I. Аутор
Име и презиме: Вишња Симић
Датум и место рођења: 19.07.1978. Крагујевац
Садашње запослење: асистент на Природно-математичком факултету Универзитета у Крагујевцу
II. Докторска дисертација
Наслов: Еластично резервисање дистрибуираних рачунарских ресурса у процесима вишекритеријумске оптимизације засноване на генетским алгоритмима
Број страница: 165
Број слика: 61
Број библиографских података: 127
Установа и место где је рад израђен: Природно-математички факултет Универзитета у Крагујевцу
Научна област (УДК): 004 (рачунарство)
Ментор: др Бобан Стојановић, ванредни професор Природно-математичког факултета Универзитета у Крагујевцу, ужа научна област: Програмирање
III. Оцена и обрана
Датум пријаве теме: 11. март 2014.
Број одлуке и датум прихватања докторске дисертације:
Комисија за оцену подобности теме и кандидата: 1. др Бобан Стојановић, ванредни професор Природно-математичког факултета Универзитета у Крагујевцу 2. др Милош Ивановић, доцент Природно-математичког факултета Универзитета у Крагујевцу 3. др Бранко Маровић, научни сарадник Електротехничког факултета Универзитета у Београду
Комисија за оцену докторске дисертације: 1. др Бобан Стојановић, ванредни професор Природно-математичког факултета Универзитета у Крагујевцу 2. др Драгић Банковић, редовни професор Државног универзитета у Новом Пазару 3. др Дејан Дивац, научни саветник у Институту за водопривреду „Јарослав Черни“, Београд 4. др Владимир Ранковић, ванредни професор Економског факултета у Крагујевцу 5. др Милош Ивановић, доцент Природно-математичког факултета Универзитета у Крагујевцу
Комисија за одбрану докторске дисертације: 1. др Бобан Стојановић, ванредни професор Природно-математичког факултета Универзитета у Крагујевцу 2. др Драгић Банковић, редовни професор Државног универзитета у Новом Пазару 3. др Дејан Дивац, научни саветник у Институту за водопривреду „Јарослав Черни“, Београд 4. др Владимир Ранковић, ванредни професор Економског факултета у Крагујевцу 5. др Милош Ивановић, доцент Природно-математичког факултета Универзитета у Крагујевцу
Датум одбране дисертације:

УНИВЕРЗИТЕТ У КРАГУЈЕВЦУ
ПРИРОДНО-МАТЕМАТИЧКИ ФАКУЛТЕТ

Вишња Симић

Еластично резервисање дистрибуираних рачунарских ресурса у процесима вишекритеријумске оптимизације засноване на генетским алгоритмима

докторска дисертација

Крагујевац, 2015.

Ова докторска дисертација је резултат вишегодишњег истраживања под менторством др Бобана Стојановића, ванредног професора у Институту за математику и информатику, Природно-математичког факултета, Универзитета у Крагујевцу. Управо њему дугујем највећу захвалност за стрпљење, конструктивне сугестије и подршку током осмишљавања, израде и прегледања овог рада. Посебну захвалност изражавам и доц. др Милошу Ивановићу који ме је увео у свет рачунарства високих перформанси, и својим креативним идејама инспирисао велики део овде представљених истраживања.

Добар део резултата приказаних у овој дисертацији настао је као плод успешне сарадње Института за математику и информатику, Природно-математичког факултета у Крагујевцу и Института за водопривреду "Јарослав Черни". Захваљујем се др Николи Миливојевићу, шефу одсека за хидроинформатику у Институту за водопривреду "Јарослав Черни", који је својим радом поставио темеље за употребу генетских алгоритама као методе оптимизације у области хидроинформатике на овим просторима. Такође се захваљујем и Владимиру Миливојевићу из Института за водопривреду "Јарослав Черни" за успешну сарадњу.

Захвалност дугујем и др Бранку Маровићу са Електротехничког факултета у Београду, како за развој Work Binder сервиса, тако и за сарадњу у реализацији научног рада који приказује део резултата ове докторске дисертације. Колегиници Ани Капларевић-Малишић са Института за Математику и информатику, Природно-математичког факултета у Крагујевцу се срдачно захваљујем за њен допринос у истраживањима која су довела до објављивања научних радова везаних за тему ове дисертације.

На крају, желим да се захвалим својим родитељима и супругу што су ми били узор и веровали у мене, те пружили безрезервну подршку и помоћ да истрајем.

Вишња Симић

Сажетак

Решавање проблема вишекритеријумске оптимизације је један од најчешћих изазова у примењеној науци и инжењерству. Генетски алгоритми су се показали као ефективан и поуздан метод када су питању сложени реални проблеми оптимизације на које се не могу применити класичне оптимизационе методе. У циљу проналажења задовољавајућих решења, генетски алгоритми морају извршити велики број евалуација јединки у популацији кроз стотине генерација. С обзиром на то да су евалуације код реалних проблема најчешће дуготрајне, оптимизација употребом генетских алгоритама може потрајати данима или чак месецима. Убрзавање оптимизационог процеса се постиже паралелизацијом генетских алгоритама и њиховом имплементацијом на дистрибуираним рачунарским платформама.

У овој дисертацији су представљена четири софтверска оквира за паралелно извршавање вишекритеријумске оптимизације засноване на генетским алгоритмима на дистрибуираним рачунарским ресурсима. Развијени оквири користе господар-слуга модел паралелизације генетског алгоритма. Сви оквири омогућавају развој евалуатора јединки независно од оптимизационих алгоритама, као и једноставно додавање нових оптимизационих алгоритама. Такође, ниједан од развијених оквира не поставља ограничења када је у питању избор програмског језика за развој програма који врше евалуацију јединки. Емпиријске студије коришћењем вештачког оптимизационог проблема, као и реалног проблема из области хидроинформатике показале су да развијени оквири достижу значајно убрзање, и то посебно за проблеме са дуготрајним евалуацијама јединки.

Међу приказаним оквирима посебно се истиче WoBinGO (***Wo**rk **Bin**der **G**enetic algorithm based **O**ptimization*) софтверски оквир, чији развој представља кључни допринос ове дисертације. WoBinGO потпуно ослобађа кориснике бриге о специфичним детаљима дистрибуираног рачунарства, и обезбеђује брз приступ ресурсима рачунарских кластера и Грида. Он врши аутоматску, еластичну алокацију дистрибуираних ресурса, користећи оно што је тренутно доступно, али заузима и додатне ресурсе чим постану слободни. За разлику од осталих представљених оквира, заузеће ресурса од стране WoBinGO-а је временски ограничено, што другим корисницима тих ресурса омогућава да им брже приступе. Експериментално добијени резултати показују да упркос адаптивној и штедљивој политици заузимања ресурса, WoBinGO пружа значајно убрзање када је реч о проблемима са дуготрајним евалуацијама. Такође, временски ограничено заузимање ресурса од стране WoBinGO-а обезбеђује да време које остали корисници ресурса проведу чекајући на њихово ослобађање буде доста краће у односу на инфраструктуре које врше статичко заузимање ресурса. Кроз студију случаја калибрације модела процуривања на хидроелектрани Вишеград анализиране су перформансе које WoBinGO испољава при решавању комплексног реалног проблема. Добијени резултати показују да се решавање реалног проблема хидроинформатике вишекритеријумском оптимизацијом заснованом на генетским алгоритмима знатно убрзава помоћу WoBinGO-а.

Abstract

One of the most frequently encountered challenges in applied science and engineering is solving multi-objective optimization problems. Genetic algorithms have proven themselves as effective and robust mechanisms when it comes to solving complex real-world optimization problems, which cannot be solved by means of classical optimization methods. In order to obtain satisfying solutions, genetic algorithms must perform a large number of evaluations of individuals in population through hundreds of generations. Due to the time-consuming evaluations found in real-world problems, it may take days or even months for the genetic algorithm to find an acceptable solution of multi-objective optimization problem. Speeding up the optimization process is achieved by parallelization of genetic algorithms and their implementation on distributed computing platforms.

This dissertation introduces four software frameworks for parallel execution of multi-objective optimization based on genetic algorithms over distributed computing resources. Developed frameworks use master-slave model of parallel genetic algorithms. Development of evaluators and optimization algorithms are independent in all frameworks, and new optimization algorithms can be easily added. Also, neither one of the developed frameworks poses any limitations when it comes to choosing programming language for the development of evaluators. Empirical studies using an artificial optimization problem, as well as a real-world problem in the field of hydroinformatics has shown that developed frameworks provide significant speed-up, especially when dealing with problems that have time-consuming evaluations.

*Among the presented frameworks, WoBinGO (**Wo**rk **Bin**der **G**enetic algorithm based **O**ptimization) software framework particularly stands out. Its development is the key contribution of this thesis. WoBinGO completely relieves the researcher burden of dealing with specific details of distributed computing, and enables fast access to resources of computing clusters and Grid. It provides automatic and elastic allocation of distributed resources, by using whatever resources are attainable at the moment, and occupying additional resources as soon as they become available. Unlike other presented frameworks, WoBinGO occupies resources only for a limited time, which enables other users to approach them faster. Experimentally obtained results show that despite WoBinGO's adaptive and frugal allocation of computing resources, it provides significant speed-up when dealing with problems that have time-consuming evaluations. Moreover, WoBinGO's limited occupation of resources significantly reduces the queuing time of other users, compared to a static pilot-job infrastructure. For the purpose of evaluating WoBinGO's performance when it comes to solving a complex real-world problem, the case study of calibration of a leakage model at the Visegrad power plant was performed. Obtained results show that solving real problem of hydroinformatics using multi-objective optimization based on genetic algorithms, can be significantly speed-up by means of WoBinGO framework.*

Садржај

Списак слика

Списак табела

Листа скраћеница

ВКГА	Вишекритеријумски Генетски Алгоритми
ВКО	Вишекритеријумска Оптимизација
ГА	Генетски Алгоритам
ГП	Генетско Програмирање
ЕА	Еволуциони Алгоритам
ЕП	Еволуционо Програмирање
ЕС	Еволуционе Стратегије
ПГА	Паралелни Генетски Алгоритам
ХЕ	Хидроелектрана
CE	Computing Element
EGI	European Grid Infrastructure
GUID	Grid Unique IDentifier
JDL	Job Description Language
LFN	Logical File Name
MPI	Message Passing Interface
SE	Storage Element
SURL	Storage URL
TPL	Task Parallel Library
TURL	Transport URL
WB	Work Binder
WCF	Windows Communication Foundation
WMS	Workload Management System
WN	Worker Node

Увод

Већина сложених инжењерских проблема подразумева оптимизацију више нелинеарних критеријума (циљних функција) под одређеним ограничењима. Критеријуми међусобно могу бити у конфликту, тако да оптимално решење у односу на један критеријум бива потпуно неприхватљиво када се у обзир узму и остали критеријуми. Јединствено решење оваквих проблема постоји само у идеалном случају, а најчешће треба испитати скуп Парето оптималних решења и у складу са додатним информацијама о проблему одабрати компромисно решење из скупа, које не може бити оптимално по свим критеријумима. Класичне методе вишекритеријумске оптимизације најчешће подразумевају један од следећа два приступа: (1) комбиновање свих циљних функција у једну функцију; (2) задржавање једне функције на месту функције циља, и премештање осталих циљних функција у скуп ограничења. Кључни недостатак класичних метода је зависност решења проблема од приоритета који је доносилац одлука на основу свог знања о проблему и тренутних потреба доделио функцијама циља. Класичне методе могу гарантовати добијање Парето оптималног решења, али увек резултују само једним решењем што најчешће не задовољава потребе доносиоца одлука коме су потребна и алтернативна решења како би могао да разматра шта се губи, а шта добија са сваким од њих. Зато ове методе, у циљу проналажења више Парето оптималних решења, захтевају репетитивне примене алгоритама. Решавање проблема вишекритеријумске оптимизације генетским алгоритмима за вишекритеријумску оптимизацију даје вишеструка Парето оптимална решења истовремено, што доносиоцима одлука пружа могућност да одаберу оно решење које је у датој ситуацији најбоље.

Генетски алгоритми (ГА) припадају класи еволуционих алгоритама као групи стохастичких оптмизационих метода које симулирају природни процес еволуције. Поступак одређивања оптималних решења сложених проблема употребом еволуционих алгоритама заснива се на природној селекцији или опстанку најспособнијих. Овај важан део еволутивног процеса подразумева да јединке које се у датим условима покажу као способније од осталих имају веће шансе да створе потомство и пренесу свој генетски материјал, чиме се добре особине умножавају. Генетски алгоритми су се показали као робустан и моћан механизам када је у питању решавање сложених реалних проблема оптимизације на које се не могу применити класичне методе.

Током извршавања генетских алгоритма у свакој генерацији се врши евалуација свих индивидуа спрам проблема који се решава. Употреба секвенцијалних алгоритама за оптимизацију подразумева да се на једном процесору редом евалуира једно по једно решење. У случају оптимизације сложених система свака евалуација може потрајати и по више десетина минута у зависности од конфигурације рачунара на коме се она извршава. Како је током извршавања алгоритма потребно евалуирати више стотина јединки у неколико стотина генерација, произилази да једна оптимизација на једном процесору може трајати месецима, што је неприхватљиво за било какву практичну примену. Мотивација за овај рад проистекла је управо из потребе да се убрза проналажење решења реалних оптимизационих проблема из области хидроинформатике. Услед сложености система који се у овој области оптимизују, попут хидроелектрана, дуготрајне евалуације јединки су неминовност. У циљу краћег извршавања генетских алгоритама врши се њихова паралелизација. Паралелно извршавање генетских алгоритама омогућено је развојем дистрибуираних платформи, попут кластера, Грида и рачунарства у Облаку (*Cloud Computing*). Међутим, имплементација паралелних генетских алгоритама на овим дистрибуираним платформама захтева значајан напор и експертско знање. Додатно, при сваком извршавању паралелне апликације на овим платформама, корисник се среће са питањем њихове заузетости од стране осталих корисника и чекањем да дође на ред. Сложеност поступка имплементације паралелних апликација на дистрибуираним архитектурама, често одвраћа истраживаче од експлоатације ових ресурса. Како се Грид рачунарство и рачунарство у Облаку развијају великом брзином и све већи број ресурса ставља на располагање корисницима, а рачунарство високих перформанси постаје све доступније, то је од великог значаја за истраживаче да их ефикасно искористе у циљу решавања реалних проблема. У том контексту, софтверски алати за поједностављивање развоја дистрибуираних апликација, којима се комплексност дистрибуираних платформи скрива од истраживача, значајно побољшавају искоришћавање ових ресурса.

Предмет истраживања у овој дисертацији је имплементација генетских алгоритама за вишекритеријумску оптимизацију у објектно оријентисаној паралелној програмској парадигми на дистрибуираним рачунарским ресурсима. Паралелизација генетских алгоритама биће извршена употребом модела „господар-слуга“ (*master-slave*), код кога господар серијски извршава главни ток генетског алгоритма, док слуге парелелно, асинхроно евалуирају јединке.

Циљеви

Циљ дисертације је развој софтверског оквира за паралелно извршавање вишекритеријумске оптимизације засноване на генетским алгоритмима на дистрибуираним рачунарским ресурсима, који укључују кластере високих перформанси и Грид. Оквир који се развија треба да омогући: (1) двојаку паралелизацију - паралелну евалуацију јединки у генетском алгоритму и паралелно извршавање више инстанци

паралелизоване оптимизације; (2) развој евалуатора јединки независно од оптимизационих алгоритама; (3) једноставно додавање нових оптимизационих алгоритама; (4) отклањање ограничења при избору програмског језика за развој програма који врше евалуацију јединки; (5) брз приступ дистрибуираним рачунарским ресурсима; (6) ослобађање корисника бриге о специфичним детаљима дистрибуираног рачунарства. Кључна особина овог сотфверског оквира је да еластичним приступом ресурсима, са временски ограниченим резервисањем, омогући осталим корисницима тих ресурса бржи приступ. Са становишта корисника самог софтверског оквира за оптимизацију, аутоматска, еластична алокација ресурса пружа додатни комфор. Она кориснику обезбеђује да не треба да чека да тачно одређене капацитети буду ослобођени, већ користи оно што је тренутно доступно, али заузима и додатне ресурсе чим постану слободни.

Полазне хипотезе

Полазне основе докторске дисертације чине резултати досадашњих истраживања на пољу паралелне вишекритеријумске оптимизације засноване на генетским алгоритмимима и њене примене у хидроинформатици, као и истраживања на пољу развоја софтвера, који омогућавају једноставну и ефикасну употребу дистрибуираних рачунарских ресурса. Хипотезе ове дисертације су:

1. Еластичном резервацијом дистрибуираних ресурса и ограниченим трајањем резервације, софтверски оквир остварује јединствен однос према осталим корисницима ресурса, обезбеђујући им несметан приступ дистрибуираним ресурсима, без обзира на дуготрајност процеса оптимизације. Еластичност у резервацији ресурса не утиче на перформансе софтверског оквира, што овакав приступ дистрибуираним ресурсима чини бољим избором од већ постојећег статичког приступа.
2. Паралелизацијом генетских алгоритама за вишекритеријумску оптимизацију у области хидроинформатике знатно се убрзава одређивање више Парето оптималних решења истовремено, што доносиоцима одлука пружа могућност да одаберу оно решење које је у датој ситуацији најбоље.

Преглед садржаја дисертације

Дисертација се састоји од седам глава и једног додатка. Након претходног увода у коме је објашњена мотивација за овај рад, циљеви којима се тежило током његове израде, као и полазне хипотезе, остатак дисертације је организован на следећи начин:

Прво поглавље даје основне дефиниције из области вишекритеријумске оптимизације и преглед метахеуристичких метода које се користе у решавању опти-

мизационих проблема. Даље се детаљније говори о генетским алгоритмима као најчешће заступљеној метахеуристичкој методи оптимизације и проблемима вишекритеријумске оптимизације који се срећу у хидроинформатици.

Друго поглавље говори о типовима паралелизације генетских алгоритама и паралелним и дистрибуираним архитектурама на којима се паралелни генетски алгоритми могу имплементирати.

Треће поглавље доноси свеобухватан преглед литературе из више области релевантних за реализацију циљева ове дисертације. Обрађена је литература о развијеним софтверским оквирима за вишекритеријумску оптимизацију засновану на генетским алгоритмима, затим о системима заснованим на пилот пословима, као и литература везана за примену генетских алгоритама за вишекритеријуску оптимизацију у хидроинформатици.

Четврто поглавље пружа детаљан опис карактеристика, архитектура и начина функционисања софтверских оквира који су развијени у настојању да се остваре наведени циљеви дисертације.

Пето поглавље излаже анализу ефикасности развијених софтверских оквира. На основу резултата спроведених експеримената извршено је међусобно упоређивање перформанси ових оквира. У последњем делу поглавља дате су смернице за оптималан избор једног од представљених оквира у складу са карактеристикама оптимизационог проблема у чијем би решавању оквир био коришћен.

Шесто поглавље представља резултате којима се потврђује прва хипотеза на којој се заснива спроведено истраживање. Кроз низ експеримената се доказује да се еластичном резервацијом дистрибуираних ресурса и ограниченим трајањем резервације може осталим корисницима обезбедити несметан приступ дистрибуираним ресурсима, а да се то при томе не нарушавају перформансе софтверског оквира.

Седмо поглавље демонстрира примену WoBinGO софтверског оквира у решавању захтевног проблема хидроинформатике - калибрације модела процуривања на хидроелектрани Вишеград. На основу резултата спроведених експеримената анализирају се перформансе које WoBinGO испољава при решавању реалног проблема са нагласком на еластичност у заузимању рачунарских ресурса. Приказани резултати показују да се употребом паралелних ГА за решавање конкретног проблема вишекритеријумске оптимизације у области хидроинформатике знатно убрзава одређивање оптималних решења, и на тај начин потврђују другу хипотезу овог рада.

Након описаних поглавља дат је закључак са прегледом резултата и предлози праваца даљег развоја.

На крају дисертације је и додатак у коме је дат *NSGA-II* алгоритам за решавање проблема вишекритеријумске оптимизације и опис специфичних оператора који се користе у његовом раду.

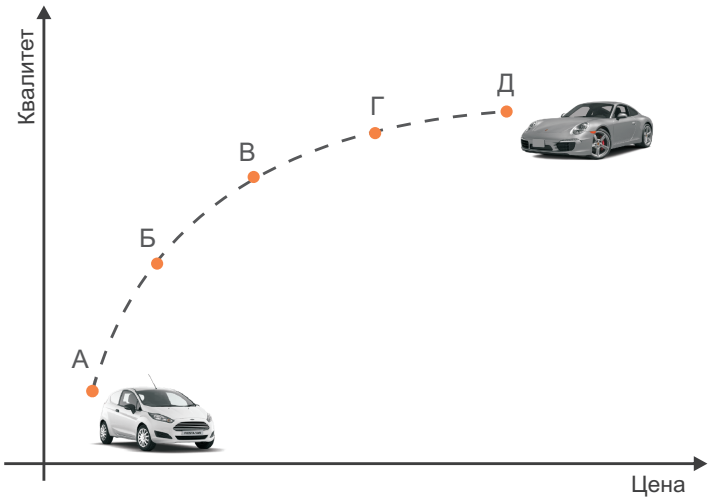
Глава 1

Теоријске основе вишекритеријумске оптимизације

Готово сваки сложен проблем реалног окружења подразумева истовремену оптимизацију неколико несамерљивих и често супротстављених циљева. Одређивање једног или више оптималних решења оваквих проблема, назива се *вишекритеријумска оптимизација - ВКО* (*Multi-Objective Optimization - MOO*). У прошлости су, услед недостатка одговарајућих техника за решавање проблема ВКО, ови проблеми вештачки свођени на проблеме једнокритеријумске оптимизације и решавани као такви. За свођење проблема ВКО на проблем једнокритеријумске оптимизације, најчешће се користи један од следећа два приступа:

- све функције циља се комбинују у једну функцију коришћењем знања о проблему који се решава,
- једна функција се задржава на месту функције циља, а остале функције циља се премештају у скуп ограничења, при чему се на основу знања о проблему одређује ограничавајућа вредност за свако ограничење добијено од функције циља.

Кључни недостатак класичних метода је зависност решења проблема од приоритета који је доносилац одлука на основу свог знања о проблему и тренутних потреба доделио функцијама циља. Додатно, примена ових метода увек резултује само једним оптималним решењем, док проблеми ВКО, уместо једног оптималног решења, имају низ компромисних оптималних решења. Дакле, важно је наћи не само једно оптимално решење, већ што је више могуће таквих оптималних решења која представљају компромис између различитих циљева оптимизације. Откривање великог броја компромисних оптималних решења пружа могућност доносиоцима одлука да разматрају шта се губи, а шта добија са сваким од њих. Класичне методе захтевају репетитивне примене алгоритама како би се пронашло више оптималних решења.



Слика 1.1: Хипотетички Парето фронт за проблем доношења одлуке приликом куповине аутомобила

Узмимо у обзир релативно чест проблем куповине новог аутомобила [26]. Цене аутомобила варирају у великом опсегу, од неколико хиљада до неколико стотина хиљада еура. Посматрајмо два хипотетичка аутомобила (слика 1.1) од којих један кошта десет хиљада (решење А), а други сто хиљада еура (решење Д). Када би цена била наш једини критеријум приликом доношења одлуке о куповини, оптимално решење би било решење А. Ако у процес одлучивања укључимо и критеријум квалитета који расте са ценом аутомобила, тада се овај двокритеријумски проблем оптимизације не може посматрати као два независна оптимизациона проблема, таква да је оптимум једног проблема решење А, а оптимум другог решење Д.

Између ова два есктремна решења, постоје и многа друга (на пример Б, В и Г), која представљају компромисе између критеријума цене и критеријума комфора. Ако посматрамо било која два компромисна решења (на пример А и В), прво решење је боље од другог када се у обзир узме један од критеријума (цена), али је зато лошије по питању другог критеријума (квалитет). Дакле, сва ова решења су на неки начин оптимална, а на доносиоцу одлука је у складу са својим потребама и могућностима прихвати једно од њих.

Из преходног примера се може уочити да уместо једног оптимума, као што је случај код једнокритеријумске оптимизације, проблеми ВКО имају скуп компромисних решења, познат као Парето скуп. Скуп Парето оптималних решења садржи решења која су за сваку функцију циља супериорна у односу на сва остала решења из простора претраге, али су за неке функције циља инфериорна у односу на остала решења из овог скупа [110]. Тако се код ВКО, разматрањем свих критеријума оптимизације истовремено, мора пронаћи скуп компромисних оптималних решења. Након проналажења низа компромисних оптималних решења, доносилац одлука може разматрањем на квалитативно вишем нивоу да направи избор. Ова процедура ВКО

шематски је приказана на слици 1.2.

Слика 1.2: Шематски приказ процедуре решавања проблема ВКО

1.1 Формулација проблема вишекритеријумске оптимизације

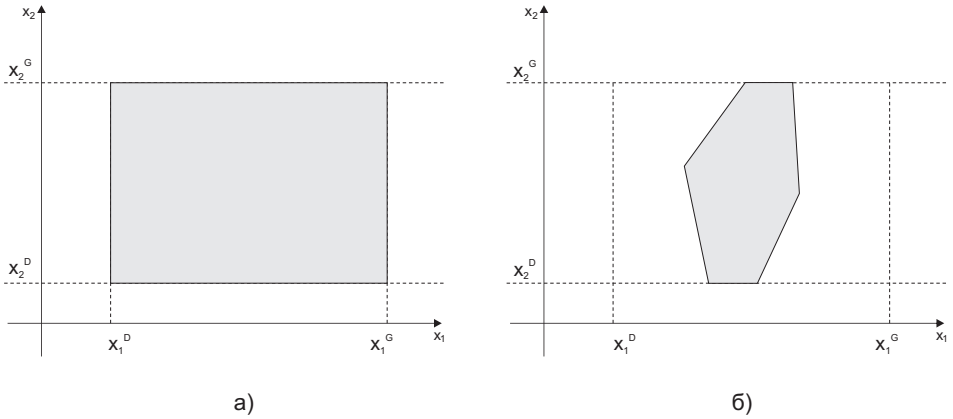
Проблем ВКО се у општем случају може дефинисати на следећи начин:

$$\begin{aligned} \text{Одредити } \min/\max \quad & f_m(\mathbf{x}) && m = 1, 2, \ldots, M \\ \text{тако да} \quad & g_j(\mathbf{x}) \leq 0 && j = 1, 2, \ldots, J \qquad (1.1) \\ & h_k(\mathbf{x}) = 0 && k = 1, 2, \ldots, K \qquad (1.2) \\ & x_i^D \leq x_i \leq x_i^G && i = 1, 2, \ldots, n \qquad (1.3) \end{aligned}$$

Решавање проблема ВКО подразумева проналажење n-димензионог вектора $\mathbf{x}^* = \vec{x^*} = (x_1^*, x_2^*, \ldots, x_n^*)$ који задовољава услове (1.1), (1.2), (1.3) и оптимизује M циљних функција $f_1, f_2, \ldots, f_M$. Услов (1.3) дефинише *простор претраге* (слика 1.3 а) за вредности које су координате вектора $\vec{x}$, тако што поставља доњу x_i^D и горњу x_i^G границу ових вредности. Решење проблема одређују и J неједнакости (1.1) и K једнакости (1.2). Функције $g_j(\mathbf{x})$ и $h_k(\mathbf{x})$ се називају функције ограничења и оне дефинишу *допустиви простор претраге* (слика 1.3 б).

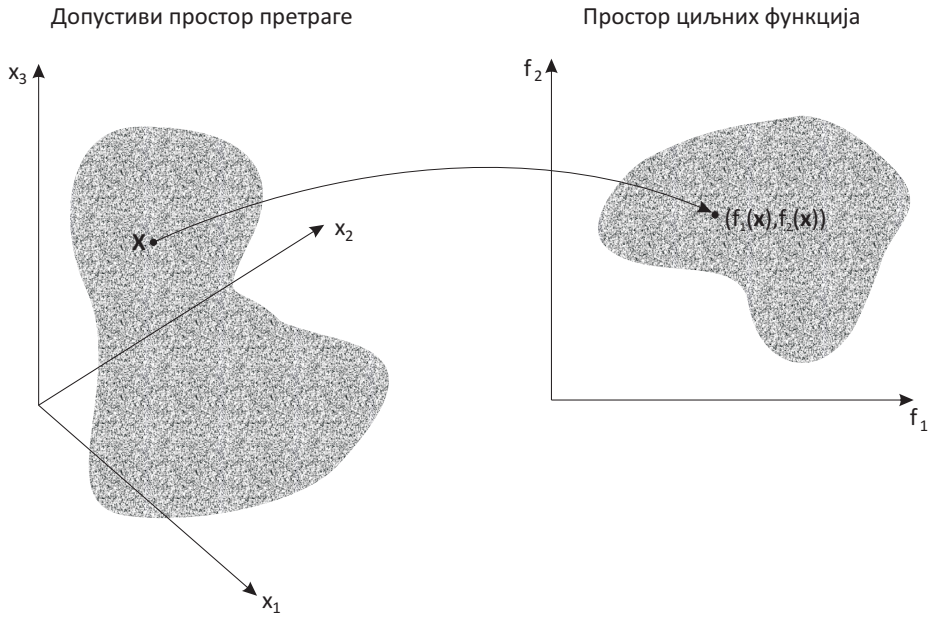
У даљем тексту ћемо (осим ако није другачије наглашено) подразумевати да под оптимизицијом функција подразумевамо њихову минимизацију што не утиче на општост проблема, јер важи да је $\max f = -\min(-f)$.

Код ВКО, циљне функције формирају вишедимензиони простор циљних функција, $\mathcal{Z}$ (циљни простор - *objective space*). За свако решење $\mathbf{x}$ из допустивог простора претраге, постоји тачка у простору циљних функција, у ознаци $\mathbf{f}(\mathbf{x}) = (f_1(\vec{x}), f_2(\vec{x}), \ldots,$



Слика 1.3: Различити простори претраге за различите типове ограничења

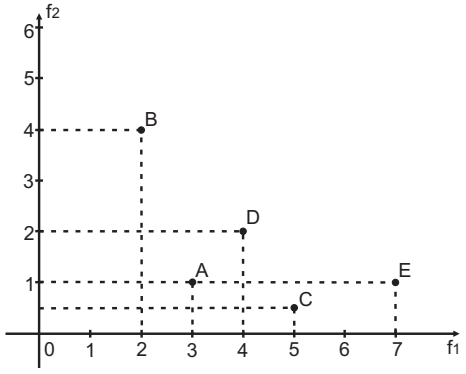
$f_K(\vec{x})$). Слика 1.4 илуструје пресликавање допустивог простора претраге у простор циљних функција. Иако алгоритми за решавање проблема ВКО проналазе решења у допустивом простору претраге, многи алгоритми за ВКО засновани на ГА у поступку претраге користе информације из простора циљних функција.



Слика 1.4: Пресликавање допустиви простор претраге у циљни простор

У случају једнокритеријумске оптимизације, допустиви простор претраге је потпуно уређен у односу на циљну функцију f, тј. за свака два решења $\vec{x}$ и $\vec{y}$ из допустивог простора претраге важи да је $f(\vec{x}) \leq f(\vec{y})$ или да је $f(\vec{y}) \leq f(\vec{x})$. Међутим, када је вишекритеријумска оптимизација у питању, допустиви простор претраге у општем случају није потпуно, већ је парцијално уређен, тако да једно решење може бити боље од другог решења за једну циљну функцију, а лошије када се узме у обзир нека од осталих функција циља. Отуда је уведен појам релације доминације [121]:

Дефиниција 1 (Релација доминације) *Вектор $\vec{x_1}$ доминира вектор $\vec{x_2}$ (у ознаци $\vec{x_1} \succ \vec{x_2}$) ако је за сваку функцију циља вектор $\vec{x_1}$ бар подједнако добар као вектор $\vec{x_2}$ и ако постоји бар једна функција циља за коју је вектор $\vec{x_1}$ строго бољи него вектор $\vec{x_2}$.*



Слика 1.5: Пример пет решења у простору циљних функција

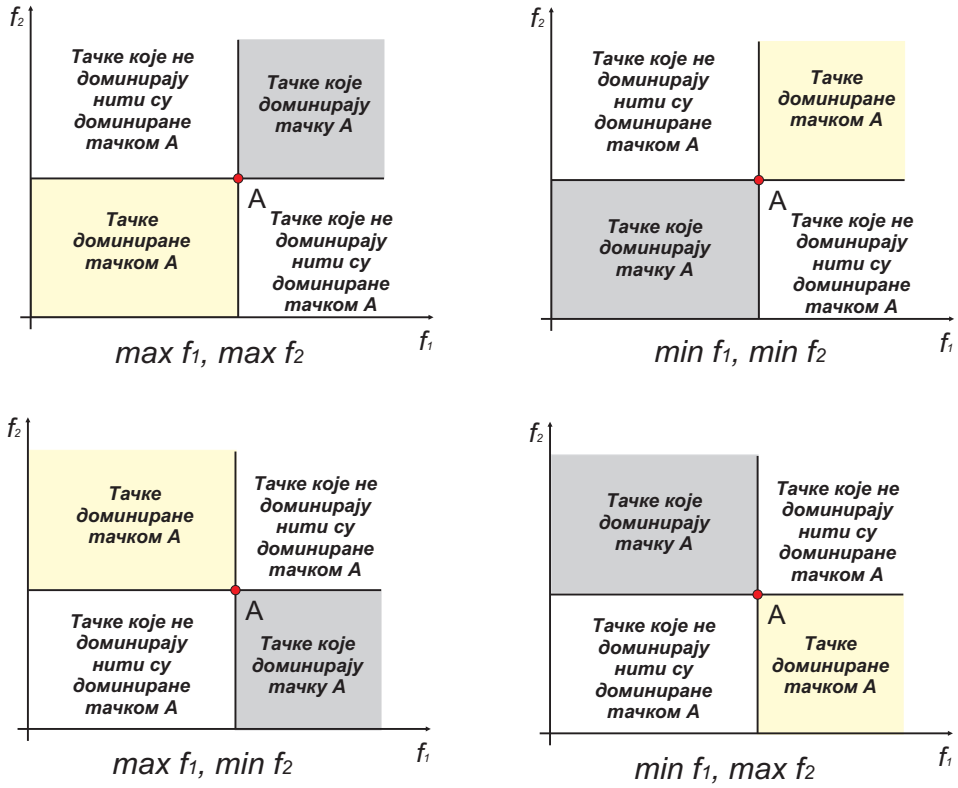
Нека је дат проблем оптимизације са две функције циља f_1 и f_2, при чему функцију f_1 треба максимизовати, а функцију f_2 минимизовати. На слици 1.5 представљено је пет различитих решења овог проблема у простору циљних функција. Коришћењем претходне дефиниције релације доминације, видимо да сва решења доминирају решење B. У општем случају, код дводимензионих проблема ВКО могу се уочити четири различита положаја области доминације у односу на фиксирану тачку A. Као што се може видети на слици 1.6 положај области доминације зависи од захтева оптимизације циљних функција – да ли је обе функције потребно максимизовати, или минизовати, или се једна функција максимизује док се друга минимизује.

Кроз пример са слике 1.5 даље се може уочити да је решење D боље од решења A у односу на функцију f_1, али је у односу на функцију f_2 решење D лошије од решења A. Дакле, не може се закључити ни да решење D доминира решење A, нити да решење A доминира решење D. За оваква два решења се каже да се међусобно не доминирају. Слично се решења C и E међусобно не доминирају, али доминирају сва остала приказана решења. Решења C и E тако формирају посебан скуп решења која су боља од свих осталих решења - *недоминирани скуп.*

Дефиниција 2 (Недоминирани скуп) *У скупу решења P, подскуп недоминираних решења P' чине она решења из P која нису доминирана ниједним чланом скупа P.*

Дефиниција 3 (Глобални Парето оптимални скуп) *Недоминирани скуп решења читавог допустивог простора претраге је глобални Парето оптимални скуп.*

Слика глобалног Парето оптималног скупа у простору циљних функција назива се *Парето фронт* (слика 1.7).

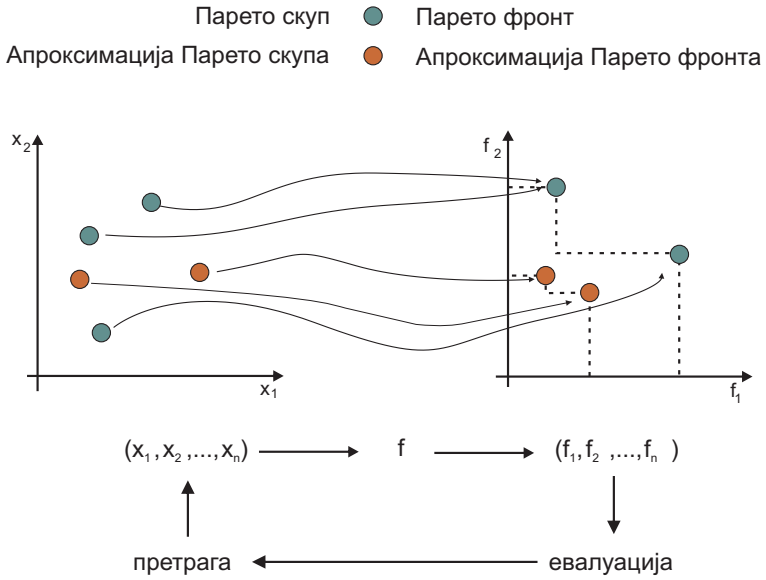


Слика 1.6: Могући положаји области доминације

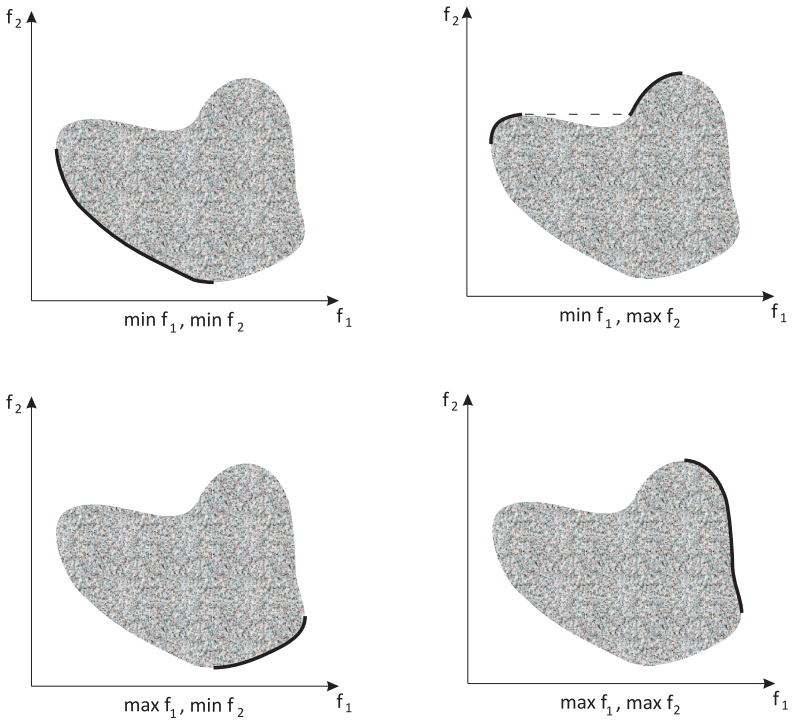
У претходном примеру (слика 1.5) тачке C и E чине Парето фронт. На слици 1.8 су обележени Парето фронтови за четири различита сценарија која могу наступити код дводимензионог проблема ВКО: обе функције треба минимизовати; прву функцију треба минимизовати, а другу максимизовати; прву функцију треба максимизовати, а другу минимизовати; обе функције треба максимизовати.

Циљ алгоритама за вишекритеријумску оптимизацију је проналажење Парето оптималног скупа. Проналажење целокупног Парето оптималног скупа је услед његове величине практично немогуће, тако да се приступа проналажењу скупа решења који најбоље репрезентује Парето оптимални скуп – најбољи познати Парето скуп, то јест најбоља апроксимација Парето скупа (слика 1.7). Имајући ово у виду, методе за решавање проблема ВКО треба да задовоље следеће услове:

- Да што боље апроксимирају прави Парето фронт. Идеално, најбољи познати Парето скуп треба да буде подскуп правог Парето скупа.
- Пронађена решења треба да буду разнолика и равномерно распоређена, како би доносилац одлука стекао праву слику о могућим компромисима.



Слика 1.7: Илустрација Парето скупа у допустивом простору претраге и Парето фронта, као његове слике у простору цильних функција. Евалуирањем вредности цильних функција за потенцијална решења врши се усмеравање претраге ка оптимуму



Слика 1.8: Парето оптимална решења за четири могуће комбинације две цильне функције

1.2 Преглед метахеуристичких метода за решавање проблема оптимизације

Методе за решавање оптимизационих проблема претрагом допустивог простора долазе до решења проблема. *Егзактне* методе за решавање оптимизационих проблема, попут линеарног, нелинеарног и динамичког програмирања [74], гарантују налажење оптималног решења или конвергенцију ка њему, али се могу применити само над *добро структурираним* проблемима, тј. проблемима за које је могуће дефинисати прецизан математички модел. Међутим, комплексност и висок степен нелинеарности реалних проблема најчешће онемогућава њихово ефикасно математичко моделирање, те се у пракси срећемо са *слабо структурираним* проблемима за чије решавање не могу бити употребљене егзактне методе. Типични слабо структурирани проблеми су с једне стране, проблеми комбинаторне оптимизације (дискретни проблеми), као што је прављење распореда часова, а са друге стране су континуални проблеми попут одређивања непознатих параметара модела минимизовањем разлике између израчунатих и измерених вредности података који карактеришу систем који се моделира. Додатно, постоје и добро структурирани проблеми чија решења, услед великих димензија проблема, не могу у разумном времену бити одређена егзактним методама.

У циљу превазилажења ограничења егзактних метода оптимизације, 70-тих година двадесетог века интезивира се употреба апроксимативних метода оптимизације. Ове методе не гарантују проналажење глобално оптималног решења, али значајно скраћују време потребно за проналажење задовољавајућих решења. У апроксимативне методе решавања оптимизационих метода спадају хеуристике и метахеуристике.

