Oblikovanje prioritetnih pravila za problem raspoređivanja s ograničenim sredstvima.

Đumić, Mateja

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Mateja Đumić

OBLIKOVANJE PRIORITETNIH PRAVILA ZA PROBLEM RASPOREĐIVANJA S OGRANIČENIM SREDSTVIMA

DOKTORSKI RAD

Zagreb, 2020.

Sveučilište u Zagrebu
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Mateja Đumić

OBLIKOVANJE PRIORITETNIH PRAVILA ZA PROBLEM RASPOREĐIVANJA S OGRANIČENIM SREDSTVIMA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Domagoj Jakobović

Zagreb, 2020.

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Mateja Đumić

DESIGN OF PRIORITY RULES FOR RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

DOCTORAL THESIS

Supervisor: Professor Domagoj Jakobović, PhD

Zagreb, 2020.

Doktorski rad izrađen je na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva, na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave.

Mentor: prof. dr. sc. Domagoj Jakobović

Doktorski rad ima: 154 stranice

Doktorski rad br.: _____

O mentoru

Domagoj Jakobović rođen je u Našicama 1973. godine. Diplomirao je, magistrirao i doktorirao u polju računarstva na Fakultetu elektrotehnike i računarstva (FER) na Sveučilištu u Zagrebu, 1996., 2001. odnosno 2005. godine.

Od travnja 1997. godine zaposlen je na Zavodu za elektroniku, mikroelektroniku, računalne i inteligentne sustave FER-a te je u siječnju 2018. godine izabran u zvanje redovitog profesora. Kao suradnik i voditelj sudjeluje na više znanstvenih projekata, među kojima je i trenutno aktivan projekt "HyDDRa" koji financira Hrvatska zaklada za znanost, a čiji je voditelj. Objavio je oko 90 radova u časopisima i zbornicima konferencija u području primjene stohastičke optimizacije i strojnog učenja u problemima raspoređivanja i kriptografiji te razvoja paralelnih evolucijskih algoritama.

Prof. dr. sc. Domagoj Jakobović član je Hrvatske akademije tehničkih znanosti i stručnih udruga IEEE i ACM. Osim toga, član je uredništva znanstvenog časopisa te sudjeluje kao recenzent u više od 15 inozemnih časopisa te kao član programskog odbora sudjeluje u organizaciji više međunarodnih konferencija.

About the Supervisor

Domagoj Jakobović was born in Našice in 1973. He received B.Sc., M.Sc. and Ph.D. degrees in computer science from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1996, 2001 and 2005, respectively.

From April 1997 he is working at the Department of electronics, microelectronics, computer and intelligent systems at FER. In January 2018 he was promoted to Full Professor. He participated in several scientific projects and currently is lead researcher on project "HyDDRa" funded by the Croatian Science Foundation. He published about 90 papers in journals and conference proceedings in the area of application of stochastic optimization and machine learning in scheduling and cryptography, as well as development of parallel evolutionary algorithms.

Prof. Jakobović is a member of Croatian Academy of Engineering, IEEE and ACM. He is a member of a journal editorial board and he serves as a technical reviewer for more than 15 international journals. As a member of the program committee, he participates in the organization of several international conferences.

Zahvala

Na početku, želim zahvaliti svom mentoru prof. dr. sc. Domagoju Jakoboviću koji me je pratio i usmjeravao kroz cijelo ovo vrijeme. Veliko hvala na svemu, prvenstveno na svom strpljenju, trudu i vremenu koje je uložio tijekom nastajanja ove disertacije.

Hvala povjerenstvima koja su moje istraživanje i rad pratila kroz kvalifikacijski ispit, javni razgovor, ocjenu i obranu rada te svojim komentarima uvelike doprinijeli kvaliteti samog rada.

Zahvaljujem se i svom Odjelu za matematiku, tadašnjoj i sadašnjoj upravi, zajedno s mentorom izv. prof. dr. sc. Domagojem Matijevićem, koji su mi iskazali povjerenje dopuštanjem upisivanja ovog doktorskog studija. Hvala im na svoj podršci i razumijevanju koje su mi pokazivali prilikom izvršavanja obveza na doktorskom studiju.

Veliko hvala i svim kolegicama i kolegama, kako s Fakulteta elektrotehnike i računarstva u Zagrebu, tako i s Odjela za matematiku u Osijeku. Hvala vam na svim razgovorima, odgovorenim mailovima, komentarima i odrađenim administrativnim poslovima. No, prije svega hvala vam što sam uvijek znala da vam se mogu obratiti i računati na vas.

Hvala i svim mojim prijateljima koji su mi kroz studiranje bili velika podrška i poticaj i koji su znali odvratiti misli od obveza i tako doprinijeti da se odmornija vratim izvršavanju istih.

Posebno hvala ide kolegici, ali prije svega prijateljici, Rebeki koja je ovo cijelo vrijeme u svemu bila zajedno sa mnom. Bilo je puno lakše sve ovo prolaziti zajednički.

Ponajviše, hvala mojoj obitelji. U prvom redu roditeljima koji su me uvijek ohrabrivali da u životu idem za onim što volim i što mislim da je ispravno, bez obzira na to koliko je to ponekad teško. Hvala im na svemu što su učinili da sve ovo mogu i ostvariti. Bez njihove bezgranične podrške i ljubavi sve ovo bi bilo teško ostvarivo i dohvatljivo. Hvala i mom bratu i Vedrani, te nećacima Emanueli, Petri i Rafaelu koji su me kroz ovo vrijeme učili da ima i drugih bitnih stvari i koji su unosili radost u svakodnevni život.

Na kraju, zahvala i Onome koji čini da sve bude dobro i onakvo kakvo treba biti što mi je davalo sigurnost i snagu kroz ovo vrijeme i život općenito.

Sažetak

Problem raspoređivanja s ograničenim sredstvima je problem u kojem je potrebno pronaći raspored aktivnosti koji će zadovoljiti uvjete prednosti i uvjete na sredstva i pri tome optimizirati jedan ili više kriterija. Ovaj problem pripada klasi NP-teških problema zbog čega pronalazak optimalnog rješenja za većinu instanci nije moguć. U skladu s tim, razvijaju se brojne heurističke metode koje su uglavnom ograničene na primjenu u statičkim okruženjima, dok su prioritetna pravila jedna od rijetkih metoda koja se može primijeniti i u dinamičkim okruženjima. Razvoj prioritetnih pravila je zahtjevan posao zbog čega se sve više koriste postupci za automatizirani razvoj pravila raspoređivanja.

Ova disertacija usmjerena je prema oblikovanju prioritetnih pravila za problem raspoređivanja s ograničenim sredstvima upotrebom genetičkog programiranja podijeljena je u tri glavna dijela. Prvi dio je usmjeren prema samom razvoju prioritetnih pravila, u drugom dijelu primjenom ansambala dodatno se povećava kvaliteta rezultata ostvarenih razvijenim prioritetnim pravilima, dok treći dio donosi prilagodbu razvijenih pravila za korištenje u statičkim uvjetima. Provedena ispitivanja pokazuju da razvijena pravila ostvaruju značajno bolje rezultate od onih ostvarenih postojećim prioritetnim pravilima te da njihovom dodatnom prilagodbom, u ovisnosti o okruženju u kojem se primjenjuju, mogu biti ostvarena značajna poboljšanja u rezultatima.

Ključne riječi: problem raspoređivanja s ograničenim sredstvima, genetičko programiranje, prioritetna pravila, hiperheuristika, strojno učenje, ansambli

Extended abstract

Design of priority rules for resource constrained project scheduling problem

Introduction

Scheduling can be defined as a process of allocating resources to activities in time period. It has an important role in manufacturing, information systems, product distribution, etc. The main objective of scheduling is to find an activity schedule that meets all given constraints while optimizing one or more criteria. Usually, in the process of scheduling, there exist some constraints like limited budget, number of available machines or employees, etc. Scheduling problems with constraints are common in practice and some of them are: resource-constrained project scheduling problem, nurse scheduling problem, the vehicle routing problem and school timetabling problems. All of these problems are NP-hard problems, which means that finding the exact solution in polynomial time is not possible.

The main focus of this thesis is on the resource constrained project scheduling problem (RCPSP) in which two types of constraints are present - precedence constraints and resource constraints. Precedence constraints define possible order of activities defining predecessors and successors of an activity, while resource constraints provide information on the availability of resources.

The solving methods for RCPSP can be divided into two main groups - exact and heuristic algorithms. Exact algorithms search whole solution space and guarantee the optimality of solution but because of the size of solution space, they are impractical and inapplicable for most problem instances. Unlike exact algorithms, heuristics do not search entire solution space and do not guarantee that found solution is optimal but they are fast and can find solutions that are good enough for practice. Some of heuristics that can be used in solving RCPSP are genetic algorithm, network decomposition and ant colony optimization. Recently there are many attempts to solve RCPSP with a combination of solving methods like memetic algorithms or combining evolutionary algorithms with some local search methods.

The main problem of the above-mentioned heuristics is that most of them can not be used in a dynamic environment in which changes in the system are common and informations are incomplete or uncertain. In literature, priority rules are mostly used for dynamic environments. Priority rules consist of two main parts - a function that determines the priority of available activities and a schedule generation scheme which, based on assigned priority, puts activities in the schedule. The process of creating a schedule, when using priority rules, starts with an empty schedule in which activities are added one by one until all activities are scheduled. The advantages of priority rules are their speed and the ability to respond to changes that can happen

in the system.

The main disadvantage of this approach is the quality of solutions which are not as good as the ones found with approaches applicable in a static environment. Also, different priority rules need to be created for different optimization criteria, which is a hard process and implies the existence of a specialist who knows the problem sufficiently to create a new rule. This is why, recently, researchers start to use hyperheuristics. In literature, two types of hyperheuristics can be found - the one that chooses which heuristic, from the set of different ones, to use for solving some problems and the second one which creates a new one by using the given elements. In this thesis the second one will be used. More precisely, hyperheuristic will be used for creating new priority rules.

Priority rules are used in many scheduling problems which result in numerous attempts to automate the creation process of priority rules by different machine learning methods. Genetic programming (GP) is one of those methods, and results from literature show that it can evolve priority rules which achieve results that are human-competitive. This approach is commonly used in a different environment like one-machine scheduling problem, job shop scheduling problem and unrelated machines environment.

Despite the fact that GP is used in scheduling more than 15 years and achieves good results, it was not used in solving RCPSP till recently. Considering the complexity of the problem, its importance in real life and only a few papers on this topic it can be concluded that there are still a lot of open questions waiting to be answered.

Motivation

The motivation for this research comes from facts that priority rules for scheduling evolved with GP achieve good results in lot of different scheduling environments and that this hyperheuristic approach is used only in few papers to evolve priority rules that will be used for solving RCPSP. This research in some way is still the beginning of exploring this topic and it is important to make some basic assumptions that can be used as a starting point in further research.

The first thing to do is to determine which terminals and functions need to be used to be able to evolve a good priority rule. In literature, there are many different approaches for choosing terminals for evolving rules in various scheduling environments and it is important to find the appropriate one for the problem in this thesis. Also, there is no research in the literature that states which scheduling generation scheme to use when evolving rules with GP or how instances used for evolving effect on evolved rule.

In most cases, priority rules do not achieve results of the same quality as one achieved with methods that can only be used in static environments. That is why the second part of the research will focus on finding ways how to further improve achieved results by evolved priority rules. Generally, different priority rules achieved results of different quality for different pro-

blem instances which leads to the idea not to use only one rule but more of them. For combining priority rules ensemble learning will be used. Ensemble learning is a machine learning technique that has already been used in different machine environments to combine priority rules and resulted in the improvement of priority rule quality.

Regardless of the expectation that using ensembles will lead to improvement of results, still they will not be able to achieve results that are good enough for usage in a static environment. Because of that it is important to find ways of using available information in a static environment to improve the quality of the solution even further. With improved quality, the speed of priority rules can lead to using priority rules in a static environment more that this is being done at the moment.

Thesis overview

The thesis is divided into 7 chapters. In Chapter 1 introduction and a brief motivation is given, chapters 2 and 3 contain an introduction in problems and methods used in the research conducted in this thesis. The main contributions of this thesis are given in the next three chapters, while in the last one a short conclusion with possible research directions is given.

Chapter 1 is the introductory chapter of the thesis. In this chapter the resource-constrained project scheduling problem (RCPSP) is introduced through its importance in real-life and its complexity. After that, short motivation for research done in the thesis is given. Additionally, the main contributions of the thesis were highlighted and an overview of the thesis sections was given at the end.

Chapter 2 gives a formal description of RCPSP and its important properties. In this chapter factors which affect the hardness of the problem are outlined and most used optimization criteria are listed. The various methods which are most often used for solving RCPSP are mentioned together with their strengths and weaknesses. At the end of the chapter, two types of schedule generation schemes that are used in combination with priority function or for creating an initial solution are described together with their differences.

Chapter 3 gives a detailed description of the genetic programming (GP) algorithm. In this chapter all main parts of the algorithm such as solution representation, the process of initialization in population, terminal and function sets, genetic operators and selection are described. Also, the fitness function is introduced which allows one to compare individuals with each other. The process of selecting individuals that participate in the crossover or are becoming part of the next generation is described. The chapter also mentions various stopping criteria and highlights that finding a good one is important to achieve a good solution without losing generalization ability.

Chapter 4 provides a literature overview of papers in which GP is used to evolve rules for scheduling problems. In the given overview, the main focus was on papers in which priority

rules were evolved for RCPSP. After that, two optimization criteria that will be used in experiments were given and explained as well as terminal and function sets used for evolving priority rules. Also, in this chapter, along with a parallel and serial schedule generation scheme used in combination with priority rules, a new custom schedule scheme was suggested. The schedule generation scheme proposed in this chapter allows idle time which is the only difference from the parallel schedule generation scheme used in other literature. At the beginning of the experiments initial parameters were determined. After that, terminal and function sets were analyzed in more detail as well as the need for normalization of terminals. Besides that, different data sets were examined and their importance for the quality of the solution pointed put. At the end of chapter examples of evolved priority rules and comparison with the human-made ones were given.

In chapter 5 ensembles of priority rules were introduced to improve results achieved with evolved priority rules. The main idea of this approach comes from the fact that different priority rules make different mistakes, so maybe using a combination of them can lead to improved results. In experiments two different ways of combining - the sum method and the vote method and four different ensemble learning approaches were used. The simplest approach used in this thesis is simple combination method in which ensemble members are chosen from previously evolved rules by using 5 different ways. The next approach used is called BagGP which represents the version of the bagging method in which GP is used. In the BagGP approach new priority rules were evolved using replication of the original learning set. The third approach is BoostGP which stands for the AdaBoost method used in combination with GP. In this approach, ensemble members are obtained by learning modified data whose modification takes into account the quality of the solution achieved on that data. The last approach mentioned in this chapter is cooperative coevolution in which all ensemble members are evolving at the same time and only interaction between them is when the ensemble is evaluated on problem instances. After the ensemble was created by one of these learning approaches, an ensemble subset search was done to further improve the results. For each ensemble learning approach tests were performed to determine the optimal parameters. The tests, among other things, determined the optimal ensemble size for each approach. Finally, a comparison of all ensemble learning approaches described in this chapter is given, as well as a comparison of them with the individual rules.

Chapter 6 describes two additional ways to adjust the priority rules for quality improvement that can be used in a static environment. The first of these are iterative priority rules that through iterations further improve the quality of the created schedule. New terminals in the terminal set are introduced for this approach to allow the transfer of information from iteration to iteration and new priority rules need to be evolved for this approach. The second approach which is used consists of an adapted schedule generation scheme used for solving new instances of the problem and for this approach it is possible to use the rules evolved in the previous chapters.

This approach is known as rollout and consists of considering several possible schedules and determining the best among them. At the end of the chapter comparison of these approaches with the individual rules is made. Also, results achieved with these two approaches were compared with ones achieved with GA as a representative of heuristic methods that can be used in a static environment.

Chapter 7, which is also the last one, gives the conclusion of research conducted in the thesis through three main scientific contributions. Further, this chapter highlights possible directions for future research based on topics that remained open after research done in this thesis or that were not covered by this thesis at all.

Conclusion

The main goal of the thesis is to find a procedure for developing priority rules using genetic programming that are applicable to RCPSP and which can adjust to different conditions and optimization criteria. The following three original scientific contributions were made in the thesis:

- Terminal and function set for development of priority rules and procedure for developing priority rules for resource constrained project scheduling problem adjustable to different optimization criteria
- Methods for development of priority rules based on ensemble learning
- Procedure for developing priority rules applicable in the static scheduling environment

In this thesis, the procedure for developing priority rules for resource constrained project scheduling problem was described and explained in detail. Also, terminal and function set that achieve statistically better results than those achieved with human-made rules were determined. The results achieved using the evolved rules are further enhanced by combining the priority rules into ensembles. Procedures for developing priority rules applicable in the static scheduling environment which result in improved results while maintaining the speed as the main advantage of priority rules are proposed. Based on the presented results, it can be concluded that the use of GP for rule development is a good choice and by using it it is possible to develop rules for complicated optimization criteria. The tests done in the thesis open new topics for study while results achieved in the thesis show great potential for further research.

Keywords: resource constrained project schedulin problem, genetic programming, priority rules, hyperheuristic, machine learning, ensembles

Sadržaj

Poglavlje 1

Uvod

Raspoređivanje je postupak dodjeljivanja sredstava aktivnostima u određenom vremenskom periodu [1]. Raspoređivanje ima značajnu ulogu u proizvodnim sustavima, informacijskim okruženjima, distribuciji proizvoda i sl. Sredstva koja se dodjeljuju mogu biti strojevi u tvornicama, zaposlenici koji sudjeluju u proizvodnom procesu, jedinice procesora, vozila i slično, dok su aktivnosti, na primjer, procesi u proizvodnji, radne smjene i računalni programi. Cilj raspoređivanja je pronaći raspored aktivnosti koji zadovoljava sve dane uvjete, a pri tom optimizira jedan ili više kriterija (minimizacija trajanja projekta, minimizacija troškova, maksimizacija dobiti i dr.).

Prilikom pravljenja rasporeda najčešće se susrećemo s dodatnim ograničenjima kao što su novčani budžet predviđen za projekt, broj strojeva koje posjedujemo, broj zaposlenika, dopušteno radno vrijeme, kapacitet prijevoznog sredstva i mnoga druga. Problemi s ograničenjima su vrlo česti u praksi. Neki od njih su problem raspoređivanja s ograničenim sredstvima [2], problem raspoređivanja medicinskog osoblja [3, 4], problem usmjeravanja vozila [5, 6, 7], problem rasporeda nastave i ispita u akademskim zajednicama [8, 9], problem održavanja zrakoplova i zračnog prometa [10], problem raspoređivanja u podzemnim stanicama [11, 12] i drugi.

Zbog svoje kompleksnosti i specifičnosti gore navedene probleme teško je zajednički proučavati te svaki od njih privlači veliki broj istraživača koji neprestano razvijaju nove metode za rješavanje. Stalni razvoj novih metoda, osim u činjenici da su to problemi koje susrećemo često u praksi, leži i u tome da svi ovi problemi pripadaju klasi NP-teških problema, odnosno njihovo egzaktno rješenje nije moguće pronaći u polinomijalnom vremenu.

U ovom radu proučavat će se problem raspoređivanja s ograničenim sredstvima. Problem raspoređivanja s ograničenim sredstvima (engl. *resource constrained project scheduling problem*, skraćeno RCPSP) je problem u kojem susrećemo dvije vrste ograničenja: uvjete prednosti i uvjete na sredstva. Uvjeti prednosti određuju redoslijed izvršavanja aktivnosti, odnosno oni nam daju informaciju koje aktivnosti moraju biti završene kako bi se određena aktivnost mogla početi izvršavati, kao i koje aktivnosti su njezini sljedbenici. Informacija o dostupnosti pojedi-

nog sredstva, koje se koristi prilikom raspoređivanja, kao i informacija o tome koliko pojedinog sredstva je potrebno za izvršenje neke aktivnosti dana je uvjetima na sredstva.

Metode rješavanja ovog problema možemo podijeliti na egzaktne algoritme i heuristike [13]. Egzaktni algoritmi pretražuju cijeli prostor rješenja, pa kao takvi osiguravaju optimalnost rješenja. Zbog veličine prostora rješenja kod većih instanci problema ovi algoritmi su nepraktični i neupotrebljivi [14], zbog čega se razvijaju heuristike. Heuristike, za razliku od egzaktnih algoritama, pretražuju samo dio dopustivog prostora rješenja te ne jamče optimalnost rješenja, no brze su i ako su dobro dizajnirane pronalaze, za praksu, dovoljno dobra rješenja većine problema. Između ostalog, u heurističke algoritme ubrajamo: genetički algoritam [15, 16], analizu mreže [17], kolonije mrava [18], kao i brojne algoritme u kojima je spojeno više metoda [19] pri čemu se najčešće kombiniraju evolucijski algoritmi s nekom metodom lokalnog pretraživanja [20].

Problem prethodno navedenih heurističkih algoritama je što oni uglavnom nisu primjenjivi u dinamičkim okruženjima koje karakteriziraju česte promjene u sustavu i u kojima nisu uvijek poznate sve informacije. Zbog toga u takvim okruženjima najčešće primjenjujemo prioritetna pravila. Pregled najboljih pravila u literaturi pronalazimo u [21]. Prioritetna pravila zajedno sa shemom za generiranje rasporeda kroz iteracije grade raspored. Postupak započinje s praznim rasporedom u koji se dodaju aktivnosti na osnovu prioriteta koji je određen prioritetnim pravilom. Postupak se ponavlja dok sve aktivnosti ne budu raspoređene. U literaturi razlikujemo slijednu i usporednu shemu za generiranje rasporeda [22]. Jedna od prednosti prioritetnih pravila je mogućnost brzog reagiranja na promjene, kao i činjenica da se, ako dođe do promjene, raspored ne mora graditi ispočetka nego se izgradnja nastavlja tamo gdje se stalo prije nego se promjena dogodila.

Nedostatak prioritetnih pravila je što ona najčešće ne daju jednako kvalitetna rješenja kao heuristički algoritmi primjenjivi u statičkim okruženjima. Osim toga, potrebno je razviti posebna prioritetna pravila za svaki kriterij koji se želi optimizirati, što uglavnom zahtijeva postojanje stručnjaka koji na temelju svog znanja i iskustva može razviti pravilo. Zbog toga se javlja potreba za izdizanjem na višu razinu problema, odnosno korištenjem hiperheurističke metode. Općenito, razlikujemo dvije vrste hiperheuristika: hiperheuristike koje nekim postupkom iz skupa heuristika odabiru najbolju među njima za rješavanje problema i hiperheuristike koje pomoću danih elemenata razvijaju novu. U ovom slučaju koristi se hiperheuristička metoda koja razvija prioritetno pravilo čijim korištenjem se izgrađuje raspored. Razvoj pravila je u tom slučaju automatiziran što čini razvoj pravila za različite kriterije lakim.

Problem razvoja prioritetnih pravila nije svojstven samo za RCPSP, nego ga pronalazimo i kod ostalih problema raspoređivanja što je rezultiralo brojnim pokušajima automatizacije razvoja primjenjujući različite metode strojnog učenja [23, 24]. Genetičko programiranje (engl. *genetic programming*, skraćeno GP) se u literaturi pokazalo kao dobar odabir za hiperheuristiku

koja razvija pravila raspoređivanja [25, 26]. Takav pristup prvi put se pojavljuje u radu [27] za okruženje jednog stroja, a zatim i u drugim složenijim okruženjima: raspoređivanje na više strojeva [28], okruženje proizvoljne obrade [29, 30, 31] i okruženje nesrodnih strojeva [32]. Pravila raspoređivanja koja su razvijena GP-om u većini slučaja daju rezultate koji su bolji od rezultata dobivenih klasičnim ručno definiranim pravilima [33].

Unatoč činjenici da se GP u raspoređivanju primjenjuje više od 15 godina i pri tome ostvaruje dobre rezultate, njegova intenzivnija primjena na RCPSP započinje tek u zadnjih par godina. Ideja o primjeni GP-a na RCPSP dana je u radu iz 2008. godine [34] da bi se, tek 10 godina nakon, detaljnije počela proučavati u radovima [35, 36]. Nastavak proučavanja iz rada [36] pronalazi se u disertaciji [37]. Ovi radovi, kao i disertacija, pokazatelj su privlačnosti problema i dobrih rezultata koji se mogu postići korištenjem ovog pristupa. S obzirom na mali broj radova na ovu temu i kompleksnost samog problema, sigurno je da u ovom području postoji još puno prostora za doprinos i odgovore na brojna otvorena pitanja.

1.1 Istraživačka motivacija

Primjena GP-a na RCPSP s ciljem razvoja prioritetnih pravila je područje istraživanja koje je tek u svojim začecima. Zbog toga se motivacija za istraživanje koje će se provesti u ovoj disertaciji prvenstveno krije u činjenici da automatizirani razvoj pravila za raspoređivanje GP-om u različitim okruženjima daje dobre rezultate [27, 28, 29, 32]. S obzirom da se radi o samim počecima istraživanja ovog područja, istraživanje treba ići u smjeru otkrivanja temeljnih postavki koje će se onda kroz daljnja istraživanja moći nadograđivati.

Jedna od otvorenih tema u području primjene GP-a na RCPSP je odabir skupa značajki i funkcija s pomoću kojih će se pravila razvijati, a čija važnost je istaknuta u svim radovima u kojima se GP primjenjuje u razvoju pravila raspoređivanja. U literaturi postoje brojni postupci za odabir značajki [38, 39, 40, 41, 42] te je potrebno pronaći način kako odabrati značajke koje su važne za prioritetna pravila koja će se koristiti u rješavanju RCPSP-a. Osim toga, potrebno je odabrati shemu za generiranje rasporeda, koja će nakon dodjeljivanja prioriteta aktivnosti kreirati raspored.

S obzirom da prioritetna pravila ne uspijevaju ostvariti rezultate koji bi bili slične kvalitete kao oni ostvareni različitim heurističkim metodama primjenjivim u statičkim okruženjima, potrebno ih je dodatno prilagoditi. Različita prioritetna pravila će na različitim instancama problema ostvariti rješenja različite kvalitete. Zbog toga se javlja ideja da se rješenje ne traži koristeći samo jedno prioritetno pravilo nego kombiniranjem više njih. Primjena ansambla je poznata metoda strojnog učenja koja je i kod razvoja pravila raspoređivanja za različita okruženja dovela do poboljšanja rezultata [43, 44, 45, 46, 47, 48].

Rješenja ostvarena prioritetnim pravilima su i nakon primjene ansambala najčešće lošije

kvalitete od onih ostvarenih heurističkim metodama, zbog čega se prioritetna pravila najviše koriste u dinamičkim okruženjima gdje druge metode nisu primjenjive. No, unatoč tome, prednosti prioritetnih pravila poput jednostavnosti i brzine ponekad mogu biti izrazito važne za praksu pa se ona znaju upotrebljavati i u statičkim uvjetima. Statički uvjeti sa sobom nose dodatne mogućnosti prilagodbe prioritetnih pravila s ciljem poboljšanja kvalitete. U skladu s istraživanjima u kojima je ova prilagodba provedena na različite probleme raspoređivanja [44, 49, 50] otvara se i pitanje provedbe prilagodbe na pravila razvijena za RCPSP.

1.2 Glavni doprinosi

U skladu s prethodno opisanom situacijom, ova disertacija usmjerena je prema oblikovanju prioritetnih pravila za RCPSP. Naglasak u oblikovanju prioritetnih pravila stavljen je na automatizirani razvoj pravila GP-om i postizanje što bolje kvalitete prilikom korištenja istih u rješavanju RCPSP-a.

Ispitivanja provedena u ovoj disertaciji sastojat će se od tri glavna dijela. Prvi od njih čini određivanje bitnih dijelova za razvoj prioritetnih pravila i sam razvoj prioritetnih pravila. Kako bi GP mogao oblikovati kvalitetno prioritetno pravilo važno je odabrati skup značajki i funkcija s pomoću kojih će se razvijati pravilo, a koji doprinosi kvaliteti rješenja. Nadalje, potrebno je odrediti i ostale parametre GP-a čije vrijednosti mogu bitno utjecati na dobrotu pravila. Prioritetna pravila uvijek dolaze u kombinaciji sa shemom za generiranje rasporeda koja na osnovu prioriteta dodijeljenog aktivnosti konstruira raspored. U sklopu disertacije potrebno je ispitati kako različite sheme za generiranje rasporeda utječu na kvalitetu rješenja te među njima pronaći najbolju za primjenu na ovaj problem. Osim toga, potrebno je postupak razvoja prioritetnih pravila učiniti što jednostavnijim i omogućiti njegovu laku prilagodbu različitim optimizacijskim kriterijima. Na kraju, potrebno je ispitati kvalitetu razvijenih prioritetnih pravila te ih usporediti s postojećim prioritetnim pravilima iz literature.

Drugi dio ispitivanja usredotočen je na otkrivanje načina kako dodatno poboljšati kvalitetu prioritetnih pravila. Pristup za poboljšanje rezultata ostvarenih prioritetnim pravilima korišten u ovoj disertaciji temeljen je na skupnom učenju, odnosno izgradnji ansambala prioritetnih pravila. Ideja iza ovog pristupa, a koji se često koristi u strojnom učenju, krije se u činjenici da korištenjem više prioritetnih pravila u donošenju odluke prilikom raspoređivanja dolazi do bolje odluke što onda utječe i na kvalitetu rješenja. U ispitivanjima će se ansambli kreirati koristeći različite pristupe iz literature, a koji će biti prilagođeni RCPSP-u. Također, provest će se ispitivanje vezano uz veličinu ansambla, načine kombiniranja prioritetnih pravila u ansambl kao i dati međusobna usporedba različitih načina kreiranja ansambala. Dobiveni rezultati bit će uspoređeni s rezultatima ostvarenim pojedinačnim pravilima s ciljem potvrde da je došlo do poboljšanja rezultata.

U trećem dijelu disertacije bit će napravljena dodatna prilagodba prioritetnih pravila za korištenje u statičkom okruženju. Za prilagodbu prioritetnih pravila koristit će se dva postupka, pri čemu je za jedan od njih potrebno prilagoditi cijeli postupak razvoja pravila, dok je za drugi dovoljno prilagoditi samo shemu za generiranje rasporeda koja se koristi prilikom rješavanja novih instanci problema s pomoću već razvijenih prioritetnih pravila. Prvi od njih je postupak u kojem se raspored gradi iterativno na način da se kroz iteracije prenose određene informacije koje mogu utjecati na kvalitetu rasporeda te je potrebno uvesti nove značajke preko kojih će se to ostvariti. Zbog uvođenja novih značajki potrebno je razviti nova prioritetna pravila koja će se zatim koristiti na novim instancama problema. Drugi postupak sastoji se od sheme u kojoj se ne prati samo jedan smjer raspoređivanja već više njih na način da se u trenucima odluke koju aktivnost rasporediti sljedeću računa kvaliteta više mogućih rasporeda, a ne samo jednog. Ideja i jednog i drugog pristupa je da se što više informacija koje su dostupne u statičkom okruženju iskoristi i na taj način značajno poboljša kvaliteta rješenja i omogući primjena prioritetnih pravila i u statičkim okruženjima. Rezultati ostvareni ovim pristupima bit će uspoređeni s pojedinačnim prioritetnim pravilima i s genetičkim algoritmom (GA) kao predstavnikom heurističkih metoda.

Na temelju prethodnog razmatranja glavne doprinose ove disertacije moguće je sažeti kroz sljedeće tri točke:

1. Skup značajki i funkcija za razvoj prioritetnih pravila i postupak stvaranja prioritetnih pravila za probleme raspoređivanja s ograničenim sredstvima prilagodljiv različitim optimizacijskim kriterijima
2. Metode razvoja prioritetnih pravila temeljene na skupnom učenju
3. Postupci razvoja prioritetnih pravila primjenjivih u statičkoj okolini raspoređivanja

1.3 Organizacija rada

Ova disertacija podijeljena je u 7 poglavlja. Poglavlje 1 i 7 sadrže uvod i zaključak same disertacije, poglavlja 2 i 3 su uvod u problem i metodu rješavanja korištenu u ispitivanjima, dok su poglavlja 4, 5 i 6 ona u kojima se nalaze doprinosi ove disertacije.

Poglavlje 1 se sastoji od uvoda u disertaciju. Ukratko je predstavljen problem kroz njegovu važnost i kompleksnost, kao i motivacija za ispitivanja provedena u disertaciji. Osim toga, istaknuti su glavni doprinosi disertacije te je na kraju dan pregled dijelova disertacije.

Poglavlje 2 donosi detaljni opis problema i njegovih svojstava. Istaknuti su faktori koji utječu na njegovu težinu i kriteriji optimizacije koji se najčešće koriste prilikom rješavanja ovog problema. U ovom poglavlju dan je detaljan pregled metoda za rješavanje RCPSP-a pri čemu je naglasak stavljen na prioritetna pravila i shemu generiranja rasporeda koja se koristi u kombinaciji s njima.

U poglavlju 3 dan je osnovni algoritam GP-a te su objašnjeni njegovi glavni dijelovi kao što

su odabir prikaza rješenja, načini odabira jedinke i genetički operatori. Osim toga, navedeni su različiti načini inicijalizacije populacije i kriteriji zaustavljanja algoritma.

Poglavlje 4 na početku donosi pregled literature u kojoj se GP-om razvijaju prioritetna pravila, a zatim se određuju parametri potrebni za pokretanje GP-a i odabire kriterij optimizacije. Nakon što su se početni parametri odredili, analizirane su značajke i funkcije, kao i potreba za normalizacijom značajki. U sklopu poglavlja ispitani su i različiti skupovi za učenje i provjeru kao i njihova važnost za razvoj pravila. Na kraju poglavlja dani su primjeri razvijenih prioritetnih pravila te je napravljena usporedba razvijenih pravila s onima iz literature.

U poglavlju 5 s ciljem poboljšanja ostvarenih rezultata uvode se ansambli prioritetnih pravila. U sklopu poglavlja predstavljena su dva različita načina kombiniranja prioritetnih pravila u ansambl, kao i četiri različita pristupa za kreiranje ansambla. Nakon što su detaljno objašnjeni korišteni pristupi, provedena su ispitivanja s ciljem određivanja optimalnih parametara za svaki od pristupa. U ispitivanjima je, među ostalim, određena i optimalna veličina ansambla za svaki pojedini pristup. Na kraju je dana međusobna usporedba svih pristupa korištenih u poglavlju, kao i njihova usporedba s pojedinačnim pravilima.

Poglavlje 6 opisuje dodatna dva načina prilagodbe prioritetnih pravila s ciljem poboljšanja kvalitete, a koje je moguće koristiti u statičkom okruženju. Prvi od njih su iterativna prioritetna pravila koja kroz iteracije dodatno poboljšavaju kvalitetu kreiranog rasporeda. Za ovaj postupak uvode se nove značajke koje omogućuju prijenos informacija iz iteracije u iteraciju, te je za ovaj pristup potrebno razviti nova prioritetna pravila. Drugi korišteni pristup predstavlja prilagodbu sheme za generiranje rasporeda koja se koristi prilikom rješavanja novih instanci problema, a za koju je moguće koristiti u prethodnim poglavljima razvijena pravila. Ovaj pristup poznat je pod nazivom rollout i sastoji se od razmatranja više mogućih rasporeda i određivanja najboljeg među njima. Na kraju poglavlja dana je usporedba ovih pristupa s pojedinačnim pravilima, ali i s GA kao predstavnikom ostalih heurističkih metoda koje se mogu koristiti u statičkom okruženju.

I na kraju, poglavlje 7 sadrži glavne zaključke i doprinose ove disertacije te ističe moguće smjerove za buduća istraživanja temeljene na temama koje su ostale otvorene nakon provedenih ispitivanja ili koje nisu bile obuhvaćene ovom disertacijom.

Poglavlje 2

Problem raspoređivanja s ograničenim sredstvima

Problem raspoređivanja s ograničenim sredstvima (engl. *resource constrained project scheduling problem*, skraćeno RCPSP) uvodi Kelley 1963. godine [51]. Unatoč činjenici da se u literaturi prvi put pojavljuje prije više od 50 godina, RCPSP je i danas vrlo aktualan problem zbog svoje kompleksnosti i prisutnosti u stvarnom svijetu te kao takav privlači veliki broj istraživača.

RCPSP je problem u kojem je za skup aktivnosti potrebno odrediti početak vremena izvršavanja pri čemu je potrebno zadovoljiti nekoliko ograničenja te optimizirati jedan ili više kriterija. Prvo ograničenje koje je potrebno zadovoljiti je redoslijed izvršavanja aktivnosti koji je dan uvjetima prednosti (engl. *precedence constraints*), a koji nam definiraju prethodnike, odnosno sljedbenike svake aktivnosti. Osim uvjeta prednosti, potrebno je paziti i na sredstva koja su potrebna za izvršavanja aktivnosti, a koja su dostupna u ograničenim količinama. Dostupna količina sredstva može biti dana za cijeli projekt te se smanjivati kroz vrijeme, pa govorimo o neobnovljivom sredstvu (engl. *non-renewable*) ili može biti stalna u svakoj vremenskoj jedinici pa govorimo o obnovljivom sredstvu (engl. *renewable*). Osim toga, sredstvo može biti i dvostruko uvjetovano (engl. *doubly-constrained*), odnosno njegova količina može biti ograničena i u vremenskom periodu i u cijelom projektu.

U ovom poglavlju definirat će se RCPSP te upoznati s njegovim osnovnim svojstvima kao što su: faktori težine i kriteriji optimizacije. Osim toga bit će dan pregled metoda za rješavanje. Na kraju će biti objašnjena shema za generiranje rasporeda, koja se koristi u različitim heurističkim pristupima, a najviše u kombinaciji s prioritetnim pravilima.

2.1 Opis i definicija problema

Općenito, RCPSP u svom osnovnom obliku definiran je na sljedeći način: dan je projekt koji se sastoji od n aktivnosti čije je vrijeme izvršavanja poznato. Kada aktivnost započne s izvršavanjem nije ju dopušteno prekinuti (engl. *no preemption*). Za svaku aktivnost unutar projekta potrebna je određena vrsta i količina sredstva. Na raspolaganju imamo m sredstava i njihova količina je u svim vremenskim periodima jednaka, odnosno sredstvo je obnovljivo. Potrebno je pronaći raspored takav da ograničenja na sredstva kao i uvjeti prednosti budu zadovoljeni pri čemu je vrijeme završetka projekta minimalno.

Osnovni oblik RCPSP-a svoja proširenja dobiva u vidu drugih kriterija optimizacije prilikom stvaranja rasporeda, dopuštanjem prekida prilikom izvršavanja, uvođenjem trenutka u kojem aktivnost postaje dostupna, trenutka do kojeg treba završiti i sl.

Formalnije, RCPSP definiramo kao problem kombinatorne optimizacije dan kao uređena sedmorka [52]:

$$RCPSP = (A, E, p, R, B, D, c). \tag{2.1}$$

Skup svih aktivnosti sadržanih u projektu dan je s $A = \{A_0, A_1, \ldots, A_n.A_{n+1}\}$, pri čemu su s A_0 i A_{n+1} označene takozvane fiktivne (engl. *dummy*) aktivnosti, koje označavaju početak, odnosno završetak projekta, i čije je trajanje jednako 0. Skup svih aktivnosti iz kojeg su izostavljene fiktivne aktivnosti označavamo s A'.

Uvjeti prednosti dani su skupom E koji se sastoji od uređenih parova (A_i, A_j), $i, j \in \{0, 1, \ldots, n, n+1\}$, koji označava da aktivnost A_i prethodi aktivnosti A_j. Posebno, aktivnost A_0 prethodi svim ostalim aktivnostima, a aktivnost A_{n+1} slijedi nakon svih ostalih aktivnosti.

Vrijeme izvršenja aktivnosti dano je vektorom $p \in \mathbb{N}_0^{n+2}$, pri čemu je i-ta komponenta vektora p, p_i, vrijeme izvršenja aktivnosti A_i. Posebno, što je već ranije spomenuto, vrijeme izvršenja aktivnosti A_0 i A_{n+1} je jednako 0, tj. $p_0 = p_{n+1} = 0$.

Za izvršenje projekta na raspolaganju imamo skup sredstava definiran s $R = \{R_1, \ldots, R_m\}$ čija je dostupnost u svakom vremenskom trenutku dana s vektorom $B \in \mathbb{N}^m$. Zahtjevi aktivnosti za pojedino sredstvo dani su matricom $D \in \mathbb{N}_0^{(n+2)\times m}$. Posebno, za fiktivne aktivnosti, potrebna količina pojedinog sredstva je 0.

Funkcija cilja je definirana kao $c : \chi \to \mathbb{R}$, odnosno kao funkcija koja rasporedu iz skupa svih dopustivih rasporeda pridružuje realnu vrijednost. Ako je ovaj član izostavljen, smatra se da je funkcija cilja vrijeme završetka projekta (engl. *makespan*).

Rješenje RCPSP-a je raspored $S \in \chi \subseteq \mathbb{R}^{n+2}$, pri čemu je i-ta komponenta vektora S u oznaci S_i vrijeme početka izvršenja i-te aktivnosti A_i, a S je takav da je funkcija cilja optimalna. Iz vremena početka aktivnosti lako dolazimo i do vremena završetka aktivnosti na način da vremenu u kojem je aktivnost započela dodamo vrijeme trajanja aktivnosti. Ako s C označimo

vektor vremena završetka aktivnosti, a s C_i njegovu i-tu komponentu, dobivamo: $C_i = S_i + p_i$. Uočimo da je $S_0 = 0$, dok je vrijeme završetka projekta dano sa S_{n+1}.

Ako, dodatno, s $A(t)$ označimo skup svih aktivnosti koje se izvršavaju u trenutku t, odnosno $A(t) = \{A_i \in A : S_i \leq t \leq S_i + p_i\}$ dolazimo do definicije osnovnog RCPSP-a:

Definicija 1 *RCPSP je problem pronalaska rasporeda S za koji je vrijeme završetka projekta minimalno, a pri čemu su zadovoljeni sljedeći uvjeti:*

$$S_j - S_i \geq p_i, \quad \forall (A_i, A_j) \in E \tag{2.2}$$

$$\sum_{A_i \in A(t)} D_{ij} \leq B_j, \quad \forall t \geq 0, \forall R_j \in R \tag{2.3}$$

U gornjoj definiciji uvjet (2.2) osigurava da se aktivnosti izvršavaju redoslijedom danim uvjetima prednosti, dok uvjet (2.3) osigurava da u svakom vremenskom trenutku potrebe za sredstvom od strane aktivnosti ne budu veće od dostupne količine pojedinog sredstva.

RCPSP pripada klasi NP-teških problema, što se može pokazati svođenjem na poznati problem okruženja proizvoljne obrade koji je NP-težak problem [53, 54]. Ako nije zadan krajnji trenutak u kojem projekt mora završiti, onda RCPSP nije NP-težak u jakom smislu, odnosno za njega je moguće, u polinomijalnom vremenu, pronaći dopustivo rješenje. Dopustivo rješenje se u tom slučaju može dobiti jednostavnim stavljanjem aktivnosti u raspored, jednu po jednu, na način da su zadovoljeni uvjeti prednosti.

2.2 Osnovna svojstva

2.2.1 Vremenski prozor za izvršavanje aktivnosti

U osnovnom obliku RCPSP-a sve aktivnosti unutar projekta su dostupne od samog početka izvršavanja projekta i ne postoji krajnji rok u kojem aktivnost treba završiti svoje izvršavanje. Ipak, zbog uvjeta prednosti, postoji vremenski prozor u kojem je moguće izvršavanje pojedine aktivnosti. Kako bi izračunali taj vremenski prozor potrebno je poznavati skup prethodnika i sljedbenika pojedine aktivnosti, kao i krajnji trenutak u planiranju. Skup svih prethodnika aktivnosti A_j označit ćemo s P_j^*, direktnih prethodnika s P_j, svih sljedbenika s F_j^*, direktnih sljedbenika s F_j i krajnji trenutak u planiranju s T.

Krajnji trenutak u planiranju T može biti dan u samom problemu, no ako to nije slučaj, onda ga je moguće izračunati nekom od heuristika ili jednostavnom formulom $\sum_{j=1}^{n} p_j$. Vrijednost T dobivena prethodnom formulom je najčešće puno veća od vrijednosti koja se dobije nekom od heuristika [2] jer se u formuli pretpostavlja da se svaka aktivnost izvršava posebno, odnosno da nije dopušteno paralelno izvršavanje više aktivnosti.

Skup svih poslova koji se mogu izvršavati u vremenskom trenutku t označavamo s $E(t)$ i za njegov izračun su nam potrebne sljedeće varijable:

- ES_j (engl. *earliest start*) - najraniji vremenski trenutak u kojem aktivnost A_j može početi sa svojim izvršavanjem,
- EF_j (engl. *earliest finish*) - najraniji vremenski trenutak u kojem aktivnost A_j može završiti sa svojim izvršavanjem,
- LS_j (engl. *latest start*) - najkasniji vremenski trenutak u kojem aktivnost A_j može početi sa svojim izvršavanjem,
- LF_j (engl. *latest finish*) - najkasniji vremenski trenutak u kojem aktivnost A_j može završiti sa svojim izvršavanjem.

Vrijednosti varijabli ES_j i EF_j dobivamo koristeći sljedeće formule:

$$\begin{gathered} ES_0 = EF_0 = 0 \\ ES_j = \max\{EF_i : i \in P_j\}, EF_j = ES_j + p_j, \quad j \in \{1,\ldots,n+1\}. \end{gathered} \tag{2.4}$$

Izračun vrijednosti varijabli LS_j i LF_j je sličan gornjem s tim da se kod njega kreće od vrijednosti T prema 0. Formule su sljedeće:

$$\begin{gathered} LF_{n+1} = LS_{n+1} = T \\ LF_j = \min\{LS_i : i \in F_j\}, LS_j = LF_j - p_j, \quad j \in \{n,\ldots,0\}. \end{gathered} \tag{2.5}$$

Vremenski prozor za izvršenje aktivnosti A_j je dan segmentom $[ES_j, LF_j]$, a skup $E(t)$ dobijemo kao:

$$E(t) = \{A_j \in A : ES_j + 1 \leq t \leq LF_j, j \in \{0,1,\ldots,n+1\}\}. \tag{2.6}$$

Važno je uočiti da je vremenski prozor za pojedinu aktivnost dobiven koristeći samo uvjete prednosti, odnosno prilikom njegovog izračuna izostavljeni su uvjeti na sredstva. Ovi uvjeti izostavljeni su i kod određivanja skupa aktivnosti koje se mogu izvršavati u trenutku t. Kako je za određivanje točnih vremenskih trenutaka u kojima se aktivnost izvršava u obzir potrebno uzeti i uvjete na sredstva prethodnim izračunom određen je samo raspon vremenskih intervala u kojima se aktivnost može izvršavati.

2.2.2 Faktori težine

Na težinu RCPSP-a utječu brojni čimbenici, a među njima u literaturi se posebno ističu dva: kompleksnost mreže i svojstva sredstava potrebnih za izvršavanje aktivnosti [13].

Kompleksnost mreže (engl. *network complexity* - NC) se definira kao prosječan broj neredudantnih bridova po čvoru u grafu prednosti koji je dan skupom E iz (2.1) [55].

Svojstva sredstava potrebnih za izvršavanje aktivnosti i njihov utjecaj analiziramo kroz dvije

stvari:

1. faktor sredstva (engl. *resource factor* - *RF*): označava prosječnu potrebu aktivnosti za sredstvima i računa se sljedećom formulom

$$RF = \frac{1}{|A'|} \cdot \frac{1}{|R|} \sum_{A_j \in A'} \sum_{R_r \in R} \begin{cases} 1, & \text{ako je } D_{jr} > 0 \\ 0, & \text{inače.} \end{cases}$$

   Ako je za svaku aktivnost potrebna neka količina svih dostupnih sredstava faktor sredstva iznosi 1, a ako aktivnosti nemaju potrebe za sredstvima, odnosno rješavamo problem raspoređivanja bez sredstava, njegova vrijednost je 0.

2. jakost sredstva (engl. *resource strenght* - RS_r): predstavlja vezu između dostupnosti sredstva i potrebe aktivnosti za njim, a njegova vrijednost se računa sljedećom formulom:

$$RS_r = \frac{B_r}{\frac{1}{|A'|} \cdot \sum_{A_j \in A'} D_{jr}}.$$

   Vrijednost jakosti sredstva je 0 ako na raspolaganju nema sredstava, odnosno 1 ako je količina dostupnih sredstava toliko velika da možemo pretpostaviti da rješavamo problem raspoređivanja bez sredstava.

Kolisch i dr. [55] su ispitivanjima pokazali da postoji negativna korelacija između vremena izvršavanja i kompleksnosti mreže, no utjecaj kompleksnosti mreže na vrijeme izvršavanja je neznatan. Najveći utjecaj na vrijeme izvršavanja imaju svojstva sredstava: faktor sredstva je u pozitivnoj korelaciji s vremenom izvršenja, dok je jakost sredstva u negativnoj korelaciji s vremenom izvršenja projekta.

2.2.3 Kriteriji optimizacije

U osnovnoj verziji RCPSP-a za kriterij optimizacije se najčešće koristi vrijeme završetka projekta, odnosno trajanje izvršavanja projekta. No, osim ovog kriterija, postoje i drugi različiti kriteriji koji uglavnom u sebi imaju vremensku komponentu. Prema [56] ti kriteriji su većinom vezani uz kašnjenje (engl. *lateness*) L_j, zaostajanje (engl. *tardiness*) T_j i preuranjenost (engl. *earliness*) E_j. Neki od njih su:

- Suma vremena završetka svih aktivnosti (engl. *sum of all activity completion times*):

$$\sum_{A_j \in A'} C_j.$$

- Težinska suma završetka svih aktivnosti (engl. *total weighted completion time*):

$$\sum_{A_j \in A'} w_j C_j.$$

- Težinsko kašnjenje TWT (engl. *total weighted tardiness*):

$$TWT = \sum_{A_j \in A'} w_j T_j,$$

  pri čemu je $T_j = \max\{0, C_j - dd_j\}$ i dd_j vremenski trenutak do kojeg aktivnost A_j treba biti završena.
- Najveće kašnjenje $L_{\max}$ (engl. *maximum lateness*):

$$L_{\max} = \max_{A_j \in A'} \{L_j\},$$

  pri čemu je $L_j = C_j - dd_j$.
- Težinska preuranjenost i težinsko kašnjenje ET_w (engl. *weighted earliness and weighted tardiness*):

$$ET_w = \sum_{A_j \in A'} \left(w_{E_j} E_j + w_{T_j} T_j\right),$$

  pri čemu je $E_j = \max\{0, -L_j\}$.

2.3 Metode rješavanja

Metode rješavanja su najčešće podijeljene u dvije kategorije: egzaktne algoritme i heurističke algoritme [13].

Egzaktne algoritme karakterizira pretraživanje cijelog dopustivog prostora rješenja te kao takvi jamče optimalnost pronađenog rješenja [57]. Zbog veličine prostora dopustivih rješenja, ovi algoritmi su nepraktični i gotovo neupotrebljivi kod većih instanci [14]. U literaturi pronalazimo tek mali broj uspješne primjene egzaktnih algoritama i to na manje instance problema, kao na primjer: primjena matematičkog [58] i dinamičkog [59] planiranja.

Egzaktne algoritme dijelimo u pet skupina:

1. Cjelobrojno programiranje (engl. *Integer Programming* - IP) - uvodi se veliki broj binarnih varijabli te se na osnovu njihovih vrijednosti kreira raspored izvršavanja aktivnosti. Unatoč činjenici da u literaturi postoji više IP formulacija, u svima njima broj binarnih varijabli raste s porastom broja aktivnosti, pa su kao takve neprimjenjive za velike instance problema. Detaljnije o formulacijama i samom IP-u pronalazimo u [2, 60].
2. Implicitno prebrojavanje (engl. *Implicit Enumeration*) - koriste se stabla prebrojavanja

i uvode određene granice kako bi se zaključilo koji dio stabla sadrži moguće rješenje, a koji dio se može odbaciti. Uz dobru formulaciju i granice, ove metode mogu dati dobre rezultate. Detaljnije u [61, 62].

3. Metoda grananja i ograđivanja (engl. *Branch-and-bound* - B&B) - koriste se stabla, uvjeti grananja i donja granica vrijednosti rješenja. Donja granica vrijednosti rješenja se dobije relaksacijom uvjeta na sredstva. Računanjem najdužeg puta u grafu prednosti eliminiraju se čvorovi koji ne vode optimalnom rješenju. Više o ovim metodama u [63, 64, 65].
4. Dinamičko programiranje (engl. *Dynamic programming*) - osnovni problem dijeli se na manje probleme koji se rješavaju te se na kraju spajaju u rješenje početnog problema [66].
5. Ostalo: skupina u kojoj se nalaze preostale različite metode poput mrežnog protoka [67], različitih dekompozicija problema [59, 68], metode popravljanja rasporeda kada se prekrše ograničenja [69] i sl.

Heuristički algoritmi razvijaju se zbog neprimjenjivosti egzaktnih algoritama na veće instance problema, a koje su prisutne u stvarnim problemima. Ovi algoritmi ne pretražuju cijeli prostor rješenja, kao što to čine egzaktni algoritmi, nego samo njegov dio, pa ne jamče optimalnost, no brzi su i u praksi, uz dobru formulaciju daju dovoljno dobra rješenja. Među njih ubrajamo: genetički algoritam (GA) [10, 13, 15, 16, 70], kolonije mrava [71], slučajno uzorkovanje [72], algoritam raspršenja [73], algoritam simuliranog kaljenja [18], poboljšanje unaprijed i unatrag (FBI) [74], analiza mreže [17] i dr.

Heurističke algoritme dijelimo na dvije skupine: konstruktivne i unaprjeđivačke heuristike. Obje skupine su naziv dobile po načinu kako pronalaze rješenje. Konstruktivne heuristike kreću od praznog rasporeda i izgrađuju ga kroz iteracije. Predstavnici ove skupine su prioritetna pravila koja s pomoću sheme za generiranje rasporeda izgrađuju raspored. Radi se o jednostavnim i brzim heuristikama te kao takve mogu odgovoriti na zahtjeve dinamičkog okruženja. Najpoznatija prioritetna pravila u literaturi [21] su dana tablicom 2.1. S ciljem postizanja boljih rezultata, često se primjenjuje kombinacija ovih metoda [75]. Detaljnije o prioritetnim pravilima i shemi za generiranje rasporeda moguće je pronaći u sljedećim poglavljima.

Unaprjeđivačke heuristike, za razliku od konstruktivnih, kreću od početnog rješenja, odnosno gotovog rasporeda, koji kroz iteracije poboljšavaju [76]. Postupak se ponavlja sve dok postoji mogućnost pronalaska boljeg rješenja ili dok se ne ispuni neki od kriterija zaustavljanja (maksimalni broj iteracija, maksimalno vrijeme izvršavanja, ...). U unaprjeđivačke heuristike ubrajamo metode kao što su evolucijski algoritmi, tabu pretraživanje, algoritam simuliranog kaljenja, kolonije mrava i slični metaheuristički postupci. Rezultati dobiveni unaprjeđivačkim heuristikama su u većini slučajeva bolji nego oni postignuti konstruktivnim heuristikama, no unaprjeđivačke heuristike su računalno zahtjevnije od konstruktivnih i nisu u mogućnosti reagirati na promjene u sustavu, pa su kao takve primjenjive jedino u statičkim okruženjima. Osim toga, kod ovih heuristika se postavlja i pitanje pronalaska dobrog početnog rješenja.

Tablica 2.1: Prioritetna pravila

Prioritetno pravilo	**Opis**	**Sortiranje**	**Formula**
GRPW*	najveći težinski pozicijski rang sljedbenika (engl. *Greatest rank positional weight all*)	max	$p_j + \sum_{i \in F_j^*} p_i$
LST	najkasniji početak (engl. *Latest starting time*)	min	LS_j
LFT	najkasniji završetak (engl. *Latest finish time*)	min	LF_j
GRPW	najveći težinski pozicijski rang direktnih sljedbenika (engl. *Greatest rank positional weight*)	max	$p_j + \sum_{i \in F_j} p_i$
SPT	najkraće trajanje izvršavanja (engl. *Shortest processing time*)	min	p_j
MSL	najmanja vremenska odgoda (engl. *Minimum slack time*)	min	$LS_j - ES_j$
MIS	najveći broj direktnih sljedbenika (engl. *Most immediate successors*)	max	$\left\|F_j\right\|$
MTS	najveći broj sljedbenika (engl. *Most total successors*)	max	$\left\|F_j^*\right\|$

U literaturi, osim jednostavnih metaheurističkih pristupa, susrećemo i njihove različite kombinacije, na primjer tu su takozvani hibridni algoritmi [77], memetički algoritmi [19] i sl. Najčešće se kombinira neki od evolucijskih algoritama s nekom od metoda lokalnog pretraživanja. Ideja ovog pristupa je da se evolucijskim algoritmom odabere dobro područje za pretragu, a lokalnim pretraživanjem ubrza dolazak do najboljeg rješenja u odabranom dijelu. Jedan od novijih radova koji pokazuje da takav pristup može dati dobre rezultate je [20] u kojem je GA spojen s lokalnim pretraživanjem u tako zvani *Genetic-Local Search Algorithm* (GLSA).

Usporedno sa spominjanjem heurističkih algoritama i dobrim rezultatima koje postižu na određenim problemima, dobro se podsjetiti i tzv. *No Free Lunch* teorema, koji dobro pokazuje situaciju koja se događa u primjeni heurističkih algoritama. Naime, po ovom teoremu [78, 79] svi algoritmi pretraživanja imaju jednaku prosječnu učinkovitost gledano na svim problemima danog konačnog skupa problema. Drugim riječima, ako neki od algoritama radi dobro na nekom podskupu problema, tada sigurno postoji neki drugi algoritam koji je od njega bolji na nekom drugom podskupu problema. Iz tog razloga, postoje pokušaji usporednog proučavanja više heuristika kako bi se mogla odabrati ona koja će biti bolja na određenim instancama problema [80, 81].

