, Shogun, Apache Mahout; али и комерцијалне софтвере попут: SPSS Modeler, SAS, Matlab, MS SQL Server (SSAS), итд.

Ипак, постоји велики проблем што су различити алгоритми присутни у различитим софтверима. То онемогућава фер поређење алгоритама, јер су писани на различитим програмским језицима, са различитим опцијама и са различитим процедурама за евалуацију. Такође, добри елементи из једног софтвера се не могу искористити у другом, а неретко је и сам формат података другачији у различитим софтверима.

Најзад, чак и номинално исти алгоритми из различитих софтвера, када се изврше над истим подацима, дају другачије резултате. Ово се дешава због детаља у имплементацији и избора одређених параметара на које корисник нема утицаја. Стога често студије пореде алгоритме само из једног софтвера (нпр. из Weka (Zhao, Zhang, 2007)), што умногоме ограничава да се сагледају аспекти шире популације алгоритама за изградњу стабла одлучивања.

2.3. Упапређења предложена у литератури

Како су се стабла одлучивања показала корисна, развијају се и разна унапређења основних алгоритама. Ова унапређења су најчешће парцијална, што значи да се уводе нови начини решавања неког дела алгоритма, за који се сматра да у одређеним ситуацијама лоше утиче на перформансе.

Једно такво унапређење је и нова мера евалуације гранања названа „*distance*“, која уводи нови начин нормализације постојећих мера нечистоће (Mantaras, 1991). Иако једноставна промена алгоритамски, показује се да има значајна унапређења у експерименталним евалуацијама, што ће се видети и у Поглављу 3 овог рада.

Од нових мера евалуације се такође појављују и однос веродостојности, ДКМ критеријум, Колмонгоров-Смирноф, *AUC*, и остали. Више детаља како се они рачунају се могу наћи у (Rokach, Maimon, 2015; Поглавље 5).

Основни алгорими за изградњу стабла користе униваријатна гранања, тј. гранања на основу једног изабраног (најбољег) атрибута. То чини границу раздвајања понекад превише једноставном, јер је у сваком потпростору граница ортогонална у односу на само једну координату (атрибут). Да би се повећала флексибилност у коначној репрезентацији стабла, уводе се мултиваријатна гранања, тј. гранања на основу више атрибута. Најчешће су услови гранања облика $\alpha \cdot X + \beta \cdot Y > m$, где су X и Y изабрани атрибути, а m граница између леве и десне гране. Оваква стабла се називају „нагнута“ (енг. *oblique*) стабла (Heath et al, 1993; Murthy et al, 1994)

Поред мера евалуације, унапређиване су доста и методе орезивања стабла. Поред већ поменутих метода, појављују се и орезивање са редукованом грешком, орезивање минималне грешке, орезивање на основу принципа минималне дужине описа (MDL), итд. Преглед критеријума за орезивање се може наћи у (Rokach, Maimon, 2015; Поглавље 6), а експерименти са различитим врстама орезивања су описани у (Frank, 2000). Процес орезивања кроз примере је описан у књизи (Delibašić et al, 2009a).

Постоје проблеми код којих је излазни (класни) атрибут заправо састављен из више атрибута. Код таквих проблема није могуће применити класичне мере квалитета, јер су оне базиране на разликовање вредности једног атрибута. За ту намену се може користити генерализација звана *Предиктивна стабла за кластеровање* (енг. *Predictive clustering trees*), која

користе мере компактности из алгоритама за кластеровање, како би оценили квалитет гранања (Todorovski et al, 2002). Сваки лист у стаблу стога даје предвиђање више атрибута истовремено.

Такође је могуће да у листовима стабла не буде једноставно правило да се предвиђа она класа која је већински присутна у том листу. Уместо тога, могуће је користити сложенија правила за одређивање класе у сваком листу. Тако постоји решење које у листовима користи алгоритам Наивног Бајеса, зван *NBTree* (Kohavi, 1996), или *LogisticModelTrees*, који користи логистичку регресију у листовима стабла (Landwehr et al, 2005)

3. Алгоритми базирани на компонентама

Алгоритми за изградњу стабала одлучивања који су обрађивани у литератури су прављени као монолитни и у облику процедуре чијим извршавањем се добија резултујуће стабло одлучивања. Овај приступ називаћемо приступом „црних кутија“, јер су од корисника сакривени детаљи алгоритма. Са друге стране, постоји начин описивања оваквих алгоритама као низа компоненти за решавање одређених потпроблема, чиме се отвара могућност комбиновања компоненти (делова) различитих алгоритама стабала одлучивања за класификацију и генерисање мноштва нових алгоритама. Корисник може при томе имати активну улогу у избору компоненти, што алгоритме може учинити прилагођенијим специфичним ситуацијама, али и разумљивијим за корисника.

Приликом сваког унапређења алгоритама, аутори покушавају да покажу да су унапређени алгоритми објективно бољи од оригиналних. Али многе студије нису могле да потврде супериорност неког од алгоритама на свим подацима, због чега заправо ниједан алгоритам не бива напуштен. Разлози за све то постају јаснији када (Wolpert, 1996) теоретски доказују да се за сваки специфичан проблем може конструисати алгоритам који је оптималан за тај проблем, иако тај алгоритам на осталим проблемима може бити јако лош. Ово је био јак сигнал да се одустане од потраге за "најбољим" алгоритмом, и да је добро да постоје више алгоритама, где сваки има потенцијал да буде "најбољи" на специфичном проблему.

Још једна мотивација сумаризована је у раду (Sonnenburg, 2007), где аутори позивају на развијање решења отвореног кода, који треба да пордже и унапреде следеће циљеве:

- репродуковање научних резултата, које је отежано због разноликих имплементација истих концепата и алгоритама,
- поређење у више детаља алгоритама, како би се тестирало шта су прави узроци перформанси,

- смањивање реимплементације и убрзавање даљег развијања алгоритама,
- брже усвајање у другим дисциплинама и индустрији,
- колаборативно грађење стандарда.

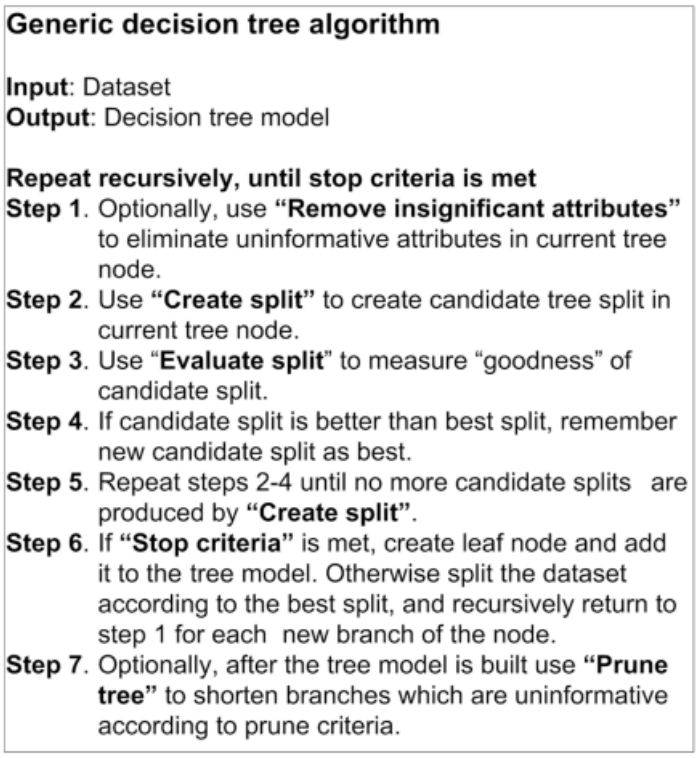
Такође, алгоритми који се посматрају монолитно отежавају да се добри елементи решења искористе у другом контексту. Такав је пример и са груписањем вредности атрибута у CHAID алгоритму, које има одличне карактеристике, а није виђено у новијим алгоритмима, а који такође имају проблем да неки атрибути имају превише вредности. Ова заборављена особина (корак) CHAID алгоритма би сигурно била од користи и у другом контексту.

У овом поглављу ће бити приказана генерализација постојећих алгоритама, као и оквир за развој и тестирање нових алгоритама и њихових градивних компоненти.

3.1. Генерички приказ постојећих алгоритама

До сада описани алгоритми, иако различити у детаљима, деле заједничку структуру, што није изненађујуће, с обзиром да мора да се суоче са истим проблемима. Сви алгоритми регурзивно праве стабло, бирајући у сваком кораку атрибут који је неком мером процењен као „најбољи“, а гранање се врши до неког критеријума заустављања. Додатни кораци, попут орезивања или редукције атрибута, су опциони и не појављују се у свим алгоритмима.

Општи алгоритам се може описати генеричком алгоритмом датој на Слици 7 (Delibašić et al, 2011). У сваком кораку се решава појединачни потпроблем, али тачан начин није дефинисан. Конкретним избором начина решавања потпроблема добиће се конкретна имплементација алгоритма. Сам ток извршавања може бити унапред имплементиран, позивајући се на непознате функције. Ове опште кораке називаћемо **потпроблеми**, док конкретна решења потпроблема ћемо називати **компоненте**.

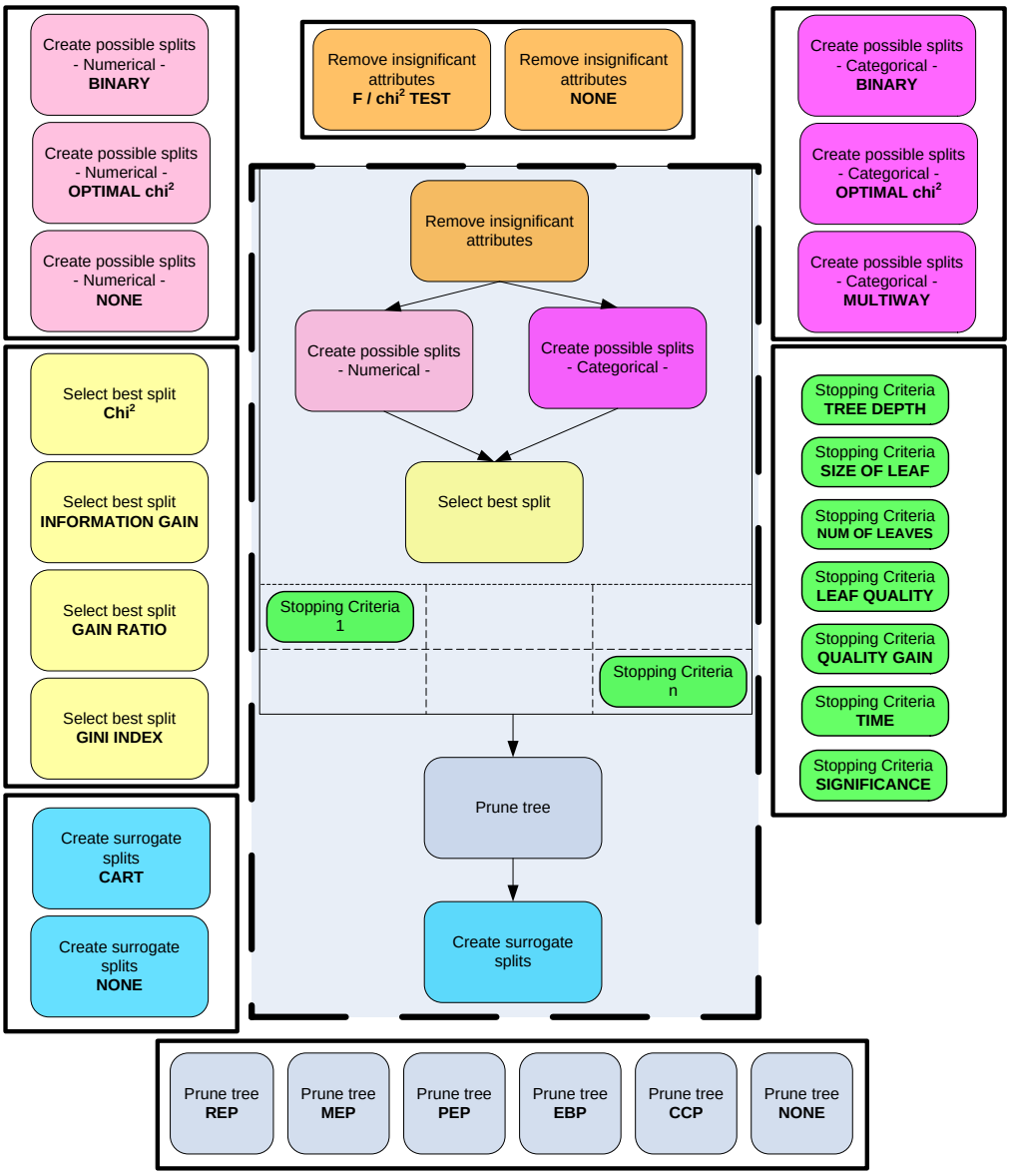


Слика 7. Генерички алгоритам за индукцију стабла одлучивања (Delibašić et al, 2011)

Структура потпроблема и компоненте за решавање ових потпроблема су дате на Слици 8, а потичу из различитих оригиналних алгоритама. Сваки потпроблем може имати низ компоненти који га решавају на својствен начин, а неки од потпроблема су и опциони, за шта постоји компонента „*None*“.

Потпроблем „*Create possible splits*“ треба да обезбеди да се у сваком чвору стабла генеришу могућа гранања која треба размотрити, и међу којима треба изабрати најбоље. Постоји варијанта за нумеричке и категоричке атрибуте. Компоненте за решавање овог потпроблема су „*Binary*“ (из алгоритма *CART*), „*Multiway*“ (из алгоритма C4.5) и „*Optimal Chi²*“ (из алгоритма *CHAID*).

После генерисања потенцијалних гранања, потпроблем „Select best split“ треба да омогући мерење квалитета потенцијалног гранања. Могуће компоненте су „*Information Gain*“ (из алгоритма *ID3*), „*Gain Ratio*“ (из алгоритма *C4.5*), „*Gini*“ (из алгоритма *CART*) и „*Chi²*“ (из алгоритма *CHAID*).



Слика 8. Структура компоненти за алгоритме стабла одлучивања (у средини), и могуће компоненте као решења за сваки потпроблем. (Delibašić et al, 2011)

Након гранања по изабраном атрибуту, следи одлука да ли сваку новонасталу грану треба даље гранати, што је одговорност потпроблема „*Stopping Criteria*“. Одлука о заустављању зависи од изабране компоненте, које могу бити „*Tree depth*“, која заустављa гранање на одређеној дубини стабла, „*Size of leaf*“, која заустављa гранање ако у грани нема минималан број случајева, „*Num of leaves*“, који заустављa ако број листова пређе одређену

границу, „*Leaf quality*“, који зауставља ако је у грани испуњен минимални предефинисани квалитет (мерен изабраном мером квалитета) „*Quality gain*“, зауставља ако пораст у квалитету није довољан, „*Time*“, ако је истекло одређено време и „*Significance*“, ако грана није статистички значајна (из алгоритма *CHAID*).

Након рекурзивне изградње целог стабла, потпроблем „*Prune tree*“ треба да одлучи како стабло треба орезати, како би се смањила комплексност модела и избегло претренирање. Могуће компоненте су „*CCP*“ (из алгоритма *CART*), „*PEP*“ (из алгоритма *C4.5*) и „*REP*“ „*MEP*“ „*EBP*“ (из унапређења, поменутих у Поглављу 2.3).

Потпроблеми су приказани као модули генеричког алгоритма на Слици 8 (у средини), али треба рећи да њихова комбинација није једноставно сукцесивно позивање, већ генерички алгоритам са Слике 7 управља како и када се позивају решења потпроблема.

Треба напоменути и да је могуће изабрати више компоненти за решавање неких потпроблема. Тако за креирање потенцијалних гранања корисник може изабрати и „*Binary*“ и „*Multiway*“ компоненту, које ће свака креирати своје кандидате за евалуацију. Такође је могуће изабрати више критеријума заустављања, где се зауставља гранање у подграни ако је било који од критеријума испуњен.

Сваки потпроблем такође има и стандардан, предефинисан, улаз и излаз, што све компоненете морају да испоштују. Овај начин уопштавања алгоритма је врло познат заједници софтверског инжењерства, где се избор конкретних решења „декуплује“ од генералне структуре алгоритма, како би се софтвер лакше одржавао, тј. да измене у имплементацији у једном делу софтвера не захтевају измене у другом делу. У овој заједници се овакво решењо назива још и *Патерном Стратегије*. Управо овај патерн је примењен и у софтверској реализацији генеричких алгоритама за стабла, о чему ће више речи бити у Поглављу 3.2.

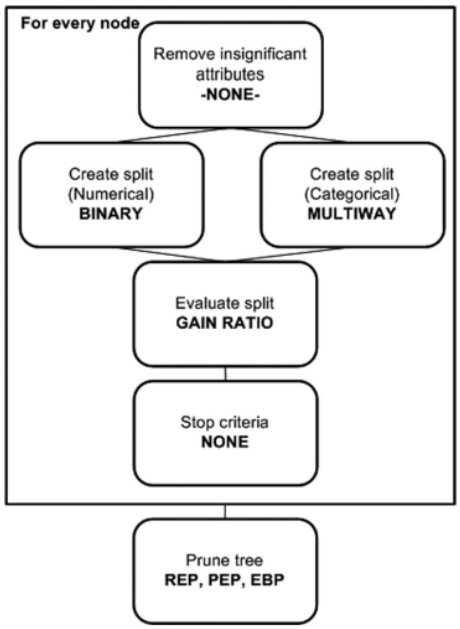
Стандардни предефинисани улази и излази потпроблема су приказани у Табели 2. Из исте табеле се може видети и да поједине компоненте имају и параметре којима се прецизније дефинише рад компоненте, а који имају своје подразумеване вредности, као и опсег могућих вредности.

Табела 2. Улази и излази потпроблема, и параметри компоненти (Vukićević et al, 2012a)

Sub-problem	Reusable Component	Parameters	Input	Output
Remove insignificant attributes	CHI SQUARE / ANOVA F TEST	alpha (def. 0.05, min 0, max 1)	Dataset in current node	Dataset in current node (reduced)
Create split (Numerical)	BINARY		Dataset in current node	Split candidates
Create split (Categorical)	BINARY			
	MULTIWAY			
	SIGNIFICANT	merge alpha (def. 0.05, min 0, max 1)		
		split alpha (def. 0.05, min 0, max 1)		
	ALL			
Evaluate split	CHI SQUARE		A split candidate	Evaluation of the split
	INFORMATION GAIN			
	GAIN RATIO			
	GINI			
	DISTANCE MEASURE			
	TWOING			
Stop criteria	MAXIMAL TREE DEPTH	tree depth (def. 10, min 1, max 20)	Current tree model	Signal for stopping tree growth in current node
	MINIMAL NODE SIZE	node size (def. 30, min 1, max 1000)		
	LEAF LABEL CONFIDENCE	confidence (def. 0.95, min 0, max 1)		
Prune tree	PESSIMISTIC ERROR PRUNING (PEP)	confidence (def. 0.25, min 0.1, max 0.5)	Current tree model	Pruned tree model
	COST COMPLEXITY PRUNING (CCP)	use-1stdev (def. 0, min 0, max 1)		
	MINIMAL ERROR PRUNING (MEP)	mParameter (def. 2, min 0, max 1000)		
	REDUCED ERROR PRUNING (REP)			
	MINIMAL LEAF SIZE (MLS)	leaf size (def. 30, min 1 max 1000)		

Овако постављена структура омогућава лакше унапређење постојећих алгоритама, прављење нових и анализу кључних компоненти. Постоји још компоненти за решавање ових потпроблема, који су идентификовани у (Suknović et al, 2012), а који би даље унапредили постојеће алгоритме.

Свака генерализација, па и ова, треба да обухвати постојеће реализације као специјална појављивања генерализације. То значи да постојећи алгоритми, који су уопштени у генерички оквир, се могу добити специфичним избором дефинисаних компоненти. Тако се, на пример, алгоритам *C4.5* може добити специфичним избором компоненти датим на Слици 9.



Слика 9. Избор компоненти који конституишу алгоритам *C4.5*

Али оно где лежи моћ овог приступа је у лаком креирању нових алгоритама, комбинацијом различитих компоненти. На тај начин би се могао дефинисати алгоритам специфичан за неки домен примене, који даје боље резултате од постојећих алгоритама.

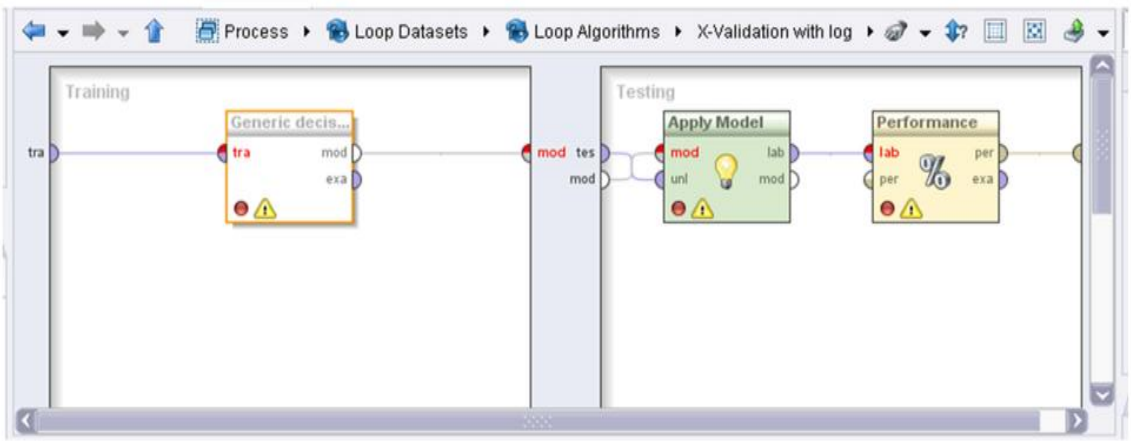
Идеја о прављењу генеричких структура алгоритама заправо превазилази алгоритме за изградњу стабала одлучивања. Сличан принцип се може применити и на друге фамилије алгоритама за откривање законитости, али и

општије од тога. Заправо је овај приступ већ примењен на алгоритмима за кластеровање, са позитивним резултатима (Delibasic et al, 2012b).