Историјски гледано, прво се почело са употребом хеуристика. Међу основним хеуристикама најчешће се прави подела на конструктивне хеуристике и хеуристике засноване на локалној претрази. Конструктивни алгоритми генеришу решења "од нуле" додајући им одговарајуће компоненте све док не добију комплетно решење. Одлука о томе која ће компонента бити додата у решење које се конструише најчешће се заснива на некој хеуристици, то јест знању о проблему које нам можда може помоћи у његовом решавању. На пример, у случају решавања општепознатог проблема трговачког путника, рута би се конструктивном методом формирала тако што би се кренуло из једног града и у сваком наредном кораку ишло ка непосећеном граду који је најближи граду у коме се тренутно налазимо. Ипак, овакав начин конструисања руте у великом броју случајева не даје оптимално решење. Генерално, конструктивне апроксимативне методе веома брзо долазе до решења, али најчешће дају лошија решења од хеуристика базираних на локалној претрази.

Хеурстике базиране на техникама локалне претраге полазе од почетног решења које задовољава ограничења проблема и итеративно покушавају да га замене бољим решењем из одговарајуће околине тренутног решења. У свакој итерацији,

претрага се помера ка решењу које се веома мало (локално) разликује од тренутног решења, све док се не дође до локалног оптимума. Одабир структуре околине прикладне за проблем који се решава пресудно утиче на перформансе алгоритма локалне претраге. На пример, у случају проблема трговачког путника може се одабрати такозвана "k-промена" структура околине код које се два суседна решења разликују у највише k грана графа који описује руту трговачког путника. Алгоритам тада из корака у корак испитује да ли се међу свим решењима чијих k грана на други начин повезује чворове графа у односу на тренутно решење, налази неко решење са рутом краћом од руте тренутног решења. Ако такво решење постоји, онда оно постаје ново тренутно решење. Хеурстике базиране на техникама локалне претраге лако западају у локални минимум, а њихов резултат у великој мери зависи од полазног решења.

Метахеуристике улазе у широку употребу 80-тих година двадесетог века као нова врста апроксимативних метода које комбинују основне хеуристичке принципе у циљу ефикаснијег и ефектнијег истраживања простора претраге које доводи до квалитетнијих решења оптимизационих проблема.

1.2.1 Класификација метахеуристика

Термин метахеуристика који је увео *Glover* [48], настао је од грчких речи *heuriskein* што значи наћи или открити и *meta* што значи изнад или над. Опште прихваћена дефиниција метахеуристика не постоји, али се може рећи да су то стратегије које ефикасно воде процес претраге простора решења у циљу проналажења решења блиског оптималном.

Класификација метахеуристика се може извршити на више начина, али у складу са радовима [13],[5] уочићемо две категорије: метахеуристике базиране на путањи и популацијске метахеуристике (слика 1.9). Главна разлика између метода које припадају овим категоријама је број привремених решења који се користе у сваком кораку алгоритма. Технике засноване на путањи почињу са једним иницијалним решењем и у сваком кораку претраге тренутно решење бива замењено другим (често и најбољим) решењем пронађеним у околини тренутног решења. Кретањем кроз простор претраге од једног ка наредном решењу алгоритам формира путању, одакле су ове технике и добиле назив. Популацијски алгоритми полазе од популације решења коју побољшавају кроз итеративни процес. У свакој генерацији процеса један део или цела популација бивају замењени новогенерисаним јединкама. Примери метахеуристика заснованих на путањи су симулирано каљење [34] и табу претраживање [48], док се у популацијске метахеуристике убрајају еволуциони алгоритми (EA), оптимизација заснована на мрављој колонији [32] и друге. У последње време актуелна је и хибридизација метахеуристика при којој се на разне начине комбинују метахеуристике, како међусобно, тако и са другим оптимизационим техникама [12].

Слика 1.9: Класификација метахеуристка

Симулирано каљење

Овај метод је развијен по аналогији са металуршким процесом каљења, при коме се материјал загрева до високе температуре, а онда постепено, контролисано хлади. Ако је хлађење довољно споро, тада материјал тежи да постигне стање минималне енергије, тј. да очврсне у правилну кристалну структуру. Овај принцип се примењује на решавање оптимизационих проблема тако што једно решење одговара једној могућој конфигурацији атома, вредност функције циља његовој унутрашњој енергији, а прелаз у суседно решење промени енергетског стања. Алгоритам симулираног каљења полази од случајно одабраног решења и почетне температуре. У свакој итерацији алгоритма бира се једно од решења из околине тренутног решења, и у случају да је ово суседно решење боље од тренутног решења оно постаје ново тренутно решење. Ако одабрано решење из околине тренутног решења није боље од тренутног решења, оно постаје ново тренутно решење са вероватноћом која зависи од параметра температуре. Што је погоршање које се добија одабиром суседног решења мање, то је вероватноћа његовог прихватања већа. Температура има задатак да контролише флексибилност прихватања погоршања функције циља, а тиме и претраживање простора решења. Полазећи од довољно велике стартне температуре у почетним итерацијама се прихватају скоро сви случајно генерисани суседи, што омогућава избегавање замки локалних минимума. Постепеним опадањем температуре смањује се вероватноћа прихватања погоршања функције циља, тако да се, после довољног броја итерација, практично прихватају само њена побољшања. Схема хлађења треба да обезбеди довољно споро смањење температуре, како би се омогућило што боље претраживање околине локалног минимума.

Табу претраживање

Табу претраживање је најчешће коришћена метахеуристика за решавање проблема комбинаторне оптимизације. Ова метода користи краткотрајну меморију како би избегла упадање у локални минимум. Краткотрајна меморија је имплементирана у виду табу листе у коју се бележе најскорије "посећена" решења у простору претраге и која забрањује кретање ка њима. Околина тренутног решења је на тај начин ограничена само на решења која се не налазе на табу листи - дозвољени скуп. У свакој итерацији најбоље решење из дозвољеног скупа постаје тренутно решење и уписује се у табу листу, док се једно решење скида са листе (најчешће по *FIFO - First In First Out* редоследу). Алгоритам се зауставља када су испуњени услови за терминацију, или када дозвољени скуп постане празан. Дужином табу листе контролише се величина дела простора претраге који ће бити истражен, и она може варирати током извршавања алгоритма, чиме се обезбеђује робустнија претрага. Табу листа некада може бити сувише рестриктивна, тако да се пружа могућност дефинисања критеријума аспирације помоћу кога се може одбацити табу статус неког потомка ако нас његов одабир води у довољно добру тачку.

Еволуциони алгоритми

Еволуциони алгоритми су група стохастичких оптмизационих метода које симулирају природни процес еволуције. Поступак одређивања оптималних решења сложених проблема употребом ЕА заснива се на природној селекцији или опстанку најспособнијих. Овај важан део еволутивног процеса подразумева да јединке које се у датим условима покажу као способније од осталих имају веће шансе да створе потомство и пренесу свој генетски материјал, чиме се добре особине умножавају и популација еволуира ка оптимуму. ЕА симулира еволуцију популације потенцијалних решења проблема ка задовољавајућим, ако не и оптималним решењима проблема, итеративном применом оператора селекције, укрштања и мутације. Селекција јединки које ће укрштањем створити потомство врши се тако да квалитетније јединке бивају одабране са већом вероватноћом него мање квалитетна решења. Квалитет јединке се назива прилагођеност (*fitness*) и оцењује се функцијом прилагођености (*fitness function*).

Развој ЕА почиње у другој половини прошлог века и то паралелно у неколико научних центара. У Немачкој, еволуционе стратегије (*EC*) развијају *Rechenberg* [101] и *Schwefel* [106] док у САД *Fogel* [40] ради на еволуционом програмирању (ЕП). ГА први уводи *Holland* [56], а четврта врста еволуционих алгоритама - генетско програмирање (ГП) се јавља нешто касније кроз рад *Cramer*-а [24] и *Koze*-а [73].

ЕС су се први пут појавиле у виду скупа правила за експерименталну оптимизацију параметара система код које се постепеном корекцијом параметара, кроз низ узастопних експеримената, систем доводи у оптимално стање. Хидродинамички проблеми, као што су оптимизација облика савијене цеви и проблеми контроле, попут

оптимизације ПИД регулатора били су прве области примене ЕС. Параметри система представљени су вектором чија је свака компонента реалан број. Најједноставнији алгоритам који је коришћен састојао се у примени мутације (мала промена параметара) и селекције (ако нове вредности параметара побољшавају понашање система, онда их треба задржати, а у супротном параметре треба вратити на старе вредности). Пошто је коришћен само један родитељ и један потомак, овај облик ЕС назван је двочлана ЕС, у ознаци $(1+1)-ES$. Са развојем рачунара, настају и остали типови ЕС, са општом ознаком $(\mu/\rho \dot{+} \lambda) - ES$, где је μ број родитеља, а λ број потомака. ρ ознашава број родитеља који учествују у стварању једног потомка; за $\rho = 1$ потомак се добија мутацијом једног родитеља, док се за $\rho > 1$ потомак добија рекомбинацијом родитеља. "$\dot{+}$" означава тип селекције родитеља који се користи. "," означава селекцију код које се родитељи за наредну генерацију бирају сортирањем вредности функције прилагођености само претходно створене деце, док "+" селекција обухвата и ту децу и њихове родитеље.

Генетско програмирање еволуира популацију рачунарских програма за решавање одређеног проблема у циљу проналажења програма који најбоље решава дати проблем. Програми су најчешће представљени као синтаксна стабла. Променљиве и константе програма се у ГП терминологији једним именом називају терминални симболи и представљају листове на стаблу. Операције над њима су унутрашњи чворови стабла, који се називају функције. Алгоритам полази од случајне популације програма. Даље се, све до испуњења услова за прекид алгоритма, над генерацијама наизменично примењују следећи кораци: (1) сваки програм из популације се извршава и додељује му се вредност функције прилагођености на основу његове успешности при решавању задатог проблема; (2) најуспешнији програми из једне генерације, аутоматски постају део наредне генерације, која се допуњава и њиховим потомцима насталим применом операција мутације и укрштања. Најчешће коришћени облик укрштања је укрштање подстабала. За дате родитеље случајно и независно одабира се тачка укрштања, тј. по један чвор сваког дрвета. Потомство се креира тако што се у копији стабла првог родитеља, подстабло чији је корен тачка укрштања замењује подстаблом чији је корен тачка укрштања из копије другог родитеља. Мутација се уобичајено врши тако што се подстабло, са кореном у случајно одабраном чвору замењује новим случајно генерисаним подстаблом.

Еволуционо програмирање такође еволуира рачунарски програм, али за разлику од онога што се дешава у ГП, у еволуционом програмирању програмска структура је фиксирана, док се еволуирају само вредности унутар чворова. Потомство се добија применом оператора мутације. Припадници популације се посматрају као део одређених врста, а не припадници исте врсте, те стога сваки родитељ даје једно дете. ЕП је између осталог примењено на проблеме планирања и рутирања саобраћаја, фармацеутски дизајн, војно планирање. ГА су најчешће употребљавана и највише развијана врста еволуционих алгоритама којима ће бити посвећено посебно поглавље овог рада.

Оптимизација заснована на мрављој колонији

Ова врста метахеуристика заснована је на начину на који мрави у природи проналазе најкраћи пут између извора хране и својих гнезда. Док се креће од гнезда ка храни и обрнуто, мрав на тлу оставља феромоне који постепено испаравају. Остали мрави прате траг феромона док се крећу од гнезда ка храни и обрнуто, али и откривају нове путеве. Неки од нових путева могу бити краћи од претходно пронађеног пута. С обзиром на то да се краћим путем брже стиже, феромонски траг на њему ће бити концентрованији од трагова на свим осталим путевима. Када треба да одлуче којим путем ће кренути, мрави са већом вероватноћом бирају путање означене већим концентрацијама феромона, тако да временом најкраћи пут бива означен најјачим феромонским трагом. У самом алгоритму који се заснива на овој природној појави, скуп вештачких мрава конструише скуп решења крећући се по конструкционом графу, у коме сваки чвор представља компоненту решења, а свака грана има своју концентрацију феромона (као вид дугорочне меморије), као и хеуристичку вредност везану за сам проблем, која најчешће представља процену цене додавања чвора који се налази на крају те гране у тренутно решење. Мрави се по графу не крећу произвољно, већ у складу са пробабилистичким одлукама које се заснивају на концентрацији феромона, цени и ограничењима проблема. Када се конструише једно решење, врши се ажурирање концентрација феромона на пређеној путањи у складу са квалитетом решења које је на том путу конструисано, а нове концентрације користе нови мрави. Након проласка свих мрава у одређеном кораку алгоритма, симулира се испаравање феромона.

Мрави се кроз граф крећу конкурентно и независно једни од других, а индиректно комуницирају остављајући трагове феромона и читајући трагове који су други мрави оставили. На овај начин се врши процес дистрибуираног учења, током кога појединац модификује конструкцију решења проблема на основу свог и туђих искустава.

Теорема *"Нема бесплатног ручка"*

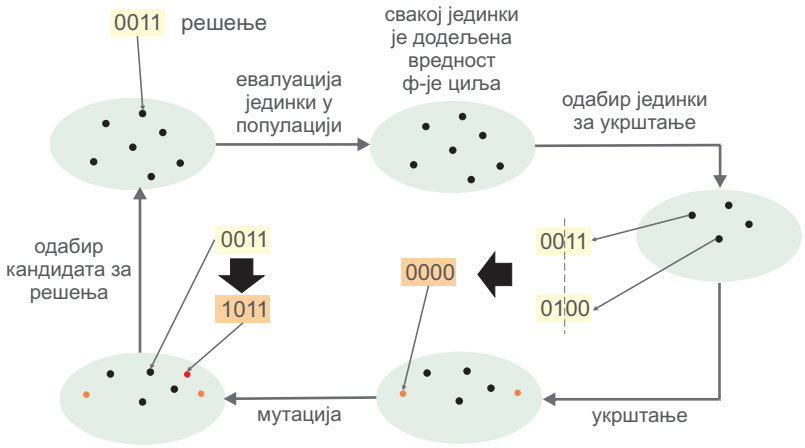
У мноштву апроксимативних метода оптимизације поставља се питање одабира методе која ће показати најбоље перформансе. У раду [119] аутори показују да нема бесплатног ручка кроз доказе *NFL* (*No Free Lunch)* теорема. Ове теореме су серија резултата којима се доказује да ниједан алгоритам над скупом свих могућих проблема није у просеку супериоран у односу на неки други алгоритам, то јест да су перформансе било која два алгоритма еквивалентне када се усредње преко свих могућих проблема. Ако се о решењу неког проблема оптимизације не могу направити никакве претпоставке онда се и не може очекивати да ће једна стратегија имати бољи учинак у његовом решавању него било која друга. Дакле, опште упутство за одабир методе која ће успешно решавати неки оптимизациони проблем, не постоји већ је на истраживачима да за конкретну класу проблема са којом раде, пронађу метод којим ће их најуспешније решавати.

1.3 Генетски алгоритми

Генетски алгоритми су најпопуларнија врста еволуционих алгоритама ([72], [67]). Они су се показали као моћан и робустан механизам када је у питању решавање сложених реалних проблема оптимизације.

У терминологији ГА сваки кандидат за решење проблема у простору претраге назива се јединка, индивидуа или хромозом. Хромозоми су сачињени од дискретних гена. Сваки ген контролише једну или више карактеристика хромозома, док сваки хромозом одговара јединственом решењу у простору претраге. Пресликавање простора претраге у хромозоме се назива кодирање. Суштински, ГА ради над кодираним, а не над оригиналним проблемом. У оригиналној имплементацији ГА од стране *Holland*-а, сваки ген хромозома је једна бинарна цифра, па се овакав тип хромозома назива бинарно кодирани хромозом. У каснијим имплементацијама, уведено је више различитих врста кодирања, па тако, на пример, имамо пермутацијски кодиране хромозоме (сваки хромозом је једна пермутација одређеног скупа бројева) и реално кодиране хромозоме (сваки ген је један реалан број).

ГА ради са колекцијом хромозома, која се зове популација. Уобичајено је да се почетна популација генерише на случајан начин, што доприноси разноврсности генетског материјала. Свака јединка у популацији се евалуира како би се одредио њен степен прилагођености (*fitness*). Евалуација јединке представља израчунавање вредности циљне функције проблема за дату јединку. ГА користи два оператора за генерисање нових решења из постојећих: укрштање и мутацију (слика 1.10).



Слика 1.10: Илустрација тока генетског алгоритма

Оператор укрштања у општем случају ради тако што два хромозома, названа родитељи, комбинује формирајући нове хромозоме, назване потомство. Процес одабира родитеља назива се селекција. Механизам селекције фаворизује натпросечно прилагођене јединке тако да оне добијају већу шансу да створе потомство. Слабије

прилагођене јединке имају смањене шансе за репродукцију, па постепено изумиру. Итеративна примена селекције и укрштања, постепено доводи до све већег броја хромозома са добрим генима у популацији, и приближавања глобалном оптимуму. Оператор мутације уводи насумичне промене у карактеристике хромозома. Мутација се генерално примењује на нивоу гена. У типичним ГА имплементацијама, стопа мутације (вероватноћа промене гена) је веома мала и зависи од дужине хромозома. Дакле, нови хромозом који настаје као производ мутације неће бити много другачији од првобитног. Ипак, мутација игра кључну улогу у ГА. Као што је раније речено, укрштањем популација конвергира тако што хромозоми у популацији постају слични. Мутација враћа генетске разноврсности у популацију и помаже бекство претраге из локалних оптимума. У општем случају ГА се може описати следећим псеудокодом:

```
1 generisi pocetnu populaciju jedinki na slucajan nacin
2 za svaku jedinku u populaciji izvrsi
3 {
4   odredi vrednost funkcije cilja
5 }
6 dok nije zadovoljen uslov_za_zavrsetak_algoritma
7 {
8   dok nije velicina_populacije_potomaka == velicina_populacije
9   {
10    selekcija
11    ukrstanje
12    mutacija
13    dodavanje potomaka u populaciju potomaka
14  }
15  populacija potomaka postaje nova populacija
16  za svaku jedinku u populaciji izvrsi
17  {
18    odredi vrednost funkcije cilja
19  }
20 }
```

Могући услови за заустављање алгоритма су: предефинисани број генерација, предефинисано време извршавања алгоритма, проналажење задовољавајућег решења, или непостојање побољшања квалитета решења током одређеног броја генерација.

Конкретна имплементација ГА подразумева одабир начина селекције родитеља, као и оператора укрштања и мутације.

Три најчешће коришћене методе селекције су турнирска селекција, пропорционална или рулет селекција и селекција заснована на рангу. Код турнирске селекције, два по два решења из популације "учествују у турниру" из кога као победник излази боље прилагођено решење, док лошије решење остаје без могућноости за репродукцију. Рулет селекција подразумева пропорционалност између степена прилагођено-

сти јединке и вероватноће да она буде одабрана за укрштање, тако да јединке са већим степеном прилагођености имају веће шансе да остваре потомство. Код селекције засноване на рангу, индивидуе се прво сортирају у растући поредак према вредности степена прилагођености, тако да индивидуа са најгорим степеном прилагођености има ранг 1, а индивидуа са најбољим степеном прилагођености има ранг N (где је N величина популације). Даље се над ранговима јединки, као над њиховим новим степенима прилагођености, примењује пропорционална селекција.

У литератури се среће мноштво оператора укрштања, али већина њих функционише тако што одабира два родитеља који у извесној мери размењују генетски материјал формирајући потомство. За бинарно кодиране хромозоме често се користи укрштање у једној тачки. Код овог укрштања, прво се из целобројног интервала $[1, l-1]$, где је l дужина хромозома одабере позиција укрштања. Потом се битови иза одабране позиције размене између родитеља и тако произведу две нове јединке. Слично, постоји и k-позиционо укрштање (*k-point crossover*) код кога се одабира k различитих позиција дуж оба родитељска хромозома делећи их тако на $k+1$ подниза битова. Затим, почев од другог подниза, родитељи међусобно размењу сваки други подниз битова. Најчешће коришћена мутација за бинарно кодиране хромозоме је мутација која са одређеном вероватноћом врши промену једног бита (*bit-flip mutation*). За реално кодиране хромозоме постоји читав низ посебних оператора укрштања и мутације [25]. Један од примера је симулирано бинарно укрштање (*SBX crossover*) [27] које симулира принцип укрштања бинарно кодираних хромозома у једној тачки, над реалним хромозомима. Када се на бинарно кодиране хромозоме примени укрштње у једној тачки, декодиране вредности оба добијена потомка леже или унутар или изван интервала ограниченог декодираним вредностима њихових родитеља. Додатно, растојање једног потомка од једног родитеља је потпуно исто као и растојање другог потомка од другог родитеља. SBX укрштање је направљено тако да поседује ово исто својство, али да се примењује на реално кодиране хромозоме.

1.4 Решавање проблема вишекритеријумске оптимизације употребом генетских алгоритама

Посебни генетски алгоритми који се развијају у сврху решавања проблема ВКО називају се једним именом *вишекритеријумски генетски алгоритми - ВКГА* (*Multiobjective Genetic Algorithm - MOGA*). Они настају прилагођавањем класичног ГА за решавање проблема ВКО коришћењем посебних метода за одређивање степена прилагођености јединки и одржавање разноликости решења. Оно што ВКГА чини погодним за решавање проблема ВКО јесте чињеница да он ради са читавом популацијом могућих решења, тако да током једног извршавања може да пронађе читав скуп Парето оптималних решења. Способност ВКГА да истовремено претражује различите регионе простора решења омогућава проналажење широког скупа решења

и за тешке проблеме са неконвексним, испрекиданим и мултимодалним допустивим простором. Додатно, већина ВКГА не захтева од корисника да рангира критеријуме оптимизације.

Током претходних година развијени су бројни ВКГА, и написани многи прегледни радови о овим алгоритмима ([124], [72], [123]). Поред широко прихваћених ВКГА, постоји велики број алгоритама које су аутори креирали комбиновањем и адаптирањем стратегија из више различитих ВКГА и то у сврху решавања конкретних проблема ВКО са којима су се сусретали.

Приликом развоја ВКГА јављају се два кључна проблема [125]:

1. Како одабрати евалуацију прилагођености и селекцију јединки, којима би се претрага решења усмеравала према Парето-оптималном скупу.
2. Како одржавати разнолику популацију да би се спречила превремена конвергенција и тако постигло широко простирање недоминираног скупа решења као и равномерна распоређеност решења унутар скупа.

У наставку овог дела рада ће бити дат преглед уобичајених техника којима се решавају претходно наведени проблеми.

1.4.1 Стратегије евалуације јединки у вишекритеријумским генетским алгоритмима

Код једнокритеријумске оптимизације, одређивање степена прилагођености јединке идентично је израчунавању вредности функције циља за ту јединку. Насупрот томе, код ВКО, резултат евалуације јединке мора да одражава њен степен прилагођености у односу на све критеријуме оптимизције. Генерално се код ВКГА могу разликовати агрегацијска, критеријумска и стратегија евалуације базирана на Парето рангу.

Агрегацијска стартегија

Класични приступ у решавању проблема ВКО јесте додељивање тежина $w_1, w_2, ..., w_K$ свакој од K нормализованих цильних функција, у ознаци $z_i^{'}, i = 1, ..., K$, тако да се проблем сведе на једнокритеријумску оптимизацију са цильном функцијом:

$$\min z = w_1 z_1^{'}(x) + w_2 z_2^{'}(x) + \cdots + w_K z_K^{'}(x) \tag{1.4}$$

Решавањем проблема чија је цильна функција 1.4 за дати тежински вектор $\mathbf{w} = (w_1, w_2, ..., w_K)$ добија се само једно решење. Да би се добило више алтернативних решења потребно је поновити овај процес више пута за различите тежинске векторе. Главна потешкоћа овог приступа је вишеструки одабир тежинског вектора. Ипак, у

раној фази развоја ВКГА, неки аутори су користили тежинске суме и проналазили начине да кроз једно извршавање алгоритма одреде више алтернативних решења. Пример алгоритма који користи тежинске суме, а обезбеђује више алтернативних решења кроз једно извршавање је *RWGA* (*Random Weights GA*) [94], код кога се у фази евалуације решења прво за свако решење случајно генерише нормализовани тежински вектор, а потом се на основу њега израчунава вредност функције циља за дато решење. Главна предност коришћења тежинских сума је једноставна имплементација, с обзиром на то да се уз мање модификације за решавање проблема ВКО користи једноставни ГА. Међутим, овај приступ у случају неконвексног Парето фронта не обезбеђује униформно распоређена решења дуж фронта [124].

Критеријумска стратегија

Критеријумске методе подразумевају да сваки пут када се врши селекција за упаривање, потенцијално различита функција циља одлучује која ће индивидуа бити селектована. Најпознатији алгоритам који користи овакву стратегију евалуације јединки је *VEGA* (*Vector Evaluated GA*) алгоритам кога је увео *David Schaffer* 1985. године [105]. *VEGA* сваку популацију дели на K (онолико колико има циљних функција) подпопулација тако да је величина сваке подпопулације $N_i = N/K$, где је N величина популације. Решења у оквиру i-те подпопулације евалуирају се тако што им се додељује вредност i-те функције циља. Потом се у свакој подпопулацији на основу вредности добијених евалуацијом врши пропорционална селекција најбољих решења. Тако селектоване подпопулације се даље комбинују у једну популацију где се, на исти начин као код једноставног ГА, обављају укрштње и мутација. Иако је предност *VEGA* алгоритма у његовој изузетној једноставности имплементације, његова велика мана огледа се у конвергенцији ка решењима која су екстреми једне од функција циља, а која произилази из начина евалуације решења.

Парето рангирање

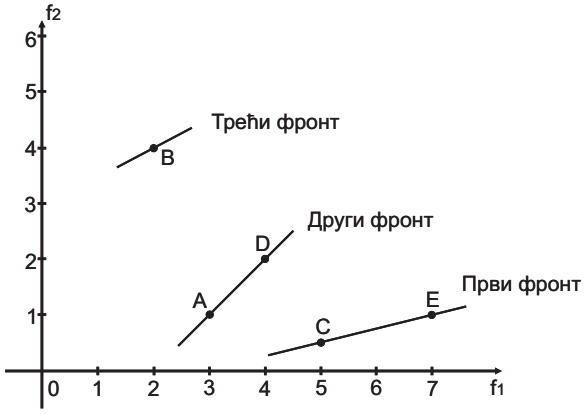
Парето рангирање подразумева коришћење релације доминације за одређивање степена прилагођености индивидуа. Популација се рангира – дели на фронтове на основу релације доминације и сваком решењу се вредност степена прилагођености додељује на основу његовог ранга у популацији. Како се претпоставља минимизација циљних функција, то нижи ранг (мањи број јединки које доминирају посматрану јединку) одговара бољем решењу. За разлику од агрегационих техника, које рачунају степен прилагођености појединца, независно од осталих индивидуа овде се прилагођеност посматра у односу на целу популацију.

Идеју одређивања степена прилагођености јединке на основу Парето ранга уводи Голдберг [49] и то на следећи начин:

```
1 Postaviti brojac i = 1 i formirati pomocnu populaciju TP = P
```

2 Odrediti nedominirana resenja u TP i premestiti ih u skup F_i
3 Ako je $TP = \emptyset$ predji na korak 4. U suprotnom uvecati brojac $i = i + 1$ i preci na korak 2.
4 Svakom resenju $\mathbf{x} \in P$ u generaciji t dodeliti rang $r(\mathbf{x}, t) = i$ ako $\mathbf{x} \in F_i$.

У претходно наведеној процедури $F_1, F_2, ...$ се називају недоминирани фронтови, док је F_1 Парето фронт.



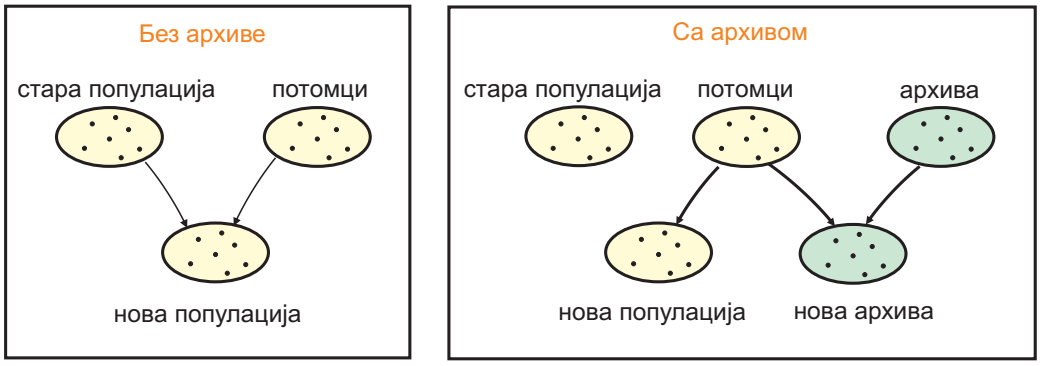
Слика 1.11: Пример пет решења у простору циљних функција и одговарајућих фронтова доминације

На слици 1.11 решења су подељена у три фронта на основу међусобних релација доминације. Решења C и D су најбоља и зато имају ранг 1, тј. припадају првом фронту. Она доминирају сва остала решења, али се не доминирају међусобно, јер је решење C боље по функцији f_2 док је решење D боље по функцији f_1. Решења A и D имају ранг 2, јер су доминирана решењима из првог фронта, сама доминирају решење B, али се не доминирају међусобно. Решење B је доминирано свим осталим решењенима и припада трећем фронту.

Идеју коју је изнео Голдберг преузимали су касније уз одређене модификације многи истаживачи, што је резултирало формирањем неколико различитих приступа одређивања степена прилагођености јединки на основу Парето ранга. Неки приступи [41] користе број јединки којима је нека јединка доминирана да би одредили њен степен прилагођености. *NSGA-II* алгоритам [28] користи Голбергове фронтове доминације, али их одређује коришћењем посебног алгоритма за брзо сортирање популације по недоминирним фронтовима чији је детаљан опис дат у додатку А.1. *SPEA* [127] и *SPEA2* [126] алгоритми додељују степен прилагођености јединки коришћењем броја јединки које јединка која се оцењује доминира.

1.4.2 Употреба елитизма у вишекритеријумским генетским алгоритмима

Елитизам представља копирање одређеног броја најбољих јединки из тренутне у наредну генерацију чиме се решава проблем губљења добрих решења услед особине оператора селекције да насумично одабира решења за укрштање. Код једнокритеријумске оптимизације елитизам се остварује тако што се најбоље решење нађено током процеса оптимизације увек преноси у наредну генерацију. Једноставна примена оваквог елитизма код ВКО није могућа јер уместо једног најбољег решења имамо скуп елитних Парето оптималних решења, који често може да буде готово исте величине као и цела популација. Рани ВКГА нису примењивали елитизам, међутим резултати истраживања [127], [25] указују да ВКГА који користе елитистичке стратегије имају тенденцију да надмаше своје неелитистичке пандане.

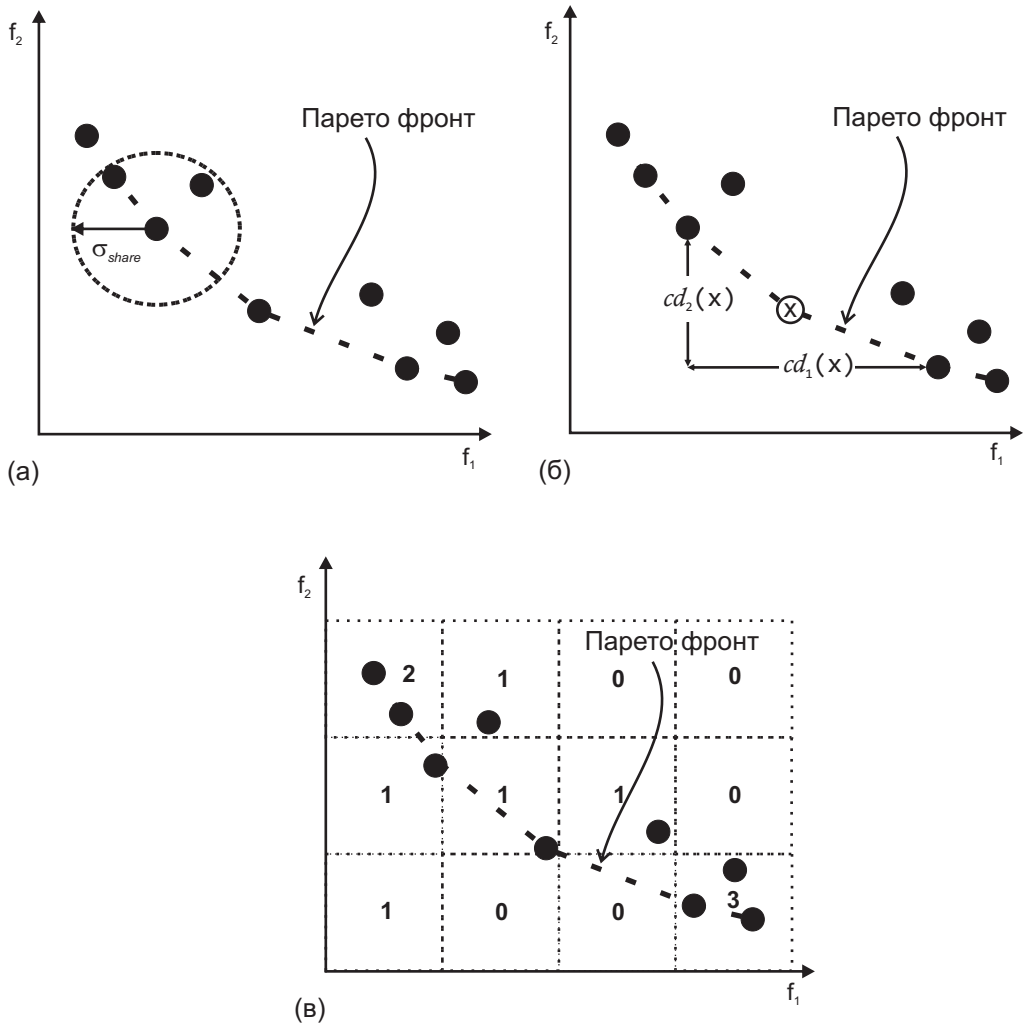


Слика 1.12: Два могућа начина за примену елитизма

Један начин примене елитизма у ВКГА је комбиновање старе популације и потомака и примена детерминистичких процедура за одабир индивидуа за нову популацију (слика 1.12 а). Пример алгоритма са овом врстом елитизма је *NSGA-II* (детаљан опис алгоритма дат је у додатку А). Други начин је да постоји екстерна архива добрих решења, у коју би се копирала обећавајућа решења из сваке генерације (слика 1.12 б). Услед ограничених меморијских ресурса неопходно је из генерације у генерацију редуковати број решења у архиви; на пример задржавати само она која нису доминирана решењима актуелног Парето скупа. Међутим, критеријум доминације често није довољан (на пример, код континуалних проблема Парето фронт може садржати бесконачно много тачака). Како би се смањио број сачуваних решења, у обзир се узимају и додатне информације, као што је густина области којој индивидуа припада, или време протекло од када је индивидуа смештена у архиву. Примери алгоритама са екстерном архивом елитних решења су *RWGA*, *SPEA*, *SPEA2*, *MOCell* [96] *MOEO* [17].

1.4.3 Одржавање разноликости решења у вишекритеријумским генетским алгоритмима

Одржавање разнолике популације је важан фактор у ВКГА, јер обезбеђује добијање равномерно распоређених решења широм Парето фронта. Без предузимања мера за одржавање разноликости решења, популација тежи кластеризацији. Приликом процеса селекције, већина ВКГА одржава разноликост решења коришћењем информација о густини суседства индивидуе. Што је суседство неке индивидуе гушће насељено, то су мање њене шансе да буде изабрана у групу јединки за упаривање.



Слика 1.13: Технике одржавања разноликости решења

Подела вредности степена прилагођености

Једна од техника за одржавање разноликости решења је подела вредности степена прилагођености (*fitness sharing*). Ова техника се заснива на природној појави

да јединке које су концентрисане у једном делу популације, то јест једној ниши, морају да деле доступне ресурсе. Зато се приликом евалуације, степен прилагођености јединки деградира пропорционално њиховој концентрацији у једној ниши. Ширина нише се одређује параметром σ_{share} (слика 1.13а). За сваку индивидуу се рачунају њена растојања од свих осталих индивидуа, која се даље користе за одређивање бројности нише којој посматрана индивидуа припада. Подељени степен прилагођености (*shared fitness*) јединке рачуна се као количник степена прилагођености те јединке и бројности нише. Суседна решења међусобно доприносе бројности својих ниша, тако да ће решења из "густо насељених" области имати ниже вредности подељеног степена прилагођености. Мане овог приступа су обавеза корисника да при сваком покретању алгоритма мора да одабере вредност за σ_{share} параметар, као и повећана дужина извршавања алгоритма због израчунавања ширине нише. Примере коришћења поделе вредности степена прилагођености налазимо у *MOGA* [41], *NSGA* [110] и *NPGA* [57] алгоритму. *SPEA2* уместо бројности нише употребљава меру густине суседства неке индивидуе, при чему је радијус суседства једнак растојању од посматране индивидуе до њеног k-тог најближег суседа. Густина суседства се израчунава као реципрочна вредност овог радијуса, и користи се тако што се између два решења истог ранга за репродукцију бира оно које има мању густину суседства. Предност коришћења k најближег суседства над нишом је у једноставнијем одабиру параметра k у односу на параметар σ_{share}.

Растојање до нагомилавања (*Crowding distance*)

Овај метод очувања разноликости решења подразумева да решења која се налазе у деловима популације где је присутна велика густина јединки имају мање шансе за репродукцију. За утврђивање густине популације око одређене тачке користи се мера растојања до нагомилавања. Растојање до нагомилавања је половина обима хиперкоцке која се може описати око посматране јединке тако да се унутар ње не налази ниједна друга јединка која је истог ранга као и посматрана јединка (слика 1.13 б). Што је веће растојање неке јединке до нагомилавања, то су њене шансе за репродукцију веће.

Предност овог приступа је да не захтева коришћење кориснички дефинисаних параметара. Први пут је представљен у *NSGA-II* алгоритму са циљем да се избегне коришћење σ_{share} параметра, а користи се и у *MOCell* алгоритму. *MOEO* алгоритам користи меру нагомилавања да би одлучио да ли ће новопронађено недоминирано решење бити додато у архиву елитних решења. Ако би се ново решење додавањем у архиву сместило у регион у коме је нагомилавање највеће онда се ово решење одбацује, док се у супротном оно додаје у архиву.

Густина ћелија

Приступ базиран на густини ћелија подразумева поделу простора циљних функција на К-димензионе ћелије (слика 1.13в). Број јединки које се налазе у свакој ћелији дефинише густину ћелије, али и густину сваке јединке која тој ћелији припада. Индивидуе које припадају гушће насељеним ћелијама имају мање шансе да буду одабране за репродукцију. Овај принцип се на различите начине реализује у конкретним алгоритмима који примењују одржавање разноликости популације базирано на густини ћелија, као што су *PESA* [22], *PESA-II* [23], *RDGA* [122] и други. Основна предност ћелијског приступа је што се рачунањем густина појединих ћелија добија читава мапа густине насељености циљног простора. Тако се претрага за решењима може усмерити ка слабије попуњеним областима ове мапе чиме се обезбеђује равномерна распоређеност решења по читавом циљном простору. Предност овог приступа у односу на приступ са нишама је већа ефикасност.

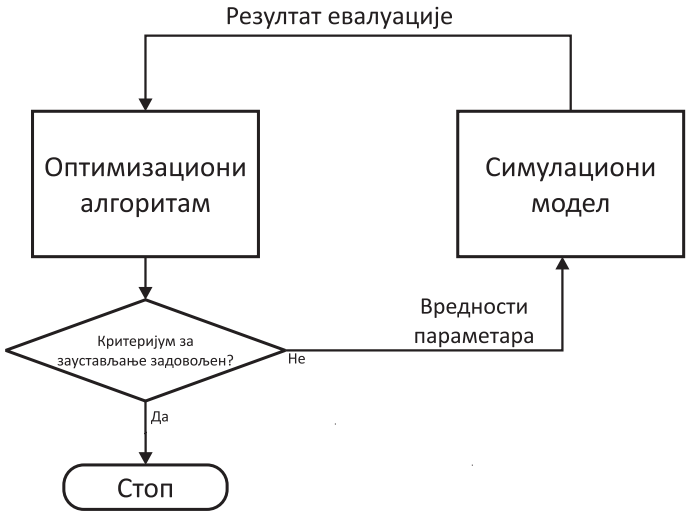
1.5 Примена вишекритеријумске оптимизације у хидроинформатици

Хидроинформатика [2], [1], [99] је интердисциплинарна област која повезује проблеме водних ресурса и животне средине са различитим методама рачунарског моделирања и информационо-комуникационих технологија. Интегрисаном употребом симулационих модела, система за подршку одлучивању који се заснивају на вештачкој интелигенцији и савремених техника оптимизације, револуционарно је измењена традиционална методологија у планирању рационалне употребе водних ресурса као и у њиховом управљању. Хидроинформатика додатно обухвата социјални аспект проблема управљања водама и као императив поставља очување еколошке равнотеже у природи.

Управљање водним ресурсима и планирање њихове експлоатације има за циљ максимизовање економске и социолошке добити, уз очување природних процеса. Постизање ових циљева без нарушавања ограничења природног окружења чини избор управљачких активности и планирање коришћења водних ресурса процесом вишекритеријумске оптимизације. Исход овог процеса је проналажење оних решења која су еколошки коректна, економски сигурна, технички остварива и социолошки прихватљива [31]. У циљу проналажења оваквих решења врши се моделовање комплексних водних система и симулација њиховог понашања при променама вредности или функционалних зависности међу параметрима система. Уз симулационе моделе као срж хидроинформатике, важну улогу, као комплемент моделирању игра оптимизација. Поред аспекта максимизовања зараде уз минималне трошкове који је одувек присутан, друштвено и еколошки одговорно понашање налаже укључивање низа других циљева и ограничења у процес оптимизације у области водних ресурса. Код проблема управљања акумулацијом, на пример, не може се разматрати само

максимизовање прихода од произведене енергије, већ се у обзир узима и спречавање поплава, контрола степена ерозије речног корита, као и одржање природног режима тока реке, ради очувања еколошке стабилности. У складу са овим захтевима, коришћење техника једнокритеријумске оптимизације у области хидроинформатике је неприхватљиво и као што се може видети из бројних радова наведених у делу 3.1, ВКГА је широко прихваћени стандард за решавање оптимизационих проблема у овој области. Неки од примера проблема ВКО у области хидроинформатике су [63]: одређивање протока воде кроз хидроелектрану тако да се максимизује годишња производња електричне енергије, али и задовоље потребе потрошача воде који се налазе низводно од хидроелектране [68], [75], [87]; одређивање плана рехабилитације дренажног система како би се смањиле последице поплава, уз задовољење ограничења буџета [9]; одређивање оптималне стратегије санације подземних вода ради минимизовања концентрације загађивача за минимални временски период [81] и други.