Problem odabira heuristike koja će biti dovoljno dobra za određeni problem, kao i izgrad-

nja nove heuristike za taj problem dovodi nas do potrebe za izdizanjem na višu razinu koje omogućuje da se umjesto prostora rješenja pretražuje prostor metoda za rješavanje problema. Izdizanje na višu razinu nam omogućuju hiperheuristike. Hiperheuristike dijelimo u dvije skupine: one koje odabiru heuristike i one koje izgrađuju nove heuristike. Jedan od prvih radova u kojem je hiperheuristički pristup korišten za rješavanje RCPSP-a je [82] u kojem je predložen hiperheuristički genetički algoritam koji izgrađuje novu heuristiku. Osim ovog hiperheurističkog pristupa, zabilježen je i manji broj radova u kojima se takav pristup primjenjuje na razvoj prioritetnih pravila [35, 36, 37]. Rezultati iz navedenih radova pokazuju da hiperheuristički pristup daje dobre rezultate u rješavanju RCPSP-a.

2.4 Shema za generiranje rasporeda

Shema za generiranje rasporeda (engl. *schedule generation scheme* - SGS) sadržana je u većem dijelu heurističkih metoda [83]. SGS se može koristiti ili za izradu rasporeda u kombinaciji s prioritetnim pravilima ili prilikom stvaranja dopustivih rješenja za početnu populaciju.

Razlikujemo dvije vrste SGS-a: slijedni (engl. *serial*) (SSGS) i usporedni (engl. *parallel*) (PSGS). Obje vrste SGS-a počinju s praznim rasporedom te kroz iteracije grade djelomične rasporede i to sve dok postoje aktivnosti koje nisu raspoređene. Glavna razlika između ove dvije verzije SGS-a je u načinu kako grade djelomične rasporede. Kod SSGS-a u svakoj iteraciji se raspoređuje točno jedna aktivnost, dok kod PSGS-a u pojedinoj iteraciji se može rasporediti više od jedne aktivnosti. Broj iteracija SSGS-a jednak je broju aktivnosti koje čine projekt, a kod PSGS-a taj broj je jednak broju vremenskih jedinica u kojem je postojala mogućnost da se neka aktivnost rasporedi.

2.4.1 Slijedna shema za generiranje rasporeda

Kao što je prethodno spomenuto, SSGS se sastoji od n iteracija, pri čemu je n broj aktivnosti sadržanih u projektu. U svakoj od iteracija traži se skup trenutno dostupnih aktivnosti za raspoređivanje (engl. *eligible activities set*) - E_i, $i \in \{1,\ldots,n\}$. Skup trenutno dostupnih aktivnosti računa se uzimajući u obzir samo uvjete prednosti, dok se o uvjetima na sredstva vodi računa u kasnijim koracima. Nakon što se dobije taj skup, iz njega se odabire jedna od aktivnosti. Odabir se može raditi na osnovu prioriteta dobivenih nekim od prioritetnih pravila ili već na neki drugi, unaprijed određeni, način. Nakon što se odabere aktivnost, ona se u raspored stavlja u najraniji mogući trenutak pri čemu trebaju biti zadovoljeni uvjeti prednosti i uvjeti na sredstva. Postupak se ponavlja dok sve aktivnosti ne budu raspoređene.

Skup aktivnosti koje su raspoređene do iteracije i označavamo sa S_i, $i \in \{1,\ldots,n\}$, a skup aktivnosti koje su aktivne, odnosno koje se izvršavaju u trenutku t označavamo s $A(t)$ i raču-

namo formulom:

$$A(t) = \{A_j \in A' : C_j - p_j \leq t \leq C_j\}.$$

Koristeći skup $A(t)$ možemo izračunati preostalu količinu sredstva $R_k, k \in \{1,\ldots,m\}$ u trenutku t:

$$\tilde{B}_k(t) = B_k - \sum_{A_j \in A(t)} D_{jk}.$$

Skup koji sadrži vremena završetka aktivnosti koje su do i-te iteracije raspoređene dan je s $C_i = \{C_j : A_j \in S_i\}$. Preciznije skup E_i definiran je formulom:

$$E_i = \{A_j \in A' \setminus S_i : P_j \subseteq S_i\}.$$

Uzimajući u obzir gore uvedene oznake i skupove, algoritam slijedne sheme za generiranje rasporeda dan je algoritmom 1 .

Algoritam 1 Slijedni SGS

1: **Inicijaliziraj:** $C_0 = \{0\}, S_0 = \{0\}$
2: **za** $i = 1$ **do** n **čini**
3: Izračunaj: $E_i, C_i, \widetilde{B}_k(t) \quad R_k \in R, t \in C_i$
4: Odaberi $A_j \in E_i$
5: $EF_j = \max_{A_h \in P_j}\{C_h\} + p_j$
6: $C_j = \min\{t \in [EF_j - p_j, LF_j - p_j] \cap C_i : D_{jk} \leq \widetilde{B}_k(\tau), R_k \in R, \tau \in [t, t + p_j] \cap C_i\} + p_j$
7: $S_i = S_{i-1} \cup A_j$
8: **kraj**

Raspored dobiven SSGS-om je aktivan raspored, odnosno raspored u kojem ne postoji aktivnost koja može završiti ranije bez da time prouzroči kasniji završetak neke druge aktivnosti. Aktivni raspored stvara se zbog toga što se u svakoj iteraciji odabire aktivnost koja se zatim raspoređuje u najraniji mogući trenutak što znači da je nemoguće neku aktivnost rasporediti ranije, a da pri tom neka druga aktivnost ne završi kasnije. Također, dobiveni raspored je raspored u kojem je moguća odgoda, odnosno s obzirom da se raspoređuje aktivnost koja je unaprijed odabrana, ako za nju nema dovoljno sredstava, ona će čekati oslobođenje dovoljne količine sredstva. Prema SSGS-u do trenutka oslobođenja dovoljne količine sredstava za odabranu aktivnost nije moguće započeti izvršavati neku drugu aktivnost bez obzira što za nju može biti dostupno dovoljno sredstava. Formalni dokaz da SSGS rezultira aktivnim rasporedom može se pronaći u [22]. Za probleme raspoređivanja u kojem se optimizira vrijeme trajanja izvršavanja projekta, optimalno rješenje se nalazi unutar aktivnih rasporeda. Vremenska složenost SSGS-a je $\mathcal{O}(n^2 \cdot m)$ [83].

2.4.2 Usporedna shema za generiranje rasporeda

Iteracije kod usporedne sheme za generiranje rasporeda su ovisne o vremenskim trenucima, odnosno svaka iteracija povezana je s točno jednim vremenskim trenutkom. Kako bi pratili raspoređene aktivnosti do trenutka t razlikujemo dva skupa aktivnosti: skup aktivnosti koje su završile sa svojim izvršavanjem do trenutka t, $C(t)$ i skup aktivnosti koje se izvršavaju u trenutku t, $A(t)$. Skup $C(t)$ je definiran formulom:

$$C(t) = \{A_j \in A : C_j \leq t\},$$

dok je $A(t)$ isti kao i kod SSGS-a. U PSGS-u se, kao i kod SSGS-a, vrši odabir aktivnosti iz skupa dostupnih aktivnosti, koji je kod PSGS-a dobiven koristeći i uvjete prednosti i uvjete na sredstva, odnosno dan je formulom:

$$E(t) = \{A_j \in A \setminus (C(t) \cup A(t)) : P_j \subseteq C(t) \wedge D_{jk} \leq \tilde{B}_k(t), R_k \in R\}.$$

Dok se SSGS-om u svakoj iteraciji raspoređuje točno jedna aktivnost, u PSGS-u se u trenutku t odabiru aktivnosti iz skupa $E(t)$ sve dok on nije prazan. Bitno je napomenuti da se, nakon što se neka aktivnost stavi u raspored, skup $E(t)$ ažurira. Pseudokod za PSGS je dan algoritmom 2.

Algorithm 2 Usporedni SGS

1: **Inicijalizraj:** $t = 0, A(0) = \{A_0\}, C(0) = \{A_0\}, \widetilde{B}_k(0) = B_k$
2: **dok je** $|A(t) \cup C(t)| \leq n+2$ **čini**
3: $t = \min_{A_j \in A(t)}\{C_j\}$
4: Izračunaj: $C(t), A(t), \widetilde{B}_k(t), E(t)$
5: **dok je** $|E(t)| > 0$ **čini**
6: Odaberi $A_j \in E(t)$
7: $C_j = t + p_j$
8: Izračunaj: $A(t), \widetilde{B}_k(t), E(t)$
9: **kraj**
10: **kraj**
11: $C_{n+1} = \max_{A_h \in P_{n+1}}\{C_h\}$

Zbog načina odabira aktivnosti iz skupa aktivnosti koje mogu započeti s izvršavanjem u trenutku t i činjenice da se aktivnosti odabiru sve dok taj skup nije prazan, u ovom rasporedu ne postoji vremenski trenutak u kojem je moguće započeti s izvršavanjem aktivnosti, a u kojem izvršavanje nije započelo, odnosno raspored dobiven PSGS-om je raspored bez odgode [22]. Rasporedi bez odgode su podskup aktivnih rasporeda i ako je kriterij optimizacije minimizacija vremena trajanja projekta, unutar njih se ne mora nalaziti optimalni raspored. Unatoč toj činjenici, u praksi, u većini slučajeva, ova verzija sheme za generiranje rasporeda pronalazi bolja rješenja od SSGS-a. Vremenska složenost PSGS-a je također $\mathcal{O}(n^2 \cdot m)$ [83].

Poglavlje 3

Genetičko programiranje

Genetičko programiranje (engl. *genetic programming*, skraćeno GP) je tehnika evolucijskog računarstva koja rješava problem automatizirano bez potrebe poznavanja ili specificiranja oblika strukture rješenja unaprijed [84]. GP je izrazito srodan genetičkom algoritmu, te se od njega razlikuje samo u prikazu jedinke, odnosno kromosomu. U GP-u jedinka je program, izraz ili matematička funkcija koja predstavlja rješenje zadanog problema [84, 85]. Ideja za GP dolazi iz prirodne evolucije. GP bez poznavanja konkretne strukture rješenja odabirom boljih jedinki, kombiniranjem njihovih dijelova te malim promjenama unutar samih jedinki razvija nove jedinke koje kroz generacije postaju sve bolje i postižu bolja rješenja danog problema.

Oslanjajući se na temelje koje još 1990. godine postavlja Koza [86] GP postaje jedna od tehnika koja se koristi za rješavanje velikog broja optimizacijskih [85] i klasifikacijskih problema [87]. Rezultati koje GP ostvaruje kod rješavanja problema u velikom broju slučajeva su kompetitivni s rezultatima ostvarenim oslanjajući se na znanje stručnjaka u pojedinim područjima [33]. Osim toga, činjenica da GP može iz jednostavnih dijelova izgrađivati puno složenije i, koristeći genetičke operatore, pretraživati prostor rješenja, čini ovu tehniku idealnom za hiperheuristički pristup [25, 26, 88, 89]. GP je u svrhu razvoja novih heurističkih procedura kao hiperheuristika korišten u problemima kao što su: problem pakiranja (engl. *bin packing*) [90, 91, 92], raspoređivanje projekata (engl. *project scheduling*) [34, 35, 36], pravljenje rasporeda aktivnosti (engl. *timetabling*) [93, 94], usmjeravanje vozila (engl. *vehicle routing problem*) [95, 96, 97] i dr.

U ostatku poglavlja dan je osnovni algoritam GP-a. Nakon toga, glavni dijelovi GP-a su izdvojeni i dodatno objašnjeni.

3.1 Osnovni algoritam

Genetičko programiranje, kao i genetički algoritam, ne radi samo s jednim rješenjem, nego koristi cijelu populaciju rješenja. Rješenje unutar populacije naziva se jedinka. Na početku je potrebno inicijalizirati populaciju ili nasumično ili koristeći neki drugi postupak koji može pro-

naći populaciju bolju od nasumično kreirane. Nakon što se inicijalizira početna populacija iz nje se odabiru jedinke od kojih se, s ciljem njihovog poboljšanja, a koristeći genetičke operatore križanja i mutacije, dobivaju nove jedinke. Operator križanja koristi se za kreiranje novih jedinki kombiniranjem dijelova, iz populacije odabranih jedinki, dok se mutacija koristi kako bi se novonastalu jedinku dodatno izmijenilo. Više o ovim operatorima moguće je pronaći u narednim poglavljima. Sama procedura GP-a dana je algoritmom 3, a njezina osnovna ideja je sadržana u evoluciji koja se svakodnevno događa u prirodi, pri čemu bolje jedinke opstaju, a lošije se eliminiraju.

Algoritam 3 Genetičko programiranje

```
1: inicijaliziraj početnu populaciju
2: ponavljaj
3:     na osnovu dobrote odaberi jedinke
4:     kreiraj nove jedinke koristeći genetičke operatore
5: dok nije zadovoljen neki od kriterija zaustavljanja
6: vrati najbolju jedinku kao rješenje
```

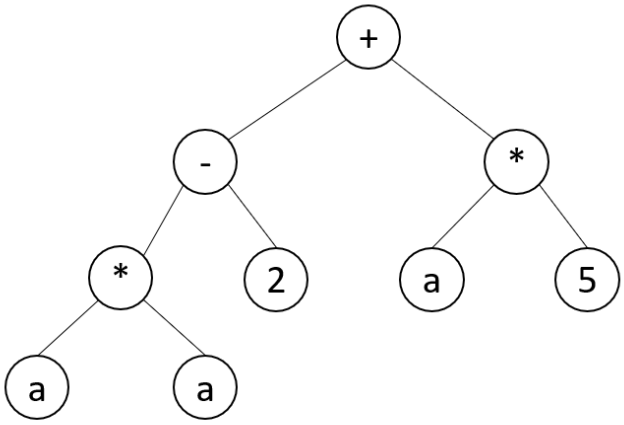
Kao i kod GA, i ovdje razlikujemo dvije osnovne verzije GP-a: eliminacijski i generacijski. Eliminacijski GP u svakoj iteraciji eliminira unaprijed određeni broj jedinki, pri čemu se s većom vjerojatnošću iz populacije eliminiraju one koje su lošije, te se u populaciju dodaje isti broj novih jedinki nastalih križanjem odabranih jedinki, a koje su bolje nego one koje se eliminiraju.

Kod generacijskog GP-a ne eliminira se dio jedinki, nego se kreira cijela nova generacija jedinki. Prilikom kreiranja novih jedinki koristi se neki od genetičkih operatora križanja i mutacije. Promjena cijele generacije jedinki može dovesti do toga da se tijekom prijelaza iz jedne generacije u drugu izgube najbolja rješenja, odnosno jedinke. Kako bi se to spriječilo koristi se elitizam. Elitizam omogućuje zadržavanje unaprijed određenog broja najboljih jedinki koje se bez promjene prenose u novu generaciju.

3.2 Prikaz rješenja

GP koristeći jednostavnije blokove izgrađuje kompleksnije strukture, matematičke funkcije ili programe koji trebaju biti u mogućnosti predstaviti rješenje problema. Kako bi GP mogao to i činiti, bitno je odabrati prikaz koji će to omogućiti. Osim što je potrebno omogućiti prikaz kompleksnijih struktura, potrebno je da se promjene koje se događaju koristeći genetičke operatore mogu lagano izvesti. Zbog toga se za prikaz jedinki najčešće koristi stablo. Primjer jedinke prikazane stablom dan je na sl. 3.1.

Prilikom korištenja GP-a u kojem je rješenje dano u obliku stabla, potrebno je odabrati što sve može biti čvor (engl. *node*) u stablu. Skup svih elemenata koji mogu predstavljati čvor



Slika 3.1: Prikaz jedinke koja predstavlja izraz: $a \cdot a - 2 + a \cdot 5$

u stablu nazivamo skupom primitiva (engl. *primitive set*). Skup primitiva sastoji se iz dva podskupa: skupa značajki (engl. *terminal set*) i skupa funkcija (engl. *function set*).

Skup značajki sastavljen je od varijabli ili konstanti te se može pojaviti samo u listovima (engl. *leaf*) stabla. Svaki od ovih čvorova predstavlja svojevrsni ulaz (engl. *input*) jedinke. Skup značajki je domenski specifičan te ga je potrebno odabrati za svaki problem posebno.

Skup funkcija sastoji se od iskaza, operatora i funkcija, te može, ali i ne mora biti domenski specifičan. Spektar dostupnih funkcija je prilično velik i obuhvaća funkcije poput Booleovih (AND, OR, NOR, XOR) i aritmetičkih, uvjeta grananja (IF, ELSE, THEN) te mnoge druge poput petlji, pridruživanja varijabli, izraza za kontrolu tijeka i sl. Prilikom korištenja određenog skupa funkcija potrebno je osigurati da su one dobro definirane za sve ulaze, pa tako na primjer ako u skupu imamo operaciju dijeljenja potrebno je definirati što se događa u slučaju dijeljenja s 0.

Odabir skupa značajki i funkcija je bitan za uspješnost GP-a. Ovi skupovi moraju biti dovoljno raznovrsni, ali i obuhvatiti sva bitna svojstva problema kako bi se mogla izgraditi struktura koja će moći dovoljno dobro predstaviti rješenje problema. S druge strane, ove skupove je potrebno držati minimalnim, jer povećavanjem njihovog broja prostor rješenja raste, pa se time i konvergencija GP-a usporava. Pravilo prilikom izbora ovih skupova može se sažeti u sljedeće: skupovi trebaju biti dovoljno dobri da reprezentiraju rješenje, ali nije ni potrebno previše vremena trošiti na pravljenje domenski specifičnih funkcija i značajki jer je GP u mogućnosti kombiniranjem jednostavnih konstrukcija stvoriti prilično složene izraze i odgovoriti na zahtjeve problema.

Također, kod prikaza rješenja korištenjem stabla najčešće se definira i njegova maksimalna veličina. Veličina se definira ili ograničavanjem broja čvorova u stablu ili ograničavanjem njegove dubine. Uvođenje ovog parametra javlja se kao odgovor na problem pretjeranog rasta (engl. *bloat*), koji predstavlja rast jedinke bez značajnog poboljšanja njezine kvalitete. O važ-

nosti ovog problema govore brojni radovi u literaturi [84, 98, 99, 100]. Veličina jedinke ne utječe samo na vremensko izvršavanje algoritma, nego utječe i na interpretabilnost rješenja.

Iako je prikaz jedinke stablom najčešće korišten u GP-u u literaturi se pronalaze i drugi prikazi poput linearnog prikaza (engl. *linear GP*) [101], prikaza temeljnog na gramatici (engl. *grammar based GP*) [102, 103], kartezijevog prikaza (engl. *Cartesian GP*) [104] i prikaza temeljenog na grafu (engl. *graph based GP*) [105].

3.3 Inicijalizacija populacije

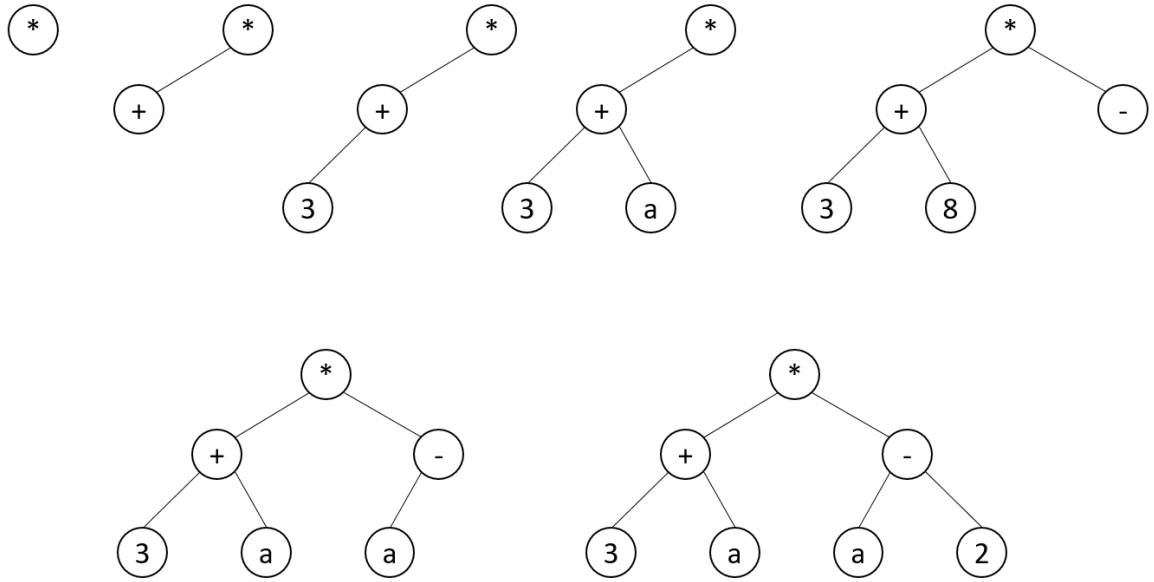
Incijalizacija populacije u GP-u, kao i u drugim evolucijskim tehnikama se najčešće radi nasumično. Postoje brojni različiti pristupi za nasumično generiranje populacije. Najčešće korištene metode su: metoda potpunosti (engl. *full method*), metoda rasta (engl. *grow method*) i pojačana pola-pola metoda (engl. *ramped half-and-half method*). Metoda potpunosti i metoda rasta su dvije najjednostavnije i najstarije korištene metode, dok pojačana pola-pola metoda je kombinacija te dvije.

Kod sve tri metode, dubina stabala nastalih jedinki ne prelazi maksimalnu, unaprijed određenu, dubinu. Dubina čvora predstavlja broj bridova koje je potrebno proći da bi se od korijenskog čvora došlo do tog čvora. Dubina korijenskog čvora je 0, dok dubina lista stabla s najvećom dubinom predstavlja dubinu stabla. Na primjer, stablo prikazano na sl. 3.1 je dubine 3.

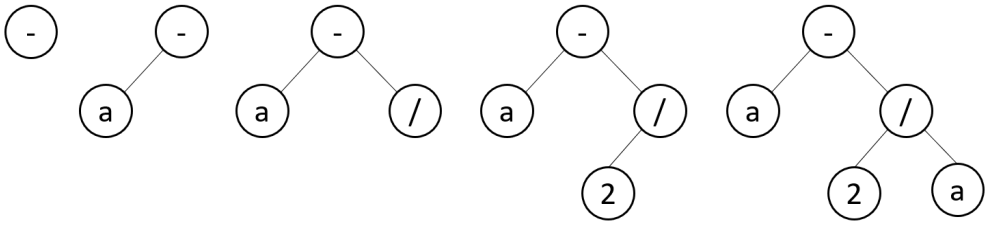
Metodom potpunosti stvaraju se jedinke kod kojih je dubina svih listova stabla jednaka maksimalnoj dubini stabla [84]. Postupak stvaranja jedinke kreće od korijenskog čvora za koji se nasumično odabire jedna od funkcija iz skupa funkcija, te se taj postupak ponavlja sve dok se ne dođe do maksimalne dubine, odnosno lista stabla, kada se za čvor odabire jedna od značajki. Primjer nastanka jedinke ovom metodom s maksimalnom dubinom 2 dan je na slici 3.2. Kako bi GP bio u mogućnosti postići dobro rješenje, potrebno je da jedinke unutar populacije budu raznolike. Iako, funkcije odabrane iz skupa, mogu primati različiti broj argumenata, jedinke nastale ovom metodom imaju sličan oblik, što rezultira nedovoljnom raznolikosti unutar populacije. U ovom slučaju, potrebni su genetički operatori koji će tu raznolikost moći ostvariti tijekom iteracija.

Za razliku od metode potpunosti, kod metode rasta čvorovi značajki se ne moraju nalaziti na maksimalnoj dubini stabla. To se postiže dopuštanjem da se u bilo kojem trenutku za čvor može odabrati ili značajka ili neka od funkcija, osim na maksimalnoj dubini gdje se nužno odabire značajka kako bi stablo ostalo na dopuštenoj dubini. Prikaz nastajanja jedinke metodom rasta dan je na sl. 3.3.

Iako, metoda rasta stvara jedinke s više oblika nego što to čini metoda potpunosti, ni dalje ta raznolikost nije prevelika niti u oblicima niti u veličinama stabla. Zbog toga, Koza u [85]



Slika 3.2: Prikaz nastajanja jedinke metodom potpunosti



Slika 3.3: Prikaz nastajanja jedinke metodom rasta

predlaže kombinaciju ovih dviju metoda, odnosno pojačanu pola-pola metodu [84]. Ideja ove metode skriva se u samom nazivu - pola jedinki populacije nastaje metodom potpunosti, a pola metodom rasta. Osim kombiniranja ovih dviju metoda, ova metoda koristi i raspon maksimalnih dubina stabla kako bi u populaciji postojala različitost jedinki. Maksimalna dubina stabla nove jedinke odabire se nasumično pri čemu ona mora biti manja ili jednaka originalnoj maksimalnoj dubini.

Navedene metode inicijalizacije su jednostavne za implementirati, no bitno je imati na umu da je njima ponekad teško kontrolirati oblik i veličinu nastalih jedinki [84]. Problem kod metode rasta je osjetljivost same metode na veličinu skupa funkcija i skupa značajki. Ako je skup funkcija puno manjeg kardinaliteta od skupa značajki, tada će jedinke nastale metodom rasta, bez obzira na postavljenu maksimalnu dubinu, većinom biti male dubine dok će u obrnutom slučaju metoda rasta biti prilično slična metodi potpunosti [84].

Ako postoje neka svojstva problema koja su poznata, a mogu utjecati na kvalitetu rješenja, ona mogu biti ugrađena u same jedinke već prilikom inicijalizacije, odnosno inicijalizacija ne mora biti potpuno nasumična. Više detalja o ovoj vrsti inicijalizacije moguće je pronaći u [84].

3.4 Funkcija dobrote

S obzirom da evolucijske tehnike svoju glavnu ideju crpe iz prirodne selekcije, a za koju je poznato da bolje jedinke uspijevaju preživjeti, dok lošije budu eliminirane, potrebno je na neki način odrediti dobrotu jedinke. Kako bi za neku jedinku mogli odrediti koliko je dobra pridružujemo joj numeričku vrijednost. Što je numerička vrijednost veća smatramo da je jedinka bolja. Funkciju koja jedinki pridružuje numeričku vrijednost nazivamo funkcija dobrote (engl. *fitness function*).

Funkcija dobrote, kao i ostali, dosad spomenuti dijelovi GP-a, može bitno utjecati na kvalitetu postignutog rješenja. Ova funkcija prvenstveno utječe na konvergenciju algoritma. Kod dizajna funkcije dobrote bitna je sposobnost razlikovanja jedinki s obzirom na njezina "dobra" i "loša" svojstva i da to bude vidljivo u numeričkoj vrijednosti kojom mjerimo dobrotu. Kod GP-a, koji se može smatrati i metodom strojnog učenja, cijela populacija se evaluira na više instanci problema, odnosno uvijek postoji skup za učenje koji bi trebao dobro reprezentirati i buduće instance problema. S jedne strane, korištenje više instanci problema utječe na sposobnost generalizacije, a s druge utječe na potrebno vrijeme za evaluaciju rješenja.

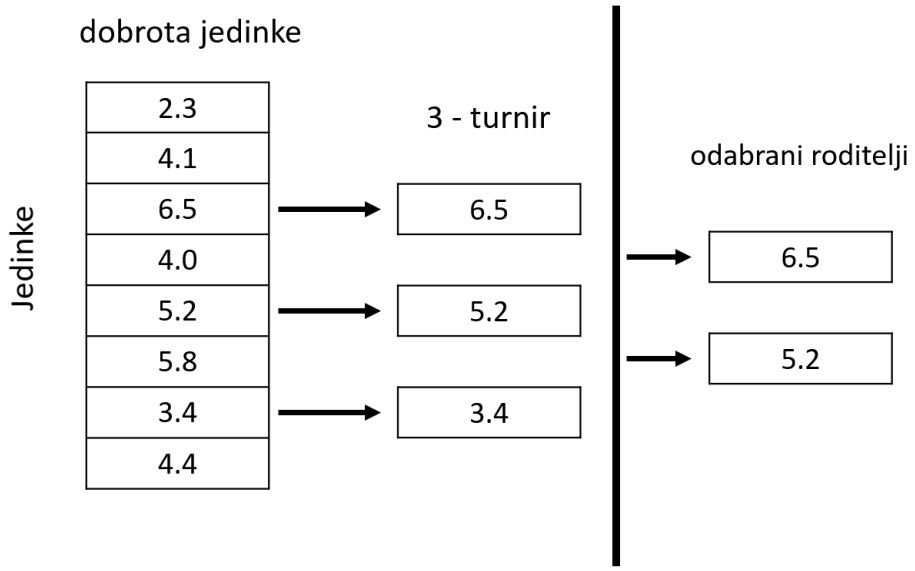
3.5 Odabir jedinke

Odabir jedinki je još jedan od bitnih dijelova GP-a, kojim se osigurava da bolje jedinke imaju veću vjerojatnost preživljenja od lošijih, te da s većom vjerojatnosti sudjeluju u stvaranju novih jedinki. U literaturi je moguće pronaći brojne postupke odabira [106, 107, 108], a u ovom poglavlju, dio njih bit će detaljnije objašnjen.

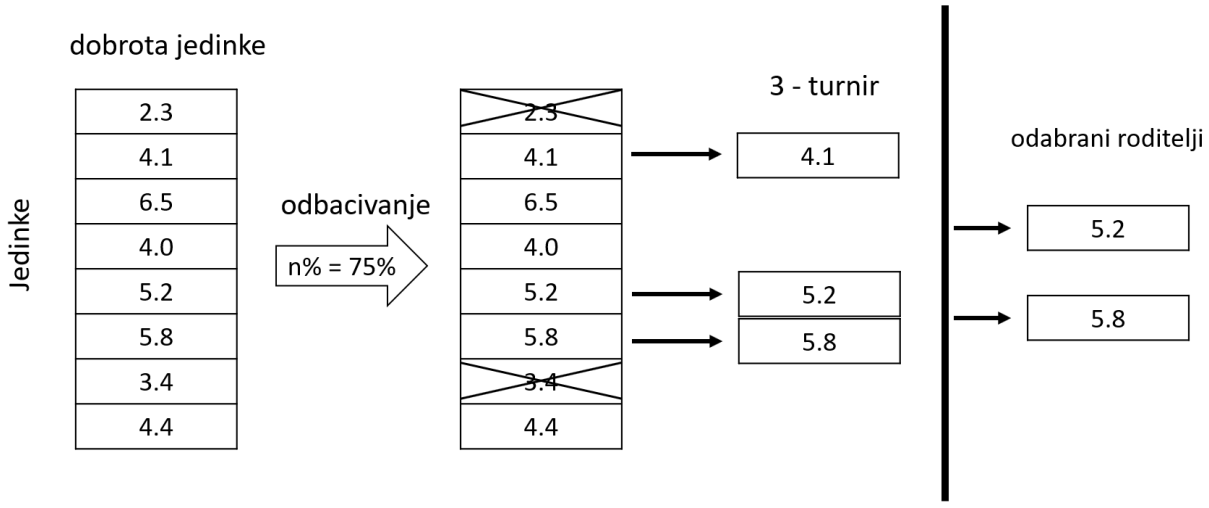
Turnirski odabir (engl. *tournament selection*) jedan je od najpoznatijih vrsta odabira kod evolucijskih algoritama, a koji je ujedno i lagano implementirati. Nasumično se odabire $k, k \geq 2$ jedinki, te se dvije najbolje među njima uzimaju za roditelje nove jedinke. Kod generacijskog GP-a novonastalu jedinku stavlja se u novu generaciju, dok se kod eliminacijskog GP-a najlošiju od k odabranih jedinki zamjenjuje s novonastalom. Iako turnirski odabir prednost daje boljim jedinkama, zbog nasumičnog odabir zadržana je mogućnost da i lošije jedinke budu roditelji nove jedinke. Veličina turnirskog odabira najčešće je 3. Prikaz turnirskog odabira vidljiv je na slici 3.4.

Sličan turnirskom odabiru je odabir odsijecanjem (engl. *truncation selection*) kod kojeg se odabire između $n\%$ najboljih jedinki. Primjer odabira odsijecanjem dan je na slici 3.5.

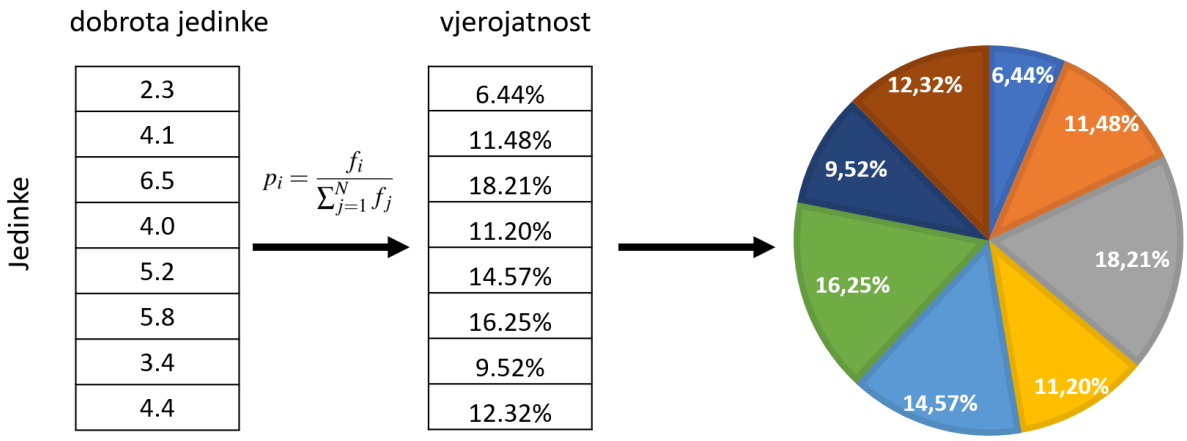
Sljedeća vrsta odabira je jednostavni odabir (engl. *roulette wheel selection, fitness proportionate selection*) u kojoj je vjerojatnost odabira jedinke određena njezinom dobrotom. Ako s $f_i, i \in \{1, \ldots, N\}$ označimo dobrotu jedinke i, tada je vjerojatnost njezinog odabira dana sljede-



Slika 3.4: Turnirski odabir za $k = 3$



Slika 3.5: Odabir odsijecanjem, za $n = 75$



Slika 3.6: Jednostavni odabir

ćim izrazom:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}.$$

Problem ove vrste odabira je njezina računalna kompleksnost, jer svaka promjena u populaciji zahtijeva ponovni izračun vjerojatnosti odabira. Osim toga, ako unutar populacije postoji nekoliko jedinki čija je dobrota puno veća od dobrote ostalih jedinki, tada će se one većinom odabirati za roditelje novih jedinki što će smanjiti raznolikost unutar populacije, te prouzrokovati da algoritam konvergira prema lokalnom optimumu. Primjer jednostavnog odabira dan je na slici 3.6. Cijeli postupak odabira može se prezentirati krugom, u kojem svakoj jedinki pripada jedan njegov isječak koji je proporcionalan s njezinom dobrotom. Odabir jedinke možemo zamisliti kao vrtnju kruga pokraj kojeg je fiksirana točka, i jedinka čiji isječak se zaustavi pokraj te točke je ona koja će biti odabrana.

U literaturi pronalazimo i druge vrste odabira, kao što su odabir linearnim rangiranjem (engl. *linear ranking selection*) i odabir eksponencijalnim rangiranjem (engl. *exponential ranking selection*), o čemu se detaljnije može pronaći u [106, 108].

3.6 Genetički operatori

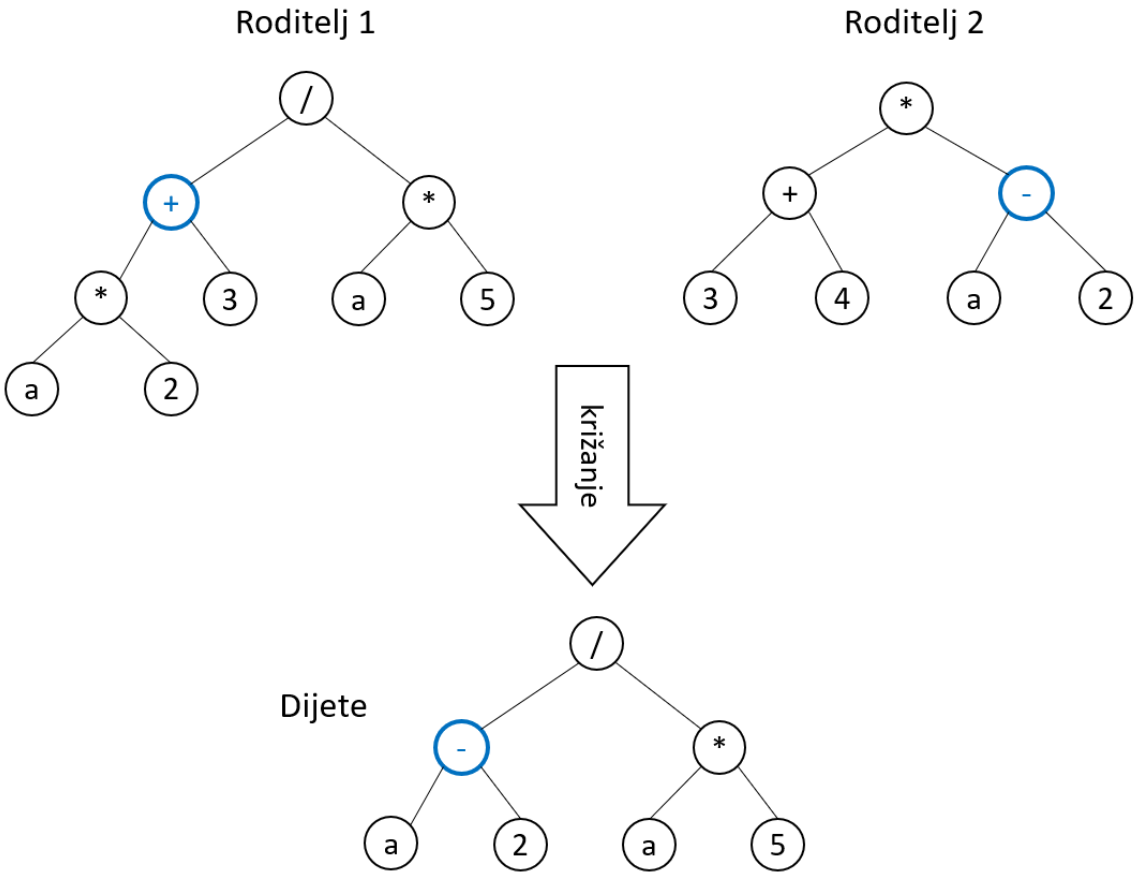
Genetički operatori su operatori koji omogućuju GP-u pretraživanje prostora rješenja. Dva najznačajnija među njima su križanje i mutacija. Zbog prikaza rješenja ovi operatori se u GP-u dosta razlikuju u odnosu na druge evolucijske algoritme [84].

3.6.1 Križanje

Križanje (engl. *crossover*) je operator koji na određeni način kombinira genetičke materijale roditelja (engl. *parents*) te na osnovu njih stvara novu jedinku - dijete (engl. *child*). Prilikom

križanja jedinka nasljeđuje jedan dio od jednog roditelja, a drugi od drugog s ciljem da novonastala jedinka ima još veću dobrotu. U literaturi je moguće pronaći brojne verzije križanja. Neke od njih su: križanje podstabala, križanje s jednom točkom prekida i jednoliko križanje [84].

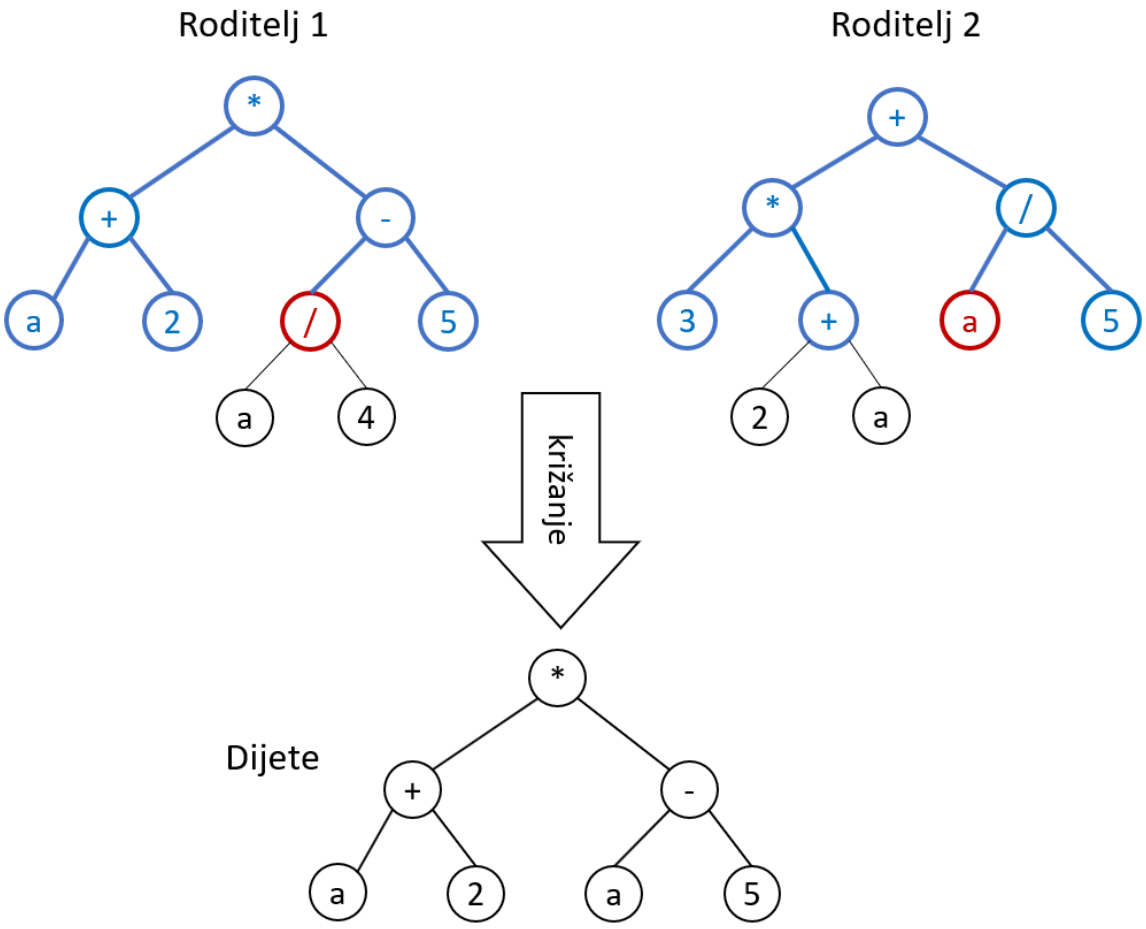
Križanje podstabala (engl. *subtree crossover*) jedno je od najjednostavnijih i najčešće korištenih križanja [84]. U svakom stablu odabere se jedan čvor, takozvana točka križanja. Nova jedinka nastaje tako da se podstablo određeno odabranim čvorom kod prvog roditelja zamijeni podstablom određenim odabranim čvorom kod drugog roditelja. Primjer ovog križanja dan je slikom 3.7, gdje je točka križanja kod oba roditelja označena plavom bojom. Kod križanja podstabla stavlja se i uvjet na odabir čvora - u 90% slučajeva odabire se funkcijski čvor, a tek u 10% značajka. Razlog tome je što se podstablo određeno značajkom sastoji samo od te značajke te ne dolazi do značajne razmjene genetičkih materijala između roditelja.



Slika 3.7: Križanje podstabala

U križanju s jednom točkom prekida (engl. *one-point crossover*) definiraju se zajednička područja (engl. *common region*), koja predstavljaju područja u kojima je oblik stabla jednak kod oba roditelja. Za točku prekida može se odabrati samo čvor koji pripada zajedničkom području i nalazi se na istoj poziciji kod oba roditelja. Kao i kod križanja podstabala, nova jedinka nastaje tako da se kod prvog roditelja podstablo određeno točkom prekida zamijeni

podstablom određenim točkom prekida kod drugog roditelja. Primjer ovog križanja dan je na slici 3.8, gdje je zajedničko područje označeno plavom bojom, a točka prekida crvenom.

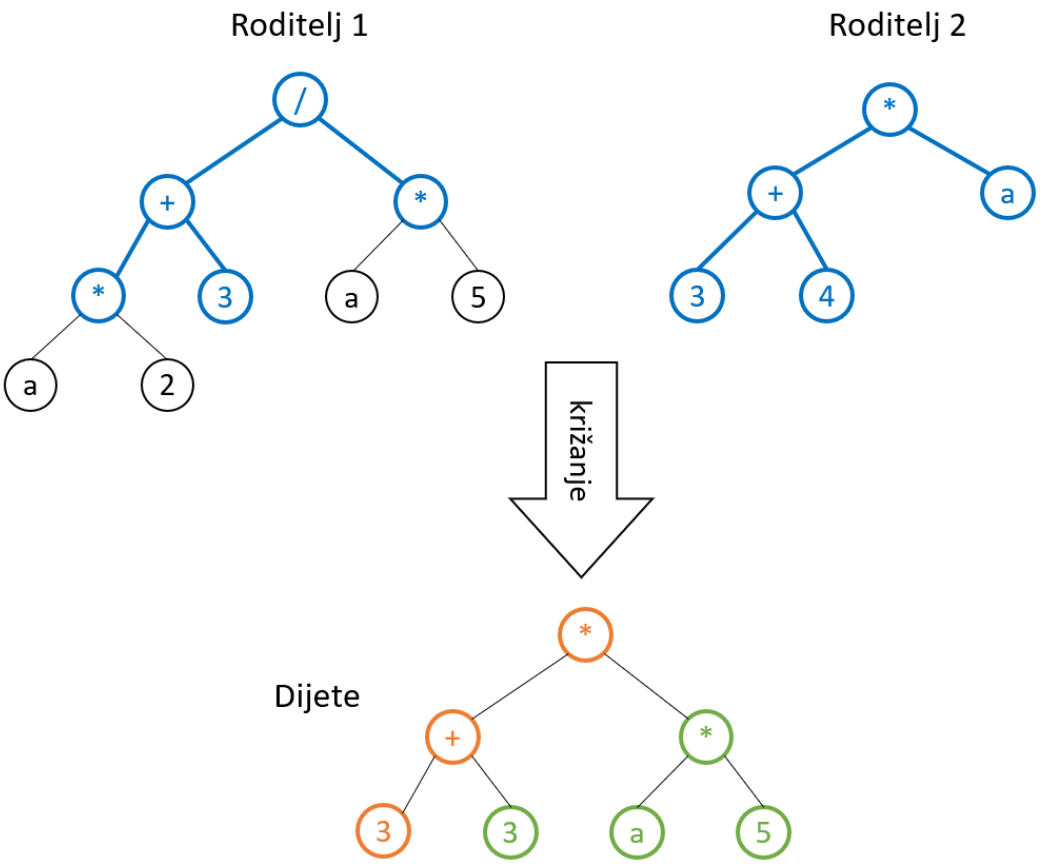


Slika 3.8: Križanje s jednom točkom prekida

Zajednička područja određuju se i kod jednolikog križanja (engl. *uniform crossover*). Nova jedinka nastaje tako da se za svaki čvor u zajedničkom području nasumično odabire hoće li biti naslijeđen od prvog ili drugog roditelja. Ako se na granici zajedničkog područja odabere funkcijski čvor, tada se nasljeđuje i cijelo podstablo određeno tim čvorom. Ovo križanje, iako slično križanju s jednom točkom prekida, omogućuje nastanak veće raznolikosti unutar populacije. Primjer ovog križanja je vidljiv na slici 3.9 gdje je zajedničko područje označeno plavom bojom, a kod djeteta čvorovi preuzeti iz prvog roditelja su zelene boje, a iz drugog narančaste.

3.6.2 Mutacija

Iako je mutacija (engl. *mutation*) korištena već u ranim primjenama kod evolucije programa [109, 110], njezinu upotrebu je izostavio sam tvorac GP-a Koza [85, 111] koji je htio pokazati da ona nije potrebna kako bi GP pronašao dobra rješenja. Njezino izostavljanje u samim počecima utjecalo je na činjenicu da se ona i danas često izostavlja kod GP-a [84]. No u radu [112]



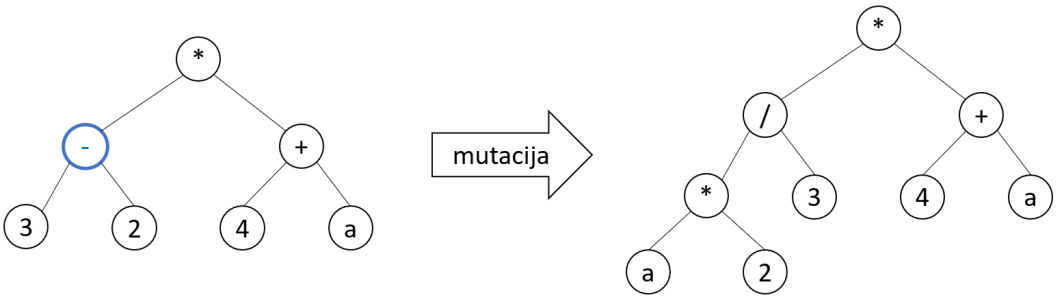
Slika 3.9: Jednoliko križanje

autorica opravdava njezinu upotrebu, a kasnije i Koza [113] preporučuje upotrebu, iako u maloj količini.

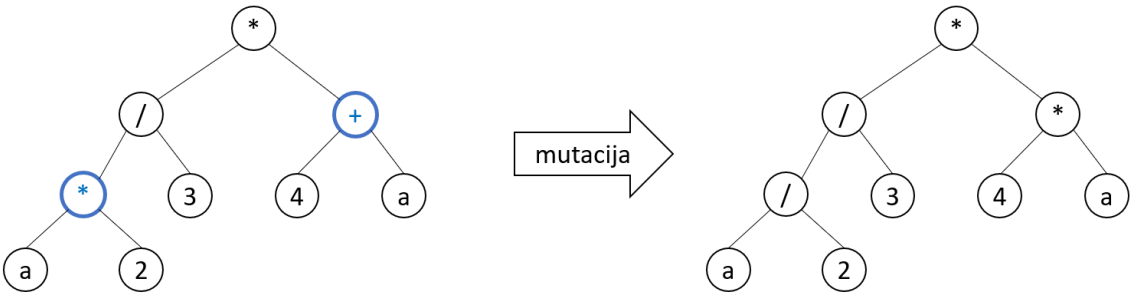
Mutacija se sastoji od nasumičnih promjena jedinki s ciljem uvođenja novog genetičkog materijala. Najčešće se koristi nad jedinkom koja je nastala križanjem. Mutacija se većinom ne izvodi u svim slučajevima, već s određenom vjerojatnošću. Njezina upotreba ne jamči postizanje boljih rješenja, no omogućava da se rješenje pomakne od lokalnog optimuma, ako je u njega upalo. GP često kovergira prema lokalnom optimumu zbog činjenice da križanje vodi k stvaranju jedinki sličnih trenutno najboljim rješenjima u populaciji. Neke vrste mutacija su: mutacija podstabla, mutacija zamjenom čvorova, mutacija permutacijom i mutacija smanjivanjem. Detaljnije o ostalim vrstama moguće je pronaći u [84].

Mutacija podstabla (engl. *subtree mutation*) je najčešće korištena verzija mutacije. Nasumično se odabere točka mutacije stabla i podstablo određeno tom točkom zamijeni se s nasumično generiranim podstablom. Primjer ove mutacije nalazi se na slici 3.10 na kojoj je točka mutacije plave boje.

Mutacija zamjenom čvorova (engl. *point mutation, node replacement mutation*) je mutacija u kojoj se, za razliku od mutacije podstabla, ne događa promjena u samo jednom čvoru, već se za svaki čvor ona događa s određenom vjerojatnošću. Mutacija čvora odvija se tako da se on zamijeni s nekim drugim čvorom iz skupa primitiva, pri čemu, ako se radi o funkcijama, je



Slika 3.10: Mutacija podstabla

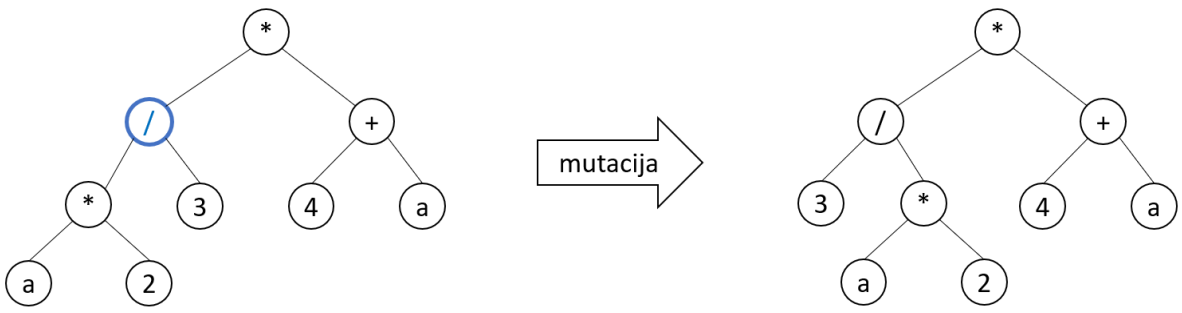


Slika 3.11: Mutacija zamjenom čvorova

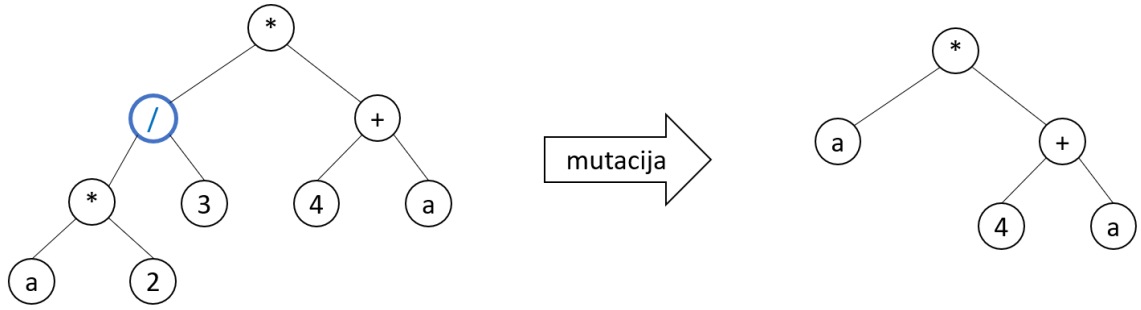
bitno da se promjena dogodi jedino između funkcija s jednakim brojem argumenata. Mutacija zamjenom čvorova na primjeru je pokazana na slici 3.11 gdje su čvorovi odabrani za mutaciju plave boje.

Mutacija permutacijom (engl. *permutation mutation*) nasumično odabire funkcijski čvor i nasumično permutira njegove argumente (podstabla). Primjer je dan na slici 3.12 na kojoj je čvor u kojem će se permutirati argumenti obojan plavom bojom.

Mutacija smanjivanjem (engl. *shrink mutation*) mijenja nasumično odabrano podstablo s nasumično odabranom značajkom. Ovom mutacijom veličina stabla se smanjuje. Primjer ove mutacije vidljiv je na slici 3.13 gdje je nasumično odabran čvor, čije podstablo će se zamijeniti nekom od značajki, plave boje.



Slika 3.12: Mutacija permutacijom



Slika 3.13: Mutacija smanjivanjem

Genetički operatori se u GP-u, kao i kod ostalih evolucijskih tehnika, primjenjuju s određenom vjerojatnošću. Vjerojatnost primjene nekog operatora križanja najčešće je 90% i više, dok je vjerojatnost primjene mutacije puno manja i najčešće je ispod 1% [84]. Ako je vjerojatnost križanja i mutacije u zbroju manja od 100% i iznosi p, tada se najčešće koristi i operator reprodukcije (engl. *reproduction*) s vjerojatnošću $1 - p$. Reprodukcija je operator koji na osnovu dobrote odabire jedinke i njihovu kopiju stavlja u sljedeću generaciju.

3.7 Kriteriji zaustavljanja

GP pripada heurističkim postupcima koji ne pretražuju cijeli prostor stanja i kao takav ne jamči pronalazak optimalnog rješenja, zbog čega je potrebno definirati određeni kriterij zaustavljanja ili više njih.

Kriterije zaustavljanja potrebno je oprezno birati, jer ako GP završi prerano, možemo ostati daleko od dobrog rješenja, ili s druge strane, ako pustimo da se predugo izvršava može doći do prenaučenosti (engl. *overfitting*). U slučaju prenaučenosti, dobiveno rješenje prilikom primjene na drugu instancu problema, ako ona nema vrlo slične karakteristike kao i instance u skupu za učenje, neće dati dobro rješenje, odnosno sposobnost generalizacije je uvelike smanjena.

Neki od najčešće korištenih kriterija zaustavljanja su: maksimalni broj iteracija, maksimalni broj generacija, maksimalni broj evaluacija, maksimalno vrijeme izvršavanja, stagnacija (prijelazom iz jedne generacije u drugu nije došlo do poboljšanja najboljeg rješenja) i dobrota pronađenog rješenja (ako je poznato optimalno rješenje, ili ako je poznata vrijednost koja je prihvatljiva za rješenje).