3.2. Реализација генеричког оквира - надградња *RapidMiner* софтвера

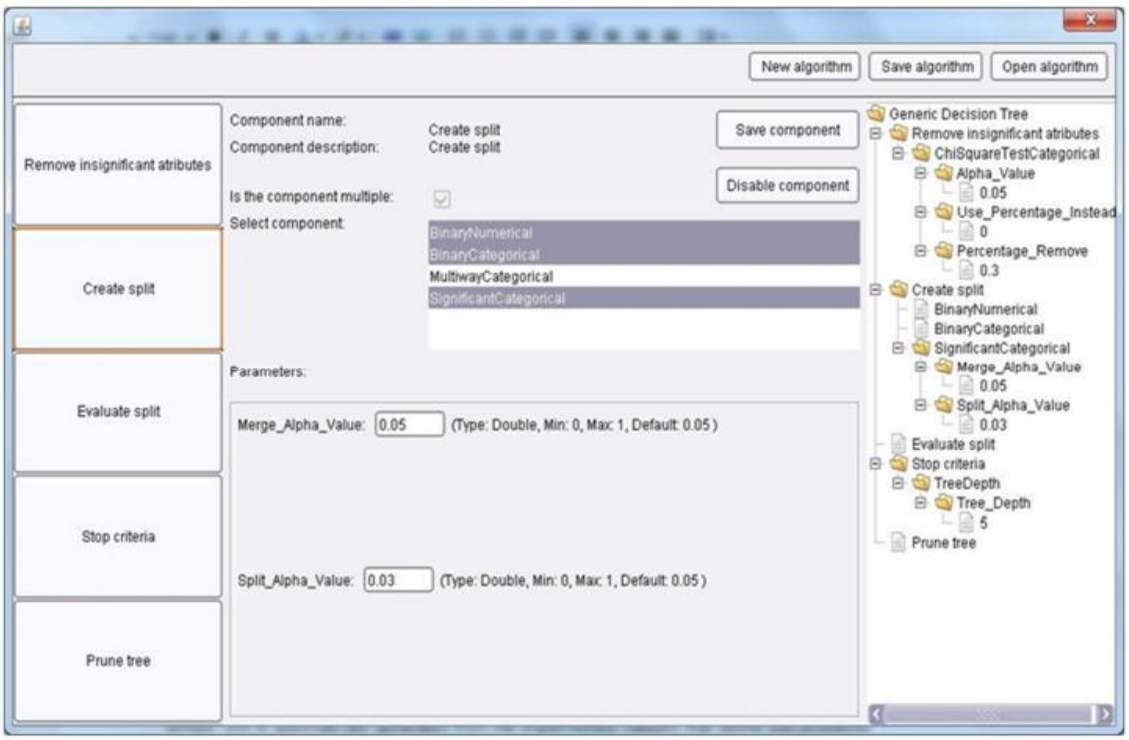
Оквир за имплементацију компонентних алгоритама је спроведен у програмском језику Јава, као надоградња популарног алата отвореног кода за откривање законитости у подацима, *RapidMiner*. Овај алат, попут многих других, је и сам замишљен као компонентни, у томе да се низ алгоритама може комбиновати да би се добили сложени процеси који решавају комплексне проблеме предвиђања. Ипак, иако се алгоритми склапају у процесе у виду компоненти, сам дизајн и имплементација алгоритама је монолитна, стога компонентни алгоритми имају битну улогу у укупном процесу. *RapidMiner* је изабран због своје популарности, отворености, лаке надоградивости, и доброј заједници која промовише коришћење целог окружења. Оквир је назван Вајбо (енг. *Whibo* од *White-Box algorithm*).

Пошто је оквир реализован као надградња *RapidMiner* софтвера, рад са алгоритмом се своди на употребу новог оператора названог *Generic Decision Tree*, приказаног на Слици 10, док се припрема података, евалуација и остали делови ОЗП процеса ослањају на постојеће операторе у *RapidMiner*-у.



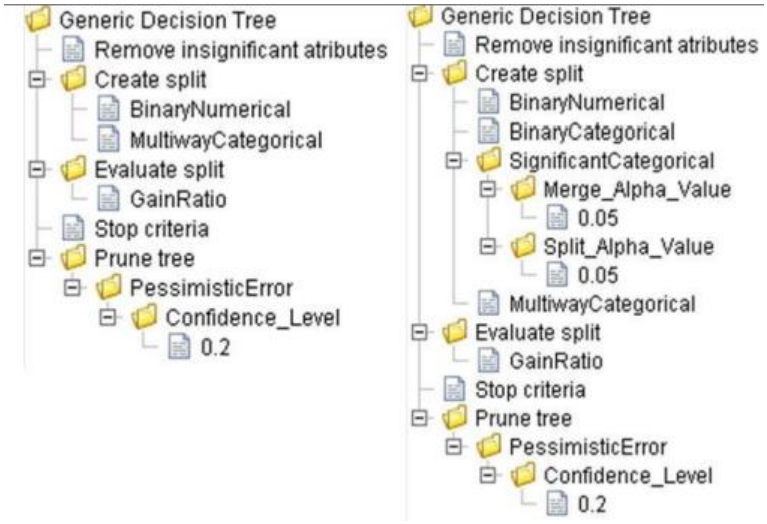
Слика 10: Пример коришћења додатог оператора у софтверу *RapidMiner*

Отварањем подешавања овог оператора добија се форма на којој се врши избор компоненти за решавање сваког од потпроблема (Слика 11). Уколико компонента дозвољава фино подешавање параметрима, дозвољава се и унос вредности параметара. Са левом делу Слике 11 видимо могуће потпроблеме, на горњем делу изабране компоненте понуђене за један потпроблем, на доњем делу параметре, а на десном делу све тренутне изборе компоненти генеричког алгоритма, приказане у виду стабла.



Слика 11: Графичка форма за дизајн компонентних алгоритама

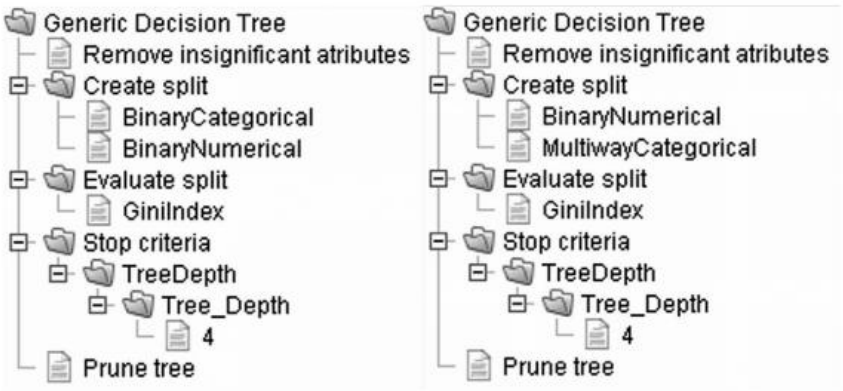
Као што је раније напоменуто, специфичним избором компоненти се могу конструисати већ постојећи алгоритми (попут C4.5, Слика 12а) или нови, хибридни, алгоритми, састављени од компоенети из различитих оригиналних алгоритама (Слика 12б).



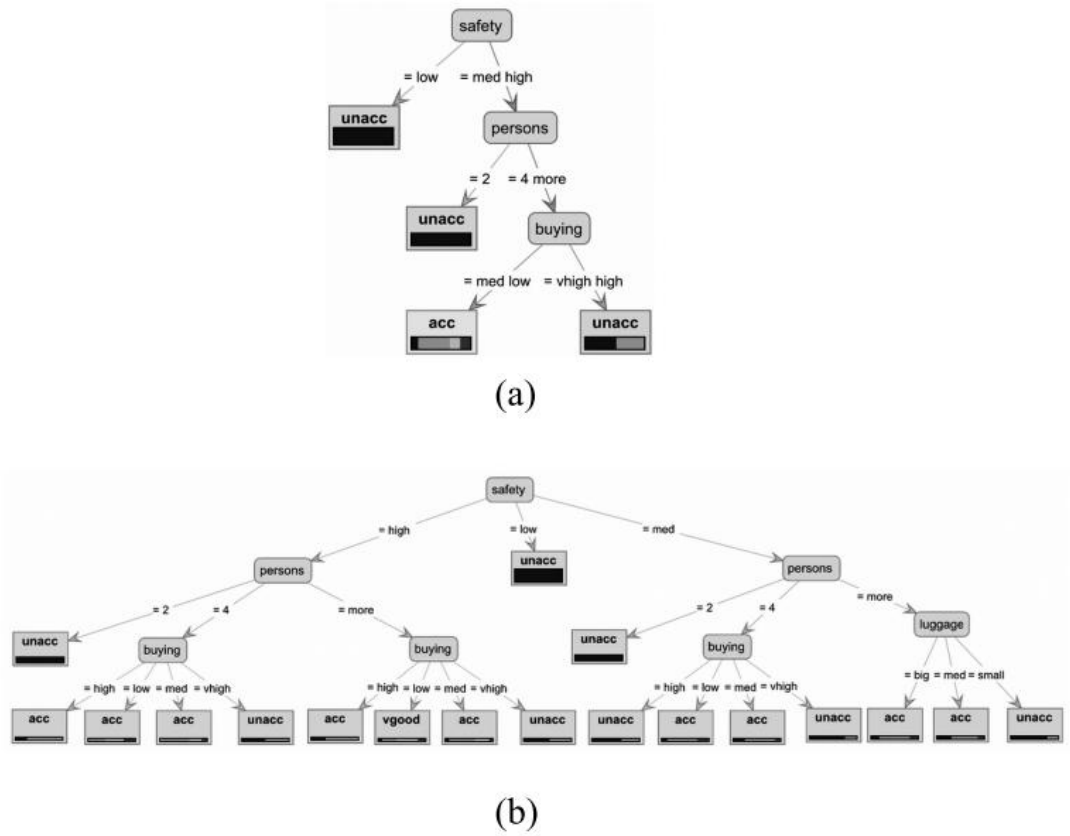
Слика 12. (а) лево: Избор компоненти за оригинални C4.5 алгоритам
(б) десно: Избор компоненти за хибридни алгоритам

Покретањем алгоритама, креира се резултујуће стабло, које се прослеђује *RapidMiner* софтверу у формату који он разуме, што резултује визуелизацијом стабла (Слика 14).

Битна особина генеричког алгоритма је да избор компоненти значајно утиче на резултујуће стабло. На пример, два алгоритма (Слика 13), која се разликују у само једној компоненти, резултују са два врло различита стабла приказана на Слици 14. Стога је избор компоненти јако важан када се решава неки специфичан случај. Са друге стране, нису све компоненте подједнако утицајне, о чему ће више речи бити у Поглављу 3.3.



Слика 13. Дефиниције два компонентна алгоритма који се разликују само у једној компоненти



Слика 14. Резултујућа стабла креирана алгоритмима који се разликују у само једној компоненти

3.2.1. Детаљи софтверског решења

Изграђени софтверски оквир је прављен са идејом лаке надоградивости и интеграције са осталим решењима. Њехову релацију са осталим деловима је могуће приказати *стек* дијаграмом (Слика 15):

1. На најнижем нивоу се налазе основне градивне јединице, садржане у разним библиотекама са математичким, статистичким и осталим фунцијама, корисним за реализацију делова алгоритма.
2. Све компоненте се граде ослањајући се једино на те библиотеке, што их чини независним од тога како ће даље бити коришћени.
3. Наредни ниво је генерички алгоритам за индуковање стабала одлучивања, који је својеврсна композиција изабраних компоненти, и

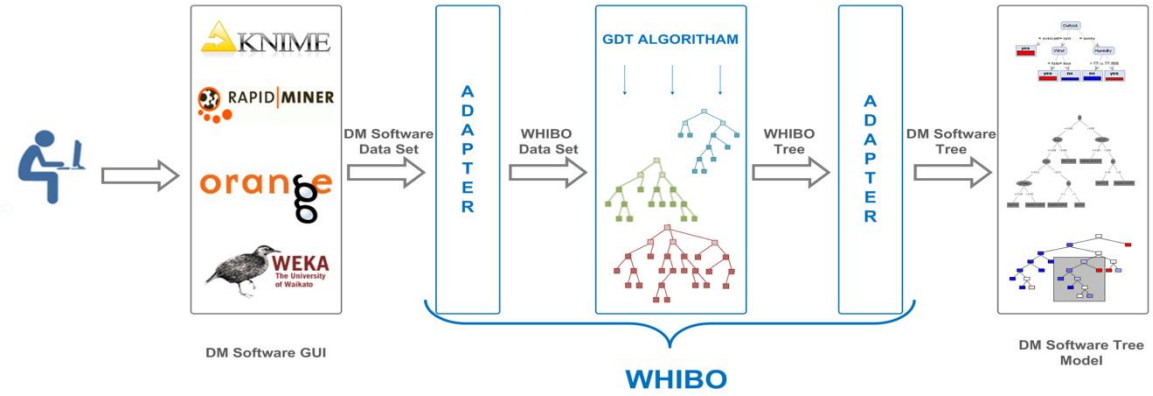
који управља током извршавања, позивајући компоненте за решавање појединих потпроблема у генеричком решењу.

4. Да би се користио генерички алгоритам, потребно је омогућити да кроз постојеће софтвере за откривање законитости у подацима, дефинишемо интерфејс ка генеричком алгоритму, кроз специфичне адаптере. Адаптери треба да омогуће интероперабилност, тако што чине да софтвери могу да проследе податке у одговарајућој форми генеричком алгоритму, али и да преузму решење како би даље са њим радили, на пример за визуелизацију стабла. Овај процес адаптације приказан је и на Слици 16.
5. На последњем нивоу су постојећи софтвери за ОЗП. Тренутно је систем реализован за *RapidMiner* софтвер, али не постоје ограничења да исти алгоритми буду доступни и кроз друге сличне софтвере. Тиме би се употребљивост генеричког алгоритма за стабла проширила на више заједница корисника, који преферирају одређени софтвер за рад.

Слика 15. Стек дијаграм за призак зависности софтверских делова, тј. на шта се ослања сваки део система. (Vukićević et al, 2012a)

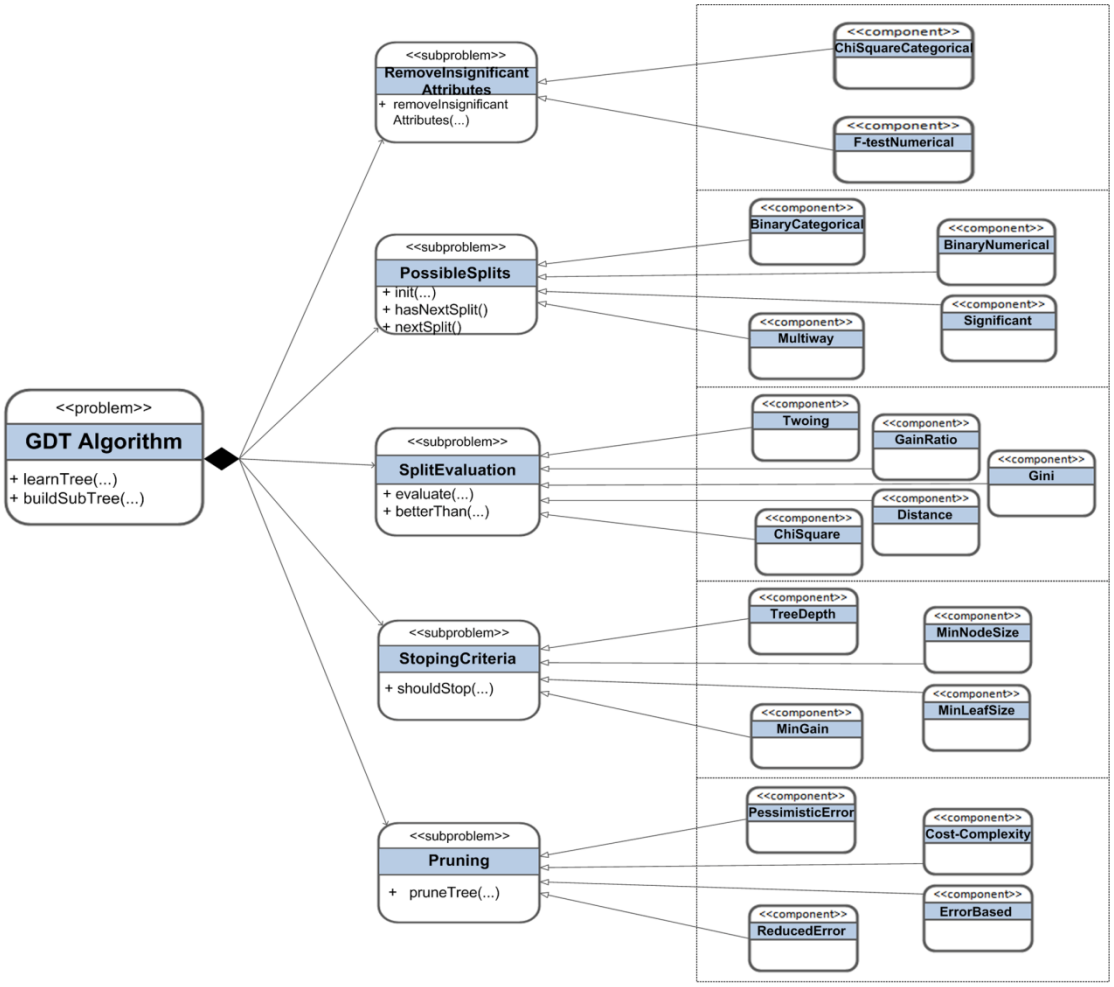
Из угла софтверског инжењерства, компонентни алгоритми су реализовани као низ програмских интефејса који дефинишу улазе и излазе потпроблема, и генеричког алгоритма који управља позивима компоненти. Ово је реализација такозваног *Патерна Стратегије*, приказан *дијаграмом класа* на

Слици 17. Интерфејси дефинишу потпроблеме, док су компоненте софтверске класе, које реализују поједине интерфејсе, имплементирајући методе са предефинисаним потписом.



Слика 16. Интеграција генеричког алгоритма са окружењем коришћењем адаптера. (Vukićević et al, 2012a)

Када се на корисничком интефејсу изаберу компоненте и параметри (Слика 11), након приска на дугме „*Save Algorithm*“ алгоритам се чува у специфичном XML формату текстуалне датотеке (Слика 18). Овај формат прецизно дефинише елементе и довољна је спецификација да генерички алгоритам може на основу те спецификација де се изврши. С обзиром да се ради о тексуалној датотеци, напредни корисник може и ручно интервенисати над овом спецификацијом, или се може независно изградити визуелни алат за манипулисање овим форматом спецификације алгоритма. Дефинисана је и XML схема која служи да се верификује исправност направљене XML датотеке. Овај специфичан формат је назван *.wba* екстензија (од *white-box algorithm*) и служи као спецификација алгоритма коју разуме овај софтвер.



Слика 17: Дијаграм класа који илуструје софтверски патерн "Стратегија", којим је имплементација генеричког алгоритма независна од изабране компоненте, тј. конкретне реализације интефејса потпроблема (Vukićević et al, 2012a)

Као што је речено, систем је направљен тако да буде лако надоградив и поновно искоришћен. Уколико желимо да дефинишемо неки нови генерички алгоритам, овај софтверски оквир нам даје лак начин за то. Потребно је направити софтверску класу која наслеђује апстрактну класу *ProblemBuilder*, и дефинисати од којих потпроблема се састоји неки алгоритам. Пример овакве дефиниције дат је на Слици 19.

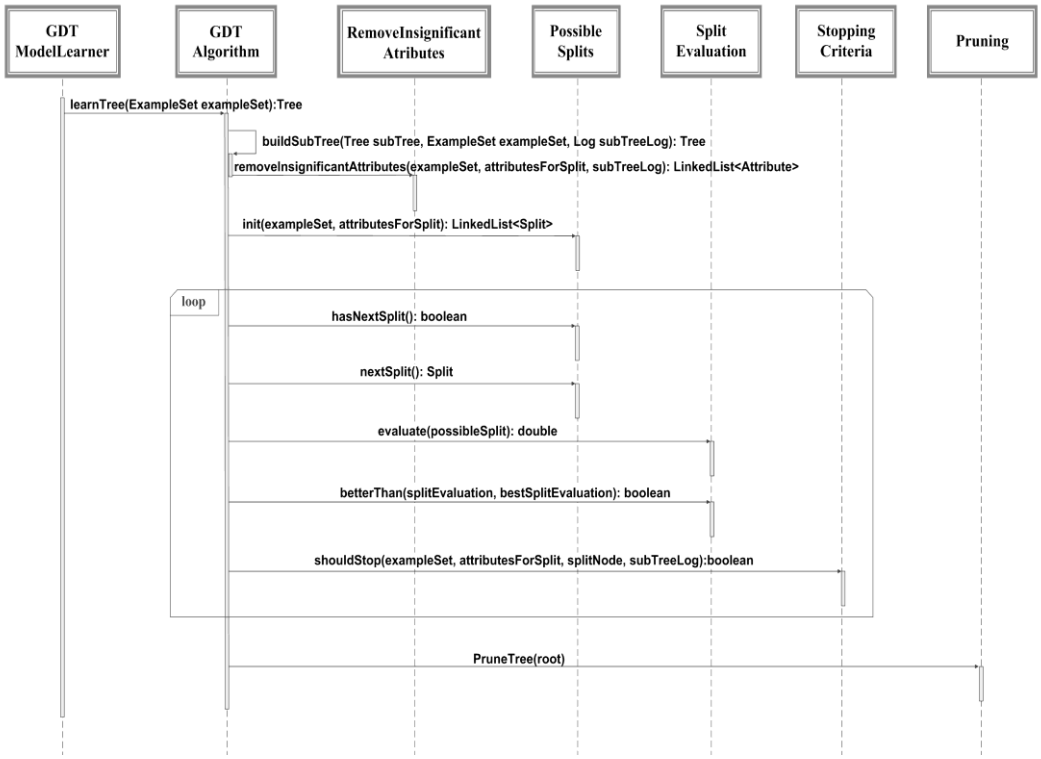
```
<rs.fon.whibo.GDT.problem.GenericTreeProblem>
 <description>Generic decision tree</description>
  <subproblems class="linked-list">
 < rs.fon.whibo.GDT.problem.subproblem.RemoveInsignificantAtributes>
   <name>Remove insignificant atributes</name>
   …
   <nameOfImplementationClass>….removeInsignificantAttributes.ChiSquareTestCategorical</
   nameOfImplementationClass>
   …
   <listOfParameters>
         <nameOfParameter>Percentage_Remove</nameOfParameter>
      …
         <EnteredValue>0.3</EnteredValue>
       </listOfParameters>
      …
     </rs.fon.whibo.GDT.problem.subproblem.RemoveInsignificantAtributes>
     <rs.fon.whibo.GDT.problem.subproblem.PossibleSplit>
      <name>Create split</name>
   … <nameOfImplementationClass>….possibleSplits.BinaryNumerical</nameOfImplementationClass>
   … <nameOfImplementationClass>….possibleSplits.BinaryCategorical</nameOfImplementationClass>
      …
   <nameOfImplementationClass>...possibleSplits.SignificantCategorical</nameOfImplementationClass>
   <listOfParameters class="linked-list">
      …
         <nameOfParameter>Merge_Alpha_Value</nameOfParameter>
         …
         <EnteredValue>0.05</xEnteredValue>
         …
         <nameOfParameter>Split_Alpha_Value</nameOfParameter>
         …
         <EnteredValue>0.03</xEnteredValue>
      …
      </listOfParameters>
   </rs.fon.whibo.GDT.problem.subproblem.PossibleSplit>
      <rs.fon.whibo.GDT.problem.subproblem.SplitEvaluation>
      <name>Evaluate split</name>
   …
      <nameOfImplementationClass> GiniIndex</nameOfImplementationClass>
</rs.fon.whibo.GDT.problem.GenericTreeProblem>
```

Слика 18. XML запис изграђеног компонентног алгоритма, са свим дефинисаним детаљима (Vukićević et al, 2012a)

```
public class GenericTreeProblemBuilder extends ProblemBuilder {
  @Override
     public Problem buildProblem() {
        Subproblem s1 = new RemoveInsignificantAtributes();
        Subproblem s2 = new PossibleSplit();
        Subproblem s3 = new SplitEvaluation();
        Subproblem s4 = new StopingCriteria();
        Subproblem s5 = new Prunning();
                  // defines order of subproblems on GUI
        List<Subproblem> subproblems = new LinkedList<Subproblem>();
        subproblems.add(s1);
        subproblems.add(s2);
        subproblems.add(s3);
        subproblems.add(s4);
        subproblems.add(s5);
                        Problem GDTproblem = new GenericTreeProblem();
        GDTProblem.setSubproblems(subproblems);
        return GDTProblem;
    }
}
```

Слика 19. Део кода којим се дефинишу потпроблеми неког генеричког алгоритма (Vukićević et al, 2012a)

Након тога, треба дефинисати ток генеричког алгоритма, позивајући се на апстрактна решења потпроблема. У овом тренутку није потрено познавати како су реализоване поједине компоненте, чак ни које компоненте постоје. Пример извршења генеричког алгоритма дат је *дијаграмом секвенци*, на Слици 20. Прве две класе су заправо адаптери који из специфичног софтвера (нпр. *RapidMiner*-а) позивају генерички алгоритам, након чега следи позив појединим потпроблемима. Зависно које је компоненте корисник изабрао на корисничком интерфејсу, током извршења програма ће бити активиране изабране компоненте.