Проблеми хидроинформатике, као и други проблеми из реалног окружења, су често веома сложени и са високим степеном нелинеарности што онемогућава њихово ефикасно аналитичко моделовање. У том случају се употребљава симулациони модел система. Код симулационих модела понашање система се посматра кроз промене стања система и излаза у зависности од улаза у систем. Процес проналажења најбољих вредности параметара система, током кога се квалитет параметара оцењује на бази квалитета резултата симулираног модела система, назива се оптимизација заснована на симулацији.



Слика 1.14: Шематски приказ тока оптимизације базиране на симулацији

Оптимизација базирана на симулацији која се извршава применом ГА, тече на следећи начин (слика 1.14): (i) алгоритам генерише популацију скупова вредности улазних параметара (једна јединка одговара једном скупу вредности, а један

ген одговара вредности једног параметра); (ii) параметри се додељују симулационом моделу; (iii) симулацијом модела се мере перформансе система, на основу чега се формира оцена квалитета сваког скупа параметара (тј. сваке јединке); (iv) оцене јединки се враћају назад алгоритму који даље примењује селекцију, укрштање и мутацију, формирајући нову генерацију потомака. Кораци (ii), (iii), и (iv) се понављају све док се не испуни критеријум за заустављање алгоритма. Код оваквог приступа, симулациони модел је заправо евалуатор јединки у форми црне кутије, с обзиром на то да су релације између улазних и излазних вредности невидљиве из позиције оптимизационог алгоритма. То за последицу има потпуну сепарацију модела система и алгоритма који се користи за решавање проблема његове оптимизације. Предност ове раздвојености је да модел система може независно да се мења и усавршава, и/или да се оптимизација може извршавати коришћењем различитих алгоритама над истим моделом.

Калибрација модела реалних система је чест проблем у науци и инжењерству. Калибрација модела представља одређивање оптималних параметара модела, тако да се симулацијом модела са овим параметрима добијају излази који се минимално разликују од осмотрених вредности битних величина. У хидроинформатици, на пример, хидролошки модели представљају окосницу бројих истраживања, те је њихова успешна калибрација од суштинске важности. Сложени хидролошки, и други модели садрже велики број параметара, чије је међусобне односе и ефекте промена њихових вредности на читав систем тешко сагледати. Ручно подешавање вредности више параметара модела је практично немогуће. Отуда се за калибрацију модела користи оптимизација заснована на симулацији. Када се проналажење оптималних параметара модела врши коришћењем ГА, гени једне јединке представљају тражене параметре модела. Приликом евалуације једне јединке извршава се симулација модела са параметрима одређеним том јединком. Добијени излази битних величина се упоређују са осмотреним вредностима, тако да је степен прилагођености јединке већи што је мања разлика добијених излаза и њихових осмотрених вредности.

Применом оптимизације базиране на симулацији помоћу ГА успешно се аутоматизује процес оптимизације и/или калибрације модела реалних система. Међутим, приликом оцењивања сваке јединке извршава се симулација модела која може трајати и више минута. Самим тим, оцењивање више стотина јединки у популацији, кроз десетине генерација трајало би месецима. Како би се оптимизациони процес модела завршио у прихватљивом року, користе се паралелни ГА.

Глава 2

Паралелизација генетских алгоритама

Током извршавања генетских алгоритма, у свакој генерацији се врши евалуација свих индивидуа спрам проблема који се решава. У случају оптимизације сложених система, свака евалуација може потрајати и по више десетина минута у зависности од конфигурације рачунара на коме се она извршава. Као што је раније наведено, код оптимизације базиране на симулацији, дуготрајне евалуације су неминовност, јер се оцена сваке јединке формира на основу вредности величина добијених симулацијом модела одговарајућег система. Имајући у виду да је током извршавања алгоритма потребно евалуирати више стотина јединки у неколико стотина генерација, произилази да једна оптимизација на једном процесору може трајати месецима, што је неприхватљиво за било какву практичну примену.

У циљу скраћења времена извршавања генетских алгоритама, врши се њихова паралелизација [92], [6], [5]. ГА су слика природног процеса еволуције који се одвија паралелно, и као такви су погодни за конкурентно извршавање. Основна идеја паралелизације неког алгоритма је подела целокупног посла на подзадатке који се могу изводити паралелно на више процесора, са циљем да се извршавање алгоритма убрза, а при томе не наруши квалитет резултата. У складу са овом идејом предложено је више модела паралелизације ГА, који ће бити приказани у другом делу овог поглавља. С обзиром на то да одабир модела паралелизације ГА у великој мери зависи од архитектуре на којој ће се програмска имплементација алгоритма извршавати, у првом делу поглавља ће бити дат преглед паралелних и дистрибуираних рачунарских архитектура и одговарајућих паралелних програмских модела.

2.1 Преглед паралелних и дистрибуираних рачунарских архитектура

Постоји више начина категоризације паралелних архитектура, али је као полазна тачка општеприхваћена тзв. Флинова таксономија [39]. Флин поделу паралелних архитектура врши уз помоћ појма инструкције и појма података. Ако се сваки процес у рачунарском систему посматра као извршавање низа инструкција којима се манипулише над подацима, онда се паралелизам може остварити на нивоу инструкција и/или на нивоу података. У зависности од нивоа на коме се паралелизам остварује, паралелне архитектуре се по Флиновој таксономији могу разврстати у четири категорије: (слика 2.1):

1. **SISD** (*Single Instruction Single Data*) - "једна инструкција, један податак" је категорија серијских рачунара код којих CPU извршава само један ток инструкција над једним током података. Пример овакве архитектуре је класични једнопроцесорски PC.

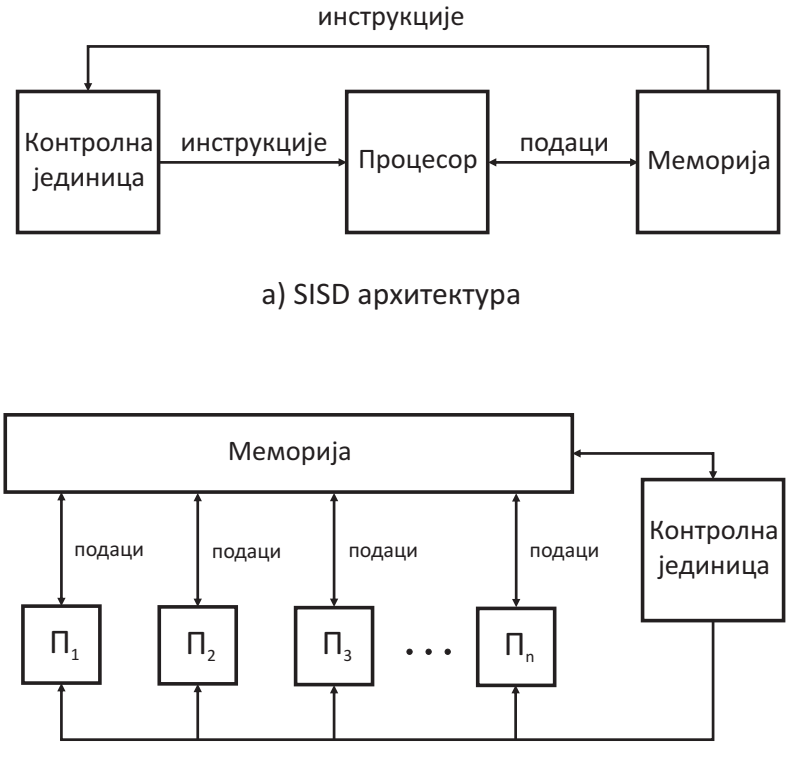
2. **SIMD** (*Single Instruction Multiple Data*) - "једна инструкција, више података" је тип паралелног рачунара код кога све процесорске јединице синхронизовано извршавају исту инструкцију, али над различитим скупом података. Пример овакве архитектуре су графички процесори опште намене (GPGPU - *General Purpose Graphic Processing Units*).

3. **MISD** (*Multiple Instruction Single Data*) - "више инструкција, један податак" представља веома редак тип паралелне архитектуре која се може апстраховати као низ процесорских јединица које независно оперишу над једним током података, прослеђујући резултате од једне јединице ка наредној.

4. **MIMD** (*Multiple Instruction Multiple Data*) - "више инструкција, више података" је данас најзаступљенији тип паралелне архитектуре, која се састоји од више процесорских елемената који синхроно или асинхроно извршавају сопствени ток инструкција над сопственим током података. Суперачунари, рачунарски кластери, рачунарски грид, па чак и вишејезгарни процесори унутар савремених рачунара опште намене спадају у ову врсту паралелне архитектуре.

Поред Флинове категоризације, користи се и подела паралелних рачунара заснована на меморијској архитектури. Актуелне су три врсте меморијске архитектуре:

1. **Архитектура дељене меморије** - заједничка особина оваквих система је да сви процесори у систему приступају целокупној меморији као глобалном адресном простору. Промене у некој меморијској локацији учињене од стране једног процесора, аутоматски су видљиве другим процесорима у систему. Архитектуре дељене меморије се даље деле на:



Слика 2.1: Флинова таксономија архитектура паралелних рачунара

(а) **Архитектуре са униформним меморијским приступом** (*UMA - Uniform Memory Access*) које подразумевају идентичне процесоре и идентично време приступа меморији и стога се називају још и симетрични мултипроцесори (*SMP - Symmetric MultiProcessor*). Пример симетричних мултипроцесорских система дељене меморије је највећи број данашњих вишејезгарних система.

(b) **Архитектуре са неуниформним меморијским приступом** (NUMA - Non-Uniform Memory Access) које подразумевају физичко повезивање више *SMP* јединица, при чему сви процесори немају апсолутно исто време приступа свим меморијама, јер је меморијски приступ преко веза између *SMP* јединица нешто спорији од приступа меморији у оквиру једне *SMP* јединице.

Предности архитектуре дељене меморије су с једне стране једноставност приступа меморијским ресурсима за програмера, као и брзо и униформно дељење података међу процесима. Највећа мана архитектуре је недостатак скалабилности између меморије и процесора, јер додавање процесора систему повећава саобраћај на магистали геометријском прогресијом. Такође се маном може сматрати и одговорност програмера да извршава синхронизације како би приступ глобалној меморији остао кохерентан.

2. **Архитектура дистрибуиране меморије** - заједничка особина система заснованих на овој архитектури је захтев за постојањем комуникационе мреже како би се омогућио приступ интерпроцесорској меморији. Сваки процесор има своју локалну меморију, а ако једном од процесора затреба податак из адресног простора другог процесора, задатак је програмера да ту комуникацију изведе. Пример система дистрибуиране меморије су рачунарски кластери. Предности ове архитектуре су скалабилност меморије у односу на број процесора (пропорционално расту), бржи приступ процесора својој сопственој меморији и исплативост услед коришћења стандардних процесора и мрежа. Највеће мане су апсолутна одговорност програмера за све детаље међупроцесне комуникације, *NUMA* време приступа и релативна тежина транзиције програма заснованих на глобалној меморији на дистрибуирану.

3. **Хибридна архитектура** - подразумева здружену употребу архитектура дељене и дистрибуиране меморије. Компонента дељене меморије је *SMP* систем, док је компонента дистрибуиране меморије мрежа између више таквих *SMP* машина. Највећи и најјачи рачунари данашњице изграђени су на овој архитектури, чија се апсолутна доминација очекује и у даљој будућности.

У складу са хардверском меморијском архитектуром система на коме дати софтвер треба да се извршава, бира се и програмски модел за развој. Тако се, аналогно хардверској класификацији може извршити и класификација паралелних програмских модела, и то поделом на:

1. **Модел дељене меморије** је архитектура код које паралелни задаци деле заједнички адресни простор, који читају и пишу асинхроно, док се различити механизми попут семафора користе за контролу приступа дељеној меморији. Најпознатији модел дељене меморије је модел нити, а најпознатије стандардне имплементације су *POSIX Threads* (*pthreads*) и *OpenMP* (*Open MultiProcessing*).

2. **Модел порука** се користи на архитектурама дељене меморије и подрзумева скуп задатака који користе своју сопствену меморију приликом извршавања, а међусобно комуницирају искључиво размењујући поруке. Више таскова може бити лоцирано на физички истој машини или више различитих машина. Стандардизација библиотека за модел порука извршена је кроз *MPI* (*Message Passing Interface*) стандард.

3. **Паралелизам података** - Највећи део паралелног посла извршава се извођењем операција над подацима који су обично организовани у једнодимензоне или вишедимензионе низове, при чему сваки процес оперише над својом порцијом података. Најпознатије имплементације су *HPF* (*High Performance Fortran*) и у новије време, разне *GPGPU* платформе, пупут *CUDA* (*Compute Unified Device Architecture*) и *OpenCL* (*Open Computing Language*).

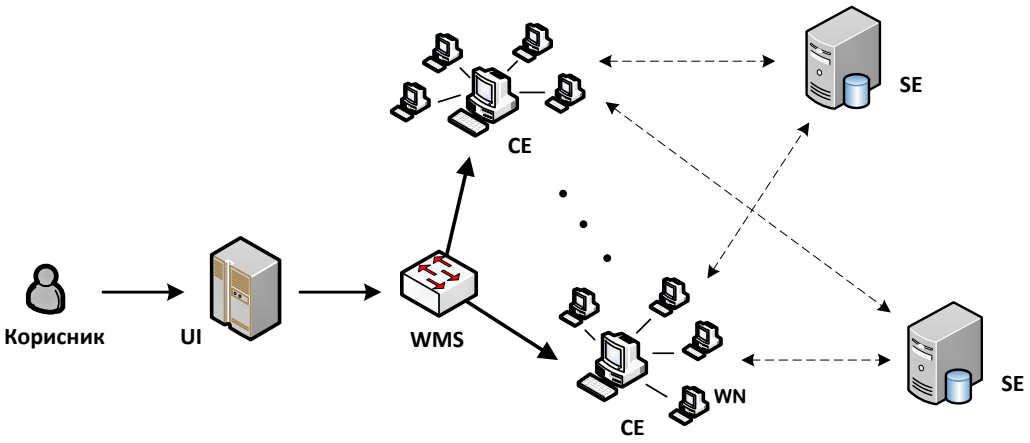
2.1.1 Грид рачунарство

Већина паралелних ГА имплементира се на кластер или Грид инфраструктури. С обзиром на то да тежиште ове дисертације представља развој софтверског оквира који врши еластично резервисање рачунарских ресурса Грида за потребе ВКО засноване на ГА, у овом делу рада ће бити дато више информација о самом Грид рачунарству. Док се кластери састоје од скупа повезаних, колоцираних, хомогених рачунара и често су потпуно на располагању истраживачима који их користе за паралелизацију својих апликација, Грид представља географски дистрибирану инфраструктуру која обједињује нехомогене рачунарске ресурсе широм света и чини их доступним својим корисницима на транспарентан начин. Појава Грид рачунарства [44] је пре свега научној заједници пружила је приступ огромним рачунарским ресурсима, обезбедила ефикасно извршавање паралелних апликација које захтевају велику рачунарску снагу и тако омогућила корисницима да боље одговоре на изазове модерне науке и инжењерства.

Грид технологије подржавају дељење и координисану употребу различитих ресурса кроз динамичке виртуелне организације, то јест креацију таквих виртуелних рачунарских система од географски и организационо дистрибуираних компоненти, који су довољно интегрисани да пруже жељени квалитет услуга (*quality of service - QoS*) [45]. Дељење ресурса у Грид рачунарству подразумева директан приступ рачунарима, софтверу, подацима и другим ресурсима за потребе науке, инжењерства и индустрије. Зато оно мора бити строго контролисано, и провајдери ресурса морају дефинисати шта се тачно дели и под којим условима. Скуп појединаца и институција који деле и приступају дељеним ресурима чини виртуелну организацију (*virtual organization - VO*). Интероперабилност на Грид инфраструктури обезбеђује се коришћењем Грид мидлвера који корисницима пружају стандардизоване сервисе који се тичу обезбеђивања сигурности, управљања подацима, пружања информација, подношења, надгледања и извршавања корисничких послова и другог. Два најчешће

коришћена Грид мидлвера су *EMI/UMD* [111] и *Globus* [43]. С обзиром на то да су сва Грид тестирања развијених софтверских оквира (која су изложена у деловима 6, 7) извршена коришћењем *EMI/UMD* мидлвера на Југоисточној Европској регионалној Грид инфраструктури [8], [38], то ће се у даљем тексту користити терминологија везана за *EMI/UMD* мидлвер.

Корисници Грида морају припадати некој од виртуелних организација и поседовати валидни Грид сертификат како би приступили Гриду. Сертификате издаје признати орган за сертификацију (*Certification Authority, CA*). Кориснички сертификат је заштићен лозинком и служи за генерисање привременог прокси сертификата који заправо обезбеђује аутентификацију потребну за приступ Грид сервисима. Приступна тачка за Грид кориснике је кориснички интерфејс (*user interface, UI*) - било који рачунар на коме постоји њихов кориснички налог и инсталиран кориснички сертификат (слика 2.2). Кориснички интерфејс пружа могућност обављања основних операција на Гриду: предавање послова на извршење, праћење статуса послова, отказивање послова, преузимање излаза извршених послова, копирање, прављење реплика и брисање датотека са Грида и слично. Посао се извршава на радном чвору (*worker node, WN*). Скуп радних чворова који су локализовани на једном месту - Грид сајту, сачињава рачунски елемент (*computing element, CE*).



Слика 2.2: Шематски приказ Грид инфраструктуре

Корисници и апликације своје податке, који могу бити употребљени у извршавању Грид послова, смештају на складишне елементе (*storage element, SE*). У Грид окружењу, датотеке могу имати своје реплике на више различитих сајтова. Реплике су идентичне копије података смештене на различите складишне елементе у циљу оптимизације приступа тим подацима од стране послова покренутих на различитим *CE*-овима. Постојање реплика омогућава пословима да датотеке потребне за свој рад преузму са себи најближег *SE*-а чиме се убрзава њихово извршавање. Реплике такође обезбеђују заштиту од потенцијалних проблема приступа подацима на одређеном *SE*-

у и повећавају робусност дистрибуираних апликација [52]. Свакој датотеци која је похрањена на *SE*-у се може приступити коришћењем неке од следећих адреса: јединствени Грид идентификатор (*Grid Unique IDentifier, GUID*), логичко име датотеке (*Logical File Name, LFN*), складишни *URL* (*Storage URL, SURL*) или транспортни *URL* (*Transport URL, TURL*). *GUID* и *LFN* идентификују фајл јединствено, независно од његове локације, док *SURL* и *TURL* садрже информације о физичкој локацији реплике.

Систем за управљање пословима (*Workload Management System, WMS*) има задатак да прихвата корисничке послове, додељује их одговарајућем *CE*-у, прати њихов статус и враћа њихов излаз. Посао се предаје на извршење и описује употребом посебног језика за опис посла (*Job Description Language, JDL*), којим се специфицира низ информација, као што су назив програма који треба извршити, називи улазних и излазних датотека, као и захтеви који се тичу *CE*-а и *WN*-а на коме ће се посао извршавати.

Грид рачунарство пружа неопходну инфраструктуру за имплементацију паралелних метахеуристика. Међутим, истраживачи се срећу са новим потешкоћама везаним за развој и примену Грид апликација. С обзиром на то да су Грид ресурси дистрибуирани, хетерогени и не могу бити предати на коришћење само једном кориснику, писање паралелних апликација које ће се извршавати на Гриду представља изазов [42] и изискује значајан напор и експертско знање. Разумевање основа Грид рачунарства и Грид мидлвера одузима доста времена и енергије. Додатно оптерећење представља и то што је приликом сваког извршавања неке апликације на Гриду потребно позабавити се питањима одабира Грид ресурса који ће бити коришћени за извршавање апликације, као и припремом, отпремањем, надгледањем и окончавањем Грид послова. Ове активности се могу обављати на различите начине код различитих мидлвера, а разлике међу мидлверима могу ограничити портабилност апликација између различитих Грид инфраструктура. Поред сложености Грид инфраструктура, присутна су још нека ограничења, попут нужног чекања (често значајно дугог) да кориснички захтев за рачунарским ресурсима буде процесиран. Сви ови отежавајући фактори везани за писање и извршавање Грид апликација одвраћају истраживаче од веће употребе Грида у научне сврхе. Отуда развој алата који скривају комплексност Грида од истраживача и олакшавају развој Грид апликација, може да доведе до значајног пораста употребе Грида за научне и инжењерске апликације.

2.2 Преглед модела паралелних генетских алгоритама

Постоје три основна модела паралелних генетских алгоритама (ПГА): глобални, острвски и решеткасти ПГА. Поред ових основних начина паралелизације ГА постоје и хијерархијски модели паралелизације који комбинују основне моделе. У

наредном делу рада ће бити представљен сваки од наведених модела паралелизације.

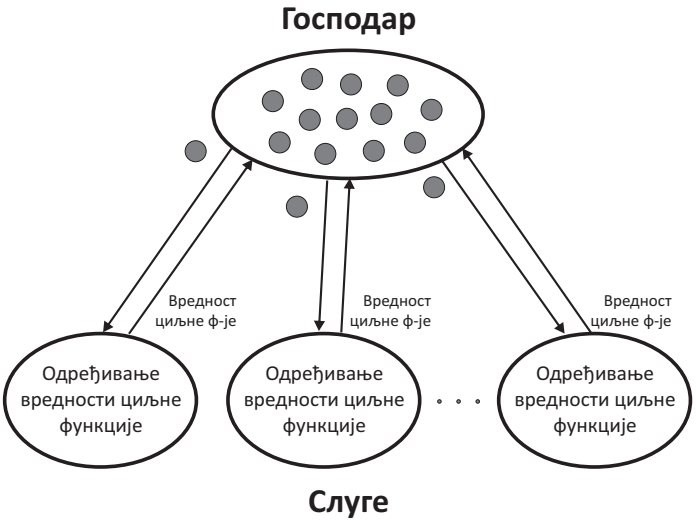
2.2.1 Глобални паралелни генетски алгоритам

Код глобално паралелног (једнопопулацијског) ГА еволуира се једна популација, али се евалуација јединки дистрибуира на више процесора. С обзиром на то да је евалуирање сваке индивидуе потпуно независан процес, овај вид паралелизације не изискује мењање ГА, те је отуда најједноставнији. Назив глобални паралелни ГА долази отуда што селекција и укрштање укључују целокупну популацију. Овај модел паралелизације назива се још и модел господар-слуга (*master-slave*) јер један процесор - господар дистрибуира јединке процесорима слугама, који их евалуирају, док он обавља операције селекције, мутације и укрштања како би произвео нову генерацију јединки. На слици 2.3 дат је графички приказ глобално паралелног ГА.

Комуникација између господара и слуга одвија се у два наврата: када господар слугама шаље јединке на евалуацију и када слуге враћају господару израчунате вредности функције циља. Псеудокод глобално паралелног ГА био би следећи:

```
1 generisi pocetnu populaciju jedinki na slucajan nacin
2 broj_generacija:=1
3 dok nije zadovoljen uslov_za_zavrsetak_algoritma
4 {
5   za svaku jedinku u populaciji izvrsi paralelno
6   {
7     odredi vrednost funkcije cilja
8   }
9   selekcija
10  ukrstanje
11  mutacija
12  broj_generacija:=broj_generacija+1
13 }
```

Глобално паралелни ГА је једноставан за имплементацију и погодан у ситуацијама лимитиране доступности рачунара на којима се обављају евалуације, јер се слуге могу додавати и уклањати динамички, без опасности да дође до губитка информација. У случају отказивања неког процесора слуге, јединка коју је изгубљени процесор евалуирао прослеђује се другом слуги. Још једна предност овог модела паралелизације је да централизовано управљање популацијом олакшава бележење статистичких података током извршавања алгоритма. Међутим, централизованост са собом носи и недостатак да пад процесора господара паралише извршавање читавог алгорима. Високи комуникацијски трошкови услед слања свих јединки на евалуацију и враћања резултата евалуације кроз мрежу, могу надмашити користи паралалелизације. Додатно успорење може бити проузроковано нехомогеношћу про-



Слика 2.3: Шематски приказ глобалног ПГА

цесора слуга и дужим трајањем евалуације на некима од њих, с обзиром на то да господар мора сачекати да све јединке из популације буду евалуиране како би наставио са алгоритмом.

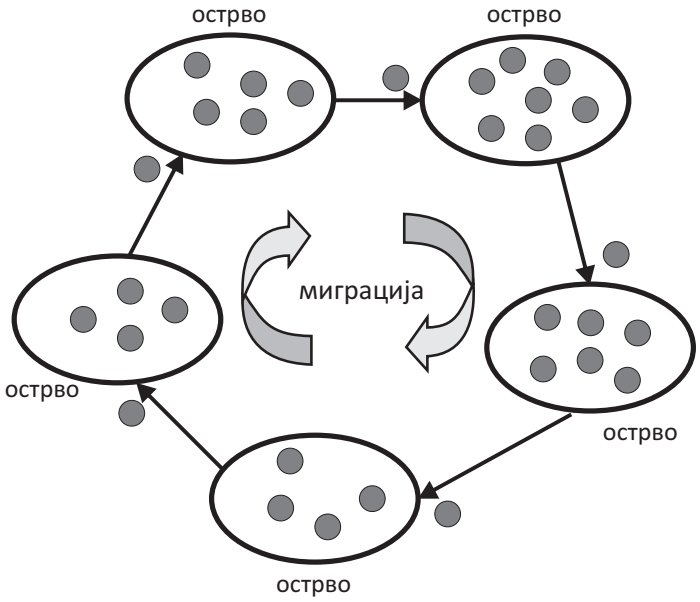
Глобални ПГА се може имплементирати како на вишепроцесорским архитектурама дељене меморије, тако и на архитектурма дистрибуиране меморије. Ипак, како перформансе архитектура дистрибуиране меморије остају релативно непромењене са додавањем процесора, оне представљају бољи избор за имплементацију овог модела. Архитектуре дељене меморије, с друге стране, пате због конкурентног приступа меморији, што нарочито долази до изражаја у случају када процес евалуације подразумева интезивне операције читања и писања у меморију.

2.2.2 Острвски паралелни генетски алгоритам

Острвски (*island*) или крупнозрни (*coarse-grained*) модел паралелизације ГА врши паралелизацију на нивоу популације. Инспирација за ову врсту паралелизације долази из природне појаве да популације имају теденцију ка просторном организовању у *деме*. Деме су полу-независне групе јединки које нису уско везане са суседним демама. Веза између суседних дема се одржава повременим миграцијама неких индивидуа из једне у другу дему.

Острвски модел има раздвојене подпопулације чија је величина најчешће мања од величине популације у серијском ГА, а којих има онолико колико има процесора на располагању. Подпопулације могу да размењују генетски материјал тако што с времена на време неке јединке прелазе (мигрирају) из једне у другу подпопулацију према различитим обрасцима миграције. Главни разлог за овај приступ је да се периодично убаци различитост у подпопулације које би у супротном прерано конвергирале ка ло-

калном оптимуму. Између миграционих фаза, у оквиру сваке подпопулације одвија се стандардни секвенцијални ГА.



Слика 2.4: Шематски приказ острвског ПГА

Миграција и механизам њеног извођења имају велики утицај на учинак острвског ГА. Поступак миграције одређују четири параметара:

1. *Миграцијски интервал* је учесталост размене јединки међу подпопулацијама. То је број итерација стандардног секвенцијалног ГА, који ће бити извршен у свакој подпопулацији између две миграције. Када је миграција пречеста, постоји опасност од преурањене стагнације, јер се губи својство да свака подпопулација претражује одређени део простора решења и цео систем почиње да се понаша као једна велика популација. Када се миграција одвија сувише ретко, разлика између јединки у подпулацијама постаје превелика, те се јединке које мигрирају не могу "уклопити" у нову средину.

2. *Миграцијска стопа* је број јединки које се размењују у поступку миграције. Сувише велика миграцијска стопа доводи до истих негативних ефеката као и сувише честе миграције.

3. *Стратегија одабира јединки* за миграцију и елиминицају одређује које ће јединке из једне подпопулације бити одабране за прелазак у другу подпопулацију, као и које ће јединке бити замењене новопристиглим јединкама. У литератури се могу срести различити приступи, али се најчешће за миграцију бирају најбоље јединке којима бивају замењене најлошије јединке.

4. *Топологија миграције* је шема по којој подпопулације размењују јединке. У употреби су различите топологије од којих су најчешће прстен, 2-Д и 3-Д мрежа,

хипер-коцка и случајни графови. На слици 2.4 је дат шематски приказ острвски дистрибуираног модела паралелизације са прстенастом топологијом миграције.

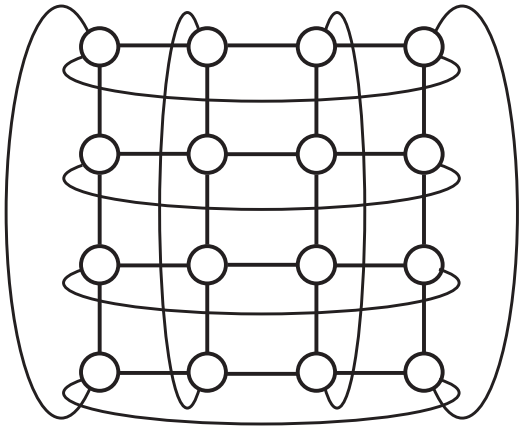
Псеудокод острвског паралелног ГА био би следећи:

```
1 generisi P podpulacija velicine N na slucajan nacin
2 broj_generacija:=1
3 dok nije zadovoljen uslov_za_zavrsetak_algoritma
4 {
5   za svaku podpopulaciju izvrsi paralelno
6   {
7     odredi vrednost funkcije cilja svih jedinki
8
9     ako broj_generacija mod migracijski_interval == 0 onda
10    {
11      posalji migracijska_stopa jedinki u drugu podpopulaciju
12      primi migracijska_stopa jedinki iz druge podpopulacije
13      zameni migracijska_stopa jedinki novopristiglim jedinkama
14    }
15    selekcija
16    ukrstanje
17    mutacija
18  }
19  broj_generacija:=broj_generacija + 1
20 }
```

Предност острвског модела је ретка, асинхрона комуникација која се дешава само у тачно одређеним тренуцима миграције, као и робусност обезбеђена непостојањем централизоване контроле. Обавеза одређивања величине подпопулација и претходно наведених параметара миграције отежава употребу овог типа паралелног ГА. Дистрибуираност резултата на чворове који извршавају ГА над попопулацијама компликује прикупљање статистичких података и њихову анализу. Острвски тип паралелног модела није погодан за извршавање на мрежама хетерогених рачунара код којих су чворови лимитарано доступни јер ако дође до пада неког од чворова, део популације који се на њему налази неће даље еволурати и може бити изгубљен. Острвски модел паралелизације се најчешће имплементира на кластерима.

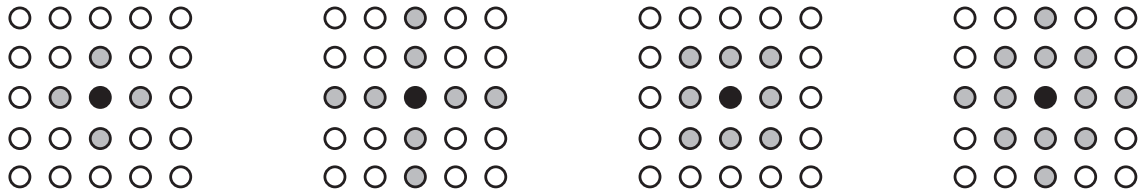
2.2.3 Решеткасти паралелни генетски алгоритам

Код решеткастог или, како се још назива, ситнозрног (*fine-grained*) модела ПГА, јединке су распоређене у тороидну једнодимензионалну или дводимензионалну решеткасту структуру тако да се у сваком чвору решетке налази по једна јединка (слика 2.5). Чворови решетке су заправо процесори, те је ова врста паралелног модела донекле слична острвском моделу са подпопулацијама које садрже само по једну јединку. Величина популације одређена је бројем процесора који су на располагању.



Слика 2.5: Дводимензионална решеткаста структура ПГА

Евалуација се врши паралелно за све индивидуе, док се селекција и укрштање обављају локално у оквиру уског суседства. Суседство може бити дефинисано на разне начине (слика 2.6), при чему се суседства међусобно делимично преклапају. Делимична преклољеност суседстава доводи до лагане дифузије добрих решења кроз мрежу чиме се квалитетан генетски материјал постепеније шири по популацији него код осталих типова ПГА. Топологија и величина суседства утиче на брзину и начин пропагације квалитетних јединки кроз популацију. Превелика суседства доводе до пребрзе конвергенције ка локалном оптимуму.



Слика 2.6: Приказ могућих суседства јединке. Јединка је обележена црном бојом, док је њено суседство обележено сивом бојом

Псеудокод решеткастог паралелног ГА био би следећи:

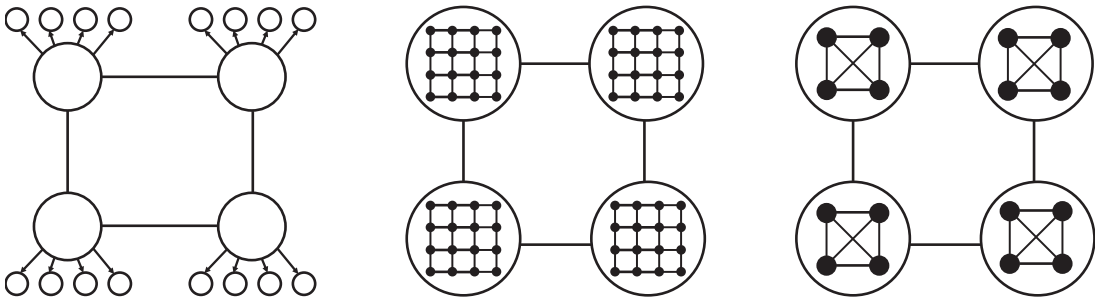
```
1 za svaku celiju resetke izvrsi paralelno
2 {
3   generisi jedinku na slucajan nacin
4 }
5 dok nije zadovoljen uslov_za_zavrsetak_algoritma
6 {
7   za svaku jedinku k izvrsi paralelno
8   {
9     odredi vrednost funkcije cilja jedinke k
10    odaberi susednu jedinku l
```

```
11      ukrsti jedinke k i l
12      zameni jedinku k novodobijenim potomkom
13      mutiraj jedinku k
14    }
15  }
```

Ситнозрни модел ПГА постиже готово линеарно убрзање с порастом броја процесора, али захтева далеко већи број процесора од осталих модела ПГА. Најприродније и најефикасније окружење за извршавање ситнозрне паралелизације су *SIMD* архитектуре.

2.2.4 Хијерархијски паралелни генетски алгоритам

Хијерархијски ПГА врше паралелизацију ГА на више нивоа, при чему се на сваком нивоу користи један од три стандардна модела паралелизације. Већина хијерархијских модела на највишем нивоу користи острвску паралелизацију, док се за следећи ниво бира или глобални [16] (слика 2.7а) или ситнозрни ПГА [33] (слика 2.7б). Постоје хијерархијски модели који на оба нивоа користе острвски ПГА (слика 2.7в), али тако да је на нижем нивоу миграциона топологија гушћа, а стопа миграције већа, него на вишем нивоу. Хијерархијски модели могу убрзати ГА више



Слика 2.7: Примери хијерархијских модела ПГА

него модели паралелизације који их сачињавају, пошто је њихово убрзање једнако умношку убрзања појединих модела од којих се хијерархијски модел састоји. Њихов недостатак се огледа у броју параметара које је потребно подесити, јер садрже све параметре модела који их чине, као и у великим захтевима када су у питању паралелне и дитрибуиране архитектуре на којима се могу имплементирати.

Глава 3

Преглед литературе

Истраживања извршена кроз ову докторску дисертацију укључују више различитих области: паралелне вишекритеријумске генетске алгоритме, извршавање ових алгоритама уз ефикасно коришћење дистрибуираних рачунарских ресурса, као и примену ВКГА за решавање оптимизационих проблема у области хидроинформатике. Отуда ће и преглед литературе бити подељен у три дела. У првом делу ће бити представљени најзначајни софтверски оквири за решавање проблема ВКО употребом паралелних ВКГА који се срећу у литератури. С обзиром на то да се за поједностављење паралелног извршавања ВКГА на Грид инфраструктури често користе системи засновани на пилот пословима, у другом делу ће бити дат преглед ових система. Коначно, у трећем делу поглавља, разматра се употреба ГА и ВКГА у области хидроинформатике, са посебним освртом на радове домаћих аутора.

3.1 Софтверски оквири за вишекритеријумске генетске алгоритме

Ефективност и ефикасност метахеуристика, а посебно ГА, у решавању реалних оптимизационих проблема, условили су њихову интензивну употребу и развој бројних софтверских оквира за њихово серијско и паралелно извршавање. У литератури се срећу бројни примери софтверских оквира који омогућавају оптимизацију коришћењем паралелних генетских алгоритама и других метахеуристика, а у овом делу рада ће бити представљени неки од њих.

Distributed BEAGLE [47] је софтверски оквир за паралелне ЕА настао као проширење *Open BEAGLE* [46] софтверског оквира који омогућава употребу ГА, генетског програмирања и еволуционих стратегија за решавање проблема оптимизације, али и додавање нових еволуционих алгоритама. *Distributed BEAGLE* имплеметира специфичну комбинацију дистрибуираних еволуција и глобалног модела паралелизације. Он има клијент-сервер архитектуру, и ради тако што процедуру еволуирања

једне генерације у оквиру популације, дели на два корака: еволуцију дема (подпопулација) и евалуацију прилагођености јединки. Сервер поседује базу дема које треба еволуирати и јединки које треба евалуирати. Клијенти који врше еволуцију потражују од сервера дему над којом ће извршити селекцију и генетске операторе, док се клијенти који врше евалуацију серверу обраћају са захтевом да им пошаље неевалуирану јединку. *Distributed BEAGLE* се може имплеметирати на *Beowulf* кластерима или на LAN мрежи рачунара, и користи TCP/IP комуникациони протокол.

Distributed BEAGLE је коришћен и у раду [35] у експериментима који су изведени са циљем да се верификује математички модел за предвиђање убрзања које се добија употребом глобалног модела паралелизације у ЕА. Резултати показују да се убрзање добијено моделом готово поклапа са убрзањем израчунатим кроз експерименте, и да софтверски оквири базирани на моделу "господар-слуга" могу пружити значајно убрзање процеса оптимизације у случају проблема које карактеришу дуге евалуације јединки. Аутори такође тврде да супротно уврежденом мишљењу, глобални ПГА могу бити подједнако, или чак и више робустни него острвски ПГА, јер могу ефикасно да користе нове ресурсе који евентуално постају слободни током њиховог рада.

У раду [109] аутори представљају *MOGA-G* - Грид оријентисани софтверски оквир за ВКО. *MOGA-G* има сервисно-оријентисану архитектуру и ради над *Globus* мидлвером. Оквир је имплеметиран тако да корисник у својству клијента користи два сервиса. Први сервис извршава ВКГА све до тренутка када је потребно евалуирати генерацију, која тада бива прослеђена клијенту. Клијент даље треба да се обрати другом сервису који обезбеђује евалуацију јединки на Гриду. Резултати евалуације се преко клијента враћају првом сервису, који онда наставља извршавање ГА. Аутори не пружају увид у перформансе развијеног оквира које свакако у великој мери морају бити деградиране високим комуникацијским трошковима које примењена архитектура намеће.

Simdist [58] је софтверски алат за паралелно извршавање ЕА на кластеру, који користи глобални модел паралелизације и MPI стандард за паралелно програмирање. *Simdist* је само систем за дистрибуирање јединки на радне чворове кластера (где ће се извршити њихова евалуација), који се кроз стандардне И/О токове може повезати са било којим софтвером који би извршавао ЕА. Комуникацијски трошкови су смањени тако што се јединке, уместо једна по једна, у серијама шаљу на евалуацију. Убрзање које се добија *Simdist*-ом је готово линеарно. Ипак, *Simdist* истраживачима пружа само систем за дистрибуирање јединки на радне чворове кластера, док се од њих очекује да развију или употребе неки од готових софтвера за извршавање ЕА, и модификују га тако да може да буде повезан са *Simdist*-ом.

Софтверски алат за острвску паралелизацију еволуционих алгоритама на Гриду који користи *Globus toolkit* представљен је у раду [78]. Као и *Simdist* ни овај алат не извршава ниједан ЕА, али се може повезати са било којим софтвером који би био задужен за извршавање ЕА.

MALLBA [3] и *ParadisEO* [14] су свеобухватни оквири за паралелне метахеуристике који се континуирано развијају од свог настанка. *MALLBA* је *C++* библиотека оптимизационих алгоритама који могу бити паралелизовани коришћењем *MPI* стандарда и то на LAN или WAN мрежама рачунара и кластерима. *ParadisEO* је *C++* објектно-оријентисани софтверски оквир за употребу постојећих и развој нових метахеуристика. Екстензија *ParadisEO* оквира, под називом *ParadisEO-CMW* [84] повезује *ParadisEO* са *MW* [51] оквиром како би се омогућило извршавање паралелних метахеуристика садржаних у *ParadisEO*-у на Гриду. *MW* оквир пружа једноставну, "господар-слуга", паралелизацију израчунавања коришћењем *Condor*-а на Гриду. *ParadisEO-CMW* користи *MPI* стандард за паралелно програмирање и дозвољава следећа три модела паралелизације: глобални модел, модел острва и паралелизацију саме евалуације једне јединке. Модел острва се због "господар-слуга" оријентације *MW* оквира имплеметира као вишенитна програмска парадигма, тако што су острва распоређена у више нити на истом процесору. Перформансе оквира су испитане кроз решавање реалног проблема и постигнути су завидни резултати.