Poglavlje 4

Oblikovanje prioritetnih pravila za problem raspoređivanja s ograničenim sredstvima

Prioritetna pravila (engl. *priority rules*) ili pravila raspoređivanja (engl. *dispatching rules*) su pravila koja se koriste prilikom raspoređivanja u dinamičkim okruženjima. Prioritetna pravila koriste se u kombinaciji sa shemom za generiranje rasporeda, koja na osnovu prioriteta određenih ovim pravilima, kroz iteracije, od praznog rasporeda gradi raspored u kojem su zadovoljeni svi uvjeti danog problema. Kreiranje ovih pravila je težak i zahtjevan posao, te zahtijeva postojanje stručnjaka koji dovoljno poznaje problem kako bi mogao odrediti na temelju kojih svojstava je potrebno određivati prioritete. Osim toga, za svaki kriterij koji se želi optimizirati potrebno je novo pravilo. Općenito, razvoj prioritetnih pravila podrazumijeva dugo ispitivanje i isprobavanje brojnih verzija pravila dok se ne pronađe ona odgovarajuća. Kako bi se automatizirao ovaj postupak koriste se brojne optimizacijske metode i metode strojnog učenja. Najčešće korištena metoda za automatizirani razvoj prioritetnih pravila je GP. GP je sposoban od jednostavnih struktura izgraditi kompleksne strukture i temeljem skupa za učenje razviti pravilo koje ima generalizacijsku sposobnost. Također, na pravila oblikovana GP-om mogu se primijeniti različite preinake i metode strojnog učenja koja će ta pravila dodatno poboljšati.

U ovom poglavlju, prvo će biti dan pregled literature u kojoj je GP korišten za razvoj pravila raspoređivanja, s naglaskom na radove u kojima se razvijaju prioritetna pravila za RCPSP. Zatim, opisat će se bitni koraci u oblikovanju prioritetnih pravila GP-om. Nakon toga, napravit će se detaljna analiza skupa značajki i funkcija koji će se koristiti prilikom razvoja prioritetnih pravila, kao i pri tom korištenih skupova za učenje i provjeru. Na temelju ispitivanja, napravit će se odabir parametara bitnih za pokretanje GP-a. Na kraju bit će prikazani rezultati koje razvijena pravila ostvaruju te će se njihova kvaliteta usporediti s kvalitetom postojećih prioritetnih pravila.

4.1 Pregled literature

U literaturi se GP pokazuje kao dobar izbor za hiperheuristiku [84, 85] koja razvija pravila za raspoređivanje, odnosno prioritetna pravila [25, 26]. U radu [27] GP je prvi put upotrebljen u svrhu automatiziranog razvoja pravila raspoređivanja i to za okruženje jednog stroja. Pravila razvijena GP-om postigla su bolje rezultate od tada postojećih pravila. Nakon toga, u literaturi se pojavljuju radovi i za pravila raspoređivanja u drugim, složenijim, okruženjima. Tako pronalazimo razvoj pravila GP-om za raspoređivanje na više strojeva [28], u okruženju proizvoljne obrade [29, 30, 31] i u okruženju nesrodnih strojeva [32]. Pregled korištenja GP-a u raspoređivanju pronalazimo u radu [23]. Osim samih primjena GP-a u razvoju pravila raspoređivanja, radovi [23, 30] analiziraju i njihov prikaz koji može bitno utjecati na kvalitetu dobivenih rješenja. Rad [33] nam donosi usporedbu rezultata postignutih pravilima razvijenim GP-om i postojećim, ručno definiranim, pravilima.

Iako su pravila raspoređivanja uglavnom namijenjena za dinamička okruženja, gdje je potrebno brzo reagirati na promjene, zbog svoje jednostavnosti i brzine ponekad se primjenjuju i u statičkim uvjetima kada je brzina pronalaska rješenja bitnija od kvalitete. S obzirom da u tim slučajevima raspolažemo s više informacija nego kod dinamičkih okruženja, moguće su dodatne prilagodbe koje mogu poboljšati kvalitetu rješenja. Jedan od postupka prilagodbe statičkim okruženjima su i iterativna pravila raspoređivanja koja pronalazimo u radu [49].

Iako se GP u razvoju pravila raspoređivanja intenzivno primjenjuje preko 15 godina i pri tome postiže dobre rezultate, njegova primjena u RCPSP-u se tek počinje otkrivati. Prva primjena GP-a u literaturi zabilježena je u radu [34] gdje se uspoređuju GA i GP, pri čemu se kod GP-a koristi vrlo mali broj značajki i funkcija za razvoj pravila. Rezultati postignuti pravilima razvijenim GP-om, očekivano, pokazuju lošije rezultate od GA, no ističe se važnost takvog pristupa za korištenje u dinamičkim okruženjima. U narednih 11 godina, u literaturi se ne pronalazi niti jedan rad koji bi bio nastavak istraživanja započetog u radu [34], kada se, gotovo istovremeno, pojavljuju dva rada koji se bave tom tematikom [35, 36].

U oba rada, skup značajki i funkcija je dodatno proširen u odnosu na skup iz rada [34]. U radu [35] skup funkcija sastoji se od aritmetičkih funkcija i dodatne 4 funkcije, pri čemu je jedna od njih funkcija IF, odnosno uvjet grananja. Skup značajki dijeli se na one statičke i dinamičke, i u svakom od njih razlikuju se one specifične za projekt ili specifične za pojedinu aktivnost. Ukupan broj promatranih značajki u radu je 27. Osim standardnih shema za generiranje rasporeda, u radu [35] uvodi se i prilagođena usporedna shema za generiranje rasporeda, s ciljem postizanja aktivnog rasporeda. Kriterij optimizacije odabran je na način da se težina same instance problema uzima u obzir. Razvijena prioritetna pravila ostvarila su bolje rezultate nego postojeća ručno definirana prioritetna pravila.

Rad [36] skup značajki smanjuje na 19 značajki, pri čemu je 10 istih kao u [35], te 9 novo

uvedenih. Skup značajki podijeljen je na one koje su vezane uz aktivnosti, uz instance problema te sustav i skup dostupnih aktivnosti. Sve vrijednosti značajki su normalizirane, odnosno njihove vrijednosti nalaze se između 0 i 1. Skup funkcija sadrži aritmetičke operacije, funkciju minimuma i maksimuma između dva argumenta i funkciju koja vraća vrijednost argumenta pomnoženu s (-1). U radu su proučavana dva prikaza rješenja: aritmetički prikaz i mješoviti prikaz. Mješoviti prikaz osim aritmetičkih pravila sadrži i pravila u obliku stabla odlučivanja. Kao kriterij optimizacije korištena je funkcija prosječnog odstupanja od donje granice na instancu problema koja se dobije ispuštanjem uvjeta na sredstva. Rezultati postignuti pravilima razvijenim u ovom radu bolji su od ručno definiranih prioritetnih pravila, a aritmetički prikaz postiže nešto bolje rezultate od mješovitog prikaza.

Nastavak rada [36] nalazi se u disertaciji [37] u kojoj se pronalaze dodatne prilagodbe GP-a za RCPSP. Jedna od njih je korištenje pristupa rollout i raspoređivanje u oba smjera (od nultog vremenskog trenutka, i od zadnjeg trenutka u planiranju projekta). Pristup rollout podrazumijeva iterativni postupak raspoređivanja i kao takav prigodan je isključivo za statička okruženja. Ovim načinom raspoređivanja postižu se bolji rezultati nego korištenjem samo osnovne sheme za generiranje rasporeda. Rezultati postignuti ovim pristupom kompetitivni su s metaheurističkim pristupima za rješavanje RCPSP-a, no glavni nedostatak im je potrebno vrijeme izvršavanja koje onemogućuje korištenje većih populacija u razvoju pravila, te zahtijevaju manji broj iteracija samog GP-a što bitno utječe na kvalitetu rješenja.

Kao drugi doprinos, a koji ne pronalazimo u radu [36], u [37] pronalazimo upotrebu GP-a u razvoju pravila za sustav s dinamičkim promjenama, odnosno sustav u kojem su prisutni određeni dinamički prekidi. Nakon prekida, moguće su dvije verzije, prva u kojoj se aktivnost mora izvršavati od početka i druga u kojoj je moguće nastaviti sa započetim izvršavanjem. Osim dinamičkih promjena, u obzir je uzeta i kompleksnost samog razvijenog pravila, odnosno koristila se višekriterijska optimizacija, gdje je uz osnovni kriterij korištena i kompleksnost rješenja. Dobiveni rezultati pokazuju da tako razvijena pravila daju bolji rezultat od klasičnih ručno definiranih pravila pri čemu njihova kompleksnost nije puno veća od kompleksnosti ručno definiranih pravila.

4.2 Oblikovanje prioritetnih pravila koristeći genetičko programiranje

Za oblikovanje prioritetnih pravila, u ovoj disertaciji, koristit će se GP. Zahvaljujući GP-u razvoj prioritetnih pravila je automatiziran, zbog čega je lako razviti pravila za različite kriterije optimizacije. Kako bi razvijena pravila bila kvalitetna potrebno je odabrati dobar skup primitiva.

GP je tehnika strojnog učenja pa mu je potrebno osigurati skup za učenje (engl. *learning set*), skup za provjeru (engl. *validation set*) i ispitni skup (engl. *test set*) koji dobro reprezen-

tiraju problem. Ovi skupovi nužno su međusobno disjunktni, odnosno ako se neka instanca problema nalazi u jednom od skupova ne može se nalaziti u preostala dva. Skup za učenje koristi se prilikom razvoja pravila, dok skup za provjeru osigurava da ne dođe do prenaučenosti pravila. Način na koji se sprječava prenaučenost je da se, nakon što GP zbog ispunjenosti unaprijed određenog kriterija zaustavljanja završi s razvojem pravila, najbolje pravilo unutar populacije rješenja odabire prema dobroti pravila na skupu za provjeru. Osim toga skup za provjeru koristi se i kod odabira kriterija zaustavljanja, što će se dodatno objasniti u narednim poglavljima. Kvaliteta prioritetnog pravila se na kraju određuje koristeći ispitni skup u kojem se nalaze instance problema koje nisu korištene u razvoju pravila. Ako ovi skupovi, po svojim karakteristikama, budu različiti od skupova na kojima će se razvijeno pravilo kasnije primjenjivati, kvaliteta rješenja može se smanjiti.

Sam proces razvoja pravila je dugotrajniji nego pronalazak rješenja za konkretnu instancu problema metaheuristikom, no nakon što je pravilo razvijeno, ono se može upotrijebiti i na drugim instancama problema, dok metaheuristički pristupi zahtijevaju ponovno pokretanje algoritma. S obzirom da se nakon razvoja pravila raspored s pomoću njega kreira u vrlo kratkom vremenu, a pravilo je moguće razviti prije početka rješavanja, prioritetna pravila pogodna su za primjenu u dinamičkim okruženjima.

4.2.1 Skup primitiva

GP iz jednostavnijih struktura može stvoriti složenije te je pogodan za oblikovanje prioritetnih pravila. Za oblikovanje dobrog pravila potrebno je da jednostavne strukture iz kojih se pravilo razvija budu takve da se njihovim korištenjem može dobro reprezentirati problem. Ako se u skupu primitiva ne nalaze dobre značajke i funkcije za oblikovanje pravila, bez obzira na ostale parametre i operatore koji su dio GP-a, pravilo neće biti dobro. U skupu primitiva treba biti dovoljno onih značajki koje mogu opisati, ne samo instance u skupu za učenje, nego i one na koje će se kasnije primjenjivati razvijeno pravilo.

Na osnovu radova [35, 36] za skup funkcija odabrane su aritmetičke funkcije, funkcija minimuma i maksimuma, množenje s (-1) te funkcija apsolutne vrijednosti koje su prikazane u tablici 4.1. Zbog rezultata iz rada [36] koji pokazuju da se korištenjem samo aritmetičkih funkcija postižu bolja rješenja, što su potvrdila i provedena preliminarna istraživanja, u skupu funkcija se ne nalaze logičke funkcije, kao ni uvjeti grananja.

Skup značajki je domenski specifičniji od skupa funkcija. Za početni skup značajki uzeta je unija značajki koje se koriste u radovima [35] i [37]. Skup značajki prema [35] dijelimo na statičke i dinamičke, te unutar svake kategorije još ih dijelimo na značajke karakteristične za projekt i značajke karakteristične za aktivnost. Statičke značajke se ne mijenjaju tijekom izvršavanja projekta pa je njihovu vrijednost dovoljno izračunati jednom i to prije nego se aktivnosti počnu raspoređivati. Popis svih statičkih značajki dan je u tablici 4.2. Vrijednost dinamičkih

Tablica 4.1: Skup funkcija

Naziv funkcije	Definicija
+, -, *	zbrajanje, oduzimanje i množenje
/	zaštićeno dijeljenje: $DIV(a,b) = \begin{cases} 1, & \lvert b\rvert < 0.000000001 \\ \frac{a}{b}, & \text{inače.} \end{cases}$
MAX	$MAX(a,b) = \begin{cases} a, & a > b \\ b, & \text{inače.} \end{cases}$
MIN	$MIN(a,b) = \begin{cases} a, & a < b \\ b, & \text{inače.} \end{cases}$
APS	apsolutna vrijednost
NEG	$NEG(a) = (-1) \cdot a$

značajki mijenja se tijekom izvršavanja projekta, te ako su sastavni dio prioritetnog pravila, njihova vrijednost se mora ponovno računati kod svake iteracije. Popis svih dinamičkih značajki dan je tablicom 4.3. Značajke koje nisu obuhvaćene radom [35], a nalaze se u [37] dane su u tablici 4.4. Iz ovog osnovnog skupa značajki, u narednim poglavljima, izdvojit će se one za koje testiranja pokazuju da bitno utječu na kvalitetu rješenja te će se taj smanjeni skup koristiti za razvoj pravila.

4.2.2 Podaci

Skup podataka koji će se koristiti prilikom testiranja dolazi iz biblioteke za problem raspoređivanja (engl. *project scheduling problem library*, skraćeno PSPLIB) [114]. PSPLIB sadrži različite instance RCPSP-a kao i njihova optimalna i najbolja poznata rješenja. Instance problema su generirane ProGen-om o čemu se više može pronaći u [115, 116].

S obzirom na broj aktivnosti koje se nalaze unutar projekta, instance problema unutar PSPLIB biblioteke podijeljene su u 4 grupe: na projekte koji imaju $30, 60, 90$ ili 120 aktivnosti, pri čemu se u obzir ne uzimaju fiktivne aktivnosti. Za generiranje instanci, neovisno o broju aktivnosti, koriste se tri skupine parametara. Prvu skupinu parametara čine fiksirani parametri za svaku grupu, drugu čine parametri ovisni o grupi i treću skupinu čine parametri koji se sustavno mijenjaju u svakoj grupi. Parametri koji se nalaze u trećoj skupini i koji se mijenjaju prilikom generiranja su kompleksnost mreže (NC), faktor sredstva (RF) i jakost sredstva (RS_r) o kojima se detaljnije može pronaći u poglavlju 2.2.2.

Za generiranje instanci problema s 30, 60 i 90 aktivnosti korištene su vrijednosti parametara prikazane u tablici 4.5, a za instance problema sa 120 aktivnosti vrijednosti prikazane u tablici

Tablica 4.2: Skup statičkih značajki

Kategorija	Značajka	Opis
Karakteristične projektu	RF	faktor sredstva
	RS	jakost sredstva
	TNA	broj svih aktivnosti (bez fiktivnih)
	TD	ukupno trajanje projekta
Karakteristične aktivnosti	D	trajanje aktivnosti
	RR	broj potrebnih sredstava
	RRT	RR pomnožen s potrebnom količinom pojedinog sredstva
	ARU	prosječna upotreba sredstva
	DPC	broj direktnih prethodnika
	DSC	broj direktnih sljedbenika
	TPC	ukupan broj prethodnika
	TSC	ukupan broj sljedbenika
	SPC	broj razina u stablu prethodnika
	SSC	broj razina u stablu sljedbenika
	GRPW*	najveći težinski pozicijski rang sljedbenika
	ES	najraniji trenutak početka aktivnosti
	EF	najraniji trenutak završetka aktivnosti
	LS	najkasniji trenutak početka aktivnosti
	LF	najkasniji trenutak završetka aktivnosti

Tablica 4.3: Skup dinamičkih značajki

Kategorija	Značajka	Opis
Karakteristične projektu	NUA	broj neizvršenih aktivnosti
	NAA	broj trenutno aktivnih aktivnosti
	NPA	broj izvršenih aktivnosti
	SUD	zbroj vremenskog trajanja neizvršenih aktivnosti
	SAD	zbroj vremenskog trajanja trenutno aktivnih aktivnosti
	SPD	zbroj vremenskog trajanja izvršenih aktivnosti
Karakteristične aktivnosti	NSP	broj raspoređenih prethodnika
	SL	vrijeme čekanja

Tablica 4.4: Dodatne značajke

Značajka	Opis
MinRReq	minimalna količina sredstva potrebnog za izvršenje aktivnosti
MaxRReq	maksimalna količina sredstva potrebnog za izvršenje aktivnosti
AvgRA	prosječna dostupnost sredstava od trenutka gledanja do planiranog vremena završetka projekta
MinRA	najmanja prosječna dostupnost sredstva od trenutka gledanja do planiranog vremena završetka projekta
MaxRA	najveća prosječna dostupnost sredstva od trenutka gledanja do planiranog vremena završetka projekta
AvgRF	prosječni broj potrebnih sredstava za aktivnosti unutar skupa dostupnih aktivnosti
AvgRU	prosječna količina potrebnih sredstava za aktivnosti unutar skupa dostupnih aktivnosti
RS	trenutni nedostatak sredstva
RC	potreba za sredstvom u odnosu na dostupnost sredstva

Tablica 4.5: Promjenjivi parametri - 30, 60, 90 aktivnosti

Parametar	Vrijednosti			
NC	1.50	1.80	2.10	
RF	0.25	0.50	0.75	1.00
RS_r	0.20	0.50	0.70	1.00

Tablica 4.6: Promjenjivi parametri - 120 aktivnosti

Parametar	Vrijednosti				
NC	1.50	1.80	2.10		
RF	0.25	0.50	0.75	1.00	
RS_r	0.10	0.20	0.30	0.40	0.50

4.6. Podsjetimo se, kompleksnost problema pada s porastom vrijednosti kompleksnosti mreže i jakosti sredstva, a raste s porastom vrijednosti faktora sredstva.

U PSPLIB biblioteci za probleme s 30, 60 i 90 aktivnosti ima $3 \cdot 4 \cdot 4 = 48$ kombinacija na osnovu parametara iz tablice i za svaku kombinaciju generirano je 10 instanci, dok za probleme sa 120 aktivnosti ima $3 \cdot 4 \cdot 5 = 60$ kombinacija i za svaku je generirano 10 instanci što ukupno čini 2040 instanci problema različitih težina.

Kako bi GP oblikovao kvalitetno prioritetno pravilo, potrebno je formirati dobar skup za učenje. Skup za učenje treba biti sastavljen od instanci problema koje dobro predstavljaju probleme koje će razvijeno prioritetno pravilo kasnije rješavati. Ako u skupu za učenje imamo probleme čije su karakteristike slične onima na koje će se pravilo kasnije primjenjivati, pravilo će biti dobre kvalitete. Osim odabira skupa za učenje, bitno je odabrati i dobre kriterije zaustavljanja, jer ako prioritetno pravilo ne bude dovoljno dugo "učilo" na skupu za učenje, bez obzira na kvalitetu skupa za učenje, rješenje neće biti dobro. S druge strane, ako pravilo bude predugo "učilo", može doći do prenaučenosti, odnosno prioritetno pravilo može se previše prilagoditi skupu za učenje što će smanjiti kvalitetu rješenja koje će postizati na novim instancama problema.

Kao odgovor na ovaj problem uvodi se skup za provjeru. Skup za provjeru sastoji se od instanci problema koje se ne nalaze u skupu za učenje, i na kojem se mjeri kvaliteta postignutog rješenja. U početku, kvaliteta rješenja, mjerena na skupu za provjeru, raste kroz iteracije, da bi se u jednom trenutku njezina kvaliteta počela smanjivati. Trenutak u kojem počinje smanjenje kvalitete rješenja je trenutak u kojem počinje prenaučenost i u kojem je potrebno zaustaviti proces učenja.

Osim ova dva skupa, uobičajeno je koristiti i treći skup kojeg nazivamo ispitni skup. Ispitni skup sastoji se od novih instanci problema, koje se ne nalaze u skupovima za učenje i provjeru, te služi kao skup na kojima se mjeri stvarna kvaliteta rješenja i generalizacijska sposobnost prioritetnog pravila.

Prilikom testiranja provedenih s ciljem određivanja početnih vrijednosti parametara, korišten je isti skup za učenje kao i u [35]. Taj skup sastoji se od 56 instanci problema, među kojima se nalaze instance iz svih grupa, gledano s obzirom na broj aktivnosti unutar projekta. Za razliku od rada [35], gdje nije korišten skup za provjeru, u ovim testiranjima on će biti korišten, a njegova upotreba opravdana. Skup za provjeru izdvojen je iz ispitnog skupa korištenog u [35] te se sastoji od 204 instance, pa za ispitni skup preostaje 1780 instanci.

4.2.3 Kriteriji optimizacije

Kriteriji optimizacije kod raspoređivanja mogu biti različiti, a najzastupljeniji u literaturi su navedeni u poglavlju 2.2.3. Za oblikovanje prioritetnih pravila u ovoj disertaciji koristit će se kriteriji iz radova [35, 36], pri čemu se kod oba kriterija radi o minimizaciji.

U radu [35] za kriterij optimizacije predlaže se funkcija koja će uzeti u obzir težinu same instance problema. Autori rada ističu kako je dobro, s obzirom da se instance razlikuju po broju aktivnosti unutar projekta, ali i po težini, pronaći kriterij optimizacije koji će omogućiti da različito teški problemi manje ili više pridonose funkciji cilja ovisno o svojoj težini. Kako bi se GP usredotočio i na one probleme koji su teži, vrijednost postignuta na njima ima veći utjecaj na dobrotu prioritetnog pravila. Dobrota prioritetnog pravila na i-toj instanci problema računa se sljedećim izrazom:

$$f_i = \frac{C_i}{p_i^{avg} \cdot \sqrt{n_i}}, \tag{4.1}$$

pri čemu je C_i vrijeme završetka i-tog projekta, p_i^{avg} prosječno trajanje aktivnosti unutar projekta i n_i broj aktivnosti unutar projekta. Kako se prioritetno pravilo ne razvija na samo jednoj instanci problema nego na skupu instanci, ukupna funkcija cilja je dana izrazom:

$$F_1 = \frac{\sum_{i=1}^{N} f_i}{N} = \frac{\sum_{i=1}^{N} \frac{C_i}{p_i^{avg} \cdot \sqrt{n_i}}}{N}, \tag{4.2}$$

gdje je N broj instanci problema unutar skupa koji se koristi prilikom evaluacije. U gornjem izrazu može se uočiti da se jedino C_i mijenja kroz iteracije dok su ostale vrijednosti fiksne, što znači da smanjenje vrijednosti C_i vodi k smanjenju vrijednosti funkcije cilja, odnosno povećanju dobrote razvijenog prioritetnog pravila.

Kriterij optimizacije korišten u [36] temelji se na donjoj granici vremena izvršenja projekta koja se računa korištenjem uvjeta prednosti izostavljajući pri tome uvjete na sredstva. Funkcija

cilja predstavlja prosječno odstupanje vremena završetka projekta od donje granice svih instanci unutar skupa koji se koristi za evaluaciju te je izražena u postotcima. Formula za izračun vrijednosti funkcije cilja dana je sljedećim izrazom:

$$F_2 = \frac{1}{N}\left(\sum_{i=1}^{N}\left(\frac{C_i - L_i}{L_i}\cdot 100\right)\right), \tag{4.3}$$

pri čemu je L_i donja granica vremena završetka za instancu i.

4.2.4 Prilagođena shema za generiranje rasporeda

Prioritetna pravila koriste se u kombinaciji sa shemom za generiranje rasporeda (SGS). SGS na osnovu prioriteta dobivenih prioritetnim pravilima određuje koju aktivnost sljedeću staviti u raspored. Postoje dvije osnovne vrste SGS-a: slijedna (SSGS) i usporedna (PSGS) koje su detaljno objašnjene u poglavlju 2.4.1.

Prilikom oblikovanja pravila u ovoj disertaciji koristit će se tri različite verzije SGS-a po uzoru na [35]. Od tri korištene SGS verzije jedna je SSGS u svom osnovnom obliku, te su dvije prilagođene PSGS verzije. Kod obje verzije PSGS-a prilagođen je dio računanja skupa dostupnih aktivnosti, koji se u prilagođenoj verziji korištenoj u testiranjima određuje samo na temelju uvjeta prednosti, kao što se radi i kod SSGS-a, a ne i na temelju uvjeta na sredstva što je uobičajeno za PSGS.

Nakon što se izračuna skup dostupnih aktivnosti koje se mogu rasporediti u određenom vremenskom trenutku na osnovu uvjeta prednosti, prvo se uzima aktivnost s najvećim, odnosno najmanjim prioritetom, ovisno o vrsti prioritetnog pravila, i ako za nju postoji dovoljna količina sredstva ona se raspoređuje. Ako dostupna količina sredstva nije dovoljna za raspoređivanje aktivnosti s najvećim, odnosno najmanjim prioritetom, tada se u prvoj verziji (PSGS bez odgode) raspoređuje druga, po prioritetu izabrana, aktivnost, dok se u drugoj verziji (PSGS s odgodom) raspoređuju one aktivnosti koje ne uzrokuju odgodu aktivnosti s najvećim prioritetom te prelazi u vremenski trenutak u kojem će se aktivnost s najvećim prioritetom moći rasporediti. Uočimo da, iako se skup dostupnih aktivnosti računa na drugačiji način nego kod osnovne verzije PSGS-a, PSGS bez odgode rezultira istovjetnim rasporedom kao i osnovna verzija PSGS-a. Ideja za prilagođeni PSGS dolazi iz činjenice da se unutar rasporeda bez odgode nužno ne nalaze optimalni rasporedi, pa prisilnim stvaranjem rasporeda s odgodom postoji mogućnost dobivanja boljeg rezultata.

4.3 Oblikovanje eksperimenta

Za implementaciju hiperheurističkog pristupa oblikovanja prioritetnih pravila korišten je C++ programski jezik u kombinaciji s ECF-om [117].

4.3.1 Početni parametri

Kao i kod ostalih tehnika evolucijskog računarstva, i kod GP-a potrebno je pronaći dobre parametre za algoritam, kako bi razvijena pravila bila dobre kvalitete. Zbog velikog broja kombinacija parametara, njihove vrijednosti teško je automatizirano odrediti te njihovom određivanju uglavnom prethodi veliki broj testiranja.

Kako bi testiranja s pomoću kojih će se odrediti vrijednosti parametara mogla započeti, potrebno je odrediti inicijalne vrijednosti parametara i skup primitiva. Inicijalne vrijednosti parametara preuzete su iz [35] i dane su u tablici 4.7.

Tablica 4.7: Inicijalne vrijednosti parametara

parametar	vrijednost
Odabir jedinki	turnirski odabir, k=3
Maksimalna dubina stabla	7
Vjerojatnost mutacije	0.3
Veličina populacije	1000
Broj generacija	25
SGS	PSGS bez odgode

Prije određivanja vrijednosti parametara provest će se testiranje u kojem će se na skupu za učenje, skupu za provjeru i ispitnom skupu odrediti koliko normalizacija značajki utječe na rezultate, kao i utječe li korištenje skupa za provjeru na kvalitetu razvijenog pravila. Prilikom ispitivanja za funkciju cilja F_1 koristit će se značajke i funkcije iz rada [35], a za funkciju cilja F_2 iz rada [36] s obzirom da su to funkcije cilja koje su korištene u navedenim radovima.

Rezultati ostvareni pravilima razvijenim koristeći funkciju cilja F_1 vidljivi su u tablici 4.8 i na prikazu kutijastim dijagramima (engl. *boxplot*) koji se nalaze na slici 4.1. Uočavamo da najbolje rezultate postižu pravila razvijena koristeći skup za provjeru i čije značajke nisu normalizirane. Kako bi potvrdili postojanje statistički značajne razlike u ostvarenim rezultatima koristimo Mann - Whitney - Wilcoxonov (MWW) test. MWW test je neparametarski test koji se koristi za uspoređivanje medijana dvije populacije i to najčešće kada nije zadovoljena pretpostavka na normalnu distribuiranost uzoraka pa se ne može koristiti t-test. MWW test pokazuje da postoji statistički značajna razlika (p-vrijednost manja od 0.01) između rezultata ostvarenih

s korištenjem skupa za provjeru i bez njegovog korištenja, bez obzira koriste li se normalizirane značajke ili ne, i ta razlika ide u korist pravila razvijenih s korištenjem skupa za provjeru. Također, MWW pokazuje da pravila razvijena sa značajkama koje nisu normalizirane postižu statistički značajno bolje rezultate (p-vrijednost manja od 0.01).

Tablica 4.8: Vrijednosti funkcije F_1 na ispitnom skupu

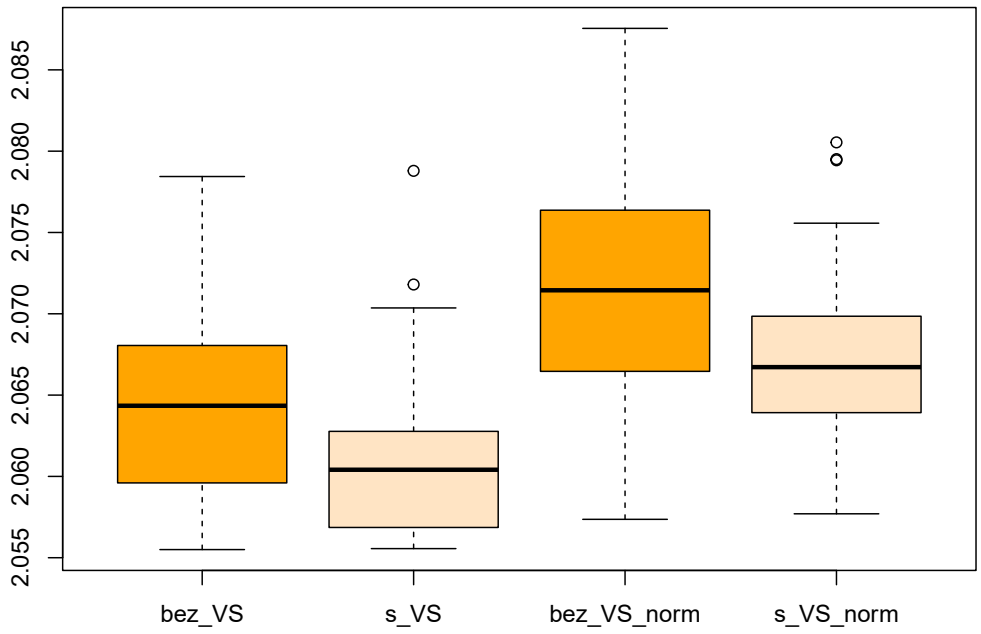
	bez_VS	s_VS	bez_VS_norm	s_VS_norm
korišten skup za provjeru	ne	da	ne	da
normalizacija	ne	ne	da	da
min	2.05550	2.05556	2.05736	2.05770
med	2.06435	2.06041	2.07145	2.06672
avg	2.06392	2.06126	2.07187	2.06717
max	2.07844	2.07879	2.08755	2.08054
stdev	0.00536	0.00489	0.00645	0.00536

Slično se događa kad se koristi funkcija F_2 što je vidljivo u tablici 4.9 i na slici 4.2 na kojoj su dani kutijasti dijagrami postignutih rezultata. Statistički značajna razlika između testova u kojima se koristi skup za provjeru i onih u kojima se ne koristi je pokazana MWW testom, ali ovaj put s p-vrijednosti 0.05, dok je razlika između testova u kojima su značajke normalizirane i u onima gdje nisu još izraženija s p-vrijednosti manjom od 0.01. Činjenica da značajke nije potrebno normalizirati, odnosno nije potrebno paziti da njihove vrijednosti budu unutar istog raspona pokazuje da GP unutar sebe ima dovoljno mehanizama kojima sam može neutralizirati problem nejednakih vrijednosti značajki, i da je u tome uspješniji nego normalizacija, odnosno držanje vrijednosti značajki između 0 i 1.

Na osnovu gornjeg razmatranja i ostvarenih rezultata, potreba za korištenjem skupa za provjeru prilikom razvoja prioritetnih pravila je očita što je i uobičajeno kod metoda strojnog učenja. Osim toga, korištene značajke, u daljnjim testiranjima, neće biti normalizirane, jer korištenje normaliziranih značajki je računalno zahtjevnije od korištenja značajki bez normalizacije, a testovi pokazuju da pravila čije značajke nisu normalizirane ostvaruju bolje rezultate.

4.3.2 Skup primitiva

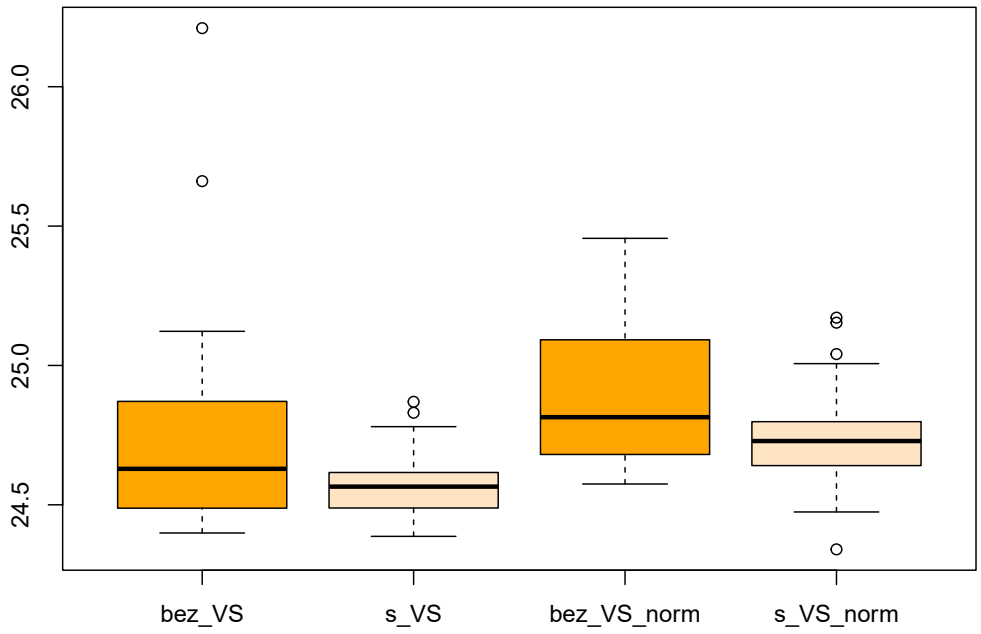
Već je više puta spomenuto koliko je odabir skupa primitiva važan, a s obzirom da je dosad za razvoj prioritetnih pravila za RCPSP korišteno 36 značajki, što prostor pretraživanja čini prilično velikim, provest će se dodatna ispitivanja kako bi se odlučilo koje značajke koristiti prilikom



Slika 4.1: Rezultati ostvareni na ispitnom skupu koristeći funkciju F_1

Tablica 4.9: Vrijednosti funkcije F_2 na ispitnom skupu

	bez_VS	s_VS	bez_VS_norm	s_VS_norm
korišten skup za provjeru	ne	da	ne	da
normalizacija	ne	ne	da	da
min	24.39887	24.38681	24.57470	24.34019
med	24.62913	24.56522	24.81439	24.72883
avg	24.73295	24.57233	24.89989	24.73988
max	26.21007	24.86964	25.45600	25.17148
stdev	0.37856	0.11429	0.25980	0.18393



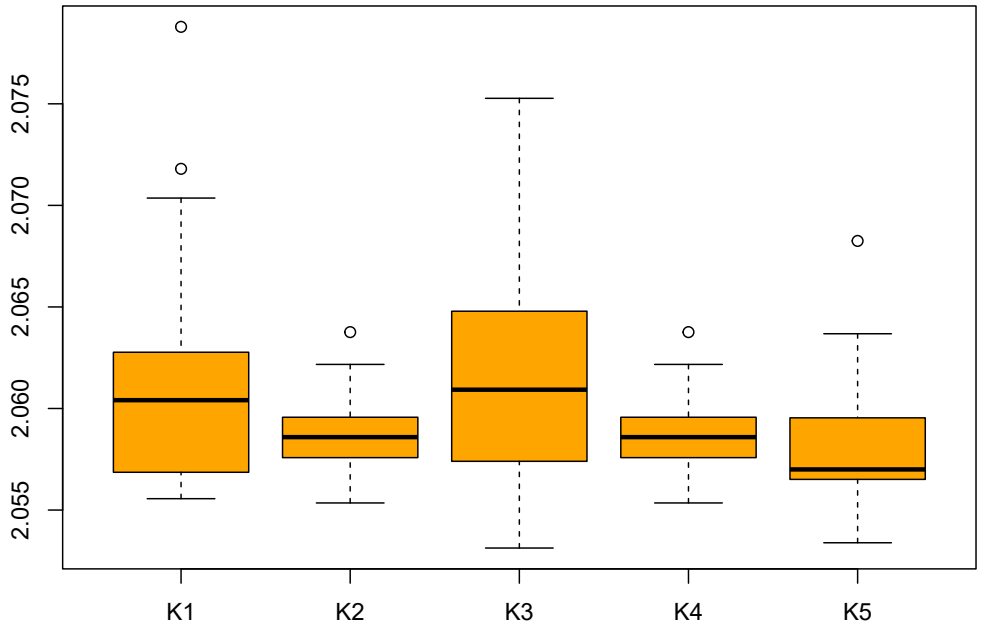
Slika 4.2: Rezultati ostvareni na ispitnom skupu koristeći funkciju F_2

razvoja pravila. U ovim testiranjima, kao i u ostalima u kojima će se određivati parametri GP-a kao funkcija cilja koristit će se funkcija F_1.

U testiranjima će se prvo ispitati različite kombinacije skupa funkcija i skupa značajki na osnovu radova [35] i [36]. Korištene kombinacije i postignuti rezultati dani su tablicom 4.10 i prikazani kutijastim dijagramom na slici 4.3. Uočavamo da rezultati postignuti pravilima u čijem su razvoju korištene funkcije iz rada [36] postižu bolje rezultate nego ako se koriste funkcije iz rada [35] bez obzira na korišteni skup značajki. S druge strane, uz isti korišteni skup funkcija, svejedno je koriste li se značajke iz rada [35] ili [36]. Korištenjem značajki iz jednog i drugog rada ostvaruju se lošiji rezultati zbog činjenice da veliki broj značajki utječe na povećanje prostora pretraživanja rješenja te unaprijed zadani broj generacija nije bio dovoljan kako bi se postiglo dobro rješenje. Zanimljivo je uočiti da ako skupu funkcija korištenom u [36] dodamo i funkciju apsolutne vrijednosti, ostvarit će se raspršeniji, ali statistički bolji rezultati. Na osnovu ovog razmatranja za skup funkcija koje će se koristiti odabran je skup dan tablicom 4.1, a kako između skupova značajki nema statistički značajne razlike, odabran je skup korišten u radu [35], odnosno dan tablicama 4.2 i 4.3.

Tablica 4.10: Vrijednosti postignute na ispitnom skupu koristeći različite kombinacije skupova funkcija i značajki

	K1	K2	K3	K4	K5
značajke	[35]	[36]	[35, 36]	[35]	[35]
funkcije	[35]	[36]	[35]	[36]	[36] i APS
min	2.05556	2.05535	2.05313	2.05440	2.05339
med	2.06041	2.05859	2.06093	2.05740	2.05700
avg	2.06126	2.05858	2.06165	2.05829	2.05789
max	2.07879	2.06376	2.07527	2.06571	2.06825
stdev	0.00489	0.00158	0.00488	0.00275	0.00284



Slika 4.3: Rezultati ostvareni na ispitnom skupu koristeći različite kombinacije skupova funkcija i značajki

4.3.3 Odabir značajki

Odabir značajki (engl. *feature selection*) je problem koji je u strojnom učenju često prisutan. Cilj ovog postupka je smanjiti broj značajki kako bi se izbacile redundantne informacije o problemu i na taj način smanjilo vrijeme potrebno za učenje, ali i povećala kvaliteta rješenja do koje dolazi zbog smanjenja prostora rješenja koje se pretražuje. Općenito, u literaturi možemo pronaći brojne postupke za odabir značajki [38, 39, 40, 41], a u ovom radu koristit će se pristup iz [42].

Mei i dr. [42] ističu da uklanjanjem pojedine značajke iz skupa značajki može doći do tri različite situacije: ako se značajka nije nalazila u pravilu, tada će pravilo ostati isto; ako ona nije značajna za problem tada će razvijeno pravilo postati jednostavnije, ali će kvaliteta ostati ista, te zadnje, ako se radi o značajnoj značajki tada će pravilo izgubiti na kvaliteti njezinim uklanjanjem. Iz ovog razmatranja, autori dolaze do ideje kako odabrati značajke. Za svaku značajku izračunat će se njezin doprinos, na način da će se ona ukloniti iz pravila, te će se na osnovu razlike u rezultatima kada je ona korištena i kada nije, znati koliko doprinosi samom pravilu. Značajka se iz pravila uklanja na način da se njezina stvarna vrijednost zamjenjuje s konstantom, odnosno s brojem 1. Uvođenjem konstante, pravilo se ne mijenja, nego s obzirom da konstanta jednako utječe na sve prioritete aktivnosti, dolazi samo do pomaka njihovih vrijednosti. Kako bi se odredio doprinos pojedine značajke, nije dovoljno postupak primijeniti na jednom pravilu, nego ga je potrebno ponoviti određeni broj puta te se, prema radu [42], kao referentna vrijednost uzima medijan ostvarenih rezultata. Korištenje medijana može se interpretirati na način da za značajku kažemo da je ona bitna, ako je njezin utjecaj vidljiv na više od pola razvijenih pravila. Kako bi ostvareni rezultati bili statistički značajni, u ovom radu za određivanje doprinosa značajki, korišteno je 50 prioritetnih pravila.

Rezultati dobiveni isključivanjem jedne po jedne značajke iz skupa značajki prikazani su u tablici 4.11. U stupcu medijan podebljani su oni rezultati čija je vrijednost veća od medijana rezultata ostvarenih prioritetnim pravilima kada su korištene sve značajke, a čije vrijednosti se nalaze u prvom redu tablice. Odabrane značajke su sljedeće: ARU, DPC, GRPW, LF, LS, NSP, NUA, TD i TNA. Možemo primijetiti da je ovim postupkom broj značajki smanjen s 27 na 9. Smanjenjem broja značajki značajno se smanjio i prostor rješenja. Pitanje koje se nameće je hoće li odabrane značajke biti dostatne za razvoj kvalitetnih prioritetnih pravila. S ciljem potvrde da je ovaj izbor značajki dovoljan za opis bitnih svojstava problema, ponovno je razvijeno 100 pravila u kojima su korištene sve značajke, i 100 pravila u kojima su korištene, ovim postupkom, odabrane značajke. Ostvareni rezultati prikazani su tablicom 4.12, iz koje je vidljivo da su prioritetna pravila razvijena korištenjem skupa odabranih značajki ostvarila bolje rezultate, što je i statistički potvrđeno MWW testom, čija je p-vrijednost manja od 0.01.

Tablica 4.11: Vrijednosti ostvarene isključivanjem značajki iz prioritetnih pravila

bez značajke	min	med	avg	max	stdev
	2.05339	2.05700	2.05789	2.06825	0.00284
ARU	2.05883	**2.06431**	2.06415	2.07267	0.00215
D	2.05359	2.05690	2.05781	2.06825	0.00269
DPC	2.05339	**2.05706**	2.05799	2.06825	0.00300
DSC	2.05339	2.05683	2.05771	2.06825	0.00294
EF	2.05339	2.05700	2.05786	2.06825	0.00281
ES	2.05339	2.05700	2.05791	2.06825	0.00286
GRPW	2.05378	**2.05706**	2.05896	2.07926	0.00509
LF	2.05378	**2.16280**	2.17564	2.34366	0.07639
LS	2.05359	**2.05732**	2.07452	2.33558	0.06397
NAA	2.05339	2.05700	2.05788	2.06825	0.00282
NPA	2.05339	2.05693	2.05783	2.06825	0.00283
NSP	2.05339	**2.05706**	2.05801	2.06858	0.00283
NUA	2.05339	**2.05715**	2.05849	2.06825	0.00331
RF	2.05339	2.05700	2.05793	2.06825	0.00295
RR	2.05341	2.05700	2.05795	2.06825	0.00285
RRT	2.05339	2.05683	2.05800	2.06596	0.00289
RS	2.05339	2.05700	2.05802	2.06825	0.00304
SAD	2.05339	2.05695	2.05858	2.09755	0.00632
SL	2.05339	2.05700	2.05789	2.06825	0.00284
SPC	2.05339	2.05700	2.05788	2.06825	0.00280
SPD	2.05339	2.05700	2.05787	2.06825	0.00280
SSC	2.05298	2.05700	2.05806	2.07443	0.00372
SUD	2.05339	2.05700	2.06078	2.20582	0.02111
TD	2.05339	**2.05791**	2.07818	2.34908	0.06045
TNA	2.05339	**2.05702**	2.05799	2.06825	0.00283
TPC	2.05339	2.05693	2.05787	2.06825	0.00283
TSC	2.05339	2.05691	2.05788	2.06825	0.00285

Tablica 4.12: Usporedba rezultata ostvarenih korištenjem svih značajki i odabranih značajki

	sve značajke	odabrane značajke
min	2.05339	2.05346
med	2.05713	2.05621
avg	2.05851	2.05700
max	2.07287	2.06836
stdev	0.00352	0.00251

4.3.4 Podjela skupa podataka

U prethodno provedenim testiranjima kao skup za učenje korišten je isti skup kao i u radu [35], a skup za provjeru dobiven je tako da se uzelo 204 instance problema iz ispitnog skupa korištenog u tom radu, te su ispitni skup bile preostale instance problema. No, u radu [35], kao ni u radu [36] nije napravljena nikakva detaljna analiza kakvi skupovi su dobri za razvoj prioritetnih pravila. U radu [35] ispitana su tek dva, po broju instanci, različita skupa za učenje, dok u radu [36] autori za skup učenja i provjeru odabiru instance problema s 30 i 60 aktivnosti ne potkrepljujući svoju odluku nekim dodatnim istraživanjem. U radu [118] autori ističu važnost da skup za učenje dobro opisuje i one probleme na kojima će se kasnije primjenjivati razvijena pravila, odnosno da skup za učenje ima sličnu distribuciju kao i skup problema na kojima će se pravilo primjenjivati. Ako pravila razvijamo na skupu koji se svojim brojem aktivnosti unutar projekta, trajanjem aktivnosti, faktorima sredstva i sl. drastično razlikuje od problema na kojima će se pravilo kasnije primjenjivati, generalizacijska sposobnost pravila ne mora biti dobra te ona mogu ostvarivati značajno lošije rezultate od onih koje su pokazivali na ispitnom skupu.

U provedenim ispitivanjima unutar ove disertacije, ispitat će se utječe li i, ako da, na koji način broj instanci u pojedinom skupu na ostvarene rezultate. Osim toga, ispitat će se je li dovoljno uzeti samo one instance problema u kojima se nalazi manji broj aktivnosti, što će smanjiti vrijeme potrebno za učenje, ili je bolje uzeti instance problema koje se međusobno razlikuju u broju aktivnosti. Uz ispitne skupove korištene u dosadašnjim testiranjima, ispitano je još 9 različitih kombinacija odabira skupa za učenje i skupa za provjeru. Njihovi rezultati ispitani su na istom ispitnom skupu koji se sastoji od 60% instanci iz PSPLIB-a, odnosno od 1224 instanci. U tablici 4.13 mogu se pronaći različite podjele skupa podataka, pri čemu se postotak odnosi na postotak ukupnog broja instanci u PSPLIB-u, a j30 i j60 označavaju instance problema s 30, odnosno 60 aktivnosti unutar projekta. Ako uz postotak nije istaknuto koliko aktivnosti ima unutar projekta, to znači da su instance birane ravnomjerno. Na primjer, 10% znači da je za svaku kombinaciju parametara uzeta jedna instanca problema (za svaku kombinaciju parametara u PSPLIB-u je generirano 10 instanci problema). Brojem 10 označena je kombinacija

dosad korištenog skupa za učenje i provjeru.

Tablica 4.13: Različite podjele skupa podataka

#Br	Skup za učenje		Skup za provjeru	
	Postotak	Broj instanci	Postotak	Broj instanci
S1	20%	408	10%	204
S2	30%	612	10%	204
S3	20% j30 i j60	192	10% j30 i j60	96
S4	10% j30 i j60	96	10% j30 i j60	96
S5	10% j30 i j60	96	20% j30 i j60	192
S6	10%	204	10%	204
S7	10%	204	20%	408
S8	10%	204	30%	612
S9	20%	408	20%	408
S10		56		204

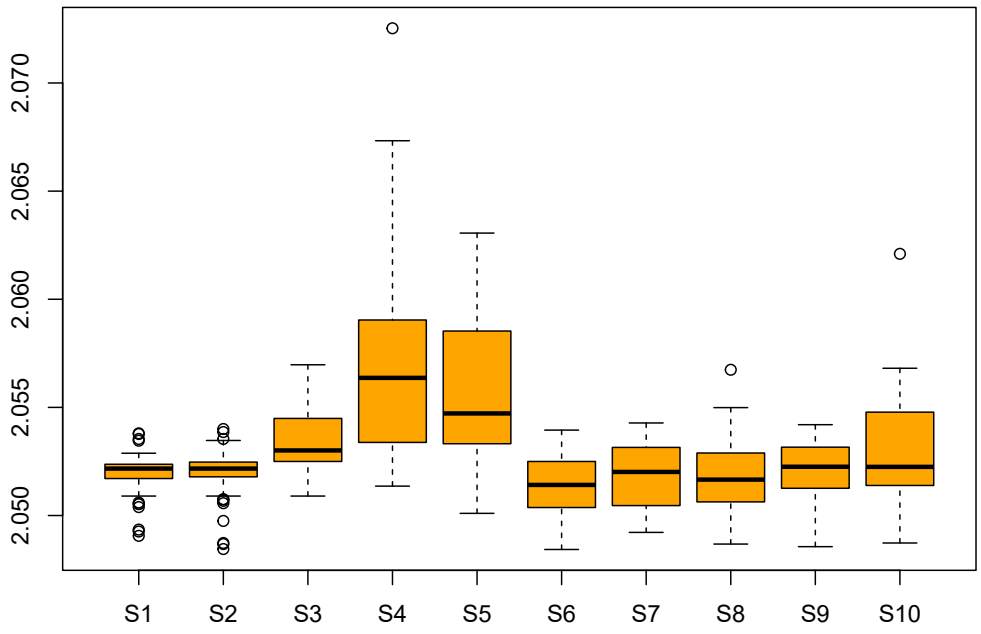
Rezultati koji se postižu korištenjem različitih skupova za učenje i provjeru dani su u tablici 4.14 i prikazani su kutijastim dijagramom na slici 4.4. Na dijagramu se lako uočava da pravila razvijena korištenjem skupova S3, S4 i S5 postižu najlošije rezultate, iz čega se zaključuje da korištenje samo projekata sastavljenih od 30 i 60 aktivnosti nije dovoljno. Isto tako, skupovi korišteni u prethodnim testiranjima nisu skupovi na kojima se postižu najbolji rezultati. Rezultati dobiveni korištenjem skupova S1 i S2 su najmanje raspršeni, što je i očekivano s obzirom na postotak instanci od kojih se sastoji skup za učenje. Zanimljivo je uočiti da, unatoč tome što se skupovi S1 i S2 sastoje od većeg broja instanci, medijani skupova S6, S7 i S8 su manji, što se možda može pripisati nedovoljnom broju generacija prilikom razvoja pravila. Na osnovu rezultata danih tablicom i dijagramom, odabrana je kombinacija S6, koja sadrži manji broj instanci i u skupu za učenje i u skupu za provjeru, a čiji se rezultati statistički značajno ne razlikuju od skupa S8.

4.3.5 Odabir parametara

Nakon odabira skupa funkcija i značajki te određivanja skupova za učenje i provjeru na kojima će se pravila razvijati, potrebno je odrediti i preostale parametre GP-a. Parametri čija će se vrijednost određivati su sljedeći: broj generacija, veličina populacije, vjerojatnost mutacije, maksimalna dubina stabla i veličina turnira u turnirskom odabiru.

Tablica 4.14: Postignuti rezultati korištenjem različitih kombinacija skupova za učenje i provjeru

#Br	min	med	avg	max	stdev
S1	2.04906	2.05217	2.05195	2.05380	0.00102
S2	2.04845	2.05217	2.05199	2.05400	0.00120
S3	2.05090	2.05301	2.05343	2.05697	0.00151
S4	2.05136	2.05637	2.05685	2.07253	0.00400
S5	2.05010	2.05472	2.05562	2.06306	0.00324
S6	2.04843	2.05142	2.05136	2.05395	0.00145
S7	2.04922	2.05202	2.05183	2.05428	0.00140
S8	2.04868	2.05166	2.05181	2.05674	0.00161
S9	2.04856	2.05226	2.05206	2.05420	0.00131
S10	2.04873	2.05225	2.05294	2.06210	0.00228



Slika 4.4: Postignuti rezultati korištenjem različitih kombinacija skupova za učenje i provjeru

Vrijednosti parametara kod evolucijskih tehnika najčešće se određuju na način da se vrijednost jednom od njih mijenja, dok su preostale vrijednosti fiksirane. Mijenjanjem vrijednosti jednog parametra dobivaju se novi rezultati, te se za vrijednost parametara odabire ona koja uz ostale fiksirane parametre postiže najbolje rezultate. Vrijednost parametara se određuje na skupu za provjeru, kako bi za krajnji rezultat imali ispitni skup koji nije sudjelovao u određivanju parametara, pa time i postignuti rezultat bio onaj koji se može očekivati za nove primjere.

Broj generacija

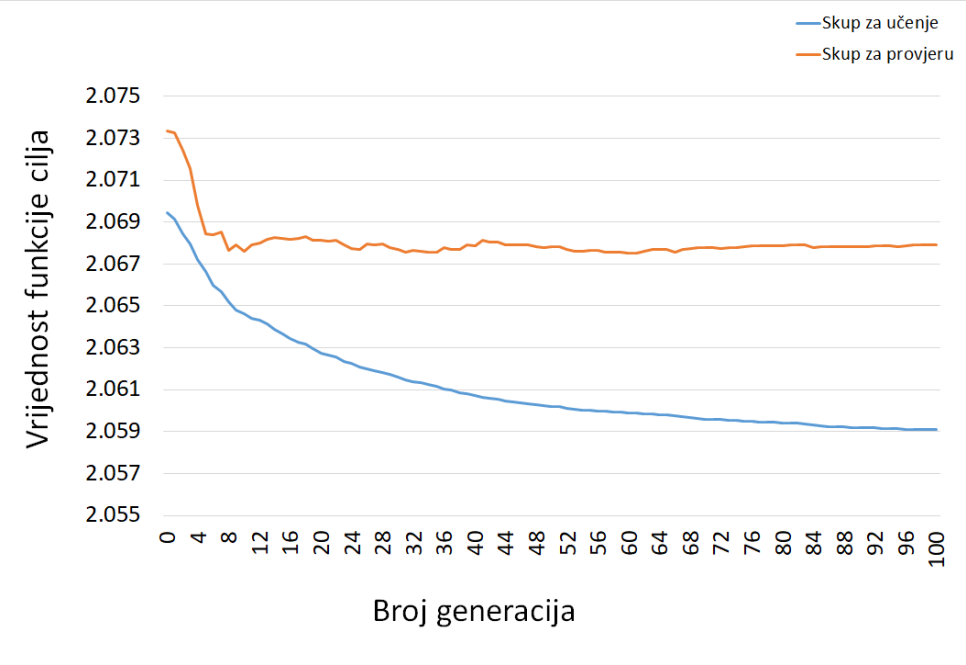
Kao kriterij zaustavljanja GP-a koristit će se maksimalni broj generacija. Za određivanje tog broja GP je pokrenut 50 puta i promatrano je što se događa s pravilima kroz generacije zaključno sa 100. generacijom. Prilikom učenja, uobičajeno je da se kroz generacije vrijednost funkcije cilja kod minimizacije smanjuje, odnosno da kroz generacije pravilo postaje sve bolje na skupu za učenje. S druge strane, što dulje pustimo GP da uči na nekom skupu, pravila se sve više prilagođavaju tom skupu, te može doći do prenaučenosti. Prenaučenost označava pojavu da pravilo ostvaruje dobre rezultate na skupu na kojem je pravilo naučeno, ali zbog prevelikog prilagođavanja tom skupu gubi generalizacijsku sposobnost koja mu omogućuje postizanje dobrih rezultata na dotad neviđenim instancama problema.

Kako bi odredili generaciju u kojoj bi GP trebao završiti, uz skup za učenje trebamo pratiti i skup za provjeru. Za razliku od ponašanja funkcije cilja na skupu za učenje, na kojem vrijednost funkcije cilja kroz generacije postaje sve manja, na skupu za provjeru ta vrijednost na početku pada, ali nakon nekog broja generacija počinje polako rasti. Trenutak u kojem se pojavljuje rast vrijednosti funkcije cilja na skupu za provjeru smatra se trenutkom u kojem počinje prenaučenost na skup za učenje. S obzirom na to, taj trenutak uzima se kao trenutak u kojem bi učenje trebalo stati, odnosno kada bi se algoritam trebao zaustaviti.

Na slici 4.5 dan je prikaz prosječnih vrijednosti funkcije cilja prioritetnih pravila kroz generacije GP-a. Na grafu je plavom bojom označeno ponašanje funkcije cilja na skupu za učenje, a narančastom na skupu za provjeru. Ono što možemo primijetiti je da u početku vrijednost funkcije cilja na skupa za provjeru naglo pada, da bi ta promjena postala sve manja te vrijednost malo rasla, pa malo padala. Negdje oko 25. generacije ta promjena postaje sve manje izražena. Nakon 25. generacije, funkcija cilja većinom raste. Iako je taj porast neznatan, uočavamo da daljnje učenje ne bi pridonijelo znatnom porastu kvalitete rješenja, a zahtijeva dodatno trošenje računalnih sredstava.