Слика 20. Дијаграм секвенци извршавања генеричког алгоритма (Vukićević et al, 2012a)

Најзад, креатори компоненти могу једноставно направити нове компоненте, које решавају постојећи потпроблем. Неопходно је само дефинисати софтверску класу која реализује програмски интефејс одређеног потпроблема и имплементирати предефинисане методе (функције). Креатор компоненти не мора да брине и о генеричком алгоритму, нити о корисничком интефејсу, који ће аутоматски препознати нову компоненту и

омогућити њен избор кориснику. Додатно, дефинисање параметара компоненте је такође олакшано, јер се једноставним анотацијама у Јава програмском језику (*@Parameter*) спецификија шта је параметар, и то ће на корисничком интерфејсу бити аутоматски приказано. Пример изградње нове компоненте, за проблем евалуације гранања стабла, дат је на Слици 21.

```
package rs.fon.WhiBo.GDT.component.splitEvaluation;
public class MySplitEvaluation extends AbstractSplitEvaluation {
    @Parameter(defaultValue="0.05", minValue ="0",  maxValue="1");
     private Double User_Defined_Parameter;
    @Override
    public double evaluate(SplittedExampleSet exampleSet){
                */
                    user implementation for candidate split evaluation
                /*
    return splitEvaluation;
    }
}
```

Слика 21. Део кода (софтверска класа) који је једино потребно имплементирати да би се додала нова компонента за постојећи потпроблем (Vukićević et al, 2012a)

Цео софтверски оквир је пријављен и одобрен као техничко решење М85 (Delibašić et al, 2009b), а описан у (Vukicevic et al, 2012a; Vukićević et al, 2010; Jovanović et al, 2009). На реализацији ове платформе су, поред аутора техничког решења, учествовали и студенти Саша Мркела, Никола Николић, Јован Чукаловић и Сандро Радовановић.

3.3. Експериментална евалуација приступа

Као што је приказано примером у Поглављу 3.2, измена чак и само једне компоненте може да резултује сасвим различитим стаблом. Стога је битно које компоненте бирамо за алгоритам који користимо. Да ли је ово случаност или ибзори компоненти стварно значајно утичу на перформансе алгоритма, и у којој мери? Ово је једна у низу хипотеза које су постављане (Delibašić et al, 2011; Vukićević et al, 2012a), а које је требало верификовати. Хипотезе које су узимане у обзир су:

1. *„Размена појединачних компоненти прави значајну разлику на перформансе алгоритма“*.
2. *„Утицај неких компоненти је значајнији за перформансе од утицаја других.“*
3. *„Алгоритми састављени разменом компоненти постојећих алгоритама имају значајно боље резултате од оригиналних алгоритама“.*
4. *„Не постоји доминантност једне алгоритамске композиције, тј. за различите проблеме (податке) различите композиције компоеннти су оптималне“.*
5. *„Компонентни алгоритми су значајно разумљивији кориснику од оригиналних алгоритама“.*

Ове хипотезе су верификоване у експериментима, где је битно истаћи неколико важних аспеката евалуације:

- мера евалуације: Колико је стабло адекватно за класификацију (предвиђање) нових случајева, треба пажљиво измерити статистичким мерама грешке предвиђања. У експериментима су коришћени ***Тачност*** (енг. accuracy), која представља процену вероватноће тачног предвиђања, тј. процента успешно класификованих инстанци током тестирања. Такође се користи и ***Површина испод РОК криве*** (енг. Area Under ROC, AUC), која процењује вероватноћу да од случајно изабраног пара инстанци из класе А и Б, тачно одредимо која инстанца је из које класе. Ово су врло распростањене мере, што омогућава лакше поређење са литературом.
- процена грешке генерализације: Грешка процењена на подацима за тренинг често није репрезентативна као **грешка за будућу примену (*грешка генрализације*)**, јер комплексни модели који управо оптимизују грешку могу фиктивно смањити грешку користећи шум у подацима за тренинг, што је раније поменуто као проблем претренирања. Зато је потребна конзервативније процена грешке

генерализације, која се често добија издвајањем посебног скупа података само за тестирање, или разним техникама вишеструког узорковања тренинг скупа. У експериментима је највише коришћен метод **крос-валидације**, који из тренинг скупа више пута издваја различите (дисјунтне) узорке за тест и процењује просечну грешку из више таквих узорака. Ово је такође распрострањена техника, и омогућава лакше поређење и са литературом.

- скупови података: Уколико тестирамо алгоритме на једном скупу података, закључке које извучемо морамо ограничити само на тај проблем. Стога је битно поновити евалуацију на више скупова података и анализирати укупне перформансе, како би резултати били генералнији. У свим експериментима је коришћен **већи број података, из различитих домена**.
- бенчмарк алгоритми за поређење: Резултати евалуације новоизграђених алгоритама се **пореде са постојећим алгоритмима**, како би се истакао допринос и оправданост примене. У експериментима су за поређење коришћени постојећи монолитни (не-компонентни) алгоритми, који су у великој употреби у научној и стручној заједници. Такође, да би се избегли проблеми различитих окружења (оперативног система, имплементационог језика, рачунара, библиотека на које се ослањају алгоритми, итд.) сви нови и постојећи алгоритми су **тестирани из исте имплементације**, с обзиром да су постојећи алгоритми само реализације компонентних алгоритама. Овиме се омогућава **фер поређење** различитих алгоритама.
- поновљивост експеримената: да би резултати експеримената могли да се провере, бирани су подаци који су јавно доступни, а софтвер је доступан под лиценцом отвореног кода.

Да би оценили све наведене хипотезе, експериментални резултати су описани у три скупа експеримената, који одговарају различитим објављеним радовима (Vukićević et al, 2012a; Delibašić et al, 2011, Delibašić et al, 2012a).

3.3.1. Први скуп експеримената

Резултати ових експеримената су објављени у раду (Vukićević et al, 2012a). Експериментима је обухваћено 960 компонентних алгоритама, који су добијени кобинацијом компоненти за различите потпроблеме. Међу тим алгоритмима су и специфичне комбинације компоненти које чине постојеће познате алгоритме: *ID3*, *CART*, *C4.5* и *CHAID*.

Алгоритми су тестирани на 12 реалних, јавно доступних скупова података (Asuncion, Newman, 2007), који су били предмет и других студија у предвиђању. Скупови података су приказани у Табели 3.

Сви резултати евалуације алгоритма су рађени применом 10-оструке крос-валидације, чиме се и стабилније процењује грешка класификације. Тако је укупан број стартовања алгоритама 960x12x10. Параметри свих компоненти су постављени на подразумеване (енг. *default*) вредности.

Свих 960 алгоритама су на основу тачности предвиђања сортирани у опадајући низ и рангирани, за сваки скуп података посебно. У Табели 3 су истанкути рангови бенчмарк алгоритама за сваки скуп података, а подвучени најбоље рангирани од бенчмарк алгоритама.

Прво што треба приметити је да постојећи алгоритми нису ни једанпут били најтачнији за неки скуп података (нису имали ранг 1). Штавише, углавном је ранг био доста лошији у односу на најбоље пласирани компонентни алгоритам. Ово подржава хипотезу да се разменом делова (компоненти) алгоритама могу произвести тачнији алгоритми, и то на свих 12 скупова података који су анализирани.

Даље, видимо и да, од постојећих алгоритама, није увек најбољи био исти алгоритам. Што значи да избор најбољег алгоритма је врло завистан од проблема који се решава, па је добро пробати са више алгоритама. Штавише, унапред је тешко рећи ко би алгоритам био добар, а још теже због чега.

Табела 3. Коришћени скупови података и рангови познатих алгоритама на основу тачности предвиђања (подвучен је најбољи међу четири позанта алгоритма на сваком скупу података) (Vukićević et al, 2012a)

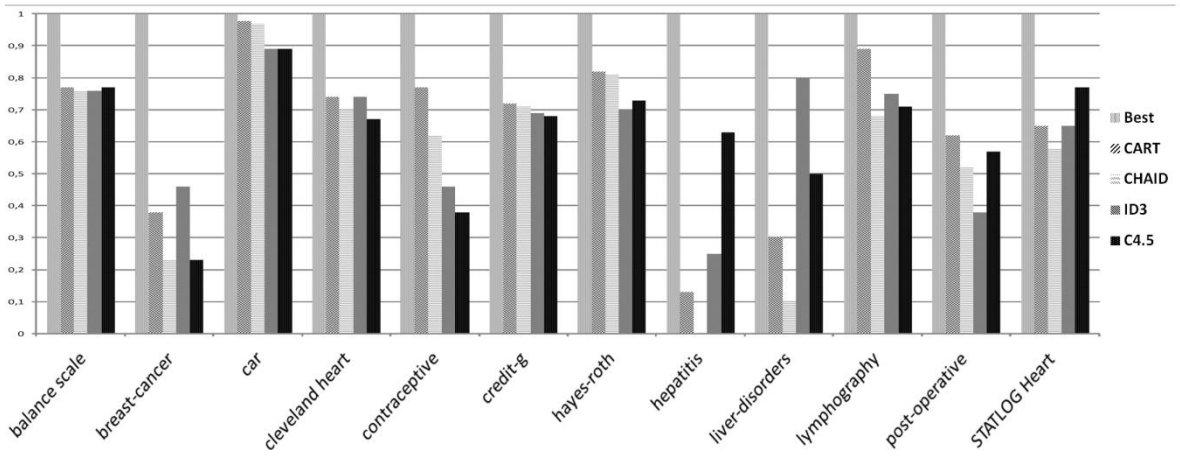
Dataset	CART	CHAID	ID3	C4.5
balance-scale	41	78	80	45
breast-cancer	130	148	89	155
car	2	23	230	227
cleveland heart	159	196	148	211
contraceptive	165	316	473	524
credit-g	52	71	79	101
hayes-roth	4	6	23	19
hepatitis	62	63	52	25
liver-disorders	77	87	15	37
lymphography	8	61	40	49
post-operative	9	11	17	10
STATLOG Heart	30	36	30	22

Да би боље сагледали разлике у алгоритмима, измерена је и нормализована удаљеност алгоритама у односу на најбоље пласирани, а по формули:

$$distAcc = \frac{Acc - \min(Acc)}{\max(Acc) - \min(Acc)}$$

где је *Acc*, тачност (*accuracy*) модела, а *max* и *min* функције над свим алгоритмима. Најбоље пласирани алгоритам (са рангом 1) ће имати удаљеност 0, док најгори 1, а сви остали ће бити на интервалу [0,1]. Резултати су приказани на Слици 22.

Из ових резултата видимо да не само да је ранг постојећих алгоритама био лош, већ је и тачност доста лошија од првопланираног компонентног алгоритма. Понекад (нпр. на подацима *breast-cancer*, *hepatitis*, *liver-disorders*) је та разлика и врло наглашена, где неки оригинални алгоритми не прелазе ни 30% перформанси најбољег компонентног алгоритма.



Слика 22. Поређење тачности предвиђања за постојеће алгоритме, нормирано у односу на најбоље пласирани компонентни алгоритам. (Vukićević et al, 2012a)

Такође је спроведена детаљнија анализа компонентних алгоритама који су дали добре резултате, гледајући које компоненте су учествовале у алгоритмима који су међу најбољим по тачности. За већину података, било је више од једног најбољег алгоритма (који деле први ранг). У Табели 4 су приказане компоненте које су биле присутне у тим најбољим алгоритмима, за сваки скуп података посебно. Уз назив скупа података стоји и број у загради, који говори колико је било прворангираних алгоритама. Остатак табеле набраја које су компоненте биле конзистентно део најбољих алгоритама.

Из Табеле 4 се такође види да за неке проблеме је потребан врло специфичан алгоритам, док за друге је довољно изабрати „кључну“ компоненту, а избор осталих није толико битан. Тако за податке „*Cleveland heart*“ битне су 4 од 5 компоненти, наиме: *RIA*, *SignificantCategorical*, *DistanceMeasure* и *MinNodeSize*, док начин поткресивања стабла не игра кључну улогу. Са друге стране, за податке „liver-disorders“ су само 2 компоненте биле кључне и то за потпроблеме *RIA* и *Evaluate*. На проблему „*Hayes-roth*“ видимо да је за потпроблем гранања стабла оптимално користити истовремено све начине гранања. Ово сугерише да је, за овај проблем, добро да се стабло у неком делу грана бинарно, а у другом вишеструко. Најзад, занимљиво је да је за проблем

„contraceptive“ само један алгоритам био са најбољом тачности предвиђања (број у загради), док код проблема „post-operative“ има 261 алгоритам са подједнаком тачношћу, што говори да избор компоненти овде није био од значаја. На подацима „credit-g“ било је само 2 најбоља алгоритма, али они нису имали ниједну заједничку компоентну.

Табела 4. Компоненте које су биле део најбољих алгоритама на сваком од скупова података (Vukićević et al, 2012a)

Dataset	RIA	Split	Evaluate	Stop	Prune
balance-scale (4)	No	BinaryCategorical		MinNodeSize	PEP
breast-cancer (6)	Yes		GainRatio		MEP
Car (2)	No	BinaryCategorical	ChiSquare		
cleveland heart (2)	Yes	Significant	DistanceMeasure	MinNodeSize	
Contraceptive (1)	No	BinaryCategorical	ChiSquare	MinNodeSize	CC
credit-g (2)					
hayes-roth (2)	No	All	DistanceMeasure		MEP
hepatitis (6)	Yes		DistanceMeasure		MEP
liver-disorders (16)	Yes		DistanceMeasure		
Lymphography (4)		BinaryCategorical			MEP
post-operative (261)					
STATLOG Heart (4)		BinaryCategorical		MinNodeSize	PEP

Овим експериментима се даје подршка хипотезама из Поглавља 3.3, наиме да компонентни алгоритми често дају боље резултате него оригинални алгоритми, да сваки пробем који се решава има другачију композицију компоненти која му одговара и да нису све компоненте од једнаког значаја за перформансе алгоритма.

3.3.2. Други скуп експеримената

Резултати ових експеримената су објављени у раду (Delibašić et al, 2011). Експериментима је овај пут обухваћено 80 компонентних алгоритама, који су добијени кобинацијом компоненти за различите потпроблеме, дати у Табели

5. За разлику од предхоних експеримената, компоненте критеријума заустављања нису вариране.

Табела 5. Компоненте чијом комбинацијом се може креирати 80 компонентних алгоритама (Delibašić et al, 2011)

Sub-problems	Reusable components				
RIA	chs/anf	none (!)			
CSC	mul	bin	sig	all	
ES	gai	inf	gin	dis	chs
PT	pep	none (!)			

У Табели 5 су такође дефинисани и кодови за сваку компоненту, који су коришћени за скраћени опис алгоритама, као и за обележавање на графиконима. Потпроблем „*RIA*“ (*Remove Insignificant Attributes*) има једну опциону компоненту „*chs/anf*“ (*Chi-Square/AnovaF*). Потпроблем „*CSC*“ (*Create Candiate Splits*) има три компоненте „*mul*“ (*Multiway*), „*bin*“ (*Binary*) и „*sig*“ (*Significant*), а компонента „*all*“ је истовремено коришћење све 3 основне. Потпроблем „*ES*“ (*Evaluate Split*) има 5 могућих мера евалуација: „*gai*“ (*Gain ratio*), „*inf*“ (*Information gain*), „*gin*“ (*Gini*), „*dis*“ (*Distance*) и „*chs*“ (*ChiSquare*). Потпроблем „*PT*“ (*Prune Tree*) има једну опциону компоненту „*pep*“ (*Pesimistic Error Pruning*). Ознака „!“ говори да ниједна компонента није изабрана за потпроблем, што је могуће код потпроблема који су опциони, као што је орезивање стабла. Параметри компоненти које их имају су постављени на подразумеване (енг. *default*) вредности.

Тако се, на пример, постојећи алгоритми могу приказати као:

1. *C4.5* (!-mul-gai-pep или !-mul-gai-!)
2. *CART* (!-bin-gin-!)
3. *CHAID* (!-sig-chs-!)

Експерименти су рађени на 15 јавно доступних, бенчмарк, скупова података из репозиторијума Универзитета у Ирвин, Калифорији (Asuncion, Newman, 2007). Табела 6 приказује пуне и скраћене називе скупова података. Колона „*Significant differences*“ ће касније бити објашњена. Карактеристике података су дате у Табели 7, где се види да су подаци различитих величина, и броја

атрибута и броја случајева. Треба напоменути да је број класа које се предвиђају у свим подацима релативно низак, па се закључци експеримената не могу генерализовати на проблеме са великим бројем класа.

Табела 6. Коришћени скупови података и број значајних разлика у алгоритмима (Delibašić et al, 2011)

ID	Dataset	Significant differences
car	Car evaluation	58%
nur	Nursery	43%
tic	Tic-tac-toe endgame	39%
aba	Abalone	19%
cmc	Contraceptive method choice	18%
spe	SPECT Heart	18%
con	Connect-4	5%
cre	Credit approval	5%
cov	Cover type	5%
adu	Adult	4%
kin	King-rook vs. king-pawn	4%
len	Lenses	2%
vot	Congressional voting records	1%
thy	Thyroid disease	0%
adv	Internet Advertisements	0%

Табела 7. Основне карактеристике коришћених скупова података (Delibašić et al, 2011)

ID	No. cat. attrib.	No. num. attrib.	No. records	No. classes
Car	6	0	1728	4
Nur	8	0	12960	4
Tic	9	0	958	2
Aba	1	7	4177	3
cmc	2	7	1473	3
Spe	22	0	187	2
Con	42	0	67557	3
Cre	9	6	690	2
Cov	10	44	581012	7
Adu	8	6	32561	2
Kin	36	0	3196	2
Len	5	0	24	3
Vot	16	0	435	2
Thy	22	6	2800	3
Adv	1555	3	2369	2

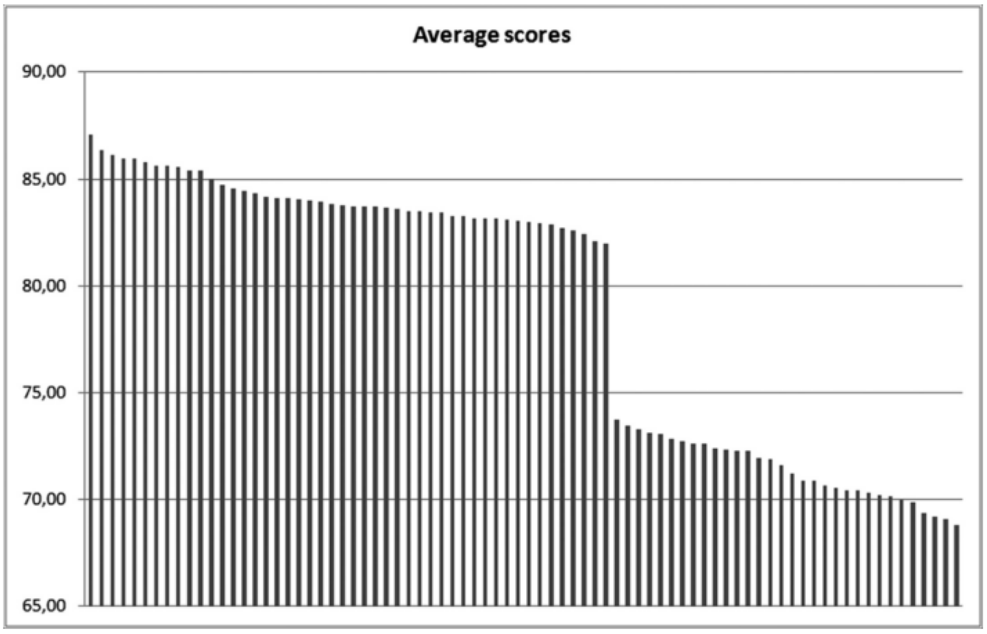
Циљ ових експеримената је да се процени и анализира:

1. Статистичка значајност разлика између перформанси 80 компонентних алгоритама, мерена на 15 скупова података.
2. Тачност предвиђања, време извршавања и комплексност резултујућих модела стабла (просечна пондерисана дубина стабла, број чворова, итд.).

Статистичка значајност разлика у перформансама

Да би се упоредиле разлике између алгоритама, парови 80 алгоритама су тестирани помоћу 2-струке крос-валидације Ф-теста (Alpaydin, 1999), јер је показано да тај тест има довољно статистичке моћи, док држи први тип грешке на ниском нивоу, у поређењу са другим тестовима (Dietterich, 1998). Тако је поређено 3160 парова алгоритама, на сваком од 15 скупова података, и мерено да ли је разлика у перформансама статистички значајна на нивоу 5%. Нулта хипотеза је да су измерене разлике у алгоритмима могле настати као резултат случаности. Подаци у Табели 6 су сортирани на основу тога колики проценат парова је био статистички значајно различит. Пошто се због много понављања тестова, очекује да буде 5% лажних аларма (грешака првог типа), видимо да су измерене разлике изнад тог прага (у просеку је 15% значајних разлика на свим проблемима, док ме максимално 58%). Тиме се показује да постоје разлике у перформансама различитих компонентних алгоритама, тј. да различите компоненте утичу на перформансе, и то се не може објаснити случајношћу, јер су разлике статистички зчачајне.

На основу парова поређења, израчунати су и скорови сваког алгоритма, тако што алгоритам добија 1 поен ако није статистички значајно бољи од другог („нерешено“), 2 поена ако је статистички значајно бољи („победник“) и 0 поена у супротном. Агрегирањем ових скорова добија се поређење алгоритама, приказано на Слици 23.



Слика 23. Агрегирани скорови алгоритама, сортирано по скору (Delibašić et al, 2011)

Са Слике 23 се примећује да постоје две групе алгоритама, са великом разликом међу њима, које смо назвали „добри“ (енг. *best*), са скором од 82 до 87 поена, и „лоши“ (енг. *worst*), са скором од 69 до 74 поена. Са тако дефинисаним обележјима, идентификована су правила која разликују те две групе, а која су приказана у Табели 8.

Табела 8. Правила за сврставање алгоритма у групу „добри“ и „лоши“ (Delibašić et al, 2011)

Rule	CSC	ES	Class
1	“bin”, “sig”		best
2	“all”	“dis”, “gai”	best
3	“all”	“chs”, “gin”, “inf”	worst
4	“mul”		worst

Из Табеле 8 видимо које компоненте су одговорне за добар или лош квалитет целог алгоритма. Видимо да су најбитнији потпроблеми креирања потенцијалних гранања (*CSC*), као и евалуација гранања (*ES*). Од компоненти, вишеструко гранање (*mul*) је индикатор лоше перформансе, док су бинарна гранања (*bin*) и гранања на основу груписања атрибута (*sig*) били индикатори добре перформансе.

Треба напоменути да су ова правила извучена на основу перформанси на свим скуповима података. На специфичном скупу података алгоритми могу имати другачије понашање. Тако алгоритам (!-mul-chs-!), који укључује компоненту за вишеструко гранање, је на подацима “aba” међу „добрим“ алгоритмима, до је у просеку међу „лошим“.

Ова анализа је рађена на основу значајних разлика у паровима алгоритама. Додатна анализа је рађена када све алгоритме оценимо на основу тачности предвиђања, а алгоритме групишемо у групе „добар“, „просечан“, „лош“. Правила која одређују којој групи алгоритам припада се могу добити анализом перформанси свих алгоритама, и та правила су приказана у Табели 9.