JG²A (Java Grid-enabled Genetic Algorithm) [11] је софтверски оквир креиран као проширење *JGA* [83] софтверског оквира, у циљу паралелног извршавања еволуционих алгоритама на Гриду. *JG²A* користи глобални модел паралелизације и омогућава паралелну евалуацију јединки у популацији генетског алгоритма, али и паралелизацију експеримената за подешавање параметара ГА. Иако је примена овог оквира у решавању реалних проблема планирања дала добре резултате са становишта убрзања, његова употреба је ограничена само на Грид који користи *Globus Toolkit 4* мидлвер и *Condor* менаџер локалних ресурса.

Један од новијих софтверских оквира за оптимизацију уз коришћење Грид ресурса је сервисно оријентисани *GridUFO* [93]. Овај оквир омогућава дељење оптимизационих алгоритама и проблема међу својим корисницима, решавање оптимизационих проблема коришћењем постојећих алгоритама, али и додавање нових алгоритама развијених у *C* програмском језику. *GridUFO* је један од ретких оквира који аутоматизује комплетан поступак заузимања грид ресурса и покретања послова на радним чворовима грид сајтова. Оно што представља главно ограничење овог решења јесте да се програми који врше евалуацију јединки морају имплементирати у *C* програмском језику. Информације о перформансама овог окрвира су јако оскудне, будући да су аутори приказали само резултате експеримената у којима је коришћен вештачки оптимизациони проблем и величина популације од само 90 јединки, а време које се проведе у реду чекања на слободне ресурсе на Гриду није урачунато.

GE-HPGA (Grid Enabled Hierarchical Parallel Genetic Algorithm) [77] је софтверски оквир за извршавање ефикасног хијерархијског паралелног ГА на Гриду. Окрвир скрива комплексност Грида од корисника и има двонивојску структуру: на првом нивоу, подпопулације се пребацују на удаљене кластере и покреће се њихова еволуција. На другом нивоу, евалуација индивидуа из подпопулација извршава се на радним чворовима. Аутори су дали теоријску анализу максималног убрзања које

је могуће постићи коришћењем *GE-HPGA*, као и практичне услове који морају бити испуњени да би се остварило минимално убрзање. Емпиријски резултати добијени решавањем тестног оптимизационог проблема са вештачком циљном функцијом потврдили су теоријска разматрања. Додатно, оквир је успешно примењен на решавање реалног проблема дизајна аеропрофила.

Грид оријентисани ГА, [59] се са великим успехом примењују у решавању многих реалних проблема. У раду [60] је приказан софтверски оквир за коришћење глобално параленог ГА на гриду за потребе решавање оптимизационих проблема из области биоинформатике, док су аутори рада [54] сличан софтверски оквир под називом *PGO* применили на проблеме хидроинформатике. Решавање проблема склапања фрагмената ДНК употребом грид оријентисаних ГА представљено је у раду [97], где глобално паралелни ГА дистрибуира евалуације на чворове Грида преко раније поменутог *MW* софтверског оквира. Детаљан преглед употребе паралелних метахеуристика у различитим областима, као и софтверских оквира за паралелне метахеуристике, дат је у раду [5], док је у раду[98] извршена обимна компаративна анализа и оцењивање софтверских оквира за оптимизацију засновану на метахеуристикама.

Велики број до сада развијених софтверских оквира се може користити само на хомогеним рачунарским ресурсима уз ограничење да се програми који врше евалуацију јединки могу развијати коришћењем само одређених програмских језика. Такође, веома мали број оквира потпуно ослобађа корисника бриге о детаљима везаним за рад са дистрибуираним рачунарским ресурсима. У циљу веће прегледности, у табели 3.1 је дат сажет приказ претходно представљених софтверских оквира и то кроз карактеристике које би биле од значаја истраживачу који жели да што једноставније примени неки од софтверских оквира за решавање сложених оптимизационих проблема.

3.2 Системи засновани на пилот пословима

Истраживачи су уложили велики напор како би олакшали коришћење Грид инфраструктуре за потребе науке и инжењерства. Један од популарних начина за поједностављење извршавања корисничких апликација на Гриду је употреба система заснованих на пилот пословима. Парадигма пилот послова олакшава коришћење Грид ресурса стварањем виртуелног приватног репозиторијума рачунарских ресурса над Грид ресурсима [108]. Између корисника и Грида поставља се додатни слој кога чине фабрика пилот послова и репозиторијум послова. Корисничке апликације не отпремају послове коришћењем стандардних мидлвера, већ их шаљу у ред централног репозиторијума послова. Фабрика пилот послова распоређује пилоте на доступне радне чворове Грида. Када пилот почне да се извршава, он се јавља фабрици тражећи конкретан посао из реда репозиторијума који ће да изврши. Дакле, пилот није прави посао већ се њиме алоцира слот у грид ресурсима који ће бити попуњен

Назив	Заступљени модели паралелизације	Дистрибуирана архитектура на којој се имплементира	Могућност додавања нових оптимизационих алгоритама	Коришћење произвољног програмског језика за имплементацију евалуатора	Аутоматизован приступ ресурсима
Distributed BEAGLE [47]	Комбинација дистрибуираних подпопулација и глобалног модела	Кластер	ДА	НЕ	НЕ
MOGA-G [109]	Глобални модел	Грид	ДА	Нејасно	ДА
Simdist [58]	Глобални модел	Кластер	ДА	ДА	НЕ
Алат за дистрибуирано извршавање ЕА [78]	Модел острва	Грид	ДА	Нејасно	ДА
MALLBA [3]	Глобални модел, модел острва и паралелизација једне евалуације	LAN или WAN мреже рачунара и кластер	ДА	НЕ	НЕ
ParadisEO-CMW [84]	Глобални модел, модел острва и паралелизација једне евалуације	Грид	ДА	НЕ	ДА
JG^{2}A [11]	Глобални модел	Грид	ДА	НЕ	НЕ
GridUFO [93]	Творац алгоритма дефинише модел паралелизације	Грид	ДА	НЕ	ДА
GE-HPGA [77]	Хијерархијски (први ниво:модел острва; други ниво: глобални модел)	Грид	НЕ	-	ДА

Табела 3.1: Преглед развијених софтверских оквира за паралелно извршавање ГА

конкретним послом. Пилот посао није везан само за један конкретан кориснички посао, нити само за једног конкретног корисника, чиме је омогућено коришћење једне пилот инфраструктуре од стране више корисника из исте виртуелне организације. Ако нема задатака који у реду репозиторијума чекају пилот посао, онда се он одмах окончава. Након обављања посла који му је додељен, пилот може да прихвати нови кориснички посао, и тако поново искористи већ заузете Грид ресурсе. Овакво повезивање са кашњењем, између корисничких послова и ресурса у великој мери унапређује корисничко искуство, с обзиром на то да скрива разуђеност Грид ресурса и пружа прецизније информације о њиховој доступности.

Најпознатији оквир за предавање послова на Грид заснован на пилот пословима је *glideinWMS* [107], који представља надоградњу *Condor WMS*-а [20]. *Falkon* [100] је још један систем пилот послова, који ради над *Globus* мидлвером, остварујући значајну скалабилност и ефикасност у отпремању послова. Остали примери система заснованих на пилот пословима су *Coasters* [53], *DIANE* [90], и *BigJob* [80]. На Уверзитету у Београду развијен је апликациони Грид сервис који се базира на пилот пословима под називом *Work Binder (WB)* [82]. *WB* клијентским апликацијама обезбеђује интерактивност и брз приступ Грид ресурсима. Он повећава искоришћење инфраструктуре пружајући аутоматизовану еластичност у њеном заузећу, на основу тренутног и недавног понашања клијента. Извршава се над *EMI/UMD* мидлвером, али се може се лако прилагодити и другим Грид платформама и платформама рачунарства у Облаку.

3.3 Генетски алгоритми у хидроинформатици

Решавање сложених проблема у области хидроинформатике неминовно захтева коришћење метода вишекритеријумске оптимизације, међу којима превагу имају оне методе које се заснивају на ГА. Бројни су радови који приказују успешну употребу ВКГА у хидроинформатици: [102], [7], [37], [10], [21], [55]. Преглед употребе ВКО засноване на еволуционим алгоритмима у хидроинформатици дат је у раду [103]. Примери паралелизације ГА у циљу убрзања процеса оптимизације у хидроинформатици могу се срести у радовима [54], [18]. У првом раду је представљена рачунарска платформа под називом *PGO*, развијена на *Perl* програмском језику која омогућава господар-слуга паралелизацију евалуација у једноставном ГА. Апликација користи *MySQL* базу података за чување података током процеса оптимизације, те је за њено коришћење, на рачунару који ће имати улогу господара потребно инсталирати *Perl* и *MySQL DBMS* и ручно ископирати рачунски модул на сваки чвор који ће вршити евалуацију јединки. Систем је тестиран на два оптимизациона проблема из области хидрологије на хетерогеној мрежи рачунара са различитим оперативним системима, и дате су криве поклапања мерених и добијених вредности параметара релеватних за решаване проблеме. У раду није дата анализа убрзања које се добија коришћењем предложеног система, те се не могу сагледати добробити његовог

коришћења у општем случају, нарочито ако се узме у обзир сложеност његове имплементације са становишта корисника. У другом раду се обрађује проблем вишекритеријумске калибрације модела отицаја коришћењем острвског паралелног ГА на кластеру и показује да се коришћењем паралелних ГА на конкретном проблему може постићи већа ефикасност калибрације и да се могу наћи боља решења него када се користи серијски ГА. У раду [116] аутори су анализирали перфомансе ϵ-*NSGAII* [71] алгоритма у зависности од стратегије паралелизације која је на њега примењена: господар-слуга или острвска. Тестови су вршени коришћењем једног проблема из области хидрологије, затим проблема водних ресурса, као и једног вештачког проблема са компликованом циљном функцијом. Показало се да је за решавање вештачког проблема ефикаснији острвски модел паралелизације, док је у случају реалних оптимизационих проблема, поред једноставности имплеметације, господар-слуга модел имао супериорне перформансе у односу на острвску паралелизацију.

У истраживањима из области хидрологије ВКГА се користе за калибрацију хидролошких модела, и то најчешће *SWAT* [64] модела речних сливова. *SWAT* је физички заснован модел, који користи дневни временски корак и предвиђа утицај кретања воде, седимента и пољопривредних хемикалија на управљање сложеним сливовима. У раду [104] калибрација *SWAT* хидролошког модела је паралелизована на *Windows* платформи. У процесу одређивања оптималних параметара модела коришћен је *SUFI2* програм за оптимизацију који нуди само ограничен број циљних функција којима се могу оцењивати излази из модела у поређењу са мереним вредностима. Систем је тестиран на највише 24 процесора, при чему је достигнуто убрзање које је далеко од идеалног. Са друге стране, група аутора је развила *gSWAT* - web базирану апликацију за калибрацију сложених *SWAT* хидролошких модела [50]. *gSWAT* користи Грид ресурсе за паралелизацију симулација *SWAT* модела у циљу бржег проналажења оптималних параметара модела. Сложеност Грида скривена је од корисника коришћењем два посредника: раније поментутог *DIANE* у улози господара и *GANGA* [91] система *Phyton* скрипти за управљање Грид пословима. Систем је тестиран на максимално 100 радника, али јасна крива и/или табела која би приказивала добијено убрзање у поређењу са идеалним није дата. Гридификација хидролошких израчунавања коришћењем *SWAT* модела приказана је у раду [120], где аутори предлажу поделу целокупног модела на подмоделе и њихову даљу дистрибуцију на Грид (коришћењем *DIANE* и *GANGA*) ради паралелног извршавања.

У складу са светским трендом решавања проблема хидроинформатике коришћењем ВКГА и домаћи аутори се ослањају на ову методу. Прва истраживања из области генетских алгоритама и њихове примене у хидроинформатици на Универзитету у Крагујевцу извршио је др Никола Миливојевић што је резултирало његовом докторском дисертацијом [86] и радом [89]. Аутор проблеме ВКО из области хидрологије своди на проблеме са једном функцијом циља, употребом тежинских сума. Проблеме даље решава адаптивним ГА са бинарно кодираним хромозомима, код кога фази логички контролер управља параметрима алгоритма (степен укрштања, степен мутације, величина популације и сл.). Господар-слуга паралелизација ГА извршена је

на мрежи *Windows* рачунара и постигнуто је значајно убрзање. Ова методологија је касније примењивана у многим истраживачко-развојним пројектима, где су учествовали и други аутори што је резултирало бројним публикацијама [70], [30], [112], [88], [29], [113]. Последњих година су рађена истраживања и у области паралелизације генетских алгоритама [63], као и о значају употребе Грид ресурса у хидролошким апликацијама [76]. Аутори су у раду [76] разматрали различите аспекте примене Грида у науци и инжењерству и то кроз опис шест различитих хидролошких апликација које за своје извршавање користе Грид. Резултати показују да скалирање рачунарских ресурса омогућава брже добијање резултата и повећава просторну и временску резолуцију модела. Међутим, апликације које се одвијају у реалном времену и даље се сусрећу са извесним потешкоћама на Грид инфраструктури, јер без одговарајућег механизма за резервацију, велики број рачунарских чворова, који је потребан за њихово успешно извршавање, није гарантовано на располагању у сваком тренутку.

3.4 Резиме прегледа литературе

У литератури се срећу бројни примери софтверских оквира која омогућавају оптимизацију коришћењем паралелних генетских алгоритама. Већина њих се може користити само на хомогеним рачунарским ресурсима уз ограничење да се програми који врше евалуацију јединки могу развијати коришћењем само одређених програмских језика. Реални проблеми вишекритеријумске оптимизације из области хидроинформатике најчешће подразумевају одређивање оптималних вредности великог броја параметара сложених система. Евалуација сваке јединке, која представља једну могућу конфигурацију параметара, врши се на основу излаза из симулираног модела система. Постојећи модели хидро система и њихови симулатори развијани су коришћењем различитих програмских језика као што су *Fortran*, *C++*, *.NET C#*, *MATLAB* итд. Отуда је употреба постојећих софтверских оквира за вишекритеријумску оптимизацију у хидроинформатици веома ограничена, због немогућности коришћења произвољног програмског језика за развој евалуатора.

Кључни недостатак свих до сада развијених софтверских оквира за оптимизацију коришћењем паралелних генетских алгоритама је статичка стратегија заузимања дистрибуираних рачунарских ресурса, која подразумева да једном заузети ресурси остају заузети све до краја оптимизационог процеса, који може потрајати и данима. С обзиром на то да дистрибуираним рачунарским ресурсима конкурентно приступају бројни корисници, статичка стратегија заузимања ових ресурса од стране софтверских оквира за оптимизацију онемогућава остале кориснике да им приступе. Еластични приступ ресурсима, са временски ограниченим резервисањем истих, омогућио би осталим корисницима тих ресурса бржи приступ. Са становишта корисника самог софтверског оквира за оптимизацију, аутоматска, еластична алокација ресурса пружила би додатни комфор, јер не захтева чекање на тачно одређене капацитете,

већ користи оно што је тренутно доступно, али заузима и додатне ресурсе чим постану слободни.

Полазне основе ове докторске дисертације чине резултати досадашњих истраживања на пољу вишекритеријумске оптимизације засноване на генетским алгоритмима и паралелизације генетских алгоритама као и истраживања у области хидроинформатике. С друге стране, дисертација ће се ослањати и на постојеће резултате у домену развоја софтвера који омогућавају једноставну и ефикасну употребу дистрибуираних рачунарских ресурса и заснивају се на пилот-пословима.

Глава 4

Опис развијених софтверских оквира за паралелну вишекритеријумску оптимизацију засновану на генетским алгоритмима

Досадашња употреба ГА за оптимизацију у области хидроинформатике од стране домаћих аутора захтевала је примену експертског знања за доделу тежина сваком од циљева, као и репетитивну примену ГА у сврху налажења више алтернативних решења проблема оптимизације. Како би процедура проналажења вишеструких решења проблема ВКО у хидроинформатици била поједностављена, уочена је потреба за коришћењем ВКГА. Додатно, с обзиром на то да је паралелизација ГА за потребе решавања комплексних реалних проблема неопходна, а да је вршена само коришћењем мреже неколико *PC* рачунара, било је потребно омогућити паралелизацију на инфраструктурама које пружају далеко бројније ресурсе (попут рачунарских кластера и Грида). Имајући у виду претходно наведене потребе развијено је неколико софтверских оквира који ће бити представљени у наставку ове главе.

4.1 Принципи развоја софтверских оквира

Како би се омогућило коришћење ВКГА и њихова паралелизација на дистрибуираним рачунарским ресурсима, приступило се развоју софтверског оквира који би испунио следеће захтеве:

1. Паралелно извршавање вишекритеријумске оптимизације засноване на генетским алгоритмима на дистрибуираним рачунарским ресурсима који укључују кластере високих перформанси и Грид.

2. Паралелизација може бити двојака - паралелна евалуација јединки у ГА и па-

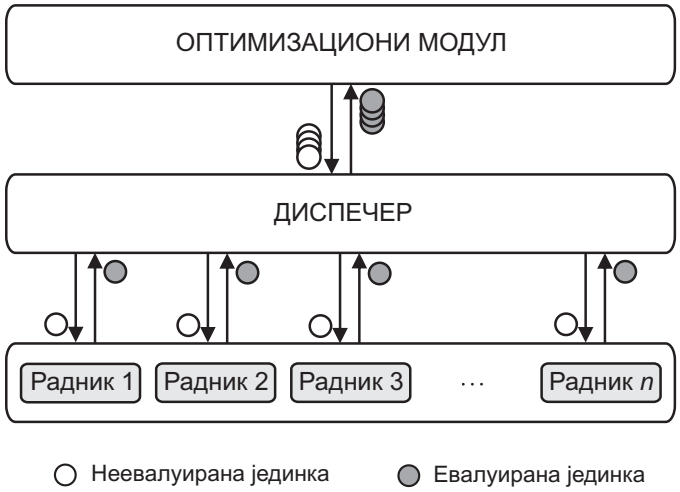
ралелно извршавање више инстанци паралелизоване оптимизације.

3. Могуће је једноставно додавање нових оптимизационих алгоритама.
4. Развој евалуатора јединки је независан од развоја оптимизационих алгоритама.
5. Евалуатори могу бити написани у било ком компајлерском или скрипт језику подржаном од стране оперативног система или окружења на коме ће се извршавати.
6. Корисници су потпуно ослобођени бриге о специфичним детаљима дистрибуираног рачунарства, и обезбеђен им је брз приступ дистрибуираним ресурсима.
7. Аутоматска, еластична алокација ресурса обезбеђује кориснику софтверског оквира да не чека на ослобађање тачно одређених капацитета, већ да користи оно што је тренутно доступно, али и да заузима додатне ресурсе чим они постану слободни.
8. Еластични приступ ресурсима, са временски ограниченим резервисањем, омогућава осталим корисницима тих ресурса бржи приступ, уз отклањање могућности загушења.

Током рада на постизању ових циљева ишло се градацијски. Развијена су четири софтверска оквира, тако да је сваки наредни отклањао недостатке претходно развијеног оквира и био ближи испуњењу свих постављених циљева. Оно што је заједничко за сва четири оквира јесте да обезбеђују паралелно извршавање једнокритеријумске и вишекритеријумске оптимизације засноване на ГА. Сви оквири користе господар-слуга модел паралелизације ГА, код кога се евалуација јединки дистрибуира на више чворова - слуга, док чвор - господар секвенцијално извршава остатак алгоритма. Употреба господар-слуга модела паралелизације ГА обезбеђује да евалуација јединки буде потпуно одвојена од остатка алгоритма и да отуда функција циља може бити написана у било ком компајлерском или скрипт језику, што ове оквире чини погодним за решавање проблема оптимизације у различитим областима науке и инжењерства.

На високом нивоу апстракције, архитектура софтверских оквира који ће надаље бити представљени може се приказати кроз три слоја, као на слици 4.1.

Сви развијени оквири се састоје од модула за ВКО, који извршава главни ток ГА, диспечера и радника. Диспечер има улогу посредника између господара и радника. Он јединке послате на евалуацију од стране господара расподељује радницима, који ће ту евалуацију обавити. Оваква арихитектура захтева раздвајање секвенцијалног ГА на два дела: главну еволуциону петљу, коју извршава господар и евалуацију јединки коју извршавају радници. Као што се може уочити са слике 4.2, господар извршава еволуциону петљу у главној нити све до тренутка када треба евалуирати генерацију. Тада се у посебној нити покреће диспечер. Он прима индивидуе од господара и смешта их у ред у коме чекају да дођу на ред за евалуацију.



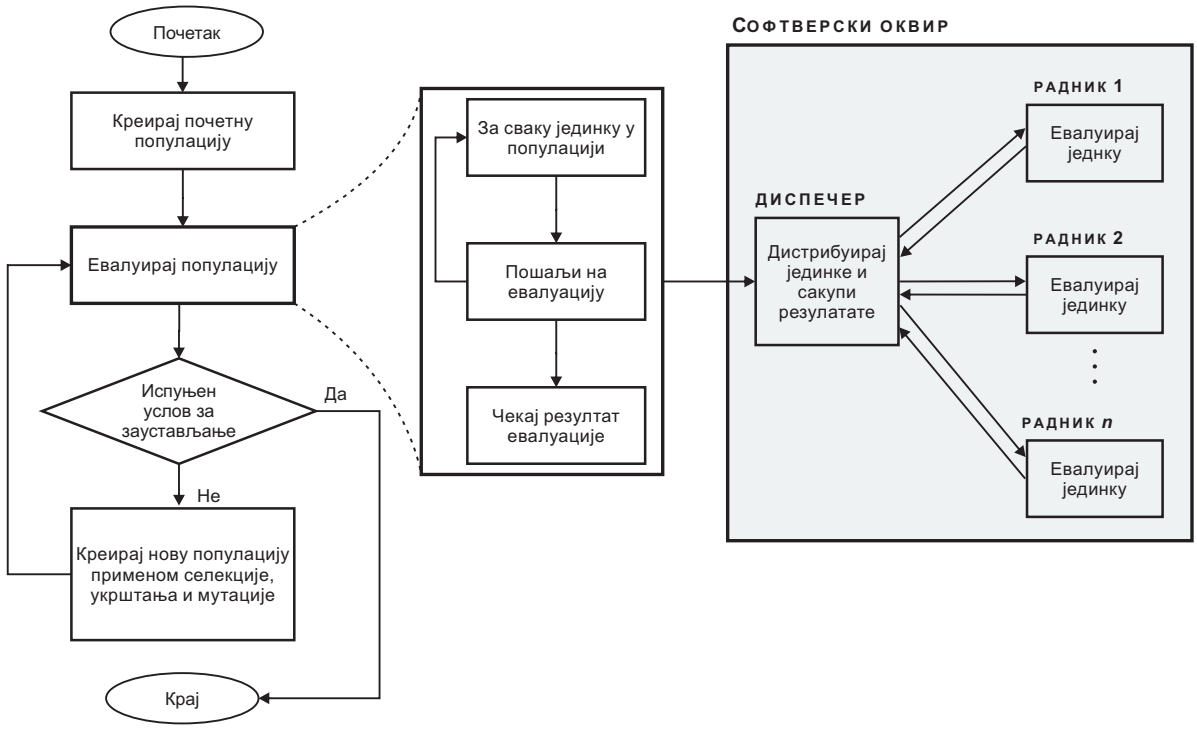
Слика 4.1: Архитектура развијених софтверских оквира

Диспечер индивидуе редом шаље радницима који их оцењују и те оцене потом враћа од радника ка господару. Главна нит остаје заустављена док све индивидуе из генерације не буду евалуиране и потом наставља са остатком ГА. Особа или тим који развија алгоритам за оптимизацију мора да раздвоји евалуациони корак од остатка алгоритма, али не мора да брине о детаљима дистрибуирања индивидуа и враћања резултата евалуације. На диспечеру је да јединке ефикасно расподели радницима и да прикупи повратне резултате.

Развој софтверског оквира за паралелно извршавање вишекритеријумске оптимизације засноване на генетским алгоритмима на дистрибуираним рачунарским ресурсима, који испуњава све наведене циљеве, захтевао је употребу широког спектра метода развоја софтвера. Објектно-оријентисаној парадигми се тежило где год је то било могуће, тако да су алгоритми, спецификације проблема и евалуатори потпуно одвојени и могу их развијати различите особе или тимови. Делови оквира који се тичу резервације дистрибуираних ресурса користе и процедуралну парадигму кроз одговарајуће скрипт језике. Прилагођавање постојећих хидро кодова улози евалуатора захтева употребу како компајлерских програмских језика попут *Fortran*-а и *C++*-а, тако и интерпретерских језика као што су *MATLAB* и *Python*.

4.2 Опис развијеног модула за оптимизацију засновану на генетском алгоритму

Модул за вишекритеријумску оптимизацију засновану на ГА назван је *JAPE*, као алузија на народску дефиницију вишекритеријумске оптимизације - "и јаре и паре". *JAPE* је објектно-оријентисана библиотека класа развијена коришћењем *.NET C#* језика, по узору на *JMetal - Java* библиотеку класа отвореног кода [36], [65].



Слика 4.2: Општи шематски приказ начина рада развијених софтверских оквира

Одлука да се развије нова верзија библиотеке за ВКО, уместо да се искористи постојећи *JMetal*, донета је из два разлога. Први разлог лежи у чињеници да у време када је отпочет рад на *JAPE*-ту, *JMetal* није нудио опције за паралелно извршавање алгоритама. Он данас нуди могућност паралелне евалуације јединки у генерацији, али само у више нити коришћењем вишејезгарних процесора. Други разлог је што у датом тренутку није постојала *.NET* верзија *JMetal*-а, која је касније развијена [66]. Избор *.NET* платформе за развој *JAPE*-та долази отуда што је највећи број постојећих хидро кодова који се користе у улози евалуатора такође развијен коришћењем *.NET* платформе.

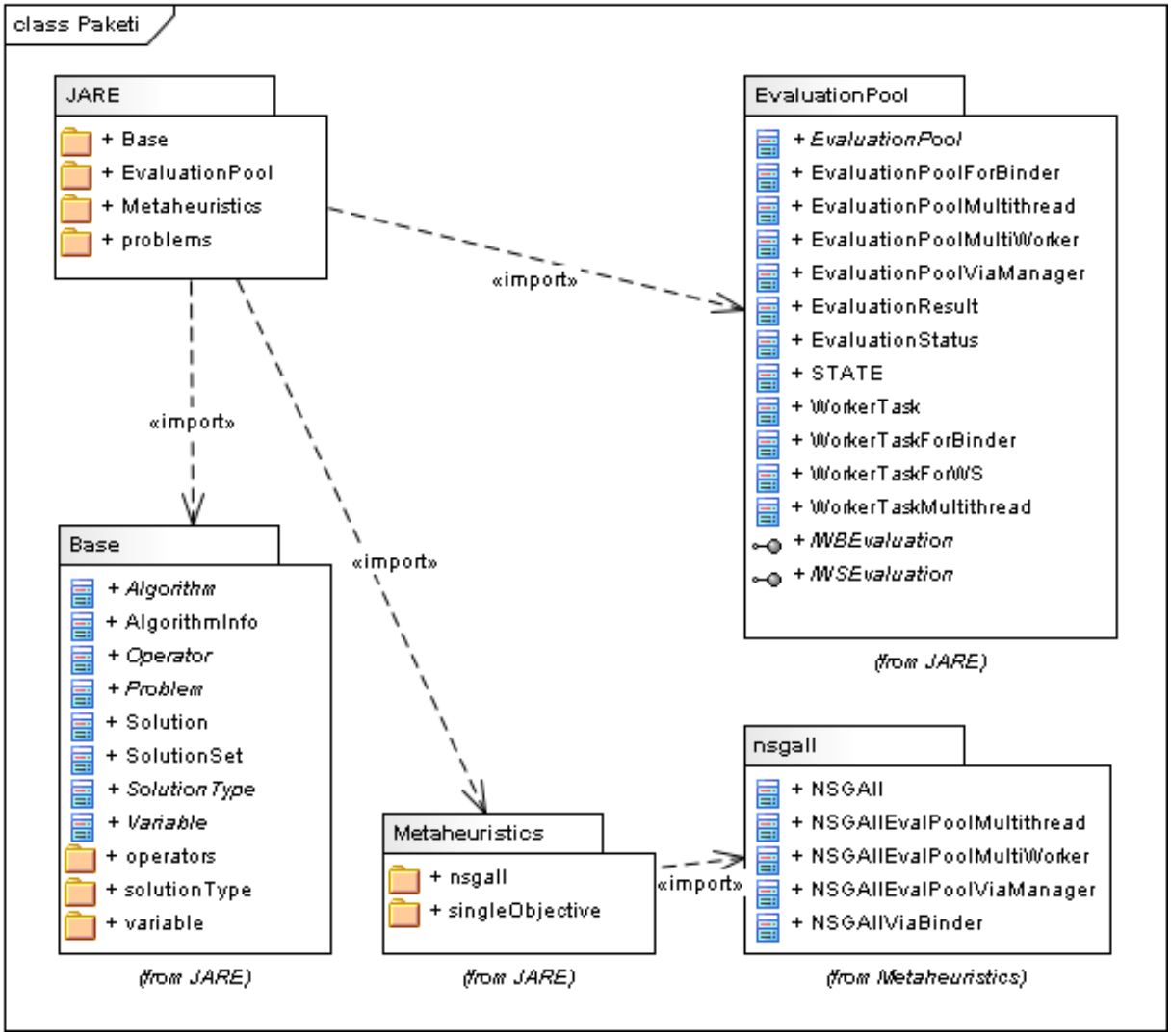
JAPE представља флексибилну и надградиву колекцију еволуционих алгоритама и пратећих компоненти неопходних за њихову реализацију. Захваљујући објектно-оријентисаној архитектури, једноставно је додавање нових и поновна употреба постојећих компоненти. Кориснику су на располагању базичне класе, које могу бити употребљене за креирање нових оптимизационих алгоритама, оператора које би ти алгоритми користили, као и нових оптимизационих проблема које је потребно решити.

JARE се састоји од неколико пакета класа од којих су најбитнији (слика 4.3):

- *Base*. Садржи основне класе за рад са еволуционим алгоритмима, као што су класе за представљање оптимизационих алгоритама, проблема, решења (јединки), оператора (селекција, укрштање, мутација,...), и слично.

- *Metaheuristics*. Садржи пакете класа које имплементирају различите метахеуристике за решавање проблема оптимизације укључујући и имплементацију *NSGA-II* еволуционог алгоритма за вишекритеријумску оптимизацију, који ће у овом раду бити коришћен.

- *EvaluationPool*. Овај пакет садржи класе неопходне за реализацију дистрибуиране евалуације јединки на вишепроцесорским рачунарима, рачунарским кластерима и Грид инфраструктури.



Слика 4.3: Основни пакети JARE библиотеке класа

На слици 4.4 приказан је дијаграм најбитнијих класа у JAPE-ту. Објекат типа *Algoritam* решава проблем (објекат типа *Problem*) над популацијом јединки у виду објекта класе *SolutionSet* и уз упоребу *Operator* објеката. Кључни метод класе *Algoritam* је метод *execute()*, којим извршава ток конкретног алгоритма. *Operator* је апстрактна класа за генетске операторе који се користе у ГА - селекцију, укрштање

и мутацију, док су и они сами апстрактне класе за конкретне врсте ових оператора (на пример: турнирску селекцију, симулирано бинарно укрштање и полиномну мутацију). Док *SolutionSet* представља популацију, тј. скуп јединки, појединачне јединке су представљене типом *Solution*. За сваку јединку се везује њен тип (*SolutionType*) и свака јединка (хромозом) се састоји од низа *Variable* објеката, који у контексту ГА представљају гене. Како би решио конкретан оптимизациони проблем употребом *JAPE*-та, корисник мора да развије нову класу која наслеђује класу *Problem*, и у њој имплеметира метод *evaluate()* којим ће се вршити оцењивање индивидуа. У случају паралелизоване евалуације јединки по моделу господар-слуга, овај метод извршавају слуге на радним чворовима дистрибуиране рачунарске инфраструктуре. У *JARE*-ту су до сада имплементирани стандардни ГА за потребе једнокритеријумске оптимизације, као и *NSGA-II* алгоритам за вишекритеријумску оптимизацију. Захваљујући објектно-оријентисаном приступу, проширивањем постојећих апстрактних класа, лако се могу додати и нови алгортми.

Како би се омогућила паралелизација процеса евалуације, *JAPE* библиотека класа је проширена *EvaluationPool* апстрактном класом чија је намена да послужи као база за извођење свих специфичних класа које врше евалуцију решења ван главне нити. Додатно су развијене и модификације стандардног ГА за потребе једнокритеријумске оптимизација и *NSGA-II* алгоритма за вишекритеријумску оптимизацију, које ове алгоритме имплементирају на такав начин да омогућавају паралелну евалуацију јединки. Тако постоје следеће класе:

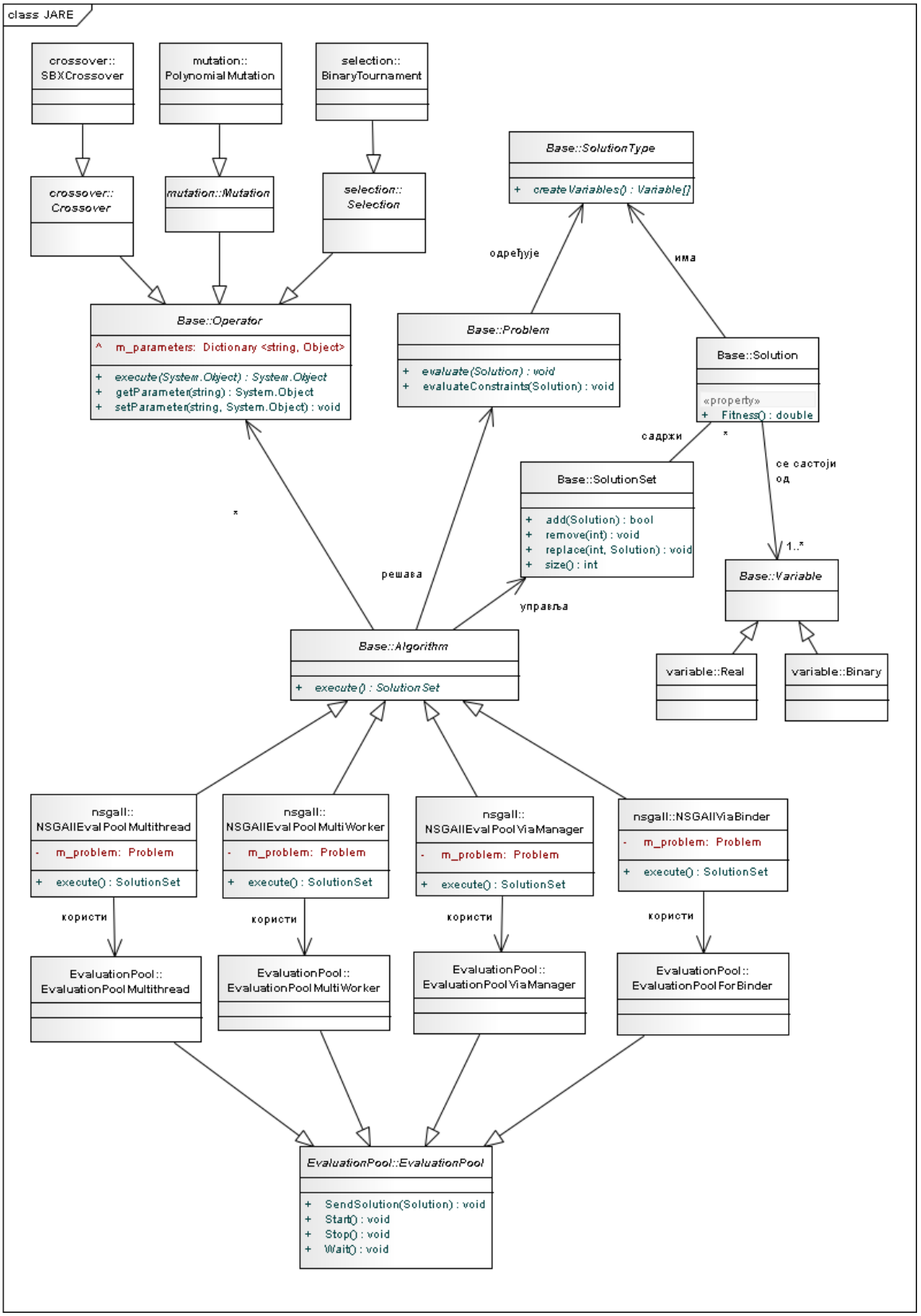
- *EvaluationPoolMultithread* обезбеђује паралелну евалуацију јединки у више нити на вишејезгарном рачунару. Одговарајуће имплеметације оптимизационих алгоритама које користе овај вид *pool*-а за евалуацију су *NSGAIIEvalPoolMultithread* и *SimpleGAEvalPoolMultithread*.

- *EvaluationPoolMultiworker* дистрибуира јединке које треба оценити на више радника имплементираних у виду веб сервиса. Одговарајуће имплеметације оптимизационих алгоритама су *NSGAIIEvalPoolMultiworker* и *SimpleGAEvalPoolMultiworker*.

- *EvaluationPoolToManager* јединке шаље менаџерском веб сервису, који их даље прослеђује својим клијентима - радницима на евалуацију. Одговарајуће имплеметације оптимизационих алгоритама су *NSGAIIEvalPoolViaManager* и *SimpleGAEvalPoolViaManager*.

- *EvaluationPoolForBinder* омогућава паралелну евалуацију јединки на рачунарском кластеру или Гриду употребом *WorkBinder* сервиса. Одговарајуће имплеметације оптимизационих алгоритама су *NSGAIIEvalPoolViaBinder* и *SimpleGAEvalPoolViaBinder*.

Сваки *EvaluationPool* садржи ред у који се смештају решења послата на евалуацију, као и методе за манипулацију овим редом. Када је потребно извршити



Слика 4.4: Дијаграм најбитнијих класа библиотеке JARE

оцењивање јединки у генерацији, ВКГА у посебној нити стартује *pool* за евалуацију позивом методе *Start()* класе *EvaluationPool*. ВКГА даље методом *SendSolution()* јединке асинхроно прослеђује у *pool* за евалуацију. Након што су све јединке из генерације послате, позива се метод *Wait()* класе *EvaluationPool* којим се обезбеђује заустављање главне нити алгоритма, све док се оцењивање генерације не заврши. Док траје оцењивање јединки евалуациони *pool* се налази у заузетом (*busy*) стању. Када се оцењивање заврши *EvaluationPool* шаље сигнал главној нити да може да настави са радом и прелази у слободно (*idle*) стање.

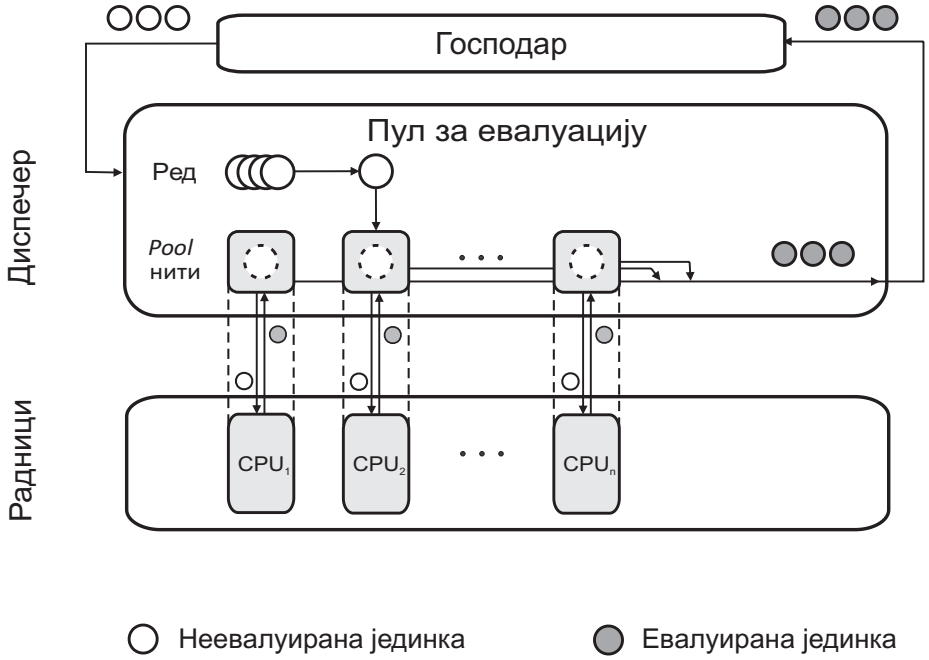
JAPE додатно, корисницима пружа могућност да процес оптимизације засноване на ГА не отпочну од случајно генерисане популације јединки, већ од популације која је добијена у некој генерацији током претходних извршавања оптимизационог алгоритма над идентичним проблемом. Ова опција се у реалној примени показала као изузетно корисна. Код сложених проблема, процедура проналажења оптималних решења може потрајати данима, и током овог процеса могу наступити непланиране потешкоће попут нестанка електричне енергије или отказивања неке рачунарске компоненте. Отуда је за корисника веома важна додатна сигурност која се огледа у чињеници да чак и када је вишедневна оптимизација нежељено прекинута, процес се не мора отпочети изнова, већ се може наставити од последње развијене генерације.

4.3 Вишенитни софтверски оквир

Први корак на путу стварања софтверског оквира који би испуњавао све захтеве наведене у делу 4.1 јесте паралелизација евалуације јединки у више нити на једном рачунару са вишејезгарним процесором.

Улогу диспечера има *pool* за евалуацију, реализован у виду раније поменуте класе *EvaluationPoolMultithread*. Он представља међукорак од ГА, кога извршава господар, ка рутинама које евалуирају јединке на појединачним процесорским јединицама. Сваки пут када треба оценити једну генерацију, господар у посебној нити старује *pool* за евалуацију и шаље му јединке из генерације. Главна нит алгоритма бива блокирана све док се не заврши оцењивање свих јединки из генерације. Када господар пошаље јединке на евалуацију, оне бивају смештене у ред *pool*-а за евалуацију (слика 4.5), где чекају да буду прослеђене на оцењивање. *Pool* за евалуацију даље стартује одређени број независних нити и кроз сваку нит додељује једном процесору једну јединку из реда за евалуацију.