Pri odluci o maksimalnom broju generacija pomažu i vrijednosti postignute kroz generacije prikazane tablicom 4.15, i kutijastim dijagramom na slici 4.6. Iz tablice možemo uočiti da je vrijednost medijana u 25. generaciji najmanja postignuta u prvih 50 generacija. Iako je u 100. generaciji ta vrijednost još nešto manja, možemo vidjeti da je ta razlika neznatna, a broj generacija je drastično veći. Osim toga, ako pogledamo kutijaste dijagrame na slici 4.6 vidimo da su

vrijednosti ostvarene u 100. generaciji dosta raspršene, dok u 25. generaciji imamo postignutu dobru minimalnu vrijednost, a maksimalna vrijednost je značajno manja od onih postignutih u drugim generacijama. Zbog svih navedenih razloga, za maksimalni broj generacija odabran je 25.



Slika 4.5: Vrijednosti funkcije cilja F_1 na skupu za učenje i provjeru kroz generacije GP-a

Veličina populacije

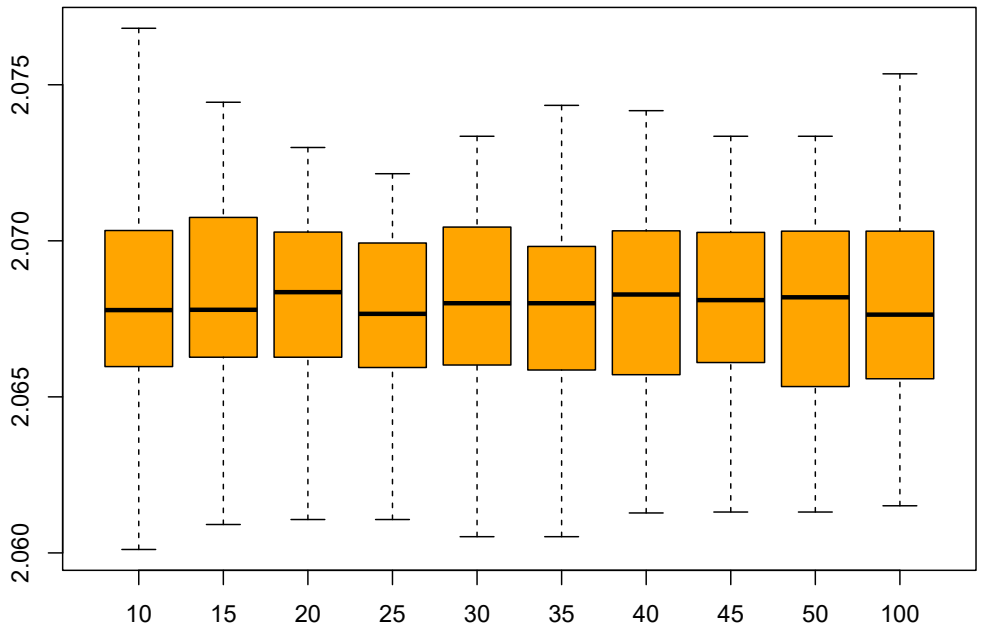
Za odabir veličine populacije, koristili smo pravila dobivena u 50 pokretanja za veličine populacija $250, 500, 750, 1000, 1024$ i 1500. Broj generacija za razvoj je 25 dok su ostali parametri kao u početnim testiranjima. Ostvareni rezultati prikazani su u tablici 4.16 i kutijastim dijagramima na slici 4.7. Razlog zbog čega su ispitani dosta bliski brojevi 1000 i 1024 je što su te veličine korištene u radovima [35, 36].

Iz tablice i dijagrama možemo primijetiti da se najmanji medijan postiže za veličinu od 750 jedinki. No, s druge strane, minimalna i maksimalna vrijednost su manje kod veličine 1024, od onih kod veličine 750, uz tek neznatno veći medijan. Iako je raspršenost rezultata kod veličine 1024 nešto veća, nego kod veličine 750, i dalje je veličina 1024 ta čije su vrijednosti niže. U skladu s tim, u daljnjim testiranjima veličina populacije bit će 1024. Zanimljivo je uočiti i to da je razlika između populacija sastavljenih od 1000 i 1024 jedinke prilično velika.

Dobro je uočiti da se postavljanjem maksimalnog broja generacija kao zaustavnog kriterija za različite veličine populacija ne koristi isti broj evaluacija, odnosno da je u tom slučaju broj evaluacija veći kod većih populacija. No, rezultati ostvareni postavljanjem maksimalnog broja evaluacija kao zaustavnog kriterija nisu bili bitno drugačiji te su iz tog razloga izostavljeni.

Tablica 4.15: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na broj generacija

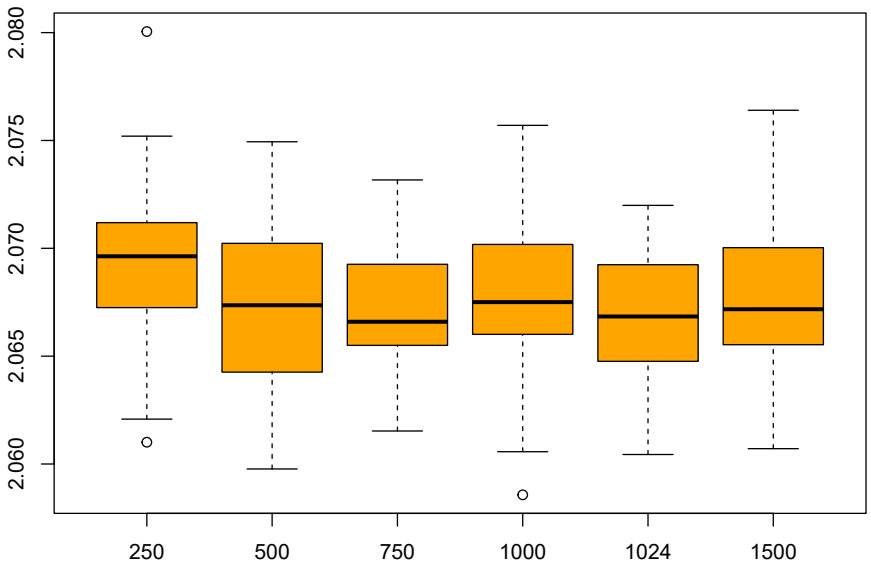
broj generacija	min	med	avg	max	stdev
10	2.06011	2.06778	2.06792	2.07681	0.00309
15	2.06091	2.06779	2.06824	2.07444	0.00304
20	2.06107	2.06836	2.06812	2.07299	0.00285
25	2.06107	2.06766	2.06774	2.07215	0.00285
30	2.06052	2.06800	2.06779	2.07335	0.00325
35	2.06052	2.06800	2.06755	2.07434	0.00320
40	2.06128	2.06828	2.06791	2.07417	0.00307
45	2.06131	2.06810	2.06793	2.07335	0.00301
50	2.06131	2.06819	2.06778	2.07335	0.00309
100	2.06151	2.06764	2.06791	2.07535	0.00315



Slika 4.6: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na broj generacija

Tablica 4.16: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na veličinu populacije

veličina populacije	250	500	750	1000	1024	1500
min	2.06101	2.05977	2.06153	2.05857	2.06044	2.06071
med	2.06963	2.06736	2.06660	2.06751	2.06684	2.06718
avg	2.06911	2.06712	2.06724	2.06770	2.06680	2.06733
max	2.08005	2.07494	2.07317	2.07570	2.07199	2.07640
stdev	0.00369	0.00387	0.00294	0.00320	0.00331	0.00348



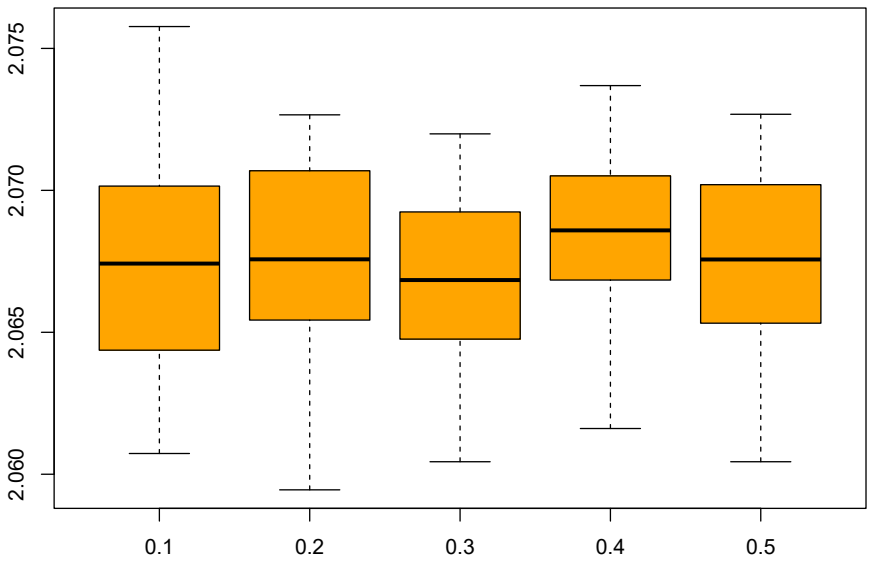
Slika 4.7: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na veličinu populacije

Vjerojatnost mutacije

Kao moguće vrijednosti parametra vjerojatnosti mutacije ispitane su sljedeće vrijednosti: 0.1, 0.2, 0.3, 0.4 i 0.5. GP je, kao i kod određivanja prethodnih parametara, pokrenut 50 puta za svaku od ovih vrijednosti. Prikaz rezultata ostvaren s obzirom na vjerojatnost mutacije dan je u tablici 4.17 i kutijastim dijagramom na slici 4.8. Iz prikazanih rezultata, kao vjerojatnost mutacije nameće se vrijednost 0.3 uz čiju se upotrebu postižu najmanje raspršeni rezultati, a medijan i maksimum su najmanji ostvareni u odnosu na one postignute korištenjem ostalih vrijednosti.

Tablica 4.17: Ostvareni rezultati na skupu za provjeru prioritetnih pravila s obzirom na vjerojatnost mutacije

vjerojatnost mutacije	0.1	0.2	0.3	0.4	0.5
min	2.06073	2.05945	2.06044	2.06161	2.06044
med	2.06742	2.06757	2.06684	2.06859	2.06757
avg	2.06715	2.06765	2.06680	2.06847	2.06756
max	2.07577	2.07266	2.07199	2.07369	2.07268
stdev	0.00352	0.00344	0.00331	0.00284	0.00342



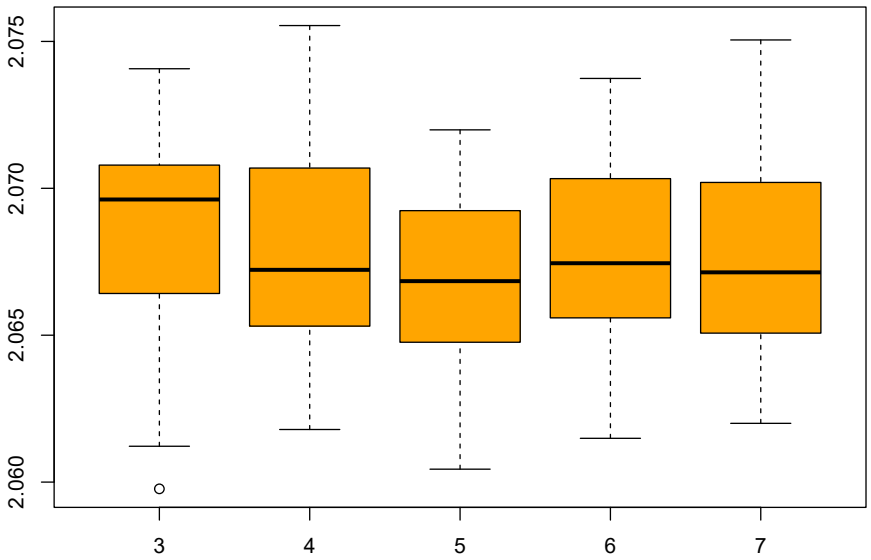
Slika 4.8: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na vjerojatnost mutacije

Maksimalna dubina stabla

Za maksimalnu dubinu stabla ispitane su vrijednosti 3, 4, 5, 6 i 7. Rezultati ostvareni u 50 pokretanja za svaku od tih vrijednosti dani su tablicom 4.18 i slikom 4.9. Iz rezultata je vidljivo da je za maksimalnu dubinu stabla jedinke najbolji izbor dubina 5.

Tablica 4.18: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na maksimalnu dubinu stabla

maksimalna dubina	3	4	5	6	7
min	2.05977	2.06179	2.06044	2.06149	2.06200
med	2.06962	2.06723	2.06684	2.06745	2.06714
avg	2.06893	2.06766	2.06680	2.06774	2.06756
max	2.07407	2.07554	2.07199	2.07374	2.07505
stdev	0.00292	0.00337	0.00331	0.00306	0.00333



Slika 4.9: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na maksimalnu dubinu stabla

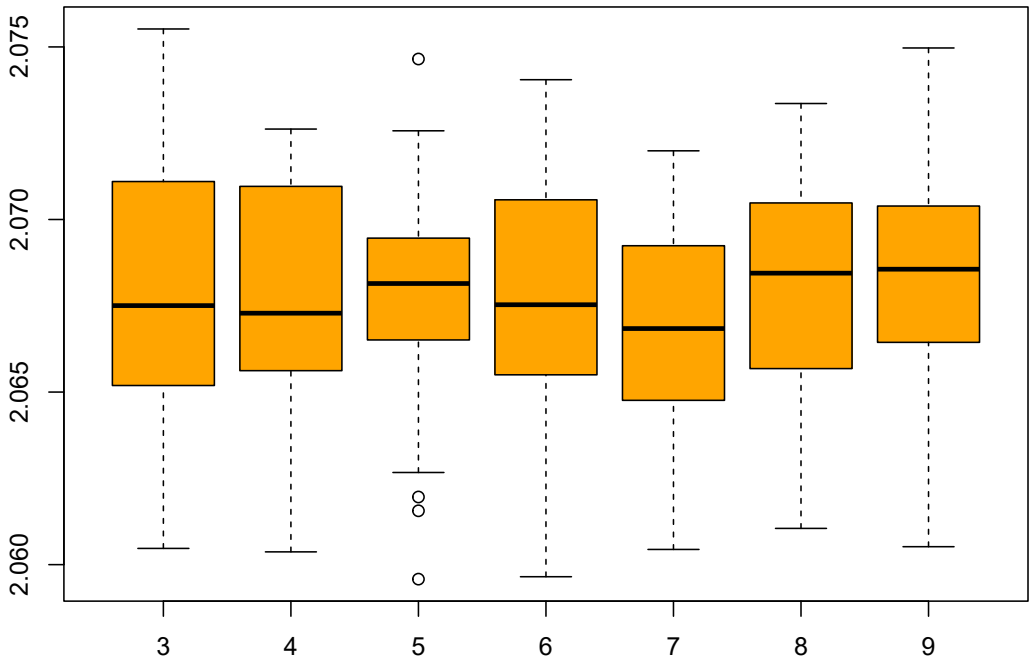
Veličina turnira u turnirskom odabiru

Zadnja vrijednost parametra koju određujemo je veličina turnira u turnirskom odabiru. Vrijednosti koje ćemo ispitati su u rasponu od 3 do 9. Za svaku od ovih vrijednosti GP je pokrenut

50 puta. Prikaz ostvarenih rezultata dan je tablicom 4.19 i slikom 4.10. Na osnovu rezultata, za veličinu turnir odabran je broj 7. Rezultati koje postižu pravila s veličinom turnira 7 u odnosu na one postignute s turnirom veličine 5 su raspršeniji, ali su vrijednosti i maksimuma i medijana manje.

Tablica 4.19: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na veličinu turnira u turnirskom odabiru

turnir	3	4	5	6	7	8	9
min	2.06047	2.06037	2.05958	2.05965	2.06044	2.06105	2.06052
med	2.06751	2.06729	2.06815	2.06753	2.06684	2.06845	2.06856
avg	2.06763	2.06772	2.06769	2.06778	2.06680	2.06780	2.06829
max	2.07552	2.07262	2.07465	2.07405	2.07199	2.07336	2.07497
stdev	0.00398	0.00340	0.00304	0.00327	0.00331	0.00335	0.00327



Slika 4.10: Ostvareni rezultati prioritetnih pravila na skupu za provjeru s obzirom na veličinu turnira u turnirskom odabiru

Konačno, sve odabrane vrijednosti za parametre koji će se koristiti prilikom oblikovanja prioritetnih pravila GP-om dane su tablicom 4.20.

Tablica 4.20: Odabrane vrijednosti parametara GP-a

parametar	vrijednost
broj generacija	25
veličina populacije	1024
vjerojatnost mutacije	0.3
maksimalna dubina stabla	5
veličina turnira	7

4.4 Rezultati

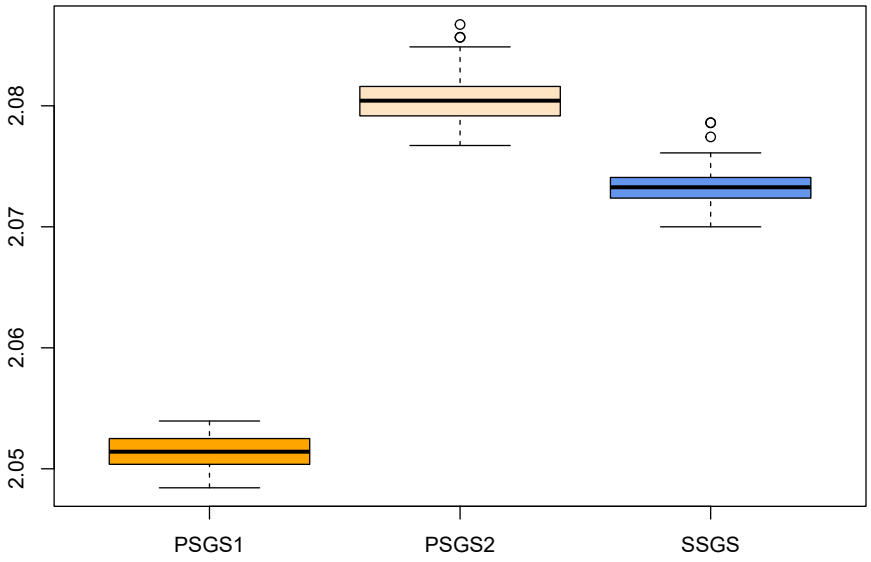
Nakon određivanja parametara, GP je pokrenut 50 puta za svaku od verzija SGS-a. U tablici 4.21 i na slici 4.11 prikazani su rezultati ostvareni na ispitnom skupu za funkciju cilja F_1, pri čemu u prikazanim rezultatima PSGS1 označava usporedni SGS bez odgode, odnosno u literaturi uobičajeni usporedni SGS, a PSGS2 usporedni SGS s odgodom.

U rezultatima možemo primijetiti da razvijena pravila korištenjem istog SGS-a postižu dosta slične vrijednosti, odnosno ostvareni rezultati za sve tri verzije SGS-a nisu previše raspršeni, što je vidljivo i na kutijastim dijagramima koji izgledaju spljošteno. Zanimljivo je uočiti da PSGS1 postiže znatno bolje rezultate od preostale dvije verzije te da nije potrebno u raspored uvoditi odgode kao što je to učinjeno u PSGS2 koji na kraju postiže rezultate lošije i od SSGS-a. Ovi rezultati još jednom potvrđuju pojavu iz prakse da unatoč činjenici da rasporedi nastali upotrebom PSGS1 ne moraju sadržavati optimalne rasporede, usporedni SGS postiže bolje rezultate od slijednog. Ova pojava je zanimljiva i zbog činjenice da je slijedni SGS računalno zahtjevniji od usporednog, a bez obzira na tu kompleksnost ne ostvaruje bolje rezultate.

Slični zaključci donose se i na osnovu rezultata za funkciju cilja F_2 danih u tablici 4.22 i slikom 4.12. Prikazani rezultati napravljeni su na temelju 50 pokretanja GP-a, odnosno 50 razvijenih pravila za sve tri verzije SGS-a. I kod funkcije F_2 korištenjem PSGS1 nastaju pravila najbolje kvalitete, dok su pravila razvijena korištenjem SSGS-a bolja od onih razvijenih korištenjem PSGS2. I kod ovog kriterija dijagrami su dosta spljošteni, odnosno razvijena prioritetna pravila su slične kvalitete u svih 50 pokretanja. Njihova raspršenost, odnosno standardna devijacija, nešto je veća od one kod funkcije F_1, što je i očekivano ako se uzme u obzir red veličine

Tablica 4.21: Ostvareni rezultati na ispitnom skupu s obzirom na korišteni SGS i funkciju F_1

SGS	PSGS1	PSGS2	SSGS
min	2.04843	2.07672	2.07000
med	2.05142	2.08043	2.07327
avg	2.05136	2.08079	2.07351
max	2.05395	2.08672	2.07860
stdev	0.00145	0.00223	0.00177

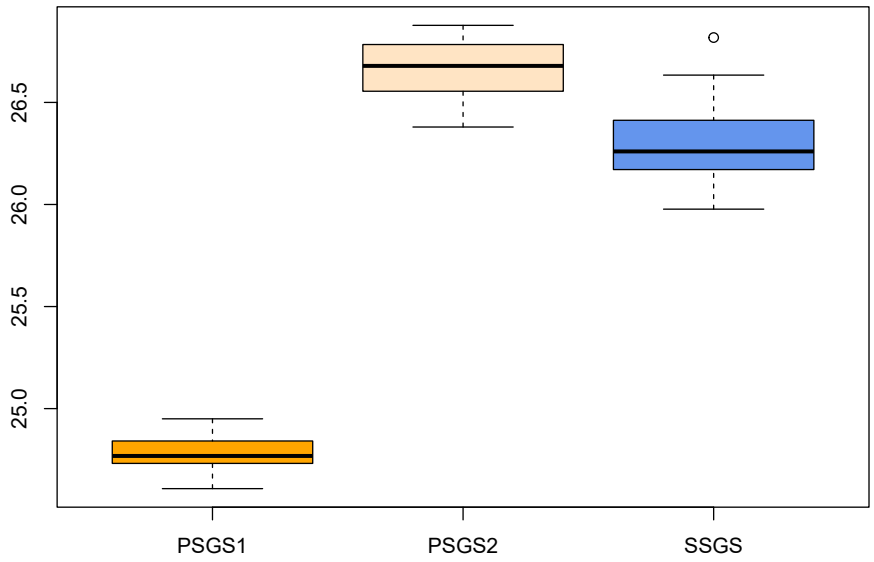


Slika 4.11: Ostvareni rezultati na ispitnom skupu s obzirom na korišteni SGS i funkciju F_1

ostvarenih rezultata za obje funkcije cilja. Zanimljivo je primijetiti da je raspršenost kod korištenja PSGS1 najmanja.

Tablica 4.22: Ostvareni rezultati na ispitnom skupu s obzirom na korišteni SGS i funkciju F_2

SGS	PSGS1	PSGS2	SSGS
min	24.60789	26.37950	25.97718
med	24.76803	26.67899	26.26001
avg	24.78413	26.66747	26.28886
max	24.94999	26.87761	26.81785
stdev	0.07678	0.13908	0.16313



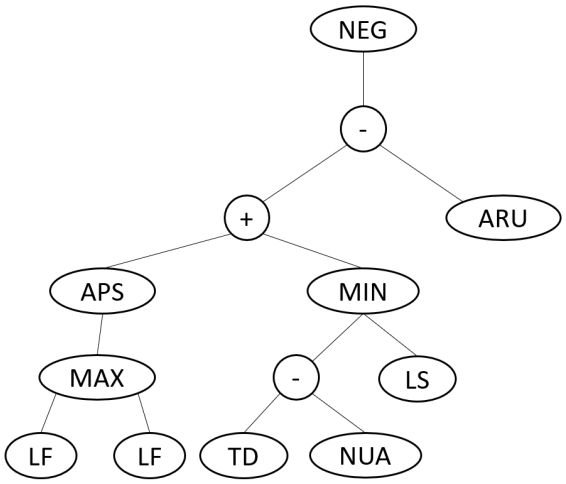
Slika 4.12: Ostvareni rezultati na ispitnom skupu s obzirom na korišteni SGS i funkciju F_2

4.4.1 Primjeri prioritetnih pravila

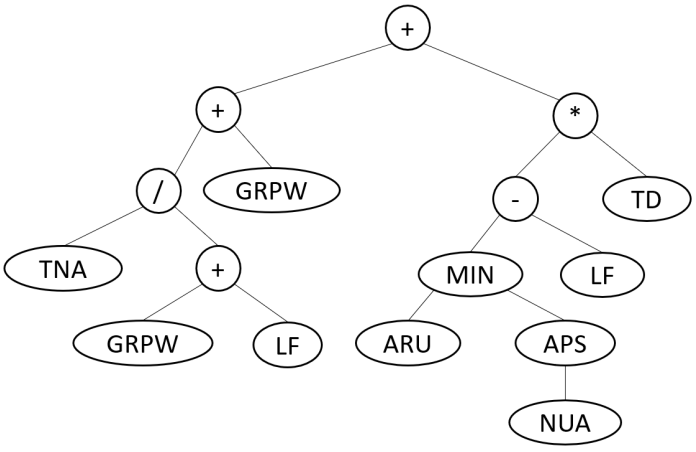
Prioritetna pravila razvijena GP-om su različitih oblika. Dva primjera razvijenih prioritetnih pravila dana su na slikama 4.13 i 4.14.

Na slici 4.13 prikazano je sljedeće pravilo:

$$NEG(APS(MAX(LF,LF)) + MIN(TD - NUA, LS) - ARU).$$



Slika 4.13: Primjer prioritetnog pravila



Slika 4.14: Primjer prioritetnog pravila

Ono što možemo primijetiti iz danog izraza je da razvijeno pravilo u sebi može imati suvišnih dijelova, odnosno dijelova koji ne doprinose vrijednosti prioriteta. Na primjer, u gornjem izrazu nije smisleno tražiti maksimum između istih vrijednosti, odnosno znamo da je $MAX(LF,LF) = LF$. Osim toga vrijednost značajke LF uvijek je veća ili jednaka od 0, pa je i $APS(LF) = LF$. Sukladno tome, prioritetno pravilo dano slikom 4.13 svodi se na pravilo:

$$NEG(LF + MIN(TD - NUA, LS) - ARU).$$

Dijelovi koji se nalaze u razvijenom prioritetnom pravilu, a matematičkim putem se mogu ukloniti, ne predstavljaju problem kod računanja prioriteta, jer se ne radi o računalno zahtjevnim izračunima, pa ne utječu bitno na vrijeme izvršenja. S druge strane, ovo pojednostavljenje utječe na lakšu interpretaciju samog rješenja. U ovom primjeru kod računanja prioriteta u obzir se uzimaju najkasniji trenutak početka i završetka, trajanje projekta, broj nezavršenih aktivnosti i prosječna upotreba sredstva. U ovom pravilu je zanimljivo primijetiti kako GP uspijeva na neki način iz traženja najvećeg prioriteta prijeći u traženje aktivnosti s najmanjim prioritetom. Naime, pravilo se može gledati u obliku izraza $LF + MIN(TD - NUA, LS) - ARU$ koji je na kraju pomnožen s (-1), odnosno ne tražimo više aktivnost koja ima najveću vrijednost prethodnog izraza, nego onu koja ima najmanju vrijednost.

Drugi primjer prioritetnog pravila, dan slikom 4.14, može se zapisati u sljedećem obliku:

$$\frac{TNA}{GRPW + LF} + GRPW + (MIN(ARU, APS(NUA) - LF) \cdot TD).$$

Ovo prioritetno pravilo ne može se sređivanjem bitno pojednostaviti kao što se moglo u prijašnjem primjeru. Jedini suvišni dio u ovom pravilu opet je dio s apsolutnom vrijednosti koja se može izostaviti za vrijednosti veće ili jednake od 0. S obzirom da su sve vrijednosti iz skupa značajki veće ili jednake 0, možemo zaključiti da apsolutna vrijednost svoj smisao može dobiti jedino u slučaju kada ne sadrži samo jednu značajku, odnosno općenitije, ako se izraz koji je pod apsolutnom vrijednosti ne može svesti na jednu značajku.

4.4.2 Usporedba s postojećim prioritetnim pravilima

Kako bi potvrdili kvalitetu razvijenih pravila potrebno je ostvarene rezultate usporediti s postojećim pristupima. Kada govorimo o rezultatima ostvarenim prioritetnim pravilima jedina poštena usporedba je ona s drugim, iz literature poznatim, prioritetnim pravilima. Usporedba prioritetnih pravila s ostalim metaheurističkim pristupima uvijek bi otišla na stranu tih pristupa zbog činjenice da su prioritetna pravila napravljena kako bi se njima moglo rješavati više instanci problema, dok se ostalim metaheurističkim pristupima svaka instanca problema rješava posebno. Ovo potvrđuje i usporedba razvijenih prioritetnih pravila i GA napravljena u radu

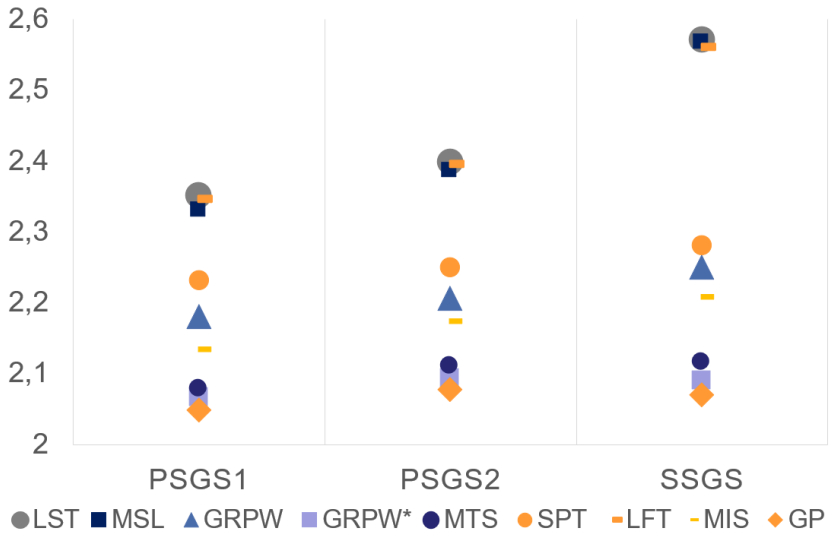
[35]. No, prednost prioritetnih pravila nije postignuta kvaliteta rješenja, nego činjenica da je to rješenje dovoljno dobro za praksu, a do njega se dolazi, u odnosu na ostale metaheurističke pristupe, brzo. Zbog toga se prioritetna pravila primjenjuju u dinamičkim okruženjima ili u onim situacijama kada se naglasak stavlja na brzinu kojom se dolazi do rješenja, a ne na kvalitetu rješenja.

Prioritetna pravila s kojima ćemo usporediti rješenja su ona dana tablicom 2.1 u poglavlju 2.3. Usporedba će biti napravljena za sve tri verzije SGS-a i za obje funkcije cilja. Rezultati ostvareni korištenjem funkcije F_1 dani su tablicom 4.23, dok su u tablici 4.24 prikazani oni ostvareni korištenjem funkcije F_2. Bitno je napomenuti da su vrijednosti ostvarene prioritetnim pravilima iz literature uvijek iste, dok vrijednost koja se nalazi u tablici kao ona koju ostvaruje pravilo razvijeno GP-om je minimalna vrijednost ostvarena u 50 pokretanja.

Pravila razvijena GP-om za obje funkcije cilja i sve tri verzije SGS-a ostvarila su najbolje rezultate. Statistički značajnu razliku između ostvarenih rezultata potvrđuje MWW test (p-vrijednost manja od 0.01). Zanimljivo je uočiti da su pravila razvijena GP-om ostvarila rezultate koji su drastično bolji u odnosu na jednostavna pravila kao što su LST i LFT, a koja su najčešće korištena u praksi. Razlika u ostvarenim rezultatima smanjuje se ako se koriste pravila GRPW* i MTS, ali je ta razlika i dalje značajna. Grafički prikazi ostvarenih rezultata dani su na slikama 4.15 i 4.16 gdje je razlika između rezultata postignutih korištenjem postojećih pravila i pravila razvijenih GP-om uočljiva.

Tablica 4.23: Usporedba s postojećim prioritetnim pravilima za F_1

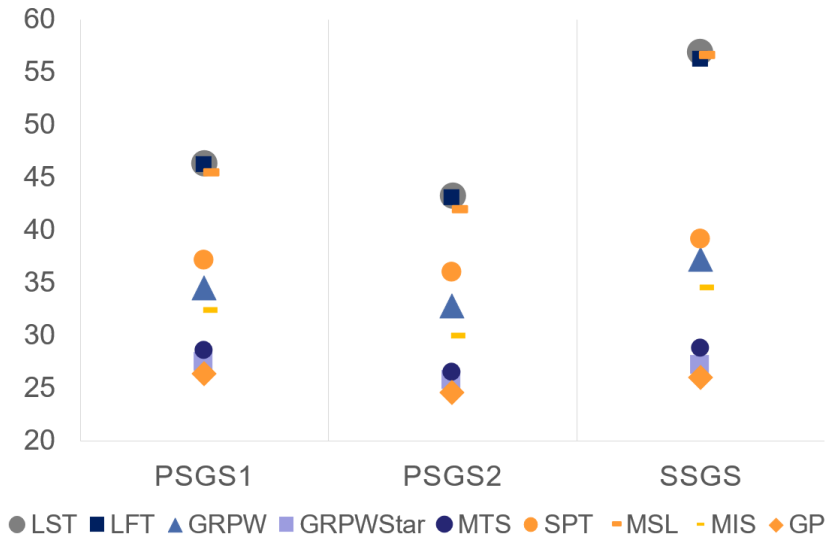
Pravilo	PSGS1	PSGS2	SSGS
LST	2.35050	2.39930	2.57140
MSL	2.32956	2.38562	2.56769
GRPW	2.17984	2.20676	2.25086
GRPW*	2.06746	2.09463	2.08979
MTS	2.07870	2.11150	2.11541
SPT	2.23121	2.25004	2.28140
LFT	2.34680	2.39637	2.56119
MIS	2.13429	2.17339	2.20779
GP	**2.04843**	**2.07672**	**2.07000**



Slika 4.15: Usporedba s postojećim prioritetnim pravilima za F_1

Tablica 4.24: Usporedba s postojećim prioritetnim pravilima za F_2

Pravilo	PSGS1	PSGS2	SSGS
LST	43.29920	46.34560	56.89380
LFT	43.09470	46.18580	56.23960
GRPW	32.81780	34.50480	37.30010
GRPW*	25.77620	27.48550	27.21040
MTS	26.46310	28.55220	28.82260
SPT	35.98900	37.19000	39.18380
MSL	41.95050	45.45720	56.66720
MIS	29.89870	32.36650	34.48180
GP	**24.60789**	**26.37950**	**25.97718**



Slika 4.16: Usporedba s postojećim prioritetnim pravilima za F_2

4.5 Zaključak

Zbog svoje prisutnosti u stvarnom svijetu, RCPSP je jedan od zanimljivih problema raspoređivanja uz ograničenja. S obzirom da kao i većina problema raspoređivanja pripada klasi NP-teških problema, u njegovom rješavanju najčešće se koriste heuristički pristupi. I dok su rješenja koje heurističke metode postižu u statičkim okruženjima dobra, za njegovo rješavanje u dinamičkom okruženju ili u situacijama kada je potrebno brzo doći do rješenja potrebno je koristiti prioritetna pravila.

Prioritetna pravila potrebno je kreirati posebno za svaki kriterij koji se želi optimizirati. Kreiranje prioritetnih pravila zahtijeva postojanje osobe koja dovoljno poznaje problem te se uglavnom svodi na metodu pokušaja i promašaja u kojoj se nakon iscrpnog testiranja dolazi do prioritetnog pravila koje se smatra kvalitetnim. S obzirom da GP može od jednostavnih struktura graditi kompleksnije, ideja je njime automatizirati postupak oblikovanja prioritetnih pravila.

Glavni naglasak ovog poglavlja je odabrati dobre jednostavne strukture čijim korištenjem će GP moći razviti kvalitetno prioritetno pravilo. U poglavlju su analizirane značajke i funkcije koje se susreću u literaturi kod razvoja prioritetnih pravila za RCPSP. Osim ovih skupova, u ovom poglavlju analizirani su skupovi za učenje i provjeru te ispitni skup. Također, s obzirom da prioritetna pravila dolaze u kombinaciji sa shemom za generiranje rasporeda, analizirane su tri različite vrste ovih shema.

Dobiveni rezultati pokazuju da za dobro pravilo skup značajki ne treba imati puno elemenata, te detaljnom analizom unutar poglavlja taj je skup s početnih 27 smanjen na samo 9 značajki što čini prostor rješenja znatno manjim nego kod velikih skupova značajki. Osim toga, rezultati pokazuju da GP-u različiti redovi veličina značajki od kojih je sastavljeno pravilo ne

predstavljaju problem i da se prilikom razvoja pravila bolje nosi s njima nego što to čini normalizacija značajki. Kod podjele podataka pokazana je važnost odvajanja dijela podataka za skup za provjeru, kao i to koliko je važno da se skup za učenje i provjeru sastoji od problema čije su karakteristike slične problemima na koje će se kasnije prioritetno pravilo primjenjivati.

Rezultati pokazuju i da se standardnim usporednim SGS-om postižu najbolja rješenja. Zanimljivo je da dodavanje odgode prilikom raspoređivanja ne doprinosi kvaliteti rješenja, odnosno unatoč činjenici da optimalan raspored može biti raspored s odgodom, prisilno dodavanje te odgode, u ovom slučaju, ne vodi k optimalnom rasporedu.

Na kraju, kako bi se pokazala kvaliteta razvijenih prioritetnih pravila, dana je usporedba s najboljim prioritetnim pravilima iz literature. Usporedba pokazuje da pravila razvijena GP-om ostvaruju statistički značajno bolje rezultate od rezultata ostvarenih postojećim prioritetnim pravilima. Čak i najbolja prioritetna pravila ostvarila su dosta lošije rezultate od onih ostvarenih razvijenim pravilima i to za sve tri, u poglavlju analizirane, verzije SGS-a. Automatizirani razvoj prioritetnih pravila donosi mogućnost da se prioritetna pravila razvijaju za različite kriterije i to bez dodatnog proučavanja problema, nego samo osiguravanjem kvalitetnih skupova za učenje i dobrim odabirom skupa primitiva.

S obzirom da ova istraživanja predstavljaju tek početak istraživanja primjene GP-a na razvoj prioritetnih pravila za RCPSP, otvorenih tema je puno. Iako je u ovom poglavlju dana dosta detaljna analiza vezana uz skup značajki, ta analiza može se produbiti kroz odabir značajki nekim drugim postupkom u kojem se doprinos značajki ne bi gledao posebno za svaku značajku nego s obzirom na njihov međusobni utjecaj, što može dovesti do boljih rezultata. Također, za konkretne probleme, ako postoji dodatno znanje koje može doprinijeti kvaliteti rješenja ono može biti dodano u samo pravilo, što se dosad nije analiziralo.

Iako je u poglavlju naglašavana važnost prioritetnih pravila za dinamička okruženja, nije napravljena analiza slučajeva u kojima se događa određena dinamička promjena i na koji način prilagoditi prioritetno pravilo da dobro reagira na takve promjene. Također, u ovom poglavlju analizirana su samo dva kriterija optimizacije, dok se ovaj pristup može primijeniti i na brojne druge, pri čemu je potrebno razmisliti i o dodavanju značajki specifičnih za taj kriterij. Dodatno, kompleksnost prioritetnih pravila može se uzeti u obzir i na neki način ograničiti, na primjer kroz višekriterijsku optimizaciju. Također, kroz višekriterijsku optimizaciju, osnovni kriterij može se kombinirati s brojnim drugim kriterijima.

Osim kompleksnosti prioritetnih pravila, trebala bi se napraviti i detaljnija analiza razvijenih pravila i na koji način prepoznati koji dijelovi unutar pravila više doprinose kvaliteti rješenja, odnosno koji čine bit tog pravila. Na taj način mogu se iz pravila izvući dodatne informacije koje se kasnije mogu upotrijebiti. Također, izostavljanjem pojedinih dijelova prioritetnih pravila moguće je da se, ako oni nisu bitni ili čak negativno utječu na kvalitetu rješenja, kvaliteta dodatno popravi.

Poglavlje 5

Oblikovanje ansambla prioritetnih pravila

Prilikom rješavanja optimizacijskih problema susrećemo se s problemom odabira postupka kojim to učiniti. Dodatno, ako se radi o NP-teškom problemu, tada bez obzira na odabir postupka, najčešće problem nije moguće egzaktno riješiti, zbog čega u rješavanju moramo koristiti neki od heurističkih pristupa koji ne jamči pronalazak optimalnog rješenja. Bez obzira koji optimizacijski problem rješavali, za njega postoje brojni heuristički pristupi koji većinom pokazuju dobre rezultate na nekim instancama problema, dok za niti jedan od njih ne možemo reći da je najbolji za cijelu klasu problema. Po *No Free Lunch* teoremu ovo je uobičajeno ponašanje za heuristike. Ovaj teorem nam govori da postupci pretraživanja u prosjeku imaju jednaku učinkovitost na konačnom skupu problema. Zbog ovog teorema se, uz pitanje koji postupak koristiti, javlja i pitanje mogu li se rezultati pojedinih postupaka kombinirati kako bi se dobilo bolje rješenje bez obzira na vrstu problema. S obzirom da u prosjeku svi postupci daju jednako dobre rezultate, a znamo da na pojedinom podskupu problema možemo utvrditi koji postupak je bolji, može se zaključiti da postupci ne griješe na isti način. Stoga, dobrim kombiniranjem pojedinačnih rezultata postoji mogućnost postizanja boljih rješenja. U ovom razmatranju nalazi se glavna ideja ansambala. Ansambli su metoda strojnog učenja u kojoj se generiraju pojedinačna rješenja koja se na kraju kombiniraju u konačno rješenje.

Ansambli se u strojnom učenju često upotrebljavaju kod klasifikacija s ciljem poboljšanja njezine točnosti [119] pri čemu se bagging [120] i boosting [121] često koriste kao postupci za nastajanje ansambla. Primjena ansambla kod GP-a nije toliko česta, no u literaturi pronalazimo uspješne primjere u radovima [122, 123, 124, 125].

Primjenu ansambala na pravila raspoređivanja razvijena GP-om pronalazimo u radovima [43, 45, 46, 47, 126]. U radu [45] korištena je kooperativna koevolucija (engl. *cooperative coevolution*) pri čemu nastaje ansambl čija je robustnost veća nego kod primjene samo jednog pravila. Sljedeći pristup koji pronalazimo u literaturi je NELLI-GP [46] u kojem se svako pravilo koje se nalazi u ansamblu specijalizira za određeni podskup problema. Rad [48] donosi primjenu pristupa iz [45] u dinamičkom okruženju proizvoljne obrade te predlaže višerazinski

GP za kreiranje ansambla koji se pokazuje bolji od kooperativne koevolucije. U dosad najdetaljnijoj analizi primjene ansambla na pravila raspoređivanja razvijena GP-om za okruženje nesrodnih strojeva u radu [43] autori analiziraju 4 različita pristupa razvoja ansambla: metodu jednostavnog kombiniranja, bagging, boosting i kooperativnu koevoluciju.

S obzirom da u literaturi dosad nije zabilježena primjena ovog pristupa na RCPSP, u ovom poglavlju, na osnovu metoda korištenih u radu [43], napravit će se detaljna analiza njegove primjene na RCPSP. Prethodno spomenute metode bit će prilagođene svojstvima RCPSP-a koja se razlikuju od svojstava problema proučavanog u navedenom radu.

5.1 Kombiniranje prioritetnih pravila u ansambl

U problemu raspoređivanja ansambl se sastoji od više pravila raspoređivanja, odnosno prioritetnih pravila. Prioritet aktivnosti s pomoću ansambla određuje se tako da se on računa koristeći sva pravila koja čine ansambl. Zbog toga je potrebno definirati način na koji pojedinačna prioritetna pravila sudjeluju u izračunu prioriteta. U literaturi pronalazimo dvije metode kombiniranja članova ansambla: metodu zbroja (engl. *sum method*) i metodu glasanja (engl. *vote method*) [119]. Ove metode koristit ćemo i kod ansambala za RCPSP.

Metoda zbroja podrazumijeva računanje prioriteta na način da se prioriteti pojedinačnih pravila zbroje te dobiveni iznos predstavlja prioritet pojedine aktivnosti. Nakon izračuna prioriteta primjenjuje se SGS kao što se to činilo kod pojedinačnih pravila. Ovaj način spajanja je vrlo jednostavan te ga je lako implementirati što je jedna od njegovih prednosti. Nedostatak ovog pristupa je to što veličine prioriteta koje se dobiju kod izračuna prioriteta koristeći pojedinačna pravila mogu biti različitih redova, pa zbog toga neka pravila mogu imati veći utjecaj na ukupni rezultat od drugih, odnosno utjecaj nekih pravila može biti neznatan. Problem velike razlike u iznosima prioriteta inače se rješava normalizacijom, no zbog dinamičkih uvjeta u kojima se prioritetna pravila najčešće primjenjuju to nije lak zadatak. No, u praksi se pokazuje da iznosi prioriteta dobiveni različitim pravilima ne variraju međusobno toliko da bi to stvorilo problem.

Druga metoda kombiniranja pojedinačnih pravila u ansambl je metoda glasanja. Kod ove metode, svako pravilo ima jednak utjecaj na odluku, odnosno svako pravilo daje svoj glas aktivnosti koja ima najveći prioritet ako se koristi to pravilo, te se aktivnost koja dobije najviše glasova raspoređuje prva. Iako se ova metoda uspješno nosi s problemom različitih iznosa prioriteta, ni ona ne dolazi bez nedostataka. Glavni nedostatak ove metode su moguća izjednačenja u glasovima. Što je brojniji skup aktivnosti za koje se računaju prioriteti, veća je mogućnost da se dogodi izjednačenje, odnosno da veliki broj aktivnosti dobije po jedan glas. U slučaju izjednačenja najčešće se postavlja neki od dodatnih uvjeta koji će to izjednačenje razriješiti. S ciljem zadržavanja jednostavnosti pristupa, u ovom radu aktivnost koja ima kraće vrijeme izvršenja imat će prednost prilikom raspoređivanja.

5.2 Načini kreiranja ansambala

Druga važna stvar kod ansambala je način na koji on nastaje, odnosno kako odabrati prioritetna pravila koja ga čine. U ovoj disertaciji za kreiranje ansambala koristit će se 4 različita pristupa: metoda jednostavnog kombiniranja, bagging, boosting i kooperativna koevolucija. Metodu jednostavnog kombiniranja (engl. *simple ensemble combination*, SEC) po prvi put u kreiranju ansambala kod pravila raspoređivanja koriste autori u radu [43], dok preostala tri pristupa, osim u tom radu susrećemo i u drugim radovima [45, 120, 127, 128, 129, 130].

5.2.1 Metoda jednostavnog kombiniranja

Metoda jednostavnog kombiniranja predstavlja način na koji se odabiru pojedinačna pravila koja će činiti ansambl. Prednost ove metode je što koristi unaprijed razvijena pravila te koristeći više postupaka odlučuje koja od tih pravila trebaju činiti ansambl. Postupci kojima se odabiru pravila za ansambl u metodi jednostavnog kombiniranja su metoda slučajnog odabira (engl. *random selection method*), metoda vjerojatnosnog odabira (engl. *probabilistic selection method*), metoda rasta (engl. *grow method*), metoda rasta i uklanjanja (engl. *grow-destroy method*) i metoda temeljena na instancama (engl. *instance based method*).

Metoda slučajnog odabira najjednostavniji je od postupaka za kreiranje ansambla korištenih u metodi jednostavnog kombiniranja. Pravila za ansambl odabiru se slučajno ne uzimajući u obzir nikakve informacije o njima. Kako bi se ostvario dobar rezultat, ovaj postupak potrebno je ponoviti više puta te se među svim tim odabirima za ansambl odabire onaj koji postiže najbolji rezultat. Što je broj ansambala koji se kreiraju veći rezultat je bolji. No, s druge strane s povećanjem broja ansambala koji se kreiraju povećava se kompleksnost i vrijeme izvršenja. S obzirom da se kod svakog kreiranja ispituje više ansambala i da svako pravilo ima jednaku vjerojatnost odabira, nije potrebno pratiti pojavu duplikata koji s obzirom na veliki broj mogućih kombinacija ne čine problem. Pseudokod ovog postupka dan je algoritmom 4.

Glavni nedostatak prethodnog pristupa je što se članovi ansambla biraju potpuno nasumično, odnosno ne koristi se nikakva informacija o njihovoj kvaliteti prilikom odabira. Kao odgovor na taj problem javlja se metoda vjerojatnosnog odabira. Za razliku od prethodne metode, ova metoda u obzir uzima dobrotu svakog pojedinačnog pravila. Zbog toga, prilikom odabira koje pravilo uključiti u ansambl veću vjerojatnost da budu izabrana imaju pravila čija funkcija cilja postiže manju vrijednost. Izračun vjerojatnosti za odabir pravila može biti različit. Pseudokod ovog pristupa, kao i formula za izračun vjerojatnosti odabira dani su algoritmom 5. Za razliku od pristupa korištenog u radu [43], kod pristupa korištenog u algoritmu 5 nije potrebno dodavanje konstante, jer za svako pravilo vjerojatnost da ono bude odabrano je veća od 0 i jednaka je normaliziranoj recipročnoj vrijednosti funkcije cilja koje pravilo postiže. Slično kao i u prethodnom pristupu, a s ciljem postizanja dobrih rezultata, nije dovoljno kreirati samo je-

Algoritam 4 Metoda slučajnog odabira

```
1: Ulaz: skup prioritetnih pravila PR, veličina ansambla, max. br. kreiranih ansambala
2: najbolji_ansambl ← ∅
3: dok je broj kreiranih ansambala manji od maksimalnog broja čini
4:    A ← ∅
5:    dok je veličina ansambla manja od predefinirane veličine čini
6:       nasumično odaberi pravilo iz skupa PR \ A i dodaj ga u A
7:    kraj
8:    ako je najbolji_ansambl prazan onda
9:       najbolji_ansambl← A
10:   inače ako A postiže bolji rezultat od najbolji_ansambl onda
11:      najbolji_ansambl ← A
12:   kraj
13: kraj
14: vrati najbolji_ansambl
```

dan ansambl nego je potrebno kreirati više njih i kao krajnji izabrati onaj koji postiže najmanju vrijednost funkcije cilja.

Algoritam 5 Metoda vjerojatnosnog odabira

```
1: Ulaz: skup prioritetnih pravila PR, veličina ansambla, max. br. kreiranih ansambala
2: najbolji_ansambl ← ∅
3: za svako pravilo iz PR čini
4:    izračunaj dobrotu pravila
5:    vjerojatnost odabira pravila postavi na recipročnu vrijednost dobrote pravila
6: kraj
7: normaliziraj vjerojatnosti pravila
8: dok je broj kreiranih ansambala manji od maksimalnog broja čini
9:    A ← ∅
10:   dok je veličina ansambla manja od predefinirane veličine čini
11:      odaberi pravilo iz skupa PR \ A na osnovu pridružene vjerojatnosti i dodaj ga u A
12:   kraj
13:   ako je najbolji_ansambl prazan onda
14:      najbolji_ansambl← A
15:   inače ako A postiže bolji rezultat od najbolji_ansambl onda
16:      najbolji_ansambl ← A
17:   kraj
18: kraj
19: vrati najbolji_ansambl
```

Metoda rasta predstavlja pohlepni heuristički pristup. Za razliku od prethodna dva pristupa, kod ovog pristupa nasumično se odabire tek prvo pravilo, dok se ostala pravila biraju na točno definirani način zbog čega ovaj algoritam, nakon izbora prvog pravila, postaje deterministički. Ideja ovog pristupa je da nakon što se prvo pravilo nasumično odabere, u svakoj sljedećoj iteraciji se odabere pravilo koje zajedno s pravilima koja se već nalaze u ansamblu ostvaruje najbolji rezultat. Bez obzira što se prilikom dodavanja novog prioritetnog pravila u ansambl može os-

tvariti lošiji rezultat u odnosu na prethodnu iteraciju, prioritetna pravila se nastavljaju dodavati u ansambl sve dok se ne postigne maksimalna veličina ansambla koja je unaprijed određena. Razlog što se dodavanje prioritetnog pravila nastavlja bez obzira na pogoršanje je što do pogoršanja ne mora doći samo zbog tog što dodano prioritetno pravilo ne pridonosi ansamblu, nego do njega dolazi i ako je u prethodnoj iteraciji ostvaren lokalni optimum. Ako je u prethodnoj iteraciji postignut lokalni optimum, dodavanje novog prioritetnog pravila predstavlja svojevrsni bijeg iz njega. Pseudokod je dan algoritmom 6. S obzirom da se radi o determinističkom algoritmu, do na izbor prvog pravila, a skup iz kojih se biraju pravila je konačan, postupak je moguće ponoviti na način da se svaki put odabere pravilo koje do sad nije bilo odabrano kao prvo.

Algoritam 6 Metoda rasta

1: **Ulaz:** skup prioritetnih pravila PR, veličina ansambla
2: A $\leftarrow \emptyset$
3: nasumično odaberi pravilo iz skupa PR i dodaj ga u A
4: **dok** je veličina ansambla manja od predefinirane veličine **čini**
5: odaberi pravilo iz skupa $PR \setminus A$ čije dodavanje u A najviše doprinosi kvaliteti rješenja
6: dodaj odabrano pravilo u A
7: **kraj**
8: vrati A

Metoda rasta i uklanjanja sastoji se od dva koraka. Prvi korak je gotovo isti kao i kod metode rasta, jedina razlika je u tome što se u ovom pristupu gradi ansambl čija je veličina dvostruko veća od unaprijed određene veličine ansambla. Nakon što se izgradi ansambl te veličine prelazi se na drugi korak u kojem se iz ansambla uklanja pravilo čijim uklanjanjem ansambl postiže najbolji rezultat. Postupak uklanjanja ponavlja se sve dok veličina ansambla ne postane jednaka unaprijed određenoj veličini. Kao i kod prethodnog postupka, nije bitno postižu li se kod dodavanja odnosno uklanjanja pravila iz ansambla bolji rezultati, nego se bira ono pravilo čijim se dodavanjem, odnosno uklanjanjem postiže najbolji rezultat. Pseudokod je dan algoritmom 7.

Zadnji pristup u sklopu metode jednostavnog kombiniranja je metoda temeljena na instancama. U ovom pristupu pravila se odabiru na način da se traži ono pravilo koje je dobro na onim instancama problema koje ansambl ne rješava dobro. Ovim načinom odabira potiče se raznolikost unutar ansambla što je bitno kod postizanja dobrih rezultata. Iz pseudokoda danog algoritmom 8 vidljivo je da je za svaku instancu problema unutar skupa za provjeru potrebno odrediti najmanju vrijednost funkcije cilja koju postiže neko od pojedinačnih pravila. Prvo pravilo koje se odabire u ansambl može se odabrati nasumično ili na neki drugi način. Na primjer, može se dodati ono pravilo koje na najviše instanci unutar skupa za provjeru ostvaruje rješenje jednako najmanjoj vrijednosti funkcije cilja koja se postiže na toj instanci problema s bilo kojim pojedinačnim pravilom. Nakon toga, kreira se novi skup instanci problema koji se sastoji

Algoritam 7 Metoda rasta i uklanjanja

```
1: Ulaz: skup prioritetnih pravila PR, veličina ansambla
2: A ← ∅
3: nasumično odaberi pravilo iz skupa PR i dodaj ga u A
4: dok je veličina ansambla manja od 2 * predefinirane veličine čini
5:     odaberi pravilo iz skupa PR \ A čije dodavanje u A najviše doprinosi kvaliteti rješenja
6:     dodaj odabrano pravilo u A
7: kraj
8: dok je veličina ansambla veća od predefinirane veličine čini
9:     odaberi pravilo u ansamblu A čije uklanjanje iz A najviše doprinosi kvaliteti rješenja
10:    ukloni odabrano pravilo iz A
11: kraj
12: vrati A
```

od onih instanci na kojima ansambl ne postiže jednake ili bolje rezultate od bilo kojeg pojedinačnog pravila. Ako pravilo ne postiže vrijednost koja je bolja od najmanje vrijednosti koja se postiže nekim pravilom na toj instanci problema, za sve instance problema unutar tog novog skupa, računa se odstupanje od tih vrijednosti te se pravilo koje ima najmanje odstupanje sljedeće dodaje u ansambl. Dodavanje pravila u ansambl se zaustavlja u trenutku kada se postigne željena veličina ansambla.