Табела 9. Правила за одређивање групе квалитета алгоритма по тачности предвиђања, а на основу коришћених компоненти (Delibašić et al, 2011)

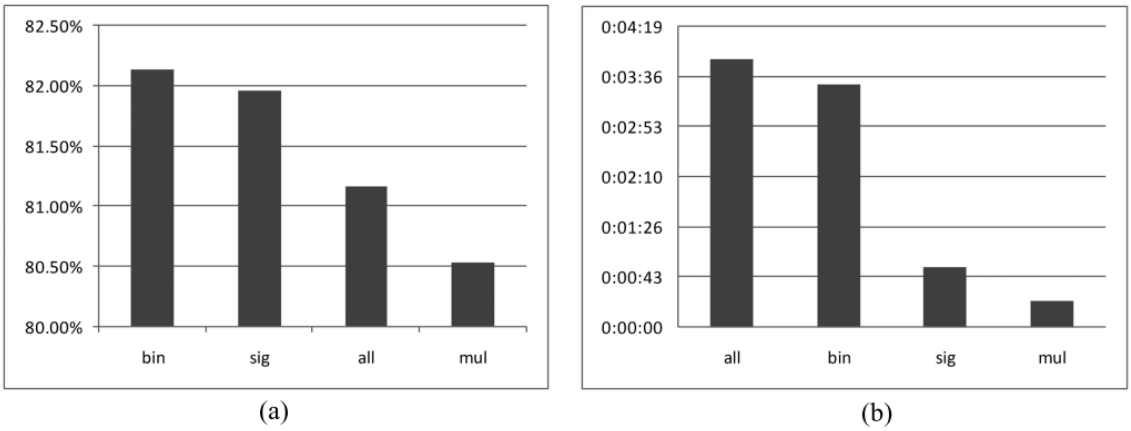
Rule	RIA	CSC	ES	Class
1	“chs/anf”	“all”	“dis”, “gai”	Best
2	“chs/anf”	“bin”, “sig”		Best
3	!	“all”	“dis”, “gai”	Average
4	!	“bin”, “sig”		Average
5	“chs/anf”	“all”	“chs”, “gin”, “inf”	Average
6	“chs/anf”	“mul”		Average
7	!	“all”	“chs”, “gin”, “inf”	Worst
8	!	“mul”		Worst

Од познатих алгоритама, према тачности предвиђања, C4.5 је сврстан као „лош“, CART као „просечан“ и CHAID као „просечан“.

Анализа ефикасности

Када је у питању брзина извршавања алгоритама, у поменутом раду су анализиране просечне брзине алгоритама који поседују неке компоненте. Пример тачности предвиђања у односу на брзину извршавања дат је на Слици 24. Ту се види, поред осталог, да од две компоненте које су дале најпрецизније моделе, „*bin*“ и „*sig*“, друга је знатно бољa по брзини

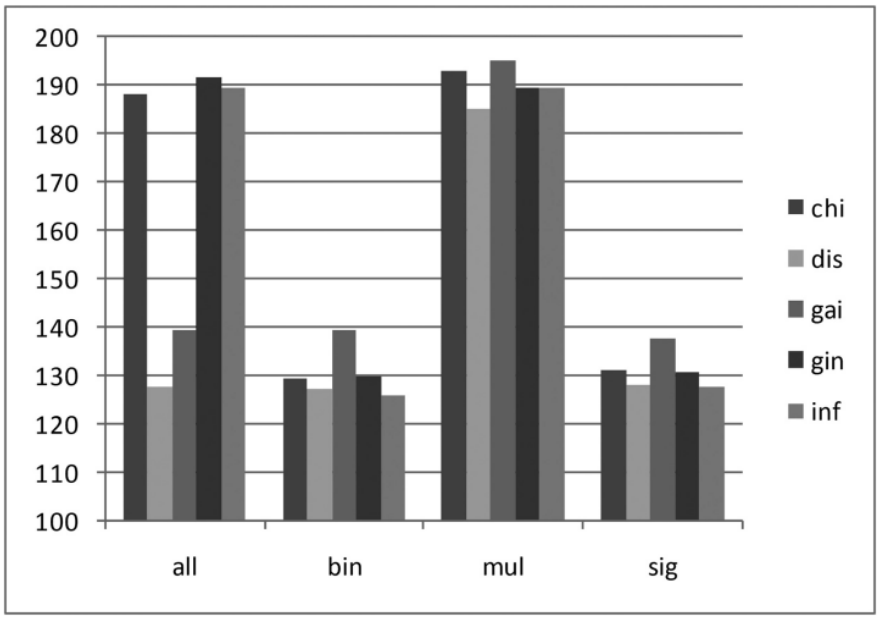
извршавања. Најбрже се извршавају алгоритми који гранају вишеструко („*mul*“), али су зато и најмање прецизни.



Слика 24. Просечне перформансе алгоритама за различите начине гранања: (а) тачност предвиђања, (б) време извршавања. (Delibašić et al, 2011)

Анализа се такође може радити и по томе колико су стабла која су креирана комплексна. Један начин за мерење комплексности је број листова стабла, а колико листова праве алгоритми са различитим компонентама, приказаној је на Слици 25. На слици се види комбинација компоненти два потпроблема, где је на x-оси потпроблем креирања потенцијалних гранања, а различитим ступцима приказане компоенте евалуације гранања. На тај начин се могу уочити међузависности компоненти, на пример, компонента гранања „*all*“ даје комплексна стабла када се евалуира са „*chi*“, док доста једноставнија стабла (мање листова) ако се евалуира компонентом „*dis*“.

Овде је наведен преглед основних резултата и закључака у вези експерименталне анализе приступа компонентних алгоритама, чиме се даје подршка овом приступу и хипотезама. Више детаља и анализа могу се наћи у (Delibašić et al, 2011; Vukićević et al, 2012a).



Слика 25. Број листова у стаблима, зависно од компоненте креирања потенцијалних гранања (x-оса) и компоненте евалуације гранања (различити ступци) (Delibašić et al, 2011)

3.3.3. Евалуација корисничког доживљаја

Компонентни приступ алгоритмима развијен је са основном намером да би се добри делови алгоритма лако поново искористили и у другим алгоритмима. Стога је сваки компонентни алгоритам дефинисан избором делова (компоненти), уместо традиционалног давања имена алгоритмима. Зато је тај систем назван системом „белих кутија“, у контрасту са алгоритмима као „црним кутијама“ у којима се осим назива алгоритма (и евентуално неког параметра) корисницима не открива пуно.

Зато се јавља хипотеза да су компонентни алгоритми корисни и за едукацију. Превише сакривања детаља онемогућава корисника да „паметно“ користи алгоритам. Са друге стране, превише откривања детаља алгоритма могу такође бити негативни, јер то корисника оптерећује и врло често одбија од коришћења алгоритма. Компонентни алгоритми су негде у средини, где откривају делове архитектуре решења, док сакривају имплементацију.

У радовима (Delibašić et al, 2012a; Delibašić et al, 2013) је испитиван ефекат компонентих алгоритама на ефективност коришћења, перцепирано разумевање, перцепирану корисност, и друге параметре корисничког доживљаја.

Студије су рађене на 118 и 51 студената Факултета организационих наука, где су студенти решавали проблеме класификације, коришћењем система белих или црних кутија. У поређењу са алгоритмима као црним кутијама, показано је да су компонентни алгоритми код корисника стварали осећај бољег разумевања, иако су затевали су више напора за прихватање. Тачности које су постигли компонентим алгоритмима су такође били у равни са класичним присупом. Стога је у (Delibašić et al, 2012a; Delibašić et al, 2013) препоручено како да се компонентни алгоритми користе у едукацији. Такође је показано како на основу когнитивног стила студенти могу да се групишу у групе којима више одговара један или други приступ рада.

Компонентни алгоритми стога имају улогу да омогуће лакше креирање нових алгоритма комбинујући делове постојећих, да омогуће анализу корисности компоненти, али и да омогуће боље разумевање рада алгоритама приликом едукације.

4. Аутоматско генерисање алгоритама

Размењујући компоненте алгоритма, отвара се могућност за креирање већег броја алгоритама, састављених из делова неколицине постојећих алгоритама. Штавише, даљом анализом побољшања сличних алгоритама из литературе, могуће је открити још добрих делова алгоритама и додатано обогатити скуп компоненти.

Проблем настаје јер је избор које компоненте користити остављен кориснику (путем интефејса приказаног на Слици 11), а он са већим избором све теже зна да изабере праве компоненте. Штавише, неке компоненте имају и параметре, па цео процес састављања алгоритама из делова и параметризација постаје још тежи. Самим тим, ручно састављање алгоритама убрзо постаје немогуће, већ је неопходно аутоматизовати састављање алгоритама из компоненти.

Такође, растом броја компоненти и претрага свих комбинација компоненти постаје немогућа, јер број алгоритама доживљава комбинаторну експлозију и цео простор је тешко енумерисати. Стога приступи решавања сировом силом (енг. *brute force*) су ограничени на мање проблеме.

Са друге стране, по (Wolpert, 1996), јасно је да би велики простор донео боље резултате алгоритама, јер ће садржати више специфичних алгоритама који могу бити бољи за изабрани проблем који се решава.

Једно решење је да се направи „паметнија“ метода претраге, која би пронашла добре алгоритме из великог простора (не мора неопходно оптималне), али тако да не мора да обиђе цео простор.

Изградња оваквог система, његова имплементација, евалуација и примена је основни допринос ове тезе.

4.1. Простор алгоритама и оптимизациони проблем

Потрага за најбољим (или добрим) алгоритмом у великом простору се може формулисати као оптимизациони проблем.

Функција циља оптимизације је процена тачности предвиђања модела стабла изграђеног неким алгоритмом. Простор претраживања су комбинације компоненти које чине један алгоритам. Стога је ово проблем комбинаторне оптимизације, где се претражује дискретан, набројив и коначан скуп могућих решења.

Ипак, простор решења може бити јако велики. Са тренутним бројем компоненти (Табела 2) укупан број могућих алгоритама је: 2x1x4x6x4x6 = 1152 алгоритама. Са откривањем нове две компоненте сваког потпроблема, број решења је: 27648. Генерално, простор расте у односу на раст димензија на следећи начин:

- Додавањем нових компонената за само један потпроблем, простор расте $O(n)$, где је n број компоненти.
- Додавањем нових компонената за све потпроблеме, простор расте $O(n^к)$, где је к број потпроблема. Пошто је к фиксно у овом контексту, раст је полиномијални, са великим степеном.
- Додавањем нових потпроблема, простор расте $O(n^к)$, у зависности од к које се увећава, па је стога раст експоненцијални.

Додатни разлог зашто је овај проблем тежак за оптимизацију је што је израчунавање функције циља врло временски захтевно. За процену тачности предвиђања, потребно је извршити алгоритам над скупом података и измерити његову тачност, а то се препоручено ради 10-струком крос валидацијом, да не би бирали претриниране моделе, што значи да се цео поступак понавља 10 пута. Нека решења на овај проблем ће бити описана у наредном поглављу. Иако у овој поставци је функција циља само тачност предвиђања, могу се замислити апликације где је битно оптимизовати и интерпретабилност добијеног стабла, брзину предвиђања нових инстанци,

или слично. Оваквим вишекритеријумским функцијама би израчунавање функције циља било још спорије.

Цео проблем претраге се може приказати као математички модел:

$$\min F\left(x_{11}, x_{12}, .., x_{1n_1}, x_{21}, x_{22}, \dots, x_{2n_2}, \dots, x_{mn_m}\right)$$

п.о.

$$\sum_{j}^{n_i} x_{ij} = 1, \quad i = 1, m$$

$$x_{ij} \in \{0,1\}$$

где је i један од m потпроблема, док је j једна од n_i компоненти потпроблема i. Промењиве x_{ij} су индикатор промењиве које означавају да ли је нека компонента део алгоритма или није. Ограничења нам гарантују да је за сваки потпроблем изабрана једна, и не више, компоненти за решавање. Функција F је процењена тачност стабла изграђеног изабраним алгоритмом, коришћењем 10-струке крос валидације. Иако су ограничења линеарна, функција F није линеарна, а ни конвексна, па је тешко користити неку оптимизациону технику из богатог скупа за такве проблеме. Такође, последња ограничења сврставају овај проблем у бинарно програмирање, што је специјални случај целобројног програмирања.

Битно је напоменути да су неки потпроблеми опциони, па није неопходно изабрати компоненту за њихово решавање. Да не би уводили додатна ограничења, овај проблем је решен увођењем нултих (енг. *null*) компоненти, које су део избора као и друге компоненте, али не раде практично ништа у алгоритму.

Над овако дефинисаним простором ипак је могуће уочити неку структуру, а то је локалност решења. Можемо рећи да су два решења (алгоритма) блискија, ако деле већи број компоненти. Тако су решења која се разликују у једној компоненти најближа (такозвана Хамингова удаљеност). С обзиром да компоненте могу поседовати и параметре, можемо рећи да је алгоритам сличнији ако има исте компонете, а различите параметре, него ако се

разликује по избору компоненти. Овако дефинисане локалности решења могу помоћи оптимизационим техникама у претрази, јер се очекује да је функција циља „глатка“ у локалном простору, тј. мање одступа у блиским решењима него у удаљеним.

У описани модел је могуће додати и додатна ограничења. Постоји могућност да се одређене компоненте једног потпроблема могу уклопити само са подскупом компоненти из неког другог потпроблема. Оваква ограничења би смањила простор претраге, што је позитивно, али би отежала рад методама оптимизације, које треба да се крећу кроз простор допустивих решења. Иако у нашој поставци тренутно таква ограничења не постоје, решење које се имплементира треба да нуди могућност укључења таквих ограничења.

Такође, приказан модел треба проширити и претрагом параметара изабраних компоненти. Ово се може посматрати и као изолован проблем, где се за изабране компоненте само оптимизују параметри, што је повољно јер постоји доста напора и решења из литературе које раде оптимизацију параметара, а која се могу употребити. Са друге стране, уколико метода оптимизације дозвољава, могао би се претраживати сложени простор компоненти и њихових параметара.

Најзад, треба рећи да се претрагом простора заправо генеришу алгоритми за стабла одлучивања, који раније нису постојали, а који, комбинујући компоненте из различитих оригиналних алгоритама, износе најбоље из сваке од њих. Уколико неки нови алгоритам покаже изванредне перформансе, може постати нови бенчмарк за поређење и за даљу употребу, чак и као монолитни, алгоритам као црна кутија.

4.2. Еволуциони алгоритам за претрагу

Како за функцију циља (тачност класификације стабала) није познато да има корисне математичке особине (попут линеарности, конвексности, итд.), а

треба решавати глобалну оптимизацију, мета-хеуристике су разуман кандидат за метод претраге.

Иако је могуће применити било коју мета-хеуристику, у овом раду је коришћена једна специфична, наиме Еволуциони Алгоритми (ЕА).

Изучавање како се механизми из природне еволуције могу искористити у оптимизацији се највише ослањају на Холаднов рад у изучавању *Генетских алгоритама* (Holland, 1975). Холанд је препознао да природа еволутивним процесом заправо спроводи оптимизацију јединки, тако да максимизује њихове шансе за преживљавање. Јединке које дуго опстају у својој околини имају висок **степен прилагођености** (енг. *fitness value*). Начин на који природа мења јединке, и самим тим претражује простор могућих јединки, је кроз генетичке операторе. Приликом стварања нове јединке, генетски код пролази кроз **мутацију**, где се одређени мањи делови мењају на случајан начин. Постоји (мала) вероватноћа да се било који део генетског кода измени мутацијом. Такође, у сваком тренутку постоји већи број јединки које су активне, и које се називају **популација**(или **генерација**). У генерацији се јединке такође укрштају како би произвеле јединке наредне генерације, а нови потомци јединки приликом **укрштања** добијају по део генетског кода од сваког родитеља. Основна идеја генетских алгоритама јесте да се било који проблем оптимизације може представити овим концептима, и спровести еволутивни процес којим се тражи (али не увек постиже) глобални оптимум неке функције.

Посматрано са становишта оптимизације, у еволутивним алгоритмима фигурирају следећи појмови:

- **Функција прилагођености**, представља функцију чији глобални ектремум тражимо.
- **Јединка**, представља једну тачку у простору претраге (једно решење), за коју можемо израчунати колико је добра коришћењем функције прилагођености.

- **Генерација или Популација**, представља скуп тачака у простору претраге које се тренутно евалуирају и на којима се спроводе генетски оператори.
- **(Генетски) код или Хромозом**, представља репрезентацију тачке у простору (тј. јединке). Основне координате тачке се репрезентују неким системом кодирања (нпр. бинарним кодом, реалним бројевима, стањима, итд.) и називају се **Гени**. Над изабраним генетским кодом се спроводе генетски оператори.
- **Мутација** је генетски оператор који узима једну тачку из генерације и помера је на случајан начин унутар околине. На коју стране ће се тачка померити се бира сасвим случајно, а неке јединке могу да остану и непромењене. Гледајући репрезентацију тачке генетским кодом, мутација случајно мења једну (или мањи број) координату кода (тј. ген). Вероватноћа мутације је типично врло ниска.
- **Укрштање** је генетски оператор који узима две тачке у простору и креира нову тачку која је интерполација узете две тачке, тј. налази се у простору између изабране две тачке. У тренуцима када генерација садржи различите јединке, овим оператором се тежи да се добије комбинација јединки која узима добре стране од оба родитеља. Наравно, када генерација конвергира ка сличним јединкама, много су мање шансе да овај оператор креира нови квалитет од родитеља који су врло слични. Такође, који делови родитеља се комбинују бира се сасвим случајно, али ако се креира бољи потомак од родитеља, он ће имати знатно више шанси да се пренесе у другу генерацију.
- **(*Природна*) Селекција** је процес избора које јединке ће формирати следећу генерацију, а избор јединки зависи од степена прилагођености. Дакле бира се случајни узорак из постојеће (модификоване) популације, али по расподели функције циља, тако да боље тачке имају већу вероватноћу да преживе.

Генетски алгоритам стога се може описати следећим псеудо-алгоритмом:

1. Иницијализује се почетна популација, нпр. изаберу се случано неки број тачака (*величина генерације* је параметар алгоритма).
2. Над тренутном генерацијом се примењују генетски оператори:
    a. Бира се свака јединка и са предорећеном вероватноћом се модификује свака позиција у генетском коду. Ово ствара нове (мутиране) јединке у генерацији.
    b. Бирају се случајно парови јединки из популације, и парови размењују делове генетске репрезентације. Тиме настају нове јединке које се придодају популацији.
3. Све јединке у популацији се евалуирају функцијом прилагођености и врши се избор нове генерације, тако да боље јединке имају већу вероватноћу преживљавања.
4. Уколико није задовољен критеријум заустављања (нпр. максималан број генерација, укупно време извршавања, стабилност популације, итд.) враћа се на Корак 2.

Треба напоменути да је овакав алгоритам велика симплификација природног процеса еволуције, који има много комплексније механизме. Нека унапређења генетских алгоритама укључују додатне детаље инспирисане биолошком еволуцијом, попут старости јединки и међусобне борбе за преживљавање. Ипак, циљ ових алгоритама није да се разуме динамика биолошких система, већ да се преузму идеје у решавању глобалне оптимизације које се у природи могу запазити.

Са математичког становишта, овај алгоритам делује произвољно, јер иако је инспирисан природом, нема јасне математичке постулате на којима гарантује неки ниво резултата. Ипак, у својој књизи (Holland, 1975) Холанд показује познатом „Шема Теоремом“ да генетски алгоритам заиста има смисла и математички. Ова теорема говори да у генетском алгоритму кратке шеме (тј. поднизови у генетском коду) са изнад-просечном функцием прилагођености имају експоненцијално већу шансу да преживе целу

еволуцију. То значи да ће крајње генерације у еволуцији садржати надпросечне делове генетског кода и самим тим решења (тачке) које су близу оптималним. Гаранције асимптотске глобалне оптималности ипак нема, као ни код других метода глобалне (неконвексне) оптимизације.

Треба напоменути да генетски алгоритми (у ужем смислу речи) се односе на Холандову верзију, која има своје специфичности. Кодирање је у генетским алгоритмима увек било бинарно, па је било неопходно репрезентовати простор претраге бинарним (генетским) кодом. Када је репрезентација бинарна, онда се могу користити већ постојећи генетски оператори за бинарне кодове. Мутација онда значи инвертовање бинарне цифре (0 у 1 и обратно), док се укрштање ради тако што се изабере позиција растављања генетског кода (код оба родитеља) и дете има генетски код левог родитеља до позције растављања, а код десног родитеља након те позиције.

Ипак, бинарна репрезентација јединки није увек погодна. Стога су генетски алгоритми генерализовани на такозване *Еволутивне алгоритме* (Michalewicz, 1996). Идеја је да се изабере природнија репрезентација јединки (решења), али се онда не могу користити постојећи генетски оператори за бинарне кодове, већ се морају осмислити специјални генетски оператори за изабрано кодирање. Пример оваквих алгоритама биће дат у наставку, пошто је то начин како је реализовано решење за компонентне алгоритме.

На крају, треба рећи и да еволуциони алгоритми спадају у врсту алгоритама који се називају метахеуристике, тј. оптимизационе алгоритме за општу глобалну оптимизацију, али без гаранција налажења глобалног оптимума. Постоји прегршт метахеуристика у литератури, а такође и у подгрупи биолошки инспирисаних метахеуристика, којој припадају и еволуциони алгоритми. Све метахеуристике, генерално гледано, имају стратегије за диверзификацију (ширења простора претраге) и за концентрацију (сужавање на потпростор добрих решења), а код еволуционих алгоритама те стратегије се огледају у генетским операторима мутације и укрштања.

За реализацију алгоритма за претрагу компонентних алгоритама у овом раду су коришћени еволуциони алгоритми (Michalewicz, 1996), са специјализованом репрезентацијом и генетским операторима, као и разним унапређењима који одговарају специфичности проблема који се решава. Наредна потпоглавља описују детаље те имплементације.

4.2.1. Функција прилагођености

Сваки јединка (хромозом) представља један компонентни алгоритам. Функција прилагођености оцењује квалитет алгоритма и директно утиче на његову вероватноћу преживљавања. За алгоритме за стабла одлучивања, квалитет се мери тиме колико добро модели стабла предвиђају нове случајеве (грешка генерализације). Да би проценили тај квалитет, користи се крос-валидација, тј. процес провере тачности предвиђања на издвојеном тест скупу, објашњено у Поглављу 3.3. Самим тим, желимо да бирамо и еволуирамо оне алгоритме који се показују као добри за предвиђање. Извршавање компонентних алгоритама се спроводи у WhiBo окружењу, описаном у Поглављу 3.2.

Специфичност ове функције прилагођености је да је временски скупа за рачунање, јер је потребно спровести алгоритам над подацима који изграђује стабло, и тестирати га на новим подацима, најчешће понављајући ту процедуру више пута. Овај проблем је више наглашен на већим скуповима података, где се и алгоритми дуже извршавају. Једно решење за овај проблем ће бити дато у Поглављу 4.2.4.

4.2.2. Репрезентација

Генерички алгоритам за изградњу стабала одлучивања је дизајниран као избор низа компонената које решавају одређене потпроблеме. Стога је за кодирање алгоритма (тачке у простору претраге) природно да прати исту структуру. То би значило да постоји посебан ген за сваки потпроблем, а да је хромозом заправо низ гена који кодирају потпроблеме. Док бинарним

кодирањем код генетских алгоритама сваки ген има исте вредности (0 или 1), у овом случају сваки ген (потпроблем) има различити број вредности, који одговара броју могућих компоненти за тај потпроблем. Могуће вредности гена се такође у терминологији еволутивних алгоритама називају **алели** (енг. *alleles*). Ипак, иако је хромозом састављен из различитих гена, генетски оператори ће моћи на исти начин да мењају алеле сваког од гена.

Пример хромозома за компонентне алгоритме је дат на Слици 26, а представља хромозомску репрезентацију *ID3* алгоритма. Ген који кодира стратегију гранања има вредност „Вишеструко гранање“ (*Multiway*), док ген који кодира избор мере евалуације има вредност „Информациона добит“ (*Information gain*). Гени за потпроблеме *RIA* и *Prune tree* имају вредност „!“, што значи да је изабрано да не буде орезивања стабла и председекције атрибута.