Сам *pool* је имплементиран коришћењем *pool*-а нити ког обезбеђује *.NET ThreadPool* класа [95], [117]. *Pool* нити је колекција нити које се употребљавају за извршавање појединачних послова у позадини, независно од главне нити. Након што заврши свој задатак, нит се враћа у ред нити у *pool*-у нити које чекају да добију задатак за извршење. На тај начин се већ креиране нити користе изнова, уместо да се стално креирају нове нити. У развијеном оквиру, свака појединачна нит се користи за



Слика 4.5: Вишенитни софтверски оквир за паралелну евалуацију јединки

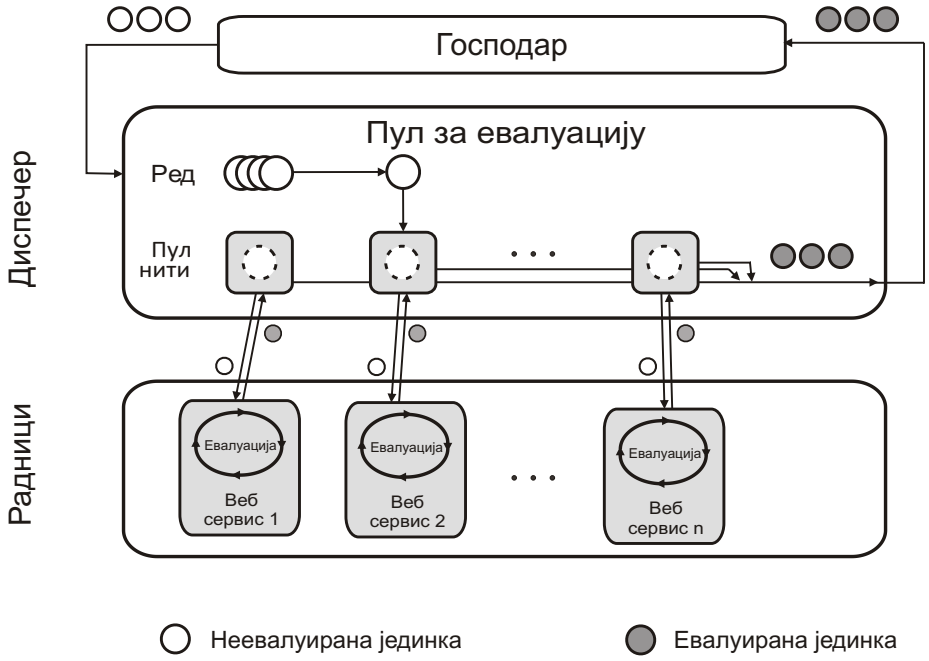
оцењивање једне јединке на једном језгру процесора. Отуда се евалуације обављају асинхроно, не заустављајући главну петљу *pool*-а за евалуацију, и не задржавајући процесирање наредних пристижућих захтева господара за евалуацију. Након што се евалуирање заврши и јединки додели резултат евалуације, нит се може поново употребити за остале индивидуе, које у свом реду у *pool*-у за евалуацију чекају да оду на оцењивање. Вишеструким коришћењем нити, избегава се трошак креирања посебне нити за сваки задатак. Када све јединке из реда *pool*-а за евалуацију буду оцењене, евалуациони *pool* шаље главној нити сигнал да су све евалуације завршене и да се може наставити са извршавањем остатка ГА.

4.4 Софтверски оквир заснован на *push* моделу дистрибуције јединки

С обзиром на то да је претходно развијени софтверски оквир нудио могућност паралелизације ГА само на једном вишепроцесорском рачунару, даљи развој је ишао у правцу паралелизације евалуације јединки на дистрибуираним рачунарским ресурсима. Са тим циљем је настао софтверски оквир заснован на *push* моделу дистрибуције јединки, који је први пут представљен у раду [114]. Као и код претходно описаног софтверског оквира, главни део диспечера је евалуациони *pool* (*EvaluationPoolMultiworker*), где се смештају јединке послате на евалуацију од стране господара. Како би се обезбедила паралелна евалуација на дистрибуираним ресурсима, радници су сада имплементирани у виду *WCF* веб сервиса који се извршавају

на радним чворовима, тако да диспечер има улогу клијента који захтева од тих сервиса да изврше евалуацију јединки.

Слика 4.6 илуструје начин рада софтверског оквира који се базира на *push* моделу дистрибуције јединки. Господар иницијализује евалуациони *pool* информацијом о доступним веб сервисима. Као и код вишенитног софтверског оквира, сваки пут када треба оценити генерацију, господар старује *pool* за евалуацију у посебној нити и шаље му јединке из генерације, а главна нит алгоритма бива блокирана све док се не заврши оцењивање свих јединки из генерације.



Слика 4.6: Софтверски оквир заснован на *push* моделу

Pool за евалуацију смешта јединке пристигле од господара у ред у коме чекају да буду послате на оцењивање. И у овом случају је за имплементацију *pool*-а за евалуацију искоришћен *pool* нити *.NET ThreadPool* класе. У развијеном оквиру се појединачне нити, које стартује *pool* за евалуацију, користе за слање јединки веб сервисима који се извршавају на радним чворовима. Свака нит позива веб сервис радника и шаље му јединку на евалуацију. Када је евалуација јединке завршена, резултат се враћа клијентској апликацији и додељује одговарајућој јединки. Када нит из *pool*-а нити обави један задатак евалуације, она се враћа у ред нити у *pool*-у нити које чекају да добију задатак за извршење. Тада се, као и одговарајући веб сервис може поново употребити за остале индивидуе, које у свом реду у *pool*-у за евалуацију чекају да оду на оцењивање.

Када сви веб сервиси врате резултате оцењивања и када у реду нема више јединки које треба оценити, у главној нити се, на основу сигнала евалуационог *pool*-а да су све евалуације завршене, наставља са извршавањем остатка ГА.

Ако током рада неки веб сервис пријави изузетак и оцењивање њему додељене јединке буде неуспешно, иста та јединка бива послата првом новом слободном веб сервису. Како би се избегло понављање грешке на неисправном веб сервису, евалуациони *pool* неће неко време слати јединке на евалуацију веб сервису који је пријавио грешку. На овај начин неисправни веб сервис не бива потпуно елиминисан из процеса евалуације, већ има шансу да се опорави.

Развијени оквир користи веб сервисе имплементиране коришћењем *Windows Communication Foundation (WCF) .NET* технологије [19], [79]. *WCF* је део *.NET* окружења који омогућава једноставан и брз развој сервисно-оријентисаних апликација. *WCF* се заснива на комуникацији путем порука између клијената и сервиса. Клијенти су апликације које иницирају комуникацију, док сервисне апликације чекају клијентске захтеве за комуникацијом и одговарају на њих. Сва комуникација са *WCF* сервисом одвија се преко његових крајњих тачака конекције (*endpoint*), који клијенту омогућавају приступ функционалностима *WCF* сервиса. Свака крајња тачка конекције пружа клијенту следеће информације: адресу која говори где се она налази; *binding* који специфицира начин комуникације клијента са крајњом тачком; уговор којим се пружају на увид доступне функционалности веб сервиса. Ове информације сервиси пружају на увид, како би било могуће генерисати одговарајуће *WCF* клијенте способне да комуницирају са жељеним сервисом. Како би поделили опис крајњих тачака конекције са клијентима, сервиси користе језик за опис веб сервиса (*Web Services Description Language - WSDL*), што сервис чини доступним за све *WCF* клијенте, без обзира на платформу на којој је сервис хостован. *WCF* подржава интероперабилност са *WCF* апликацијама које се извршавају на истом *Windows* рачунару или на различитим *Windows* рачунарима, али и са стандарднм веб сервисима попут *Java* веб сервиса који се извршавају на *Windows* или неком другом оперативном систему.

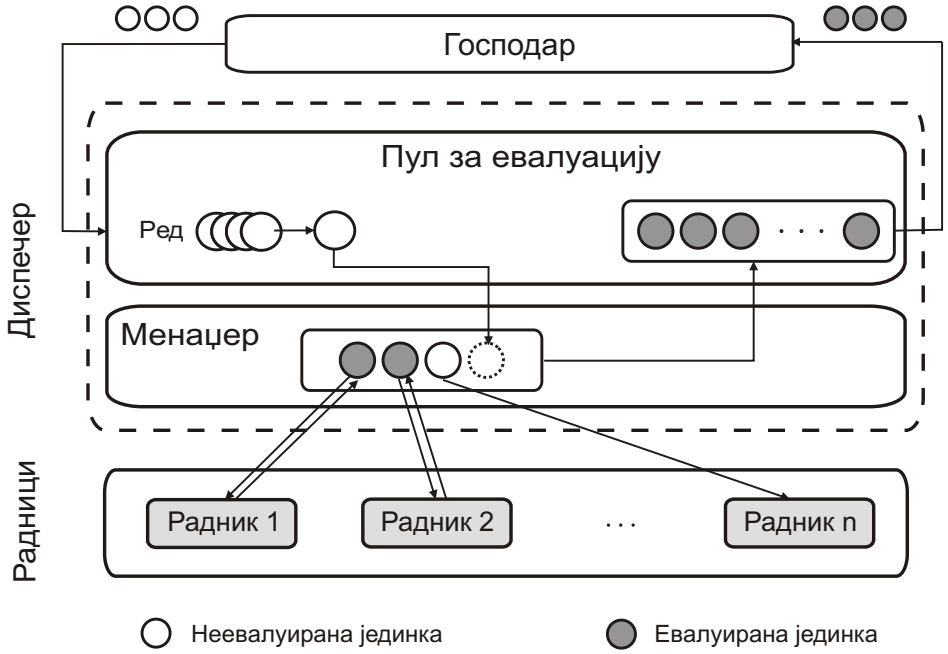
Мотивација за употребу *WCF* веб сервиса произилази из чињенице да је одређени број постојећих научних апликација које би се користиле за евалуацију јединки развијен коришћењем *.NET* платформе. На тај начин *push* оквир омогућава апликацијама базираним на *.NET* платформи да лако дистрибуирају евалуацију индивидуа на платформе које подржавају *WCF* (*Windows*, *Mono/Linux*, итд.). Додатно, када је реч о оцењивању прилагођености јединки, развијени оквир пружа могућност употребе како евалуатора базираних на *.NET* платформи, тако и евалуатора развијених на другим платформама, које су подржане од стране оперативног система који се налази у основи.

Треба нагласити да *push* модел дистрибуирања јединки има ограничење у раду на кластерима и Гриду, где радни чворови често обитавају иза *firewall*-а и у оквиру *NAT (Network Address Translation)* домена. *Firewall* и *NAT* спречавају диспечера да директно побуђује сервисе који обитавају на радним чворовима, што се решава *pull* комуникационим моделом који је искоришћен у софтверском оквиру који ће бити представљен у наредном делу овог поглавља.

4.5 Софтверски оквир заснован на *pull* моделу дистрибуирања јединки

Софтверски оквир који примењује *pull* модел дистрибуирања јединки природно се наметнуо као решење које отклања поменути недостатак *push* софтверског оквира. *Pull* софтверски оквир је представљен у раду [61].

Код *pull* комуникационог модела, диспечер прима захтеве и од господара и од радника. Господар шаље индивидуе диспечеру где оне чекају да буду оцењене. С друге стране, радници диспечеру шаљу захтеве за јединкама које је потребно евалуирати, извршавају евалуацију и диспечеру враћају резултате. На захтев господара, диспечер му враћа натраг евалуиране индивидуе.



Слика 4.7: Софтверски оквир заснован на *pull* моделу

Диспечер се код *pull* модела састоји од две компоненте: *pool*-а за евалуацију и менаџера (слика 4.7). За разлику од *push* модела, *pool* за евалуацију (*EvaluationPoolToManager*) код *pull* модела не комуницира са радницима директно, већ посредством менаџера који је уједно и једини веб сервис у целом софтверском оквиру. Менаџер је тај коме се обраћају и *pool* за евалуацију и радници, а његова улога је да буде посреднички *WCF* сервис између захтева за евалуаторима и доступних радника који би вршили евалуацију. Све активности менаџера представљају реакције на захтеве клијената.

Господар стартује *pool* за евалуацију у посебној нити и шаље му индивидуе на евалуацију, које се смештају у ред чекања у *pool*-у. Чим јединке почну да пристижу, евалуациони *pool* реагује тако што их шаље менаџеру. Када је цела генерација по-

слата, господар чека да све индивидуе буду оцењене и враћене из *pool*-а за евалуацију.

Радник се менаџеру пријављује као доступан да обавља оцењивање јединки и потражује од менаџера јединку коју би оценио. Када менаџер одговори на захтев радника, радник оцењује испоручену јединку, враћа резултат менаџеру, и након тога је поново слободан да од менаџера затражи нову јединку коју би евалуирао. Радници са менаџером комуницирају асинхроно, и имају улогу клијената који менаџеру приступају путем два различита типа обраћања. Први тип обраћања се користи када радник потражује посао од менаџера, док други тип обраћања подразмева враћање резултата оцењивања јединке менаџеру. Приликом сваког клијентског позива, менаџер стартује посебну нит сервиса како би могао истовремено паралелно да одговори вишеструким захтевима клијената.

Учесталост захтева упућених менаџеру од стране радника има кључни утицај на свеукупне перформансе *pull* софтверског оквира. Генерацијска природа ГА је таква да се јединке менаџеру на евалуацију не шаљу непрекидно, већ да постоје кратки временски размаци између периода оцењивања генерација, током којих нема јединки које би биле прослеђење радницима на оцењивање. То за последицу има повећану учесталост захтева радника, који се током ових периода непрекидно обраћају менаџеру са захтевима за послове, које им он тада не може доделити. Превише чести позиви менаџера смањују брзину његовог одзива, с обзиром на то да менаџер, како би се одазвао и одговорио на позив радника, мора сваки пут да заузме ресурсе и потом их ослободи. Како би се избегло ово уско грло у перформансама развијеног софтверског оквира, сваки радник који једном добије одговор од менаџера да тренутно нема индивидуа које треба евалуирати постаје неактиван одређени временски период (тзв. *workerIdleInterval*), током кога не може менаџеру слати нове захтеве за посао. Иако превише чести захтеви оптерећују и успоравају менаџера, и превише дуг период неактивности радника може довести до одлагања почетка оцењивања јединки и самим тим успорења рада читавог система. Дакле, веома је битно утврдити праву меру трајања *workerIdleInterval* периода, како би он био у складу са временским размаком између евалуација две узастопне генерације, и тако се избегло и преотерећење менаџера захтевима радника, и одлагање процеса дистрибуиране евалуације јединки.

Иако је развој софтверског оквира који се заснива на *push* моделу дистрибуирања јединки довео до значајних побољшања полазног вишенитног оквира, а *pull* модел дистрибуирања јединки додатно отклонио недостатке *push* модела, остварена је само половина наведених циљева. Прецизније, остварени су циљеви под редним бројевима 1, 3, 4 и 5. Потпуно остварење свих циљева постигнуто је развојем WoBinGO софтверског оквира који ће бити детаљно описан у наредном делу дисертације.

4.6 WoBinGO софтверски оквир

WoBinGO (***Wo**rk **Bin**der **G**enetic algorithm based **O**ptimization*) [62] је софтверски оквир за паралелно извршавање једнокритеријумске и вишекритеријумске оптимизације засноване на ГА и то употребом хетерогених ресурса, који укључују кластере високих перформанси и Грид базиран на *Globus* мидлверу. Кључна идеја приликом његовог развоја била је да се што је могуће једноставније и ефикасније искористе Грид ресурси у сврху убрзања оптимизације коришћењем паралелног ГА. WoBinGO је дизајниран тако да:

- Ослобађа истраживача од терета прибављања Грид ресурса и бављења различитим Грид мидлверима.
- Обезбеђује брзу алокацију Грид послова како би се избегло чекање да захтеви за Грид ресурсима буду обрађени од стране Грид мидлвера.
- Пружа флексибилну алокацију Грид послова у складу са динамиком корисничких захтева, чиме се избегава беспотребно заузимање драгоцених рачунарских ресурса.

Оно што посебно издваја WoBinGO оквир, како у односу на остале сличне оквире који се срећу у литератури, тако и у односу на три претходно представљена оквира, је то што он инкорпорира *Work Binder (WB)* сервис [82], који обезбеђује готово тренутни приступ Грид ресурсима и интерактивност са клијентским апликацијама. Интеграција *WB*-а у софтверски оквир омогућава програмеру да се фокусира на оптимизациони проблем и ослобађа га бриге о детаљима Грид рачунарства. Додатно, *WB* повећава искоришћеност Грид инфраструктуре тако што пружа аутоматизовану еластичност у њеном заузећу, базирану на садашњем и недавном понашању клијента.

4.6.1 Карактеристике *Work Binder* сервиса

Главна намена *WB*-а је да брзо алоцира нове послове за кориснике Грида и да од њих сакрије сложеност Грид инфраструктуре. *WB* се базира на употреби пилот послова, али има додатних предности. Као и остале инфраструктуре које користе пилот послове, и *WB* асинхроно стартује пилот послове како би их упарио са правим пословима које треба извршити. Међутим, код инфраструктура са пилот пословима упаривање се врши са подацима који описују који посао треба обавити, док *WB* успоставља директну везу клијента и посла са којим ће клијент бити у интеракцији. Уз овакав приступ, посао који нема одговарајућег клијента може одређено време провести у *pool*-у спремних послова, за разлику од правих пилот послова који бивају отказани чим немају задатак који треба да обаве. Комуникација између клијента и радника, у случају *WB*-а може бити надгледана и на њу се може утицати, док код

осталих система са пилот пословима то није могуће. Додатно, један *WB* сервис може да опслужује више корисника који деле исти *pool* спремних послова.

WB еластично користи Грид, у складу са доступношћу ресурса и тренутним оптерећењем. Као систем заснован на пилот пословима, *WB* управља скупом послова тако да увек тежи да има довољно послова у скупу за надолазеће клијенте, и да минимизује притисак на Грид инфраструктуру. Величина скупа послова може значајно да варира у зависности од тренутног оптерећења и динамике пристизања корисничких захтева. Ако нема активних клијената, *WB* се налази у моду мировања одржавајући број послова у *pool*-у на предефинисаном минимуму тако да Грид ресурси буду доступни осталим корисницима. Када постоји макар један активан клијент, *WB* се налази у активном моду и прилагођава број послова у *pool*-у динамици пристизања корисничких захтева. У активном моду постоје две стратегије допуњавања *pool*-а новим пословима: *регуларна* и *пун гас* (*full throttle*) стартегија. Регуларна стратегија подразумева да се послови равномерно дистрибуирају на сваки од *CE*-ова, и да се број спремних послова на њима одржава на одговарајућем нивоу. Број послова који ће бити послати на извршење на неки *CE* зависи од; (а) циљне величине *pool*-а, (б) тренутног броја спремних послова на *CE*-у, (в) броја послова који су већ прослеђени на *CE* и за које се очекује да постану спремни, (г) броја послова који ће се ускоро завршити и постати поново употребљиви. Када се израчуна број послова који се морају послати на извршење, *WB* сервис те послове шаље постепено током времена, како би се надоместило то што одзив *CE*-а није тренутан, и избегло нагло излагање система за пакетну обраду (*batching system*) високом оптерећењу. Стратегија "пуног гаса" се употребљава када број спремних послова падне испод предефинисаног прага услед високе фреквенције пристизања корисничких захтева, или засићења неког *CE*-а. *WB* шаље послове на извршење чим детектује да је укупна потражња за пословима већа од суме спремних послова и послова за које се очекује да ће ускоро постати спремни. Да би брзо обезбедио нове спремне послове, он одабира групу *CE*-ова са најкраћим временом одзива и на њих равномерно распоређује послове. Ако је на *CE*-овима са најкраћим временом одзива достигнута максимална величина *pool*-а, а и даље постоји потреба за новим пословима, онда ће они бити распоређени на преостале *CE*-ове. Оваква адаптивна политика предаје послова на извршење на Гриду и управљања *pool*-ом обезбеђује велику доступност спремних послова и готово оптимално коришћење Грид ресурса.

4.6.2 Карактеристике WoBinGO софтверског оквира

Кључне карактеристике које WoBinGO-у дају предност над сличним решењима која се срећу у литератури су:

- Комплексност Грид инфраструктуре је скривена од корисника.
- Аутоматска адаптивна алокација послова омогућава кориснику WoBinGO оквира

да не чека на ослобађање капацитета Грида, већ одмах користи оно што је доступно, и заузима нове ресурсе чим они постану слободни.

- Ограничени животни век послова обезбеђује бржи приступ осталим корисницима ресурса.
- Истовремено одвијање више инстанци паралелног ВКГА које деле исти *pool* послова.
- Развој функција циља је независан од развоја оптимизационих алгоритама.
- Функције циља могу бити написане на било ком компајлерском или скрипт језику подржаном од стране оперативног система или окружења на коме ће се функција извршавати.
- Примењене су стандардне сигурносне *GSI* (*Grid Security Infrastructure*) мере и откривање ресурса употребом *MDS* (*Monitoring and Discovery System*) система.

Истраживачи који користе грид оријентисане ГА су често суочени са проблемом мануелног проналажења и одабира доступних Грид ресурса. Услед многобројности и динамике Грид ресурса, овај процес је дуготрајан и непрактичан. Чак и након што су ресурси одабрани, следи процедура припреме Грид послова, предавања послова на Грид, чекања да послови дођу на ред да буду додељени раднику и њиховог мониторинга. Ослањајући се на *WB* сервис, WoBinGO скрива од корисника сва питања која се тичу коришћења Грид окружења. *WB* врши аутоматски одабир Грид ресурса и подношење послова, омогућавајући корисницима да се фокусирају само на оптимизациони проблем.

Главна одлика WoBinGO оквира, настала као последица инкорпорације *WB*-а је штедљивост у коришћењу ресурса која се постиже на два начина: (1) систем прилагођава број пилот послова тренутном оптерећењу и (2) предефинисани максимални животни век послова спречава да послови осталих корисника чекају на свој ред све до завршетка оптимизационог процеса.

Један од основних принципа Грид парадигме, који га чини тако привлачним за провајдере ресурса, је висока аутономија коју ужива сваки од кластера који улази у састав Грида. Та аутономија омогућава кластерима учешће у систему без жртвовања квантитета и квалитета рачунарских ресурса који су на располагању локалним корисницима тих истих кластера. Као последица тога, корисници Грида могу захтевати да користе рачунарске ресурсе, али немају гаранцију да ли ће и када ти ресурси постати доступни, тако да често бивају у ситуацији да немају на располагању довољан број ресурса за извршавање својих послова. Чекање да задовољавајућ број ресурса постане слободан може бити фрустрирајуће, трајати сатима и тако одвући истраживача од његовог примарног задатка. Управо у оваквим ситуацијама је аутоматска адаптивна алокација ресурса коју пружа WoBinGO оквир од велике важности. Инкорпорирани *WB* сервис кориснику обезбеђује да извршавање послова отпочне одмах

на тренутно доступним ресурсима. У складу са раније описаном адаптивном политиком предавања послова на Грид, *WB* даље аутоматски, чим нови ресурси постану слободни на њих распоређује додатне послове. Уз WoBinGO-ов аутоматизован, еластичан приступ при заузимању ресурса, корисник може бити сигуран да ће његове послове увек извршавати максимални могући број радника који је у датом тренутку доступан.

Оно што посебно издваја WoBinGO у односу на друге сличне софтверске оквире је ограничено трајање послова, које за последицу има брже ослобађање ресурса и њихову већу доступност осталим корисницима. Када животни век неког посла истекне, упркос даљој потреби за евалуацијама, систем га уклања и на извршење шаље одговарајући број нових послова. На овај начин, други корисници Грида добијају шансу да њихови предати послови постану активни током кратког временског прозора у коме се услед уклањања истеклих послова ослобађају ресурси. Код сличних система заснованих на пилот пословима, пилоти заузимају ресурсе током целог тока оптимизације, без обзира на то да ли стварно обављају неки посао или не, онемогућујући послове осталих корисника Грида и локалне пакет послове корисника датог кластера да дођу на ред за извршавање.

Способност *WB* сервиса да истовремено опслужује више корисника који деле исти *pool* послова, омогућава паралелно спровођење више инстанци оптимизационог алгоритма од стране више корисника WoBinGO-а. При томе се, за сваку инстанцу алгоритма, оцењивање јединки извршава паралелно. Ова могућност WoBinGO оквира показала се посебно значајном у његовој примени за калибрацију параметара хидролошким модела. С обзиром на сложеност математичких модела који се калибришу, често је тешко претпоставити праве границе у оквиру којих треба тражити вредности параметара модела. Испробавање различитих граничних вредности и њиховог утицаја на квалитет параметара модела захтева вишеструко спровођење оптимизационог алгоритма. Овај процес се уз WoBinGO оквир убрзава паралелним извршавањем више инстанци оптимизационог алгоритма. Свака инстанца примењује различите границе за параметре модела, те се истовремено, паралелно испитује које граничне вредности доводе до најбољих резултата. Још једна област примене ове особине WB сервиса је процедура подешавања параметара ГА. Вредности параметара ГА, попут степена укштања и степена мутације, у великој мери утичу на ефикасност алгоритма и његову способност да нађе решење блиско оптималном. Одабир оптималних вредности параметара одузима пуно времена, па се коришћењем WoBinGO оквира процес подешавања параметара може убрзати паралелним извршавањем више независних инстанци оптимизације са различитим параметрима алгоритма.

Као и код свих претходно описаних софтверских оквира, тако је и код WoBinGO-а развој функција циља потпуно независан од развоја оптимизационих алгоритама. Тиме је обезбеђено да функције циља могу бити написане на било ком компајлерском или скрипт језику подржаном од стране оперативног система или окружења на коме ће се функција извршавати, као и да развој алгоритама и развој цильних функција

могу обављати различите особе. WoBinGO софтверски оквир скрива сву комплексност Грид окружења од развијача алгоритама и циљних функција. С обзиром на то да не постоји потреба за "гридификацијом" кода који врши евалуацију јединки, време које треба посветити развоју евалуатора значајно је редуковано.

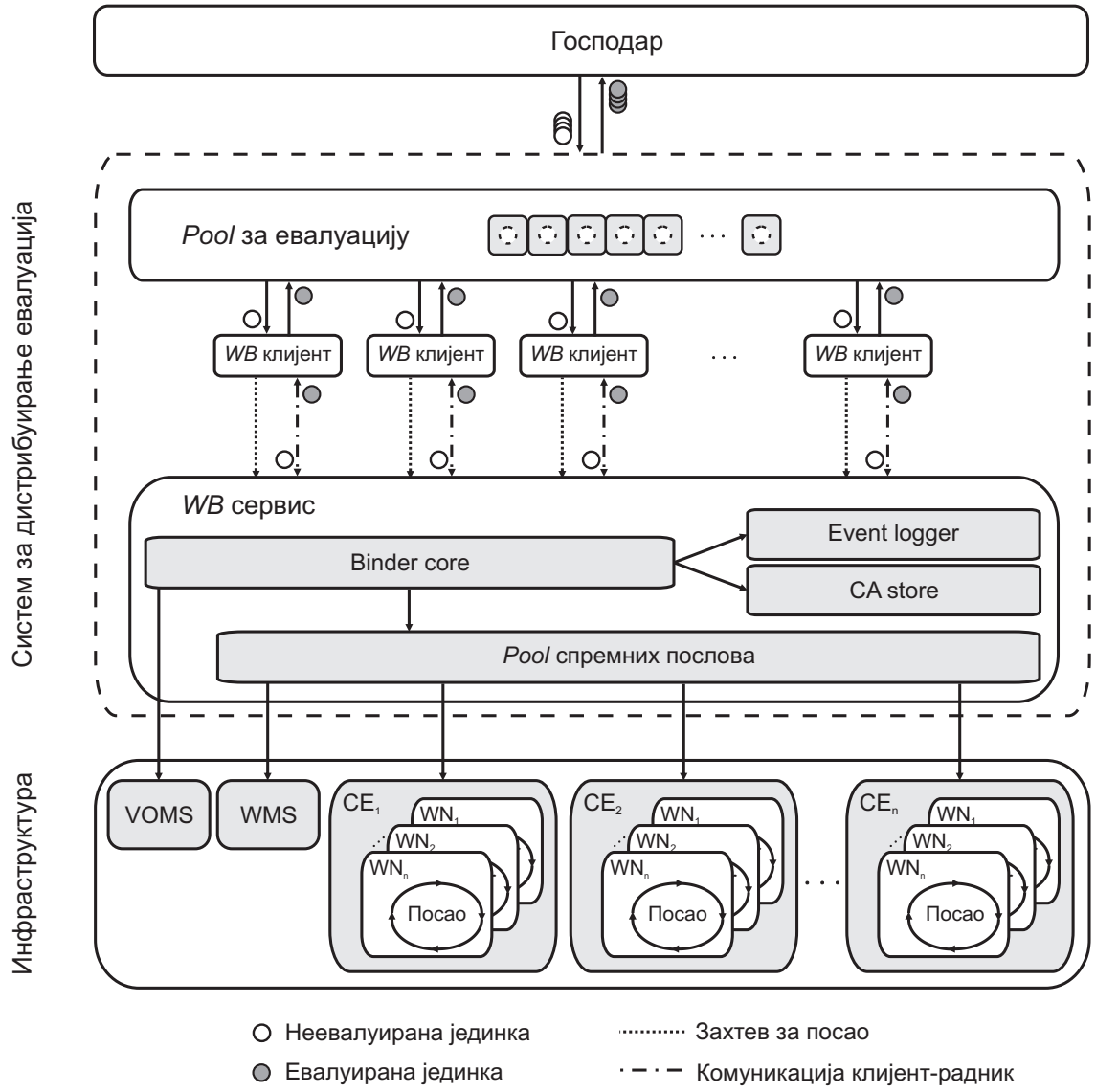
За разлику од претходно описаних оквира, као и већине оквира који се срећу у литератури, дистрибуција евалуатора на радне чворове Грида је у потпуности аутоматизована. Треба рећи да је приликом решавања реалних оптимизационих проблема веома чест случај да евалуатор буде јако комплексан, тако да подразумева извршавање симулације модела и да захтева велику количину података. На пример, у случају хидролошких проблема који укључују рачун коначних елемената, величина пакета за евалуацију се креће од 50MB до 500MB. Дешава се и да у пакету за евалуацију мора да се нађе комплетно окружење за извршење евалуације (на пример *Mono* окружење, одговарајућа верзија *Python* интерпретера и пратећих библиотека и слично), у случају да оно није локално доступно. Због обима датотека потребних за извршење евалуације јединки се пре почетка рада алгоритма, врши њихово пребацивање на складишне елементе Грида у виду пакета датотека. Послови које *WB* сервис распоређује на Грид, аутоматски преузимају пакет за евалуацију са *SE*-а и распакују га, чиме постају спремни да врше евалуацију јединки.

WoBinGO узима у обзир питања сигурности у приступу Гриду. Он користи стандардне *Globus* сигурносне мере обезбеђене кроз *GSI* и *MyProxy* сервис. Свака комуникација корисника и Грид послова преко *WB* сервиса мора бити сертификована одговарајућим *p12* сертификатом, који је издат од стране одговарајуће *CA*.

4.6.3 Архитектура

Основна структура WoBinGO софтверског оквира приказана је на слици 4.8. Оквир се састоји од главног модула за оптимизацију у виду JAPE десктоп апликације и диспечера који је далеко сложенији него код раније представљених оквира. Главни модул за оптимизацију, у терминима господар-слуга модела паралелизације ГА, представља господара који извршава главну еволуциону петљу. Диспечер WoBinGO оквира у себи садржи и читав *WB* сервис, те ћемо га, због његове сложености, назвати систем за дистрибуирање евалуација. Као што се може уочити на слици 4.8, систем за дистрибуирање евалуација јединки се састоји од *pool*-а за евалуацију и *WB* подсистема. Систем за дистрибуирану евалуацију јединки брине о извршењу евалуација јединки на радним чворовима Грида. Треба напоменути да су *pool* за евалуацију и *pool* спремних послова у *WB*-у потпуно различити ентитети.

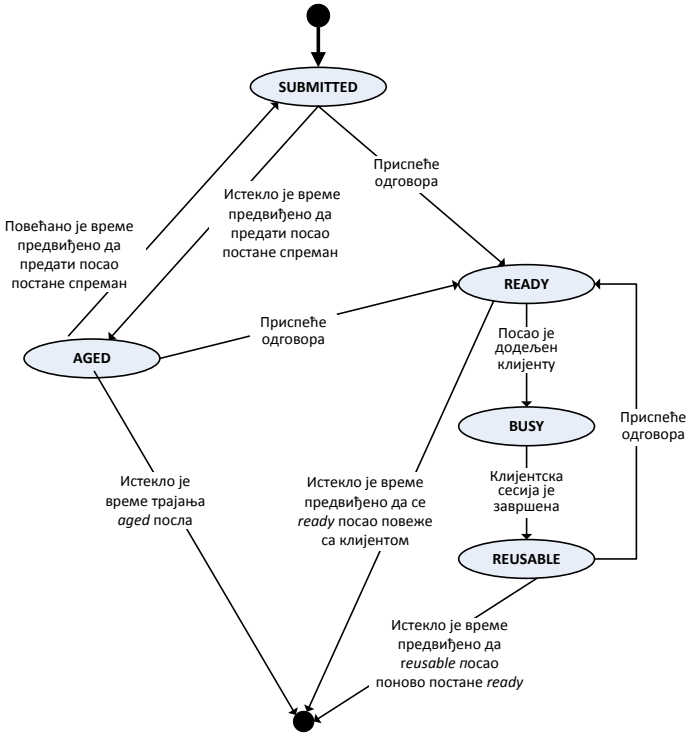
Господар извршава главну петљу алгоритма све до тачке када је потребно евалуирати генерацију. У том тренутку, он све јединке из генерације шаље у *pool* за евалуацију. Након што су све јединке евалуиране, и додељена им је вредност функције циља, господар наставља са извршавањем остатка алгоритма све док услов за заустављање не буде задовољен.



Слика 4.8: Архитектура WoBinGO софтверског оквира

Pool за евалуацију има улогу средњег слоја између господара и *WB* сервиса. Он обезбеђује асинхроне паралелне евалуације индивидуа из једне генерације. Сваки пут када треба евалуирати једну генерацију, *pool* за евалуацију добија од господара индивидуе из те генерецаје и смешта их у ред. *Pool* за евалуацију покреће *WB* клијента за сваку од јединки у реду. Када је јединка евалуирана, *pool* за евалуацију од *WB* клијента добија резултат и посматраној јединки додељује вредност функције циља. За разлику од претходно описаних софтерских оквира, *pool* за евалуацију код WoBinGO оквира није имплементиран коришћењем *pool*-а нити ког обезбеђује *.NET ThreadPool* класа. Разлог због ког није употребљена *ThreadPool* класа лежи у чињеници да она број нити у *pool*-у ограничава на 64, што је неприхватљиво мало за софтверских оквир који евалуацију јединки дистрибуира на Грид. Зато је код

WoBinGO оквира *pool* за евалуацију реализован коришћењем класе *Task* која је део библиотеке класа *TaskParallelLibrary* (*TPL*) [15], [118], први пут уведене кроз *.NET Framework 4*. *TPL* поједностављује увођење паралелизма и конкурентности у *.NET* апликације. Он динамички скалира степен конкурентности апликација у циљу што ефикаснијег искоришћења свих доступних процесора. *TPL* се заснива на концепту паралелних послова *Parallel Task*, то јест асинхроних операција које се могу обављати истовремено. Ови паралелни послови и њихово пратеће окружење обезбеђују богат *API* који подржава чекање, отказивање, настављање, робусно руковање изузецима, детаљно праћење статуса паралелних послова и друго. Отуда је *TPL* постао први избор када је у питању развој вишенитног, асинхроног и паралелног кода у *.NET* окружењу.



Слика 4.9: Дијаграм стања Грид послова

WB окружење се састоји од компоненти распоређених у три слоја: клијентски слој, *WB* сервис и слој радника. Клијент успоставља везу са *WB* сервисом и потражује услугу радника. *WB* сервис одржава *pool* спремних послова на Гриду и повезује их са клијентима који захтевају евалуацију. Када посао, кога је *WB* сервис предао на Грид, стигне на радни чвор побуђује се диспечер. Треба напоменути да је реч о интерном диспечерском програму, кога не треба мешати са диспечером евалуација, то јест системом за дистрибуирање евалуација WoBinGO оквира. Да би се нагласила разлика, овог диспечера ћемо назвати радников диспечер. Радников диспечер успоставља везу са *WB* сервисом, који потом статус одговарајућег посла мења из стања "предат" (*submitted*) у стање "спреман" (*ready*). Послови који након

одређеног временског периода не пређу у стање *ready* се сматрају застарелим и прелазе у стање "застарео" *aged* (слика 4.9). Овај период се током рада *WB* сервиса може кориговати у зависности од понашања конкретног *CE*-а. У случају споријег или бржег одзива *CE*-а, максимални дозвољени период за који предат посао треба да постане спреман, може да расте, односно опада. Зато је прелаз између *submitted* и *aged* послова на слици 4.9 двосмеран. Посао који је у стању *aged* провео више од времена предвиђеног за ово стање сматра се неуспешним, и *WB* сервис прекида везу са њим. Слично и посао који у *ready* стању дуже од времена које је предвиђено да се током њега *ready* посао повеже са клијентом, бива одбачен од стране *WB* сервиса. Када се *ready* посао повеже са клијентом, он постаје "заузет" (*busy*), а *WB* сервис се понаша као прокси преко кога иде сав саобраћај између клијента и одговарајућег посла. У случају дистрибуиране евалуације, клијент преко *WB* сервиса раднику шаље индивидуу, радник израчунава вредност функције циља за ту индивидуу и кроз *WB* прокси шаље резултат натраг клијенту. Након завштетка клијентске сесије посао постаје "поново употребљив" (*reusable*) и одређено време чека да буде поново пребачен у *ready* стање. Ако у предвиђеном временском периоду посао не постане *ready*, *WB* сервис прекида везу са њим.

Захваљујући вишеслојној архитектури са строго дефинисаним интерфејсима између слојева, господар-слуга модел паралелизације ГА се може заменити одређеним хијерархијским паралелним ГА, а да при томе остале компоненте софтверског оквира остану нетакнуте. Хијерархијски модели паралелизације ГА који могу да замене господар слуга модел комбинују острвски и господар-слуга модел, односно острвски и решеткасти модел паралелизације (слика 2.7а и 2.7б).

Ток рада WoBinGO софтверског оквира

Ради пружања јасније слике о начину функционисања оптимизације коришћењем WoBinGO оквира, у овом делу дисертације ће бити детаљно приказан ток рада WoBinGO софтверског оквира у виду корака неопходних да би се извршила једна оптимизација:

Корак 1.

Како би добио право да коришћењем WoBinGO оквира приступи Грид ресурсима, корисник пре свега мора да обезбеди исправан Грид сертификат. Он такође на почетку рада мора да припреми и конфигурациону датотеку за *WB* сервис. У конфигурационој датотеци за *WB* налазе се *URL* и кориснички акредитиви за Грид кориснички интерфејс, као и листа *CE*-ова које ће послови бити предавани. Ова датотека садржи и податак о минималном броју пилот послова (у ознаци p_{min}) које ће *WB* сервис распоредити на сваки *CE* да чекају захтеве клијената. У њој се такође задаје и максималан (у ознаци p_{max}) број послова који ће, у циљу испуњења клијентских потраживања, *WB* сервис моћи да преда на сваки од *CE*-ова. Број послова *WB*

сервиса ће се током рада еластично прилагођавати динамици пристизања захтева клијената и доступности жељених ресурса и варирати између p_{min} и p_{max} на сваком од *CE*-ова. Конфигурациона датотека за *WB* може садржати и друге *WB* параметре: максимално дозвољено време да предати посао пређе у стање када је спреман да буде отпочет, максимално дозвољено време да спреман посао успостави везу са клијентом и отпочне евалуацију, затим животни век посла итд.

Корак 2.

Користећи шаблонске датотеке и податке из *WB* конфигурационог фајла господар генерише скрипте и интерне конфигурационе датотеке потребне за рад *WB*-а. *WB* сервис ће аутоматски бити распоређен на чвор способан за предају Грид послова (преднодно поменути *UI* хост). Господар успоставља везу са *UI* чвором коришћењем *SFTP* протокола, и на њега пребацује два пакета датотека и *deployment* скрипту. Први пакет укључује сам *WB*, генерисане скрипте и конфигурационе датотеке. У другом пакету се налазе извршна датотека за евалуацију и додатне датотеке непоходне за рад евалуатора (ако има таквих).

Корак 3.

Господар покреће *deployment* скрипту чији је први задатак да креира валидан *Globus* прокси. Потом, она складишти горепоменути извршни пакет за евалуацију на Грид складишне елементе. Како би се време потребно за трансфер података са *SE*-а на *WN* свело на минимум и тако повећала брзина одзива распоређених послова, креирају се реплике складиштених података. Формирање реплика, међутим, не гарантује да ће датотека бити превучена са складишног елемента који је најближи радном чвору на коме се посао извршава. Ако датотека има реплике на више складишних елемената, она ће бити преузета са случајно одабраног *SE*-а. Како би посао сигурно преузимао реплику са себи најближег *SE*-а, употребљава *SURL* адресу датотеке [115].

Корак 4.

Пре почетка спровођења алгоритма, господар покреће *WB* сервис. Када је сервис покренут, он предаје одређени број Грид послова сваком *CE*-у који је наведен у *WB* конфигурационој датотеци. Када посао стигне на радни чвор и пребаци пакет за евалуацију са одговарајућег *SE*-а, побуђује се радников диспечер који успоставља везу са *WB* сервисом и постаје спреман за извршење евалуације.

Корак 5.

Господар започиње еволуциони процес. Сваки пут када је потребно евалуирати генерацију, господар шаље индивидуе у *pool* за евалуацију где се оне смештају у ред (слика 4.10). Главна нит алгоритма остаје заустављена све док све индивидуе, послате на евалуацију, не буду евалуиране. За сваку јединку коју треба евалуирати, *pool* за евалуацију покреће *WB* клијента у посебној нити. При томе, *pool* за евалуацију не чека да све јединке из генерације буду смештене у ред, већ асинхроно стартује појединачне клијенте за сваку пристиглу јединку. Оцењивање јединки се, дакле, обавља

независно од главне нити *pool*-а за евалуацију и без одлагања процесирања будућих захтева.