Kompleksnost ovih pristupa može se analizirati i kroz broj ansambala koji se evaluiraju prilikom pokretanja svake pojedine metode. Broj ansambala koji se evaluiraju kod metoda slučajnog i vjerojatnosnog odabira jednak je broju ansambala koje je potrebno kreirati, a koji je unaprijed određen. Kod metode rasta broj evaluacija ovisi o broju prioritetnih pravila koja se koriste za izgradnju ansambala kao i o veličini ansambla koji se kreira. U ovoj metodi, prvo pravilo bira se nasumično dok je za svako preostalo potrebno evaluirati više ansambala kako bi se odlučilo dodavanjem kojeg pravila se ostvaruje najbolji rezultat. Broj evaluacija u pojedinoj iteraciji ovisi o broju pravila koje je moguće dodati u ansambl, a koji se nakon svake iteracije smanji za jedan. Osim toga, u svakoj iteraciji evaluira se ansambl drugačije veličine, ovisno o tome koliko se pravila trenutno nalazi u ansamblu. Na primjer, ako kreiramo ansambl veličine 5 metodom rasta, tijekom postupka kreiranja evaluirat će se ansambli veličine 2, 3, 4 i 5. Broj evaluacija ansambla kod metode rasta i uklanjanja znatno je veći od broja evaluacija kod metode rasta. Povećanje broja evaluacija događa se iz dva razloga: prvi razlog je što se kod metode rasta i uklanjanja prvo gradi ansambl koji ima veličinu dva puta veću od predefinirane, a drugi je što je nakon toga potrebno ukloniti polovicu pravila iz ansambla kako bi se došlo do predefinirane veličine. Potrebno je primijetiti i da se zbog kreiranja ansambla koji ima dvostruko veću veličinu od predefinirane, tijekom kreiranja evaluiraju i ansambli veći od onog koji će ova metoda vratiti. Broj ansambala koji se evaluiraju prilikom kreiranja najmanji je kod metode temeljene na instancama, u kojoj se evaluira tek jedan ansambl po iteraciji. Naime, kod ove metode, u svakoj iteraciji dodaje se po jedno pravilo u ansambl na način da se prvo ansambl koji sadrži

Algoritam 8 Metoda temeljena na instancama

1: **Ulaz:** skup prioritetnih pravila *PR*, veličina ansambla, skup instanci problema u skupu za provjeru *VS*
2: A $\leftarrow \emptyset$
3: nasumično odaberi pravilo iz skupa *PR* i dodaj ga u *A*
4: **za** svaku instancu problema *p* u *VS*-u **čini**
5: pronađi najbolju vrijednost funkcije cilja ostvarenu na *p* bilo kojim pravilom iz *PR* i označi tu vrijednost s $opt[p]$
6: **kraj**
7: **dok** je veličina ansambla manja od predefinirane veličine **čini**
8: odredi skup *I* koji sadrži instance problema na kojima ansambl *A* ne postiže rezultat bolji ili jednak onom u *opt*
9: **za** svako pravilo *r* iz $PR \setminus A$ **čini**
10: $iopt[r]$=0
11: **za** svaku instancu problema *i* u *I* **čini**
12: izračunaj vrijednost dobrote pravila *r* na instanci *i* i označi je s fit
13: **ako** je $fit > opt[i]$ **tada**
14: $iopt[r] += fit - opt[i]$
15: **kraj**
16: **kraj**
17: **kraj**
18: odaberi pravilo *r* iz $PR \setminus A$ koje ima najmanju vrijednost $iopt[r]$ i dodaj u *A*
19: **kraj**
20: vrati *A*

pravila dodana do te iteracije evaluira na skupu instanci te se pronađu one instance na kojima ansambl ne ostvaruje rezultate bolje ili jednake onima ostvarenim pojedinačnim pravilima. Nakon toga, svako pojedinačno pravilo ispituje se na tim instancama te ono koje ostvari najbolji rezultat dodaje se u ansambl. Veličina ansambla koji se evaluira raste zajedno s iteracijama, a najveći ansambl koji se evaluira ima veličinu koja je za jedan manja od predefinirane.

5.2.2 BagGP

Bagging metoda [120] jedna je od najranije korištenih i najpopularnijih metoda kod ansambala. Ideja ove metode je da se učenje ne odvija na originalnom skupu za učenje, nego se na početku postupka metodom uzorkovanja s ponavljanjem (engl. *bootstrap method*) kreira više replika podataka koje se onda koriste za učenje. Prilikom uzorkovanja svaka instanca iz originalnog skupa za učenje ima jednaku vjerojatnost biti odabrana. Broj replika jednak je veličini ansambla, a veličina skupa za učenje dobivena metodom uzorkovanja s ponavljanjem može biti jednaka veličini originalnog skupa za učenje, ali može biti i manja ili veća. U novonastalim skupovima za učenje može se nalaziti više istih instanci problema.

Ova metoda strojnog učenja koja se koristi kod nastajanja ansambla primjenjuje se i kod GP-a te je poznata pod nazivom BagGP [127]. Skupovi nastali uzorkovanjem s ponavljanjem

koriste se kod razvoja prioritetnih pravila s ciljem postizanja specijalizacije pravila za određene instance problema, kao i postizanja veće raznolikosti unutar ansambla do koje dolazi zbog tog što se pravila razvijaju na različitim skupovima za učenje. Pseudokod ove metode dan je algoritmom 9.

Algoritam 9 BagGP

1: **Ulaz:** skup instanci problema za učenje *LS*, veličina ansambla, veličina novokreiranog skupa za učenje
2: A $\leftarrow \emptyset$
3: **dok** je veličina ansambla manja od predefinirane veličine **čini**
4: kreiraj skup za učenje *S* uzorkovanjem s ponavljanjem iz skupa *LS*
5: koristeći GP i skup *S* razvij novo prioritetno pravilo
6: dodaj razvijeno pravilo u ansambl *A*
7: **kraj**
8: vrati *A*

5.2.3 BoostGP

Boosting je još jedna od često korištenih metoda kreiranja ansambala kojoj je glavna ideja da se članovi ansambla dobiju slijednim učenjem modificiranih podataka, a krajnje rješenje se, najčešće, dobije kao težinska suma članova ansambla [131]. Unutar ove metode najpoznatiji je AdaBoost [121] algoritam kod kojeg je na početku težina svih instanci problema jednaka te se ona kasnije kroz iteracije mijenja, stavljajući naglasak na one probleme kod kojih su postignuti lošiji rezultati na način da se njihova težina povećava.

Primjena AdaBoost-a na GP u literaturi je poznata pod nazivom BoostGP [127, 128, 129]. Algoritam koji se primjenjuje prilikom kreiranja prioritetnih pravila od originalnog algoritma razlikuje se u funkciji cilja, jer za razliku od regresijskih problema gdje se radi o nadziranom učenju, u problemu raspoređivanja nije poznata optimalna vrijednost za niti jednu instancu problema, pa tako ni za one u skupu za učenje. Iz tih razloga, kao funkcija cilja za BoostGP uzima se ista funkcija koja se koristi kod običnog GP-a, koja ne može biti manja od 0 pa njezino korištenje ne stvara problem u originalnom pristupu. Pseudokod je dan algoritmom 10.

S obzirom da je uobičajeni način kombiniranja dobivenih modela AdaBoost metode težinski zbroj, za ovu metodu, načinima kombiniranja ansambala iz potpoglavlja 5.1 pridodat ćemo težinski zbroj i težinsko glasanje, pri čemu će težine pojedinog pravila odgovarati njegovoj pouzdanosti dobivenoj tijekom kreiranja ansambla.

5.2.4 Kooperativna koevolucija

Kooperativna koevolucija je evolucijski algoritam u kojem se problem dijeli na više potproblema koji se međusobno neovisno rješavaju te na kraju spajaju kako bi dali rješenje originalnog

Algoritam 10 BoostGP

Ulaz: skup instanci problema za učenje LS, skup instanci problema za provjeru VS, veličina ansambla
A $\leftarrow \emptyset$
inicijaliziraj težine za svaku instancu problema: $D_1(k) = \frac{1}{|LS|}, k \in \{1, \ldots, |LS|\}$
za i od 1 do veličine ansambla **čini**
    razvij prioritetno pravilo GP-om, koristeći sljedeću funkciju cilja:
    $fit_i = \sum_{k=1}^{|LS|} (f(ind, p_k) \cdot D_i(k)) \cdot |LS|$ pri čemu f označava originalnu funkciju cilja,
    p_k instancu problema na kojoj se računa vrijednost funkcije cilja i ind jedinku za koju se računa vrijednost funkcije cilja
    odredi najbolju jedinku ind_{best} u populaciji na skupu VS
    dodaj ind_{best} u ansambl A
    izračunaj gubitak za svaku instancu problema: $L_k = \frac{f(ind_{best}, p_k)}{\max_{l=1,\ldots,|LS|} f(ind_{best}, p_l)}$
    izračunaj prosječni gubitak: $L = \sum_{k=1}^{|LS|} L_k \cdot D_i(k)$
    izračunaj pouzdanost razvijenog pravila: $\beta_i = \frac{L}{1-L}$
    ažuriraj težine $D_{i+1}(k) = \frac{D_i(k) \cdot \beta_i}{Z_i}$, gdje Z_i predstavlja faktor normalizacije
kraj
vrati A

problema [130]. Prilikom rješavanja, svaki potproblem rješava jedna potpopulacija, a jedina interakcija između njih je u trenutku kada se kombiniraju kod evaluiranja. S obzirom na broj jedinki u populaciji nemoguće je napraviti kombinaciju svake jedinke sa svakom jedinkom iz druge potpopulacije, zbog čega se uvodi lista reprezentanata. U listi reprezentanata se iz svake potpopulacije nalazi jedna jedinka. U cilju pronalaska najboljeg reprezentanta za pojedinu potpopulaciju, svaka jedinka evaluira se u kombinaciji s ostalim reprezentantima i kao reprezentat odabire se ona koja u kombinaciji s ostalim reprezentantima ostvaruje najbolje rezultate. Pseudokod ovog pristupa je dan algoritmom 11. Kod kreiranja ansambla prioritetnih pravila ovaj pristup koristi se tako da svaka potpopulacija razvija pravilo koje postaje članom ansambla, a prilikom razvoja pravila njegova dobrota ocjenjuje se u kombinaciji s preostalim pravilima iz ansambla.

5.2.5 Podskup pravila ansambla

Traženje podskupa pravila u ansamblu (engl. *ensemble subset search*, skraćeno ESS) je pristup kojim se dodatno pokušava popraviti kvaliteta dobivenog rješenja. Ideja ovog pristupa je da sva pravila unutar ansambla ne moraju biti važna, odnosno da zbog načina kreiranja u ansamblu mogu postojati pravila koja ne doprinose kvaliteti rješenja, štoviše čak ju mogu i pogoršati. Takve situacije mogu se dogoditi prilikom neovisnog stvaranja pravila koja čine ansambl, npr.

Algoritam 11 Metoda kooperativne koevolucije

Ulaz: broj populacija (veličina ansambla), skup za provjeru
inicijaliziraj populacije P čiji broj je jednak veličini ansambla
inicijaliziraj listu reprezentanata R na način da za se svaku populaciju nasumično odabere jedna jedinka iz populacije
evaluiraj R na skupu za provjeru
za svaku populaciju p u skupu populacija P **čini**
    za svaku jedinku *ind* u p **čini**
        označi s C kopiju od R
        $C[p] \leftarrow ind$
        evaluiraj C na skupu za provjeru
        ako C postiže bolju vrijednost funkcije cilja od R **tada**
            $R \leftarrow C$
        kraj
    kraj
kraj
ponavljaj
    za svaku populaciju p u skupu populacija P **čini**
        odaberi k jedinki iz populacije p
        primjeni operator križanja na dvije najbolje jedinke od odabranih
        primjeni operator mutacije nad novonastalom jedinkom *ind*
        zamijeni najlošiju jedinku od odabranih s novonastalom jedinkom
        označi s C kopiju od R
        $C[p] \leftarrow ind$
        evaluiraj C na skupu za provjeru
        ako C postiže bolju vrijednost funkcije cilja od R **tada**
            $R \leftarrow C$
        kraj
    kraj
dok kriterij zaustavljanja nije ispunjen
vrati R

kod BagGP-a i BoostGP-a. ESS-om se unutar ansambla pronalazi podskup pravila koji daje najbolji rezultat na skupu instanci problema. Skup na kojem se određuje kvaliteta podskupa se treba razlikovati od onog na kojem su se pravila razvijala i na temelju kojeg se biralo najbolje pravilo. Pronalaskom najboljeg podskupa smanjuje se veličina ansambla što utječe na vrijeme potrebno za kreiranje rasporeda prilikom rješavanja problema, ali i na interpretabilnost rezultata. Kada veličina kreiranih ansambala nije prevelika, kao što će biti u slučaju testiranja u ovoj disertaciji, moguće je ispitati sve podskupove pravila unutar ansambla i pronaći najbolji. U ovoj disertaciji, prilikom testiranja neće se koristiti unaprijed određena veličina podskupa, kao što je to napravljeno u [43], već će se tražiti najbolji podskup pravila, bez obzira na njegovu veličinu. Pseukod za ESS dan je algoritmom 12.

Algoritam 12 Traženje podskupa pravila u ansamblu

```
1: Ulaz: ansambl, skup instanci problema za provjeru VS
2: pronađi sve podskupove pravila koje čine ansambl S
3: najbolji← ∅
4: za svaki podskup u S čini
5:    izračunaj vrijednost funkcije cilja na VS dobivenu korištenjem podskupa pravila
6:    ako je najbolji prazan onda
7:       najbolji← podskup
8:    inače ako je vrijednost funkcije cilja za podskup manja od vrijednosti funkcije
         cilja od najbolji onda
9:       najbolji← podskup
10:   kraj
11: kraj
12: vrati najbolji
```

5.3 Oblikovanje eksperimenta

Za ispitivanje metoda za kombiniranje, načina za kreiranje i veličine ansambla koristit će se instance problema koje se nalaze u PSPLIB-u. Za metodu jednostavnog kombiniranja koristit će se 50 pravila razvijenih u ispitivanjima provedenim u sklopu prethodnog poglavlja. Za ostala tri korištena pristupa, nova pravila razvijat će se na istim skupovima na kojima su se razvijala pojedinačna pravila u prethodnom poglavlju. Za kreiranje ansambala kao i za određivanje optimalnog podskupa pravila koristit će se novi skup koji će sadržavati 10% dosad nekorištenih instanci problema iz PSPLIB-a, tj. 204 instance. Parametri korišteni kod GP-a dani su tablicom 4.20. Kako bi se mogla utvrditi statistička značajnost ostvarenih rezultata, prilikom razvoja novih pravila, neovisno o pristupu, napravljeno je 30 pokretanja.

5.4 Analiza rezultata postignutih metodom jednostavnog kombiniranja

U ovom poglavlju analizirat će se rezultati postignuti s različitim pristupima unutar metode jednostavnog kombiniranja s ciljem pronalaska najboljeg pristupa koji će se koristiti prilikom usporedbe s drugim načinima kreiranja ansambala. Unutar analize uzet će se u obzir veličina ansambla, način kreiranja i način kojim se, koristeći ansambl, dodjeljuje prioritet aktivnosti. Osim toga, kod metoda slučajnog i vjerojatnosnog odabira analizirano je kako broj ansambala koji se kreiraju prilikom svakog pokretanja utječe na ostvareni rezultat. Kod obje metode, za broj ansambala koji se kreiraju odabrani su brojevi: 100, 500, 1000, 5000, 10000 i 20000. Kako bi ostvareni rezultati bili statistički značajni, algoritmi metoda slučajnog i vjerojatnosnog odabira, za sve navedene brojeve pokrenuti su 30 puta. Algoritmi za metodu rasta, metodu rasta i uklanjanja i metodu temeljenu na instancama pokrenuti su 50 puta na način da je za prvo pravilo

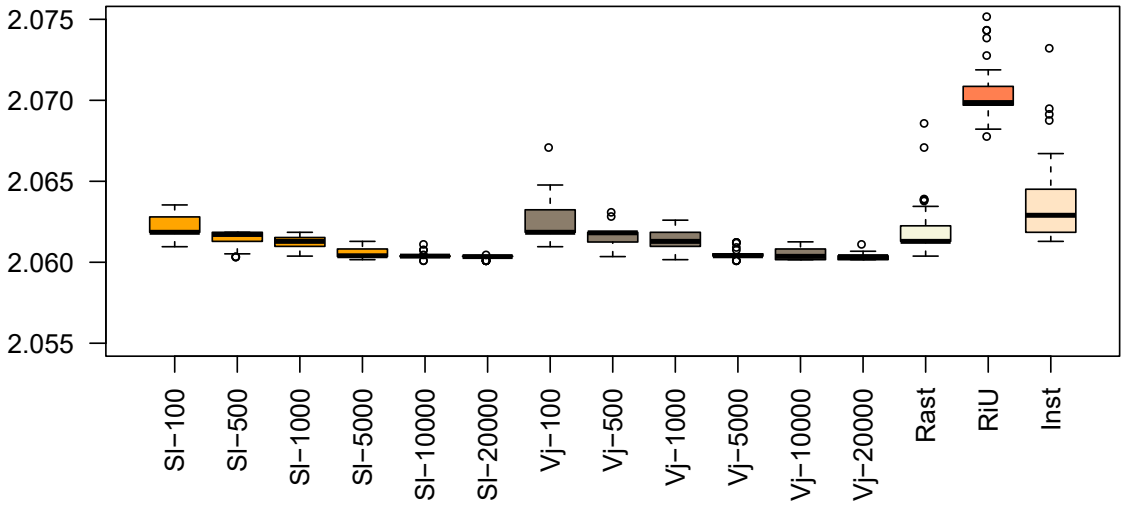
svaki put odabrano neko drugo od 50 prioritetnih pravila koja su korištena za kreiranje ansambla. Na ovaj su način, s obzirom da kod ovih metoda algoritam postaje deterministički nakon odabira prvog pravila, ispitane sve kombinacije koje se mogu dobiti koristeći dana prioritetna pravila.

Rezultati ostvareni korištenjem metoda jednostavnog kombiniranja dani su tablicama 5.1 - 5.4, pri čemu su rezultati u tablicama 5.1 i 5.2 dobiveni korištenjem funkcije F_1, a u tablicama 5.3 i 5.4 korištenjem funkcije F_2. U prethodno navedenim tablicama, ćelija je obojana sivo ako je rezultat bolji od rezultata ostvarenog pojedinačnim pravilima, a najmanja vrijednost u stupcu je dodatno podebljana. Prilikom ispitivanja analizirani su ansambli veličine 3, 5 i 7.

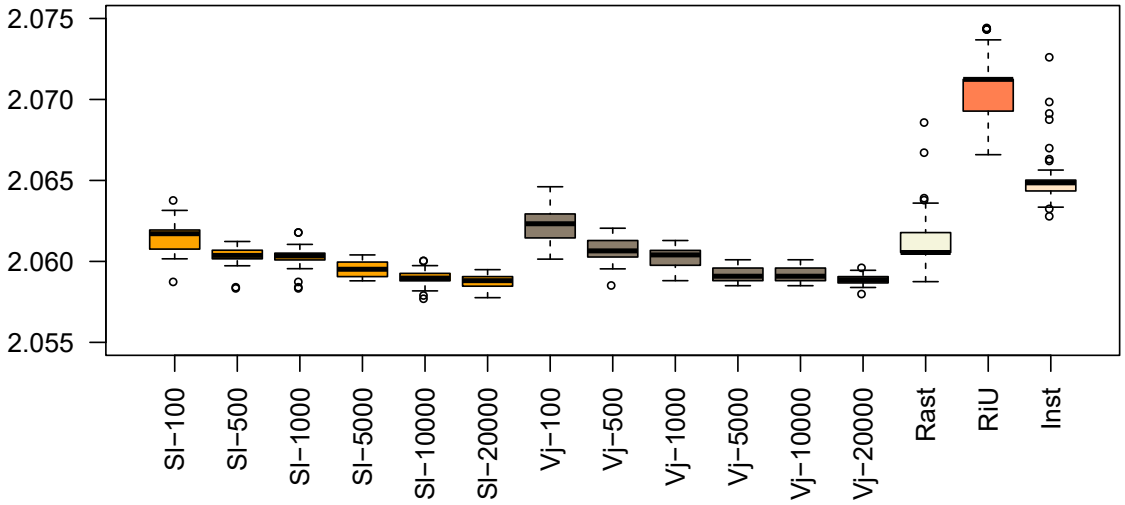
Rezultati pokazuju da metode slučajnog i vjerojatnosnog odabira, kao i metoda rasta ostvaruju rezultate koji su bolji od onih dobivenih pojedinačnim pravilima, bez obzira na korišteni kriterij optimizacije, kao i veličinu i metodu kombiniranja pravila u ansambl, dok za metodu rasta i uklanjanja i metodu temeljenu na instancama to nije slučaj. Kod metode rasta i uklanjanja prilikom korištenja funkcije F_1 i metode zbroja bez obzira na veličinu ansambla nije postignuta niti jedna vrijednost koja je bolja od onih postignutih pojedinačnim pravilima, dok za metodu glasanja ansambl je jedino uspio smanjiti maksimalnu dobivenu vrijednost u odnosu na pojedinačna prioritetna pravila. Metoda rasta i uklanjanja u kombinaciji s funkcijom F_2 i metodom glasanja postiže nešto bolje rezultate te uspijeva postići manji medijan nego onaj ostvaren korištenjem pojedinačnih pravila, no ipak ne uspijeva ostvariti manju minimalnu vrijednost. Metoda temeljena na instancama ostvaruje bolje rezultate od metode rasta i uklanjanja za obje funkcije cilja, no ipak ne tako dobre kao preostale metode jednostavnog kombiniranja.

Ako gledamo rezultate dobivene korištenjem funkcije F_1 i metode zbroja, možemo primijetiti da se povećanjem veličine ansambla postiže bolji rezultat, što pokazuju i Mann - Whitney - Wilcoxonov (MWW) test (p-vrijednost < 0.01), dok za metodu glasanja test ne pokazuje statističku značajnost u ostvarenim rezultatima za veličine 5 i 7. Razlika u rezultatima ostvarenim korištenjem metoda zbroja i glasanja ovisi o veličini ansambla te se na skupu za provjeru ne može utvrditi njihova značajna razlika.

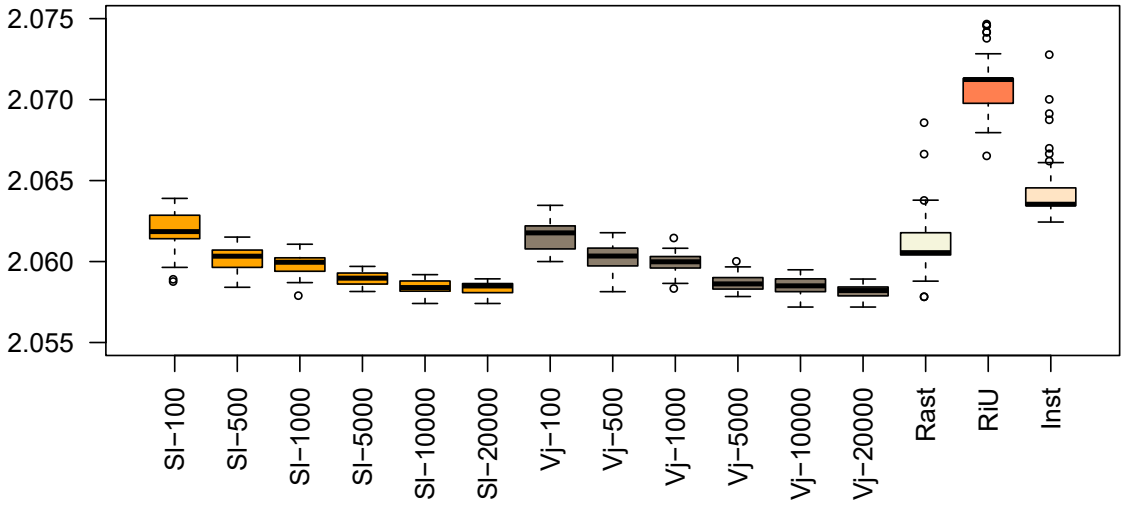
Na slikama 5.1 i 5.2 kutijastim dijagramima prikazani su rezultati dobiveni korištenjem funkcije F_1 i metode zbroja, odnosno metode glasanja s obzirom na veličinu ansambla. Na slikama metoda slučajnog odabira označena je sa Sl, a vjerojatnosnog odabira s Vj te uz njihove kratice stoji i broj kreiranih ansambala, metoda rasta je označena s kraticom Rast, metoda rasta i uklanjanja s RiU i metoda temeljna na instancama s Inst. Bez obzira na veličinu ansambla i metodu kombiniranja prioritetnih pravila unutar ansambla, razlika između korištenih pristupa je lako uočljiva na dijagramu. Metode slučajnog i vjerojatnosnog odabira postižu rezultate koji su puno bolji od rezultata postignutih s preostale tri metode, neovisno o broju kreiranih ansambala prilikom pokretanja.



(a) veličina ansambla 3

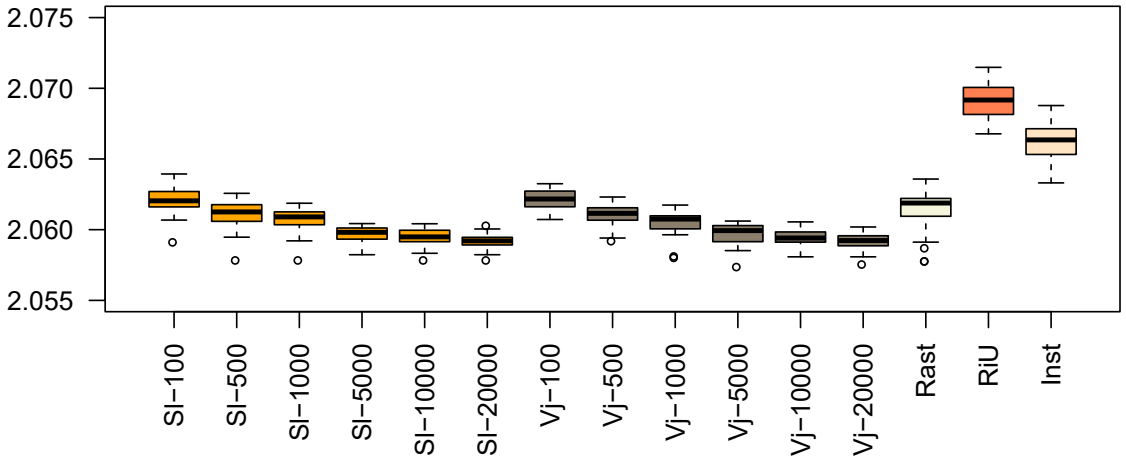


(b) veličina ansambla 5

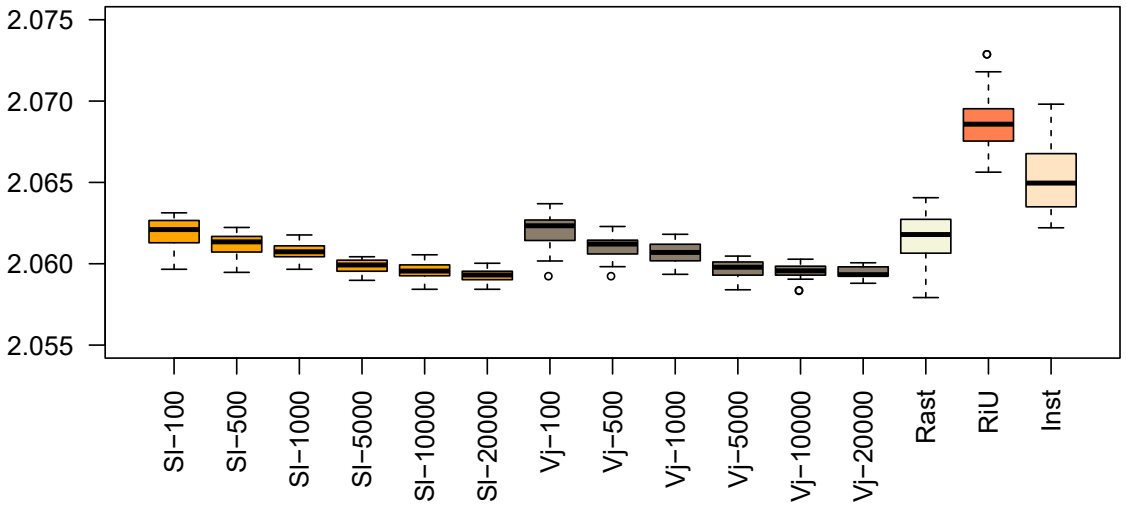


(c) veličina ansambla 7

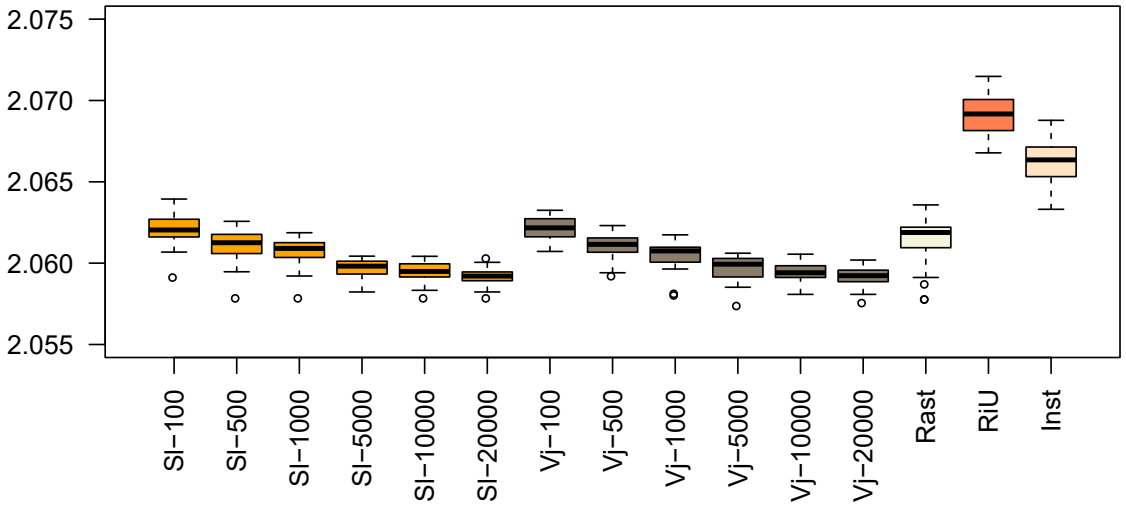
Slika 5.1: Rezultati za metodu jednostavnog kombiniranja uz korištenje funkcije F_1 i metode zbroja



(a) veličina ansambla 3



(b) veličina ansambla 5



(c) veličina ansambla 7

Slika 5.2: Rezultati za metodu jednostavnog kombiniranja uz korištenje funkcije F_1 i metode glasanja

Tablica 5.1: Rezultati za metode jednostavnog kombiniranja ostvareni na skupu za provjeru za F_1 metodom zbroja

Metoda		Veličina ansambla								
		3			5			7		
		min	med	max	min	med	max	min	med	max
slučajnog odabira	100	2.06096	2.06185	2.06354	2.05880	2.06171	2.06383	2.05883	2.06185	2.06390
	500	2.06038	2.06172	2.06185	2.05840	2.06037	2.06123	2.05841	2.06033	2.06151
	1000	2.06038	2.06129	2.06185	2.05840	2.06034	2.06185	2.05796	2.05996	2.06107
	5000	**2.06016**	2.06040	2.06129	2.05880	2.05952	2.06040	2.05815	2.05898	2.05970
	10000	**2.06016**	2.06038	2.06117	**2.05776**	2.05897	2.06010	2.05741	2.05840	2.05919
	20000	**2.06016**	**2.06035**	**2.06052**	**2.05776**	**2.05881**	**2.05949**	2.05741	2.05849	2.05893
vjerojatnosnog odabira	100	2.06096	2.06185	2.06715	2.06014	2.06233	2.06461	2.06000	2.06177	2.06347
	500	2.06035	2.06182	2.06315	2.05858	2.06065	2.06205	2.05814	2.06034	2.06178
	1000	**2.06016**	2.06129	2.06260	2.05881	2.06040	2.06129	2.05840	2.05999	2.06152
	5000	**2.06016**	2.06040	2.06129	2.05850	2.05909	2.06010	2.05784	2.05862	2.06008
	10000	**2.06016**	2.06037	2.06126	2.05850	2.05909	2.06010	**2.05719**	2.05850	2.05949
	20000	**2.06016**	**2.06035**	2.06117	2.05805	2.05887	2.05967	**2.05719**	**2.05821**	**2.05892**
metoda rasta		2.06038	2.06129	2.06864	2.05875	2.06055	2.06864	2.05789	2.06055	2.06864
metoda rasta i uklanjanja		2.06783	2.06986	2.07523	2.06659	2.07124	2.07448	2.06659	2.07124	2.07473
metoda temeljena na instancama		2.06129	2.06290	2.07328	2.06285	2.06485	2.07267	2.06244	2.06354	2.07284
pojedinačna pravila		2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408

Tablica 5.2: Rezultati za metode jednostavnog kombiniranja ostvareni na skupu za provjeru za F_1 metodom glasanja

Metoda		Veličina ansambla								
		3			5			7		
		min	med	max	min	med	max	min	med	max
slučajnog odabira	100	**2.05953**	2.06157	2.06353	2.05966	2.06210	2.06313	2.05918	2.06204	2.06394
	500	**2.05953**	2.06122	2.06232	2.05947	2.06135	2.06223	2.05790	2.06126	2.06257
	1000	**2.05953**	2.06108	2.06194	2.05966	2.06074	2.06177	2.05790	2.06091	2.06187
	5000	**2.05953**	2.05998	2.06117	2.05898	2.05993	2.06043	2.05823	2.05982	2.06043
	10000	**2.05953**	2.05998	2.06042	2.05843	2.05955	2.06055	2.05790	2.05949	2.06042
	20000	**2.05953**	**2.05976**	2.06042	2.05843	**2.05931**	**2.06003**	2.05790	**2.05921**	2.06035
vjerojatnosnog odabira	100	2.06052	2.06216	2.06322	2.05930	2.06234	2.06369	2.06072	2.06217	2.06325
	500	2.05998	2.06119	2.06217	2.05930	2.06121	2.06229	2.05926	2.06116	2.06231
	1000	**2.05953**	2.06042	2.06164	2.05935	2.06070	2.06181	2.05808	2.06075	2.06174
	5000	**2.05953**	2.05998	2.06053	2.05840	2.05979	2.06047	**2.05743**	2.05994	2.06061
	10000	**2.05953**	2.05998	2.06042	2.05840	2.05957	2.06028	2.05808	2.05941	2.06055
	20000	**2.05953**	2.05998	**2.06014**	2.05880	2.05933	2.06006	2.05761	2.05923	**2.06019**
metoda rasta		2.06013	2.06223	2.06509	**2.05792**	2.06180	2.06406	2.05783	2.06189	2.06358
metoda rasta i uklanjanja		2.06493	2.06807	2.07192	2.06563	2.06858	2.07294	2.06678	2.06917	2.07148
metoda temeljena na instancama		2.06142	2.06436	2.06802	2.06221	2.06496	2.06981	2.06331	2.06635	2.06878
pojedinačna pravila		2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408

Tablica 5.3: Rezultati za metode jednostavnog kombiniranja ostvareni na skupu za provjeru za F_2 metodom zbroja

Metoda		Veličina ansambla								
		3			5			7		
		min	med	max	min	med	max	min	med	max
slučajnog odabira	100	24.57135	24.59576	24.72043	24.57677	24.68986	24.74359	24.57094	24.70136	24.75004
	500	24.51049	24.58685	24.59576	24.57094	24.59097	24.68150	24.56017	24.63910	24.69184
	1000	24.51049	24.58685	24.59576	24.55639	24.58685	24.62376	24.54283	24.61421	24.65550
	5000	**24.51049**	24.56214	24.58605	24.45906	24.57094	24.58685	24.55639	24.57113	24.59961
	10000	**24.51049**	24.52907	**24.57044**	24.45906	24.56266	24.57113	**24.46400**	**24.56549**	24.59218
	20000	**24.51049**	**24.51049**	**24.57044**	**24.43371**	24.53575	**24.56549**	**24.46400**	**24.56549**	**24.58077**
vjerojatnosnog odabira	100	24.58887	24.67449	24.90328	24.59043	24.68046	24.86182	24.57094	24.69956	24.77956
	500	24.51049	24.58887	24.65751	24.56670	24.60210	24.64992	24.56549	24.64313	24.71344
	1000	24.51049	24.58786	24.69334	24.50511	24.58685	24.65347	24.55639	24.61317	24.67444
	5000	**24.51049**	24.57044	24.58685	24.52026	24.57094	24.58685	24.56549	24.57657	24.61550
	10000	**24.51049**	24.57044	24.58605	24.48934	24.56549	24.57135	24.56017	24.56581	24.59218
	20000	**24.51049**	24.52907	24.57134	**24.43371**	**24.52724**	24.57094	24.50760	**24.56549**	24.58552
metoda rasta		24.57134	24.61567	25.01811	24.53295	24.58952	25.01811	24.54667	24.58177	25.01811
metoda rasta i uklanjanja		24.89570	25.14963	25.40571	24.90713	25.18299	25.43604	24.90345	25.23821	25.45340
metoda temeljena na instancama		24.58685	24.75594	25.35651	24.58605	24.73713	25.15108	24.71667	24.78602	25.19809
pojedinačna pravila		24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447

Tablica 5.4: Rezultati za metode jednostavnog kombiniranja ostvareni na skupu za provjeru za F_2 metodom glasanja

Metoda		Veličina ansambla								
		3			5			7		
		min	med	max	min	med	max	min	med	max
slučajnog odabira	100	24.44938	24.58539	24.64138	24.45892	24.58847	24.67332	24.32209	24.60044	24.70306
	500	**24.41209**	24.52431	24.59849	24.38659	24.49362	24.55800	24.32209	24.51289	24.58277
	1000	24.42737	24.50313	24.54474	24.36649	24.45292	24.53394	24.39034	24.48941	24.55370
	5000	**24.41209**	24.47453	24.50641	24.34509	24.41823	24.45936	24.34407	24.43900	24.50427
	10000	**24.41209**	24.45531	24.50439	24.28272	24.38312	24.46120	24.32209	24.41369	24.46424
	20000	**24.41209**	24.44938	24.47521	**24.25819**	**24.36992**	**24.41536**	24.31608	**24.38620**	24.44646
vjerojatnosnog odabira	100	24.48174	24.59733	24.68234	24.44538	24.56688	24.68346	24.37377	24.58085	24.68371
	500	**24.41209**	24.50448	24.57980	24.36413	24.50673	24.59757	24.42027	24.52829	24.57007
	1000	**24.41209**	24.52758	24.59849	24.30554	24.49795	24.57856	24.40453	24.51021	24.57430
	5000	**24.41209**	24.46458	24.56650	24.30554	24.42139	24.49593	**24.29687**	24.43078	24.49121
	10000	**24.41209**	24.43838	24.48414	**24.25819**	24.39168	24.49772	**24.29687**	24.40455	24.47842
	20000	**24.41209**	**24.42737**	**24.46458**	**24.25819**	24.37053	24.42836	**24.29687**	24.39393	**24.44316**
metoda rasta		**24.41209**	24.58912	24.92909	24.36490	24.53478	24.74835	24.31825	24.48142	24.78493
metoda rasta i uklanjanja		24.64422	24.92968	25.18766	24.73526	25.00402	25.24227	24.73854	24.98022	25.37255
metoda temeljena na instancama		24.52197	24.72658	24.92218	24.50050	24.84560	25.09209	24.67627	24.82498	25.04317
pojedinačna pravila		24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447

S druge strane, ako gledamo samo te dvije metode, na ostvarene rezultate bitno utječe broj kreiranih ansambala, odnosno povećanjem broja ansambala koji se kreiraju ostvaruju se bolji i manje raspršeni rezultati. Zanimljivo je uočiti da se kod manjih veličina ansambala, nakon određenog broja, postiže ista minimalna vrijednost ansambla i kod manjeg i kod većeg broja kreiranih ansambala, za razliku što se kod većih ansambala ona dodatno smanjuje s povećanjem broja kreiranih ansambala. No, iako se kod ansambala manje veličine s povećanjem broja kreiranih ansambala dodatno ne smanjuje minimalna ostvarena vrijednost smanjuje se medijan, zbog čega ima smisla kreirati veći broj ansambala. Pojava da se vrijednosti kod manjih ansambala stabiliziraju ranije, odnosno postignu određene vrijednosti s manjim brojem kreiranih ansambala je i očekivana jer je puno manje mogućih kombinacija za ansambl manje veličine nego kod većih veličina.

Rezultati postignuti metodom slučajnog i vjerojatnosnog odabira prilično su slični te između njih ne postoji statistički značajna razlika. S obzirom da vjerojatnosni odabir, za razliku od slučajnog odabira, u sebi povlači i izračun određenih vjerojatnosti što utječe na vrijeme izvršavanja, u daljnjim ispitivanjima i usporedbi s ostalim metodama kreiranja ansambala koristit će se slučajni odabir s 20 tisuća kreiranih ansambala.

5.5 Analiza rezultata s obzirom na način kreiranja ansambala

U ovom poglavlju bit će napravljena analiza rezultata postignutih korištenjem različitih načina kreiranja ansambla. Prilikom razvoja prioritetnih pravila, kao i kreiranja ansambla koristit će se funkcije F_1 i F_2, a pojedinačna pravila unutar ansambla kombinirat će se metodama zbroja i glasanja. Osim načina kreiranja ansambla, analizirat će se i veličina kreiranih ansambala s ciljem određivanja optimalne veličine. Analiza ostvarenih rezultata napravit će se za svaki način kreiranja posebno te će se unutar te analize odrediti i optimalna veličina ansambla za pojedini način kreiranja. Prilikom analize koristit će se skup za provjeru, dok će se na kraju napraviti međusobna usporedba načina kreiranja na ispitnom skupu.

Nakon što su ansambli kreirani, koristit će se ESS kako bi se utvrdilo postoji li mogućnost pronalaska podskupa prioritetnih pravila od kojih je sastavljen ansambl, a koji će ostvarivati bolje rezultate od originalnog ansambla. Ovaj postupak, osim što može dodatno poboljšati ostvarene rezultate, može i smanjiti veličinu ansambla, što utječe na vrijeme potrebno za evaluaciju i kreiranje rasporeda. Veličina ansambla nastalog nakon primjene ESS-a neće se dodatno analizirati te će u tablici s rezultatima biti naznačena veličina ansambla na kojem je primijenjen ESS i rezultati ostvareni nakon primjene.

Za ispitivanje načina kreiranja ansambla i određivanje optimalne veličine ansambla koristit će se skup za provjeru kao i u prethodnom poglavlju. Isti taj skup koristit će se i za ispitivanje

ESS-a, dok će se za konačnu usporedbu različitih načina kreiranja ansambala koristiti ispitni skup.

Rezultati će biti prikazani u tablicama na način da će ćelije unutar tablice biti obojane sivom bojom ako se postiže rezultat bolji od onog koji je ostvaren pojedinačnim pravilima, te će najbolja vrijednost u stupcu biti dodatno podebljana. Statistička značajnost u ostvarenim rezultatima ispitivat će se MWW testom s razinom značajnosti od 0.05 ako to nije drugačije naglašeno.

5.5.1 Metoda jednostavnog kombiniranja

Rezultati postignuti metodom jednostavnog kombiniranja uz korištenje funkcije F_1 dani su tablicom 5.5, te uz korištenje funkcije F_2 tablicom 5.6. Prikazani rezultati dobiveni su korištenjem metode slučajnog odabira uz 20 tisuća kreiranih ansambala i 30 pokretanja. Razlika u ostvarenim rezultatima između kreiranih ansambala i pojedinačnih pravila je statistički značajna za sve slučajeve (MWW test, p-vrijednost < 0.01). Dodatno, rezultati uz korištenje obje funkcije prikazani su kutijastim dijagramom na slici 5.3 pri čemu je za metodu zbroja korištena oznaka Z, a za metodu glasanja G, te uz njih stoji broj koji označava veličinu kreiranog ansambla.

Prvo što možemo primijetiti na dijagramima, bez obzira na korištenu funkciju cilja i metodu kombiniranja unutar ansambla, je da za ansamble veličine 2 imamo samo ravnu crtu, odnosno u svih 30 pokretanja odabran je isti ansambl. Razlog tome leži u činjenici da se ansambl kreira 20 tisuća puta u jednom pokretanju što je nešto više od 16 puta veći broj od ukupnog broja svih kombinacija 50 pravila za ansambl veličine 2 zbog čega se u svakom pokretanju pronašao upravo onaj koji je najbolji. Na temelju ovog razmatranja može se zaključiti da je za manje veličine moguće ispitati sve kombinacije umjesto da se kreira 20 tisuća nasumičnih ansambala. Ispitivanje svih kombinacija moguće je za veličinu 2 što bi uvelike smanjilo broj kreiranja i veličinu 3 gdje bi taj broj uz pretraživanje svih kombinacija i nasumično kreiranje 20 tisuća ansambala bio podjednak, dok već za veličinu 4 pretraživanje svih kombinacija zahtijeva evaluaciju oko 10 puta više ansambala što je računalno izrazito vremenski zahtjevno.

U slučaju korištenja funkcije F_1 kao funkcije cilja i jednakih veličina ansambala, metoda glasanja ostvaruje statistički značajno bolje rezultate od metode zbroja za veličine 2 i 3, za veličinu 4 ta razlika nije statistički značajna, dok za ansamble veličine 5 i veće statistički značajno bolji rezultati ostvaruju se kod metode zbroja. Kod funkcije F_2 situacija je drugačija te se kod nje korištenjem metode glasanja za sve veličine postižu statistički značajno bolji rezultati od onih ostvarenih korištenjem metode zbroja.

Kod određivanja optimalne veličine za metodu zbroja uz korištenje funkcije F_1 uočava se da se najmanji medijan ostvaruje za veličinu 10. MWW test (p-vrijednost < 0.01) potvrđuje da su rezultati ostvareni korištenjem ansambla veličine 10 statistički značajno bolji u odnosu na one gdje su korišteni ansambli veličine od 2 do 7, dok statistički značajna razlika između ovih rezultata i rezultata ostvarenih korištenjem ansambala veličina 8 i 9 nije utvrđena. U skladu s

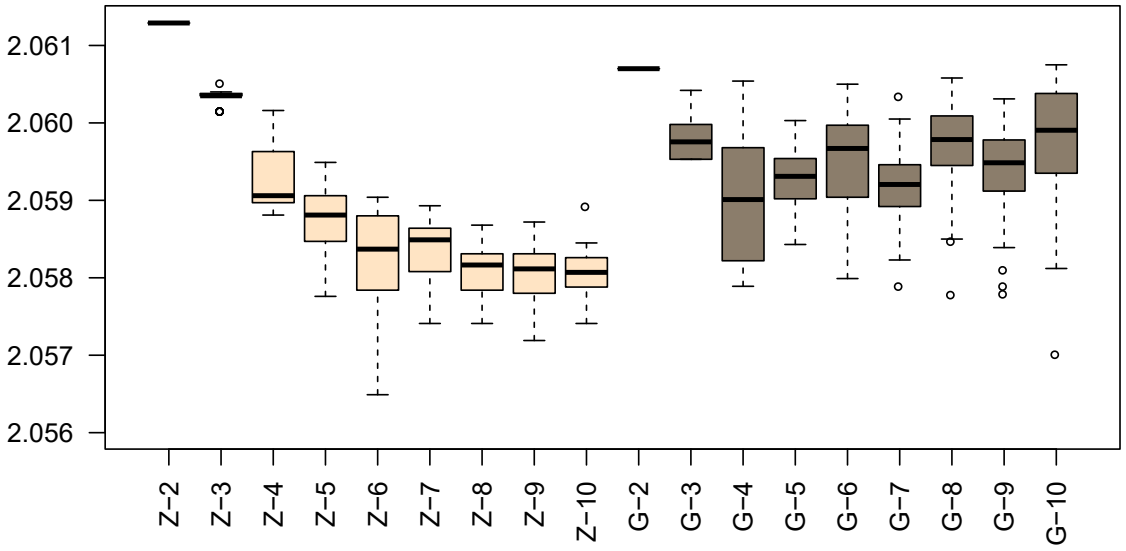
ovim razmatranjem kod metode zbroja moguće je koristiti veličine 8, 9 i 10. S ciljem smanjenja kompleksnosti i vremena izvršenja u daljnjim usporedbama za kombinaciju funkcije F_1 i metode zbroja koristit će se ansambl veličine 8. S druge strane, s promjenom veličine ansambla za metodu glasanja nije moguće uočiti neki trend povećanja ili smanjenja. Iz rezultata prikazanih u tablici, vidljivo je da se najmanji medijan ostvaruje za ansambl veličine 4, te statistički testovi pokazuju postojanje značajne razlike između ansambla te veličine i ansambala veličine $2, 3, 6, 8$ i 10. Za daljnje usporedbe za metodu glasanja i funkciju F_1 koristit će se ansambl veličine 4.

Na isti način možemo promatrati situaciju za funkciju F_2 kod koje, za metodu zbroja, primjećujemo da se najbolji rezultati postižu za veličine ansambla 3 i 4, te su ti rezultati statistički značajno bolji od rezultata ostvarenih za ostale veličine ansambla. Za daljnju usporedbu prilikom korištenja metode zbroja i funkcije F_2 uzet će se veličina ansambla 3. Kod metode glasanja odluka je jednostavna jer se za veličinu ansambla 5 postižu statistički značajno bolji rezultati od rezultata za sve ostale veličine.

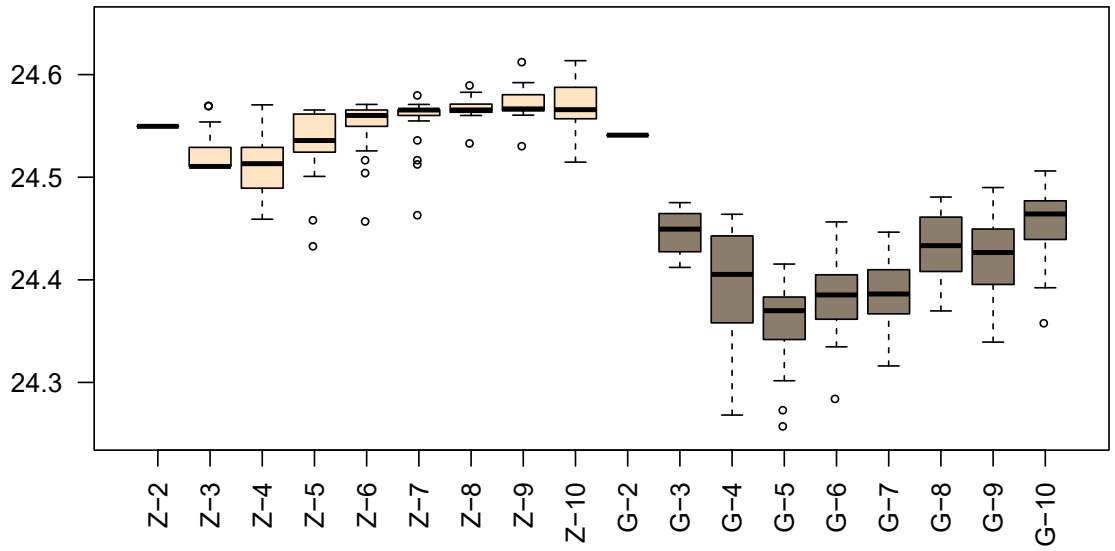
Tablica 5.5: Rezultati postignuti za metodu jednostavnog kombiniranja koristeći F_1

Veličina	metoda zbroja			metoda glasanja		
	min	med	max	min	med	max
2	2.06129	2.06129	2.06129	2.06070	2.06070	2.06070
3	2.06016	2.06035	2.06052	2.05953	2.05976	2.06042
4	2.05881	2.05906	2.06016	2.05789	**2.05901**	2.06054
5	2.05776	2.05881	2.05949	2.05843	2.05931	**2.06003**
6	**2.05649**	2.05837	2.05904	2.05799	2.05967	2.06050
7	2.05741	2.05849	2.05893	2.05790	2.05921	2.06035
8	2.05741	2.05817	**2.05868**	2.05779	2.05979	2.06058
9	2.05719	2.05812	2.05872	2.05780	2.05949	2.06031
10	2.05741	**2.05807**	2.05893	**2.05702**	2.05991	2.06075
pojedinačna pravila	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408

Primjenom ESS-a na ansamble kreirane metodom jednostavnog kombiniranja dobivamo rezultate prikazane tablicom 5.7 u slučaju funkcije cilja F_1, odnosno tablicom 5.8 u slučaju funkcije cilja F_2. Ostvareni rezultati prikazani su i kutijastim dijagramima na slici 5.4 uz iste oznake kao na prethodnoj slici. Iz danih rezultata lako uočavamo da korištenje ESS-a nije dovelo do značajnijih poboljšanja rezultata na skupu za provjeru neovisno o korištenoj funkciji cilja i metodi kombiniranja. S obzirom da korištenje ESS-a zahtjeva dodatno vrijeme, a ne rezultira dodatnim poboljšanjem kod metode jednostavnog kombiniranja nema ga potrebe koristiti.



(a) funkcija cilja F_1



(b) funkcija cilja F_2

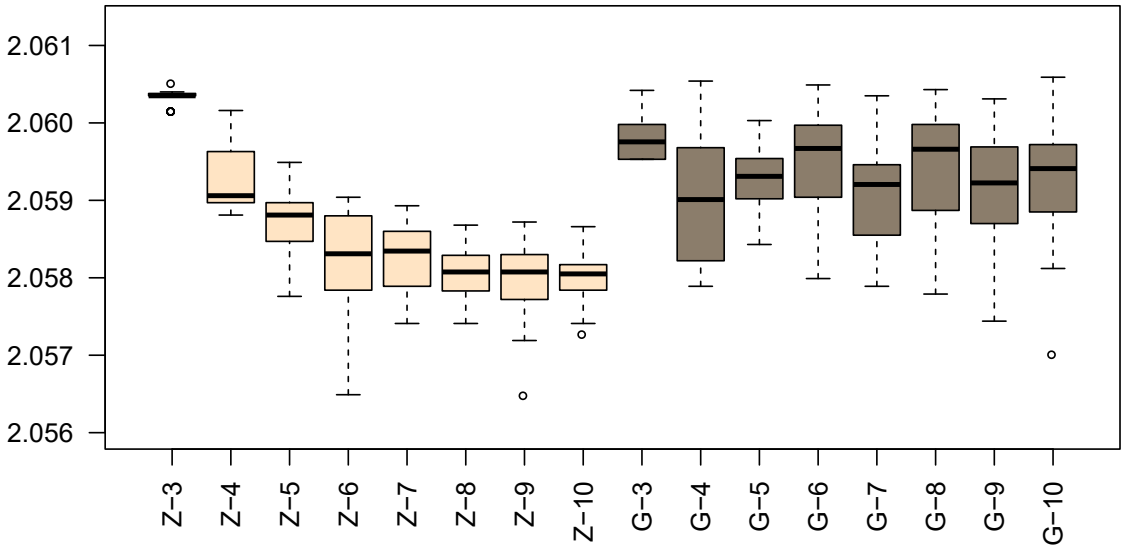
Slika 5.3: Rezultati za metodu jednostavnog kombiniranja

Tablica 5.6: Rezultati postignuti za metodu jednostavnog kombiniranja koristeći F_2

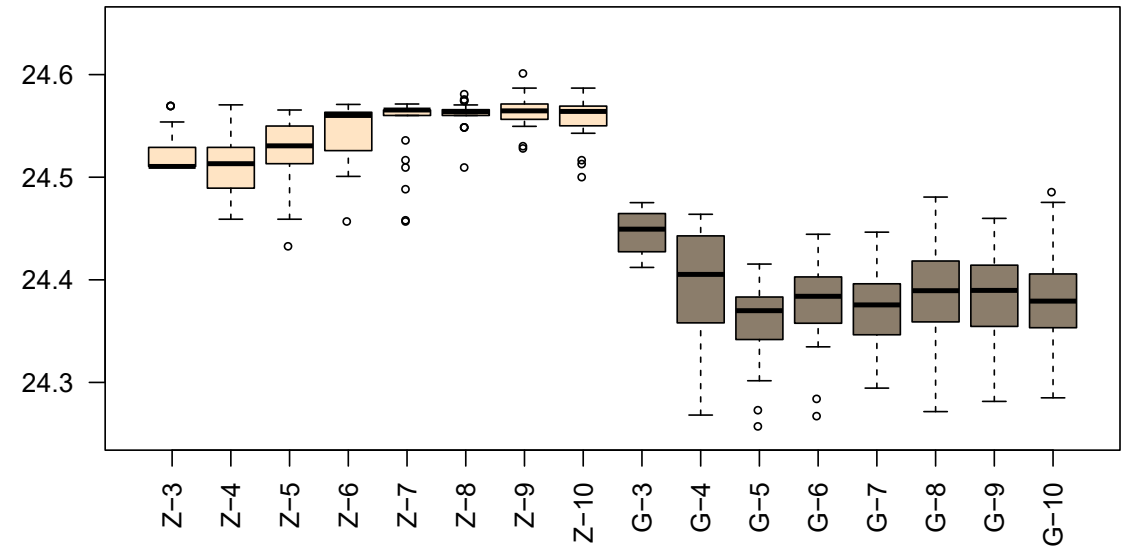
Veličina	metoda zbroja			metoda glasanja		
	min	med	max	min	med	max
2	24.54957	24.54957	**24.54957**	24.54096	24.54096	24.54096
3	24.51049	**24.51049**	24.57044	24.41209	24.44938	24.47521
4	24.45906	24.51318	24.57050	24.26819	24.40528	24.46388
5	**24.43371**	24.53575	24.56549	**24.25819**	**24.36992**	**24.41536**
6	24.45789	24.56017	24.57094	24.28500	24.38528	24.45643
7	24.46400	24.56549	24.58077	24.31608	24.38620	24.44646
8	24.53396	24.56555	24.59042	24.36969	24.43333	24.48066
9	24.53124	24.56658	24.61317	24.33931	24.42658	24.48993
10	24.51472	24.56582	24.61357	24.35872	24.46419	24.50609
pojedinačna pravila	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447

Tablica 5.7: Rezultati ESS-a postignuti za metodu jednostavnog kombiniranja koristeći F_1

Originalna veličina	metoda zbroja			metoda glasanja		
	min	med	max	min	med	max
3	2.06016	2.06035	2.06052	2.05953	2.05976	2.06042
4	2.05881	2.05906	2.06016	2.05789	**2.05901**	2.06054
5	2.05776	2.05881	2.05949	2.05843	2.05931	**2.06003**
6	**2.05649**	2.05831	2.05904	2.05799	2.05967	2.06049
7	2.05741	2.05835	2.05893	2.05789	2.05921	2.06035
8	2.05741	2.05808	2.05868	2.05779	2.05966	2.06043
9	**2.05649**	2.05808	2.05872	2.05744	2.05923	2.06031
10	2.05728	**2.05805**	**2.05866**	**2.05702**	2.05941	2.06059
pojedinačna pravila	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408



(a) funkcija cilja F_1



(b) funkcija cilja F_2

Slika 5.4: Rezultati ESS-a za metodu jednostavnog kombiniranja

Tablica 5.8: Rezultati ESS-a postignuti za metodu jednostavnog kombiniranja koristeći F_2

Originalna veličina	metoda zbroja			metoda glasanja		
	min	med	max	min	med	max
3	24.51049	**24.51049**	24.57044	24.41209	24.44938	24.47521
4	24.45906	24.51318	24.57050	24.26819	24.40528	24.46388
5	**24.43371**	24.53058	**24.56549**	**24.25819**	**24.36992**	**24.41536**
6	24.45789	24.56017	24.57094	24.26819	24.38394	24.44440
7	24.45789	24.56549	24.57134	24.29446	24.37556	24.44646
8	24.51049	24.56383	24.58193	24.27157	24.38942	24.48066
9	24.52907	24.56466	24.60216	24.28151	24.38968	24.45987
10	24.50110	24.56410	24.58685	24.28500	24.37925	24.48649
pojedinačna pravila	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447

5.5.2 BagGP

Za ispitivanje BagGP-a korištene su funkcije cilja F_1 i F_2 te metode zbroja i glasanja. Osim toga, tijekom ispitivanja koristile su se različite veličine skupa za učenje kako bi se utvrdio utjecaj veličine skupa na ostvarene rezultate. Ispitivane su sljedeće veličine skupova za učenje: 100, 150, 200, 250 i 300.