Gene (sub-problem)	RIA	Create split	Evaluate split	Stop criteria	Prune tree
Alleles (RC)	!	Multiway categorical	Information gain	Tree depth (10)	!

Слика 26. Хромозом са генима који репрезентују избор компоненти за ID3 алгоритам (Jovanovic et al, 2014)

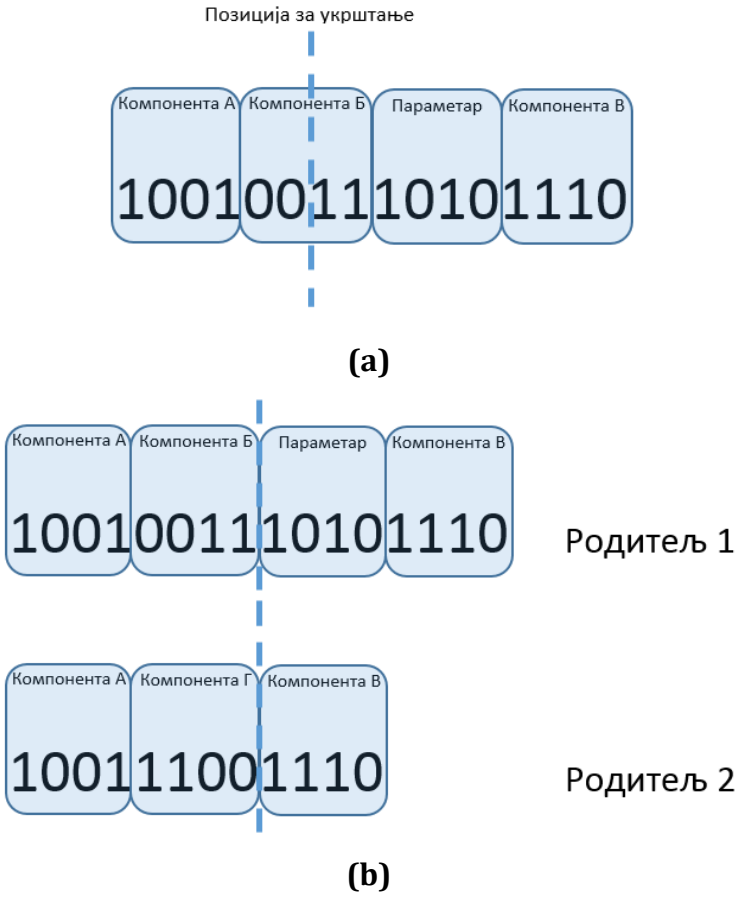
Треба напоменути и да гени кодирају не само избор компоненте, него и избор параметра за компоненте које имају параметре. Ово ће захтевати да генетски оператори могу да утичу на компоненте и/или параметре. Пример кодираног параметра у гену се види и на Слици 26, за ген који кодира избор начина заустављања гранања (*Stop criteria*).

Да је репрезентација компонентних алгоритама била бинарна, било би више проблема:

- Бинарни кодови за сваки избор компоненте би били спојени један уз други, али алгоритам не би знао где је граница између бинарног кодирања суседних потпроблема. То би значило да оператором

укрштања хромозом родитеља могао бити „исечен“ на пола потпроблема, што је илустровано на Слици 27(а).

- Такође различити хромозоми би били различите дужине (због параметара који су специфични за компоненте), па би укрштање или мутација могли да произведу алгоритам који има параметар који му не припада. Ово је илустровано на Слици 27(б).
- Најзад, случајним ивертовањем битова, приликом мутације, би се могао добити код који је „неправилан“, тј. ко се не кодира ни један алгоритам. Када се ово деси, такве јединке би требало да остранимо, али тиме би успорили рад еволуционог алгоритма, и онемогућили ефективну еволуцију.



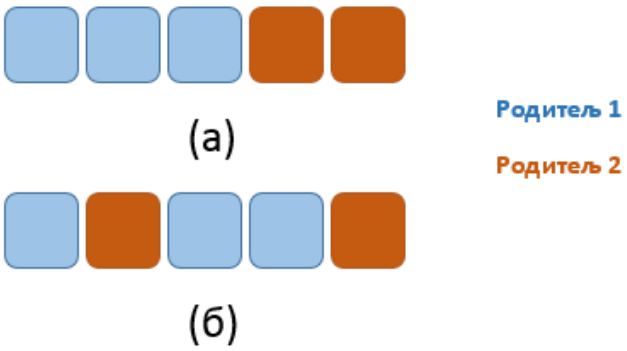
Слика 27. Проблеми бинарног кодирања компонентних алгоритама

Сви ови проблеми неће постојати са изабраним специфичним (небинарним) кодирањем. Терет имплементације ће стога понети дизајнирање специфичних генетских оператора, који производе увек валидне јединке.

4.2.3. Оператори

Оператор укрштања треба да омогући да се размене гени два родитеља. Најчешће се у генетским алгоритмима користи укрштање на основу изабране позиције цепања хромозома. Тако увек имамо гене лево и десно од случајно изабране позиције цепања. Иако је такав оператор валидан, у нашем случају би он био рестриктиван, јер би врло често чувао суседне потпроблеме, иако је редослед набрајања потпроблема произвољан. Слика 28(а) приказује како би изгледало укрштање на основу позиције цепања.

Да би превазишли тај проблем, коришћено је такозвано *униформно укрштање*, где се случајно бира пола потпроблема који ће бити наслеђени од једног родитеља, док ће остатак бити преузет од другог родитеља, независно од позиције и редоследа навођења потпроблема у хромозому. Илустрација таквог укрштања дата је на Слици 28(б).



Слика 28. Могућа укрштања: (а) позиционо и (б) униформно

Оператор мутације се примењује на свакој јединки, али сваки ген има предодређену (обично ниску) вероватноћу да буде мутиран. Гени који буду изабрани да мутирају мењају изабрану компоненту за потпроблем који

кодира дати ген. Мутација случајно бира једну од могућих компоненти, тако да је свака компонента подједнако вероватна. Такође, оператор мутације може оставити постојећу компоненту, али утицати само на параметар. Да ли ће мутација мењати компоненту или параметар ће зависити од унапред дефинисаног параметра за вероватноћу мутације. Уколико бисмо подесили да је вероватноћа мутирања параметра већа од мутирања компоненте, тиме би изрекли став да је алгоритам са истим компонентама а различитим параметрима „ближи“ неко алгоритам са различитим компонентама. Тиме можемо усмеравати претрагу више на компоненте или више на параметре.

Механизам природне селекције се спроводи класичним „рулет коцкањем“. Све јединке у популацији се поређају на котур рулета, али тако да већи простор на рулет котуру заузму јединке које су боље прилагођене (тј. алгоритми који имају бољу моћ предвиђања). Окретањем рулета се добија избор јединке, и овакав случајан избор се понавља онолико пута колико је дефинисана величина популације. Ово наравно одговара вишеструком узорковању јединке из расподеле вероватноће чија је функција густине пропорционална прилагођености јединке.

Како свако решење у популацији има вероватноћу са којом је изабрано у наредну генрацију, постоји могућност да се изгубе чак и добра решења, игром случаја. Због тога се користи и стратегија **елитизма**, где се најбоље јединке у популацији аутоматски бирају у следећу генерацију, без обзира на процес природне селекције. На тај начин се и памти најбоље решење нађено кроз целу еволуцију, и оно ће сигурно бити и део финалне генерације.

4.2.4. Остале специфичности

Како је напоменуто у Поглављу 4.2.1. функција прилагођености је рачунски скупа, јер је потребно извршити алгоритам (више пута због крос-валидације) на изабраним подацима. Неколико стратегија су развијене како би се решио овај проблем, тј. да би се отклонило ово уско грло у укупном времену извршавања еволутивног алгоритма.

Прво, имплементирана је **кеш меморија** за већ прерачунате функције прилагођености. Током еволуције, исти алгоритми су често појављују у наредној генерацији. Разлог може бити вишеструк: (а) јединка можда није претрпела измене, јер се генетски оператори примењују са одређеном вероватноћом; (б) јединка је пренесена у наредну генерацију због елитизма, јер је показала јако добар квалитет; (в) генетски оператори могу креирати нови хромозом (јединку) који има исте особине као нека јединка у прошлости. Кеширање вредности функција омогућава да се уштеди време, јер се спречава поновно рачунање. Кеширане вредности са друге стране заузимају меморијски простор, па се може одредити и максимално заузеће простора кеш меморије. Кеш је имплементиран као „хеш“ (енг. *hash*) мапа, тако да је време приступања елементу у кешу сразмерно времену израчунавања хеш функције, што је константно, тј. рачунске сложености O(1) у односу на величину кеш меморије. У експериментима приказаним у Поглављу 4.4 ће бити дискутована и ефективност кеш меморије.

Још једна стратегија која је коришћена да се убрза рад еволутивног алгоритма, је увођење **сурогат функција прилагођености**. Сурогат функције су апроксимације оригиналне функције прилагођености, али такве да се доста брже рачунају, а дају излаз који се не разликује значајно од оригиналне функције. Пошто оригинална функција прилагођености је евалуација предвиђања алгоритма и њена комплексност расте са величином скупа података, разумна апроксимација квалитета алгоритма се може добити евалуацијом алгоритма на мањем узорку података. Ако је узорак репрезентативан, онда се ни процена тачности стабла на узорку неће бити далеко од праве тачности. Репрезентативност узорка се постиже случаним узорком над оригиналним скупом податка. Варирањем процента узорка (10%, 30%, 50%, 70%) може се контролисати колика је уштеда у времену, а колико је апроксимација непрецизна. Узорковање се може радити једанпут унапред или се може радити у свакој генерацији другачије, како би се избегла пристрасност само једног узорка.

Да би се направио баланс брзине извршавања и прецизности, еволуциони алгоритам се извршава у **две фазе**. У првој фази се користи сурогат функција прилагођености, која се извршава брзо, али уз грешку апроксимације оригиналне функције. Ова фаза траје предодређен број генерација, и служи да грубо изолује добре јединке. Након ње остаје популација која садржи велики број квалитетних јединки, који улазе у другу фазу. У другој фази, алгоритам користи оригиналну функцију прилагођености, која је спорија, али прецизнија, а извршава се мањи број генерација. Прва фаза је доста брза у извршавању, и она прави апроксимацију добре популације, док је друга фаза спорија, и само фино (и прецизно) подешава решења добијена првом фазом. На овај начин, само мањи број алгоритама се процењује на целом скупу података, што значајно штеди време извршавања.

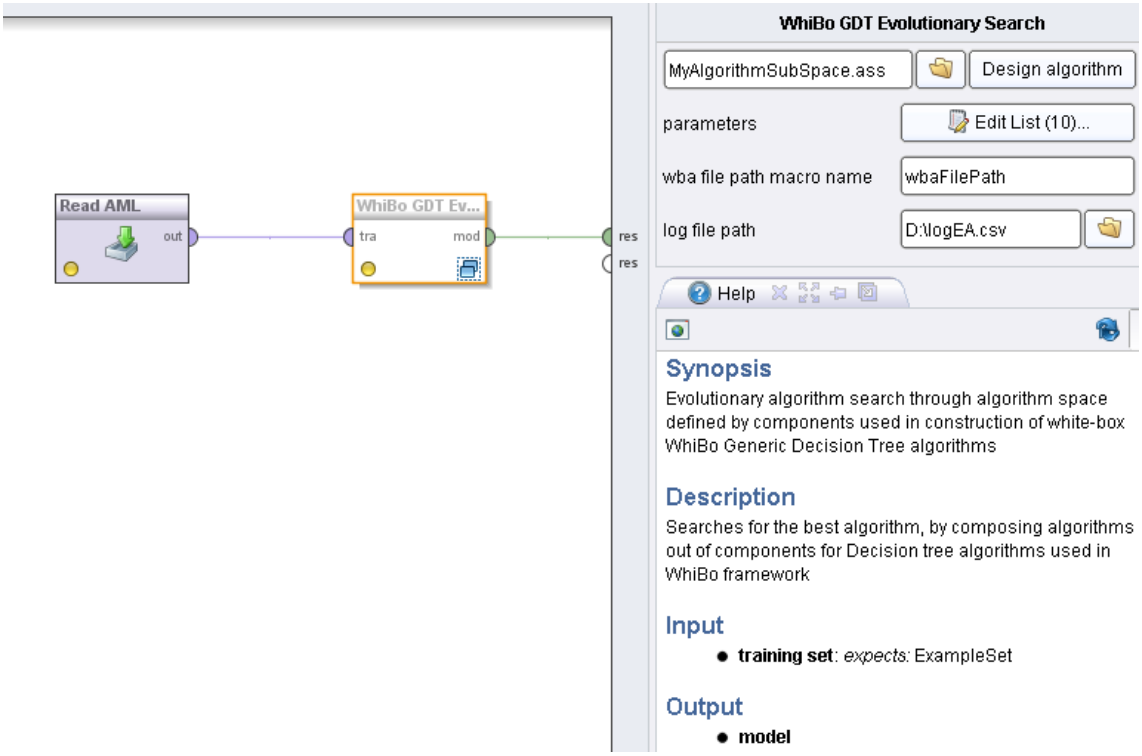
Још један битан детаљ у дизајну еволуционог алгоритма, је случајно узорковање за крос валидацију. Крос валидација издваја узорак за тестирање, како би се модел стабла одлучивања евалуирао на посебним подацима и тиме измерила грешка генерализације. Проблем може настати када се кроз генерације користи исти узорак за тестирање, па укупан алгоритам може претренирати на тај тест узорак. Стога се у свакој генерацији другачије (случајно) прави тест узорак за крос-валидацију. Уколико неки алгоритам даје добре резултате само на једном тест узорку у једној генерацији (претрениран на тај узорак), у наредној генерацији ће бити кажњен јер не генерализује добро.

4.3. Имплементација

Еволутивна претрага алгоритама за стабла одлучивања по простору свих комбинација компоненти је имплементирана коришћењем JGAP библиотеке у Јава програмском језику (Meffert, 2011). Ова библиотека помаже изградњу специфичних репрезентација и генетских оператора, а нуди и лак начин за надградњу, која је неопходна због спцифичиности описаних у Поглављу 4.2.4. Имплементација у Јава програмском језику је такође погодна јер је потребно

интегрисати овај систем са *Rapid Miner* платформом (Mierswa et al, 2006), у којој су реализовани и *WhiBo* компонентни алгоритми за стабла одлучивања (Delibasic et al, 2011).

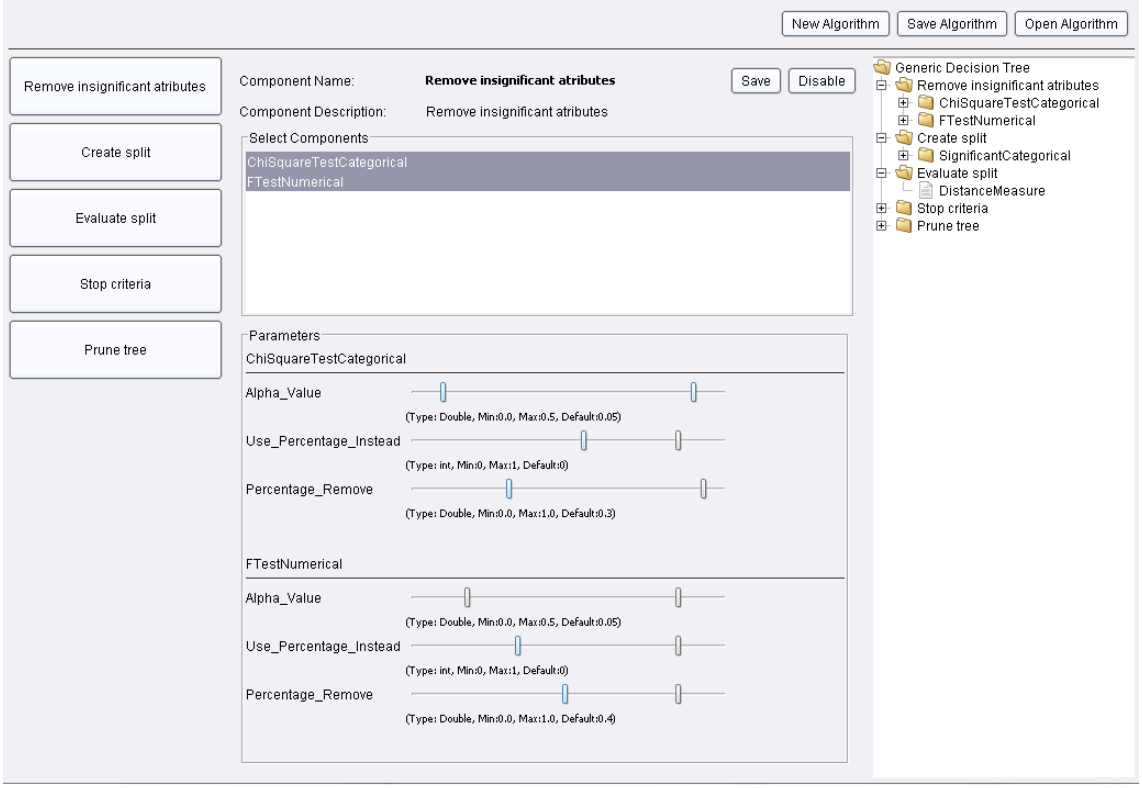
Цело решење је спаковано у додатак (*plugin*) *Rapid Miner* софтвера, где се увожењем додатка нуди нови оператор „*Whibo GDT Evolutionary Search*“ који позива имплементацију еволуционог алгоритма, а приказан је на Слици 29. Овај оператор се понаша као алгоритам за учење стабла, јер на улазу прима скуп података, а на излазу даје најбољи модел стабла који је добијен победничким алгоритмом из целог процеса еволуционе претраге. Овако имплементиран опратор се може користити у многим процесима откривања законитости у подацима, који су подржани *Rapid Miner* платформом.



Слика 29. Коришћење Еволутивног алгоритма у *Rapid Miner* окружењу (Jovanovic et al, 2014)

У делу за подешавање оператора (Слика 29, десно), корисник може подесити еволуциони алгоритам. Прво нуди се могућност дефинисања простора претраге. Подразумевано је да се претражују простор свих компоненти и

параметара, али се може дефинисати и неки потпростор од интереса, избором опције „Design Algorithm“. Тада се отвара прозор призакан на Слици 30, који доста подсећа на интерфејс *Whibo* генеричког алгоритма за стабла (Поглавље 3.2.).



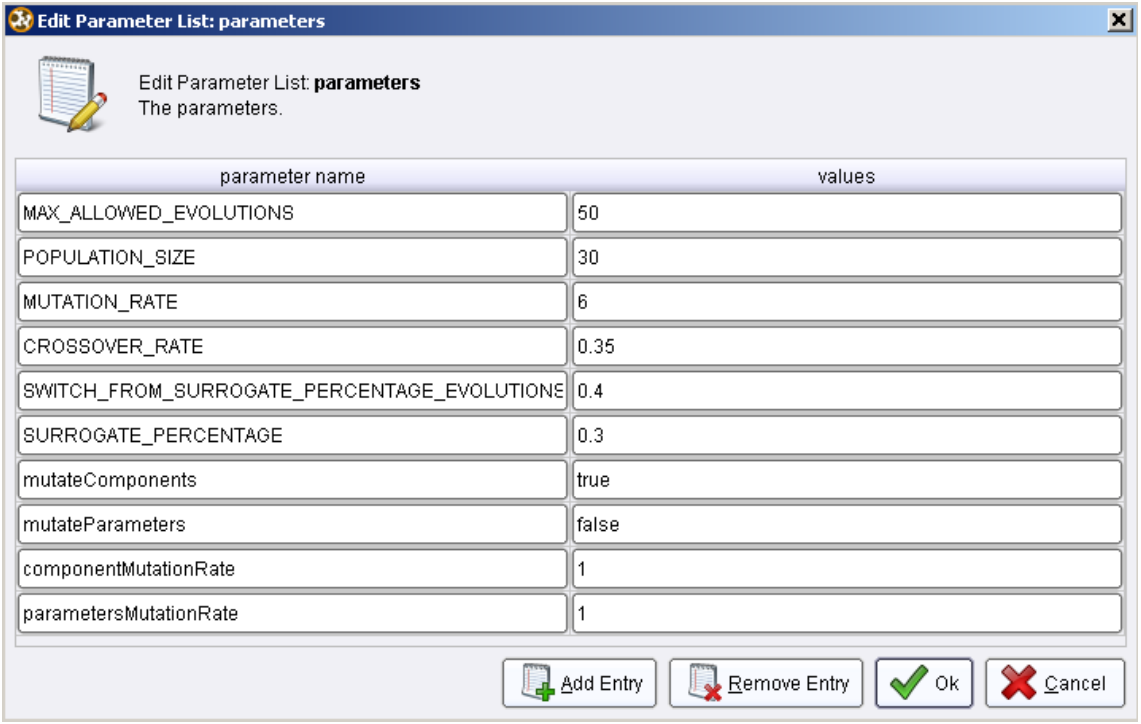
Слика 30. Дефиниција простора претраге од стране корисника (Jovanovic et al, 2014)

На овом делу интерфејса се бирају компоненте које ће бити размењиване у еволутивном процесу, као и распон параметара који ће бити претраживан. Овим корисник може ограничити простор претраге на компоненте и параметре које сматра „разумним“. Подешени простор претраге се чува у фајлу изабраног назива, са екстензијом *.ass* (од *algorithm search space*).

Омогућено је и да се понуђене компоненте (за дефинисање простора претраге) аутоматски динамички ажурирају из репозиторијума доступних компоненти у Whibo окружењу.

Друга опција за подешавање еволуционог алгоритма (Слика 29, десно) је избор вредности параметара који утичу на сам начин одвијања еволуционог процеса. Изабиром ове опције отвара се прозор приказан на Слици 31, који дозвољава подешавање параметара који су помињани у предходном поглављу, а чије је значење:

- MAX_ALLOWED_EVOLUTIONS – максималан број генерација (итерација примене генетских оператора и природне селекције)
- POPULATION_SIZE – број јединки (компонентних алгоритама) у популацији
- MUTATION_RATE – вероватноћа мутације сваког гена. Целобројни број X представља једну мутацију у X гена.
- CROSSOVER_RATE – вероватноћа укрштања, као проценат јединки у популацији које су изабране за укрштање.
- SWITCH_FROM_SURROGATE_PERCENTAGE_EVOLUTIONS – проценат првих генерација које користе сурогат функцију прилагођености, од укупног броја генерација. Накнадне генерације користе оригиналну функцију прилагођености.
- SURROGATE_PERCENTAGE – проценат узорковања скупа података за сурогат функцију.
- mutateComponents – избор да ли мутација мења компоненте.
- mutateParameters – избор да ли мутација мења параметре компоненти.
- componentMutationRate – вероватноћа да мутација (када се деси) промени компоненту. Овим параметром се претрага фокусира на компоненте.
- parametersMutationRate – вероватноћа да мутација (када се деси) промени параметре. Овим параметром се претрага фокусира на праметре.



Слика 31: Подешавање параметара еволуционог алгоритма (Jovanovic et al, 2014)

На крају, подешавање оператора (Слика 29, десно) омогућава подешавање лог фајла, који ће бележити прогрес еволуционог алгоритма, компонентне алгоритме који су евалуирани, њихове перформансе, као и додатне статистике о извршавању (број евалуираних алгоритама, време извршавања, коришћење кеш меморије, итд). Изучавањем овог лог фајла касније се може анализирати рад и ефекат алгоритма. Исечак једног лог фајла се може видети на Слици 32.