Корак 6.

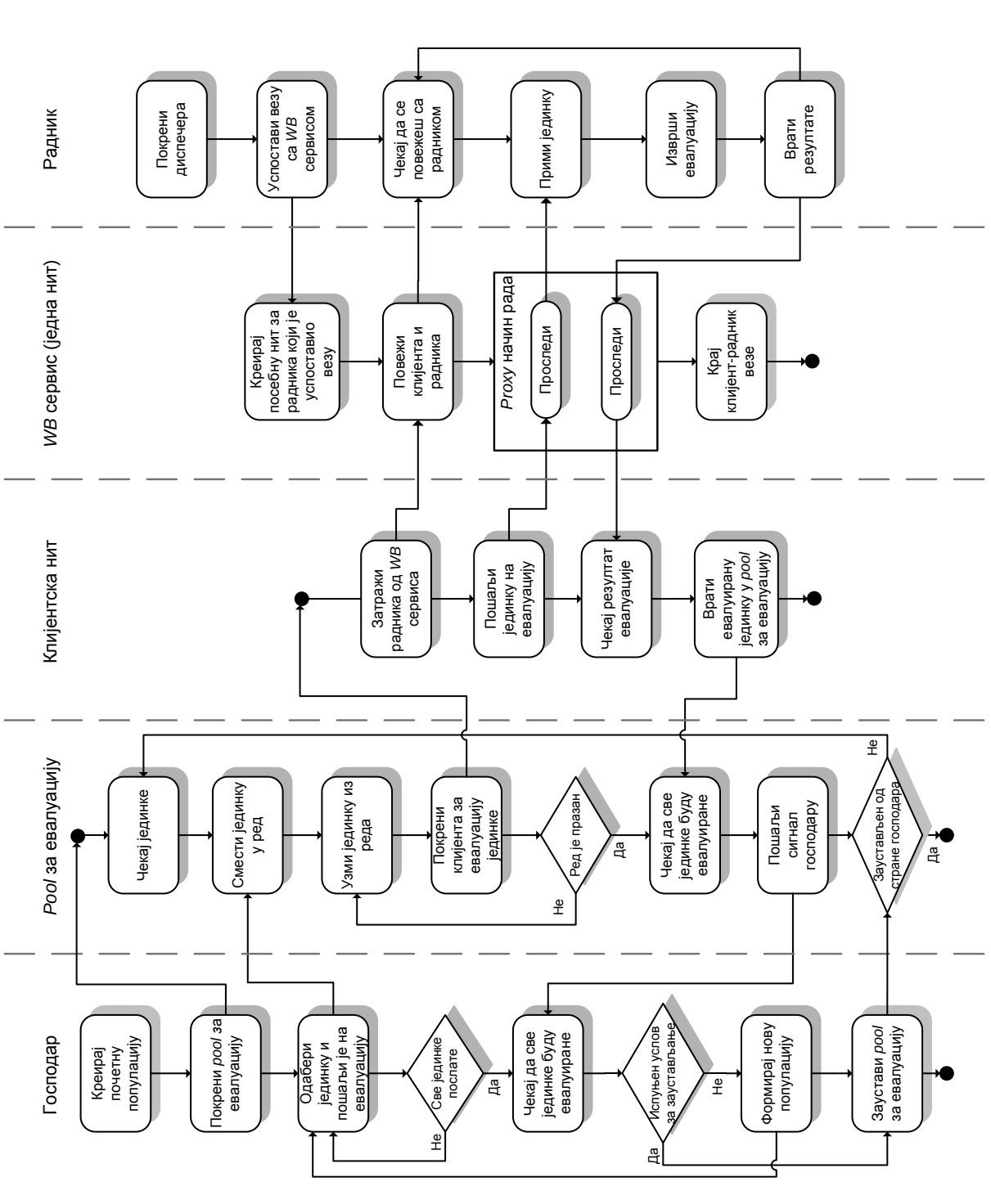
Сваки клијент, након што је покренут, покушава да успостави везу са *WB* сервисом. Када се веза успешно успостави, клијент шаље сервису своје идентификационе податке, док *WB* сервис за клијента проналази одговарајући посао, који се раније регистровао као спреман за обављање евалуације. Клијент и пронађени посао се упарују преко *WB* сервиса, који надаље има улогу проксија између њих, али и наставља да ослушкује нове захтеве. С обзиром на то да у *pool*-у за евалуацију постоји велики број решења која чекају на евалуацију, *WB* сервис, готово тренутно, прима стотине нових захтева и аутоматски *CE*-овима предаје на извршавање нове послове, како би брзо испунио ове захтеве.

Корак 7.

Када се клијент и радник споје, клијент преко *WB* проксија раднику шаље јединку коју треба оценити. С обзиром на то да је јединка састављена од одређеног броја гена, раднику се прво шаље број гена који сачињавају јединку, а потом вредности тих гена једна по једна. Радник затим покреће апликацију која врши евалуацију јединке. Пошто се код проблема ВКО јединка оцењује на основу више кријеријума, то се као резултат рада евалуационог софтвера добија неколико оцена. Зато, након што је евалуација завршена, евалуациони софтвер враћа раднику, а овај даље посредством *WB* проксија клијенту, прво број оцена, а потом вредности оцена једну по једну. Клијент резулатате евалуације враћа у *pool* за евалуацију, који их даље додељује одговарајућој јединки. Након што сви клијенти врате резултате и у реду за евалуацију нема више индивидуа, *pool* за евалуацију шаље главној нити сигнал да су све евалуације завршене и да господар може да настави са остатком алгоритма.

Након успешно обављене евалуације, радников диспечер ће бити рестартован како би се обезбедила поновна употреба истог посла на радном чвору. У случају да је по завршетку евалуације истекао предефенисани животни век посла, посао бива уклоњен. Тиме се осталим корисницима Грида, чији послови чекају на извршење пружа могућност да њихови послови дођу на ред.

Сви послови се предају на извршавање на Гриду коришћењем стандардног *Globus* проксија. Његово трајање од 12 до 24 сата најчешће није довољно да би се завршила оптимизација у случају релних проблема. Решење је у употреби *MyProxy* сервера. *MyProxy* сервер је заштићени репозиторијум на који се може похранити прокси чији је животни век дужи од стандардних 12 сати, и обично износи једну недељу. На основу дуготрајног прокси сертификата који се налази на *MyProxy* серверу, са њега се могу аутоматски у одређеним временским тренуцима преузимати краткорични прокси сертификати и то без употребе Грид лозинке.



Слика 4.10: Ток рада WoBinGO софтверског оквира (ради јасноће приказа кораци 1-4 су избачени)

Глава 5

Анализа перформанси развијених софтверских оквира за дистрибуирану евалуацију јединки

Циљ овог поглавља је да се свеобухватно процени и међусобно упореди ефикасност, ефективност, поузданост и једноставност употребе развијених софтверских оквира за вишекритеријумску оптимизацију засновану на ГА. У првом делу поглавља ће бити изложена теоријска разматрања перформанси развијених софтверских оквира, док ће у другом бити приказани резултати емпиријских истраживања у виду компаративне анализе перформанси ових оквира. Трећи део представља студију случаја управљања бранама у систему Ибарских каскадних хидроелектрана. Студија је изведена са циљем да се оцени утицај обимних операција писања и читања са диска, које карактеришу евалуцију јединки код овог реалног оптимизационог проблема, на перформансе софтверских оквира. На основу међусобног поређења перформанси и лакоће коришћења појединих оквира, на крају поглавља ће бити дате и препоруке за практични одабир једног од оквира у зависности од појединих карактеристика проблема са којим је корисник суочен.

5.1 Теоријска разматрања

Паралелизација генетских алгоритама се врши са циљем да се повећа њихова ефикасност и тако смањи време потребно за извршавање алгоритма. Мерење релативне добити коју остварујемо спровођењем паралелизације врши се метриком која се назива убрзање [4]. Ова метрика представља однос просечног времена потребног да се неки посао обави секвенцијално, тј. на једном процесору и просечног времена потребног да се тај исти посао обави расподелом на више процесора. Убрзање се

може израчунати коришћењем следеће формуле [85]:

$$S = \frac{T_{ser}}{T_{par}} \tag{5.1}$$

где T_{ser} означава време потребно да популација индивидуа еволуира серијски, док T_{par} означава време потребно да иста популација еволуира паралелно. За популацију величине n, време T_{ser} је:

$$T_{ser} = \lambda(n) + \gamma(n) \tag{5.2}$$

где $\lambda(n)$ означава време потребно за извршење дела алгоритма који се одвија секвенцијално, а $\gamma(n)$ означава време потребно за секвенцијално извршење оног дела алгоритма који се може паралелизовати. Секвенцијални део алгоритма обухвата генетске операторе селекције, укрштања и мутације које извршава господар, док се евалуација јединки врши паралелно од стране више слуга - радника. Ако је број процесора радника p, онда је време T_{par} дато са:

$$T_{par} = \lambda(n) + \frac{\gamma(n)}{p} + O(n,p) \tag{5.3}$$

где $O(n,p)$ означава укупне трошкове комуникације. Учесталост комуницирања делова паралелног система битно утиче на његове перформансе. Глобално паралелни ГА подразумева слање кроз мрежу свих јединки из генерације на евалуацију и потом враћање резултата од радника ка господару. Овако висок степен комуникације која се преко мреже одвија између радника и господара потенцијални је недостатак коришћеног модела паралелизације [35].

Ако је p број процесора који врше евалуацију n јединки, а t_{eval} просечно време потребно да се изврши евалуација једне јединке, онда се T_{par} може изразити на следећи начин:

$$T_{par} = \lambda(n) + \frac{nt_{eval}}{p} + O(n,p) \tag{5.4}$$

Израз $\lambda(n) + \frac{nt_{eval}}{p}$ ћемо означити са $T_{teo}(n,p)$, што представља теоријски идеално време у коме се паралелно на p процесора оцени генерација од n јединки.

Ако време потребно да се исти оптимизациони проблем реши серијски изразимо као:

$$T_{ser} = \lambda(n) + nt_{eval} \tag{5.5}$$

тада из једначина (5.4) и (5.5) произилази следећа једначина убрзања:

$$S = \frac{\lambda(n) + nt_{eval}}{\lambda(n) + \frac{nt_{eval}}{p} + O(n,p)}. \tag{5.6}$$

У складу са Амдаловим законом [85], теоријска граница максималног убрзања које се може достићи паралелним алгоритмом добија се ако претпоставимо да трошкови комуникације не постоје, тј. да је $O(n,p) = 0$, па једначина (5.6) постаје:

$$S_{max} = \frac{\lambda(n) + nt_{eval}}{\lambda(n) + \frac{nt_{eval}}{p}} \tag{5.7}$$

Да би убрзање уопште било постигнуто, следеће мора да важи:

$$\begin{aligned} T_{ser} &> T_{par} \\ nt_{eval} &> \frac{nt_{eval}}{p} + O(n,p) \\ t_{eval} &> \frac{p}{n(p-1)} \cdot O(n,p) \end{aligned} \tag{5.8}$$

Развијени оквири могу дакле, убрзати оптимизацију ако трошкови комуникације, број процесора, трајање једне евалуације и број јединки које се евалуирају задовољавају услов (5.8). За велики број процесора, једначина (5.8) може бити апроксимирана на следећи начин:

$$t_{eval} > \frac{1}{n} \cdot O(n,p). \tag{5.9}$$

За разлику од осталих описаних софтверских оквира, као и оквира за дистрибуирану оптимизацију који се срећу у литератури (попут [93], [77]), WoBinGO не одликује статичан број доступних радника, те се индивидуе које треба евалуирати не могу равномерно поделити између радника. У складу са политиком еластичног заузимања ресурса, број радника се код WoBinGO оквира током времена мења, те се отуда, претходно наведене једнакости и неједнакости у случају WoBinGO оквира разликују.

Ако је $\bar{p}$ просечни број процесора који врше евалуцију n индивидуа, онда се T_{par} за WoBinGO софтверски оквир може изразити на следећи начин:

$$T_{par} = \lambda(n) + \frac{nt_{eval}}{\bar{p}} + O(n,\bar{p}) \tag{5.10}$$

Отуда, код WoBinGO оквира да би убрзање уопште било постигнуто, мора да важи следеће:

$$t_{eval} > \frac{\bar{p}}{n(\bar{p}-1)} \cdot O(n,\bar{p}) \tag{5.11}$$

док је апроксимација једначине (5.11) за велики број процесора:

$$t_{eval} > \frac{1}{n} \cdot O(n,\bar{p}). \tag{5.12}$$

5.2 Емпиријски резултати

У овом делу поглавља ће бити изложени резултати емпиријске студије изведене у циљу одређивања: (1) убрзања процеса оптимизације, које се остварује коришћењем развијених софтверских оквира као и (2) комуникацијских трошкова који настају као последица начина рада ових оквира.

Имајући у виду коришћени модел паралелизације, са једном тачком синхронизације и то након евалуације свих јединки у једној генерацији, најважнији податак за

одређивање убрзања је време потребно да се оцени једна генерација паралелизацијом овог процеса на p процесора. Убрзање је израчунавано по раније наведеној формули (5.1).

Комуникацијски трошак је израчунаван као релативна вредност:

$$\mathcal{O}_r(n,p) = \frac{T_{par} - T_{teo(n,p)}}{T_{teo}(n,p)} \tag{5.13}$$

где је T_{par} експериментално утврђено време потребно да се оцени генерација од n јединки паралелизацијом овог процеса на p процесора, а $T_{teo}(n,p)$ је теоријски идеално време у коме се паралелно на p процесора оцени генерација од n јединки.

У експериментима је решаван вештачки тест проблем коришћењем једноставног ГА са реално кодираним хромозомима, симулираним бинарним укрштањем и полиномном мутацијом. С обзиром на то да је циљ експеримената био одређивање трошкова комуникације, питање да ли радници врше израчунавања или једноставно спавају, није било од важности за експеримент. Зато је за функцију циља је одабрана лажна функција која прихвата јединку коју треба да евалуира, спава t_{eval} секунди и враћа случајно одабрану оцену јединке. Употреба функције која спава, уместо праве функције евалуације обезбеђује да на измерену вредност комуникацијских трошкова утиче само брзина мрежног саобраћаја, јер трајање ове "лажне евалуације" не зависи од брзине процесора.

Експерименти су извршавани на две врсте паралелних архитекура:

1. Архитектури дељене меморије у виду симетричног мултипроцесора
2. Архитектури дистрибуиране меморије представљене *Beowulf* типом рачунарског кластера.

Симетрични мутипроцесор је поседовао 2 *AMD Opteron 6174* процесора (2.2GHz, сваки са по 12 језгара, што укупно чини 24 језгра) и 64GB RAM меморије. Кластер типа *Beowulf* се састојао од 14 чворова. Сваки чвор је представљао један *2.4GHz Intel Core2Quad Q6600* процесор са 8GB RAM меморије, што чинило укупно 56 процесорских јединица са 112GB RAM меморије. Чворови су функционисали под *Scientific Linux 5.8 x86 64* оперативним системом, док су се кластерски послови извршавали унутар *PBS Torque* окружења за пакетну обраду. С озиром да *MS .NET* окружење није било доступно на овој платформи, коришћена је замена у виду *.NET* окружења имплементираног у отвореном коду – *Mono v2.10.5*.

Специфичности развијених оквира природно су условиле избор архитектура на којима су тестиране перформансе појединих оквира. Као што се може уочити из табеле 5.1 вишенитни оквир је у складу са својом наменом да оцењивање индивидуа врши у посебним нитима на једном рачунару са великим бројем процесора, тестиран на симетричном мултипроцесору. Перформансе *push* и *pull* оквира су тестиране и на симетричном мултипроцесору и на кластер архитектури, док је WoBinGO сходно свом начину дистрибуције јединки тестиран само на кластеру.

	Симетрични мултипроцесор	Кластер
Вишенитни софтверски оквир	√	×
Push софтверски оквир	√	√
Pull софтверски оквир	√	√
WoBinGO	×	√

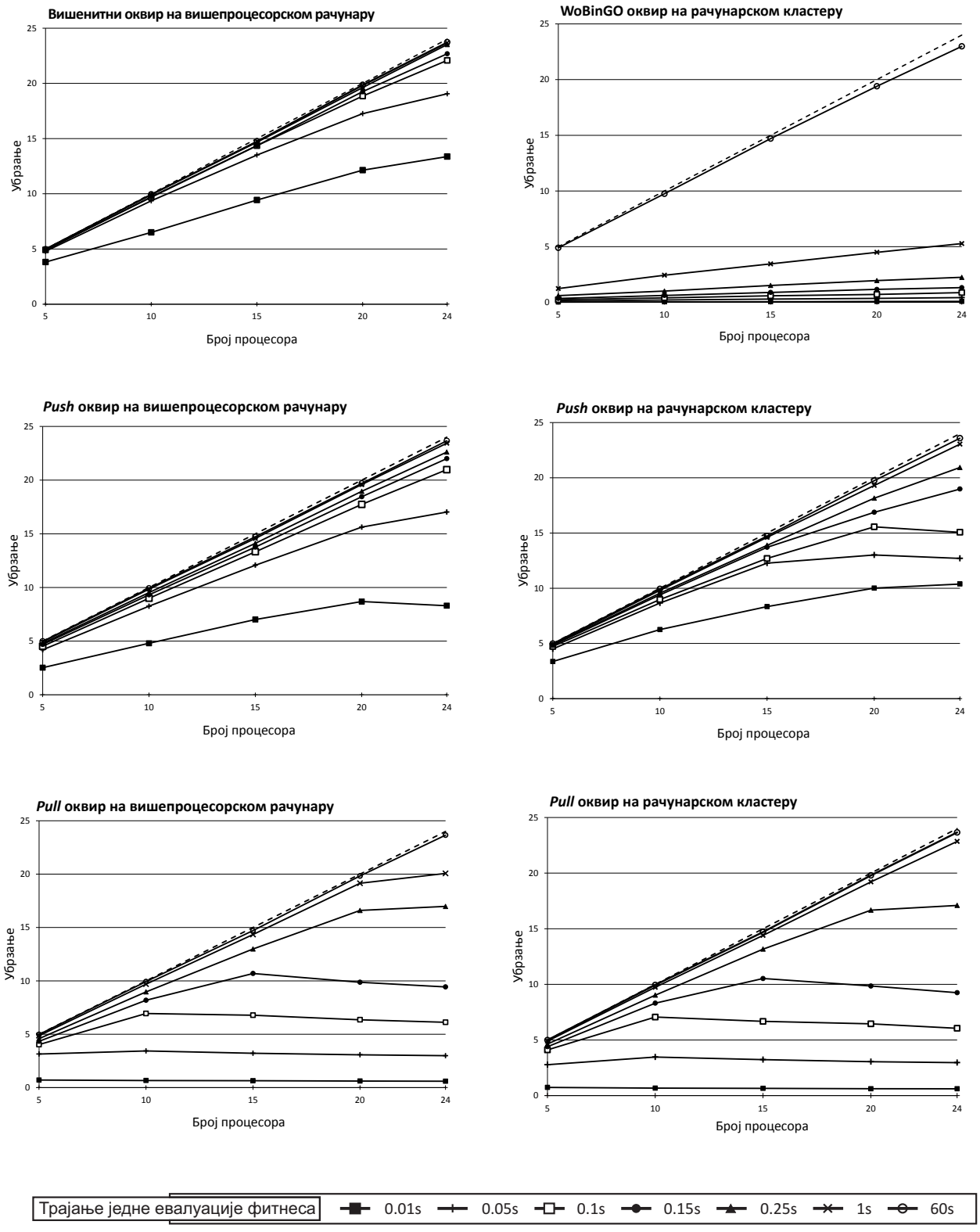
Табела 5.1: Табеларни приказ архитектура и оквира који су на њима тестирани

5.2.1 Анализа резултата добијених мерењем убрзања развијених софтверских оквира

Убрзања процеса оптимизације која се постижу његовом паралелизацијом коришћењем развијених овира приказана су на слици 5.1. Величина популације је током свих експеримената износила 500 јединки, број радника који су извршавали њихову евалуацију био је фиксиран на 24, једноставни ГА је извршаван кроз 10 генерација, а приказани резултати представљају просечну вредност из 10 поновљених експеримената. Одабир броја 24, као максималног броја радника који је коришћен у експериментима, условљен је бројем процесора на мултипроцесорском рачунару.

Еластично заузимање ресурса, које је главна одлика WoBinGO оквира и које условљава да се број упослених радника током времена мења, овде је избегнуто, ради упоредивости резултата добијених у експериментима са WoBinGO оквиром и осталим оквирима. Суспензија еластичног понашања врши се постављањем једне исте вредности као минималног (p_{min}) и максималног (p_{max}) броја радних чворова које *Work Binder* сервис сме да заузме на кластеру.

Оно што се прво може уочити при разматрању ових графикона јесте да је скалабилност *push* и *pull* оквира слична (на обе архитектуре), док се графикони вишенитног, а посебно WoBinGO оквира разликују од осталих. Код свих оквира, убрзање конвергира ка идеалном за временски захтевније евалуације (испрекидана линија на графиконима приказује теоретски идеално убрзање). За веома кратке евалуације уочава се разнолико понашање оквира, при чему вишенитни оквир даје најбоље резултате, јер су код њега трошкови комуникације најмањи. Код *push* и *pull* оквира, убрзање у некој тачки достиже максимум, и након тога почиње да опада. Разлог за овакво понашање лежи у трошковима комуникације, који се са порастом броја радника, увећавају, и за довољно кратке евалуације њихов утицај на успорење система превазилази позитивне ефекте паралелизације. Са повећањем трајања евалуације, преломна тачка на линијама које описују убрзање ових оквира се очекивано помера у десно. Евидентно је и споријe напредовање ефикасности *pull* оквира у односу на *push* оквир са порастом времена евалуације, које се јавља као последица *pull* начина добијања послова од стране радника. *Push* оквир се понаша боље на кластеру него на вишепроцесорском рачунару, услед ефеката трошкова комуникације (који ће бити додатно објашњени у наредном делу овог поглавља). Понашање *pull* оквира је готово идентично на обе архитектуре. WoBinGO оквир пружа убрзање процеса



Слика 5.1: Резултати мерења убрзања резвијених оквира

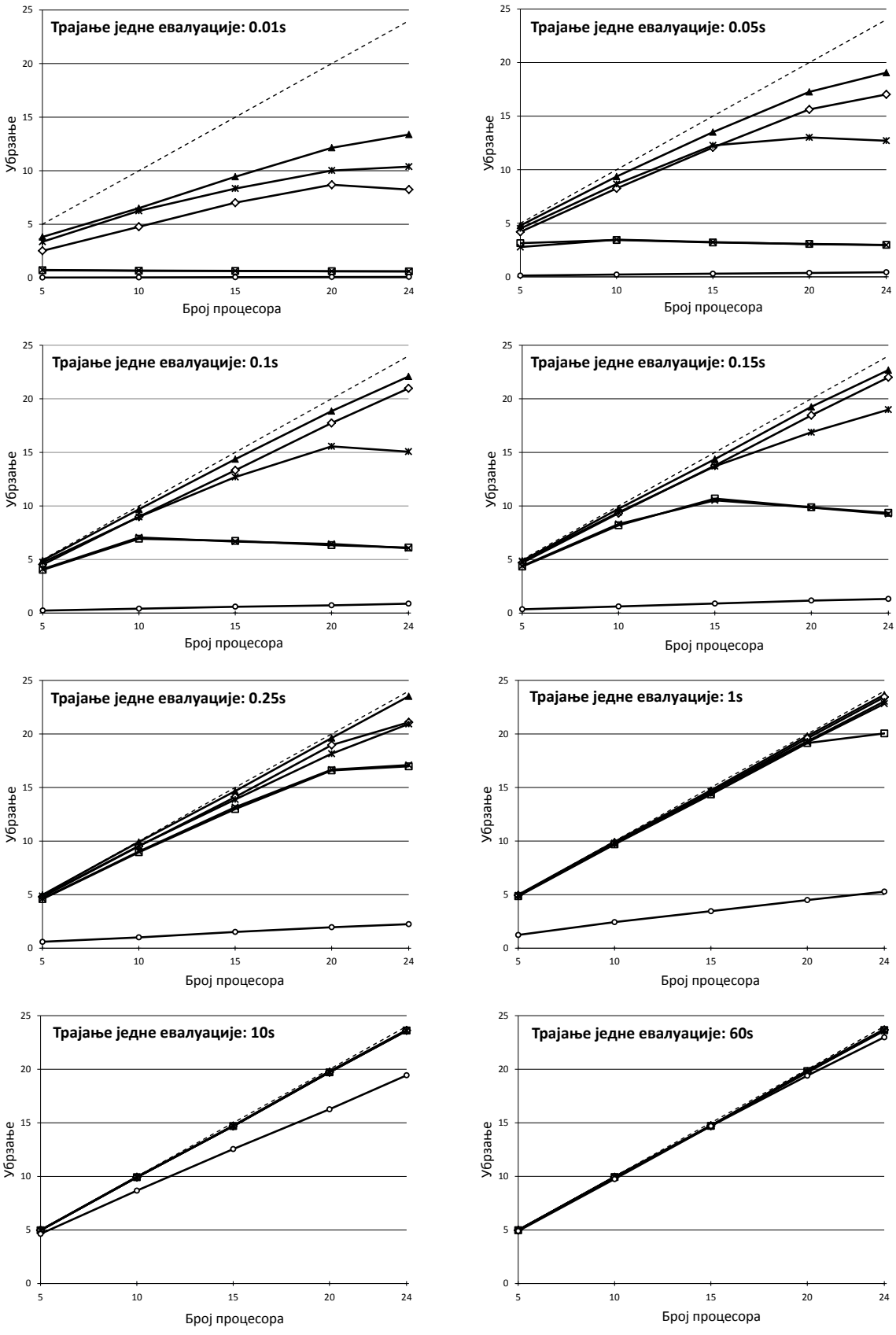
оптимизације тек за временски захтевније евалуације, што је и очекивано с обзиром на високе комуникацијске трошкове који се код овог оквира јављају за кратке евалуације.

На слици 5.2 је дат низ графикона, по један за свако трајање појединачне евалуације које је разматрано током експеримената. На сваком графикону је приказано убрзање сваког од развијених оквира. Овакав поглед на добијене резултате дат је са циљем да се утврди који софтверски оквир и на којој архитектури пружа највеће убрзање процеса оптимизације за различите дужине трајања евалуације. Може се запазити да линије убрзања различитих оквира са променом трајања евалуације мењају свој положај налик на полуотворену лепезу која се постепено склапа. Вишенитни оквир, у складу са најнижим трошковима комуникације, показује највеће убрзање. *Push* оквир на мултипроцесору са порастом трајања евалуације брзо достиже резултате вишенитног оквира, док је учинак *Push* оквира на кластеру, за већи број радника, нешто лошији. Као што је раније уочено, убрзање *pull* оквира не зависи од архитектуре, али оно спорије тежи идеалном убрзању од претходно разматраних оквира. WoBinGO за евалуације трајања испод 10s, има изразито лошије резултате од свих осталих оквира. Код евалуације које трају 10s он се приближава осталим окрвирима, који за ову дужину евалуације испољавају готово идентично убрзање, блиско идеалном. Коначно, у случају када оцењивање једне индивидуе траје 60s сви оквири пружају скоро идеално убрзање.

5.2.2 Анализа резултата добијених мерењем трошкова комуникације развијених софтверских оквира

Трошкови комуникације представљају највећи потенцијални недостатак господар-слуга модела паралелизације. Како приликом евалуације сваке генерације све јединке из популације морају бити послате радницима, и радници морају вратити резултате, постоји ризик да трошкови комуникације која се том приликом одвија, превазиђу границу исплативости. Отуда је у сврху испитивања перформанси развијених софтверских оквира веома важно експериментално утврдити трошкове комуникације истих.

Са циљем да се оцени степен утицаја комуникацијских трошкова на брзину рада развијених оквира анализирана је функционална зависност између комуникацијских трошкова и броја јединки у једној генерацији, као и између комуникацијских трошкова и трајања евалуације једне јединке. Дужина хромозома је кроз све експерименте била фиксирана на 10 *double* вредности (80 бајтова), а број ангажованих радника на 24. Еластично заузимање ресурса од стране WoBinGO оквира је, као и код експеримената везаних за мерење убрзања, било искључено. Величина популације је варирала и узмала вредности 100, 250, 500 и 1000, док су за трајање једне евалуације узимане вредности $0.01s, 0.1s, 1s, 10s$ или $60s$.



Слика 5.2: Упоредни приказ убрзања развијених оквира груписаних по трајању евалуације јединки

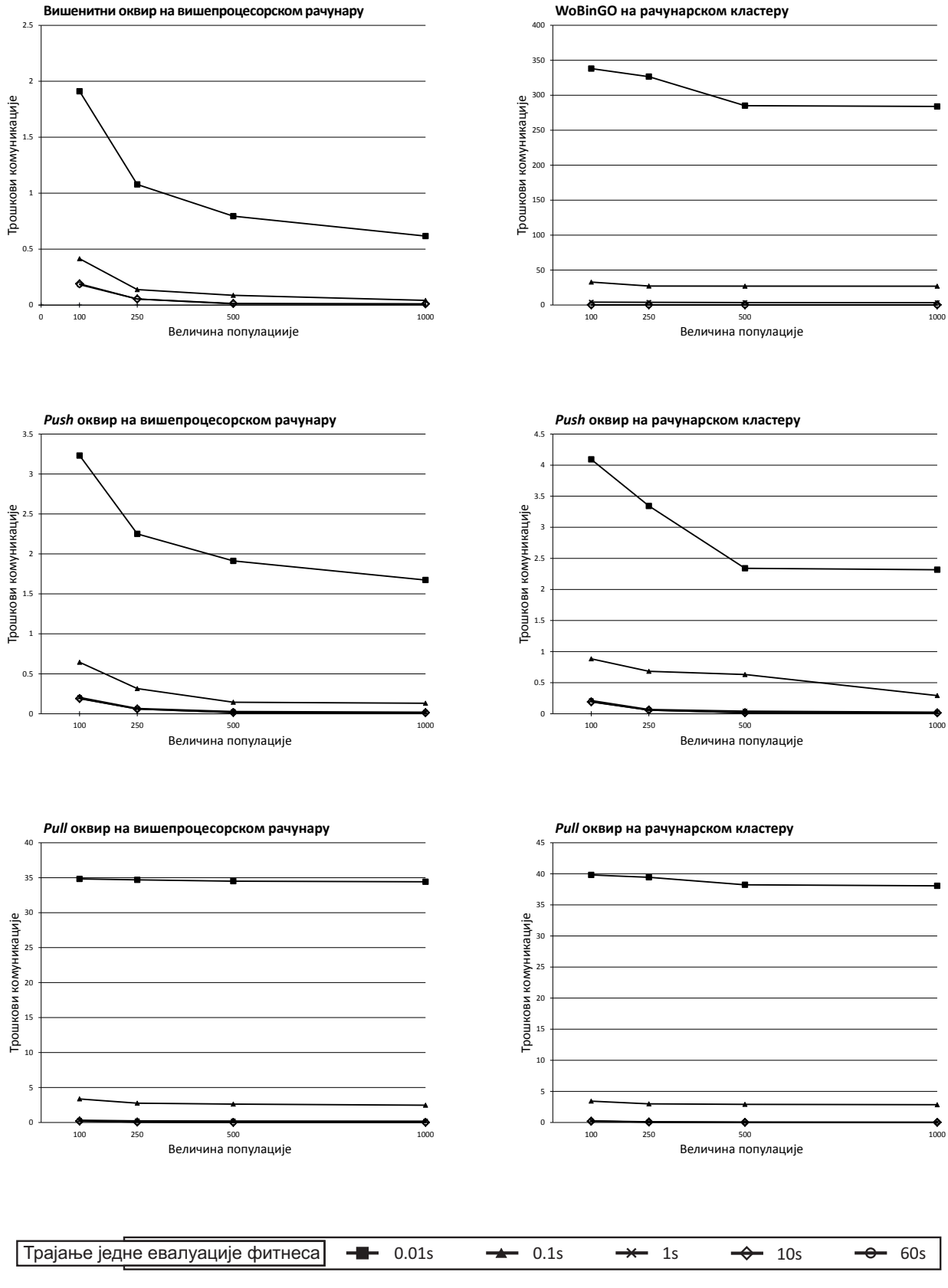
Експерименти су показали да на релативне трошкове комуникације највише утиче трајање евалуације једне индивидуе. Овакав резултат је условљен начином функционисања развијених оквира. Господар мора да чека да све индивидуе из једне генерације буду оцењене пре него што настави са извршавањем алгоритма, тако да релативни трошкови комуникације опадају са порастом трајања појединачне евалуације. Добијени резултати приказани су на слици 5.3. Слика садржи пет графикона на којима је дат упоредни приказ зависности релативних комуникацијских трошкова свих софтверских оквира од величине популације за пет различитих вредности трајања евалуације јединке. Оно што се прво може уочити са ових графикона, јесте да величина популације значајно утиче на трошкове комуникације само за временски захтевне функције евалуције. Ово се може објаснити чињеницом да је за кратке евалуције, учесталост синхронизације радника висока без обзира на величину популације, јер је и укупно трајање евалуције већих популација кратко. Код временски захтевнијих евалуација, са повећањем популације долази до значајног умањења учесталости синхронизације радника, а самим тим и до значајног пада релативних трошкова комуникације.

Релативни трошкови комуникације у случају *WoBinGO*-а знатно су виши него код осталих оквира, што је и очекивано с обзиром на његов начин рада. Поред комуникације на релацији клијент - *WB* сервис - радник која се одвија код свих оквира, овде постоји и додатна комуникација. Заправо, све док има неоцењених јединки у популацији, *pool* за евалуацију покреће клијенте који се обраћају *Work Binder* сервису са захтевом за евалуацију без обзира да ли слободних радника има или не. *WB* сервис одговара на ове захтеве тако што клијента повезује са радником. У случају да нема слободних радника, *WB* сервис клијента о томе обавештава одговарајућом поруком. Како је током експеримената број процесора који су обављали евалуацију био неколико пута мањи од величине популације, већина клијентских захтева се није могла одмах успешно испунити, те су их клијенти изнова и изнова упућивали *WB* сервису, што је условило висок комуникацијски трошак.

Поређењем измерених перформанси *push* и *pull* оквира, може се уочити да *pull* оквир даје лошије резулате за краће трајање евалуације, уз спорији пад релативних трошкова комуникације са повећањем величине популације. С друге стране, за довољно дугачке евалуације, линије релативних комуникацијских трошкова оба оквира, конвергирају нули.

Вишенитни софтверски оквир, очекивано, показује значајну предност спрам осталих оквира за краће трајање евалуације, јер је код њега присутан само додатни трошак који прави сам оперативни систем. Са порастом времена потребног да се оцени једна индивидуа, разлика у висини трошкова комуникације између овог и *push* и *pull* оквира постаје занемарљива.

Током раније дате теоријске анализе, одређен је услов (5.8) који треба да буде задовољен како би употребом развијених софтверских оквира било остварено убр-



Слика 5.3: Резултати мерења релативних трошкова комуникације развијених софтверских оквира

зање процеса оптимизације. Ако услов (5.8) напишемо у облику:

$$O(n,p) < \frac{nt_{eval}(p-1)}{p} \tag{5.14}$$

и обе стране неједнакости (5.14) поделимо са теоријски идеалним временом у коме се паралелно на p процесора оцени генерација од n јединки, $T_{teo}(n,p)$, добијамо следећу неједнакост:

$$\frac{O(n,p)}{\lambda(n) + \frac{nt_{eval}}{p}} < \frac{nt_{eval}(p-1)}{p\lambda(n) + nt_{eval}} \tag{5.15}$$

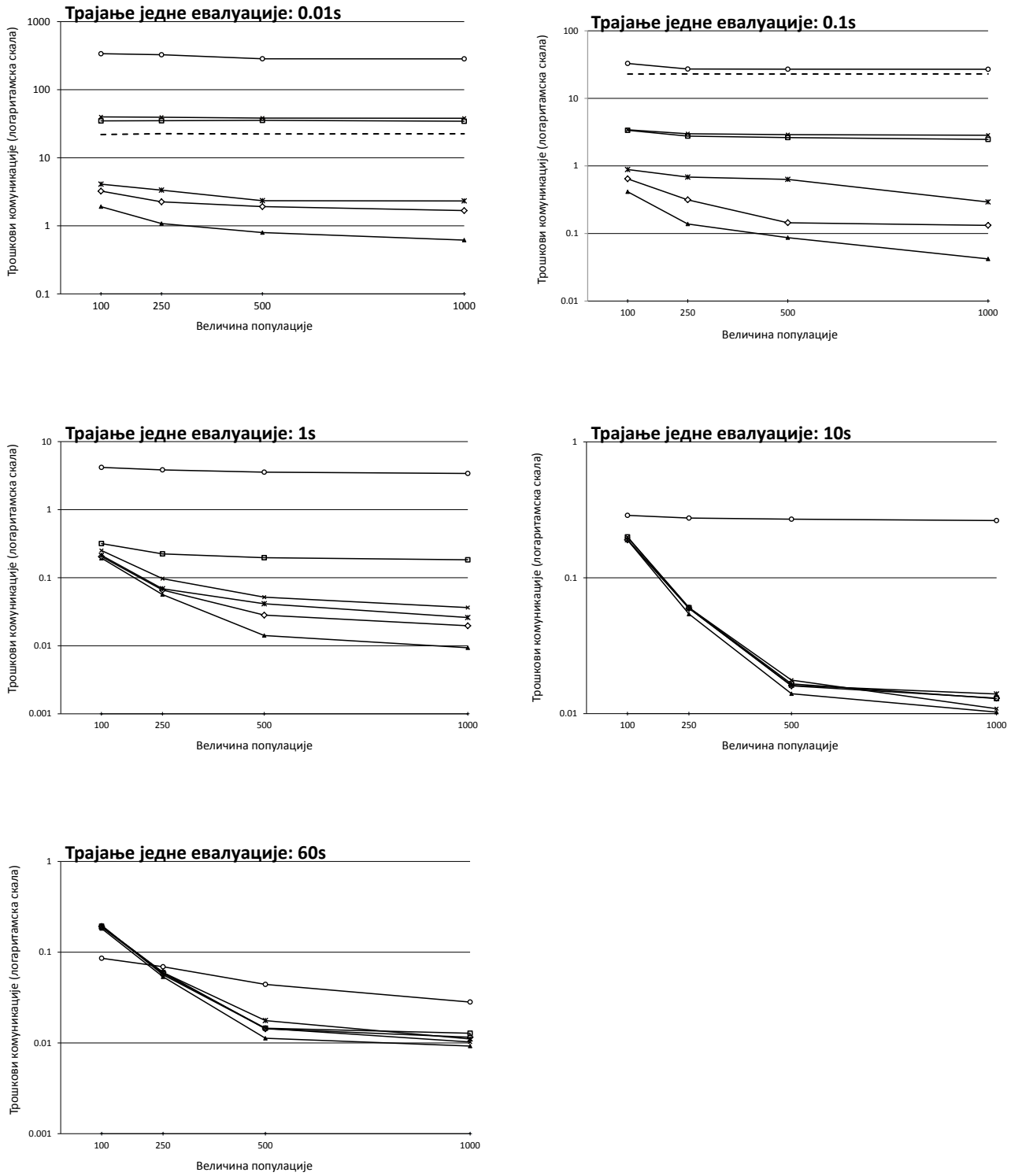
Лева страна неједнакости (5.15) представља релативне комуникационе трошкове, те се може написати у следећем облику:

$$O_r < \frac{nt_{eval}(p-1)}{p\lambda(n) + nt_{eval}} \tag{5.16}$$

Неједнакост (5.16) даје теоријску горњу границу коју релативни трошкови комуникације развијених софтверских оквира не би смели да премаше како би било остварено убрзање процеса оптимизације.

Трошкови комуникације који су добијени експериментално, упоређени су са теоријском горњом границом, која је за одређене вредности трајања евалуације, величине популације и броја процесора израчуната из једначине (5.16). Прва два графикона на слици 5.4 поред линија које приказују релативне трошкове комуникације појединих софтверских оквира, садрже и додатну, испрекидану линију. Ова линија представља израчунату горњу границу за релативне трошкове комуникације. За евалуације чије је трајање $0.01s$ може се уочити да је употреба вишенитног и *push* оквира исплатива, док паралелизација процеса оцењивања јединки коришћењем *pull* или WoBinGO оквира, код овако кратких евалуација, не доводи до његовог убрзања, независно од величине популације.

На другом графикону на слици 5.4, на основу положаја линија трошкова комуникације у односу на горњу границу може се закључити да у случају када евалуације трају $0.1s$ употреба вишенитног, *push* и *pull* оквира независно од величине популације доводи до убрзања оптимизације, док је употреба WoBinGO окрвира код овако кратких евалуација и даље неисплатива. Горња граница трошкова комуникације није исцртана на осталим графиконима, јер увелико превазилази трошкове комуникације свих оквира.



Слика 5.4: Упоредни приказ релативних комуникацијских трошкова развијених оквира груписаних по трајању евалуације степена прилагођености јединки

5.2.3 Анализа утицаја обимних операција писања и читања у процесу евалуације јединки на перформансе развијених оквира. Студија случаја оптималног управљања бранама у систему Ибарских каскадних хидроелектрана

Оптимизациони проблеми у хидроинформатици често подразумевају да оцењивање јединки садржи симулацију модела, при чему се опис модела и конфигурација симулације учитавају из датотека. Квалитет резултата добијених симулацијом модела се утврђује њиховим поређењем са осмотреним вредностима, које се такође учитавају из датотека. Поред читања датотека, честе су и евалуационе методе које захтевају обимно писање на диск или комбинацију ових улазно-излазних операција. Са циљем да се испитају перформансе и понашање развијених софтверских оквира приликом решавања оваквих реалних проблема, изведени су додатни експерименти, и то на примеру оптималног управљања бранама у систему Ибарских каскадних хидроелектрана. Пример је одабран тако да функција евалуације степена прилагођености јединки обилује операцијама писања и читања са чврстог диска, чије извршавање тече секвенцијално на фајл систему дељеном од стране више процесора и као такво деградира перформансе.