Rezultati ostvareni BagGP-om u ovisnosti o veličini skupa i uz korištenje funkcije F_1 dani su tablicom 5.9. Iz rezultata dobivenih korištenjem funkcije cilja F_1 lako se primjećuje da se korištenjem metode glasanja za kombiniranje pojedinačnih pravila u ansambl postižu bolji rezultati nego korištenjem metode zbroja. Prilikom korištenja metode zbroja za fiksiranu veličinu skupa, rezultati se poboljšavaju povećanjem veličine ansambla do jednog trenutka nakon kojeg medijan počinje ponovno rasti. Poveznica između kvalitete ostvarenih rezultata i veličine skupa za učenje pri korištenju metode zbroja ne može se utvrditi jer se kod nekih veličina ansambla kvaliteta rješenja poboljšava povećanjem veličine skupa za učenje, dok je kod drugih suprotna situacija. Iz prethodnog, može se zaključiti kako rezultati ne ovise samo o veličini skupa, nego da postoji određena koreliranost između veličine skupa za učenje i veličine ansambla te ih je potrebno zajednički gledati.

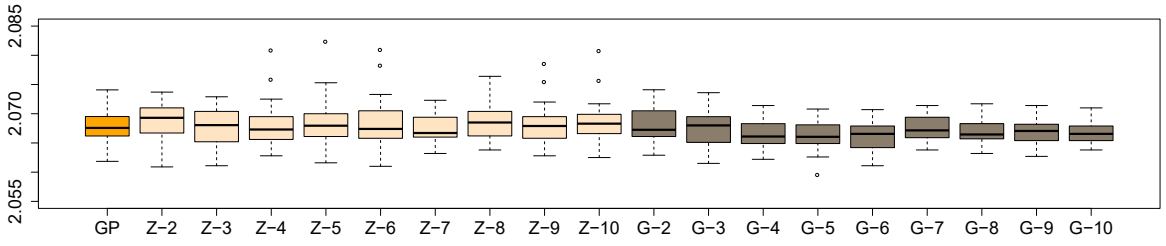
Tablica 5.9: Ostvareni rezultati za BagGP i funkciju F_1

	veličina	100			150			200			250			300		
		min	med	max	min	med	max	min	med	max	min	med	max	min	med	max
metoda zbroja	2	2.06090	2.06930	2.07370	2.06360	2.06820	2.07740	2.06390	2.06855	2.07730	2.06270	2.06890	2.07540	2.06380	2.06855	2.07260
	3	2.06110	2.06805	2.07290	2.06410	2.06810	2.07600	2.06110	2.06910	2.07660	2.06450	2.06870	2.07950	2.06330	2.06835	2.07550
	4	2.06280	2.06730	2.08090	2.06200	2.06875	2.07570	2.06390	2.06790	2.07570	2.06320	2.06930	2.07830	2.06150	2.06820	2.07460
	5	2.06160	2.06795	2.08240	2.06180	2.06885	2.07570	2.06370	2.06920	2.07810	2.06200	2.06925	2.07500	2.06090	2.06690	2.07110
	6	2.06100	2.06740	2.08100	2.06090	2.06700	2.07190	2.06290	2.06685	2.08230	2.06160	2.06750	2.07950	2.06240	2.06675	2.07140
	7	2.06320	2.06670	2.07230	2.06340	2.06745	2.07500	2.06010	2.06820	2.08210	2.06310	2.06745	2.07950	2.06200	2.06860	2.07550
	8	2.06380	2.06850	2.07640	2.06170	2.06715	2.07150	2.06290	2.06910	2.07680	2.06320	2.06830	2.07950	2.06140	2.06740	2.07100
	9	2.06280	2.06790	2.07860	2.06360	2.06780	2.07890	**2.05960**	2.06950	2.07670	2.06170	2.06765	2.07440	2.06160	2.06745	2.07410
	10	2.06250	2.06830	2.08080	2.06330	2.06725	2.07950	2.06340	2.06860	2.07570	2.06330	2.06885	2.07950	**2.05860**	2.06690	2.07540
metoda glasanja	2	2.06290	2.06725	2.07410	2.06290	2.06830	2.07560	2.06450	2.06885	2.07320	2.06480	2.06750	2.07350	2.06340	2.06820	2.07270
	3	2.06150	2.06800	2.07360	**2.05920**	2.06755	2.07250	2.06210	2.06710	2.07130	2.06280	2.06645	2.07230	2.06240	2.06780	2.07340
	4	2.06220	2.06610	2.07140	2.06190	2.06730	2.07250	2.06300	2.06670	**2.06950**	2.06330	2.06755	2.07330	2.06260	2.06705	2.07080
	5	**2.05960**	**2.06605**	2.07080	2.06370	2.06690	2.07160	2.06060	2.06715	2.07250	2.06430	2.06730	2.07240	2.06300	2.06725	2.07310
	6	2.06110	2.06655	**2.07070**	2.06130	2.06680	**2.07060**	2.06240	2.06645	2.07030	**2.05980**	2.06690	2.07090	2.06360	2.06700	2.07050
	7	2.06380	2.06715	2.07140	2.06210	2.06680	2.07120	2.06250	2.06710	2.06970	2.06250	**2.06610**	2.07050	2.06230	2.06665	2.07070
	8	2.06320	2.06645	2.07170	2.06310	**2.06630**	2.07170	2.06190	**2.06640**	2.07010	2.06100	2.06710	**2.06990**	2.06310	**2.06570**	2.07090
	9	2.06270	2.06705	2.07140	2.06280	2.06760	2.07190	2.06030	2.06715	2.07050	2.06460	2.06715	2.07140	2.06200	2.06675	**2.07010**
	10	2.06380	2.06655	2.07100	2.06300	2.06640	**2.07060**	2.06370	2.06680	2.07020	2.06300	2.06675	**2.06990**	2.06290	2.06665	2.07080
pojedinačna pravila		2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408

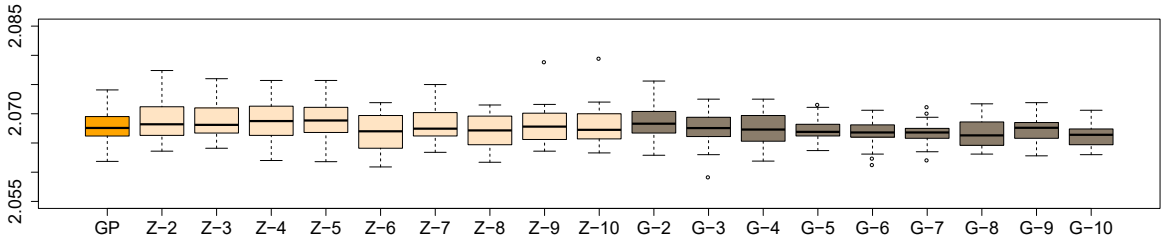
Unatoč tome što se za pojedine veličine ansambla i skupova za učenje uz metodu zbroja postižu rezultati bolji od rezultata ostvarenih pojedinačnim pravilima, ne može se utvrditi statistička značajnost tih razlika. S druge strane, korištenjem metode glasanja za istu tu funkciju cilja, postižu se ne samo rezultati bolji od onih postignutih korištenjem metode zbroja, nego i od pojedinačnih pravila, te se kod većine kombinacija ta razlika pokazala kao statistički značajna. Ni kod metode glasanja ne postoji jasan utjecaj veličine skupa za učenje na ostvarene rezultate. Kao i kod metode zbroja, iz rezultata se može primijetiti da kvaliteta rješenja ovisi o kombinaciji veličine ansambla i veličine skupa za učenje zbog čega je potrebno njihove vrijednosti tražiti zajedno, a ne fiksirajući jednu od ovih vrijednosti. Prilikom odabira kako veličine skupa za učenje tako i veličine ansambla potrebno je uzeti u obzir da se njihovim povećanjem povećava i vrijeme potrebno za kreiranje ansambla.

Ostvareni rezultati za funkciju F_1, u ovisnosti o veličini skupa za učenje, prikazani su kutijastim dijagramima na slici 5.5 . Ansambli koji su nastali kombiniranjem pojedinačnih pravila metodom zbroja na slici su označeni sa Z, a metodom glasanja s G. Dodatno, uz obje oznake stoji broj koji označava veličinu ansambla, a rezultati ostvareni pojedinačnim pravilima su označeni s GP. Na osnovu rezultata u tablici 5.9 i grafičkog prikaza rezultata danog na slici 5.5, kao najbolja kombinacija za funkciju F_1 i metodu zbroja izabrana je veličina skupa za učenje 300 i veličina ansambla 5, a za metodu glasanja odabrana je ista veličina ansambla uz veličinu skupa za učenje od 100 instanci.

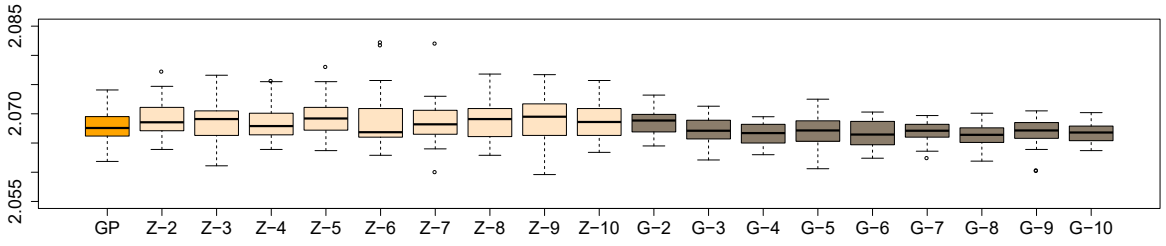
Slična situacija je i s rezultatima ostvarenim korištenjem funkcije cilja F_2 koji su dani tablicom 5.10 i prikazani, u ovisnosti o veličini skupa za učenje, kutijastim dijagramima na slici 5.6. Metoda zbroja i u ovom slučaju postiže lošije rezultate od metode glasanja te su, u odnosu na pojedinačna pravila, postignuta tek neznatna poboljšanja za mali broj kombinacija veličine skupa za učenje i veličine ansambla. Metoda glasanja ostvaruje nešto bolje rezultate, no ni s njom se ne postižu statistički značajno bolji rezultati od pojedinačnih pravila. Iako je medijan u većini promatranih kombinacija manji od onog ostvarenog pojedinačnim pravilima, ta razlika nije velika. S druge strane, ni metoda zbroja ni metoda glasanja u većini slučaja nisu uspjele ostvariti minimum manji od onog postignutog pojedinačnim pravilima. Kao i kod korištenja funkcije cilja F_1, veličinu skupa za učenje i veličinu ansambla potrebno je tražiti zajednički. Na osnovu tablice s ostvarenim rezultatima i danih dijagrama za metodu zbroja odabran je ansambl veličine 10, a za metodu glasanja veličine 5 dok je za obje metode odabrana veličina skupa za učenje 150.



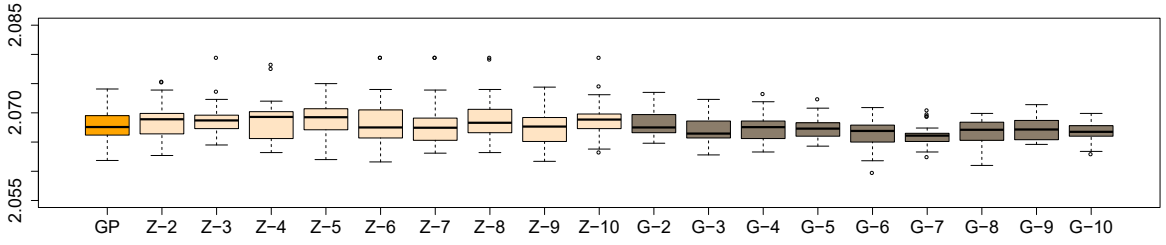
(a) veličina skupa 100



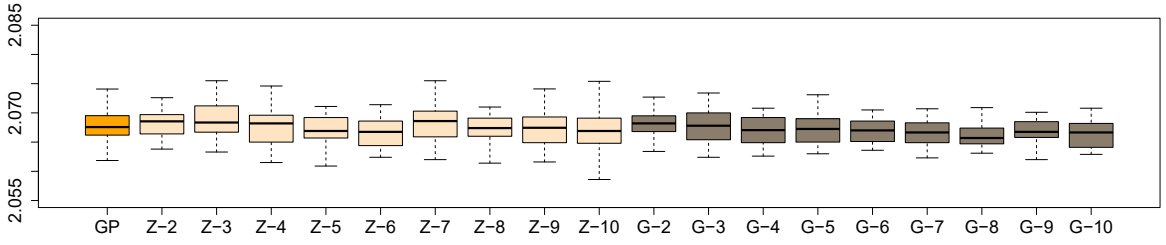
(b) veličina skupa 150



(c) veličina skupa 200



(d) veličina skupa 250



(e) veličina skupa 300

Slika 5.5: Rezultati za BagGP uz korištenje funkcije F_1 u ovisnosti o veličini skupa

Tablica 5.10: Ostvareni rezultati za BagGP i funkciju F_2

	veličina	100			150			200			250			300		
		min	med	max	min	med	max	min	med	max	min	med	max	min	med	max
metoda zbroja	2	24.69800	25.05600	25.42600	24.76400	25.07450	25.54600	24.68000	25.09350	25.73900	24.58600	25.00550	25.29800	24.79200	25.06700	25.44200
	3	24.75200	24.98350	25.76800	24.69800	25.07500	25.73500	**24.51400**	25.00200	25.66700	24.61200	25.00800	25.24100	**24.55600**	25.05500	25.40900
	4	24.59800	24.96500	25.69600	24.66600	24.97650	25.43100	24.62000	24.97200	25.28000	24.57900	24.96950	25.58800	24.61500	24.94150	25.34400
	5	24.68800	25.05600	25.82600	24.69200	24.96500	25.45000	24.59100	25.09200	25.64700	24.57700	**24.93000**	25.43000	24.71300	25.01600	25.49800
	6	**24.56300**	25.08200	25.63700	24.76900	25.06000	25.60700	24.72000	25.08700	25.47900	24.62200	25.02850	25.29300	24.57500	25.02800	25.40500
	7	24.62600	24.95950	25.33200	24.61600	24.95700	25.53700	24.68400	24.98950	25.73300	**24.56300**	24.98850	25.73100	24.74100	25.01600	25.16300
	8	24.67400	25.04700	25.72400	24.81700	25.00850	25.51300	24.66700	25.03900	25.57700	24.56900	24.94800	25.71400	24.73700	25.00300	25.31800
	9	24.70600	24.96550	25.64600	24.71100	25.09350	25.32400	24.67700	25.05700	25.64800	24.65200	25.03750	25.37200	24.70700	25.05250	25.34600
	10	24.62900	25.06850	25.67800	24.69200	24.95500	25.42600	24.57900	24.96900	25.63900	24.62200	25.02450	25.72500	24.57600	25.02050	25.42400
metoda glasanja	2	24.78900	25.11800	25.45800	24.81700	25.03850	25.52800	24.76700	25.09700	25.39800	24.66600	25.03600	25.32700	24.66500	25.05500	25.49500
	3	24.64400	24.94100	25.27400	24.58200	24.99100	25.30300	24.72300	24.97400	25.26700	24.63100	24.95350	25.25700	24.60300	25.00000	25.22000
	4	24.63500	24.96050	**25.21300**	24.67100	24.94250	**25.18600**	24.62400	24.95400	**25.17600**	24.61100	24.93650	25.23400	24.65400	24.92700	25.19800
	5	24.65500	24.96000	25.30400	24.70200	24.94400	25.22200	24.71200	**24.92200**	25.28700	24.66000	24.97700	25.34400	24.70800	24.95250	25.29200
	6	24.61700	**24.91250**	25.22700	24.69400	24.99500	25.43400	24.57500	24.96400	25.19700	24.66700	24.94050	25.20000	24.74200	25.01650	25.38900
	7	24.70800	24.96550	25.29100	**24.56700**	24.98100	25.25300	24.64700	24.96850	25.24400	24.75000	24.98450	**25.16400**	24.65300	24.96100	25.28000
	8	24.74200	24.99500	25.24100	24.67400	24.96900	25.21400	24.69700	24.97450	25.17900	24.73900	24.96000	25.23200	24.62900	24.95750	**25.15500**
	9	24.65400	25.01600	25.26700	24.60100	**24.92300**	25.31000	24.69300	24.93500	25.20900	24.76800	24.98850	25.17200	24.64800	24.98900	25.20400
	10	24.65700	25.01500	25.33000	24.58300	24.98600	25.20000	24.70800	25.00450	25.24500	24.68600	24.96300	25.27400	24.71100	**24.92700**	25.19300
pojedinačna pravila		24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447

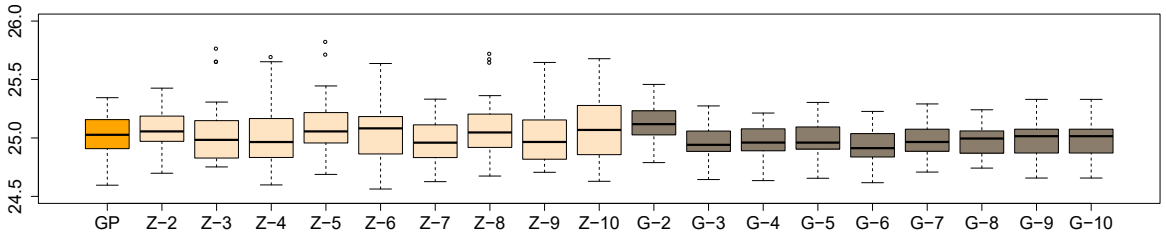
Rezultati ostvareni primjenom ESS-a na ansamble dobivene BagGP-om dani su tablicom 5.11 za funkciju cilja F_1 i tablicom 5.12 za funkciju cilja F_2. Primjena ESS-a kod BagGP-a, neovisno o funkciji cilja koja se koristi, donosi znatno poboljšanje u odnosu na ostvarene rezultate bez ESS-a. S obzirom da se pravila koja se nalaze u ansamblu razvijaju međusobno neovisno ova pojava je očekivana. S druge strane, ono što je bitnije je da rezultati ansambala nakon ESS-a postižu i rezultate bolje od pojedinačnih pravila i to u većini slučaja. Kod funkcije F_1 tek u dvije promatrane kombinacije tako kreirani ansambl nije uspio ostvariti manji minimum od onog postignutog pojedinačnim pravilima. Za funkciju F_2 takvih kombinacija je nešto više, te kod nje postoji i jedna kombinacija u kojoj je ostvareni maksimum nešto veći od onog ostvarenog pojedinačnim pravilima. No, najvažnije, za sve kombinacije neovisno o korištenoj funkciji cilja ostvaren je manji medijan od onog ostvarenog korištenjem pojedinačnih pravila. Za daljnje usporedbe BagGP-a s ESS-om za funkciju F_1 kod metode zbroja odabrana je kombinacija veličine ansambla 9 i veličine skupa za učenje 300, a za metodu glasanja veličina ansambla 9 i veličina skupa za učenje 150. Za funkciju F_2 kod obje metode kombiniranja odabrana je kombinacija veličine ansambla 10 i skupa za učenje veličine 200.

5.5.3 BoostGP

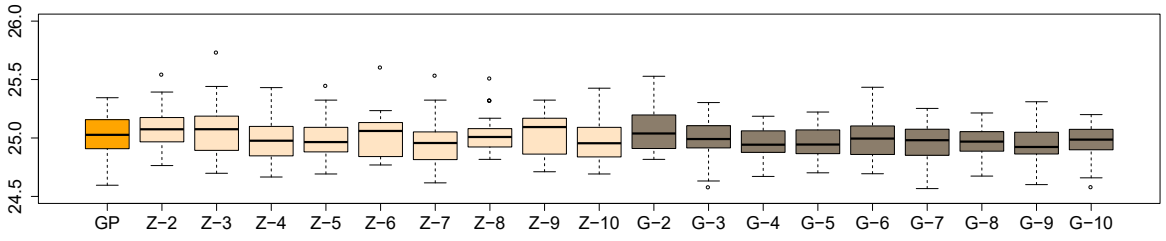
U ispitivanju BoostGP-a uz metode zbroja i glasanja za kombiniranje pojedinačnih pravila u ansambl korištene su i njihove težinske verzije. Kod težinskih metoda zbroja i glasanja kao težine korištene su pouzdanosti pravila koje se dobiju prilikom razvoja pravila. Rezultati BoostGP-a uz korištenje funkcije F_1 dani su tablicom 5.13. Prikaz rezultata kutijastim dijagramima dan je na slici 5.7 gdje je oznaka za metodu zbroja Z, metodu glasanja G te broj uz njih označava veličinu ansambla.

Kao i kod BagGP-a, možemo primijetiti da metoda glasanja, bez obzira radi li se o onoj bez ili s težinama, ostvaruje bolje rezultate od metode zbroja. Metoda glasanja, kao i njezina težinska verzija, ostvaruju bolje rezultate od pojedinačnih pravila te je razlika u rezultatima statistički značajna. Postignuti medijani za sve veličine ansambala, osim za veličinu 2 kod metode glasanja, bolji su od onih postignutih pojedinačnim pravilima. Iako je medijan za veličinu 2 veći od medijana pojedinačnih pravila, rezultati ostvareni ansamblom ove veličine manje su raspršeni te imaju manju minimalnu i maksimalnu vrijednost.

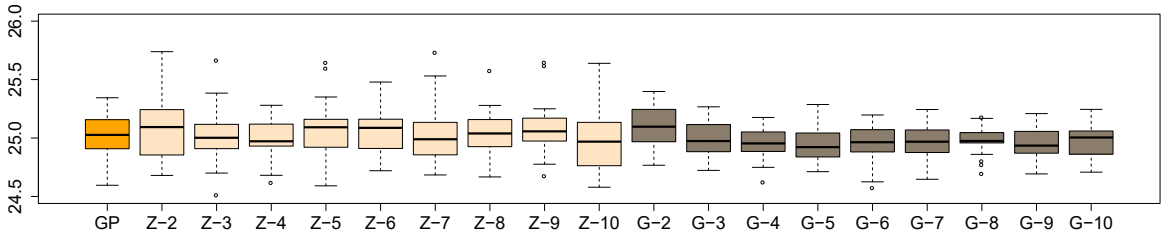
Situacija kod metode zbroja i njezine težinske verzije je nešto drugačija te se bolji medijan od onog ostvarenog pojedinačnim pravilima postiže za veličine 2, 7, 9 i 10. Manja minimalna vrijednost kod metode zbroja postiže se tek kod nekoliko veličina ansambla, dok se manja maksimalna vrijednost postiže za većinu ispitanih veličina.



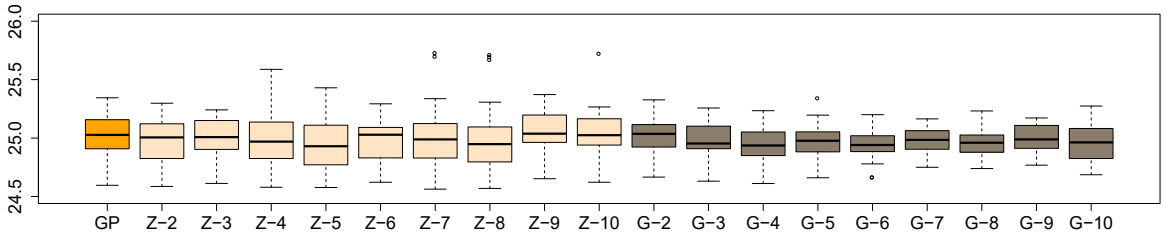
(a) veličina skupa 100



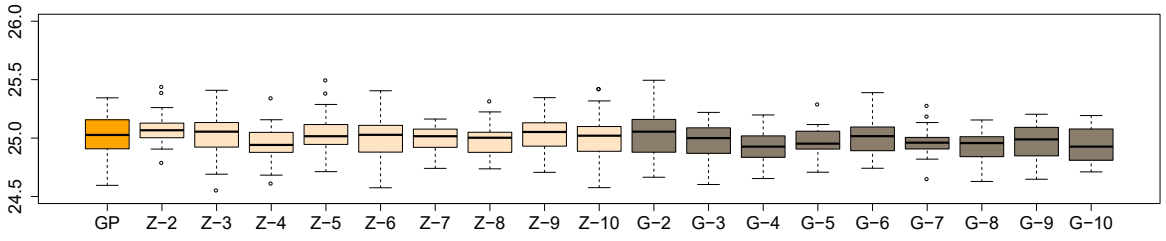
(b) veličina skupa 150



(c) veličina skupa 200



(d) veličina skupa 250



(e) veličina skupa 300

Slika 5.6: Rezultati za BagGP uz korištenje funkcije F_2 u ovisnosti o veličini skupa

Tablica 5.11: Ostvareni rezultati ESS-a za BagGP i funkciju F_1

	originalna		100			150			200			250			300	
	veličina	min	med	max	min	med	max	min	med	max	min	med	max	min	med	max
metodazbroja	3	2.06106	2.06671	2.07077	2.06162	2.06675	2.07181	2.06110	2.06640	2.07170	2.06365	2.06592	2.06978	2.06276	2.06653	2.07155
	4	2.06035	2.06552	2.06988	2.06070	2.06554	2.06961	2.06177	2.06598	2.06979	2.06066	2.06616	2.06994	2.06121	2.06455	2.06888
	5	2.05941	2.06462	2.06999	2.06081	2.06402	2.06825	2.06014	2.06470	2.06743	2.06144	2.06465	2.06851	2.06089	2.06518	2.06770
	6	2.06026	2.06410	2.06740	2.06068	2.06334	2.06679	2.05967	2.06402	2.06742	2.05823	2.06349	2.06684	2.06071	2.06343	2.06759
	7	2.06079	2.06349	2.06641	2.05981	2.06371	2.06568	2.05936	2.06342	2.06636	2.06038	2.06414	2.06663	2.06012	2.06265	2.06626
	8	2.05958	2.06273	2.06557	2.05894	2.06272	2.06666	2.05941	2.06326	2.06535	2.05928	2.06285	2.06530	2.06039	2.06276	2.06560
	9	2.05901	2.06188	2.06505	2.05917	2.06145	2.06527	2.05982	2.06173	2.06480	2.05844	2.06171	2.06452	2.05885	2.06110	**2.06374**
	10	2.06023	2.06284	2.06460	2.05992	2.06227	2.06528	**2.05827**	2.06267	2.06500	2.05968	2.06188	2.06530	2.05863	2.06253	2.06604
metoda glasanja	3	2.06152	2.06576	2.07098	2.05915	2.06623	2.06995	2.06207	2.06616	2.07010	2.06279	2.06612	2.07072	2.06017	2.06629	2.07103
	4	2.06072	2.06500	2.06825	2.06117	2.06399	2.06850	2.06010	2.06506	2.06809	2.05963	2.06420	2.06703	2.06086	2.06453	2.06672
	5	2.05942	2.06323	2.06760	2.06152	2.06369	2.06564	2.05905	2.06379	2.06737	2.06040	2.06388	2.06664	2.05881	2.06328	2.06684
	6	2.06077	2.06300	2.06557	2.06078	2.06328	2.06503	2.05968	2.06223	2.06581	2.05958	2.06272	2.06549	2.05872	2.06189	2.06491
	7	2.05897	2.06249	2.06491	2.05885	2.06222	2.06579	2.06061	2.06193	2.06512	2.05866	2.06238	2.06551	2.06000	2.06167	2.06545
	8	2.06044	2.06314	2.06533	2.05935	2.06307	2.06627	2.05874	2.06224	2.06487	2.05934	2.06230	2.06459	2.05999	2.06213	2.06550
	9	2.05747	2.06126	2.06345	2.05912	2.06121	2.06378	2.05855	2.06145	2.06417	2.05934	2.06159	2.06397	**2.05762**	2.06114	2.06419
	10	**2.05700**	**2.06094**	**2.06283**	**2.05756**	**2.06069**	**2.06272**	2.05836	**2.06091**	**2.06308**	**2.05811**	**2.06100**	**2.06303**	2.05829	**2.06031**	2.06381
pojedinačna pravila		2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408	2.06185	2.06757	2.07408

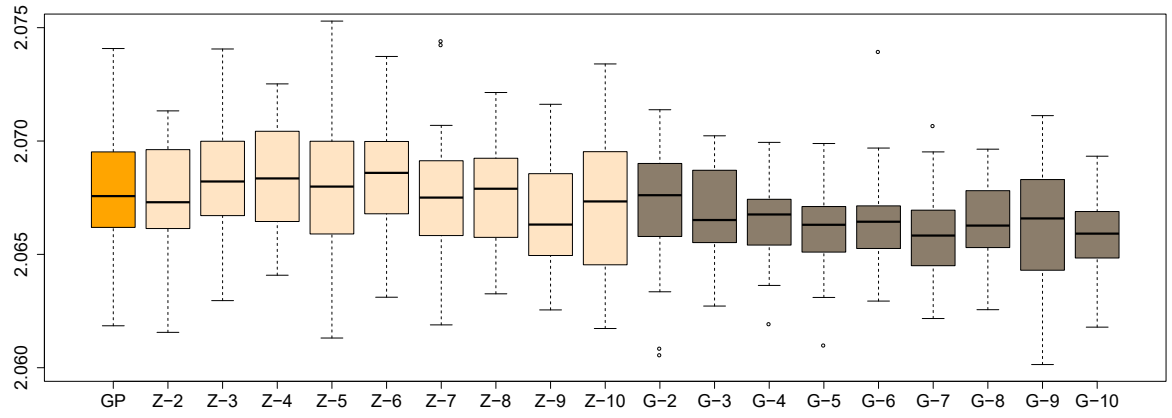
Tablica 5.12: Ostvareni rezultati ESS-a za BagGP i funkciju F_2

	originalna		100			150			200			250			300	
	veličina	min	med	max	min	med	max	min	med	max	min	med	max	min	med	max
metoda zbroja	3	24.57035	24.92342	25.17559	24.69477	24.91931	25.19116	24.51426	24.97770	25.40448	24.56933	24.90498	25.14105	24.55559	24.91443	25.26096
	4	24.59767	24.81955	25.10043	24.52320	24.85227	25.03966	24.61197	24.86491	25.19244	24.56288	24.81182	25.10638	24.46450	24.84669	25.01600
	5	24.55718	24.80455	25.04491	24.60424	24.82155	24.98110	24.60424	24.82155	24.98110	24.53899	24.77819	24.98982	24.66901	24.87113	25.02313
	6	24.44383	24.77242	25.06295	24.54598	24.77155	24.94630	24.55930	24.79438	25.09204	24.47399	24.74337	24.90640	24.43731	24.77872	25.10059
	7	24.56288	24.72224	24.93660	24.45400	24.74545	25.08456	24.60101	24.78282	24.98884	24.52658	24.64783	24.91675	24.41240	24.67921	25.03679
	8	24.52903	24.71126	24.87634	24.59933	24.73855	24.90188	24.57048	24.73552	25.04506	24.48414	24.70602	24.89019	24.40772	24.69614	24.86308
	9	24.50727	24.67938	24.88802	**24.38803**	24.70604	24.96815	24.50644	24.66961	24.93210	24.43196	24.68042	24.87871	24.46836	24.65437	24.88327
	10	24.54043	24.69368	24.85542	24.48243	24.65646	24.84812	24.48597	24.68062	24.83734	24.47257	24.61865	24.91346	24.37436	24.62089	24.85290
metoda glasanja	3	24.64414	24.85831	25.27353	24.58245	24.88539	25.01933	24.64791	24.90436	25.13596	24.63107	24.92518	25.11436	24.60327	24.87704	25.15831
	4	24.62824	24.82669	25.06912	24.59007	24.82246	25.11632	**24.39474**	24.78215	25.02811	24.50729	24.84353	25.00820	24.57422	24.76926	24.93432
	5	24.50916	24.75178	25.05408	24.40797	24.75848	24.98383	24.40797	24.75848	24.98383	24.58113	24.78202	24.97157	24.59235	24.76802	24.90438
	6	24.41081	24.70731	24.89911	24.47750	24.69526	24.95812	24.51777	24.69664	24.92218	24.55052	24.70827	24.89059	24.46966	24.70008	24.88646
	7	24.46284	24.66662	24.82438	24.42403	24.68631	24.91361	24.51388	24.67829	24.85391	24.50444	24.67406	24.82030	24.44576	24.64514	24.80385
	8	24.49739	24.61200	24.85038	24.45681	24.66817	24.81892	24.43316	24.65610	24.86793	24.42349	24.67391	24.83343	24.47602	24.63355	24.76326
	9	24.45401	24.58804	**24.77312**	24.44521	24.58551	24.74442	24.46876	24.61593	24.72881	24.45844	24.62823	24.87891	**24.34455**	24.60220	24.74846
	10	**24.36328**	**24.55740**	24.78389	24.45248	**24.55602**	**24.71479**	24.43878	**24.57650**	**24.71650**	**24.40314**	**24.60088**	**24.70815**	24.43497	**24.56887**	**24.69575**
pojedinačna pravila		24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447	24.59576	25.02683	25.34447

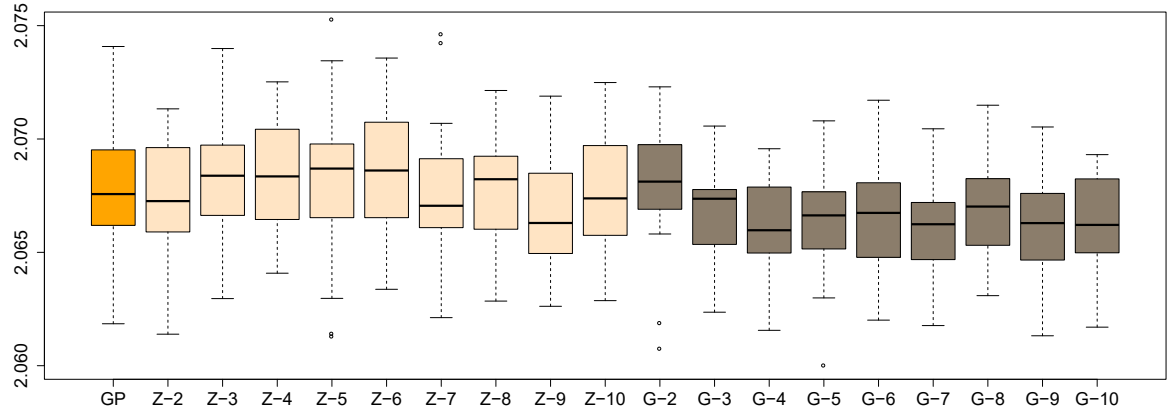
Ako usporedimo rezultate dobivene korištenjem težina i bez njihovog korištenja, možemo primijetiti da u njima ne postoji značajna razlika. Njihovo međusobno odstupanje ovisi o veličini ansambla, odnosno kod nekih ansambala neznatno su bolji rezultati dobiveni korištenjem težina prilikom kombiniranja, a kod nekih su neznatno bolji oni u kojima se težine nisu koristile. U skladu s tim možemo zaključiti da težine ne utječu bitno na postignuti rezultat te zbog jednostavnijeg računanja možemo koristiti metode zbroja i kombiniranja bez težina kao i kod ostalih metoda. Za daljnju usporedbu za metodu zbroja i njezinu težinsku verziju odabrana je veličina ansambla 9, a za metodu glasanja i njezinu težinsku verziju veličina 10.

Tablica 5.13: Ostvareni rezultati za BoostGP i funkciju F_1

	Veličina	metoda zbroja			metoda glasanja		
		min	med	max	min	med	max
bez težine	2	2.06156	2.06730	**2.07133**	2.06057	2.06761	2.07138
	3	2.06296	2.06822	2.07406	2.06272	2.06652	2.07023
	4	2.06408	2.06835	2.07252	2.06194	2.06676	2.06994
	5	**2.06131**	2.06799	2.07529	2.06100	2.06631	2.06989
	6	2.06311	2.06860	2.07373	2.06294	2.06644	2.07395
	7	2.06189	2.06751	2.07442	2.06217	**2.06583**	2.07068
	8	2.06326	2.06790	2.07214	2.06256	2.06627	2.06964
	9	2.06255	2.06632	2.07162	2.06014	2.06659	2.07112
	10	2.06173	2.06734	2.07340	2.06179	2.06592	2.06933
s težinama	2	2.06139	2.06726	**2.07133**	2.06077	2.06812	2.07230
	3	2.06296	2.06838	2.07399	2.06236	2.06737	2.07057
	4	2.06408	2.06835	2.07252	2.06156	2.06598	2.06957
	5	**2.06131**	2.06870	2.07529	**2.06003**	2.06663	2.07080
	6	2.06337	2.06861	2.07357	2.06201	2.06674	2.07171
	7	2.06212	2.06706	2.07464	2.06177	2.06624	2.07045
	8	2.06285	2.06823	2.07214	2.06309	2.06702	2.07149
	9	2.06262	**2.06630**	2.07189	2.06132	2.06629	2.07053
	10	2.06287	2.06738	2.07249	2.06170	2.06621	**2.06931**
pojedinačna pravila		2.06185	2.06757	2.07408	2.06185	2.06757	2.07408



(a) bez težina



(b) s težinama

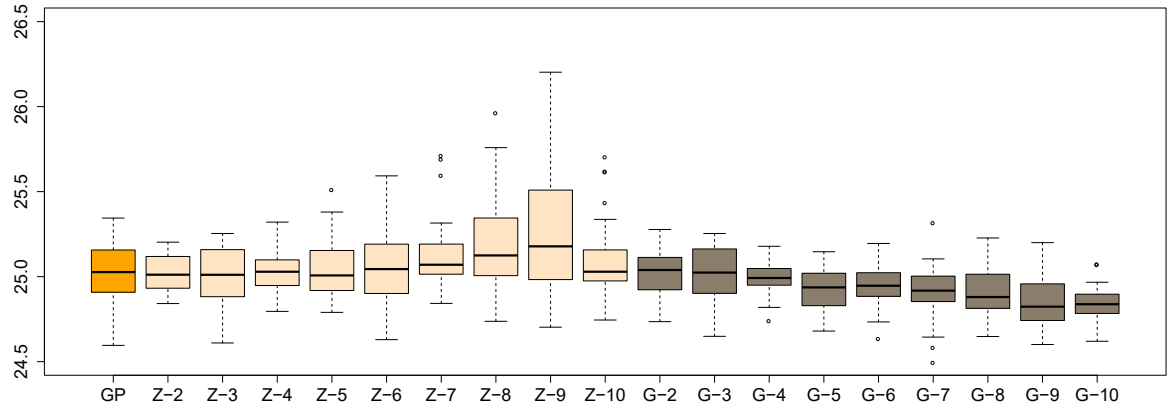
Slika 5.7: Rezultati za BoostGP uz korištenje funkcije F_1

Rezultati za BoostGP ostvareni korištenjem funkcije cilja F_2 dani su tablicom 5.14 i kutijastim dijagramima na slici 5.8. Rezultati ostvareni ovim pristupom ne postižu za sve veličine ansambla značajno bolje rezultate od pojedinačnih pravila, štoviše, kod metode zbroja poboljšanje se postiže tek za nekolicinu veličina. Nešto bolja situacija je kod metode glasanja kod koje se ostvaruje manja vrijednost medijana u odnosu na onaj ostvareni pojedinačnim pravilima za sve veličine osim za veličinu 2 u kojoj se ne koristi težina. Iz danih dijagrama lako je uočljivo da u obje verzije metode zbroja povećanjem veličine ansambla dolazi do većeg raspršenja rezultata kao i njihovog pogoršanja. S druge strane, povećanje veličine ansambla kod metode glasanja neznatno povećava raspršenost rezultata, ali i dovodi do njihovog poboljšanja. Za daljnju usporedbu u obje verzije metode zbroja koristit će se veličina ansambla 3, dok će se kod metode glasanja koristiti veličina 10, odnosno veličina 7 za njezinu težinsku verziju.

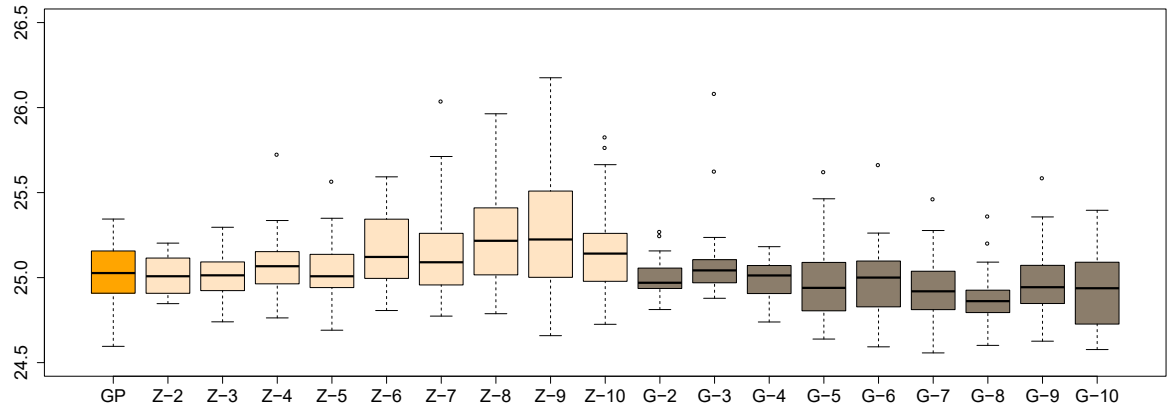
Rezultati ESS-a primjenjenog na ansamble kreirane BoostGP-om dani su tablicom 5.15 za funkciju F_1 i 5.16 za funkciju F_2. ESS, slično kao i kod BagGP-a, postiže statistički značajna

Tablica 5.14: Ostvareni rezultati za BoostGP i funkciju F_2

	Veličina	metoda zbroja			metoda glasanja		
		min	med	max	min	med	max
bez težine	2	24.84171	25.01145	25.20284	24.73539	25.03909	25.27744
	3	24.60998	25.01090	25.25381	24.64896	25.02374	25.25381
	4	24.79572	25.02884	25.32071	24.74098	24.99190	25.17855
	5	24.78990	**25.00711**	25.51187	24.68000	24.93668	25.14644
	6	24.62950	25.04444	25.59276	24.63576	24.94670	25.19525
	7	24.84263	25.07009	25.71240	**24.49576**	24.91766	25.31712
	8	24.73725	25.12469	25.96359	24.64787	24.88015	25.22748
	9	24.70275	25.17817	26.20244	24.60077	**24.82337**	25.20007
	10	24.74495	25.02898	25.70464	24.62000	24.83778	**25.07465**
s težinama	2	24.84695	25.00806	**25.20284**	24.81220	24.96974	25.27128
	3	24.60998	25.01090	25.25381	24.64896	25.02374	25.25381
	4	24.79572	25.02884	25.32071	24.74098	24.99190	25.17855
	5	24.69071	25.00799	25.56689	24.63869	24.93964	25.62285
	6	24.80643	25.12154	25.59276	24.59295	25.00022	25.66438
	7	24.77361	25.09034	26.03847	24.55723	24.91947	25.46331
	8	24.78777	25.21658	25.96359	24.60145	24.86162	25.36222
	9	24.65860	25.22437	26.17547	24.62615	24.94360	25.58709
	10	24.72543	25.14132	25.82743	24.57692	24.93756	25.39603
pojedinačna pravila		24.59576	25.02683	25.34447	24.59576	25.02683	25.34447



(a) bez težina



(b) s težinama

Slika 5.8: Rezultati za BoostGP uz korištenje funkcije F_2

poboljšanja u rezultatima. Kod funkcije F_1 nakon primjene ESS-a, ansambli ostvaruju bolje rezultate od onih ostvarenih pojedinačnim pravilima i to za sve veličine ansambla. Slično vrijedi i kod funkcije F_2 u kojoj također dolazi do značajnijih poboljšanja u ostvarenim rezultatima nakon korištenja ESS-a. Iz tablice s rezultatima, može se uočiti da kod nekih veličina i nakon primjene ESS-a nije postignuta bolja minimalna vrijednost nego ona ostvarena kod pojedinačnih pravila. No, ono što je bitnije, i u tim situacijama medijan je manji od onog ostvarenog pojedinačnim pravilima, što znači da je došlo do poboljšanja rezultata. Za daljnju usporedbu BoostGP-a s ESS-om kod funkcije F_1 i F_2 za sve četiri vrste kombiniranja pravila u ansambl odabrana je veličina 10.

Tablica 5.15: Ostvareni rezultati ESS-a za BoostGP i funkciju F_1

	Originalna veličina	metoda zbroja			metoda glasanja		
		min	med	max	min	med	max
bez težine	3	2.06112	2.06621	2.06866	2.06272	2.06567	2.06876
	4	2.06064	2.06509	2.07043	2.05831	2.06446	2.06723
	5	2.06068	2.06297	2.06751	2.06059	2.06356	2.06605
	6	2.06082	2.06358	2.06783	2.06113	2.06264	2.06559
	7	2.05927	2.06309	2.06660	2.05935	2.06174	2.06373
	8	2.06026	2.06292	2.06492	2.05845	2.06143	2.06547
	9	2.05853	2.06205	**2.06441**	2.05866	2.06085	2.06304
	10	2.05979	**2.06164**	2.06474	2.05775	2.06011	2.06270
s težinama	3	2.06080	2.06614	2.06866	2.06236	2.06628	2.07057
	4	2.06026	2.06514	2.07043	2.06092	2.06456	2.06825
	5	2.06123	2.06337	2.06751	2.05876	2.06314	2.06603
	6	2.06027	2.06347	2.06783	2.05932	2.06295	2.06623
	7	2.05918	2.06307	2.06660	2.05936	2.06175	2.06471
	8	2.06103	2.06289	2.06502	2.05977	2.06196	2.06532
	9	**2.05784**	2.06206	**2.06441**	2.05793	2.06095	2.06360
	10	2.05966	2.06181	2.06516	**2.05676**	**2.05997**	**2.06202**
pojedinačna pravila		2.06185	2.06757	2.07408	2.06185	2.06757	2.07408

Tablica 5.16: Ostvareni rezultati ESS-a za BoostGP i funkciju F_2

	Originalna veličina	metoda zbroja			metoda glasanja		
		min	med	max	min	med	max
bez težine	3	24.65115	24.94735	25.23110	24.71066	24.90082	25.06751
	4	24.70359	24.90302	25.10372	24.54822	24.82931	25.01765
	5	24.64081	24.83583	25.00313	24.52163	24.73250	24.92966
	6	24.60266	24.79384	24.95095	24.47932	24.66127	24.83057
	7	24.54521	24.77732	24.91948	24.36975	24.60410	24.80443
	8	**24.49419**	24.79374	25.03595	24.39763	24.60610	24.70990
	9	24.55460	24.73720	**24.86847**	24.40190	24.56585	24.76385
	10	24.49712	**24.67417**	24.90582	24.40390	24.54870	**24.66587**
s težinama	3	24.69179	24.94192	25.23708	24.74284	24.96886	25.14518
	4	24.61338	24.87592	25.06236	24.71946	24.91000	25.05945
	5	24.63241	24.80593	25.01299	24.57132	24.78429	25.03858
	6	24.58649	24.80359	24.93158	24.38909	24.71765	24.98661
	7	24.59359	24.80013	24.94519	24.47356	24.67482	24.88505
	8	24.52115	24.74940	24.87060	24.43887	24.61200	24.75464
	9	24.51611	24.72537	24.88048	24.40659	24.58785	24.82621
	10	24.51599	24.70842	24.88105	**24.35393**	**24.54140**	24.70126
pojedinačna pravila		24.59576	25.02683	25.34447	24.59576	25.02683	25.34447

5.5.4 Kooperativna koevolucija

Kooperativna koevolucija predstavlja vremenski najzahtjevniji pristup kreiranja ansambla razmatran u ovoj disertaciji. Prioritetna pravila koja su članovi ansambla razvijaju se istovremeno, bez međusobnog utjecaja osim kod evaluacije kada se evaluiraju zajednički. Zbog istovremenog razvoja svih članova ansambla ovaj proces traje dugo, dok je paralelizacija moguća tek djelomično. Kod razvoja ovih pravila, kao i kod svih prethodno korištenih pristupa, korišteni su parametri iz tablice 4.20. Kako je nakon kreiranja svake generacije pravila potrebno evaluirati zajednički, što nije slučaj kod razvoja standardnih pravila, jedno pokretanje za bilo koju veličinu ansambla traje više nego razvoj isto toliko pojedinačnih pravila. Smanjenje broja generacija, i na taj način smanjenje vremena potrebnog za kreiranje ansambla, dovelo bi do dodatnog pogoršanja ostvarenih rezultata jer pravilo ne bi imalo dovoljno vremena za učenje. Vrijeme izvršavanja na računalu prosječnih performansi za veći dio veličina ansambala traje više od 24 sata, zbog čega je napravljeno samo 5 pokretanja za funkciju F_1.

Rezultati ostvareni ovim pristupom uz korištenje funkcije F_1 dani su tablicom 5.17. Iako zbog broja pokretanja nije moguće govoriti o statistički značajnim rezultatima, lako se uočava da bez obzira na korištenu metodu kombiniranja ovaj pristup postiže lošije rezultate od onih ostvarenih pojedinačnim pravilima. Primjenom ESS-a na nastale ansamble dobivamo rezultate dane tablicom 5.18. I ovi rezultati, iako nešto bolji od onih ostvarenih bez primjene ESS-a, ne predstavljaju poboljšanje pojedinačnih prioritetnih pravila. Zbog loše kvalitete ostvarenih rezultata i nemogućnosti ostvarenja boljih rezultati od onih postignutih pojedinačnim pravilima, ovaj pristup neće se koristiti u daljnjim usporedbama.

5.5.5 Usporedba rezultata

Usporedba načina kreiranja ansambla dana je tablicom 5.19 za funkciju F_1 i tablicom 5.20 za funkciju F_2. Uz dosad korišten način prikazivanja rezultata dodan je stupac *značajnost* u koji se bilježi statistička značajnost u razlici rezultata. Ako MWW test pokazuje da kreirani ansambl ostvaruje statistički bolji rezultat od pojedinačnih pravila (p-vrijednost < 0.05) u stupac značajnost stavljen je znak $+$, ako je pojedinačno pravilo statistički značajno bolje od kreiranog ansambla znak $-$ i ako se statistički ne može utvrditi značajnost u razlici znak $\approx$. Usporedba ansambla, u ovisnosti o načinu kreiranja ansambla i funkciji cilja, s pojedinačnim pravilima prikazana je i kutijastim dijagramom na slici 5.9. Na slici 5.9 ispred metode kreiranja nalazi se slovo Z ukoliko je korištena metoda zbroja kod kombiniranja pojedinačnih pravila u ansambl i slovo G za metodu glasanja. Osim toga, kod BoostGP-a u slučaju da su korištene težine stavljen je znak $+$.

Za funkciju cilja F_1 svi ansambli daju jednak ili bolji rezultat od pojedinačnih pravila. Statistički značajno poboljšanje u rezultatima ostvaruje se za sve načine kreiranja osim kod BagGP-a

Tablica 5.17: Ostvareni rezultati za kooperativnu koevoluciju i funkciju F_1

Veličina	metoda zbroja			metoda glasanja		
	min	med	max	min	med	max
2	2.07099	2.07307	2.07795	2.06960	2.07416	2.07955
3	2.06952	2.07932	2.15177	2.06420	2.07388	2.07788
4	2.07546	2.07749	2.07920	2.07107	2.07238	2.08934
5	2.07056	2.08889	2.23607	2.06772	2.07405	2.07781
6	2.07852	2.08201	2.22024	2.06950	2.07182	2.08953
7	2.07833	2.12359	2.17812	2.06766	2.07351	2.07836
8	2.08072	2.10902	2.23156	2.06867	2.07608	2.07824
9	2.07854	2.08282	2.09300	2.06880	2.07388	2.08025
10	2.06903	2.08534	2.15413	2.07730	2.08348	2.09502
pojedinačna pravila	**2.06185**	**2.06757**	**2.07408**	**2.06185**	**2.06757**	**2.07408**

Tablica 5.18: Ostvareni rezultati ESS-a za kooperativnu koevoluciju i funkciju F_1

Originalna veličina	metoda zbroja			metoda glasanja		
	min	med	max	min	med	max
3	2.06557	2.07233	2.07884	2.06420	**2.06718**	2.07788
4	2.07381	2.07625	2.07749	2.06795	2.07186	2.08781
5	2.06426	2.07366	2.07693	2.06772	2.07060	2.07466
6	2.06793	2.07013	2.07759	2.06461	2.06930	2.08009
7	2.06601	2.07394	2.07789	2.06466	2.06883	**2.07079**
8	2.06916	2.07128	2.07236	2.06382	2.06733	2.07212
9	2.06371	2.07288	2.07590	2.06697	2.06799	2.07118
10	2.06679	2.06913	**2.07271**	2.06931	2.07606	2.08277
pojedinačna pravila	**2.06185**	**2.06757**	2.07408	**2.06185**	2.06757	2.07408

i BoostGP-a (s težinom i bez težine) za metodu zbroja, odnosno BagGP-a kod metode glasanja. Najbolji rezultati ostvareni su s ansamblima kreiranim jednostavnom metodom kombiniranja, dok su najlošiji ostvareni s BagGP-om. Slično se događa i kod funkcije cilja F_2 gdje je ansambl kreiran metodom jednostavnog kombiniranja ostvario najbolje rezultate. Za ovu funkciju, ansambl kreiran BagGP-om uz korištenje metode zbroja postiže rezultat koji je statistički značajno lošiji od onog ostvarenog pojedinačnim pravilima, a BagGP s primjenom ESS-a, BoostGP (s težinama i bez težina) i BoostGP bez težina na koji je primijenjen ESS daju statistički slične rezultate kao one koje ostvaruju pojedinačna pravila. No, ako gledamo metodu glasanja i funkciju F_2, tada primjećujemo da bez obzira na način kreiranja, ansambl uvijek ostvaruje statistički značajno bolje rezultate od onih ostvarenih pojedinačnim pravilima.

Metoda jednostavnog kombiniranja za funkciju F_1 pokazuje bolje rezultate ako se pojedinačna pravila kombiniraju metodom zbroja, od onih koji se postižu korištenjem metode glasanja. No, s druge strane, veličina ansambla korištenog kod metode zbroja je 8 što ansambl čini dvostruko većim od onog korištenog kod metode glasanja, pa samim time i vrijeme izrade rasporeda je nešto veće od onog kod metode glasanja. Ova metoda, osim po dobrim rezultatima, od drugih se izdvaja i zbog činjenice da se za nju mogu koristiti već razvijena prioritetna pravila, te da ih nije potrebno ponovno razvijati. Iako korištenje već razvijenih pravila smanjuje potrebno vrijeme za izvršavanje, potrebno je u obzir uzeti i broj ansambla koji se kreiraju, a koji može bitno utjecati na vrijeme izvršenja. Primjena ESS-a na metodu jednostavnog kombiniranja ne vodi poboljšanju ostvarenih rezultata te oni najčešće ostaju vrlo slični onim ostvarenim bez primjene ESS-a. Razlog ovakvom ponašanju je činjenica da je krajnji ansambl izabran na temelju velikog broja kreiranih ansambla te na njegovu kvalitetu utječe svako pojedino pravilo koje se nalazi u ansamblu, odnosno izbor pravila koja se nalaze u ansamblu nije neovisan te uklanjanjem jednog od njih može doći do pogoršanja rezultata. Upravo zbog toga, primjena ESS-a, osim što ne utječe znatno na kvalitetu rješenja, ne dovodi ni do značajnog smanjenja ansambla, nego se najčešće zadržava originalni ansambl i nakon primjene ESS-a. Slično ponašanje kod metode jednostavnog kombiniranja primjećujemo i za funkciju F_2 kod koje su također rezultati ostvareni korištenjem metode zbroja nešto bolji od onih ostvarenih metodom glasanja te primjena ESS-a ne dovodi do poboljšanja ostvarenih rezultata.

BagGP je metoda kojom su ostvareni najlošiji rezultati. Rezultati dobiveni korištenjem metode zbroja za kombiniranje pravila unutar ansambla su raspršeniji od onih dobivenih metodom glasanja za obje funkcije cilja. Primjena ESS-a na BagGP dovodi do znatnih poboljšanja u rezultatima. Jedan od razloga što dolazi do poboljšanja krije se u činjenici da su pravila unutar ansambla neovisno razvijana i dodavana u ansambl dok on nije postao željene veličine. Neovisnim dodavanjem pravila u ansambl može doći i do dodavanja pravila koje ne doprinose kvaliteti ansambla, nego ga čak i čine lošijim. Pronalazak optimalnog podskupa na skupu za provjeru dovodi do uklanjanja takvih pravila iz ansambla te se primjenom ESS-a ostvareni rezultat po-

boljšava.