RIA	CreateSplit	EvaluateSplit	StoppingCriteria	Prune	Performance
null	SignificantCategorical(0.05; 0.049)	ChiSquare	null	PessimisticError(0.25)	91.33%
RIA	MultiwayCategorical	GainRatio	LeafLabelConfidence(0.95)	CostComplexity(0)	75.72%
RIA	MultiwayCategorical	InformationGain	LeafLabelConfidence(0.95)	ReducedError	77.26%
null	SignificantCategorical(0.05; 0.049)	InformationGain	MinNodeSize(30)	ReducedError	87.29%
null	All	DistanceMeasure	null	null	93.06%

.
.
.

Cache cleared----------
Number of values returned from cache: 613
Number of evaluations of fitness function: 170
Execution time: 04:02
Cache cleared----------

null	BinaryCategorical	ChiSquare	null	null	98.09%
null	BinaryCategorical	ChiSquare	null	PessimisticError(0.25)	98.09%
RIA	All	ChiSquare	null	null	91.61%
null	SignificantCategorical(0.05; 0.049)	ChiSquare	null	PessimisticError(0.25)	97.22%
null	All	ChiSquare	null	PessimisticError(0.458)	88.94%
RIA	BinaryCategorical	ChiSquare	null	PessimisticError(0.473)	97.34%
null	BinaryCategorical	ChiSquare	null	PessimisticError(0.0472)	98.09%

.
.
.

The best solution fitness value: 98.32%
Best Solution:

null	BinaryCategorical	ChiSquare	null	PessimisticError(0.0473)

Number of values returned from cache: 1061
Number of evaluations of fitness function: 123

Слика 32. Пример дела лог фајла еволуционог алгоритма (Jovanovic et al, 2014)

Из оваквог лог фајла се може видети стање популација и перфомансе алгоритама у популацији. На пример прва популација (Слика 32, горе) има алгоритме са врло различитим компонентама, као и различитим перформансама тих алгоритама. Касније генерација (Слика 32, доле) је конвергирала на неке компоненте, док су друге изгубљене због лоше перформансе. Такође у каснијим генерацијама већина алгоритама добро предвиђа. У крају лог фајла се такође види и коначно најбољи алгоритам.

Додатне информације из лог фајла су и:

- *Number of evaluations of fitness function* – број јединствених алгоритама који су евалуирани у свакој фази.

- *Number of values returned from cache* – колико је пута коришћена кеш меморија, што говори колико је времена уштеђено на евалуацијама предходно виђених алгоритама.
- *Execution time* – време извршавања.

Треба напоменути и како се креиране јединке евалуирају у овој имплементацији. Еволуциони алгоритам заправо еволуира само генетске репрезентације алгоритама, али се те репрезентације трансформишу у ".wba" XML фајл, који разуме Whibo оператор за генеричка стабла. Тај фајл дефиниције једног алгоритма се прослеђује оператору за генеричке алгоритме, који извршава алгоритам и након тога проверава тачност крос-валидацијом. Стога је додатак (*plugin*) за еволуциону претрагу зависан од *Whibo* додатка (*plugin*).

4.4. Експериментална провера

Први део експерименалне провере има намеру да утврди да ли је претрага алгоритама еволуционим алгоритмом ефективна. Ово заправо проверава да ли су решења претраге блиска или једнака оптималном решењу, када је могуће знати тачно оптимално решење. Такође је циљ да се блиско-оптимално решење нађе у што мање евалуација тачака у простору.

Овај експеримент обухвата 960 компонентних алгоритама, који се добијају комбиновањем компоненти приказаних на Табели 10. Приказани скраћени кодови ће бити коришћени за компактни приказ компонентних алгоритама (! означава одсуство компоненте за потпроблем), нпр. C4.5 алгоритам има код: !-B-M-GR-!-P

Сви параметри компоненти су фиксирани на подразумеване вредности, како би могли да претражимо овај простор и сировом силом (енг. *brute force*), тј. енумерисањем и тестирањем свим могућих тачака у том простору. Та тривијална стратегије сигурно није ефикасна, али има гаранцију најбољег решења, са којим ће проверавати најбоље решење које пронађе еволуциони

алгоритам. Уколико евлуциони алгоритам проналази добра решења, онда имамо потврду да га можемо користити и на много већем простору алгоритама, где сирову силу није могуће употребити.

Табела 10. Компоненте које сачињавају простор претраге (Jovanovic et al, 2014)

Sub-problem	Reusable component	Abbreviation and code (in brackets)	Parameters
Remove insignificant attributes – optional –	Chi square/ anova F test	chs/anf (C)	Alpha (def. 0.05, min 0, max 1)
Create split (numerical)	Binary	bin (B)	
Create split (categorical)	Binary	bin (B)	
	Multiway	mul (M)	
	Significant	sig (S)	Merge alpha (def. 0.05, min 0, max 1) Split alpha (def. 0.05, min 0, max 1)
	All	bin + mul + sig (A)	
Evaluate split	Chi square	chs (C)	
	Information gain	inf (I)	
	Gain ratio	gai (GR)	
	Gini	gin (G)	
	Distance measure	dis (D)	
Stop criteria – optional –	Maximal tree depth	mtd (D)	Tree depth (def. 10, min 1, max 20)
	Minimal node size	mns (S)	Node size (def. 30, min 1, max 1000)
	Leaf label confidence	llc (C)	Confidence (def. 0.95, min 0, max 1)
Prune tree – optional –	Pessimistic	pe (P)	Confidence (def. 0.25, min 0.1, max 0.5)
	Error pruning	pe (P)	
	Cost complexity pruning	cc (C)	Use−1stdev (def. 0, min 0, max 1)
	Minimal error pruning	me (M)	
	Cost complexity pruning	cc (C)	Use−1stdev (def. 2, min 0, max 1000)
	Reduced error pruning	re (R)	
	Minimal leaf size	mls (S)	Leaf size (def. 30, min 1 max 1000)

Иницијална популација се одређује потпуно случајно, насумично изабирајући компоненте за све алгоритме иницијалне популације. Репрезентација алгоритама (хромозома), као и генетски оператори, су исти како је описано у Поглављу 4.2. Функција циља је тачност предвиђања проверена 10-струком крос-валидацијом. Параметри самог еволуционог алгоритма су приказани на Табели 11. Поред раније помињаних параметара, пријављен је и фактор случајности (енг. *random seed*), који је коришћен за генератор псеудо-случајних бројева. Ово је битно, пошто су делови еволутивног алгоритма

стохастичке природе, па експерименти не би били поновљиви и не би могли да се верификују.

Табела 11. Изабрани параметри еволуционог алгоритма (Jovanovic et al, 2014)

Parameter	Value
Population size	15
Generations	30
Mutation rate	12.5%
Crossover rate	35%
Percentage of generations with surrogate function	70%
Surrogate dataset sample size	30%
Random seed in RapidMiner	1

Цео експеримент је рађен на више (16) скупова података, како би могли да се извуку генералнији закључци. Сви скупови података су јавно доступни (Asuncion, Newman, 2007), а њихови називи и основне карактеристике дати су у Табели 12. Приказани су: величина података (*size*), број атрибута (*attributes*), број класа (*classes*), присуство недостајућих вредности (*missing values*), дистрибуција класа, тј. број случајева у свакој класи (*class distribution*).

Табела 12. Коришћени скупови података за експерименте (Jovanovic et al, 2014)

ID	Dataset	Size	Attributes	Classes	Missing values	Class distribution
Bre	Breast	286	9	2	Yes	201/85
Car	Car	1728	6	4	No	1210/384/69/65
Cmc	Contraceptive	1473	9	3	No	629/511/333
Cra	Credit-a	690	14	2	Yes	383/307
Crg	Credit-g	1000	20	2	Yes	700/300
Har	Hayes roth	160	5	3	No	65/64/31
Hep	Hepatitis	155	19	2	Yes	123/32
Hyp	Hypothyroid	3772	21	3	Yes	3488/191/93
Krk	King-rook-vs-King-pawn	3169	36	2	No	1669/1527
Lid	Liver-disorders	345	7	2	No	200/145
Lym	Lymphography	148	18	4	No	81/61/4/2
Nur	Nursery	12960	8	5	No	4320/4266/4044/328/2
Pos	Post-operative	90	8	3	Yes	64/24/2
Spe	Spectrometer	531	102	2	Yes	172/15
Sta	Statlog heart	270	13	2	No	150/120
Vot	Voting	435	16	2	Yes	267/168

Резултати су приказани у Табели 13. За сваки скуп података еволуциони алгоритам је предложио најбољи алгоритам, за који је измерена тачност предвиђања и разлика најбољег могућег алгоритма из целог простора и пронађеног (*Max(Brute Force)-EA*). Тако видимо колико је нађени алгоритам

далеко од оптималног. Поред тога, у табели се виде и најбоља и најгора могућа тачност свих алгоритама у простору. Најзад, приказано је и колико је функција евалуирано сурогат функцијом прилагођености (која се доста брзо извршава), а колико оригиналном функцијом прилагођености. Ово нам даје информацију колика је уштеда у времену у односу на потпуну претрагу.

Табела 13. Резултати поређења пронађеног и најбољег компонентног алгоритма (Jovanovic et al, 2014)

Data-set	Proposed algorithm	Max (Brute force) – EA accuracy	Max (Brute force)	Min (Brute force)	No. of evaluations with surrogate function (out of 960)	No. of evaluations with fitness function (out of 960)
Bre	C-S-C-S-P	2.77%	76.60%	64.06%	130	63
Car	!-A-GR-!-!	0.17%	98.09%	76.85%	80	38
Cmc	C-B-I-C-C	0.13%	55.73%	43.18%	94	49
Cra	C-A-G-C-S	0.00%	86.38%	77.83%	115	64
Crg	!-M-D-S-R	0.00%	74.90%	65.10%	79	32
Har	!-A-GR-C-!	0.62%	83.75%	37.50%	122	43
Hep	!-A-D-S-P	2.54%	84.46%	76.08%	163	45
Hyp	!-A-I-!-P	0.03%	99.79%	95.44%	132	53
Krk	!-B-I-!-!	0.00%	99.69%	94.09%	109	35
Lid	!-M-C-C-P	7.55%	68.41%	57.97%	97	32
Lym	!-B-C-D-M	0.05%	83.19%	54.81%	106	43
Nur	!-S-D-!-P	0.00%	99.88%	83.05%	88	40
Pos	C-S-I-C-S	0.00%	71.11%	50.00%	136	52
Spe	!-B-I-S-S	0.00%	91.99%	81.73%	155	72
Sta	C-B-C-C-R	5.92%	81.85%	55.56%	89	25
Vot	C-S-GR-!-M	0.01%	96.55%	61.38%	123	41

Резултати показују да еволуциони алгоритам проналази у већини случајева решења која су врло близу оптималном. У 12 од 16 случајева, пронађен је алгоритам који је мање од 1% лошији од најбољег, а у 14 од 16 случајева мање од 3% лошији од најбољег. Најгори резултат је на подацима “Lid”, где је пронађен само просечан алгоритам. Ови резултати сугеришу да еволутивни алгоритам успева да пронађе „добре“ компонентне алгоритме за стабла, за више различитих примера података.

Битно је приметити да еволуциони алгоритам у просеку прави 45 пуних евалуација тачака у простору, од укупно 960 алгоритама, што указује на то да је претрага такође ефикасна (мање од 5% простора је проверавано).

Такође, изучавајући лог фајлове, примећено је да је кеш меморија у великој мери убрзала извршење алгоритма. На различитим подацима, укупно је

евалуирано између 111 и 227 функција прилагођености (заједно сурогата и оригиналне), од укупно 450 хромозома (јединки) који су се појавили током еволуције (30 генерација * 15 хромозома по генерацији). То значи да је кеш меморија убрзала време од 98 до 305%.

Други део експерименталне провере је била шира претрага, која је укључује и параметре компоненти. У том случају није могуће применити потпуну претрагу (сирову силу), па су као бенчмарк коришћени предходни резултати који су постигли разни алгоритми за учење стабала одлучивања (који нису компонентни), а који су објављени на (Blockeel, 2006). Ови бенчмарк резултати представљају базу ранијих експеримената, а заправо укључују и алгоритме за стабла који нису разматрани у овој студији, и који имају компоненте изван простора који претражује еволуциони алгоритам, али су ипак јавни резултати са којима се може поредити, а добијени су на истим подацима који су јавно доступни. Такође, за проверу квалитета ови бенчмарк резултати су користили исти начин евалуације, тачност предвиђања коришћењем 10-струке крос-валидације, па је поређење могуће.

Резултати су приказани у Табели 14, где се (као и раније) приказује најбољи алгоритам који пронађен претрагом компонентног простора, његова тачност, стандардна девијација, као и тачност најбољег компонентног алгоритма без оптимизације параметара (нађеног у предходном експерименту). Поређење са предходним експериментом нам може рећи колико су параметри битни у оптимизацији. Стандардна девијација добијена је понављањем евалуације 30 пута, са различитим фактором случајности (*random seed*). Последње две колоне приказују назив и квалитет најбољег алгоритма из (Blockeel, 2006).

Еволуциони алгоритам у овом експерименту је радио 50 генерација, са популацијом величине 30. Вероватноћа укрштања је била 35%, док је вероватноћа мутације повећана на 16.67 (због мутирања и параметара). У

првих 30 генерација мутиране су само компоненте, док у последњих 20 генерацију су мутирани само параметри, за компоненте које су преживеле први део еволуције. Ово је рађено јер када компонента мутира, она аутоматски анулира предходно побољшање параметара предходне компоненте. Најбољи резултат за сваки скуп података у Табели 14 је подвучен.

Табела 14. Резултати поређења бенчмарк резултата из (Blockeel, 2006) и алгоритма конструисаног еволуцијом (Jovanovic et al, 2014)

Data-set	Best algorithm design found by EA	Accuracy	Std. Dev.	Best accuracy found in the first experiment	Best decision tree	Accuracy from best decision tree
Bre	C - M - D - C (0.50) - R	75.24%	1.49%	73.83%	C4.5	76.22%
Car	! - B - C - ! - P (0.05)	98.32%	0.27%	97.92%	C4.5	97.28%
Cmc	! - B - C - S (727) - C (0)	56.61%	0.72%	55.60%	C4.5	56.08%
Cra	C - M - G - C (0.21) - S (86)	86.67%	0.23%	86.38%	ADTree	85.51%
Crg	C - B - I - C (0.05) - C (1)	75.10%	0.88%	74.90%	LMT	75.90%
Har	! - A - D - D (12) - !	85.62%	1.21%	83.13%	C4.5	71.43%
Hep	C - M - I - S (731) - R	83.88%	0.01%	81.92%	C4.5	85.16%
Hyp	C - S (0.076; 0.042) - GR - D (9) - M (13)	99.79%	0.04%	99.76%	C4.5	99.60%
Krk	! - B - I - ! - P (0.26)	99.69%	0.07%	99.69%	LMT	99.75%
Lid	C - S (0.24; 0.03) - G - ! - M (424)	69.86%	0.30%	60.86%	C4.5	70.72%
Lym	! - B - G - D (10) - M (221)	83.19%	1.55%	83.14%	LMT	83.11%
Nur	! - S (0.28; 0.018) - D - ! - !	99.88%	0.04%	99.88%	C4.5	99.51%
Pos	! - M - G - S (719) - S (305)	71.11%	0.00%	71.11%	Rep-tree	71.11%
Spe	C - A - D - S (50) - C (1)	91.99%	0.00%	91.99%	C4.5	91.99%
Sta	! - B - I - S (343) - P (0.06)	81.85%	0.01%	75.93%	LMT	84.44%
Vot	C - B - GR - ! - M (204)	96.54%	0.01%	96.54%	LMT	96.78%

Резултати показују да је пронађени компонентни алгоритам компететиван са најбољим из базе експеримената (Blockeel, 2006). За 9 од 16 скупова података пронађен је најбољи алгоритам, и он сада може постати бенчмарк. На другим подацима резултати су упоредиви, са максимално 2% разлике (осим на подацима „Sta“). Неколико пута је компонентне алгоритме победио LMT (*Logistic Model Tree*) (поменут у Поглављу 2.3), који у листовима стабла ради нешто што тренутно не ради ни један компоненти алгоритам, тј. користи логистичку регресију. Додавањем ове компоненте у репозиторијум, и тиме проширивањем простора претраге, компоненти алгоритам би могао да покаже и боље резултате.

На “Lid” подацима, где је у предходном експерименту пронађени компонентни алгоритам био најлошији, сада даје доста боље резултате, где је само 1% лошији од бенчмарка. Напредак у односу на предходни експеримент је 9%, што говори да је подешавање параметара на овом скупу података врло корисно.

Такође, у односу на предхони експеримент, на простору са укљученим параметрима је увек пронађено боље или једнако решење. Ипак, побољшање мењањем параметара, иако конзистентно, није велико, што говори да је размена компоненти доста важнија од оптимизације параметара, како је и сугерисано у (Suknović et al, 2012)

Коначно, ови експерименти (објављени у (Jovanovic et al, 2014)) показују да се интелигентним претраживањем простора компонентних алгоритама може конструисати алгоритам који има добре перформансе на специфичном проблему и подацима. Резултати још нису конклузивни, али дају јаку потврду да је ова линија истраживања потентна.

4.5. Примена у предвиђању успеха студената

Да би добили додатне потврде корисности овог приступа, цела методологија је примењена на још једном реалном проблему, наиме предвиђању успеха студената, што је објављено у (Jovanovic et al, 2012a).

Анализа успеха студената спада у специфичну област откривања законитости у подацима, *Educational Data Mining* (EDM) (Romero, Ventura, 2010), у којој се анализирају подаци који долазе из едукационих система. Циљеви ове области обухватају анализу података ради (Kumar, Chadha, 2011): организације програма и силабуса, предвиђања уписа студената, предвиђање успешности студената, откривање превара, идентификације грешака, итд.

Успешност студената је један од параметара многих одлука, укључујући селекцију за стипендије, сарадњу и запослење, па је рана детекција успешних

студената битан задатак. Предвиђање успеха се може радити на основу података из студентске службе, или из система за електронско учење, што је приказано у (Jovanovic et al, 2012b). Ова студија показује да је могуће предвиђати успех студената после само једне године студирања, применом алгоритама за стабла одлучивања. У већој фамилији могућих компонентних алгоритама, претрагом тог простора могу се наћи алгоритми који су добри за овај конкретан задатак.

Дакле, циљ задатка је: Предвидети коначан успех студената (коначну просечну оцену), након завршетка прве године студија. За изградњу модела користи се скуп података од 366 студената, описаних атрибутима датим у Табели 15. Класа представља индикатор да ли је студент "одличан", тј. да ли има просек већи од 9.0

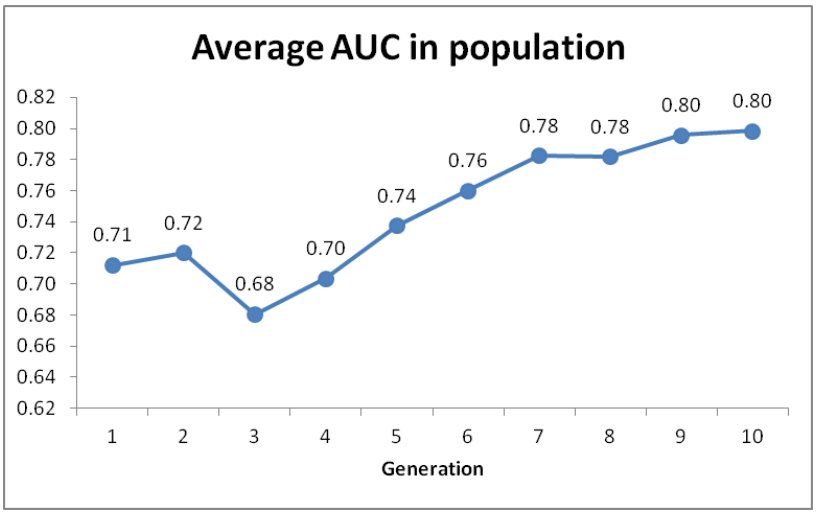
Табела 15. Променљиве коришћене у предвиђању успеха студената

	Variable	**Variable description**
Input variables	*Gender*	Student gender Value: male/female (1/0)
	Score on enter qualification exam	Points achieved in the entrance qualification exam Values: 40-100
	Study –programme	Choice of study program in time when student is enrollment in Faculty Values: Information system and technology program and Management program
	Grades at the first year of studies	Grades: marks of each of 11 subjects: examines at the first year of studies Values: 6-10
Output variables	*Students' success*	Students' success at the end of studies Valus: "Excelent" (GPA>=9.0) and "Other" (GPA<9.0)

После пуштања еволуционог алгоритма, добија се модел стабла одлучивања које има AUC 0.8 (у просеку после вишеструке евалуације), што сугерише висок ниво квалитета предвиђања. Овај резултат се може протумачити на следећи начин: уколико би случајно изабрали једног одличног и једног не-

одличног студента, очекује се да модел у 4 од 5 пута успешно препозна који од њих двојице је одличан.

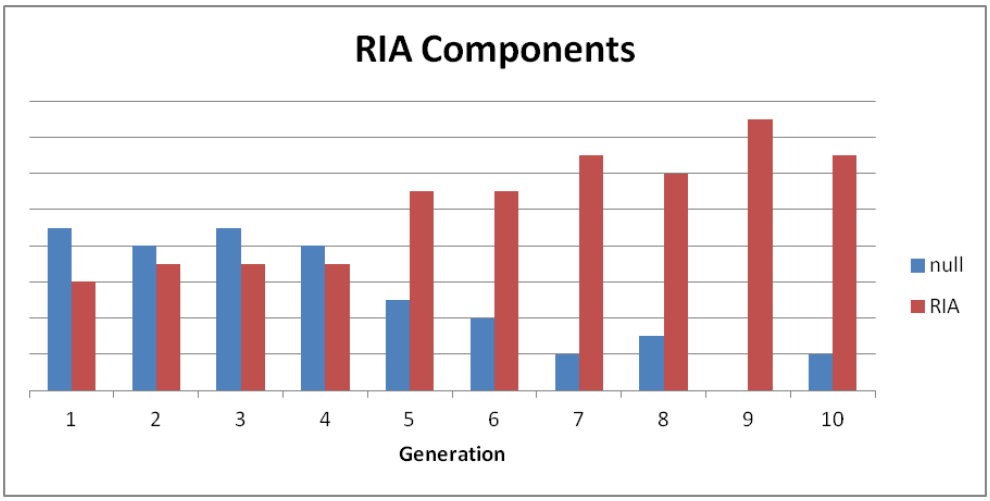
Још занимљивије од високог AUC скора најбољег алгоритма је еволуција квалитета алгоритама кроз генерације, приказана на Слици 33. Ту видимо да кроз генерације, популације систематски садрже боље и боље јединке (алгоритме) за овај проблем. Ово сугерише да еволуција заиста бира и креира популације које максимизирају изабрани квалитет.