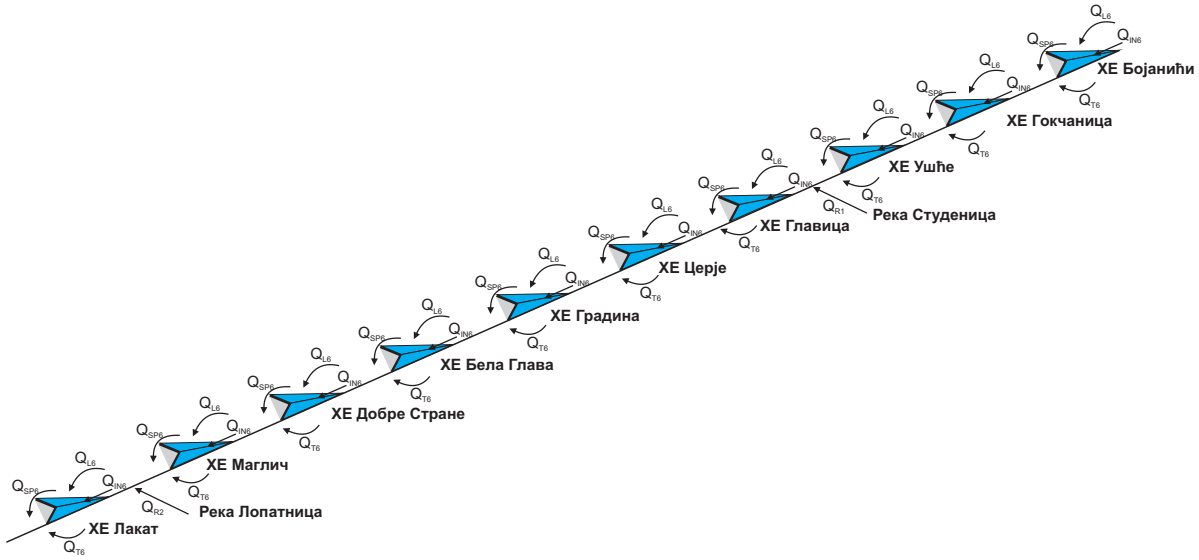
Формулација проблема

У складу са иницијативом за искоришћење хидроенергетског потенцијала реке Ибар на делу тока између Краљева и Рашке била је планирана изградња већег броја проточних ХЕ са малим падом, и у складу са тим израђен генерални пројекат и студија оправданости. Према пројекту је планиран каскадни систем од 10 хидроелектрана које раде у проточном режиму. Систем би чиниле следеће хидроелектране: ХЕ "Лакат", ХЕ "Маглич", ХЕ "Добре Стране", ХЕ "Бела Глава", ХЕ "Градина", ХЕ "Церје", ХЕ "Главица", ХЕ "Ушће", ХЕ "Гокчаница" и ХЕ "Бојанићи". Шематски приказ планираног система дат је на слици 5.5.

Улаз у систем су дотицаји на главном току и две притоке. Дотицај у k-ту хидроелектрану представља збир протицаја кроз турбине $Q_{T_{k-1}}$ и преливања $Q_{SP_{k-1}}$, као и међудотока $Q_{L_{k-1}}$ и протицаја са притоке (уколико постоји) Q_R. Дакле, управљање једном браном је уско повезано са радом узводних брана. За моделирање понашања брана у систему ибарских електрана искоришћен је пропорционално-интегрални регулатор са спрегом унапред (*feed-forward PI controler*), код кога је проток кроз брану дефинисан једначином:

$$Q(t) = k_p e(t) + k_i \int_0^t e(t)dt + Q_{FF}(t), \tag{5.17}$$

где је $e(t) = H_{GV}(t) - KNU$, при чему је $H_{GV}(t)$ кота горње воде, а KNU кота нормалног успора. k_p и k_i су пропорционални и интегрални коефицијент ПИ регулатора, а Q_{FF} просечни дотицај са узводне електране у претходних T_{FF} секунди. Примењени



Слика 5.5: Приказ планираног каскадног система од 10 хидроелектрана

пропорционално-интегрални регулатор тежи да регулацијом протока минимизује одступање коте горње воде од задате коте, тако што проток коригује пропорционално тренутном одступању, при чему је коефицијент пропорционалности k_p. Међутим, сувише мале вредности овог коефицијента доводе до слабог одзива на појаву грешке, док превелике вредности могу изазвати претеране реакције и нестабилност система. Из тог разлога је уведен и интегрални члан који појачава корекцију протока уколико временом пропорционални члан не успе да елиминише одступање, чиме се уноси додатна стабилност у систем. Члан спреге унапред $Q_{FF}(t)$ представља предвиђање поремећаја у реци и уколико би био идеалан, сам ПИ контролер би био беспотребан, јер би се унапред тачно могло одредити како одговорити на предвиђене поремећаје у циљу одржавања жељене коте. Најчешће је члан $Q_{FF}(t)$ нека врста нископропусног филтера, чиме се у обзир узима простирање таласа и временско кашњење. С обзиром на то да динамику простирања таласа није могуће адекватно моделирати овом врстом филтера, постојање ПИ контролера је неопходно, како би се кориговала предвиђања $Q_{FF}(t)$ члана и у локалу пратило одступање коте од задате. Члан $Q_{FF}(t)$ представља просечан дотицај у акумулацију. Када је у питању прва акумулација, $Q_{FF}(t)$ је дотицај реком у одређеном периоду пре посматраног тренутка. У случају осталих акумулација, $Q_{FF}(t)$ представља истицање на узводној брани, или истицање на узводној брани увећано за дотицај са притоке, и све то у одређеном периоду пре посматраног тренутка. На тај начин, знајући блиску историју рада узводне електране, као и брзину простирања њеног утицаја, свака електрана може приближно проценити потребан отицај како би се постигла жељена кота. С обзиром на то да овај члан даје само приближну процену утицаја узводне електране, друга два члана ПИ регулатора ће извршити корекцију евентуалног одступања коте, на раније описан начин.

Вредности коефицијената, k_p, k_i и T_{FF} сваке од 10 хидроелектрана одређиване су тако да модел што је могуће боље одржава коту акумулације на котама нормалног успора (KNU) за историјски забележене вредности дотицаја. Коришћен је једноставни ГА са реално кодираним хромозомима. Сваки хромозом се састојао од 30 гена, тако да су свака 3 узастопна гена представљала коефицијенте k_p, k_i и T_{FF} за једну електрану. Циљ оптимизације је био да сума разлика између израчунате коте горње воде и задате коте нормалног успора на свих 10 брана, буде минимална. Дакле, циљна функција коју је требало минимизовати имала је следећи облик:

$$F = \sum_{k=1}^{10} \int_0^t e(t)dt \tag{5.18}$$

где је $e(t) = H_{GV}(t) - KNU$, при чему је $H_{GV}(t)$ израчуната кота горње воде, а KNU задата кота нормалног успора.

Процедура естимације параметара

Процес естимације започиње формирањем почетне генерације параметара чије су вредности случајно изабране из задатих опсега. Оцењивање сваке јединке у генерацији тече на следећи начин:

1. Вредности параметара k_p, k_i и T_{FF}, за сваку од 10 брана, уписују се у одговарајућу датотеку, чиме се на диску формира 10 текстуалних датотека.

2. Даље се покреће конзолна апликација *WinFEQ* која вредности параметара из претходно формираних датотека ишчитава и додељује моделима управљања бранама. *WinFEQ* затим врши прорачун неустаљеног течења што резултује вредностима протицаја и нивоа воде на свим бранама. Добијене резултате апликација уписује у низ бинарних датотека.

3. Након завршетка рада апликације *WinFEQ* резултати се исчитавају из бинарних датотека и уписују у *.csv* датотеке, како би била омогућена евентуална анализа добијених вредности.

4. Израчунате коте горње воде се даље читају из формираних *.csv* датотека и користе за одређивање оцене јединке по формули 5.18. Добијена оцена се прослеђује алгоритму и употребљава се за рангирање јединки по њиховом степену прилагођености.

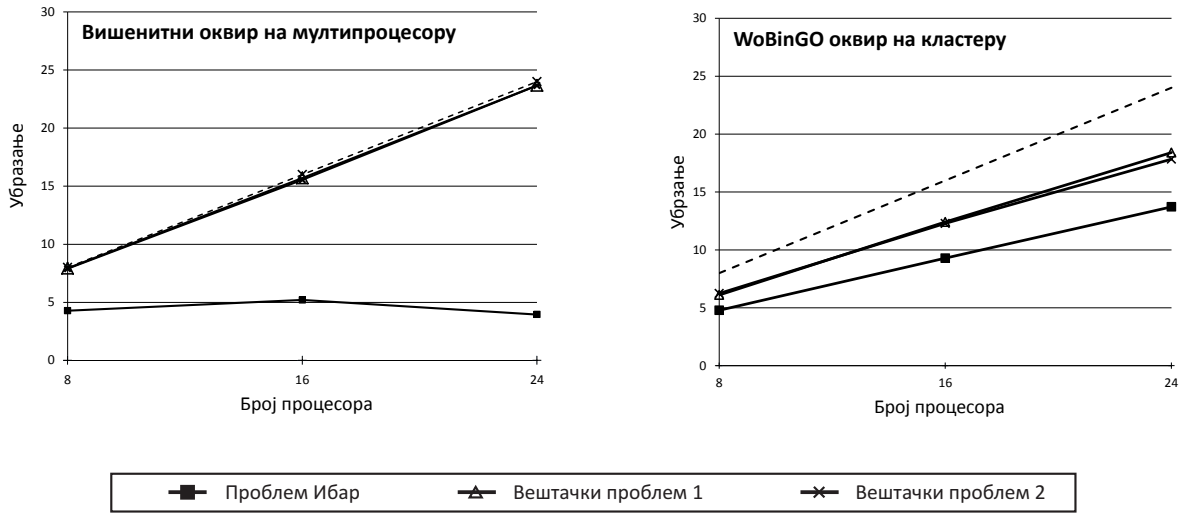
На основу оцена целе генерације параметара, алгоритам бира најбоља решења и њиховим укрштањем ствара нову генерацију параметара модела са којима се изнова улази у читав процес. Након достизања задовољавајућег поклапања израчунатих кота горње воде и задатих кота нормалног успора, параметри се проглашавају за најбоље могуће.

Експериментална подешавања

Резултати добијени у решавању примера на реци Ибар упоређени су са резултатима који се у идентичној експерименталној поставци добијају у решавању два вештачка проблема. Код првог вештачког проблема, евалуациона функција врши рекурзивно одређивање броја π, а код другог вештачког проблема евалуациона функција спава. Евалуациона функција код вештачких проблема извршава израчунавање, односно спава онолико дуго колико је времена просечно потребно евалуационој функцији из проблема Ибра да оцени јединку. Вештачки проблеми су одабрани са циљем да се упореде утицаји конкурентног приступа чврстом диску, конкурентног обрађања процеса оперативном систему и комуникационих трошкова на убрзање које се добија паралелизацијом. У случају проблема Ибра, на убрзање поред комуникацијских трошкова и конкурентности процесора, негативно утиче и њихов чест конкурентан приступ чврстом диску. Код првог вештачког проблема на успорење могу утицати комуникацијски трошкови и конкурентност процесора, док код другог вештачког проблема постоји само утицај комуникацијских трошкова.

Утицај операција читања са диска и писања на диск, на брзину рада апликације зависиће од архитектуре на којој се апликација извршава, то јест од начина на који радници приступају чврстом диску. Из тог разлога експерименти нису вршени употребом свих развијених софтверских оквира, већ на две различите архитектуре: вишепроцесорском рачунару и кластеру. Описи ових архитектура дати су у делу 5.2. Међу софтверским оквирима који се могу имплементирати на вишепроцесорском рачунару одабран је вишенитни софтверски оквир, који је током раније описаних тестова показао боље перформансе од свих осталих софтверских оквира на овој архитектури. С друге стране, на кластер архитектури је тестиран WoBinGO оквир, који због могућности еластичног резервисања ресурса, претендује да буде најчешће коришћен софтверски оквир за решавање конкретних проблема хидроинформатике.

Број радника који су коришћени за сваки од наведених проблема био је 8, 16 или 24. Разлог за одабир ове три вредности лежи у чињеници да кластер који је коришћен за тестирање поседује 3 радна чвора са по 16 *AMD* процесора који користе један диск. Да би резултати били валидни, неопходно је да број процесора који деле један диск буде костантан. Тако је кроз овај експеримент увек 8 процесора на кластеру делило један диск: 8 процесора са једног чвора, по 8 процесора са два чвора и по 8 процесора са три чвора. Максималан број процесора је поново 24, како би резултати добијени на кластеру били упоредиви са резултатима са вишепроцесорског рачунара. За решавање сва три проблема коришћен је једноставни ГА. Параметри алгоритма су имали следеће вредности: величина популације је била 500 јединки, максималан број генерација је био 100, коришћено је симулирано бинарно укрштање са вероватноћом од 0.9 и индексом расподеле 20, и полиномна мутација са вероватноћом $1/l$, где је l дужина хромозома. Сви резултати су добијени из 10 независних извршавања експеримента.



Слика 5.6: Убрзања вишенитног и WoBinGO оквира на вишепроцесорском рачунару и кластеру, респективно, добијена приликом решавања реалног проблема и два вештачка проблема

Резултати експеримента

Слика 5.6 приказује добијене резултате. Утицај конкуретног обраћања процесора оперативном систему на убрзање је незнатан, с обзиром на то да је убрзање за први и други вештачки проблем готово идентично.

Оба убрзања су у случају примене вишенитног оквира на вишепроцесорком рачунару готово идеална. Када је реч о WoBinGO оквиру убрзања нису тако близу идеалном, што је последица високих комуникационих трошкова. Ипак, када у разматрање узмемо реалан проблем Ибра који као и већина реалних проблема укључује обимно писање на/читање са чврстог диска, превагу остварује WoBinGO софтверски оквир који је тестиран на кластеру. Код вишепроцесорског рачунара, са порастом броја радника повећава се учесталост комуникације процесора са чвстим диском, те убрзање пада са повећањем броја радника. Са друге стране, кластер архитектура у овом експерименту, обезбеђује сталан број радника који деле један чврсти диск, тако да је скалабилност задржана.

Очекивано, услед конкурентног приступа диску, убрзање је доста даље од идеалног у односу на убрзања добијена решавањем вештачких проблема.

Дакле, у случају решавања проблема код којих евалуација прилагођености јединки захтева обимне улазно-излазне операције са чврстим диском, кластер архитектура код које увек исти број радика дели један чврсти диск, представља далеко бољи избор од вишепроцесорске архитектуре.

5.2.4 Смернице за оптимални избор софтверског оквира за вишекритеријумску оптимизацију

На основу резултата емпиријских студија изложених кроз ово поглавље већ је делимично могуће извести закључке о употребној вредности сваког од развијених софтверских оквира. Да би слика била потпунија, у овом делу поглавља ће за сваки од њих бити описан поступак припреме за отпочињање процеса оптимизације, као и графички приказ укупног трајања овог процеса на сваком од софтверских оквира у зависности од дужине евалуације једне јединке.

Пре отпочињања оптимизације коришћењем вишенитног оквира, на рачунар на коме ће се одвијати процес оптимизације треба икопирати све датотеке неопходне за евалуацију јединке, као и датотеке потребне за рад господара, и покренути господара.

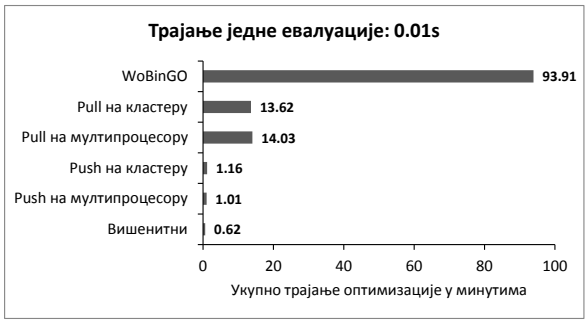
Код *push* софтверског оквира треба прво ископирати на кластер све датотеке неопходне за евалуацију јединке, утврдити колико је у датом тренутку слободних радних чворова доступно на кластеру и на кластеру покренути низ идентичних послова којих ће бити онолико колико има слободних ресурса. Покренути радници су заправо *WCF* веб сервиси чије се адресе уписују у одговарајућу листу активних радничких веб сервиса. Ову листу адреса даље треба ископирати у конфигурациону датотеку апликације која има улогу господара и на крају покренути самог господара.

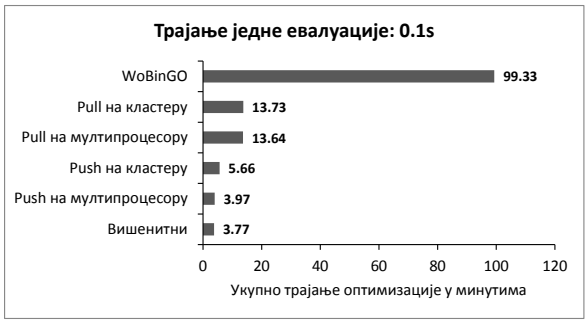
Ако се за дистрибуцију евалуација употребљава *pull* оквир, пре почетка рада треба ископирати на кластер све датотеке неопходне за евалуацију јединке. На рачунар на коме ће се извршавати менаџер треба ископирати датотеке потребне за рад менаџера и покренути менаџера. Даље треба утврдити колико је у датом тренутку слободних радних чворова доступно на кластеру и покренути низ идентичних послова, којих ће бити онолико колико има слободних ресурса. На крају треба покренути господара.

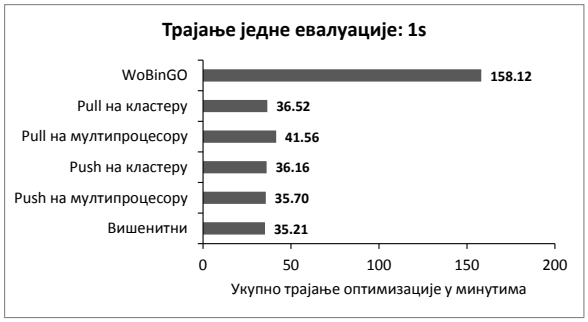
Као што је наведено у делу 4.6.3, пре почетка рада са WoBinGO оквиром корисник треба да обезбеди исправан Грид сертификат како би добио право да приступи Грид ресурсима и конфигурише одговарајуће датотеке које су неопходне за функционисање WoBinGO-а. Након тога је само потребно покренути господара.

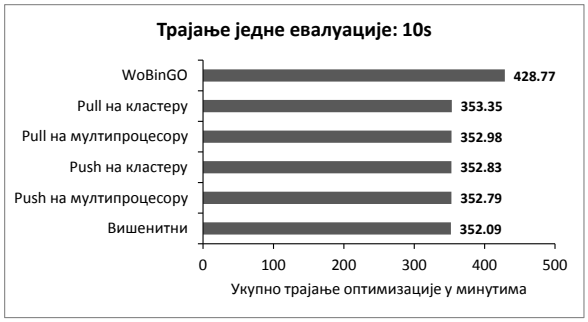
Из наведеног се може закључити да WoBinGO оквир изискује најмање припремних радњи. За њим следи вишенитни оквир, потом *pull* и на крају *push* софтверски оквир.

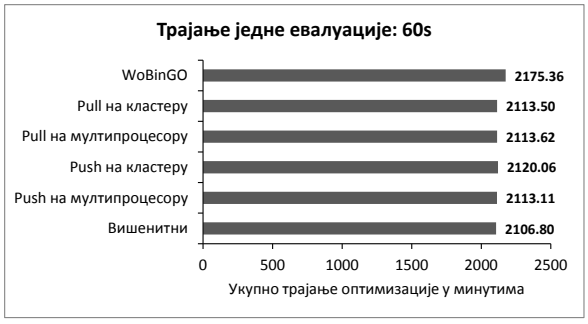
Коначно, укупно време трајања оптимизације на сваком од развијених софтверских оквира пружа интуитивнији увид у ефикасност сваког од представљених решења. На слици 5.7 је приказано укупно трајање процеса тестне оптимизације у минутима и то за различите дужине трајања евалуације јединки. Сви приказани резултати подразумевају величину популације од 500 јединки и оптимизацију која је извршавана кроз 100 генерација. Процес евалуације био је паралелизован на 24





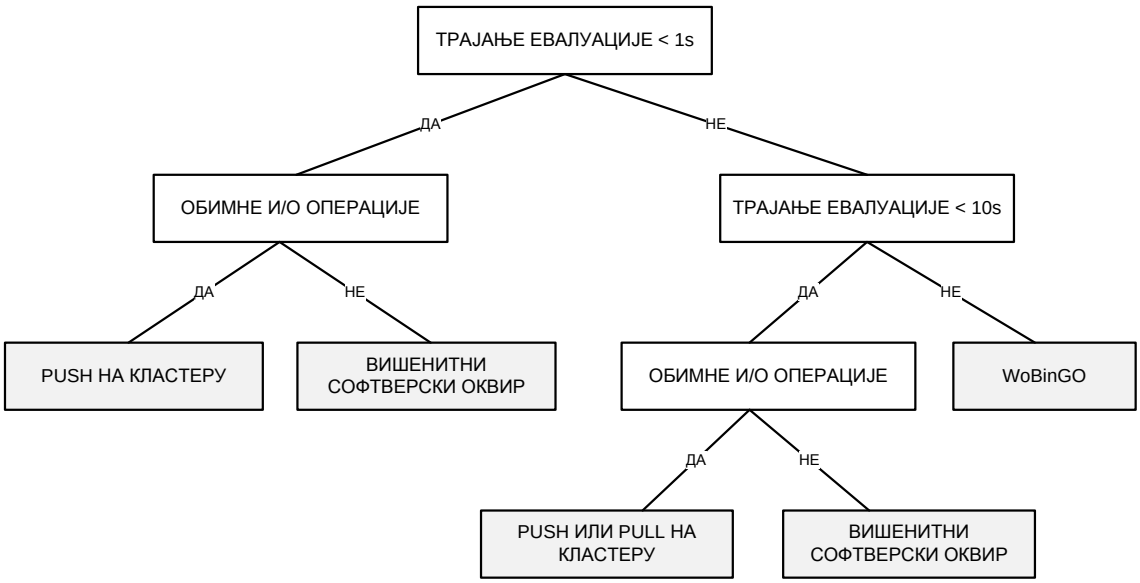






Слика 5.7: Упоредни приказ укупног трајања тестне оптимизације коришћењем развијених оквира

радника. За веома кратке евалуације од 0.01s Вишенитни оквир обезбеђује далеко најкраћи процес оптимизације који траје мање од минута, док за евалуације чије је трајање испод 1s вишенитни и *push* оквир дају сличне резултате на вишепроцесорској архитектури где ће оптимизација трајати приближно 4 минута. Укупно трајање процеса оптимизације коришћењем вишенитног, *push* или *pull* оквира готово је идентично за евалуације од 1s и 10s, и износи приближно 36 минута, односно приближно 6 часова, респективно. WoBinGO оквир је пак, далеко мање ефикасан од осталих развијених софтверских оквира у случају јако кратких евалуација на овако малом броју процесора. Са друге стране, за евалуације чије је трајање 10s, са временом извршавања оптимизације од приближно 7 часова, WoBinGO се значајно приближава временима осталих оквира. У случају евалуација од 60s процес оптимизације траје готово подједнако дуго на сва четири оквира, и износи у просеку приближно 35 часова.



Слика 5.8: Стабло одлучивања за одабир адекватног софтверског оквира за решавање оптимизационог проблема у звисности од дужине трајања евалуације јединке и обима операција писања на диск и читања са диска током процеса оцењивања

На основу свих резултата изложених у овом поглављу, може се извршити систематизација развијених софтверских оквира са циљем да будући корисници, у зависности од проблема који треба да решавају, могу једноставније да одаберу адекватан оквир. У ту сврху је на слици 5.8 приказано стабло одлучивања. За евалуације које трају мање од 1s најбрже и најједноставније се до решења долази коришћењем вишенитног оквира. Међутим, у светлу резултата добијених у студији случаја управљања бранама на Ибру, одлуку не можемо заснивати само на дужини трајања евалуације, већ морамо узети у обзир и обим операција читања и писања по тврдом диску. С обзиром на то да је показано да код вишепроцесорског рачунара, за евалуације које

захтевају обимне операције са тврдим диском убрзање пада са повећањем броја радника, за овај тип евалуација морамо се окренути кластерској архитектури, код које је скалабилност задржана, У овом случају, бољи избор је *push* оквир, јер он за евалуације краће од 1s даје најбоље резултате на кластеру, као што се може уочити са слика 5.2, 5.4 и 5.7. Ако је трајање једне евалуације дуже од 1s, а краће од 10s, поново је у случају да нема великих захтева за рад са чврстим диском, најбоље одабрати вишенитни оквир. Из горе наведених разлога, за евалуације које подразумевају писање и читање тврдог диска, адекватнији су *push* или *pull* оквир на кластеру, с обзиром на то да оба оквира испољавају сличне резултате за евалуације дужине од 1s до 10s (слике 5.2, 5.4, 5.7). За све дуже евалуације свакако је најбоље употребити WoBinGO софтверски оквир. Међутим, његова еластичност у резервисању дистрибуираних ресурса и ограничено трајање резервације ресурса представља велику компаративну предност у односу на друга предложена решења. Зато у случају да се оптимизациони проблем решава дистрибуцијом евалуација на чворове рачунарског кластера који је дељен између већег броја корисника, WoBinGO представља право решење чак и за евалуације које су нешто краће од 10s.

Глава 6

Перформансе WoBinGO софтверског оквира

С обзиром на то да од свих представљених софтверских оквира једино WoBinGO оквир представља комплетно решење за паралелизовану оптимизацију засновану на ГА уз еластично и економично коришћење Грид ресурса, посебно поглавље ове дисертације ће бити посвећено представљању резутата емпиријске студије за одређивање перформанси WoBinGO оквира.

Са WoBinGO софтверским оквиром, циљеви готово оптималног коришћења Грид инфраструктуре и високе доступности спремних послова постигнути су имплементацијом адаптивне политике када је у питању предавање послова на Грид, као и управљање *pool*-ом послова. Додатно, ови циљеви су остварени без непотребног жртвовања Грид ресурса, за разлику од ранијих решења која користе статичке инфраструктуре пилот послова. У овом делу рада ће емпиријски бити утврђено да ли еластичан приступ у коришћењу Грид ресурса утиче на перформансе система и у ком обиму, као и како ограничен животни век Грид послова утиче на време чекања Грид послова и пакетних (кластер) послова осталих корисника.

6.1 Емпиријски резултати

Први циљ спроведене емпиријске студије био је одређивање перформанси WoBinGO оквира у терминима искоришћења инфраструктуре, достигнутог убрзања и инхерентних трошкова комуникације, и то уз варирање времена t_{eval}. Други циљ је био да се одреди убрзање WoBinGO оквира када је t_{eval} константно. Трећи циљ је био да се изврши процена користи, коју остали корисници Грида имају захваљујући карактеристици WoBinGO оквира да ограничи трајање послова у свом *pool*-у.

Као и у експериментима описаним у делу 5.2, решаван је вештачки тест проблем употребом једноставног ГА са реално кодираним хромозомима. Функција циља

је поново била лажна функција, која не ради ништа осим што прими јединку, спава t_{eval} секунди и врати случајну вредност степена прилагођености јединке господару. Као што је и раније речено, овакав одабир функције циља је оправдан чињеницом да је сврха експеримената одређивање трошкова комуникације, те је једино битно време које је радницима потребно да врате свој одговор.

Тестови су извшени на једном Грид сајту како би били избегнути сви интермрежни ефекти и испитани раније објашњени теоријски лимити WoBinGO оквира. Грид сајт AEGIS04-KG се састојао од 6 чворова, од којих је сваки имао 2 *AMD 16-core* процесора и 96GB RAM, што је укупно чинило 192 процесора. Чворови су били повезани стандардним гигабитним Етернетом, док је оперативни систем на њима био *Scientific Linux 6.4 x86_64* са *PBS Torque* окружењем за пакетну обраду и *Maui* распоређивачем послова. Како је AEGIS04-KG део *EGI* инфраструктуре, послови нису предавани директном употребом *batching* система, већ коришћењем *EMI/UMD* сервиса, укључујући и *WMS (Workload Management System)*, који није колоциран са сајтом.

Величина популације је током свих експеримената износила 500 јединки. Једноставни ГА је извршаван кроз 10 генерација, а приказани резултати су просечне вредности настале из 10 поновљених експеримената. Процењено просечно време потребно за извршење секвенцијалног дела алгоритма (означено са $\lambda(n)$ у једначини (5.10)) је 200 ms.

Добијени резултати су приказани у табели 6.1. Просечан број процесора који су вршили евалуацију јединки $\bar{p}$ одређен је експериментално. WoBinGO је био конфигурисан тако да је максимални дозвољени број послова (p_{max}) био 100. Ипак $\bar{p}$ је варирало у складу са временом t_{eval} потребним да се изврши евалуација једне индивидуе. Код евалуација које краће трају, радници брзо постају поново употребљиви тако да ретко постоји потреба да се нови послови предају на извршење. Са друге стране, код евалуација које трају дуже прође доста времена пре него што радници постану поново употребљиви, па се нови послови чешће шаљу на извршење. Као последица овога, $\bar{p}$ има веће вредности код дужих евалуација. T_{ser} је време потребно да се једна популација оцени на једном процесору, док је T_{par} време потребно да се иста популација оцени паралелно на $\bar{p}$ процесора. T_{ser}/T_{par} је убрзање које се добија WoBinGO оквиром, а $O(n, \bar{p})$ су трошкови комуникације. Вредност $O(n, \bar{p})$ је одређена заменом T_{par}, $\lambda(n)$, t_{eval} и $\bar{p}$ у једначини (5.10).

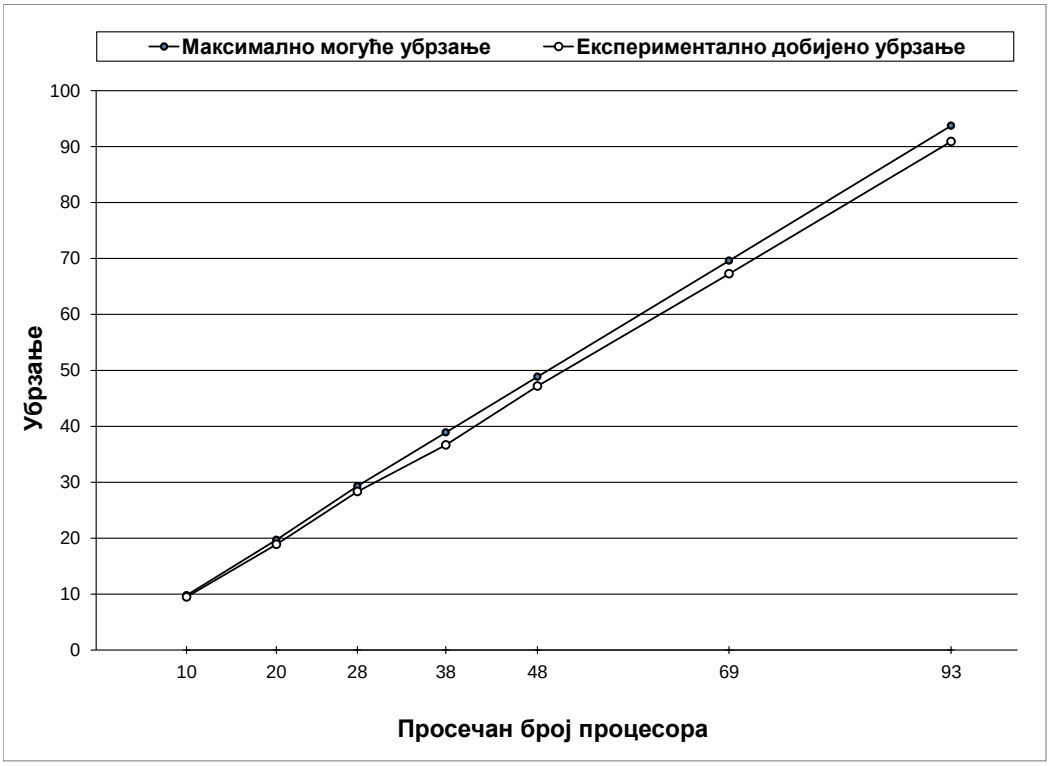
Код WoBinGO софтверског оквира, укупни трошкови комуникације се састоје од трошкова комуникације која се између клијената и радника одвија преко *WB* сервиса, и која је потребна да би се успоставила њихова веза и размењивали параметри и резултати, као и јединствених WoBinGO комуникацијских трошкова. WoBinGO за сваку неевалуирану јединку креира посебног клијента који се непрекидно обраћа *WB* сервису са захтевом за спреман посао, све док га не добије. Непрекидно обраћање клијената *WB* сервису ствара додатне комуникацијске трошкове. Овако настали комуникацијски трошкови су у великој мери умањени тиме што *WB* сервис настоји да

t_{eval}	$\bar{p}$	T_{ser}	T_{par}	T_{ser}/T_{par}	$O(n,\bar{p})$	$O(n,\bar{p})/T_{par}[\%]$
0.5	25	2.5E+03	4.13E+02	6.05	3.13E+02	76
1	49	5.0E+03	4.09E+02	12.23	3.07E+02	75
2	48	1.0E+04	4.37E+02	22.86	2.29E+02	52
5	74	2.5E+04	5.17E+02	48.4	1.78E+02	35
10	80	5.0E+04	7.40E+02	67.52	1.15E+02	16
30	91	1.5E+05	1.76E+03	85.21	1.12E+02	6
60	95	3.0E+05	3.25E+03	92.22	9.51E+01	3
120	96	6.0E+05	6.35E+03	94.55	1.17E+02	2

Табела 6.1: Емпиријски резултати добијени решавањем вештачког проблема помоћу једноставног ГА преко WoBinGO софтверског оквира (време је дато у секундама).

у *pool*-у увек има довољно спремних послова, али још увек постоје. Као што се може уочити из табеле 6.1, код проблема са краћим евалуацијама трошкови комуникације имају велики удео у времену које се утроши за паралелну евалуацију јединки. Ово се може објаснити тиме што је код проблема са краћим трајањем евалуација фреквенца захтева које клијенти и радници упућују *WB* сервису далеко већа него код проблема са дужим евалуацијама. Виша фреквенца имплицира дуже чекање да захтеви буду испуњени. Коришћењем WoBinGO оквира се за проблеме са дуготрајним евалуацијама могу остварити значајна убрзања, јер комуникацијски трошкови постају готово константни за $t_{eval} > 5s$.

Отуда је WoBinGO оквир погоднији за решавање проблема дуготрајним евалуацијама јединки. Како би овај резултат био додатно потврђен, вршени су експерименти са вештачким тест проблемом где је $t_{eval} = 60s$, а p_{max} узима следеће вредности: 10, 20, 30, 40, 50, 75 и 100. На слици 6.1 су приказане две криве убрзања: (1) крива максималног могућег убрзања и (2) крива убрзања која је добијена као резултат експеримената. Прва крива је добијена из једначине (5.7) заменом следећих параметара: $n = 5000(= 500 \cdot 10)$, $t_{eval} = 60s$, $\lambda(n) = 200ms$ и $\bar{p}$ које је одређено експериментално. Друга крива представља просечно убрзање добијено из 10 поновљених експеримената. Као што се може уочити са слике 6.1, теоријски претпостављено и експериментално добијено убрзање се готово поклапају. Разлика која међу њима ипак постоји потиче од комуникационих трошкова који су искључени из једначине (5.7). Као што је очекивано, трошкови комуникације расту линеарно са повећањем броја радника. Експериментално добијена крива убрзања још једном показује да је



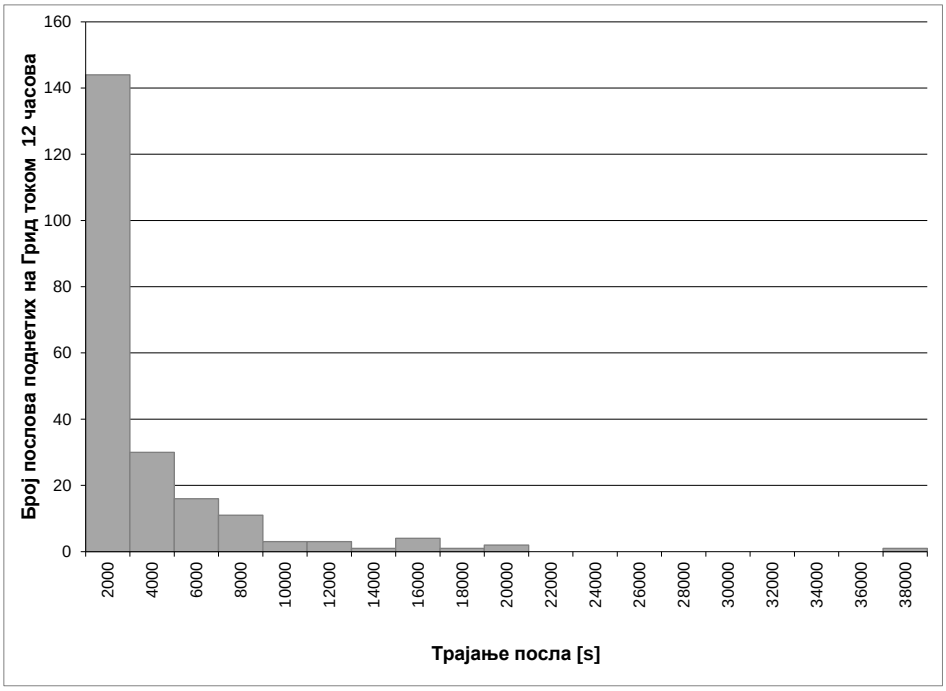
Слика 6.1: Убрзање добијено решавањем вештачког оптимизационог проблема WoBinGO-ом

WoBinGO оквир погодан за проблеме са рачунски скупим евалуацијама које се често срећу у реалним проблемима. Осим тога, ако се у обзир узму резултати приказани у делу 5.2.2, експериментално је утврђено да нема сврхе користити WoBinGO оквир у случајевима када је $t_{eval} < 0.1s$.

Ако се у WoBinGO оквиру на највишем нивоу уместо господар-слуга модела употреби хијерархијски паралелни ГА са господар-слуга демама или *PEGA*, убрзање које је приказано на слици 6.1 ће важити и даље у оквиру експерименталне грешке. Ово долази отуда што ће једна инстанца *WB*-а опслуживати захтеве за евалуацијама свих дема, док се трајање остатка ГА може занемарити с обзиром на дуготрајне евалуације.

Последњи циљ ове емпиријске студије било је тестирање оне одлике WoBinGO оквира која га разликује од претходних статичких решења базираних на пилот пословима - ограничени животни век послова у *pool*-у. Ова специфична карактеристика обезбеђује осталим корисницима Грида и директним корисницима кластера који шаљу пакетне послове да њихови послови који су током рада WoBinGO оквира предати на извршење краће чекају да дођу на ред. Како би се проценило колико

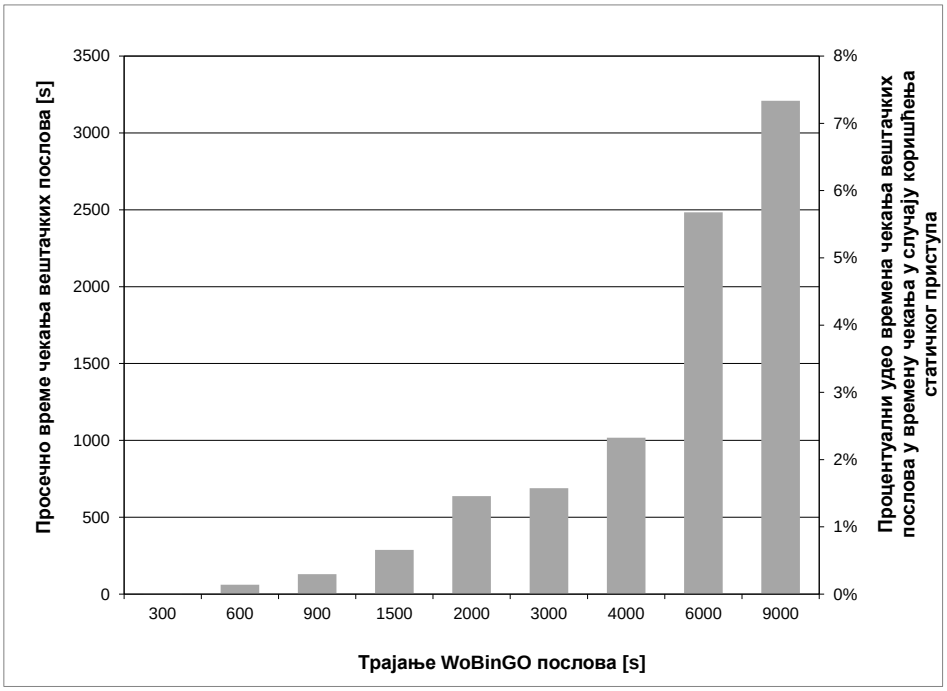
дуго ови послови чекају на ред за извршење, коришћен је вештачки тест проблем, као и раније. Време t_{eval} је поново фиксирано на $60s$, величина популације на 500 јединки, а максималан број процесора је био 150. Осим тога, направљено је вештачко оптерећење Грида од 18 послова по сату, при чему је динамика предаје послова пратила Поасонову расподелу. Просечан број процесора који је заузиман од стране ових вештачки генерисаних послова је био 1.3, при чему је однос једнопроцесорских и четворопроцесорских послова био 9:1. Просечно трајање ових послова износило је 44 минута у складу са Гама расподелом чији је параметар облика 0.344, а параметар скалирања 7680 (слика 6.2).