BoostGP kreira ansamble čiji su rezultati bolji ako se za kombiniranje pojedinačnih pravila koristi metoda glasanja, nego ako se koristi metoda zbroja. Ostvareni rezultati za obje funkcije cilja manje su raspršeni nego što je to slučaj kod BagGP-a. Korištenjem pouzdanosti dobivenih razvojem pravila unutar ansambla kao težina u kombiniranju ne postižu se značajno različiti rezultati od onih dobivenih kombiniranjem bez težina. Slično kao i kod BagGP-a, zbog neovisnog razvoja pravila koje čine ansambl primjena ESS-a dovodi do poboljšanja rezultata.

Na temelju prethodnog razmatranja i prikazanih rezultata, lako se uočava kako metoda jednostavnog kombiniranja daje značajno bolje rezultate kako od onih koji su ostvareni pojedinačnim pravilima, tako i od rezultata postignutih ansamblima kreiranim korištenjem ostalih načina. Dodatna prednost ove metode je što kod nje nije potrebno razvijati nova pravila ili praviti replike skupa za učenje kao što je to slučaj kod BagGP-a ili modificirati težine pojedinih instanci kao što je to slučaj kod BoostGP-a. S druge strane, primjena ESS-a na ansamble kreirane ovom metodom nije potrebna, jer ona ne dovodi do poboljšanja rezultata, dok ako koristimo preostala dva načina kreiranja, svakako je preporučljivo nakon nastanka ansambla primijeniti ESS-a.

Uspoređivanjem metode glasanja i metode zbroja kao načina kombiniranja pojedinačnih pravila u ansambl nije moguće doći do generalnog zaključka ako se pri tome ne uzima u obzir način kreiranja ansambla. Ako je ansambl kreiran metodom jednostavnog kombiniranja preporučljivo je koristiti metodu zbroja koja ostvaruje bolje rezultate za obje funkcije cilja od rezultata ostvarenih metodom glasanja, dok kod BagGP-a i BoostGP-a bolji rezultati ostvaruju se kod metode glasanja.

Veličinu ansambla također je potrebno gledati u ovisnosti o načinu kreiranja te se za metodu jednostavnog kombiniranja mogu koristiti već veličine 3, 4 i 5, dok za BagGP i BoostGP se s većim veličinama poput 9 i 10 postižu bolji rezultati. Što se tiče primjene ESS-a ona najbolje rezultate daje za ansamble većih veličina što je i očekivano jer u tom slučaju ima više podskupova između kojih bira onaj najbolji. S druge strane, kod ESS-a u obzir treba uzeti i da što je ansambl veći vrijeme potrebno za pronalazak optimalnog podskupa je značajno veće.

5.6 Zaključak

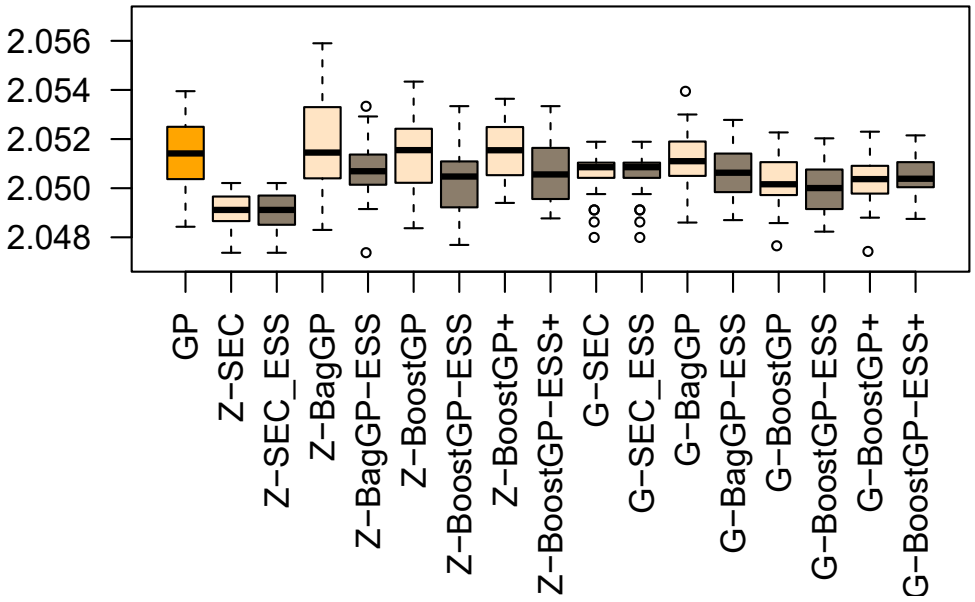
U ovom poglavlju napravljena je detaljna analiza oblikovanja ansambla prioritetnih pravila koja su rijetko korištena u postupcima raspoređivanja. Kod oblikovanja ansambla razmatrani su načini kombiniranja prioritetnih pravila u ansambl te načini kreiranja ansambala. U ispitivanjima unutar ovog poglavlja korišteno je 4 različite metode iz literature: metoda jednostavnog kombiniranja, BagGP, BoostGP i kooperativna koeveloucija. S obzirom da dosad u literaturi nije zabilježena primjena ansambla na RCPSP, sve nabrojane metode dodatno su prilagođene za RCPSP.

Tablica 5.19: Usporedba načina kreiranja ansambla za funkciju cilja F_1

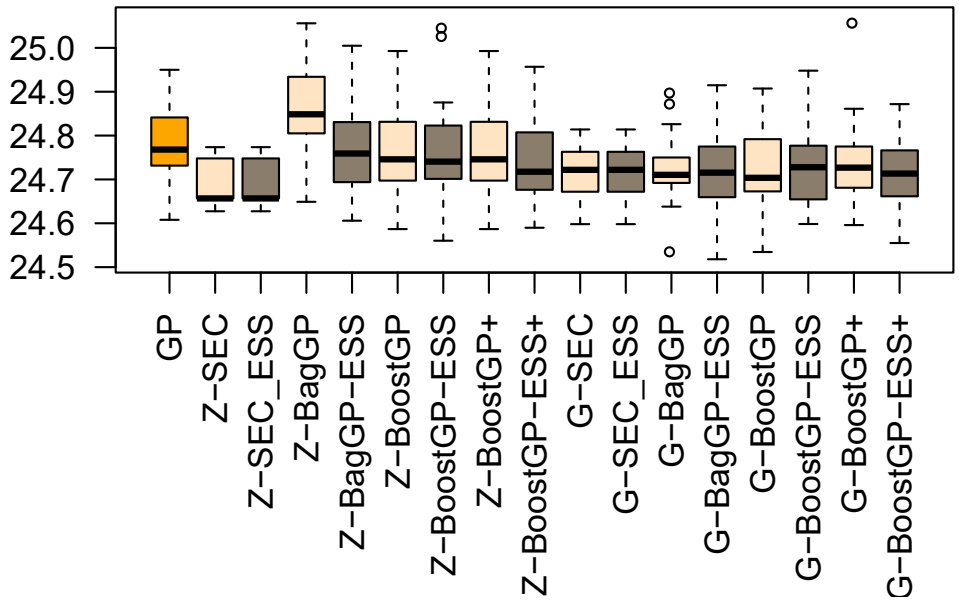
Metoda	min	med	max	značajnost
metoda zbroja				
SEC-8	**2.04737**	**2.04912**	**2.05021**	+
SEC_ESS-8	**2.04737**	**2.04912**	**2.05021**	+
BagGP-5-300	2.04830	2.05145	2.05590	≈
BagGP_ESS-9-300	2.04743	2.05069	2.05339	+
BoostGP-9	2.04940	2.05146	2.05373	≈
BoostGP_ESS-10	2.04769	2.05048	2.05334	+
BoostGP_težine-9	2.04940	2.05155	2.05364	≈
BoostGP_težine_ESS-10	2.04877	2.05056	2.05334	+
metoda glasanja				
SEC-4	2.04805	2.05086	2.05189	+
SEC_ESS-4	2.04805	2.05086	2.05189	+
BagGP-5-100	2.04860	2.05110	2.05400	≈
BagGP_ESS-10-150	2.04870	2.05063	2.05278	+
BoostGP-10	2.04771	2.05016	2.05227	+
BoostGP_ESS-10	2.04823	2.05001	2.05203	+
BoostGP_težine-10	2.04748	2.05037	2.05230	+
BoostGP_težine_ESS-10	2.04875	2.05039	2.05215	+
GP	2.04843	2.05142	2.05395	

Tablica 5.20: Usporedba načina kreiranja ansambla za funkciju cilja F_2

Metoda	min	med	max	značajnost
metoda zbroja				
SEC-3	24.62744	**24.65668**	**24.77359**	+
SEC_ESS-3	24.62744	**24.65668**	**24.77359**	+
BagGP-10-150	24.64900	24.84850	25.05600	-
BagGP_ESS-10-200	24.60594	24.75905	25.00482	≈
BoostGP-3	24.58074	24.79902	25.06196	≈
BoostGP_ESS-10	24.56024	24.74054	25.04802	≈
BoostGP_težine-3	24.58670	24.74583	24.99277	≈
BoostGP_težine_ESS-10	24.58979	24.71755	24.95681	+
metoda glasanja				
SEC-5	24.59805	24.72185	24.81392	+
SEC_ESS-5	24.59805	24.72185	24.81392	+
BagGP-6-150	24.53800	24.71050	24.90000	+
BagGP_ESS-10-200	**24.51799**	24.71545	24.91475	+
BoostGP-10	24.53451	24.70370	24.90730	+
BoostGP_ESS-10	24.59837	24.72776	24.94795	+
BoostGP_težine-7	24.59592	24.72685	25.05947	+
BoostGP_težine_ESS-10	24.55497	24.71336	24.87168	+
GP	24.60789	24.76803	24.94999	



(a) funkcija cilja F_1



(b) funkcija cilja F_2

Slika 5.9: Usporedba rezultata ostvarenih različitim načinima kreiranja ansambla

Nakon što je ansambl kreiran nekom od ovih metoda, prioritetna pravila sadržana u njemu kombinirana su metodom zbroja i metodom glasanja. Dodatno za BoostGP uvedena je i metoda težinskog zbroja i težinskog glasanja u kojoj su kao težine korištene vrijednosti pouzdanosti pravila izračunate tijekom razvoja pojedinih pravila. Dobiveni rezultati pokazuju kako korištenje ansambla može dodatno poboljšati kvalitetu rezultata ostvarenih pojedinačnim pravilima. Upotrebom metode jednostavnog kombiniranja te dodatno prilagođenih BagGP-a i BoostGP-a ostvaruju se statistički značajno bolji rezultati od onih ostvarenih korištenjem pojedinačnih pravila, dok korištenje kooperativne koevolucije ne dovodi do poboljšanja. Zbog ovog razloga kooperativna koevolucija nije dodatno uspoređivana s ostalim metodama.

S obzirom da je veličina ansambla bila unaprijed određena, nakon kreiranja ansambla postupkom ESS-a, dodatno je tražen optimalan podskup pravila koje ga čine. Iako je ovaj postupak u literaturi već korišten kod ansambla, unutar ovog poglavlja napravljena je njegova određena promjena. U ovom poglavlju korišteni ESS nije tražio podskup unaprijed određene veličine, kao što je dosad to bio slučaj, nego je pretraživao podskupe svih veličina. Ovako postavljen ESS je vremenski zahtjevniji, ali na ovaj način se ne sprječava uklanjanje iz ansambla svih pravila koja ne doprinose njegovoj kvaliteti te se s druge strane ne zahtijeva uklanjanje nekog pravila ako su sva pravila važna za ansambl. ESS se pokazao koristan kod onih metoda kreiranja ansambla kod kojih se pravila razvijaju neovisno jedno o drugima. U skladu s prvim razmatranjem, može se zaključiti da je ESS dobro koristiti kod BagGP-a i BoostGP-a, dok kod metode jednostavnog kombiniranja on ne dovodi do poboljšanja rezultata.

Što se tiče metode kombiniranja pojedinačnih pravila u ansambl, iz rezultata je vidljivo da je kod metode jednostavnog kombiniranja bolje koristiti metodu zbroja dok je kod preostala dva načina bolje koristiti metodu glasanja. Veličina ansambla koju treba koristiti također ovisi o načinu kreiranja te je za metodu jednostavnog kombiniranja moguće koristiti manju veličinu, dok je za BagGP i BoostGP bolje koristiti ansamble veće veličine. Isto tako, za primjenu ESS-a bolje je imati ansambl veće veličine jer se tada ispituje više podskupova pa postoji mogućnost pronalaska boljeg podskupa, no s druge strane povećanjem veličine ansambla povećava se i potrebno vrijeme za pronalazak optimalnog podskupa pravila.

Iako je u ovom poglavlju dana detaljna analiza primjene ansambla na RCPSP i dalje je ostalo puno otvorenih tema za buduća istraživanja. Neke od njih su primjena novih načina kreiranja ansambla i primjena određenih informacija sadržanih u instancama problema koja se nalaze unutar skupa za učenje prilikom kreiranja ansambla. S druge strane, i u ovom poglavlju korištene načine kreiranja moguće je dodatno prilagoditi kako bi ostvarili bolje rezultate. Također, moguće je u obzir uzeti i kompleksnost dobivenih ansambala dodavanjem kompleksnosti kao kriterija optimizacije. Osim toga, moguće je proučavati i brojne druge kriterije optimizacije i to samostalno ili u nekoj međusobnoj kombinaciji što će dovesti do višekriterijske optimizacije koja nije proučavana u sklopu ove disertacije.

U budućim istraživanjima potrebno je dodatno proučiti i pristup kooperativne koevolucije koja nije dala dobar rezultat u korištenju kod ovog problema. Mogućnost da ona s nekim drugim postavkama daje bolje rezultate leži u činjenici da u literaturi postoje primjeri njezine primjene na druge probleme u kojima ostvaruje dobre rezultate. Dodatno, potrebno je proučiti dolazi li kod nje do prenaučenosti koja bi onda mogla biti razlog ostvarenju loših rezultata, kao i na koji način smanjiti vrijeme izvršavanja potrebno za ovaj pristup.

Mjesta za dodatna poboljšanja ima i kod BagGP-a i BoostGP-a. Kod BagGP je moguće izučiti druge veličine skupova za učenje kao i načine stvaranja replika skupa za učenje, dok se kod BoostGP-a u ispitivanjima može na neki drugi način računati pouzdanost pravila ili mijenjati težina pojedine instance za učenje tijekom procesa razvoja pravila.

Poglavlje 6

Prilagodba prioritetnih pravila za primjenu u statičkim okruženjima

Jednostavnost, kao i njihova mogućnost brzog odgovara na promjene nastale unutar sustava, čine prioritetna pravila pogodnim za primjenu u dinamičkim okruženjima. Unatoč tome što su prvenstveno dizajnirana za korištenje u dinamičkim uvjetima, ona se mogu koristiti i u statičkim uvjetima, posebno u situacijama kada je vrijeme potrebno za dobivanje rješenja važnije od dobrote dobivenog rješenja. Jedna od prednosti prioritetnih pravila je i što se njihovim korištenjem raspored izgrađuje iterativno te raspoređene aktivnosti mogu početi sa svojim izvršavanjem dok se ostatak rasporeda još uvijek gradi. Glavni nedostatak prioritetnih pravila je kvaliteta rješenja koja može biti značajno lošija u odnosu na onu koja se postiže ostalim metaheurističkim pristupima. Razlog tome je što se kod metaheurističkih pristupa svaka instanca problema rješava posebno, dok se kod prioritetnih pravila jednim pravilom (ili s više njih u slučaju ansambala) rješava cijeli skup instanci problema.

Prilagodba pravila raspoređivanja za problem nesrodnih strojeva u statičkom okruženju detaljno je analizirana u disertaciji [44]. Unutar disertacije analizirano je uvođenje novih statičkih značajki, pogled unaprijed (engl. *look-ahead*), iterativna pravila raspoređivanja, pristup rollout te njihove kombinacije. Uvođenje novih značajki koje su specifične za statička okruženja može donijeti nove informacije prilikom raspoređivanja te utjecati na kvalitetu rješenja, dok pogled unaprijed omogućuje sustavu da prilikom raspoređivanja u obzir uzme i aktivnosti koje će u skorijem vremenskom periodu biti dostupne za raspoređivanje. Iterativna pravila raspoređivanja raspored izgrađuju više puta, pri čemu u svakoj iteraciji u obzir uzimaju informacije iz prethodne iteracije s ciljem poboljšanja rješenja, dok pristup rollout prilikom odluke koju aktivnost sljedeću rasporediti prati više smjerova rasporeda, odnosno u obzir uzima više aktivnosti kao mogući izbor za rasporediti te računa rješenje za svaku od njih te nastavlja s onom aktivnošću koja vodi k najboljem rješenju. Unutar ovog poglavlja analizirat će se iterativna pravila raspoređivanja i pristup rollout s ciljem dodatnog poboljšanja razvijenih prioritetnih pravila.

6.1 Iterativna prioritetna pravila

Iterativna prioritetna pravila (engl. *iterative priority rules*, IPR) predstavljaju pristup u kojem se raspored gradi kroz iteracije pri čemu se u svakoj iteraciji u obzir uzimaju informacije dobivene iz prethodne iteracije. Raspored se gradi sve dok se njegova kvaliteta ne prestane poboljšavati. Pseudokod prilagođene sheme za raspoređivanje dan je algoritmom 13 te se ova shema kod IPR-a koristi i prilikom razvoja pravila umjesto standardnih verzija SGS-a. Bitno je uočiti kako se kod ove prilagođene sheme raspoređivanja prilikom završetka ne vraća zadnji kreirani raspored nego raspored iz predzadnje iteracije jer je njegova dobrota veća od dobrote rasporeda iz posljednje iteracije.

Algoritam 13 SGS za raspoređivanje korištenjem IPR-a

1: **Ulaz:** inicijalne vrijednosti P_0 skupa parametara P čije vrijednosti se prate kroz iteracije
2: $P \leftarrow P_0$
3: dobrota $\leftarrow \infty$
4: raspored$\leftarrow \emptyset$
5: najbolji_raspored$\leftarrow \emptyset$
6: **čini**
7: najbolji_raspored$\leftarrow$ raspored
8: raspored$\leftarrow$ raspored dobiven korištenjem SGS-a i prioritetnog pravila
9: dobrota*$\leftarrow$ dobrota
10: dobrota $\leftarrow$ vrijednost dobrote dobivenog rasporeda
11: izračunati nove vrijednosti za skup parametara P na temelju dobivenog rasporeda i spremiti ih u varijablu P
12: **dok** dobrota* $>$ dobrota
13: vrati najbolji_raspored

S obzirom da se prilikom generiranja novog rasporeda koriste informacije iz prethodne iteracije, potrebno je uvesti određeni skup parametara preko kojih će se te informacije prenijeti iz jedne iteracije u druge. Najlakši način da se vrijednosti prenesu u drugu iteraciju je da ti parametri budu uključeni u pravilo u obliku značajki. Zbog toga je potrebno uvesti nove značajke koje će se koristiti kod razvoja pravila. Uvođenje novih značajki omogućuje da se prilikom razvoja pravila informacije iz prethodne iteracije iskoriste na najbolji način. Značajke koje treba uvesti ovise o samom problemu koji se rješava, kao i o kriteriju optimizacije. U skladu s tim, za RCPSP i za funkcije cilja F_1 i F_2 uvedene su tri nove značajke dane u tablici 6.1. Uvedene značajke sadrže jednostavne informacije kao što su vrijeme završetka aktivnosti, vrijeme čekanja aktivnosti na raspoređivanje, odnosno vrijeme proteklo od trenutka kad je aktivnost postala dostupna do trenutka kada je raspoređena te udaljenost trajanja projekta od donje granice problema koja se dobije ispuštanjem uvjeta na sredstva uzimajući u obzir samo uvjete prednosti. Ove značajke odabrane su jer njihove vrijednosti bitno utječu na ostvareno rješenje. Na primjer, vrijeme završetka aktivnosti utječe na ukupno trajanje projekta, a upravo to trajanje je jedina promjenjiva vrijednost za funkciju F_1. Isto tako i vrijeme čekanja aktivnosti može utjecati na

ukupno trajanje projekta. S druge strane, za funkciju F_2 je bitna udaljenost od donje granice, s obzirom da je upravo ta udaljenost, do na svojevrsnu normalizaciju, kriterij optimizacije. Vrijednosti ovih značajki računaju se na temelju vrijednosti ostvarenih u prethodnoj iteraciji, zbog čega se one moraju inicijalizirati za prvu iteraciju. Za sve tri uvedene značajke, manje vrijednosti dovode do kvalitetnijeg rješenja, pa se njihova inicijalna vrijednost, kako bi se spriječilo zaustavljanje raspoređivanja već u prvoj iteraciji, postavlja na broj koji je veći od onih vrijednosti koje mogu biti ostvarene za pojedinu značajku. Na primjer, za inicijalnu vrijednost može se odabrati gornja granica problema koja se izračuna na način da se zbroje trajanja svih aktivnosti iz projekta, odnosno koja se dobije raspoređivanjem u kojem se svaka aktivnost izvršava sama bez mogućnosti da se neka druga aktivnost izvršava u istom vremenu.

Tablica 6.1: Dodatne značajke za IPR

Značajka	**Opis**
ActFin	vrijeme završetka aktivnosti u prethodnom rasporedu
ActWait	vrijeme čekanja aktivnosti na raspoređivanje u prethodnom rasporedu
LBDist	udaljenost trajanja projekta od donje granice problema u prethodnom rasporedu

6.2 Pristup rollout

Rollout je jednostavni pristup koji se primjenjuje kod različitih heurističkih metoda s ciljem postizanja boljih rezultata [132, 133, 134]. Ovaj pristup kombinira iscrpno pretraživanje i metaheurističke metode. Ideja koja leži iza ovog pristupa je primjenjivanjem iscrpnog pretraživanja u nekim dijelovima algoritma postići rezultate bolje nego metaheurističke metode i to u vremenu kraćem od vremena potrebnog za iscrpno pretraživanje. Kako bi se to ostvarilo, ovaj pristup u trenutku odluke u obzir uzima sve mogućnosti, no za razliku od iscrpnog pretraživanja, prilikom određivanja koja od tih odluka je najbolja ne nastavlja dalje provoditi iscrpno pretraživanje, nego preostale odluke donosi koristeći zadanu heuristiku. Nakon što se na temelju rezultata ostvarenih heuristikom odabere najbolja mogućnost, nastavlja se daljnje raspoređivanje do sljedećeg trenutka odluke, kada se postupak ponavlja.

S obzirom na jednostavnost ovog pristupa i činjenicu da je u literaturi moguće pronaći dobre rezultate ostvarene primjenom ovog pristupa na problem raspoređivanja, vrijedno je ispitati može li on poboljšati kvalitetu rješenja postignutu primjenom prioritetnih pravila na RCPSP. Kako je RCPSP različit od problema na koje je dosad primjenjivan ovaj pristup, potrebno je prilagoditi u literaturi korišteni algoritam. Pseudokod prilagođenog algoritma dan je algoritmom 14. Raspored se konstruira počevši od praznog rasporeda, odnosno nultog vremenskog

Algoritam 14 Pristup rollout

```
1: Ulaz: n - broj aktivnosti koje se uzimaju u obzir prilikom odluke, π - prioritetno pravilo
2: vrijeme ← 0
3: prethodna_dobrota← ∞
4: najbolja_dobrota ← ∞
5: najbolji_raspored ← ∅
6: dok postoje neraspoređene aktivnosti čini
7:    raspored ← najbolji_raspored
8:    postavi vrijeme na sljedeći vremenski trenutak u kojem postoji dostupna aktivnost
9:    koristeći π izračunaj prioritete dostupnih aktivnosti A_i
10:   poredaj aktivnosti s obzirom na prioritete
11:   za aktivnost A_i, i = 1,...,n čini
12:      dodaj aktivnost A_i u raspored
13:      koristeći prioritetno pravilo π konstruiraj raspored do kraja
14:      dobrota← dobrota rasporeda
15:      ako je dobrota < najbolja_dobrota tada
16:         najbolja_dobrota←dobrota
17:         aktivnost*← A_i
18:      završi
19:   završi
20:   ako je najbolja_dobrota < prethodna_dobrota tada
21:      prethodna_dobrota ← najbolja_dobrota
22:      dodaj aktivnost* u najbolji_raspored
23:   inače
24:      dodaj aktivnost s najvećim prioritetom u najbolji_raspored
25:   završi
26: završi
27: vrati najbolji_raspored
```

trenutka te se koraci ponavljaju sve dok postoji neka neraspoređena aktivnost. Ako u vremenskom trenutku koji se promatra ne postoji aktivnost koja se može rasporediti, potrebno je prijeći na prvi sljedeći u kojem ona postoji. Nakon toga, koristeći dano prioritetno pravilo računaju se prioriteti za dostupne aktivnosti te se one sortiraju po njegovoj vrijednosti. Nakon što su aktivnosti sortirane, za unaprijed određeni broj aktivnosti, potrebno je koristeći prioritetno pravilo konstruirati raspored koji nastaje ako se rasporedi određena aktivnost. Nakon što se konstruiraju svi rasporedi, odabire se aktivnost čijim raspoređivanjem se dobije najbolji raspored. Ako je vrijednost dobrote rasporeda bolja nego dobrota do tada najboljeg rasporeda, odabrana aktivnost se raspoređuje, a u suprotnom raspoređuje se aktivnost s najvećim prioritetom. Tijekom cijelog ovog postupka vodi se računa da se najbolji raspored pamti te se on vraća kao konačno rješenje. Što je veći broj aktivnosti koje se uzimaju u obzir prilikom odlučivanja, postoji veća mogućnost postizanja boljeg rezultata. S druge strane, povećanje tog broja dovodi i do povećanja vremena potrebnog za konstruiranje rasporeda.

6.3 Rezultati

U ovom poglavlju bit će prikazani rezultati dobiveni iterativnim prioritetnim pravilima i pristupom rollout. Ako to nije u tekstu drugačije naglašeno, prikazani rezultati dobiveni su u 30 pokretanja te se značajnost u razlici ostvarenih rezultata razvijenim prioritetnim pravilima i korištenjem standardne sheme za generiranje rasporeda i ova dva pristupa ispituje statističkim MWW testom na razini značajnosti 0.01. Ispitivanja su provedena za funkcije cilja F_1 i F_2 koje su korištene i u prethodnim poglavljima.

S obzirom da se raspoređivanje provodi u statičkim uvjetima te je moguće primijeniti i metaheurističke metode, za usporedbu, osim pravila razvijenih GP-om, koristit će se i GA. Parametri korišteni u GA dani su tablicom 6.2 te su jednaki onima u radu [135].

Tablica 6.2: Vrijednosti parametara korištenih u GA

parametar	**vrijednost**
prikaz	broj s pomičnom točkom
broj generacija	80
veličina populacije	1000
vjerojatnost mutacije	0.7

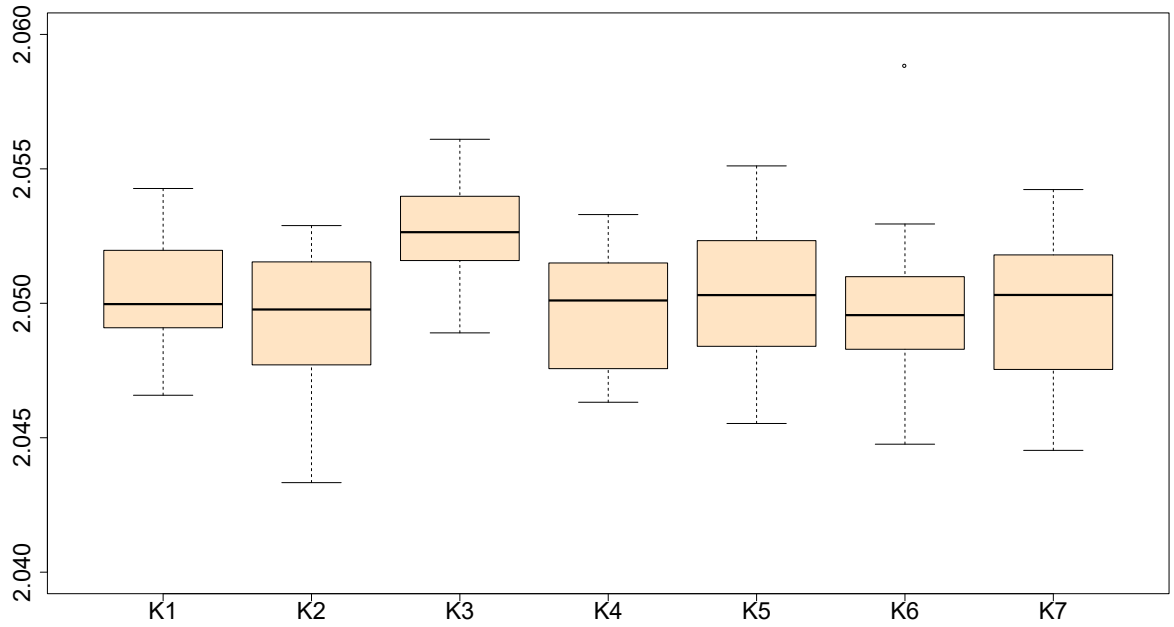
6.3.1 Rezultati za IPR

IPR pristup podrazumijeva razvoj novih prioritetnih pravila pri čemu se u skup značajki dodaju i dodatne značajke na temelju čijih vrijednosti se raspored kroz iteracije poboljšava. Za RCPSP i funkcije cilja F_1 i F_2 uvedene su nove značajke dane tablicom 6.1. S obzirom da su uvedene tek 3 značajke, njihov utjecaj na rezultat moguće je ispitati odvojeno, ali i u svim međusobnim kombinacijama. Broj kombinacija koje je u tom slučaju potrebno ispitati je 7.

Za svaku od tih 7 kombinacija razvijeno je 30 pravila koristeći isti skup za učenje i provjeru kao kod razvoja prioritetnih pravila u prethodnim poglavljima. Prilikom razvoja IPR-a korišteni su isti parametri kao i kod standardnih pravila. Razlika u razvoju IPR-a u odnosu na standardna prioritetna pravila je u dodatnim značajkama, koje se razlikuju u ovisnosti o kombinaciji koja se ispituje, i shemi za generiranje rasporeda. Shema za generiranje rasporeda korištena prilikom razvoja IPR-a dana je algoritmom 13, a ista shema korištena je i kod evaluacije razvijenih pravila. Rezultati ostvareni na dodatnom skupu za provjeru IPR-om uz korištenje funkcije F_1 dani su tablicom 6.3 i kutijastim dijagramom na slici 6.1, a uz korištenje funkcije F_2 tablicom 6.4 i kutijastim dijagramom na slici 6.2.

Tablica 6.3: Rezultati ostvareni korištenjem IPR-a i funkcije F_1

Kombinacija	dodana značajka	min	med	max
K1	ActFin	2.04658	2.04997	2.05427
K2	ActWait	**2.04333**	2.04977	**2.05289**
K3	LBDist	2.04890	2.05265	2.05610
K4	ActFin, ActWait	2.04632	2.05011	2.05330
K5	LBDist, ActFin	2.04553	2.05031	2.05511
K6	LBDist, ActWait	2.04476	**2.04956**	2.05885
K7	LBDist, ActFin, ActWait	2.04453	2.05031	2.05423



Slika 6.1: Rezultati za IPR uz korištenje funkcije F_1

Ako promatramo različite kombinacije dodanih značajki prilikom razvoja IPR-a za obje funkcije cilja možemo primijetiti da odstupanja u ostvarenim rezultatima nisu značajno različita. No, uz pomoć prikaza kutijastim dijagramom moguće je pronaći onu kombinaciju koja daje manje raspršene rezultate, odnosno koja postiže nešto manji minimum i maksimum od ostalih.

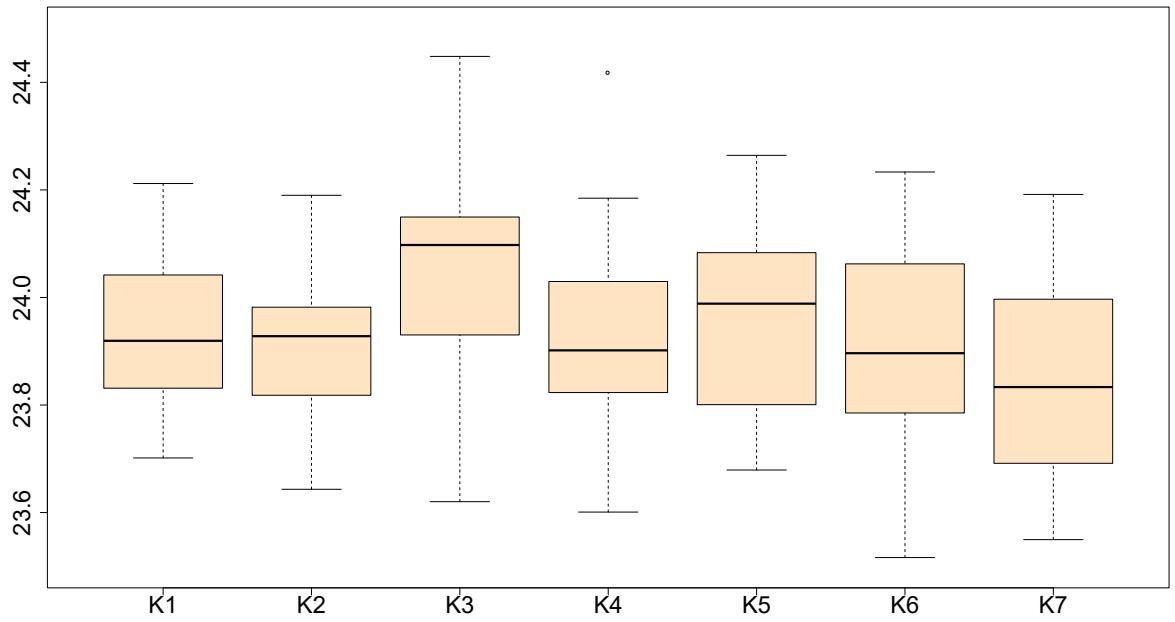
Za funkciju F_1 dodavanjem značajke ActWait ostvaruje se najmanji minimum i maksimum, dok se dodavanjem značajki LBDist i ActWait ostvaruje nešto manji medijan nego onaj uz dodavanje samo značajke ActWait. Može se primijetiti da se u obje kombinacije nalazi značajka ActWait što znači da vrijeme čekanja aktivnosti od trenutka kada je postala dostupna do njezinog raspoređivanja bitno utječe na kvalitetu rješenja ostvarenog kod korištenja funkcije F_1 kao kriterija optimizacije. Kako je razlika u ostvarenim medijanima između ovih kombinacija mala, a ona između ostvarenih minimuma i maksimuma nešto veća, prilikom razvoja IPR-a ako se koristi funkcija F_1 dodat će se samo značajka ActWait.

Tablica 6.4: Rezultati ostvareni korištenjem IPR-a i funkcije F_2

Kombinacija	dodana značajka	min	med	max
K1	ActFin	23.70156	23.91941	24.21187
K2	ActWait	23.64329	23.92800	**24.18997**
K3	LBDist	23.62028	24.09754	24.44826
K4	ActFin, ActWait	23.60090	23.90161	24.41879
K5	LBDist, ActFin	23.67911	23.98847	24.26429
K6	LBDist, ActWait	**23.51636**	23.89635	24.23332
K7	LBDist, ActFin, ActWait	23.54966	**23.83336**	24.19158

Kod rezultata ostvarenih korištenjem funkcije F_2 situacija je nešto drugačija. Najmanji maksimum ostvaruje se ako se doda samo značajka ActWait, najmanji minimum ako se uz nju doda i značajka LBDist, dok se najmanji medijan ostvaruje ako su dodane sve tri značajke. No, iako su rezultati najmanje raspršeni ako se doda samo značajka ActWait, raspršenost u slučaju dodavanja sve tri značajke ide u korist poboljšanju kvalitete rješenja. Maksimum kod rezultata ostvarenih korištenjem sve tri značajke je tek nešto veći od onog ostvarenog dodavanjem samo značajke ActWait, a s druge strane ostvareni minimum uz korištenje sve tri značajke je značajno bolji. U skladu s ovim razmatranjem zaključuje se da je kod korištenja funkcije F_2 potrebno dodati sve tri značajke u skup značajki.

S obzirom na dijelove od kojih se sastoji funkcija F_2 očekivano je da se značajka LBDist treba dodati u skup značajki. No, zanimljivo je da se dodavanjem samo značajke LBDist u skup značajki ostvaruju lošiji rezultati u odnosu na one ostvarene bili kojom drugom kombinacijom dodanih značajki i da ova značajka tek u kombinaciji sa značajkom ActWait dovodi do



Slika 6.2: Rezultati za IPR uz korištenje funkcije F_2

poboljšanja rezultata. I u ovom slučaju, značajka ActWait je ona koja najviše utječe na rezultat.

6.3.2 Rezultati za pristup rollout

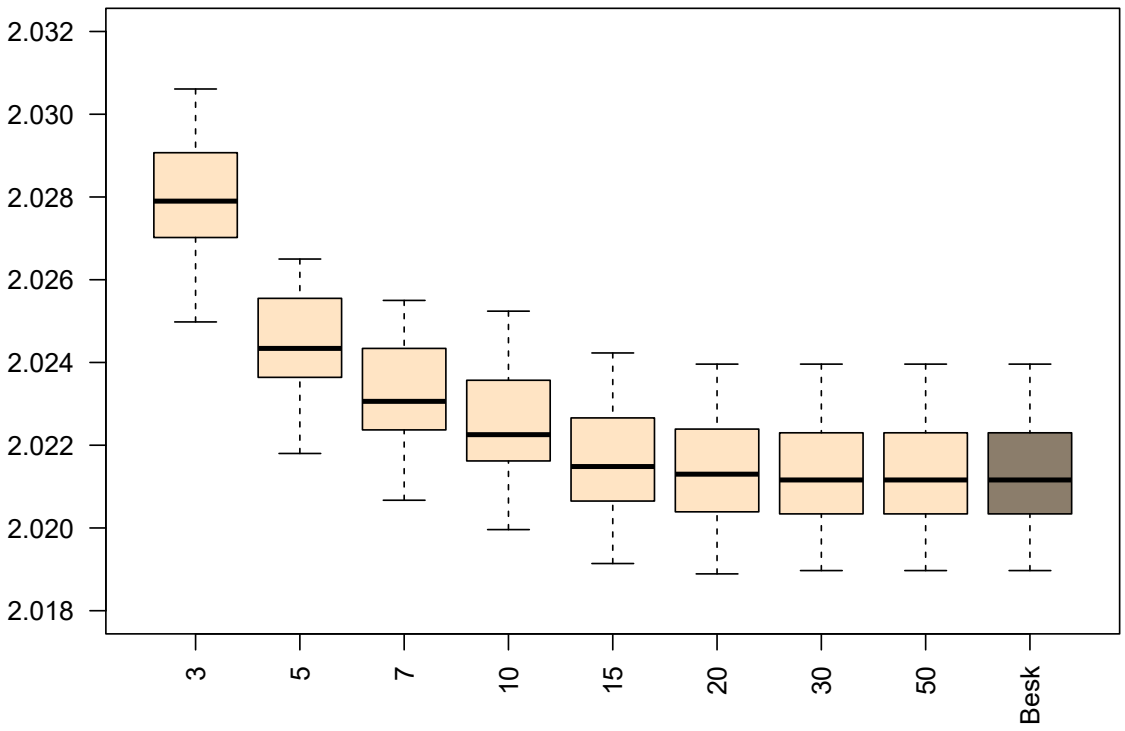
Za razliku od IPR-a za pristup rollout nije potrebno razvijati nova pravila nego je moguće koristiti prethodno razvijena standardna pravila što je jedna od značajnijih prednosti ovog pristupa. Prilikom korištenja pristupa rollout potrebno je osim prioritetnih pravila odrediti i broj aktivnosti koji će se razmatrati prilikom donošenja odluke. Kako bi se utvrdio optimalan broj aktivnosti ispitano je više opcija te su rezultati ostvareni na skupu za provjeru korištenom i za prikaz rezultata kod IPR-a dani tablicom 6.5 za funkciju F_1 i tablicom 6.6 za funkciju F_2, dok je prikaz kutijastim dijagramom dan na slici 6.3 za funkciju F_1, odnosno 6.4 za funkciju F_2.

S ciljem pronalaska optimalnog broja aktivnosti, razmatrano je 8 različitih brojeva iz intervala od 3 do 50 aktivnosti kao i opcija kada se razmatraju sve u tom trenutku dostupne aktivnosti koja je označena s ∞ (tablica) ili Besk (slika).

Neovisno o funkciji koja se koristi prilikom optimizacije, ponašanje koje uočavamo kod ovog pristupa je slično. Povećavanjem broja aktivnosti koje se razmatraju prilikom donošenja odluke koju aktivnost sljedeću rasporediti, poboljšava se i kvaliteta ostvarenog rezultata. Ostvareno poboljšanje u početku je veliko, dok se kasnije povećanjem broja aktivnosti ostvaruje sve manje poboljšanje da bi se nakon određenog broja aktivnosti koje se razmatraju to stabiliziralo u nekoj vrijednosti. Razlog tome mogu biti dvije stvari: broj dostupnih aktivnosti je manji od onih koje se razmatraju u trenutku odluke, odnosno u svakom trenutku odluke sve se aktivnosti

Tablica 6.5: Rezultati ostvareni korištenjem pristupa rollout i funkcije F_1

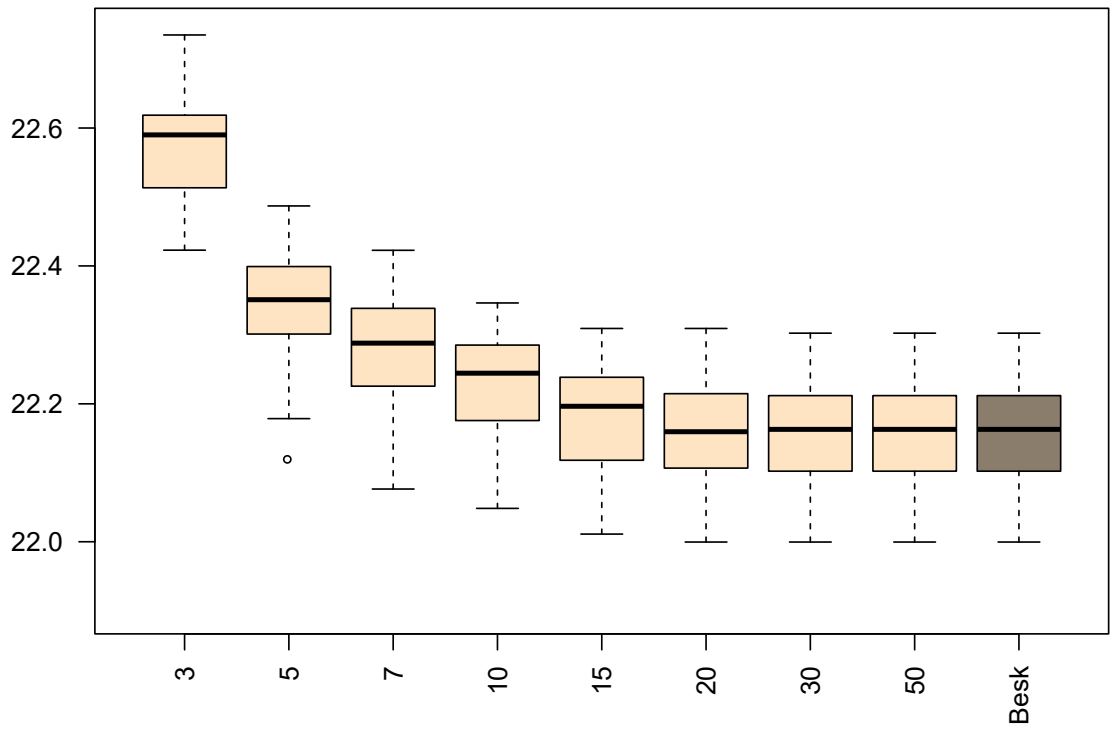
Broj aktivnosti	min	med	max
3	2.02498	2.02790	2.03061
5	2.02180	2.02434	2.02650
7	2.02067	2.02306	2.02550
10	2.01996	2.02226	2.02524
15	2.01914	2.02149	2.02423
20	2.01889	2.02130	**2.02396**
30	**2.01897**	**2.02116**	**2.02396**
50	**2.01897**	**2.02116**	**2.02396**
∞	**2.01897**	**2.02116**	**2.02396**



Slika 6.3: Rezultati za pristup rollout uz korištenje funkcije F_1

Tablica 6.6: Rezultati ostvareni korištenjem pristupa rollout i funkcije F_2

Broj aktivnosti	min	med	max
3	22.42286	22.59005	22.73496
5	22.12102	22.35103	22.48702
7	22.07645	22.28805	22.42267
10	22.04837	22.24445	22.34631
15	22.01111	22.19646	22.30930
20	**21.99957**	**22.15958**	**22.30930**
30	**21.99957**	22.16299	**22.30250**
50	**21.99957**	22.16299	**22.30250**
∞	**21.99957**	22.16299	**22.30250**



Slika 6.4: Rezultati za pristup rollout uz korištenje funkcije F_2

razmatraju ili aktivnosti koje se nalaze pri kraju nakon sortiranja po prioritetu nisu one koje je potrebno razmatrati. U svakom slučaju najbolje rješenje koje se može dobiti ovim pristupom uvijek možemo ostvariti ako kod svake odluke razmatramo sve aktivnosti, no s druge strane povećanjem broja aktivnosti koje se razmatraju u trenutku odluke znatno se povećava i vrijeme potrebno za pronalazak rješenja.

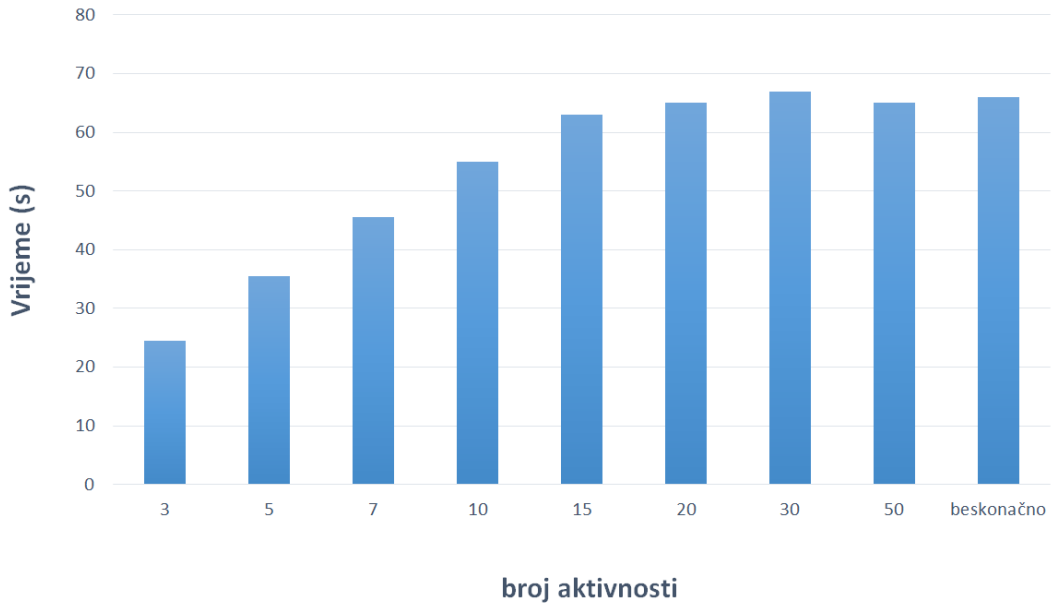
Vrijeme potrebno za evaluciju u ovisnosti o broju aktivnosti koje se razmatraju u trenutku odluke dano je tablicom 6.7 i grafom na slici 6.5. Rezultati su izraženi u sekundama i odnose se na potrebno vrijeme za evaluaciju svih instanci koje se nalaze u skupu za provjeru. U skupu za provjeru nalazi se po 48 instanci problema s po 30, 60 i 90 aktivnosti te 60 instanci sa 120 aktivnosti, odnosno ukupno 204 instance problema. Prikazani rezultati dobiveni su na osnovu korištenja svih 50 razvijenih prioritetnih pravila. Iz danih rezultata primjećuje se da se povećanjem broja razmatranih aktivnosti povećava vrijeme potrebno za evaluaciju, no nakon nekog vremena ono se stabilizira. Broj razmatranih aktivnosti kod kojeg dolazi do stabilizacije vremena je isti kao i broj čijim daljnjim povećanjem ne dolazi do dodatnog poboljšanja ostvarenih rezultata. Zbog toga se može zaključiti da se, neovisno o tome postavlja li se za broj aktivnosti koje se razmatraju 30, 50 ili više aktivnosti, razmatraju sve aktivnosti dostupne u tom trenutku, odnosno broj je veći nego što u bilo kojem trenutku ima dostupnih aktivnosti. Iz ovog razloga se rezultati ostvareni kod brojeva 30 i 50 mogu zanemariti, odnosno zamijeniti onima ostvarenim u slučaju kad se sve dostupne aktivnosti razmatraju. Slično se može zaključiti i koristeći graf na slici 6.6 kojim je prikazan odnos između vrijednosti funkcije cilja F_1 i vremena potrebnog za evaluaciju instanci skupa za provjeru u ovisnosti o broju aktivnosti koje se razmatraju. Na tom grafu vidljivo je da su razlike u rezultatima i potrebnom vremenu za evaluaciju kada se razmatra 30, 50 i sve aktivnosti zanemarive. No, iako vrijeme potrebno za evaluaciju instanci iz skupa za provjeru i za najveći broj razmatranih aktivnosti ne prelazi vrijeme od 1.5 minute ono je znatno veće od onog potrebnog za evaluaciju kod manjeg broja razmatranih aktivnosti. Na primjer, ako razmatramo sve dostupne aktivnosti u svakom trenutku odluke, vrijeme potrošeno za evaluaciju bit će u prosjeku skoro 3 puta veće od onog potrebnog u slučaju da se u obzir uzimaju samo 3 aktivnosti.

6.3.3 Usporedba rezultata

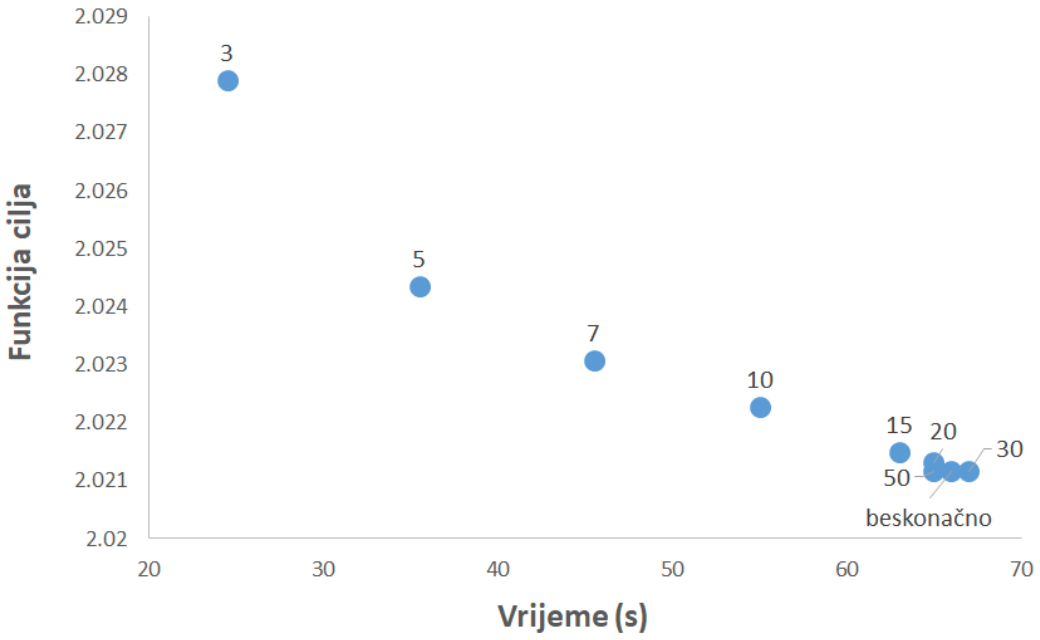
U ovom dijelu napravit će se usporedba ostvarenih rezultata IPR-om, pristupom rollout i pravilima razvijenim koristeći standardni SGS, odnosno GP. S obzirom da se IPR i pristup rollout koriste u statičkom okruženju i kao takvi raspolažu s više informacija koje onda koriste s ciljem poboljšanja rezultata, od njih se očekuje postizanje boljih rezultata od standardnog GP-a. Osim toga, kako je u statičkom okruženju moguće koristiti i druge metaheurističke metode, usporedba samo s prioritetnim pravilima ne bi bila dostatna. Zbog toga će se ovi pristupi dodatno usporediti s rezultatima ostvarenim genetičkim algoritmom. Osim postignutih rezultata u obzir će se

Tablica 6.7: Vrijeme potrebno za evaluaciju instanci skupa za provjeru u ovisnosti o broju aktivnosti koje se razmatraju izraženo u sekundama

Broj aktivnosti	min	med	max
3	23	24.5	26
5	28	35.5	39
7	42	45.5	50
10	43	55	59
15	58	63	71
20	58	65	74
30	60	67	75
50	52	65	80
∞	59	66	79



Slika 6.5: Vrijeme potrebno za evaluaciju instanci skupa za provjeru u ovisnosti o broju aktivnosti koje se razmatraju



Slika 6.6: Odnos vrijednosti funkcije F_1 i vremena potrebnog za evaluaciju instanci skupa za provjeru u ovisnosti o broju aktivnosti koje se razmatraju

uzeti i potrebno vrijeme za pronalazak rješenja. Upravo zbog razlike u potrebnom vremenu za pronalazak rješenja, za pristup rollout analizirani su rezultati u slučaju kada se prilikom donošenja odluke u obzir uzimaju 3 aktivnosti i kad se uzimaju sve aktivnosti, što predstavlja dva krajnja slučaja ovog pristupa.

Rezultati ostvareni na ispitnom skupu za funkciju F_1 dani su tablicom 6.8 i kutijastim dijagramom na slici 6.7, dok su za funkciju F_2 dani tablicom 6.9 i kutijastim dijagramom na slici 6.8. Ponašanje ostvarenih rezultata za obje funkcije je dosta slično pa ih je moguće zajedno analizirati.

Iz rezultata kao i iz grafičkog prikaza istih lako se uočava kako i IPR i pristup rollout bez obzira na broj aktivnosti koje se razmatraju prilikom donošenja odluke postižu značajno bolje rezultate od onih koji su ostvareni korištenjem prioritetnih pravila razvijenih standardnim GP-om i to za obje promatrane funkcije cilja. Bolji rezultati ostvareni IPR-om su posljedica toga što se kod običnih prioritetnih pravila raspored izgrađuje samo jednom, dok se kod IPR-a on kroz iteracije poboljšava koristeći pri tom informacije iz prethodne iteracije. Također i bolji rezultati ostvareni pristupom rollout su očekivani i to zbog činjenice da se i kod njega gradi više rasporeda, odnosno prati se više smjerova prilikom izgradnje rasporeda što nužno dovodi do jednakih ili boljih rezultata.

Iako su ovi pristupi ostvarili bolje rezultate od prioritetnih pravila razvijenih GP-om, ipak sa svojom kvalitetom još uvijek ne mogu konkurirati rezultatima ostvarenim korištenjem GA. Razlog što GA postiže značajno bolji rezultat se krije u tome što unatoč činjenici da se kod oba pristupa raspored gradi više puta, i kod IPR-a i kod rollout-a imamo jedno pravilo za cijeli skup

instanci, a kod GA se svaka instanca rješava posebno. Osim toga, potrebno je naglasiti da se prilikom korištenja GA ne mogu koristiti izravno funkcije F_1 i F_2 jer one u obzir uzimaju sve instance problema unutar skupa za učenje, dok GA zbog prikaza jedinki rješava jednu po jednu instancu. U skladu s tim, kod GA se kao kriterij optimizacije koristilo trajanje projekta, te se ukupna ostvarena vrijednost dobila nakon što je svaka instanca posebno riješena. Takav pristup daje prednost GA zbog činjenice da na ovaj način algoritam može biti usredotočen na samo jednu instancu u svakom trenutku, dok se kod prioritetnih pravila zbog zajedničke evaluacije algoritam može u nekim trenucima usmjeriti na lošije riješene instance zbog čega se kod drugih ne ostvari onaj rezultat koji bi se mogao ostvariti da se pravilo usredotočilo na samo tu instancu problema.