Слика 33. Просечна мера AUC кроз генерације еволуционог алгоритма

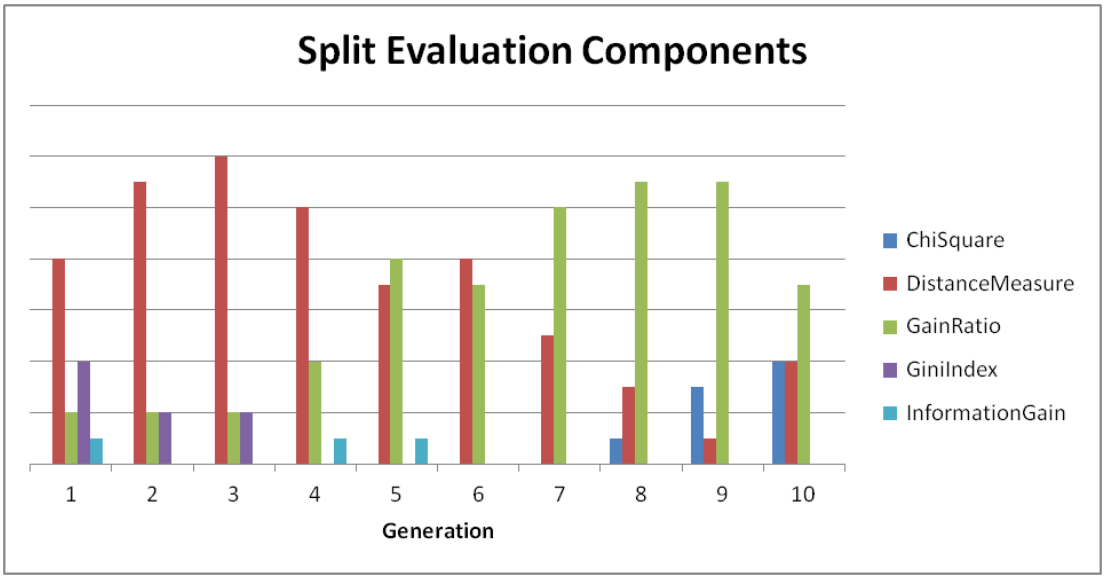
Додатно, интересантно је видети које компоненте су присутне у популацијама кроз еволуцију. На Слици 34 је приказано учешће опционе компоненте *RIA* за преселекцију атрибута. Видимо да како популација еволуира, све више јединки користи ову компоненту, па можемо закључити да је она корисна за "преживљавање", тј. корисна за алгоритме за стабла да би имали већу моћ предвиђања.

Још једна интересантна ствар са Слике 34, је и појављивање "null" компоненте у десетој генерацији, иако у деветој генерацији је ова компонента "изумрела", тј. ниједан алгоритам у популацији је није садржао. Ово говори да оператор мутације игра своју улогу диверзификације, и уноси новине у популације које су конвергирале ка сличним алгоритмима.



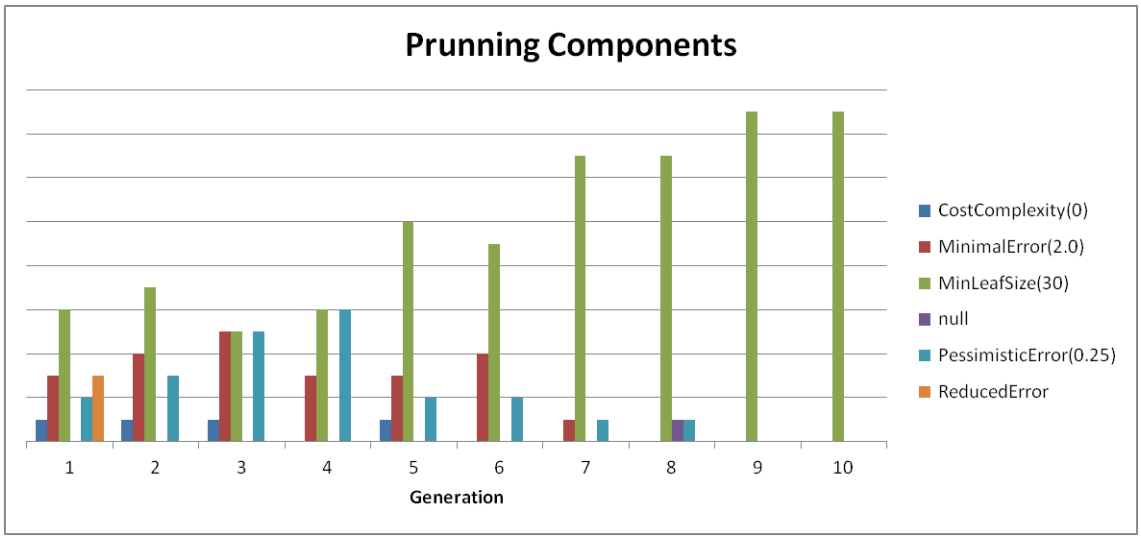
Слика 34. Учешће компоненти потпроблема преселекције атрибута у различитим генерацијама

Слично, на Слици 35 се види учешће компоненти за потпроблем евалуације гранања стабла, кроз генерације. Ту се такође назире да неке компоненте су корисније за добро предвиђање (*GainRatio*), док су неке мање корисне (*InformationGain*).



Слика 35. Учешће компоненти потпроблема евалуационе мере гранања у различитим генерацијама

Коначно, сличну анализу можемо спровести и за разне компоненте за орезивање стабла, приказано на Слици 36. У овом случају је очигледније да је за орезивање стабла најкорисније користити компоненту *MinLeafSize*, са параметром 30.



Слика 36. Учешће компоненти потпроблема орезивања стабла у различитим генерацијама

Овакве анализе омогућавају боље разумевање тога које компоненте су кључне у дизајну алгоритама, за сваки специфичан случај примене. Такође, за разлику од сличних анализа приказаних у Поглављу 3.3, коришћењем еволуционог алгоритма не мора се евалуирати читав простор алгоритама, већ само мањи подскуп, који садржи алгоритме високог квалитета.

5. Сродна предходна истраживања

Проблем претраживања простора алгоритама састављањем компоненти још није директно решаван у литератури, али постоје слични и аналогни проблеми који јесу били предмет скоријег истраживања. Генерални преглед могућности спајања оптимизације (тј. операционих истраживања) и алгоритама ОЗП се може наћи у (Olafsson et al, 2008).

У наставку ће бити приказани приступи који су најрелевантнији овој тези и који, иако представљају алтернативне приступе, дају идеје како се приступ из ове тезе може унапредити.

Селекција алгоритама

Избором алгоритама су се истраживачи и раније бавили, где је семинални рад (Rice, 1976) описао и основне проблеме при том процесу. Добар преглед новијег датума се може наћи у (Smith-Miles, 2009). Напори у овим истраживањима се више базирају на томе како исправно мерити успешност алгоритма и како помоћи одлучивање при избору алгоритма за неки задатак.

Ипак, примена оптимизационих техника није могућа у селекцији алгоритама када се они посматрају као црне кутије, јер не постоји простор алгоритама, већ само неуређени скуп. Стога је могуће једино пробати све доступне алгоритме из скупа, тј. енумерисати цео скуп.

Због тога се предмет ове тезе дефинише у оквиру једне фамилије алгоритама (за стабла одлучивања), где је могуће препознати заједничку структуру тих алгоритама који дефинису простор претраге, који се може паметније претраживати.

Генетски алгоритми за изградњу модела директно

Мета-хеуристике су генерички оптимизациони метод и често су коришћени као алати за претрагу у простору модела за класификацију (нпр. подешавању коефицијената регресија), у односу на простор алгоритама из ове тезе.

Метахеуристике су нарочито корисне када функција грешке која се оптимизује није конвексна. Када су у питању стабла одлучивања, на пример, у (Papagelis, Kalles, 2000) су коришћени генетски алгоритми да би еволуирали стабла, тј. популацију стабала које генетским операторима размењују гране. Слично, у (Kretowski, Czajkowski, 2010) су еволуционим алгоритмима индукована стабла за регресију, уместо класификацију.

Такође, гентски алгоритми су коришћени на лексикографски вишектиретијумким функцијама, за изградњу стабала одлучивања која имају више жељених особина (Basgalupp et al, 2009). У (Cha, Tappert, 2009) у се исто обрађивали генетски алгоритми, али за конструкцију искључиво бинарних стабала.

Поред овога, коришћене су и друге метахеуристике, инспирисане природом. Тако у (Marinakis et al, 2009; Parpinelli et al, 2002; Sousa, 2004) се истражује могућност коришћења оптимизације колонијом мрава, као и оптимизације ројевима, за изградњу класификационих модела.

Додатно, мета-хеуристике су се показале корисним и за помоћ у другим аспектима изградње стабла одлучивања. Тако, хибридни метод за "стабло/генетски алгоритам" је коришћен за решавање проблема малих узорака података у листовима стабла (Carvalho, Freitas, 2004). Такође, генетски алгоритми су коришћени и за унапређење модела стабла одлучивања у односу на цену грешака класификације (Turney, 1995).

Оптимизација хипер параметара

Још један популарни приступ у коришћењу оптимизационих техника у алгоритмима за класификацију је подешавање хипер параметара алгоритма. Многи алгоритми излажу одређене параметре које корисник треба да подеси пре пуштања алгоритма (попут разних параметра за регуларизацију, броја суседа, дубине стабла, итд.) Ови параметри се не подешавају током тренирања (учења) модела и стога се зову хипер-параметри.

Примена генетских алгоритама за подешавање параметара класификатора базираних на фази логици, који се иницијализују стаблима одлучивања, поже се наћи у (Abonyi, 2003). Даље, (Lin et al, 2008) се користи метахеуристика симулираног каљења за оптимизацију параметара за кернел функције у машинама са векторима ослонца. Слично томе, комбиновање симулираног каљења и генетских алгоритама за подешавање параметара кернела је коришћено у (Qilong, 2009). У (Rozsypal, Kubat, 2003) су коришћени генетски алгоритми да узоркују примере и атрибуте како би створили репрезентативније узорке за учење класификатора.

Хипер хеуристике

Алгоритми за изградњу стабала одлучивања су такође врста оптимизационих техника која претражују простор свих могућих стабала која се могу конструисати из доступних атрибута. Та потрага за добрим стаблом се ради из више корака, где су сви ти кораци заправо хеуристике, тј. правила која теже ка бољим стаблима, али без гаранција. Сам алгоритам за стабла је такође хеуристика, која се назива и "похлепна" (greedy) хеуристика.

Хипер-хеуристике су познати методи (Burke et al, 2003) за избор које хеуристике користити код разних оптимизационих проблема. Ако се стога свака компонента може сматрати хеуристиком за решавање потпроблема, процедуре које оптимизују избор тих хеуристика се могу третирати хипер-хеуристикама, као у (Vella et al, 2009).

Генетичко програмирање

Још један приступ који је аналоган решењу из ове тезе, је приступ генетског програмирања (Koza, 1991). Ту се на много нижем нивоу алгоритми представљају појединачним инструкцијама, чиме се могу описати било који алгоритми. Генетски алгоритми се онда користе да оптимизују низ инструкција, како би се добио алгоритам који решава неки проблем. Познат пример еволуирање низа инструкција како би се добио алгоритам за сортирање бројева. Функција сваког низа инструкција је у којој мери је решен

проблем, на пример колико су бројеви сортирани коришћењем неког низа инструкција. Генетским операторима се онда модификују низови инструкција, док се не добију јединке које успешно решавају задати проблем.

Проблем код овог приступа је што је на јако ниском нивоу (инструкција), што би за компликоване алгоритме, какви су алгоритми ОЗП, било врло непрактично, јер је простор могућег низа инструкција превелики.

Компоненте у приступу коришћеном у овој тези се могу третирати врстом инструкције, само на високом нивоу, тј. инструкције доменски специфичног језика (Fowler, 2010). Претрага таквог простора је стога много ефикаснија, јер избори тих инструкција увек стварају алгоритме које, барем у некој мери, решавају задати проблем.

Учење система за класификацију

Учење система за класификацију (енг. *Learning Classifier System*) је приступ који је поставио Холанд, отац генетских алгоритама. Добар увод се може наћи у (Urbanowicz, Moore, 2009). Идеја је да се еволуирају системи правила, тако да се добије скуп правила који добро класификују инстанце неког проблема. Постоје варијанте у којима су јединке скупови правила, а оне размењују појединачна правила генетским операторима; док у другој варијанти се тражи комбинација правила из једног супер-скупа правила.

Овај приступ заправо оптимизује модел за класификацију, а не сам алгоритам који креира модел, што одговара приступу описаном на почетку овог поглавља.

Сличност са моделима стабла је и у томе што се изграђена стабла могу представити као низ правила, и користити као класичан систем базиран на ако-онда правилима.

Аутоматско генерисање алгоритама

Аутоматско генерисање алгоритама је ипак било предмет скорашњих истраживања. У (Schimpf, Wallace, 2002) се описује оквир за тражење хибридне комбинације алгоритама, али не алгоритама за класификацију, већ алгоритама за комбинаторну оптимизацију, са применом на познати CAT проблем. У књизи (Pappa, Freitas, 2010) аутори дискутују систем за аутоматски дизајн алгоритама за откривање законитоси у подацима и детаљније анализирају примену на системе класификације помоћу правила. Аутори такође користе еволуционе алгоритме као метод претраге. Најзад, у (Barros et al, 2012) се такође користе еволуциони алгоритми за оптимизацију алгоритама за стабла одлучивања, што врло резонира са приступом у овој тези, и даје валидацију да постоји мотив да се даље ради на овом проблему.

Мета учење

У истраживањима о избору који алгоритам користити у спрецификној ситуацији, постоји и приступ мета-учења (Smith-Miles, 2009), где је идеја да се прате перформансе алгоритама на разним скуповима података, и да се покуша научити када који алгоритам даје добре перформансе. Из великог броја предходних експеримената евалуације алгоритама прави се нови мета-скуп података, који садржи мета-атрибуте скупова података (разне статистике о подацима, типовима података, итд.) и постигнут квалитет алгоритма. Над тим мета-скупом података се тренира модел за предвиђање перформанси алгоритма.

Овај приступ дакле има исти циљ као и ова теза, а то је да изабере најбољи алгоритам из скупа постојећих. Притом се захтева постојање базе експеримената, сличе оној у (Blockeel, 2006). Са друге стране, овим се предвиђа квалитет алгоритма без пуштања алгоритма на подацима, већ предвиђањем на основу мета-података. Због тога се решење добија доста брже од приступа у овој тези, али са одређеном грешком предвиђања.

Такође, за мета-учење је неопходно да постоје записи у бази експеримената за сваки алгоритам за који се предвиђа. За разлику од тога, еволуциона претрага спорије евалуира алгоритам, али нема грешку у евалуирању алгоритма, а такође и обилази само део алгоритама у току претраге. Такође, за еволуциону претрагу није неопходно априори знање о понашању алгоритма, садржано у базама експеримената.

На крају, ова два приступа нису само алтернативни начини за решавање избора алгоритама, већ комплементарни приступи, где један предвиђа перформансе, док други мери перформансе на мањем делу простора. Стога би за оба приступа било корисно наћи начин интеграције оба приступа.

6. Дискусија и Закључак

Проблеми предвиђања на основу историјских података су присутни у готово свим организацијама, а њихово решавање је битно за доношење рационалних одлука на свим нивоима. Један од често коришћених модела за предвиђање је и **стабло одлучивања** (Поглавље 2), а алгоритми који из историјских података уче овакав модел зависности су изучавани деценијама уназад. У литератури је описано више алгоритама за ову намену, али ниједан није доминантан у свим применама, већ је потребно испитати који алгоритам је погодан за одређену примену.

Генерализацијом постојећих алгоритама у виду **компонентних алгоритама** за стабла одлучивања (Поглавље 3) отворио се доста већи **простор алгоритама** (Поглавље 4.1) које је могуће конструисати из делова постојећих. Иако се показује да тај простор садржи алгоритме који су бољи од постојећих познатих алгоритама, претрага тог простора за добрим алгоритмом није једноставна, јер потпуна претрага често није могућа, посебно узимајући у обзир да простор расте убрзано додавањем нових компоненти.

Ова теза се стога бави **изучавањем, имплементацијом и тестирањем ефективних начина за претрагом простора алгоритама, коришћењем метахеуристике еволуционих алгоритама** (Поглавље 4.2 и 4.3). Претрагом се заправо аутоматски генерише нови компонентни алгоритам, који је ефективан за изабрани проблем предвиђања. Еволуциона претрага нема гаранције проналажења глобалног оптимума, али екпериментална провера (Поглавље 4.4) показује да су пронађена **решења приближно добра као оптимална**, боља од постојећих алгоритама, као и да је **претрага временски ефикасна**. Додатно треба напоменути да је изабрана метахеуристика врло погодна за паралелну имплементацију, па би значајна убрзања могла да се добију извршавањем у дистрибуираном окружењу.

Поред генерисаног алгоритма за изабрани проблем предвиђања, врло је корисна и анализа популације алгоритама која еволуира, чиме се може **идентификовати корисност сваке компоненте** за "преживљавање" алгоритма, тј. утицај компоненте на квалитет предвиђања (пример анализе у Поглављу 4.5). Интересантно је и да ће компонента "изумрети" једино ако није корисна у више комбинација у различитим алгоритмима, што је прави тест њеног квалитета. Анализе потенцијалних узрока алгоритама су и раније рађене (нпр. Loh, Shih, 1997), али су рађене ручно и захтевале су доста напора, док су сада те анализе аутоматске, због компонентне природе алгоритама.

Овиме су скупљене **потврде предходно изнетих хипотеза** (Поглавље 1.3) и **створено оправдано веровање** да се овим приступом могу још боље и лакше решавати проблеми предвиђања из података.

Наравно, постоје и доста начина како би се додатно унапредио овај приступ.

Прво, и најочигледније, откривањем нових компоненти и потпроблема би се простор алгоритама значајно проширио и омогућио (аутоматско) креирање још бољих компонентних алгоритама. **Нове компоненте** се могу извлачити из објављених унапређења постојећих алгоритама (Поглавље 2.3), попут гранања на основу више атрибута истовремено (*Oblique* стабла). Гледајући у добре резултате бенчмарк алгоритама у Поглављу 4.4, сугерисано је такође да би било корисно додати и **нови потпроблем**, предвиђања у листу, који не би увек предвиђао већинску класу у листу, већ користио сложенију функцију, попут логистичке регресије.

Такође је могуће применити исти приступ на **друге фамлије алгоритама**. Насупрот стаблима одлучивања за класификацију, врло једноставно би се приступ могао применти и на **стабла за регресију**. Такође, показано је и како се **алгоритми за кластеровање** могу генерализовати у компонентне алгоритме (Delibašić et al, 2009c), као и да то носи значајне користи (Delibašić et al, 2012b), па би претраживање (тј. генерисање) алгоритама за кластеровање била још једна област примене овог приступа.

У овој тези је за претрагу коришћена метахеуристика еволуционих алгоритама, јер омогућава ефикасну претрагу када простор претраге и функција циља немају лепе математичке особине (попут конвексности) које оптимизационе технике могу да искористе. Такође, претрага компоненти је директна реализација "хипотезе градивних блокова" (енг. *building block hypothesis*), која лежи у основи генетских алгоритама.

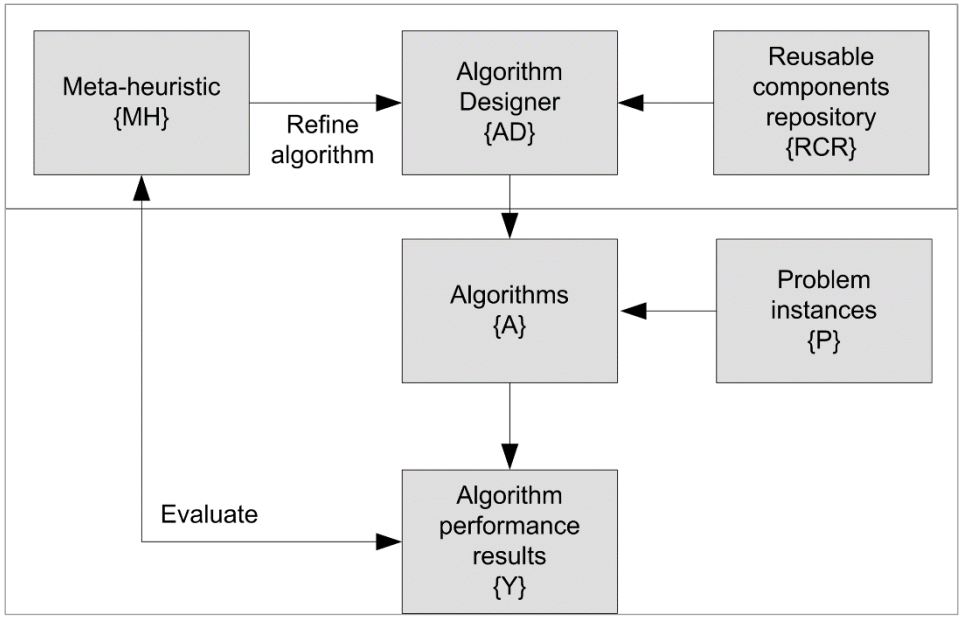
Ипак, за претрагу је могуће користити и **друге доступне метахеуристике**, попут табу претраживања, методе променљивих околина, оптимизација колонијом мрава, итд. Примена сваке од ових метода би затевало посебно прилагођавање проблему претраге компонентних алгоритама, и изградњу специфичне реализације изабране метахеуристике. Тако би, нпр. за примену методе променљивих околина (Hansen et al, 2010), морало да се дефинише неколико околина (суседства), на пример, околина променом параметара, околина променом једне компоненте, итд. Овакве реализације других метахеуристика и поређење са овде примењеним еволуционим алгоритмом, би било добро видети у будућности.

У проблемима конструкције алгоритама за изградњу стабла одлучивања, који су изучавани у овој тези, не постоје ограничења у томе како се могу комбиновати компоненте различитих потпроблема. Свака комбинација је могућа, само су неке успешније у предвиђању, док друге мање. Међутим, лако је замислити да додавањем нових компоненти, потпроблема, или примени на другим фамилијама алгоритама, се појаве ограничења. Та **ограничења** би заправо рекла да је немогуће стартовати алгоритам са одређеним, некомпатибилним, компонентама. Уколико се генетским операторима створи такав **некомпатибилан хромозом**, он би аутоматски био елиминисан из популације. Међутим, ако су ограничења строга, популације ће садржати много оваквих хромозома који ће бити отстрањени, што може да поремети рад еволуционог алгоритма. Стога би се такви случајеви могли решити изменом самих генетских оператора, тако да не дозволе креирање некомпатибилних хромозома.

У Поглављу 5 се такође помиње приступ мета-учења, као алтернативан начин да се изабере добар алгоритам, али без стартовања самог алгоритма, већ на основу предвиђања на основу особина проблема и самог алгоритма, које је научено на ранијем пуштању алгоритама на сличним проблемима (нпр. из базе експеримената (Blockeel, 2006)). У простору компонентних алгоритама, **мета-учење би могло да се користи у синергији са еволуционом претрагом**. Систем за мета-учење може, из предходних искустава рада компоненти, да предвиди које компоненте би се понашале добро, чиме би се сузио простор на такве компоненте. У том потпростору еволуциони алгоритам може интелигентно извршити претрагу на добрим алгоритмом, заправо пуштајући део алгоритама на проблему који се решава. На тај начин би се избегле грешке које предвиђање мета-учења прави, а уједно и знатно убрзала претрага коју еволуциони алгоритам прави.

Може се рећи да би у таквој интеграцији систем мета-учења одговарао априори расподели вероватноће алгоритма у компонентном простору, док би извршавање алгоритама које евлоцуиони алгоритам спроводи било по апостериори расподели, тј. тражио би се модус апостериори расподеле компонентних алгоритама.

Како би се систем мета-учења и еволуционог алгоритмом интегрисали, може се приказати и Сликом 37, где се види како се добијају правила које компоненте користити (мета-учење, мета-хеуристика) из резултата евалуације перформанси алгоритама, док креирање (дизајн) алгоритама зависи и он мета-правила, и од простора алгоритама дефинисаних постојећим компонентама у репозиторијуму. Овај дијаграм заправо представља проширење оквира за аутоматски дизајн и селекцију алгоритама (Smith-Miles, 2009), као и адаптацију на компонентне алгоритме.



Слика 37. Проширење оквира (Smith-Miles, 2009) и интеграција мета-учења и аутоматског генерисања алгоритама

Ова теза одговара на питање како је могуће аутоматизовати креирање алгоритама за изградњу стабла одлучивања, тако да се добије боље прилагођен алгоритам подацима и проблему који се решава, уз то омогућавајући и анализу добијеног решења. Ипак, овом се тезом отвара још више питања које је неопходно истражити у будућности, како би ови алгоритми били још кориснији ширем кругу људи, као помоћ у предвиђању, ради бољег доношења одлука.