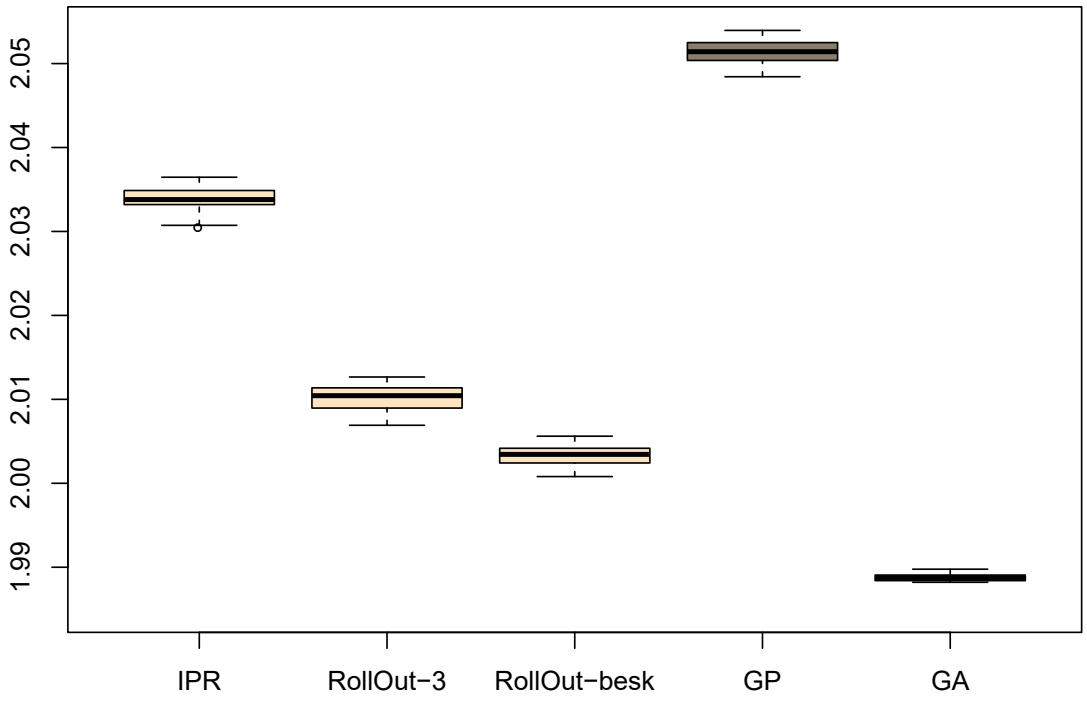
Tablica 6.8: Usporedba rezultata ostvarenih različitim metodama uz korištenje funkcije F_1

Metoda	min	med	max
IPR	2.03058	2.03380	2.03646
RollOut-3	2.00691	2.01044	2.01266
RollOut-∞	2.00080	2.00345	2.00561
GP	2.04843	2.05142	2.05395
GA	1.98821	1.98875	1.98977

Tablica 6.9: Usporedba rezultata ostvarenih različitim metodama uz korištenje funkcije F_2

Metoda	min	med	max
IPR	23.45423	23.69920	23.87690
RollOut-3	22.05307	22.26630	22.39924
RollOut-∞	21.63508	21.82767	21.97313
GP	24.60789	24.76803	24.94999
GA	20.93309	20.97285	21.03402

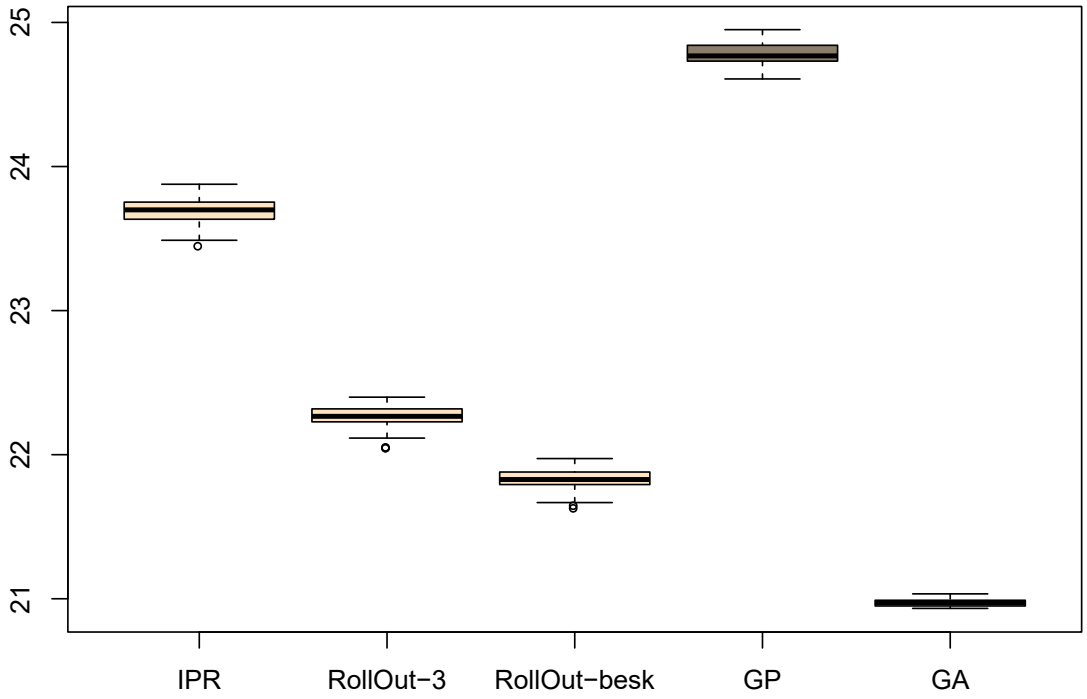
No, osim rezultata bitno je usporediti i vrijeme potrebno za razvoj pravila kao i brzinu pronalaska rješenja ako se koriste razvijena pravila, IPR, pristup rollout ili GA. Prilikom usporedbe nije korištena nikakva paralelizacija s ciljem skraćivanja trajanja postupka kako bi usporedba bila pravedna. Usporedba potrebnog vremena za razvoj standardnog pravila GP-om i onog kod kojeg se koristi prilagođena shema za IPR dana je tablicom 6.10. U tablici su vremena izražena u sekundama, a dano je prosječno vrijeme potrebno za jednu generaciju kao i ukupno vrijeme



Slika 6.7: Usporedba rezultata ostvarenih različitim metodama uz korištenje funkcije F_1

Tablica 6.10: Trajanje razvoja pravila u ovisnosti o metodi

Metoda	trajanje (s)	
	prosječno po generaciji	ukupno
GP	139.34	3483.43
IPR	458.54	11463.60



Slika 6.8: Usporedba rezultata ostvarenih različitim metodama uz korištenje funkcije F_2

Tablica 6.11: Trajanje rješavanja instanci problema ispitnog skupa u ovisnosti o metodi

Metoda	trajanje (s)	
	ukupno	prosječno po instanci
IPR	68	0.06
RollOut-3	149.17	0.12
RollOut-∞	370.07	0.30
GP	28	0.02
GA	526683	430.30

potrebno za razvoj pravila kod kojeg se kao kriterij zaustavljanja uzima 25 generacija. Iz tablice se vidi kako je za razvoj jednog pravila koristeći standardni SGS potrebno nešto manje od sat vremena, dok je to vrijeme za IPR nešto veće od 3 sata, odnosno može se reći da je za razvoj pravila koristeći prilagođenu shemu za IPR potrebno otprilike 3 puta više vremena od onog potrebnog za razvoj običnog prioritetnog pravila. Prednost običnih prioritetnih pravila kao i IPR-a je što se razvoj pravila može napraviti unaprijed te na taj način to vrijeme ne utječe na kasnije rješavanje novih problema.

Od ove usporedbe svakako je važnija ona koja uspoređuje vrijeme potrebno za rješavanje novih instanci problema, a koja je dana tablicom 6.11. U ovoj tablici vidljiva je brzina prioritetnih pravila i koliko je velika razlika u potrebnom vremenu za primjenu prioritetnih pravila i neke druge metaheurističke metode. Dani rezultati su oni koji su dobiveni prilikom rješavanja instanci iz ispitnog skupa. Ispitni skup kao i u prethodnim poglavljima je sastavljen od 1224 instance od čega njih 360 se sastoje od 120 aktivnosti, a po 288 ih je koji imaju $30, 60$ i 90 aktivnosti.

Prioritetna pravila razvijena GP-om rješenja za cijeli ispitni skup pronalaze u svega 28 sekundi, što je u prosjeku 0.02 sekunde po instanci problema. Vrijeme potrebno za pronalazak rješenja korištenjem IPR-a je otprilike 3 puta veće od onog potrebnog običnim prioritetnim pravilima. Iz ovog se može primijetiti da je omjer vremena potrebnog za pronalazak rješenja koristeći ova dva pristupa i razvoj pravila za ova dva pristupa otprilike jednak. Razlog tome je što se, i kod razvoja pravila i kod rješavanja konkretnih problema, koristi ista shema za generiranje rasporeda. Kako se shema za generiranje rasporeda kod IPR-a od one standardne sheme razlikuje samo zbog iteracija koje se događaju u shemi korištenoj kod IPR-a, može se zaključiti da se u prosjeku kod svake evaluacije koristeći IPR napravi 3 iteracije.

Vrijeme potrebno za pronalazak rješenja korištenjem pristupa rollout je značajno veće od onog potrebnog i kod običnih prioritetnih pravila i kod IPR-a. Iako postoji značajna razlika u potrebnom vremenu za pronalazak rješenja bitno je uočiti da bez obzira na broj aktivnosti koje se razmatraju prilikom odluke, u prosjeku, za rješavanje pojedine instance problema ovim pristupom je potrebno manje od pola sekunde. S druge strane, GA u prosjeku jednu instancu problema rješava preko 7 minuta, što znači da je za pronalazak rješenja za sve instance ispitnog skupa koristeći GA potrebno i nekoliko dana. Naravno, u obzir treba uzeti da je moguće paralelizirati pojedine postupke u rješavanju te da je moguće koristiti računalo jačih performansi, no i dalje će omjer ovih vremena ostati sličan.

6.4 Zaključak

Prioritetna pravila većinom se koriste u dinamičkim okruženjima u kojima je naglasak stavljen na brzinu pronalaska rješenja i mogućnost reagiranja na promjene sustava. S obzirom da se rje-

šenja postignuta prioritetnim pravilima najčešće kvalitetom ne mogu mjeriti s onim postignutim nekom metaheurističkom metodom, njihova primjena u statičkim okruženjima nije česta. Bez obzira na to, postoje situacije u kojima prednosti prioritetnih pravila kao što su brzina i jednostavnost mogu biti važniji i u statičkim okruženjima nego što su njihovi očigledni nedostatci. Zbog toga su u ovom poglavlju analizirani načini s pomoću kojih se prioritetna pravila mogu dodatno poboljšati koristeći informacije koje su dostupne prilikom raspoređivanja u statičkim okruženjima.

U ovom poglavlju analizirana su dva pristupa za prilagodbu prioritetnih pravila u statičkom okruženju, iterativna prioritetna pravila (IPR) i pristup rollout. U sklopu IPR-a analizirano je uvođenje tri različite značajke u skup značajki za razvoj prioritetnih pravila. Uvođenjem tih značajki omogućen je prijenos informacije iz iteracije u iteraciju s ciljem postizanja boljih rješenja. U ovisnosti o funkciji cilja pronađene su značajke koje je potrebno uvesti kako bi se ostvarilo najveće poboljšanje u rezultatu. Kod pristupa rollout analizirani su različiti brojevi aktivnosti koje su razmatrane prilikom donošenja odluke koju aktivnost sljedeću rasporediti. Osim toga, kod ovog pristupa analizirano je i vrijeme potrebno za pronalazak rješenja u ovisnosti o broju aktivnosti koje se uzimaju u obzir.

Nadalje, dana je usporedba ovih pristupa s običnim prioritetnim pravilima i GA, kao predstavnikom metaheurističkih pristupa. IPR i pristup rollout ostvarili su bolje rezultate od običnih pravila, no i dalje nisu uspjeli ostvariti bolje rezultate od onih ostvarenih s GA. Osim kvalitete rezultata, razmatrano je i vrijeme potrebno za pronalazak rješenja te je utvrđeno da je ono u izravnoj vezi s kvalitetom rješenja, odnosno pristupi za koje je potrebno više vremena prilikom rješavanja su oni koji su ostvarili bolje rezultate. U skladu s tim, prilikom odabira pristupa u obzir je potrebno uzeti i vrijeme i kvalitetu te se odlučiti koju cijenu u obliku vremena smo spremni platiti za postignutu kvalitetu rješenja.

Na osnovu rezultata danih u ovom poglavlju može se zaključiti da se primjenom dodatnih informacija, sadržanih unutar statičkih okruženja, rješenja ostvarena pravilima koja su razvijena GP-om mogu poboljšati. Ovaj zaključak otvara nove teme za daljnja istraživanja. Jedna od njih je otkrivanje drugih načina prilagodbi prioritetnih pravila za statička okruženja, a koji bi poput ova dva analizirana, doprinijeli kvaliteti rješenja. Osim toga, moguće je dodatno prilagoditi, u ovom radu, analizirane pristupe korištenjem drugačije sheme za generiranje rasporeda i izgrađivanjem rasporeda u oba smjera (engl. *bidirectional planning*). Neovisno o odabiru smjera za daljnja istraživanja, potrebno je pronaći način kako što više informacija koje su dostupne u statičkom okruženju ugraditi u prioritetna pravila ili postupke izgradnje rasporeda. Također, potrebno je osigurati da pri tome ne dođe do značajnog porasta vremena potrebnog za izvršenje čija mala vrijednost u ovom trenutku predstavlja najveću prednost primjene prioritetnih pravila.

Poglavlje 7

Zaključak

Problem raspoređivanja s ograničenim sredstvima (RCPSP) je problem s kojim se susrećemo svakodnevno i koji ima veliku važnost u proizvodnji i drugim industrijama. Zbog svoje važnosti i kompleksnosti i nakon više od 50 godina od svog uvođenja privlači brojne istraživače. RCPSP pripada klasi NP-teških problema zbog čega ga je u većini slučajeva nemoguće egzaktno riješiti. Upravo zbog toga razvijaju se brojne heurističke metode s ciljem pronalaska što boljeg rješenja. Iako postoje metode koje u praksi pronalaze dobra rješenja ovog problema, nažalost, njihova primjena je najčešće ograničena na statička okruženja što nije dostatno za probleme koji se javljaju u praksi.

Kao odgovor na potrebe dinamičkih okruženja javljaju se prioritetna pravila. Prioritetna pravila dolaze u kombinaciji sa shemom generiranja rasporeda koja na osnovu aktivnosti dodijeljenih prioriteta gradi raspored. Prednost prioritetnih pravila u odnosu na druge heurističke metode je njihova jednostavnost i brzina te mogućnost reagiranja na promjene koje se mogu dogoditi unutar proizvodnog sustava. Nažalost rezultati koji se ostvaruju prioritetnim pravilima su u većini slučajeva lošiji od onih koje druge heurističke metode postižu u statičkim okruženjima. Osim toga dobro prioritetno pravilo nije lako kreirati i potrebno ga je kreirati za svaki optimizacijski kriterij posebno. Upravo zbog toga, uvode se različiti postupci i metode za automatizirani razvoj prioritetnih pravila u raspoređivanju. Najčešće korištena metoda za razvoj prioritetnih pravila kod različitih vrsta problema raspoređivanja je genetičko programiranje (GP). GP predstavlja svojevrsnu metodu strojnog učenja, koja s pomoću skupa za učenje i skupa za provjeru uz pomoć dobro odabranih značajki i funkcija razvija prioritetna pravila za različite optimizacijske kriterije.

7.1 Glavni zaključci i doprinosi

Za različite oblike problema raspoređivanja, kao i okruženja, moguće je pronaći brojne radove u kojima je postupak razvoja prioritetnih pravila automatiziran GP-om. S obzirom na dobre re-

zultate koji se pri tome ostvaruju kao i činjenicu da GP do sada nije u velikoj mjeri primjenjivan na RCPSP, u ovoj disertaciji naglasak je stavljen upravo na primjenu GP-a na RCPSP. Konkretnije, u ovoj disertaciji GP je primjenjen u razvoju prioritetnih pravila za RCPSP. Osim razvoja prioritetnih pravila, u disertaciji su analizirani i dodatni načini poboljšanja rezultata ostvarenih prioritetnim pravilima temeljeni na skupnom učenju kao i prilagodba prioritetnih pravila za statičku okolinu raspoređivanja.

7.1.1 Oblikovanje prioritetnih pravila za RCPSP

U ovoj disertaciji analiziran je razvoj prioritetnih pravila za dva različita kriterija optimizacije. Prilikom razvoja prioritetnih pravila korišten je skup značajki i funkcija za koji je prethodnim ispitivanjem utvrđeno da bitno utječe na kvalitetu postignutog rješenja. Tijekom razvoja korišteni su skupovi za učenje i provjeru sastavljeni od instanci s 30, 60, 90 i 120 aktivnosti. Ispitivanjem je utvrđena važnost korištenja skupa za provjeru prilikom razvoja pravila, kao i da skupovi za učenje i provjeru imaju sličan sastav kao i problemi koji će se kasnije rješavati razvijenim pravilima. Isto tako, ispitivanjem je zaključeno da normalizacija korištenih značajki ne dovodi do poboljšanja rezultata te je nije potrebno provesti.

Osim skupa značajki i funkcija analizirane su tri različite sheme za generiranje rasporeda i to standardna usporedna i slijedna shema i usporedna shema s odgodom. Usporedna shema za generiranje rasporeda uz odgodu predložena je zbog činjenice da optimalni rasporedi ne moraju biti rasporedi bez odgode, a standardna usporedna shema rezultira upravo njima. Unatoč tome, ispitivanja su pokazala kako je standardna usporedna shema za generiranje rasporeda ona koja je najbolja i da nije potrebno uvoditi odgode, jer time ne dolazi do poboljšanja kvalitete rješenja.

Rezultati ostvareni pravilima razvijenim GP-om uspoređeni su s rezultatima ostvarenima korištenjem postojećih prioritetnih pravila te je utvrđeno da neovisno o kriteriju optimizacije i korištenoj shemi za generiranje rasporeda pravila razvijena GP-om ostvaruju značajno bolje rezultate. Iako su u ovoj disertaciji analizirana tek dva kriterija za optimizaciju, dani postupak lako je prilagoditi za bilo koji drugi kriterij optimizacije.

7.1.2 Oblikovanje ansambala prioritetnih pravila

Iako razvijena prioritetna pravila ostvaruju bolje rezultate od postojećih prioritetnih pravila, javlja se pitanje mogu li se rezultati dodatno poboljšati korištenjem neke metode skupnog učenja. U ovoj disertaciji ispitano je može li se korištenjem ansambla ostvariti bolja rješenja od onih postignutih pojedinačnim pravilima. Tijekom ispitivanja korištena su četiri različita pristupa u kreiranju ansambla te dva u njihovom kombiniranju.

Rezultati ostvareni ansamblima značajno su bolji od onih ostvarenih pojedinačnim prioritetnim pravilima. Najbolji ispitani način kreiranja ansambala u ovoj disertaciji je metoda jed-

nostavnog kombiniranja. Dodatno, na kreirane ansamble primijenjen je postupak za traženje najboljeg podskupa pravila sadržanih u ansamblu. Određivanjem podskupa pravila moguće je smanjiti veličinu ansambla i na taj način utjecati na kvalitetu rješenja, ali i na vrijeme potrebno za rješavanje instance problema. Za metode kreiranja u kojima se pravila koja se nalaze u ansamblu razvijaju ili biraju neovisno jedno o drugome, postupak pronalaska optimalnog podskupa se pokazao korisnim, dok za one gdje izbor nije neovisan ovaj postupak ne dovodi do poboljšanja.

Ansambli su korisni zbog toga što dovode do poboljšanja kvalitete rješenja, a da pri tome ne gube značajno na jednostavnosti i brzini koja je prisutna kod pojedinačnih pravila. Načini kreiranja ansambla su brojni te ih je moguće dodatno proširiti uz tek male promjene što ovaj postupak čini prilično jednostavnim. Osim toga, u metodi jednostavnog kombiniranja, koja se pokazala kao najbolja za kreirane ansambla za RCPSP moguće je koristiti već unaprijed razvijena pravila, odnosno nije ih potrebno posebno razvijati što je dodatna prednost ovog pristupa.

7.1.3 Prilagodba prioritetnih pravila za primjenu u statičkim okruženjima

Ansambli prioritetnih pravila dovode do određenog poboljšanja kvalitete postignutih rezultata, no ipak prilikom svog oblikovanja ne koriste informacije koje su dostupne u statičkim okruženjima. Korištenje takvih informacija može dovesti do dodatnog poboljšanja u rezultatima. Zbog toga su u sklopu disertacije ispitana dva različita pristupa koja je moguće primijeniti ako se raspoređivanje odvija u statičkim uvjetima.

Prvi od tih pristupa su iterativna prioritetna pravila (IPR) koja kroz iteracije dodatno poboljšavaju kvalitetu ostvarenog rezultata. Kako bi se određene informacije mogle prenijeti iz iteracije u iteraciju, za ovaj postupak potrebno je uvesti dodatne značajke u skup značajki. Ispitivanja pokazuju da IPR postiže rezultate koji su značajno bolji od onih ostvarenih pojedinačnim pravilom, a pri čemu se vrijeme potrebno za pronalazak rješenja ne povećava značajno.

Drugi pristup proučavan u ovoj disertaciji je rollout. Ovaj pristup razlikuje se od IPR-a u više stvari. Prva od njih je to što za njega nije potrebno razvijati posebna prioritetna pravila, nego je moguće koristiti već razvijena. Osim toga, kod ovog pristupa nije potrebno uvoditi dodatne značajke u skup značajki. Kvaliteta rješenja pomoću ovog postupka poboljšava se na način da se u trenutku odluke razmatra više aktivnosti koje je moguće rasporediti te se, tek određivanjem kvalitete rasporeda koji se dobija raspoređivanjem svake od tih aktivnosti, određuje koja aktivnost se sljedeća raspoređuje. Pristupom rollout ostvaruju se značajno bolja rješenja i od onih ostvarenih pojedinačnim pravilima i od onih ostvarenih IPR-om. No, s druge strane vrijeme potrebno za pronalazak rješenja se značajno povećava.

Na temelju ispitivanja provedenih u ovoj disertaciji može se zaključiti da je prioritetna pravila moguće dodatno prilagoditi za statička okruženja. Osim pristupa analiziranih u ovom radu moguće je primijeniti i brojne druge te ostvariti još bolje rezultate, a da se pri tome ipak, kao i

u ovim pristupima, očuva jednostavnost metode kao i prilično malo vrijeme potrebno za pronalazak rješenja.

7.2 Buduća istraživanja

Ispitivanja provedena u ovoj disertaciji predstavljaju svojevrsni početak primjene razvoja prioritetnih pravila za RCPSP GP-om te kao takva otvaraju veliki broj tema za daljnja ispitivanja. S obzirom da u literaturi postoji tek mali broj radova na ovu temu, u sklopu ove disertacije bilo je važno detaljno ispitati stvari poput skupa značajki i skupa funkcija te parametara koje će GP prilikom oblikovanja prioritetnih pravila koristiti. Zbog toga, za formulaciju RCPSP je korištena osnovna formulacija koja pretpostavlja da u sustavu nema prekida, da su sve aktivnosti dostupne od prvog trenutka u planiranju, kao i da se svaka od njih može izvesti na jedan način. Korištenje ovakve formulacije otvara mogućnost za brojna druga ispitivanja u kojima će se ove pretpostavke izostavljati te model dodatno proširivati.

Osim proširivanja modela, buduća istraživanja mogu ići i u smjeru višekriterijske optimizacije u kojoj bi se optimiziralo više kriterija, pri čemu jedan od njih može biti i kompleksnost razvijenih prioritetnih pravila. Kompleksnost prioritetnih pravila može se ograničiti na više načina. Neki od njih su maksimalni broj čvorova u stablu, dubina stabla, omjer broja korištenih značajki i funkcija u pravilu i sl. Smanjenjem kompleksnosti razvijenih prioritetnih pravila došlo bi i do povećanja interpretabilnosti pravila što može povećati primjenu prioritetnih pravila u praksi. Ako razvijeno pravilo nije moguće interpretirati njegova upotreba je upitna bez obzira na postignute rezultate u različitim ispitivanjima.

Dodatno, moguće je uvesti nove značajke i funkcije u skup primitiva te koristiti neke druge načine određivanja njihove važnosti za oblikovanje prioritetnih pravila. Dodavanje novih značajki i funkcija može biti važno kod primjene drugih kriterija optimizacije, što je također jedna od novih tema za istraživanje.

Pretpostavke na model korišten u sklopu ove disertacije na neki način su bliske statičkom okruženju, zbog čega je potrebno ispitati kako oblikovati prioritetna pravila za one probleme u kojima će biti prisutne promjene karakteristične za dinamička okruženja, kao što su određene neizvjesnosti u informacijama o aktivnosti (njezino trajanje, količina potrebnih sredstava i sl.) kao i o sredstvima (dostupna količina sredstva se smanjila, sredstvo je privremeno nedostupno i sl.).

Ova disertacija ostavlja otvorene teme i vezane uz oblikovanje ansambala kao što su otkrivanje nekih drugih načina kreiranja i kombiniranja prioritetnih pravila u ansambl te može li se dodatnim prilagođavanjem vrijednosti korištenih parametara ostvariti bolji rezultati. Osim toga, otvara se pitanje može li i primjena nekog drugog postupka strojnog učenja dovesti do poboljšanja kvalitete rješenja.

Najveća mogućnost napretka svakako leži u dodatnoj prilagodbi prioritetnih pravila za korištenje u statičkim okruženjima. Ispitivanja provedena u sklopu ove disertacije predstavljaju tek početak u otkrivanju različitih načina prilagodbe statičkim uvjetima. Kao područja za daljnje istraživanje nameće se otkrivanje novih postupaka prilagodbe kao i dodatna prilagodba pristupa korištenih u ovoj disertaciji. Prilagodba korištenih pristupa je moguća, na primjer, kroz otkrivanje i dodavanje nekih drugih značajki u skup značajki za IPR, ili drugačijom shemom za generiranje rasporeda kod pristupa rollout.

Literatura

[1] Pinedo, M. L., Scheduling: Theory, algorithms, and systems, 4th ed. Springer, 2012, dostupno na: http://link.springer.com/10.1007/978-1-4614-2361-4

[2] Klein, R., Scheduling of resource-constrained projects. New York, SAD: Springer-Science+ Business Media LLC, 2000.

[3] Kawanaka, H., Yamamoto, K., Yoshikawa, T., Shinogi, T., Tsuruoka, S., "Genetic algorithm with the constraints for nurse scheduling problem", in International conference on Evolutionary Computation, Vol. 2, Seoul, Korea, May 2001, str. 1123-1130.

[4] Oughalime, A., Ismail, W., Yeun, L., "A tabu search approach to the nurse scheduling problem", in International Symposium on Information Technology(ITSim), Vol. 1, Kuala Lumpur, August 2008, str. 1-7.

[5] Toth, P., Vigo, D., The vehicle routing problem. Philadelphia, SAD: Society for Industrial and Applied Mathematics, 2015.

[6] Fosin, J., "Metoda rješavanja vremenski ovisnog problema usmjeravanja vozila zasnovana na profilima brzina.", Doktorski rad, Fakultet prometnih znanosti, Sveučilište u Zagrebu, 2015.

[7] Kumar, S. N., "A survey on the vechile routing problem and its variants", Intelligent Information Management, Vol. 4, 2012, str. 66-74.

[8] Soria-Alcaraz, J. A., Carpio, M., Terashima-Marin, H., "Several strategies to improve the performance of hyperheuristics for academic timetabling design problem", in International conference on Electronics, Robotics and Automotive Mechanics, Cuernavaca, Mexico, 2010, str. 102-107.

[9] Leonardo, A., Humberto, C., "The school timetabling problems: a focus on elimination of open periods and isolated classes", in 6th International Conference on Hybrid Intelligent Systems (HIS 2006), Auckland, New Zealand, December 2006, str. 13-15.

[10] Gargiulo, F., Quagliarella, D., “Genetic algorithms for the resource constrained project scheduling problem”, in 13th IEEE International Symposium on Computational Intelligence and Informatics, Budapest, Hungary, November 2012, str. 39-47.

[11] Sun, J., Xhafa, F., “A genetic algorithm for ground station scheduling”, in International Conference on Complex, Intelligent, and Software Intensive Systems, 2001, str. 138-145.

[12] Xhafa, F., Herrero, X., Barolli, A., Takizawa, M., “A struggle genetic algorithm for ground stations scheduling problem”, in IEEE Fourth International Conference on Intelligent Networking and Collaborative Systems, 2012, str. 70-76.

[13] Hindi, K. S., Yang, H., Fleszar, K., “An evolutionary algorithm for resource-constrained project scheduling”, Evolutionary Computation, IEEE Transactions on, Vol. 6, No. 5, 2002, str. 512–518.

[14] Brucker, P., Schoo, A., Thiele, O., “A branch and bound algorithm for the resource constrained project scheduling problem”, European Journal of Operation Research, Vol. 17, 1998, str. 143-158.

[15] Kadam, S., Kadam, N., “Solving resource-constrained project scheduling problem by genetic algorithms”, Business and Information Management (ICBIM), 2014, str. 159-164.

[16] Ouerfelli, H., Dammak, A., “The genetic algorithm with two point crossover to solve the resource-constrained project scheduling problems”, Modeling, Simulation and Applied Optimization (ICMSAO), 2013, str. 1-4.

[17] Sprecher, A., “Network decomposition techniques for resource-constrained project scheduling”, Journal of the Operational Research Society, Vol. 53, No. 4, Apr 2002, str. 405–414, dostupno na: https://doi.org/10.1057/palgrave.jors.2601308

[18] Valls, V., Ballestín, F., “Population-based approach to the resource-constrained project scheduling problem”, Annals of Operations Research, Vol. 131, 2004, str. 305-324.

[19] Ali, I., Elsayed, S., Ray, T., Sarker, R., “Memetic algorithm for solving resource constrained project scheduling problems”, in Evolutionary Computation (CEC), 2015, str. 2761-2767.

[20] Kadam, S., Mane, S., “A genetic-local search algorithm approach for resource constrained project scheduling problem”, in Computing Communication Control and Automation (ICCUBEA), 2015, str. 841-846.

[21] Klein, R., "Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects", European Journal of Operational Research, Vol. 127, No. 3, 2000, str. 619–638.

[22] Kolisch, R., "Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation", European Journal of Operational Research, Vol. 90, No. 2, 1996, str. 320–333.

[23] Branke, J., Nguyen, S., Pickardt, C. W., Zhang, M., "Automated design of production scheduling heuristics: A review", IEEE Transactions on Evolutionary Computation, Vol. 20, No. 1, Feb 2016, str. 110-124.

[24] Nguyen, S., Mei, Y., Zhang, M., "Genetic programming for production scheduling: a survey with a unified framework", Complex & Intelligent Systems, Vol. 3, No. 1, Mar 2017, str. 41–66, dostupno na: https://doi.org/10.1007/s40747-017-0036-x

[25] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J. R., Exploring Hyper-heuristic Methodologies with Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, str. 177–201, dostupno na: https://doi.org/10.1007/978-3-642-01799-5_6

[26] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J. R., A Classification of Hyper-heuristic Approaches. Boston, MA: Springer US, 2010, str. 449–468, dostupno na: https://doi.org/10.1007/978-1-4419-1665-5_15

[27] Dimopoulos, C., Zalzala, A., "Investigating the use of genetic programming for a classic one-machine scheduling problem", Advances in Engineering Software, Vol. 32, No. 6, 2001, str. 489 - 498, dostupno na: http://www.sciencedirect.com/science/article/pii/S0965997800001095

[28] Jakobović, D., Budin, L., "Dynamic scheduling with genetic programming", in Genetic Programming, Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, str. 73–84.

[29] Jakobović, D., Marasović, K., "Evolving priority scheduling heuristics with genetic programming", Applied Soft Computing, Vol. 12, No. 9, 2012, str. 2781 - 2789, dostupno na: http://www.sciencedirect.com/science/article/pii/S1568494612001780

[30] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem", IEEE Transactions on Evolutionary Computation, Vol. 17, No. 5, Oct 2013, str. 621-639.

[31] Hunt, R., Johnston, M., Zhang, M., "Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming", in 2014 IEEE Congress on Evolutionary Computation (CEC), July 2014, str. 618-625.

[32] Đurasević, M., Jakobović, D., Knežević, K., "Adaptive scheduling on unrelated machines with genetic programming", Applied Soft Computing, Vol. 48, 2016, str. 419 - 430, dostupno na: http://www.sciencedirect.com/science/article/pii/S1568494616303519

[33] Koza, J. R., "Human-competitive results produced by genetic programming", Genetic Programming and Evolvable Machines, Vol. 11, No. 3, Sep 2010, str. 251–284, dostupno na: https://doi.org/10.1007/s10710-010-9112-3

[34] Frankola, T., Golub, M., Jakobovic, D., "Evolutionary algorithms for the resource constrained scheduling problem", in 30th International Conference on Information Technology Interfaces, 2008.

[35] Đumić, M., Šišejković, D., Čorić, R., Jakobović, D., "Evolving priority rules for resource constrained project scheduling problem with genetic programming", Future Generation Computer Systems, Vol. 86, 2018, str. 211 - 221, dostupno na: http://www.sciencedirect.com/science/article/pii/S0167739X1732441X

[36] Chand, S., Huynh, Q., Singh, H., Ray, T., Wagner, M., "On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems", Information Sciences, Vol. 432, 2018, str. 146 - 163, dostupno na: http://www.sciencedirect.com/science/article/pii/S0020025517311350

[37] Chand, S., "Automated design of heuristics for the resource constrained project scheduling problem", Doktorski rad, School of Engineering and Information Technology The University of New South Wales Australia, 2018.

[38] Hanchuan Peng, Fuhui Long, Ding, C., "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 8, Aug 2005, str. 1226-1238.

[39] Xue, B., Zhang, M., Browne, W., Yao, X., "A survey on evolutionary computation approaches to feature selection", IEEE Transactions on Evolutionary Computation, Vol. 20, 01 2015, str. 606 - 626.

[40] Xue, B., Fu, W., Zhang, M., "Multi-objective feature selection in classification: A differential evolution approach", in Simulated Evolution and Learning, Dick, G., Browne, W. N., Whigham, P., Zhang, M., Bui, L. T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K. C., Tang, K., (ur.). Cham: Springer International Publishing, 2014, str. 516–528.

[41] Ok, S., Miyashita, K., Nishihara, S., "Improving performance of gp by adaptive terminal selection", in Proceedings of the 6th Pacific Rim International Conference on Artificial Intelligence, ser. PRICAI'00. Berlin, Heidelberg: Springer-Verlag, 2000, str. 435–445, dostupno na: http://dl.acm.org/citation.cfm?id=1764967.1765024

[42] Mei, Y., Zhang, M., Nyugen, S., "Feature selection in evolving job shop dispatching rules with genetic programming", in Proceedings of the Genetic and Evolutionary Computation Conference 2016, ser. GECCO '16. New York, NY, USA: ACM, 2016, str. 365–372, dostupno na: http://doi.acm.org/10.1145/2908812.2908822

[43] Đurasević, M., Jakobović, D., "Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment", Genetic Programming and Evolvable Machines, Vol. 19, 2017.

[44] Đurasević, M., "Automated design of dispatching rules in the unrelated machines environment", Doktorski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2017.

[45] Park, J., Nguyen, S., Zhang, M., Johnston, M., "Evolving ensembles of dispatching rules using genetic programming for job shop scheduling", in Genetic Programming, Machado, P., Heywood, M. I., McDermott, J., Castelli, M., García-Sánchez, P., Burelli, P., Risi, S., Sim, K., (ur.). Cham: Springer International Publishing, 2015, str. 92–104.

[46] Hart, E., Sim, K., "A hyper-heuristic ensemble method for static job-shop scheduling", Evolutionary Computation, Vol. 24, No. 4, 2016, str. 609-635, pMID: 27120113, dostupno na: https://doi.org/10.1162/EVCO_a_00183

[47] Park, J., Mei, Y., Chen, G., Zhang, M., "Niching genetic programming based hyper-heuristic approach to dynamic job shop scheduling: An investigation into distance metrics", in Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, ser. GECCO '16 Companion. New York, NY, USA: ACM, 2016, str. 109–110, dostupno na: http://doi.acm.org/10.1145/2908961.2908985

[48] Park, J., Mei, Y., Nguyen, S., Chen, G., Johnston, M., Zhang, M., "Genetic programming based hyper-heuristics for dynamic job shop scheduling: Cooperative coevolutionary approaches", in Genetic Programming, Heywood, M. I., McDermott, J., Castelli, M., Costa, E., Sim, K., (ur.). Cham: Springer International Publishing, 2016, str. 115–132.

[49] Nguyen, S., Zhang, M., Johnston, M., Tan, K. C., "Learning iterative dispatching rules for job shop scheduling with genetic programming", The International Journal of Advanced Manufacturing Technology, Vol. 67, No. 1, Jul 2013, str. 85–100, dostupno na: https://doi.org/10.1007/s00170-013-4756-9

[50] Hildebrandt, T., Heger, J., Scholz-Reiter, B., "Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach", in Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, ser. GECCO '10. New York, NY, USA: ACM, 2010, str. 257–264, dostupno na: http://doi.acm.org/10.1145/1830483.1830530

[51] Kelley, J. E., The Critical Path Method: Resource Planning and Scheduling. Englewood Cliffs, NJ, SAD: Prentice-Hall, 1963, str. 347-365.

[52] Artigues, C., Demassey, S., Neron, E., Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. ISTE/Wiley, 2008, dostupno na: https://hal.archives-ouvertes.fr/hal-00482946

[53] Blazewicz, J., Lenstra, J. K., Kan, A. R., "Scheduling subject to resource constraints: classification and complexity", Discrete Applied Mathematics, Vol. 5, No. 1, 1983, str. 11–24.

[54] Garey, M. R., Johnson, D. S., Sethi, R., "The complexity of flowshop and jobshop scheduling", Math. Oper. Res., Vol. 1, No. 2, May 1976, str. 117–129, dostupno na: http://dx.doi.org/10.1287/moor.1.2.117

[55] Kolisch, R., Sprecher, A., Drexl, A., "Characterization and generation of a general class of resource-constrained project scheduling problems", Management science, Vol. 41, No. 10, 1995, str. 1693–1703.

[56] Hartmann, S., Briskorn, D., "A survey of variants and extensions of the resource-constrained project scheduling problem", European Journal of Operational Research, Vol. 207, No. 1, 2010, str. 1-14, dostupno na: https://EconPapers.repec.org/RePEc:eee:ejores:v:207:y:2010:i:1:p:1-14

[57] Abdolshah, M., "A review of resource-constrained project scheduling problems (rcpsp) approaches and solutions", International Transaction Journal of Engineering, Management, Applied Sciences and Technologies, 2014.

[58] Deckro, R. F., Winkofsky, E. P., Hebert, J. E., Gagnon, R., "A decomposition approach to multi-project scheduling", European Journal of Operation Research, Vol. 51, 1991, str. 110-118.

[59] Icmeli, O., Rom, W. O., "Solving the resource-constrained project scheduling problem with optimization subroutine library", Computers and Operation Research, Vol. 23, 1996, str. 801-817.

[60] Pritsker, A. A. B., Watters, L., Wolfe, P., “Multiproject scheduling with limited resources: A zero-one programming approach”, Management Science, Vol. 16, 1969, str. 93-108.

[61] Schrage, L., “Solving resource-constrained network problems by implicit enumeration - nonpreemptive case”, Operations Research, Vol. 18, 1970, str. 263-278.

[62] Patterson, J., Talbot, F., Slowinski, R., Weglarz, J., “Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained project scheduling problems”, European Journal of Operational Research, Vol. 49, 1990, str. 68-79.

[63] Stinson, J. P., Davis, E. W., Khumawala, B. M., “Multiple resource-constrained scheduling using branch and bound”, AIIE Transactions, Vol. 10, 1978, str. 252-259.

[64] Demeulemeester, E., Herroelen, W., “A branch-and-bound procedure for the multiple resource-constrained project scheduling problem”, Management Science, Vol. 38, 1992, str. 1083-1818.

[65] Brucker, P., Knust, S., Solving Large-Sized Resource-Constrained Project Scheduling Problems, 01 1999, str. 27-51.

[66] Elmaghraby, S., “Resource allocation via dynamic programming in activity networks”, European Journal of Operational Research, Vol. 64, 1993, str. 199-215.

[67] Herroelen, W., Leus, R., “The construction of stable project baseline schedules”, European Journal of Operational Research, Vol. 156, No. 3, 2004, str. 550-565, dostupno na: https://EconPapers.repec.org/RePEc:eee:ejores:v:156:y:2004:i:3:p:550-565

[68] Fisher, M. L., “Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I”, Operations Research, Vol. 21, No. 5, October 1973, str. 1114-1127, dostupno na: https://ideas.repec.org/a/inm/oropre/v21y1973i5p1114-1127.html

[69] Bell, C. E., Park, K., “Solving resource-constrained project scheduling problems by a* search”, Naval Research Logistics (NRL), Vol. 37, No. 1, 1990, str. 61-84, dostupno na: https://onlinelibrary.wiley.com/doi/abs/10.1002/1520-6750%28199002%2937%3A1%3C61%3A%3AAID-NAV3220370104%3E3.0.CO%3B2-S

[70] Alcaraz, J., Maroto, C., Ruiz, R., “Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms”, The Journal of the Operational Research Society, Vol. 54, No. 6, 2003, str. 614–626, dostupno na: http://www.jstor.org/stable/4101753

[71] Merkle, D., Middendorf, M., Schmeck., H., “Ant colony optimization for resource-constrained project scheduling”, IEEE Transactions on Evolutionary Computation, Vol. 6, 2002, str. 333–346.

[72] Coelho, J., Tavares, L., “Comparative analysis of meta–heuristics for the resource constrained project scheduling problem”, 2003.

[73] Fleszar, K., Hindi, K. S., “Solving the resource-constrained project scheduling problem by a variable neighbourhood search”, European Journal of Operational Research, Vol. 155, No. 2, 2004, str. 402 - 413, financial Risk in Open Economies, dostupno na: http://www.sciencedirect.com/science/article/pii/S0377221702008846

[74] Tormos, P., Lova, A., “A competitive heuristic solution technique for resource-constrained project scheduling”, Annals of Operations Research, Vol. 102, No. 1, Feb 2001, str. 65–81, dostupno na: https://doi.org/10.1023/A:1010997814183

[75] Boctor, F. F., “Some efficient multi-heuristic procedures for resource-constrained project scheduling”, European Journal of Operational Research, Vol. 49, No. 1, 1990, str. 3-13, dostupno na: https://EconPapers.repec.org/RePEc:eee:ejores:v:49:y:1990:i:1:p:3-13

[76] Li, K. Y., Willis, R. J., “An iterative scheduling technique for resource-constrained project scheduling”, European Journal of Operational Research, Vol. 56, No. 3, 1992, str. 370-379, dostupno na: https://EconPapers.repec.org/RePEc:eee:ejores:v:56:y:1992:i:3:p:370-379

[77] Chenga, R., Gena, M., “Resource constrained project scheduling problem using genetic algorithms”, Intelligent Automation and Soft Computing, Vol. 3, No. 3, 1997, str. 273-286.

[78] Whitley, D., Watson, J. P., Complexity Theory and the No Free Lunch Theorem. Boston, MA: Springer US, 2005, str. 317–339, dostupno na: https://doi.org/10.1007/0-387-28356-0_11

[79] Wolpert, D., Macready, W., “No free lunch theorems for optimization”, IEEE Transactions on Evolutionary Computation, Vol. 4, 1997, str. 67–82.

[80] Oguz, O., Bala, H., “A comparative study of computational procedures for the resource constrained project scheduling problem”, European Journal of Operational Research, Vol. 72, No. 2, 1994, str. 406-416, dostupno na: https://EconPapers.repec.org/RePEc:eee:ejores:v:72:y:1994:i:2:p:406-416

[81] Coelho, J., Valadares Tavares, L., "Comparative analysis on approximation algorithms for the resource constrained project scheduling problem", in International Workshop on Project Management and Scheduling, Valencia, 03 2002.

[82] Anagnostopoulos, K., Koulinas, G., "A genetic hyperheuristic algorithm for the resource constrained project scheduling problem", in Evolutionary Computation (CEC), 2010, str. 1-6.

[83] Kolisch, R., Hartmann, S., Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. Boston, MA: Springer US, 1999, str. 147–178, dostupno na: https://doi.org/10.1007/978-1-4615-5533-9_7

[84] Poli, R., Langdon, W. B., McPhee, N. F., A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008, (With contributions by J. R. Koza).

[85] Koza, J. R., Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press, 1992.

[86] Koza, J. R., "Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems", 1990.

[87] Espejo, P. G., Ventura, S., Herrera, F., "A survey on the application of genetic programming to classification", Trans. Sys. Man Cyber Part C, Vol. 40, No. 2, Mar. 2010, str. 121–144, dostupno na: http://dx.doi.org/10.1109/TSMCC.2009.2033566

[88] Burke, E. K., Hyde, M. R., Kendall, G., Woodward, J., "Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one", in Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ser. GECCO '07. New York, NY, USA: ACM, 2007, str. 1559–1565, dostupno na: http://doi.acm.org/10.1145/1276958.1277273

[89] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., "Hyper-heuristics: a survey of the state of the art", Journal of the Operational Research Society, Vol. 64, No. 12, Dec 2013, str. 1695–1724, dostupno na: https://doi.org/10.1057/jors.2013.71

[90] Kumar, R., Joshi, A. H., Banka, K. K., Rockett, P. I., "Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming", in Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, ser. GECCO '08. New York, NY, USA: ACM, 2008, str. 1227–1234, dostupno na: http://doi.acm.org/10.1145/1389095.1389335

[91] Özcan, E., Parkes, A. J., "Policy matrix evolution for generation of heuristics", in Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, ser. GECCO '11. New York, NY, USA: ACM, 2011, str. 2011–2018, dostupno na: http://doi.acm.org/10.1145/2001576.2001846

[92] Burke, E. K., Hyde, M. R., Kendall, G., Woodward, J., "Automating the packing heuristic design process with genetic programming", Evolutionary Computation, Vol. 20, No. 1, 2012, str. 63-89, pMID: 21609273, dostupno na: https://doi.org/10.1162/EVCO_a_00044

[93] Pillay, N., "Evolving hyper-heuristics for the uncapacitated examination timetabling problem", Journal of the Operational Research Society, Vol. 63, No. 1, Jan 2012, str. 47–58, dostupno na: https://doi.org/10.1057/jors.2011.12

[94] Bader-El-Den, M., Poli, R., Fatima, S., "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework", Memetic Computing, Vol. 1, No. 3, Oct 2009, str. 205, dostupno na: https://doi.org/10.1007/s12293-009-0022-y

[95] Oltean, M., Dumitrescu, D., "Evolving tsp heuristics using multi expression programming", in Computational Science - ICCS 2004, Bubak, M., van Albada, G. D., Sloot, P. M. A., Dongarra, J., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, str. 670–673.

[96] Beham, A., Kofler, M., Wagner, S., Affenzeller, M., "Agent-based simulation of dispatching rules in dynamic pickup and delivery problems", in 2009 2nd International Symposium on Logistics and Industrial Informatics, Sep. 2009, str. 1-6.

[97] Vonolfen, S., Beham, A., Kommenda, M., Affenzeller, M., "Structural synthesis of dispatching rules for dynamic dial-a-ride problems", in Computer Aided Systems Theory - EUROCAST 2013, Moreno-Díaz, R., Pichler, F., Quesada-Arencibia, A., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, str. 276–283.

[98] Luke, S., Panait, L., "A comparison of bloat control methods for genetic programming", Evolutionary Computation, Vol. 14, No. 3, 2006, str. 309-344, dostupno na: https://doi.org/10.1162/evco.2006.14.3.309

[99] Whigham, P. A., Dick, G., "Implicitly controlling bloat in genetic programming", IEEE Transactions on Evolutionary Computation, Vol. 14, No. 2, April 2010, str. 173-190.

[100] Poli, R., McPhee, N. F., Parsimony Pressure Made Easy: Solving the Problem of Bloat in GP. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, str. 181–204, dostupno na: https://doi.org/10.1007/978-3-642-33206-7_9

[101] Brameier, M. F., Banzhaf, W., Linear Genetic Programming, 1st ed. Springer, 2010.

[102] Mckay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., O'Neill, M., "Grammar-based genetic programming: A survey", Genetic Programming and Evolvable Machines, Vol. 11, No. 3-4, Sep. 2010, str. 365–396, dostupno na: http://dx.doi.org/10.1007/s10710-010-9109-y

[103] Palka, D., Zachara, M., "Automatic grammar induction for grammar based genetic programming", in Artificial Intelligence and Soft Computing, Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L. A., Zurada, J. M., (ur.). Cham: Springer International Publishing, 2015, str. 350–360.

[104] Miller, J. F., Cartesian Genetic Programming. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, str. 17–34, dostupno na: https://doi.org/10.1007/978-3-642-17310-3_2

[105] Poli, R., "Parallel distributed genetic programming", 1996.

[106] Goldberg, D. E., Genetic Algorithms in Search, Optimization and Machine Learning, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[107] Blickle, T., Thiele, L., "A comparison of selection schemes used in evolutionary algorithms", Evolutionary Computation, Vol. 4, No. 4, Dec 1996, str. 361-394.

[108] Wieczorek, W., Czech, Z. J., "Selection schemes in evolutionary algorithms", in Intelligent Information Systems 2002, Kłopotek, M. A., Wierzchoń, S. T., Michalewicz, M., (ur.). Heidelberg: Physica-Verlag HD, 2002, str. 185–194.

[109] Bickel, A. S., Bickel, R. W., "Tree structured rules in genetic algorithms", in Genetic Algorithms and their Applications: Proceedings of the second International Conference on Genetic Algorithms, Grefenstette, J. J., (ur.). MIT, Cambridge, MA, SAD: Lawrence Erlbaum Associates, 1987, str. 77–81.

[110] Cramer, N. L., "A representation for the adaptive generation of simple sequential programs", in Poceedings of an International Conference on Genetic Algorithms and the Applications, Grefenstette, J. J., (ur.), Carnegie-Mellon University, Pittsburgh, PA, SAD, 1985, str. 183–187.

[111] Koza, J. R., Genetic Programming II: Automatic Discovery of Reusable Programs. Cambridge, MA, USA: MIT Press, 1994.

[112] O'Reilly, U.-M., "An analysis of genetic programming", Doktorski rad, Carleton University, Ottawa-Carleton Institute for Computer Science, Ottawa, Ontario, Canada, 1995.

[113] Koza, J. R., III, F. H. B., Andre, D., Keane, M. A., "Four problems for which a computer program evolved by genetic programming is competitive with human performance", in Proceedings of the 1996 IEEE International Conference on Evolutionary Computation,, Vol. 1. IEEE Press, 1996, str. 1–10.

[114] Wittemann, F., "Psplib resource constrained project scheduling problem", dostupno na: http://www.om-db.wi.tum.de/psplib, srpanj 2019.

[115] Kolisch, R., Sprecher, A., "Psplib - a project scheduling problem library: Or software - orsep operations research software exchange program", European Journal of Operational Research, Vol. 96, No. 1, 1997, str. 205 - 216, dostupno na: http://www.sciencedirect.com/science/article/pii/S0377221796001701

[116] Kolisch, R., Sprecher, A., Drexel, A., "Characterization and generation of a general class of resource-constrained project scheduling problems", 2001.

[117] Jakobovic, D., et al., "Evolutionary computation framework", dostupno na: http://ecf.zemris.fer.hr/, srpanj 2019.

[118] Mei, Y., Zhang, M., "A comprehensive analysis on reusability of gp-evolved job shop dispatching rules", in 2016 IEEE Congress on Evolutionary Computation (CEC), July 2016, str. 3590-3597.

[119] Polikar, R., "Ensemble learning", Ensemble Machine Learning, Vol. 4, 01 2012, str. 1-34.

[120] Breiman, L., "Bagging predictors", Machine Learning, Vol. 24, No. 2, Aug 1996, str. 123–140, dostupno na: https://doi.org/10.1023/A:1018054314350

[121] Freund, Y., Schapire, R. E., "A decision-theoretic generalization of on-line learning and an application to boosting", Journal of Computer and System Sciences, Vol. 55, No. 1, 1997, str. 119 - 139, dostupno na: http://www.sciencedirect.com/science/article/pii/S002200009791504X

[122] Bhowan, U., Johnston, M., Zhang, M., Yao, X., "Evolving diverse ensembles using genetic programming for classification with unbalanced data", IEEE Transactions on Evolutionary Computation, Vol. 17, No. 3, June 2013, str. 368-386.

[123] Bhowan, U., Johnston, M., Zhang, M., Yao, X., "Reusing genetic programming for ensemble selection in classification of unbalanced data", IEEE Transactions on Evolutionary Computation, Vol. 18, No. 6, Dec 2014, str. 893-908.

[124] Folino, G., Pizzuti, C., Spezzano, G., "Training distributed gp ensemble with a selective algorithm based on clustering and pruning for pattern classification", IEEE Transactions on Evolutionary Computation, Vol. 12, No. 4, Aug 2008, str. 458-468.

[125] Folino, G., Pizzuti, C., Spezzano, G., "Gp ensemble for distributed intrusion detection systems", in Pattern Recognition and Data Mining, Singh, S., Singh, M., Apte, C., Perner, P., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, str. 54–62.

[126] Đurasević, M., Jakobović, D., "Creating dispatching rules by simple ensemble combination", Journal of Heuristics, May 2019.

[127] Iba, H., "Bagging, boosting, and bloating in genetic programming", in Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2, ser. GECCO'99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, str. 1053–1060, dostupno na: http://dl.acm.org/citation.cfm?id=2934046.2934063

[128] Paris, G., Robilliard, D., Fonlupt, C., "Applying boosting techniques to genetic programming", in Artificial Evolution, Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, str. 267–278.

[129] Souza, L., Pozo, A., Chaves Neto, A., Correa da Rosa, J., Genetic Programming and Boosting Technique to Improve Time Series Forecasting, 10 2009.

[130] Potter, M. A., Jong, K. A. D., "A cooperative coevolutionary approach to function optimization", in Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, ser. PPSN III. London, UK, UK: Springer-Verlag, 1994, str. 249–257, dostupno na: http://dl.acm.org/citation.cfm?id=645822.670374

[131] Frayman, Y., Rolfe, B. F., Webb, G. I., "Solving regression problems using competitive ensemble models", in AI 2002: Advances in Artificial Intelligence, McKay, B., Slaney, J., (ur.). Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, str. 511–522.

[132] Bertsekas, D. P., Castanon, D. A., "Rollout algorithms for stochastic scheduling problems", in Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171), Vol. 2, Dec 1998, str. 2143-2148 vol.2.

[133] Bertsekas, D. P., Castanon, D. A., “Rollout algorithms for stochastic scheduling problems”, Journal of Heuristics, Vol. 5, No. 1, Apr 1999, str. 89–108, dostupno na: https://doi.org/10.1023/A:1009634810396

[134] Bertsekas, D. P., Rollout Algorithms for Discrete Optimization: A Survey. New York, NY: Springer New York, 2013, str. 2989–3013, dostupno na: https://doi.org/10.1007/978-1-4419-7997-1_8

[135] Čorić, R., Đumić, M., Jakobović, D., “Complexity comparison of integer programming and genetic algorithms for resource constrained scheduling problems”, in 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), May 2017, str. 1182-1188.

Životopis

Mateja Đumić rođena je 8. kolovoza 1989. godine u Vukovaru, Hrvatska. Godine 2008. upisuje preddiplomski studij matematike na Odjelu za matematiku na Sveučilištu J. J. Strossmayera u Osijeku koji završava 2011. godine. Na Odjelu za matematiku 2014. godine završava diplomski studij matematike smjer: financijska i poslovna matematika s diplomskim radom: „Cjelobrojno linearno programiranje i primjene“, a 2017. godine završava i sveučilišni nastavnički studij matematike i informatike.

U razdoblju od prosinca 2013. godine do lipnja 2014. godine radi kao mlađi softverski inženjer u tvrtki Siemens CVC. Od lipnja 2014. godine zaposlena je kao asistent na katedri za računarstvo na Odjelu za matematiku na Sveučilištu J. J. Strossmayera u Osijeku gdje sudjeluje u izvođenju više kolegija na preddiplomskom i diplomskom studiju iz područja matematike i računarstva. Njezini istraživački interesi obuhvaćaju područja evolucijskog računarstva, problema raspoređivanja, metoda optimizacija i strojnog učenja. U dosadašnjem razdoblju objavila je 2 rada u časopisima, od čega je jedan u časopisu indeksiranom u bazama CC i SCI Expanded, te 3 rada na međunarodnim znanstvenim skupovima i 1 stručni rad.

Popis objavljenih djela

Radovi u časopisima

1. Đumić, M., Šišejković, D., Čorić, R., Jakobović, D., Evolving priority rules for resource constrained project scheduling problem with genetic programming, Future Generation Computer Systems 86, rujan 2018, str. 211-221.
2. Čerkez, N., Čorić, R., Đumić, M., Matijević, D., Finding an optimal seating arrangement for employees traveling to an event, Croatian Operational Research Review 6/2, listopad 2015, str. 419-427.

Radovi na međunarodnim znanstvenim skupovima

1. Čorić, R., Đumić, M., Jelić, S., A clustering model for time-series forecasting, 42nd International Convention - MIPRO 2019, Opatija, 2019, str. 1295-1299.

2. Čorić, R., Đumić, M., Jelić, S., A Genetic Algorithm for Group Steiner Tree Problem, 41st International Convention - MIPRO 2018, Opatija, Hrvatska, 2018, str. 1113-1118.
3. Čorić, R., Đumić, M., Jakobović, D., Complexity Comparison of Integer Programming and Genetic Algorithms for Resource Constrained Scheduling Problems, 40th International ICT Convention - MIPRO 2017, Opatija, 2017, str. 1394-1400.

Stručni radovi

1. Đumić, M., Jukić Bokun, M., Euklidov algoritam, Osječki matematički list 13, 2013, str. 121-137.

Biography

Mateja Đumić was born on August 8, 1989 in Vukovar, Croatia. In 2008, she enrolled the undergraduate study of mathematics at the Department of Mathematics of the J. J. Strossmayer University of Osijek, which she completed in 2011. In 2014, she graduated from the Department of Mathematics in the field of Financial and Business Mathematics with the topic: "Integer Linear Programming and Applications". In 2017 she completed integrated undergraduate and graduate university study program in mathematics and computer science at the same Department.

From December 2013 to June 2014, she worked as a junior software engineer at Siemens CVC. Since June 2014, she has been working as a teaching and research assistant in Computer Science Research Group at the Department of Mathematics of the J. J. Strossmayer University of Osijek. She is actively involved in teaching of the several courses for undergraduate and graduate study in field of mathematics and informatics. Her research interests include the fields of evolutionary computing, scheduling problems, optimization methods and machine learning. So far, she published 2 papers in journals, one of which is in the journal indexed in Current Contents and SCI Expanded, and 3 conference papers.