7. Научни и стручни доприноси

У оквиру ове дисертације као кључни *научни доприноси* могу се издвојити:

- ***Формулација новог проблема претраживања простора компонентних алгоритама (тј. новог модела комбинаторне оптимизације),*** чијим решавањем се могу генерисати алгоритми за специфичну примену. Овим се отвара нови истраживачки проблем, вредан решавања и другим методама које ова дисертација не обрађује.
- ***Развој нове методе за претрагу простора компонентних алгоритама за стабла одлучивања***, дефинисањем реализације хиперхеуристике еволуционих алгоритама. Оквир ове метахеуристике је проширен специфичном репрезентацијом решења, генетским операторима, и надоградњама које унапређују ефикасност претраге.
- ***Емпиријска евалуација изграђеног приступа***, низом пажљиво осмишљених експеримената, чиме се добија потврда о ефективности и ефикасности метода, поређењем у односу на оптимална решења и у односу на бенчмарк алгоритме, на јавно доступним подацима за проверу. Емпиријска провера омогућила је и карактеризацију креираних алгоритама у виду значајности појединих компоненти за укупан учинак алгоритма.
- ***Иновативна примена*** изграђеног приступа на проблем предвиђања успеха студената, уз анализу решења и уочавање компоненти алгоритама за стабла одлучивања који играју кључну улогу у решавају тог проблема, што може имати користи и изван примене компонентних алгоритама.
- Допринос ове дисертације огледа се и у датој систематизацији прегледа литературе из области стабала одлучивања и алгоритама базираних на компонентама, као и критичком осврту на досадашња

истраживања у предметној области са посебним акцентом на релевантне сродне приступе.

- Коначно, својеврсни допринос ове дисертације је мултидисциплинарност остварена повезивањем области откривања законитости у подацима, метода оптимизације и софтверског инжењерства.

Такође, дисертација носи и одређен *стручни допринос*:

- ***Нова софтверска имплементација*** предложеног решења, уз интеграцију као модул за популарни софтвер отвореног кода за откривање законитости у подацима, *RapidMiner*, чиме се знатно олакшава коришћење методе и приближава постојећој стручној заједници. Такође је омогућен оквир за даљу надоградњу софтвера и проширење, посебно проузроковано имплементацијом нових компоненти за алгоритаме за стабла одлучивања.

Доприноси које ова дисертација доноси су и верификовани у научној заједници, публикацијом у релевантним и угледним међународним часописима и конференцијама. Преглед публикација са парцијалним доприносом (Поглавље 3 и 5) и водећим доприносом (Поглавље 4) дати су у наставку. Сви приказани радови су објављени у часописима и конференцијама релевантним за ужу научну област дисертације.

Поглавље 3.1:

- Suknović Milija, Delibašić Boris, **Jovanović Miloš**, Vukićević Milan, Bečejski-Vujaklija Dragana, Obradović Zoran. Reusable Components in Decision Tree Induction Algorithms. *Computational Statistics*, Vol 27, No 1, (2012), pp. 127-148, http://dx.doi.org/10.1007/s00180-011-0242-8, **IF:0.482** (ISSN: 0943-4062) (**M23**)
- **Jovanović Miloš**, Delibašić Boris, Vukićević Milan: A "white box" data mining platform for decision support in decision tree induction algorithm design, *Proceedings of the 23 EURO conference*, Bonn, Germany, 2009., pp. 137. (**M34**)
- Delibašić Boris, **Jovanović Miloš**, Vukićević Milan, Suknović Milija, Kirchner Kathrin, Ruhland Johannes, Obradović Zoran: A decision support system architecture for data mining based on reusable components (patterns),

Proceedings of the EWG-DSS London 2011 Workshop on Decision Support Systems, London, UK, 2011., pp. 35. (**M34**)

- **Jovanović Miloš**, Suknović Milija, Vukićević Milan, Delibašić Boris: A white box approach in modeling phase of the data mining process, *Zbornik radova sa konferencije SYMOPIS*, Ivanjica 2009., pp. 709-712. (**M63**)

Поглавље 3.2:

- **Jovanović Miloš**, Delibašić Boris, Vukićević Milan, Suknović Milija: An open-source platform for design and testing of data mining algorithms, *Zbornik radova sa konferencije SYMOPIS*, Tara 2010., pp. 769-772. (**M63**)
- Delibašić Boris, **Jovanović Miloš**, Vukićević Milan, Suknović Milija: WhiBo: An open-source data mining framework, *Platforma za razvoj algoritama za otkrivanje zakonitosti u podacima napisana u programskom jeziku Java*, besplatno dostupna na Internet adresi: www.whibo.fon.bg.ac.rs, 2009. (**M85**)

Поглавље 3.3:

- Delibašić Boris, **Jovanović Miloš**, Vukićević Milan, Suknović Milija, Obradović Zoran. Component-based decision trees for classification. *Intelligent Data Analysis*, Vol 15, No 5, (2011), pp. 671-693, http://dx.doi.org/10.3233/IDA-2011-0489, **IF:0.448** (ISSN: 1088-467X) (**M23**)
- Vukićević Milan, **Jovanović Miloš**, Delibašić Boris, Išljamović Sonja, Suknović Milija. Reusable component-based architecture for decision tree algorithm design. *International Journal on Artificial Intelligence Tools*, Vol 21, No 5, (2012), http://dx.doi.org/10.1142/S0218213012500224, **IF:0.25** (ISSN: 0218-2130) (**M23**)
- Delibašić Boris, Vukićević Milan, **Jovanović Miloš**, Suknović Milija. White-Box or Black-Box Decision Tree Algorithms: Which to Use in Education?. *IEEE Transactions on Education*, Vol 56, No 3, (2011), pp. 287-291, http://dx.doi.org/10.1109/TE.2012.2217342, **IF:0.95** (ISSN: 0018-9359) (**M22**)

Поглавље 4:

- **Jovanović Miloš**, Delibašić Boris, Vukićević Milan, Suknović Milija, Martić Milan. Evolutionary approach for automated component-based decision tree algorithm design. *Intelligent Data Analysis*, Vol 18, (2014), pp. 63-77, http://dx.doi.org/10.3233/IDA-130628, **IF:0.5** (ISSN: 1088-467X) (**M23**)
- **Jovanović Miloš**, Delibašić Boris, Vukićević Milan, Suknović Milija. Optimizing performance of decision tree component-based algorithms using evolutionary algorithm in RapidMiner, *Proceedings of the 2nd RapidMiner Community Meeting and Conference*, Dublin, Ireland, 2011., pp. 135-149. (**M33**)

- **Jovanović Miloš**, Vukićević Milan, Išljamović Sonja, Suknović Milija: Automatic evolutionary design of decision tree algorithm for prediction of university student success, *Proceedings of the Stochastic Modeling Techniques and Data Analysis International Conference* (SMTDA 2012), Chania, Crete, Greece, 2012., pp. 48-49. (**M34**)

Поглавље 5:

- Vukićević Milan, Kirchner Kathrin, Delibašić Boris, **Jovanović Miloš**, Ruhland Johannes, Suknović Milija. Finding best algorithmic components for clustering microarray data. Knowledge and Information Systems, Vol 35 , No 1, (2012), pp. 111-130, http://dx.doi.org/10.1007/s10115-012-0542-5, **IF:2.49** (ISSN: 0219-1377) (**M21**)
- Delibašić Boris, Vukićević Milan, **Jovanović Miloš**, Kirchner Kathrin, Ruhland Johannes, Suknović Milija. An architecture for component-based design of representative-based clustering algorithms. *Data & Knowledge Engineering*, Vol 75, (2012), pp. 78-98, http://dx.doi.org/10.1016/j.datak.2012.03.005, **IF:1.519** (ISSN: 0169-023X) (**M22**)
- Delibašić Boris, Kirchner Kathrin, Ruhland Johannes, **Jovanović Miloš**, Vukićević Milan. Reusable components for partitioning clustering algorithms. *Artificial Intelligence Review*, Vol 32, No 1, (2009), pp. 59-75, http://dx.doi.org/10.1007/s10462-009-9133-6, **IF:0.057** (ISSN: 0269-2821) (**M23**)
- **Jovanović Miloš**, Vukićević Milan, Milovanović Miloš, Minović Miroslav. Using data mining on student behavior and cognitive style data for improving e-learning systems: a case study. *International Journal of Computational Intelligence Systems*, Vol 5, No 3, (2012), pp. 597-610, http://dx.doi.org/10.1080/18756891.2012.696923, **IF:0.451** (ISSN: 1875-6891) (**M22**)

Литература

Abonyi J (2003). Data-driven generation of compact, accurate, and linguistically sound fuzzy classifiers based on a decision-tree initialization, International Journal of Approximate Reasoning 32(1), pp. 1–21.

Almuallim H, Kaneda S, Akiba Y (2001). Development and applications of decision trees. *Proceedings in the Expert Systems - The Technology of Knowledge Management and Decision Making for the 21st Century Six-Volume Set*, Academic Press, 2001., pp. 53-77.

Alpaydin E (1999). Combined 5 × 2 cv F test for comparing supervised classification learning algorithms, *Neural Computation*, 11, pp.1885–1892.

Asuncion A, Newman D.J. (2007). *UCI Machine Learning Repository*. University of California, School of Information and Computer Science. [www.ics.uci.edu/~mlearn/MLRepository.html].

Barros R.C, Basgalupp M.P, de Carvalho A.C, Freitas A.A (2012). A hyper-heuristic evolutionary algorithm for automatically designing decision-tree algorithms. in Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference, GECCO '12, pp. 1237–1244

Basgalupp M.P, Barros R.C, de Carvalho A.C, Freitas A.A, Ruiz D.D (2009). LEGAL-tree: A lexicographic multi-objective genetic algorithm for decision tree induction, in: Proceedings of the 2009 ACM Symposium on Applied Computing, Hawaii, USA, pp. 1085–1090.

Blockeel H (2006), Experiment databases: A novel methodology for experimental research, in: *KDID 2005, LNCS*, 3933, Springer, Heidelberg, 2006, pp. 72–85.

Breiman L, Friedman J, Stone C.J, Olshen R.A (1984). *Classification and regression trees*, CRC Press.

Burke E.K, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003). Hyper-heuristics: an emerging direction in modern search technology. in Handbook of Metaheuristics, pp. 457–474.

Carvalho D, Freitas A (2004). A hybrid decision tree/genetic algorithm method for data mining, Information Sciences 163, pp. 13–35.

Cha S, Tappert C.C (2009). A genetic algorithm for constructing compact binary decision trees, J Pattern Recognition Research (JPRR) 4(1), pp. 1–13.

Čupić M, Suknović M (2008), *Odlučivanje*, FON, Beograd

Delibašić B, Suknović M, Jovanović M (2009a). *Algoritmi mašinskog učenja za otkrivanje zakonitosti u podacima*. FON, Beograd.

Delibašić B, Jovanović M, Vukićević M, Suknović M (2009b). *WhiBo: An open-source data mining framework*. Platforma za razvoj algoritama za otkrivanje zakonitosti u podacima napisana u programskom jeziku Java, besplatno dostupna na Internet adresi: www.whibo.fon.bg.ac.rs

Delibašić B, Kirchner K, Ruhland J, Jovanović M, Vukićević M (2009c). Reusable components for partitioning clustering algorithms, *Artificial Intelligence Review*, Vol 32, No 1, pp. 59-75.

Delibašić B, Jovanović M, Vukićević M, Suknović M, Obradović Z (2011). Component-based decision trees for classification, *Intelligent Data Analysis*, Vol 15, No 5, pp. 671-693.

Delibašić B, Vukićević M, Jovanović M, Suknović M (2012a). White-Box or Black-Box Decision Tree Algorithms: Which to Use in Education?, *IEEE Transactions on Education*, Vol 56, No 3, pp. 287-291.

Delibašić B, Vukićević M, Jovanović M, Kirchner K, Ruhland J, Suknović, M (2012b). An architecture for component-based design of representative-based clustering algorithms. *Data & Knowledge Engineering*, 75, p78-98

Delibašić B, Vukićević M, Jovanović M, Suknović M (2013). White-box decision tree algorithms: A pilot study on perceived usefulness, perceived ease of use, and perceived understanding. *The International Journal of Engineering Education* Vol 29, No 3, pp. 674–687.

Dietterich T (1998). Approximate statistical tests for comparing supervised classification learning algorithms, *Neural Computation* 10, 1895–1923.

Fowler M (2010). Domain-Specific Languages, Addison-Wesley.

Frank E (2000). *Pruning Decision Trees and Lists*. PhD Thesis, University of Waikato.

Gnanadesikan R (1977). *Methods for statistical data analysis of multivariate observations*. Wiley, New York

Hansen P, Mladenovic N, Perez J.A.M. (2010). Variable neighbourhood search: methods and applications. *Annals of Operations Research* 175, pp 367–407.

Heath D, Kasif S, Salzberg S (1993). Induction of Oblique Decision Trees. In *Proc. of the 13th Intl. Joint Conf. on Artificial Intelligence*, p1002-1007, Chambery, France.

Holland JH (1975). *Adaptation in natural and artificial systems*, MIT Press, Cambridge, MA, USA.

Jovanović M, Delibašić B, Vukićević M (2009). A "white box" data mining platform for decision support in decision tree induction algorithm design, *Proceedings of the 23 EURO conference*, Bonn, Germany, pp. 137.

Jovanović M, Vukićević M, Išljamović S, Suknović M (2012a). Automatic evolutionary design of decision tree algorithm for prediction of university student success. *Stochastic Modeling Techniques and Data Analysis* (SMTDA) 2012, Chania, Crete, Greece

Jovanović M, Vukićević M, Milovanović M, Minović M (2012b), Using data mining on student behavior and cognitive style data for improving e-learning systems: a case study, *International Journal of Computational Intelligence Systems* 5(3), 597-610

Jovanović M, Delibašić B, Vukićević M, Suknović M, Martić M (2014). Evolutionary approach for automated component-based decision tree algorithm design, *Intelligent Data Analysis* 18(1), 25-42

Kass G.V (1980). An exploratory technique for investigating large quantities of categorical data, *Applied Statistics*, Vol 29, pp. 119-127.

Kohavi R (1996). Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid. In: *Second International Conference on Knoledge Discovery and Data Mining* 1996, p202-207.

Koza J (1991). Concept formation and decision tree induction using the genetic programming paradigm, Parallel Problem Solving from Nature 1, pp. 1–6.

Kretowski M, Czajkowski M (2010). An Evolutionary Algorithm for Global Induction of Regression Trees. Artifical Intelligence and Soft Computing, 6114, pp. 157-164.

Kumar V, Chadha A (2011). An Empirical Study of the Applications of Data Mining Techniques in Higher Education. Int Journal of Advanced Computer Science and Applications, 2(3), pp 80-84.

Landwehr N, Hall M, Frank E (2005). Logistic Model Trees. *Machine Learning*, 95, p161-205.

Lee S.M, Moeller G.L, Digman L.A (1981). Decision Tree Applications. *Network Analysis for Management Decisions, International Series in Management Science/Operations Research*, pp. 174-178, Springer Netherlands.

Lim T-S, Loh W-Y, Shih Y-S (2000). A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms, *Machine Learning*, 40(3), pp 203-228.

Lin S-W, Lee Z-J, Chen S-C, Tseng T-Y (2008). Parameter determination of support vector machine and feature selection using simulated annealing approach, Applied Soft Computing 8(4), pp. 1505–1512.

Loh W.Y, Shih Y.S (1997). Split selection methods for classification trees, *Statistica Sinica*, Vol 7, pp. 815-840.

Mantaras R.L (1991). A distance-based attribute selection measure for decision tree induction, *Machine Learning*, Vol 6, pp. 81-92.

Marinakis Y, Marinaki M, Doumpos M, Zopounidis C (2009). Ant colony and particle swarm optimization for financial classification problems, Expert Systems with Applications 36(7), pp. 10604–10611.

Meffert K, *JGAP – Java Genetic Algorithms and Genetic Programming Package*, URL: http://jgap.sf.net.

Michalewicz Z (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.

Mierswa I, Wurst M, Klinkenberg R, Scholz M, Euler T (2006). YALE: Rapid prototyping for complex data mining tasks, *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM Press, 2006, pp. 935–940.

Murthy S.K (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data mining and knowledge discovery*, Vol 2, No 4, pp. 345-389.

Murthy S.K, Kasif S, Salzberg S (1994). A system for induction of oblique decision trees, *Journal of Artificial Intelligence Research*, 2, p1-33.

Olafsson S, Li X, Wu S (2008). Operations research and data mining. European Journal of Operational Research, 187(3), pp. 1429-1448.

Pang-Ning T, Steinbach M, Kumar V (2006). *Introduction to data mining*. Addison-Wesley.

Papagelis A, Kalles D (2000). GA Tree: Genetically evolved decision trees, in: Proceedings of 12th IEEE International Conference on Tools with Artificial Intelligence, Vancouver, Canada, pp. 203–206.

Pappa G, Freitas A (2010). Automating the Design of Data Mining Algorithms An Evolutionary Computation Approach, Natural Computing Series, Springer.

Parpinelli R.S, Lopes H.S, Freitas A.A (2002). Data mining with an ant colony optimization algorithm, IEEE Transactions on Evolutionary Computation 6(4), pp. 321–332.

Qilong Z, Ganlin S, Xiusheng D, Zining Z (2009). Parameters optimization of support vector machine based on simulated annealing and genetic algorithm, in: 2009 IEEE International Conference on Robotics and Biomimetics, Guilin, China, pp. 1302–1306.

Quinlan J.R (1986). Induction of decision trees, *Machine Learning*, Vol 1, pp. 81-106.

Quinlan J.R (1993). *C4.5 programs for machine learning*, Morgan Kaufmann, 1993.

Rice J.R (1976). The algorithm selection problem. Advances in Computers 15, pp 65–118.

Rokach L (2015). Data mining with decision trees: theory and applications, 2nd Ed, World Scientific.

Romero C, Ventura S (2010). Educational Data Mining: A Review of the State of the Art. *Systems, Man, and Cybernetics*, 40(6), pp 601-618

Rozsypal A, Kubat M (2003). Selecting representative examples and attributes by a genetic algorithm, Intelligent Data Analysis 7, pp. 291–304.

Schimpf J, Wallace M (2002). Finding the right hybrid algorithm - a combinatorial meta-problem, Annals of Mathematics and Artificial Intelligence, 34 (4), pp. 55–67.

Shannon C.E (1948). A Mathematical Theory of Communication, *Bell System Technical Journal*, 27, pp. 379–423 & 623–656.

Smith-Miles K.A (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection, *ACM Computing Surveys*, 41(1), pp 1–25.

Sonnenburg S, Braun M.L, Ong C.S, Bengio S, Bottou L, Holmes G, LeCun Y, Mueller K.R, Pereira F, Rasmussen C.E, Raetsch G, Schoelkopf B, Smola A (2007). The need for open source software in machine learning, *Journal of Machine Learning Research* 8, 2443–2466.

Sousa T (2004). Particle swarm based data mining algorithms for classification tasks, Parallel Computing 30, pp. 767–783.

Suknović M, Delibašić B, Jovanović M, Vukićević M, Bečejski-Vujaklija D, Obradović Z (2012). Reusable Components in Decision Tree Induction Algorithms, *Computational Statistics*, Vol 27, No 1, pp. 127-148.

Tan P-N, Steinbach M, Kumar V (2005), *Introduction to Data Mining*, Pearson pub.

Todorovski Lj, Blockeel H, Dzeroski S (2002) Ranking with Predictive Clustering Trees, *European Conference on Machine Learning 2002*, Helsinki, Finland

Tsiptis K, Chorianopoulos A (2009). *Data Mining Techniques in CRM: Inside customer segmentation*. John Wiley & Sons.

Turney P.D (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm, Artificial Intelligence Research 2, pp. 369–409.

Urbanowicz R.J, Moore J.H (2009). Learning Classifier Systems: A Complete Introduction, Review, and Roadmap, Journal of Artificial Evolution and Applications,1,1-25.

Vella A, Corne D, Murphy C (2009). Hyper-heuristic decision tree induction, 2009 World Congress on Nature & Biologically Inspired Computing, Coimbatore, India, pp. 409–414.

Vukićević M, Jovanović M, Delibašić B, Suknović M (2010). WhiBo - RapidMiner plug-in for component based data mining algorithm design, *Proceedings of the 1st RapidMiner Community Meeting and Conference*, Dortmund, Germany,pp. 30-35.

Vukićević M, Jovanović M, Delibašić B, Išljamović S, Suknović M (2012a). Reusable component-based architecture for decision tree algorithm design, *International Journal on Artificial Intelligence Tools*, Vol 21, No 5.

Wolpert D.H (1996). The lack of a priori distinctions between learning algorithms, *Neural computation*, Vol 8, pp. 1341–1390.

Wu X, Kumar V, Quinlan J.R, Ghosh J, Yang Q, Motoda H, McLachlan G.J, Ng A, Liu B, Yu P.S, Zhou Z.H, Steinbach M, Hand D.J, Steinberg D (2008). Top 10 algorithms in data mining, *Knowledge information systems*, Vol 14, pp. 1-37.

Zhao Y, Zhang Y (2007). Comparison of decision tree methods for finding active objects*, Advances in Space Research*, 41(12), pp 1955-1959.

Биографија аутора

Милош Јовановић је рођен 1982. године у Струги, Република Македонија. Од тада живи у Београду, где је завршио основно и средње образовање.

Факултет Организационих Наука, Универзитета у Београду, уписује 2001. године. Дипломирао је на одсеку за Информационе системе 2006. године, са темом: „Репрезентација знања као мост између Дејта-мајнинга и Експертних система“.

Од децембра 2007. године је запослен на Факултету организационих наука, на Катедри за Организацију пословних система, Центар а пословно одлучивање, где ради до данас. Предмети на којима је радио на основним академским студијама су: Теорија одлучивања, Пословна интелигенција, Системи за подршку одлучивању, Машинско учење; а на мастер академским студијама: Системи пословне интелигенције, Складишта података, Откривање законитости у подацима, и Развој алгоритама машинског учења. Такође је у току рада на Факултету организационих наука био члан више комисија за одбрану завршних радова.

Докторске студије уписује 2008. године на Факултету организационих наука, Универзитета у Београду, на одсеку за Операциона истраживања, где је положио све предмете са оценом 10.

У 2012. години постаје лиценцирани експерт за софтвер *RapidMiner*, један од најкоришћенијих софтвера отвореног кода за откривање законитости у подацима и анализу података.

У марту 2012. године је био ангажован као гостујући предавач на предмету „MW 31.1 Business intelligence“, на *Friedrich-Schiller* Универзитету у Јени (Немачка), у оквиру катедре за Информационе системе.

Од октобра 2014. до јула 2015. године гостује као истраживач на Темпл Универзитету, у Филаделфији, Пенсилванија, САД, у Центру за анализу података и биоинформатику, под руководством др Зорана Обрадовића.

У току студирања објавио је већи број радова на домаћим и страним конференцијама, као и у међународним часописима, од чега и 9 радова на часописима са импакт фактором из релевантне уже научне области. Такође је и коаутор пријављеног техничког решења, као једне стручне књиге „Алгоритми машинског учења за откривање законитости у подацима". Учествовао је и на неколико домаћих и међународних пројеката.

Прилог 1.

Изјава о ауторству

Потписани-а Милош Јованович

број индекса 38/2008

Изјављујем

да је докторска дисертација под насловом

Аутоматско генерисање алгоритама стабала одлучивања за класификацију

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 15.01.2016.

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора Милош Јовановић

Број индекса 38/2008

Студијски програм Операциона истраживања

Наслов рада Аутоматско генерисање алгоритама стабала одлучивања за класификацију

Ментор др Милија Суковић

Потписани/а Милош Јовановић

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, 15.01.2016.

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

Аутоматско генерисање алгоритама стабала одлучивања за класификацију

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

(1) Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 15.01.2016.