Слика 6.2: Расподела дужине трајања вештачких послова предатих на извршење

Све наведене вредности су узете као апроксимација реалног оптерећења чвора AEGIS04-KG током двогодишњег периода. Додатно, AEGIS04-KG распоређивач послова је био конфигурисан тако да сума процесора које користи WoBinGO и процесора које користе вештачки послови буде 150. Мерења су вршена са циљем да се утврди како просечно време чекања вештачки предатог посла зависи од дужине животног века WoBinGO посла. Сви резултати су добијени експериментом чије је трајање било 12 часова. Добијени резултати приказани су на слици 6.3. Може се приметити да за краћи животни век WoBinGO послова, вештачки послови који су предати на извршење брзо почињу са извршавањем. Просечно време чекања расте са порастом животног века WoBinGO послова, али не прелази један сат, чак ни када животни век WoBinGO послова траје два и по сата. Имајући у виду чињеницу да би са статич-

ким инфраструктурама заснованим на пилот пословима, послови вештачки предати на извршење свакако морали да чекају у реду све до краја оптимизације (што може бити 12 часова и више) приказани резултати су изузетно значајни. На секундарној ординати је приказано који проценат времена послови вештачки предати на извршење проведу у реду када се оптимизација врши WoBinGO оквиром, од времена који би у реду провели када би се оптимизација вршила коришћењем софтверског оквира са нееластичним приступом. Чак и за WoBinGO послове са јако дугим животним веком, време чекања послова вештачки предатих на извршење представља мање од 8 % времена које би они провели у реду у случају коришћења статичке инфраструктуре уместо WoBinGO оквира.



Слика 6.3: Просечно време чекања вештачких послова предатих на извршење

Глава 7

Анализа ефикасности еластичног резервисања ресурса коришћењем WoBinGO софтверског оквира. Студија случаја калибрације модела процуривања на хидроелектрани Вишеград

До сада је у дисертацији приказана емпиријска студија извршена са циљем да се одреде перформансе WoBinGO оквира и процени корист коју остали корисници Грида добијају захваљујући ограниченом трајању послова у *pool*-у WoBinGO оквира. Резултати ове студије су важни због бољег разумевања начина функционисања WoBinGO оквира и његовог потенцијала када је у питању решавање оптимизационих проблема са дуготрајним евалуацијама појединачних решења. Ипак, крајњи циљ развоја WoBinGO-а је свакако решавање реалних оптимизационих проблема. Зато ће у овом поглављу бити приказан случај употребе WoBinGO оквира у проналажењу решења за комплексан инжењерски проблем. У питању је калибрација сложеног модела процуривања на хидроелектрани "Вишеград" (Република Српска, Босна и Херцеговина).

Хидроелектрана "Вишеград" налази се на реци Дрини, узводно од града Вишеграда и већ у току прве године њене експлоатације, низводно од бране је уочена појава променљивог броја извора. Они су указивали на постојање подземних карстних пукотина кроз које је вода понирала, проузрокујући губитке из акумулације, који су се временом повећавали и са првобитних 1,4 m^3/s (1990. год.) досегли 14,68 m^3/s (децембар 2012. год.). Даље интезивирање ове појаве могло је проузроковати разне штетне последице, па је било неопходно извршити санацију процуривања уграђивањем гранулисаног материјала у подземне шупљине. Међутим, извођење заптивних

санационих радова је јако сложен процес, који мора бити праћен и контролисан током целог трајања. Како би се у току извођења санације доносиле исправне одлуке везане за извршење овог процеса, успостављен је систем за подршку одлучивању, који готово у реалном времену процењује успешност реализације технолошких поступака и доносиоцима одлука на терену помаже у планирању будућих активности. У циљу реализације оваквог система за подршку одлучивању било је потребно направити одговарајући математички модел. Основни задатак истраживања на математичком моделу подземних токова био је да се коришћењем осматрања добијених преко мониторинг система, у континуитету врши процена просторног распореда главних карстних проводника, њихових физичких карактеристика, као и хидрауличких величина у систему и тако добију информације значајне за доношење одлука о оптималним параметрима уградње у поступку санације провирања. Такође, након завршетка санације, математичким моделом је извршена свеобухватна естимација ефеката уградње гранулисаног материјала.

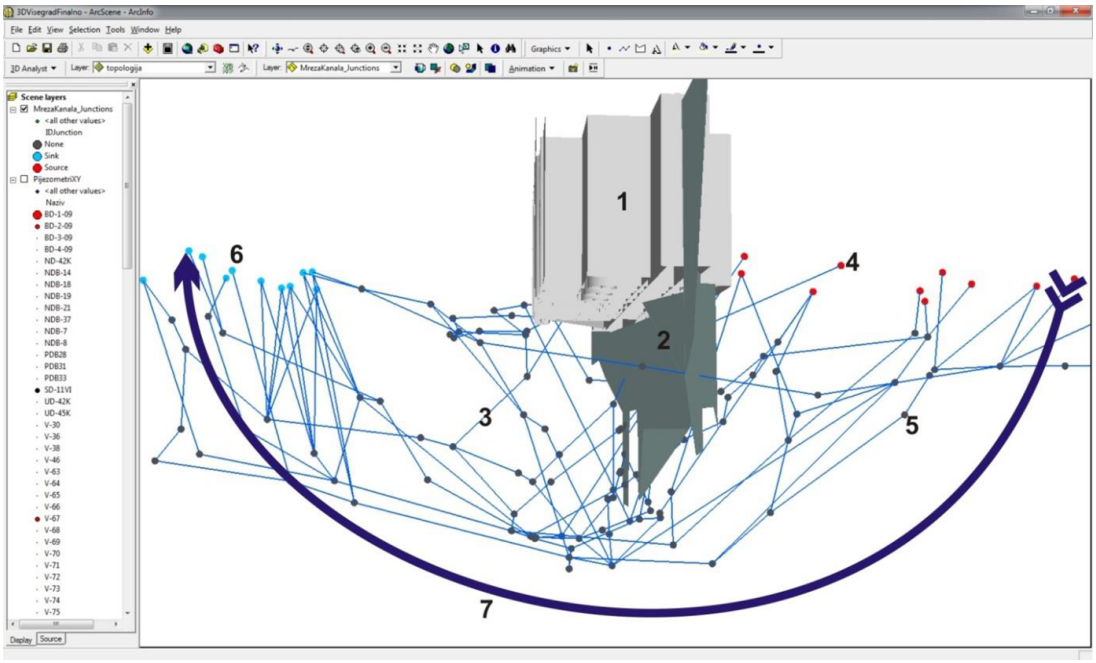
У даљем тексту ће прво бити описан коришћени математички модел. Потом ће калибрација параметара модела бити формулисана као проблем ВКО и дат преглед непознатих параметара и циљних функција као и сама процедура калибрације. У наставку ће бити речи о софтверском систему за подршку одлучивању у процесу санације и биће представљени резултати добијени калибрисаним математичким моделом. Коначно, биће представљени резултати експеримената везаних за употребу WoBinGO оквира у решавању овог комплексног реалног проблема.

7.1 Опис математичког модела

Проблем провирања испод бране ХЕ "Вишеград" укључује неколико различитих физичких процеса који су међусобно спрегнути до те мере да их није могуће посматрати независно. Отуда је и математички модел развијен као скуп компоненти које симулирају одређени осмотрени феномен, при чему су све симулације међусобно спрегнуте. Овај сложени мулти-модел чине следећа три модела:

1. Хидраулички модел подземног течења.
2. Модел транспорта растворљивих материја.
3. Модел транспорта и седиментације гаранулисаног материјала.

Иницијални хидраулички модел подземног течења је формиран на основу претходно изведених геолошких истраживања. Коришћењем *ArcGIS* програмског пакета формиран је 3D геолошки модел (слика 7.1) у виду просторне мреже потенцијалног пружања карстних канала која је послужила као основа за даље математичко моделирање подземног карстног развића.



Слика 7.1: 3D мрежа потенцијалног пружања карстних канала: 1. брана, 2. инјекциона завеса, 3. претпостављено пружање карстног канала - веза, 4. чвор - зона понирања, 5. чвор у мрежи - рачвање веза, 6. зона дренирања - извори, 7. генералан правац сифонског залегања карстних канала испод бране ХЕ Вишеград

Овако добијена топологија подземних токова је у моделу представљена као скуп водопропусних канала, док су везе међу њима представљене чворовима. Гранични услови модела су коте горње и доње воде, које су задате као познати потенцијали у чворовима који су у контакту са акумулацијом, односно зоном извирања. Као резултат хидрауличког прорачуна над задатим моделом система добијају се потенцијали (пијезометарски нивои) у свим чворовима система, као и брзине и протоци кроз све елементе система. Симулација течења кроз формирани модел при задатим граничним условима вршена је методом коначних елемената.

С обзиром на то да је сам хидраулички модел стационаран и не даје временску димензију течења, ради утврђивања динамике течења у систему, извршен је одређени број експеримената простирања трасера, који су убацивани у неке од тачака система. Развијен је и одговарајући математички модел, који би на основу извршеног хидрауличког прорачуна могао да одреди динамику простирања трасера. За реализацију симулације простирања трасера изабрана је метода коначних разлика, која као резултат даје динамику појављивања трасера у појединим тачкама система. Слично као и у случају хидрауличког модела, модел транспорта растворљивих материја се састоји од линијских елемената који представљају водопропусне канале и чворова који представљају спојеве између елемената. Као један од познатих граничних услова овог модела узимају се брзине течења и протоци кроз све елементе, које се добијају као резултат хидрауличког прорачуна. Други гранични услов је задата концентрација трасера током времена у чворовима у којима се врши његово инјектирање. На основу

ових граничних услова врши се израчунавање транспорта растворљиве материје што као резултат даје промену концентрације трасера током времена у свим чворовима система.

За пружање подршке у процесу одлучивања о параметрима уградње гранулисаног материјала, хидраулички модел је проширен моделом транспорта гаранулисаног материјала. Коришћењем овог модела врши се симулација уградње, транспорта и задржавања, односно изношења гранулисаног материјала. Истраживањима на овом математичком моделу добијају се параметри који се користе као основни показатељи за одређивање оптималне комбинације материјала различитих гранулација, брзине у градње и сл., а све у циљу задржавања агрегата у систему на најповољнији начин.

7.2 Формулација проблема

Параметри овако сложеног математичког модела не могу се одредити експериментално, већ се морају проценити кроз поређење резултата које даје модел са измереним величинама у реалном систему. Отуда се поступак одређивања параметара модела, то јест калибрација модела, своди на оптимизациони проблем који се може формулисати на следећи начин: над скупом могућих параметара модела подземних токова одредити оне параметре који минимизују разлику између резултата добијених симулацијом модела и величина измерених у реалном систему. Услед комплексности система који је моделиран, калибрација је укључивала више различитих критеријума којима је оцењиван ниво усаглашености модела и реалног физичког система. Отуда је калибрација мулти-модела процуривања на ХЕ "Вишеград" морала бити сведена на решавање проблем ВКО.

Током трајања пројекта санације процуривања испод бране ХЕ "Вишеград" калибрација развијеног математичког модела вршена је у више фаза. У првој фази, пре почетка санације, модел је калибрисан да би се утврдио тренутни распоред подземних токова и у складу са њиме направио почетни план санације. Приликом ове калибрације, из прорачуна је изузета симулација транспорта и седиментације гранулисаног материјала, која има удела тек у фази коришћења модела за саму санацију. У току процеса санације читав мулти-модел је више пута поново калибрисан како би у одабраним кључним тренуцима одсликавао тренутно стање у подземним шупљинама чије су се димензије и отпори мењали током времена услед уградње инертног материјала. Тренутне димензије и отпори водопропусних канала обједињени су под појмом "конфигурација" модела, док појам нулта конфигурација подразумева иницијално стање система, пре почетка уградње гранулисаног материјала. Узастопне кофигурације модела нису могле бити израчунаване надовезивањем прорачуна седиментације на претходну конфигурацију, већ је свака конфигурација морала бити одређена независно од претходне. Разлог лежи у повременој појави изношења глине из система, која није била претпостављена у фази израде модела. С обзиром на то да је ефекте ове појаве, услед недовољног познавања њене локације и интензитета, било

немогуће моделирати, прорачун седиментације није могао бити поуздан. Отуда је током коришћења модела за планирање даље уградње гранулата, вршена повремена "рекалибрација" модела током које су утврђиване вредности одређених параметара како за тренутну кофигурацију, тако и за одабране кључне конфигурације које су јој претходиле. У процес рекалибрације увек је била укључена и нулта конфигурација. Разлог за стално преиспитивање параметара претходних конфигурација је немогућност да се потпуно прецизно одреде просторни распоред и особине подземних канала услед сложености мреже коју они сачињавају. Дакле, на основу понашања које је реални систем испољио након урадње одређене количине и врсте гранулата, вршене су евентуалне корекције почетно претпостављеног модела система. Приликом сваке нове рекалибрације нулте конфигурације, њени параметри одређени у претходној калибрацији коришћени су за одређивање граничних вредности у оквиру којих ће бити тражене нове вредности параметара.

Иницијалном калибрацијом модела извршена је естимација нулте конфигурације модела. За елементе топологије подземних токова у моделу, било је потребно одредити њихове неопознате димензије (еквивалентни радијус и корекције дужина), и отпоре, тако да конфигурација буде усклађена са стварним стањем система. Усклађеност процене конфигурације са стварним стањем система у датом временском тренутку одређивана је на основу три показатеља:

- Подударање пијезометарских нивоа конфигурације која се процењује са измереним вредностима;
- Подударање брзина течења на изворима конфигурације која се процењује са измереним вредностима;
- Подударање динамике транспорта трасера кроз конфигурацију која се процењује са динамиком измереном на стварном систему.

Како би при наредним калибрацијама били узети у обзир ефекти уградње гранулисаног материјала и чињенице да је свака конфигурација последица ефеката седиментације на претходну конфигурацију, уводе се два нова параметра везана за стварање локалних отпора услед нагомилавања материјала у проводним елементима. С обзиром на то да се запремина једне конфигурације мора разликовати од запремине претходне конфигурације приближно за количину материјала уграђеног током времена протеклог између те две конфигурације, уводи се и четврти показатељ поклапања конфигурација са реалним стањима у систему - морфолошки критеријум.

7.2.1 Непознати параметри система

Као непознати параметри система које је потребно одредити узимају се:

1. Еквивалентни радијуси канала у нултој конфигурацији. За остале конфигурације се као непознате узимају процентуалне промене радијуса у односу на

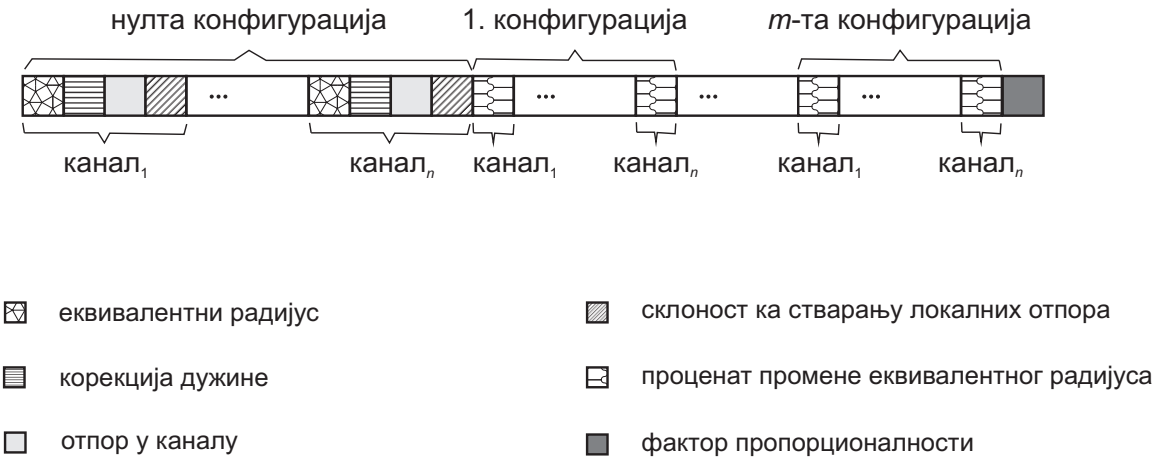
претходну конфигурацију и то само за канале у којима је могуће таложење материјала (на основу ранијих прорачуна седиментације).

2. Корекције дужина канала услед геометријске неправилности. Корекције дужина су непроменљиве током времена, тако да се одређују само у почетној конфигурацији.

3. Отпори у каналима у почетној конфигурацији. Отпори канала у осталим конфигурацијама се могу израчунати на основу отпора у почетној конфигурацији и количине уграђеног материјала у сваком од канала.

4. Параметар склоности сваког елемента ка стварању локалних отпора услед нагомилавања материјала. Ови параметри су непроменљиви током времена и заједнички су за све конфигурације.

5. Фактор пропорционалности утицаја количине нагомиланог материјала на локалне отпоре, који је исти за све елементе и све конфигурације.



Слика 7.2: Приказ једног хромозома: сваки квадрат означава један ген - реалан број који представља вредност којом се естимира један непознати параметар модела. Број гена у хромозому једнак је производу броја водопропусних канала у моделу и броја конфигурација које се разматрају

У циљу решавања проблема употребом генетских алгоритама било је потребно извршити кодирање непознатих параметара читаве мреже канала у виду реално кодираних хромозома. Сваки хромозом је био састављен од низа гена који су представљали непознате параметре свих подземних канала, за сваку конфигурацију која је разматрана приликом калибрације (слика 7.2). За нулту конфигурацију је за сваки канал било потребно одредити по четири непозната параметра, док је код осталих конфигурација за сваки канал био непознат по један параметар. Додатно, један ген је одређивао фактор пропорционалности, који је исти за све елементе свих конфигурација. Како је у систему било преко 200 канала, хромозоми су, у зависности од броја конфигурација, имали и по више од 1000 гена.

7.2.2 Циљне функције

У складу са раније наведеним критеријумима по којима се оцењује усклађеност конфигурације са стварним стањем система, утврђене су следеће циљне функције:

1. **Циљна функција нивоа подземних вода**. Функција, у ознаци J_p, дефинисана је на следећи начин:

$$J_p = \sum_{i=1}^{n} |\bar{z}_i - z_i| \tag{7.1}$$

где је $\bar{z}_i$ осмотрени пијезометарски ниво у i-том пијезометру или бушотини, z_i прорачунски пијезометарски ниво у одговарајућем чвору модела, а n број чворова модела у којима се врши поређење. Чворови модела за поређење су изабрани тако да њихова удаљеност од осмотреног пијезометра буде што мања. У случају када се више чворова пореди са истим пијезометром, узима се у обзир само онај чвор чија је прорачунска вредност најприближнија осмотреној.

2. **Циљна функција интензитета провирања**. Функција, у ознаци J_q, дефинисана је на следећи начин:

$$J_q = \sum_{i=1}^{n} |\bar{v}_i - v_i| + |\bar{Q}_{prfin} - Q_{prfin}| \tag{7.2}$$

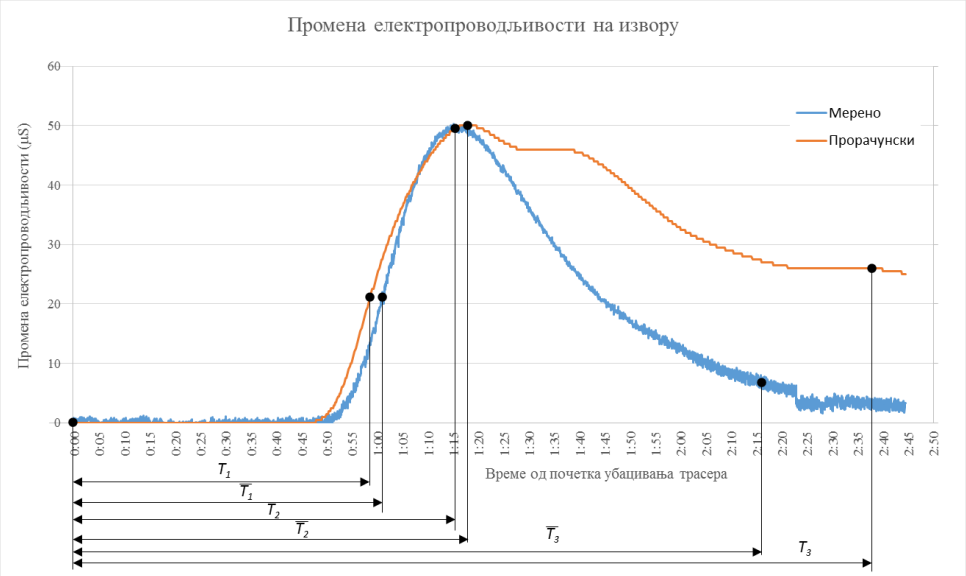
где је $\bar{v}_i$ осмотрена брзина на i-том извору, v_i прорачунска брзина у одговарајућем каналу, а $\bar{Q}_{prfin}$ и Q_{prfin} су укупно осмотрено, односно прорачунско, провирање после санације, респективно. Променљива n означава број чворова модела у којима се врши поређење

3. **Циљна функција транспорта растворљивих материја**. Функција, у ознаци J_T, дефинисана је на следећи начин:

$$J_T = \sum_{i=1}^{n} |\bar{T}_{1,i} - T_{1,i}| + \sum_{j=1}^{n} |\bar{T}_{2,j} - T_{2,j}| + \sum_{k=1}^{n} |\bar{T}_{3,k} - T_{3,k}| \tag{7.3}$$

где су $\bar{T}_{1,i}$, $\bar{T}_{2,j}$, $\bar{T}_{3,k}$ и $T_{1,i}$, $T_{2,j}$, $T_{3,k}$ вредности три карактеристична временска интервала из осмотрених, односно прорачунских, серија проводности, док је n број извора на којима се осматра проводност. Карактеристични временски интервали у детекцији трасера, приказани на слици 7.3 су време појављивања T_1, време достизања максимума T_2 и време проласка трасера T_3. Време појављивања се узима као време од почетка опита до проласка 5% укупне количине детектованог трасера. Време проласка Т3 се узима као време од почетка опита до проласка 95% укупне количине детектованог трасера. Ова два времена се рачунају по следећим формулама:

$$\int_0^{T_1} s(t)dt = 0.05 \int_0^{T_U} s(t)dt \quad \text{и} \quad \int_0^{T_3} s(t)dt = 0.95 \int_0^{T_U} s(t)dt \tag{7.4}$$



Слика 7.3: Карактеристични временски интервали који учествују у циљној функцији

где је $s(t)$ осмотрена проводност у тренутку t, а T_U је укупно време током кога је осматрана проводност у датом експерименту.

4. **Циљна функција промене морфологије модела**. Функција, у ознаци J_M, израчунава се на следећи начин:

$$J_M = \left| \sum_{i=1}^{n} V_{+,i} - V_{izn} \right| + \left| \sum_{j=1}^{m} V_{-,j} - V_{ugr} \right| \tag{7.5}$$

при чему је $V_{+,i}$ ($V_{-,j}$) увећање (умањење) i-тог (j-тог) елемента између конфигурација. V_{izn} је претпостављена запремина изнетог материјала и глине у односу на претходну конфигурацију, а V_{ugr} је запремина уграђеног материјала у односу на претходну конфигурацију. Вредност ове циљне функције у нултој конфигурацији је једнака 0, јер не постоји конфигурација која претходи нултој.

Да би било могуће упоредити нивое подземних вода (пијезометарске нивое) и интензитете провирања на изворима процењене и стварне конфигурације, при истим котама горње и доње воде, неопходно је извршити хидраулички прорачун за процењену конфигурацију и резултате упоредити са измереним вредностима.

Након извршеног хидрауличког прорачуна и добијања потенцијала и протока у систему, за потребе одређивања вредности треће циљне функције, симулира се простирање трасера убачених у одређеним тачкама модела. Резултат прорачуна су временски дијаграми проводности на изворима који се упоређују са временстим интервалима из осмотрених серија проводности.

Под променом морфогије модела између узастопних конфигурација подразумевају се промене у димензијама канала, које треба да буду у сагласности како са

количином уграђеног материјала, тако и са појавом изношења глине и раније уграђеног материјала из система канала. Стога се за сваки појединачни елемент одређује промена запремине између сваке узастопне конфигурације. Смањење запремине елемената се третира као количина инертног материјала задржаног у каналима, па се у складу са тим и пореди са познатим количинама уградње. Увећање запремине елемената се третира као изношење глине и раније уграђеног материјала, па се пореди са процењеним вредностима изнетог материјала и глине.

7.3 Процедура калибрације модела

На слици 7.4 приказан је процес естимације параметара модела коришћењем ГА. Процес започиње формирањем почетне генерације параметара модела, при чему се параметри за све елементе нулте конфигурације претпостављају на основу геолошких осматрања. За све остале конфигурације претпостављају се процентуалне промене радијуса у односу на претходну конфигурацију само за елементе у којима је било могуће таложење уграђеног материјала. Оцењивање сваке јединке у генерацији тече на следећи начин:

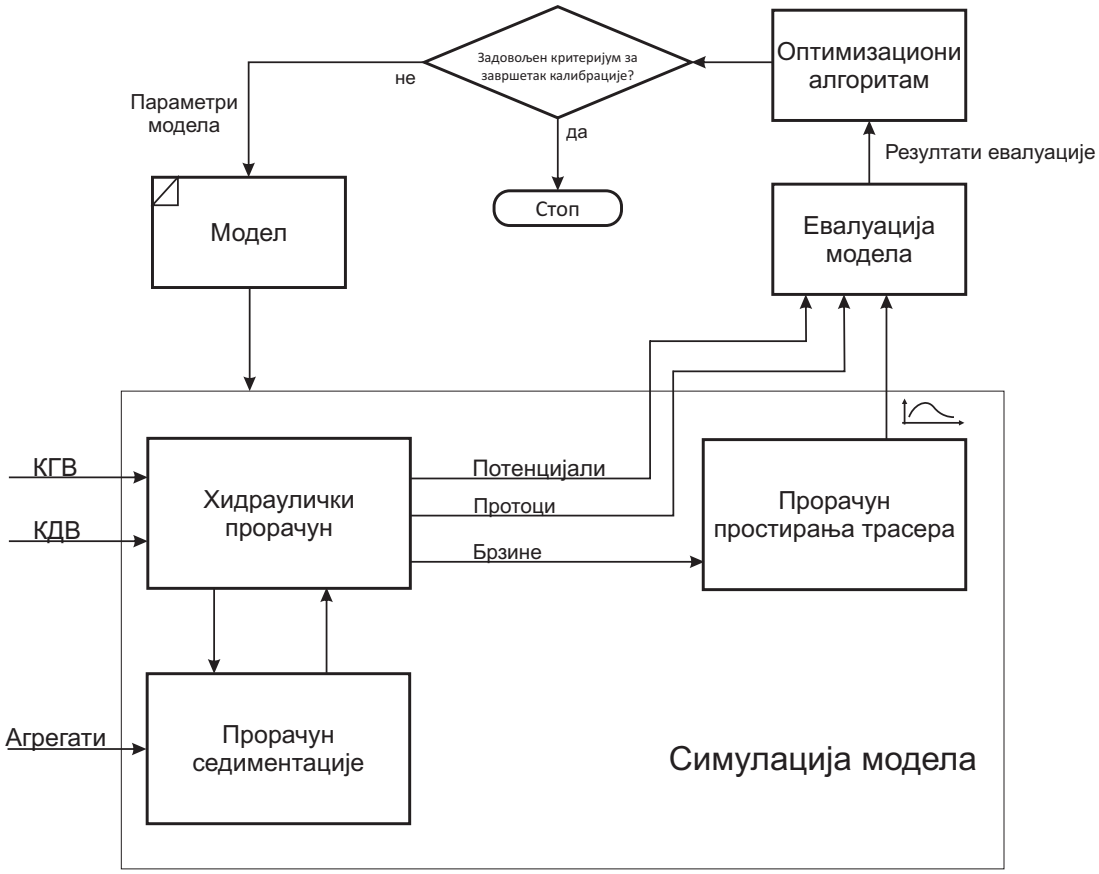
1. На основу претпостављених вредности параметара врши се хидраулички прорачун који за резултат има потенцијале, протоке и брзине флуида у моделу. Коришћењем добијених протока врши се прорачун простирања трасера. Када је позната хидрауличка слика (потенцијали, протоци и брзине) и времена појављивања трасера у одређеним тачкама система, њиховим поређењем са измереним вредностима се добија оцена ваљаности претпостављеног скупа параметара у нултој конфигурацији. Морфолошки критеријум се не узима у обзир приликом оцењивања нулте конфигурације, јер још није било уградње материјала, па је вредност четврте циљне функције за нулту конфигурацију једнака нули.

2. Даље се врше хидраулички прорачуни за остале конфигурације модела које се разматрају. Параметри ових конфигурација, као што је раније наведено, зависе од претпостављених параметара нулте конфигурације као и од количине уграђеног материјала. Конфигурације се, независно једна од друге оцењују по прве три циљне функције, након чега се врши оцењивање према морфолошком критеријуму. Приликом формирања ове оцене узима се обзир чињеница да је свака конфигурација последица ефеката седиментације на претходну конфигурацију.

3. Након успешно извршених прорачуна свих конфигурација сабирају се вредности одговарајућих оцена и добијају се коначне вредности за све четири циљне функције, које се у алгоритму даље користе у процесу рангирања јединки према њиховом степену прилагођености по сваком од критеријума.

На основу оцена целе генерације параметара алгоритам бира најбоља решења и њиховим укрштањем ствара нову генерацију параметара модела са којима се изнова

улази у читав процес. Након достизања задовољавајућег поклапања израчунатих и измерених вредности, параметри се проглашавају за највероватније могуће.

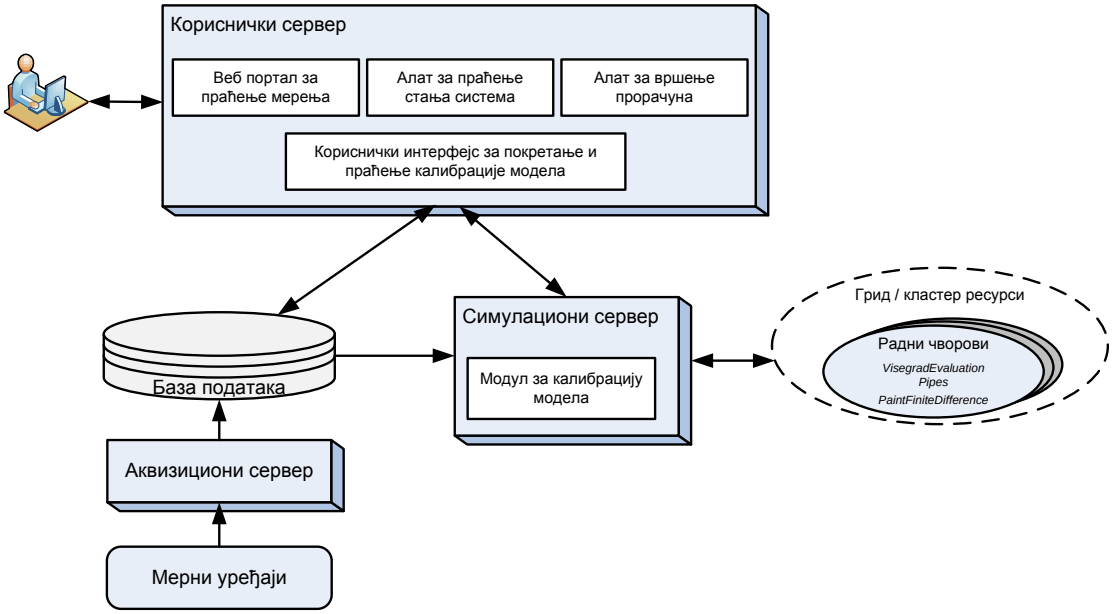


Слика 7.4: Процес естимације параметара модела

7.4 Софтверски систем за подршку одлучивању у процесу санације процуривања испод хидроелектране "Вишеград"

Процес уградње гранулисаног материјала у подземне канала у циљу санације процуривања је јако сложен процес, који је морао бити праћен и контролисан током целог трајања. Као подршка одлучивању у процесу запуњавања понора коришћен је софтверски систем за подршку одлучивању (слика 7.5), који врши естимацију конфигурације мреже карстних шупљина и канала и симулацију процеса уградње агрегата, те готово у реалном времену процењује успешност реализације технолошких поступака. Иза успешног функционисања овог софтверског система стоји претходно

описани математички модел који у сваком тренутку мора бити ажуран. Зато је раније приказани поступак одређивања његових параметара спровођен пре израде сваког периодичног плана уградње, као и онда када су се праћењем процеса уградње агрегата у реалном времену уочавала знатна одступања осмотрених и прорачунских величина.

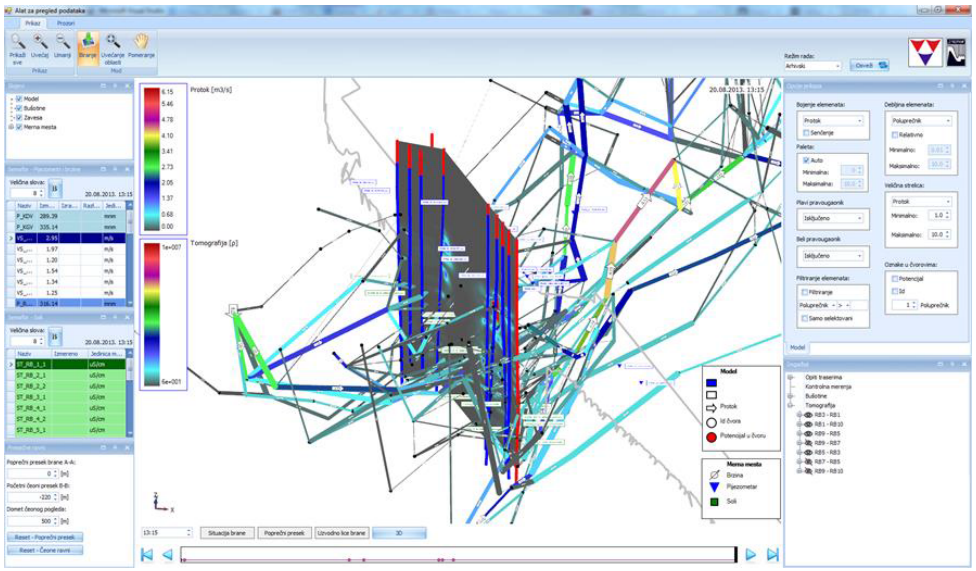


Слика 7.5: Компоненте софтверског система за подршку одлучивању

Слика 7.6: Праћење промена осматраних величина на веб порталу

Софтверски систем је коришћен у два режима:

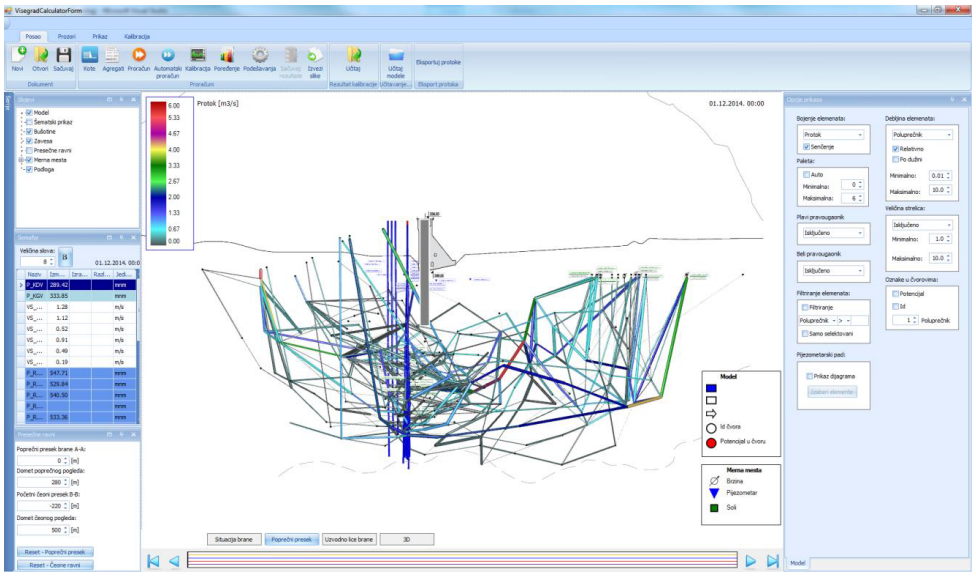
- у реалном времену за упоредни приказ и анализу прорачунских и осмотрених вредности одабраних величина,
- повремено су вршене симулације предстојеће уградње у циљу избора оптималних параметара процеса уградње.



Слика 7.7: Приказ алата за праћење стања система

Коришћење успостављеног система у реалном времену подразумевало је да експерт преко веб портала (слика 7.6) директно прати вредности појединих величина које су тренутно мерене уређајима постављеним на терену. Алатом за праћење стања система - *VisegradObserver* (слика 7.7) даље се могао извршити хидраулички прорачун над текућом конфигурацијом система за тренутно измерене граничне услове (кота горње и кота доње воде), и добити визуелна информација о ефектима које уградња агрегата има на промену значајних параметара подземних канала. Аутоматским коришћењем математичког модела естимиране су вредности које би се мериле да није било уградње, па се поређењем са осмотреним величинама добијала информација о ефикасности уградње.

Поред употребе система у реалном времену, периодично се приступало вршењу симулација различитих сценарија уградње речног агрегата коришћењем алата за вршење прорачуна (слика 7.8). Над ажурним стањем математичког модела вршена је симулације уградње уз задавање коте горње и коте доње воде и динамике уградње агрегата. Анализирани су резултати који су указивали на место и количине задржавања агрегата, промене брзина, протока и пијезометарских нивоа. Стање система је кретањем по временској оси анализирано и у тренуцима појављивања битних догађаја у протеклом времену. На основу анализа, утврђивани су оптимални параметри

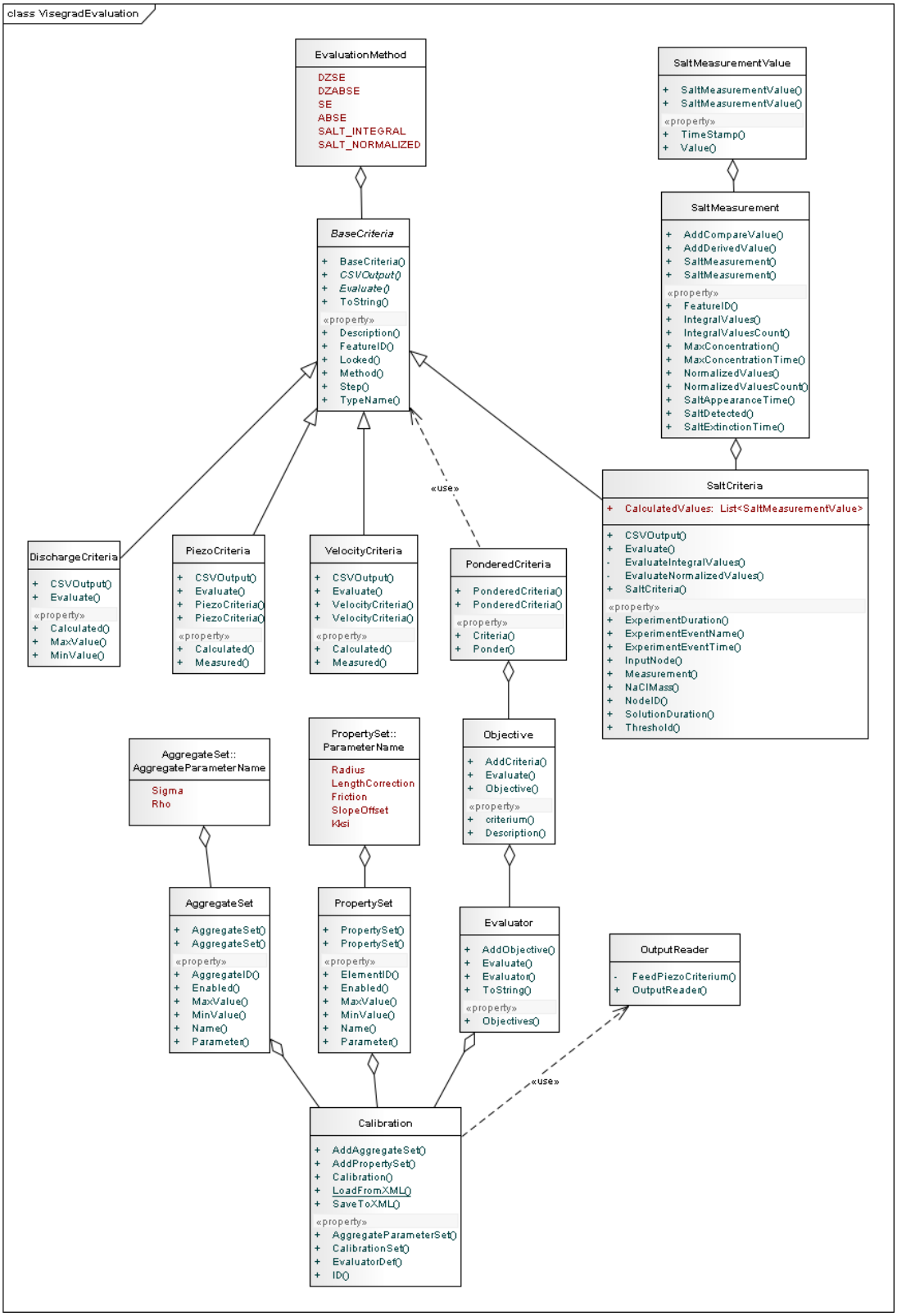


Слика 7.8: Приказ алата за вршење прорачуна

процеса (гранулације, дневни распоред количина, динамика уградње као и количине потребних гранулација) и дефинисани периодични планови уградње.

Саставни део софтверског система за подршку одлучивању у процесу санације провирања вода испод ХЕ Вишеград чини и модул за калибрацију модела који је имплементиран кроз неколико софтверских библиотека. Основу овог модула чини библиотека за вишекритеријумску оптимизацију *JARE* описана у делу 4.2. Као што је раније наведено проблем естимације параметара модела се може формулисати као проблем вишекритеријумске оптимизације, те је за његово решавање искоришћен *NSGA-II* алгоритам. Софтверска реализација проблема естимације је реализована наслеђивањем посебне класе *VisegradProblem* из апстрактне класе *Problem*, која је саставни део библиотеке *JARE*. С обзиром на то да је оцењивање сваке јединке подразумевало 4 различите оцене, евалуација сваке генерације захтевала је значајно процесорско време, па је стога читав процес паралелизован употребом WoBinGO софтверског оквира. Отуда је за решавање проблема одређивања оптималних параметара модела коришћена модификација класе *NSGA-II*, класа *NSGAIIViaBinder* која имплементира *NSGA-II* алгоритам на такав начин да омогућава дистрибуирану евалуацију јединки путем WoBinGO-а. Израчунавање вредности циљних функција вршено је на радним чворовима рачунарског кластера или Грида.

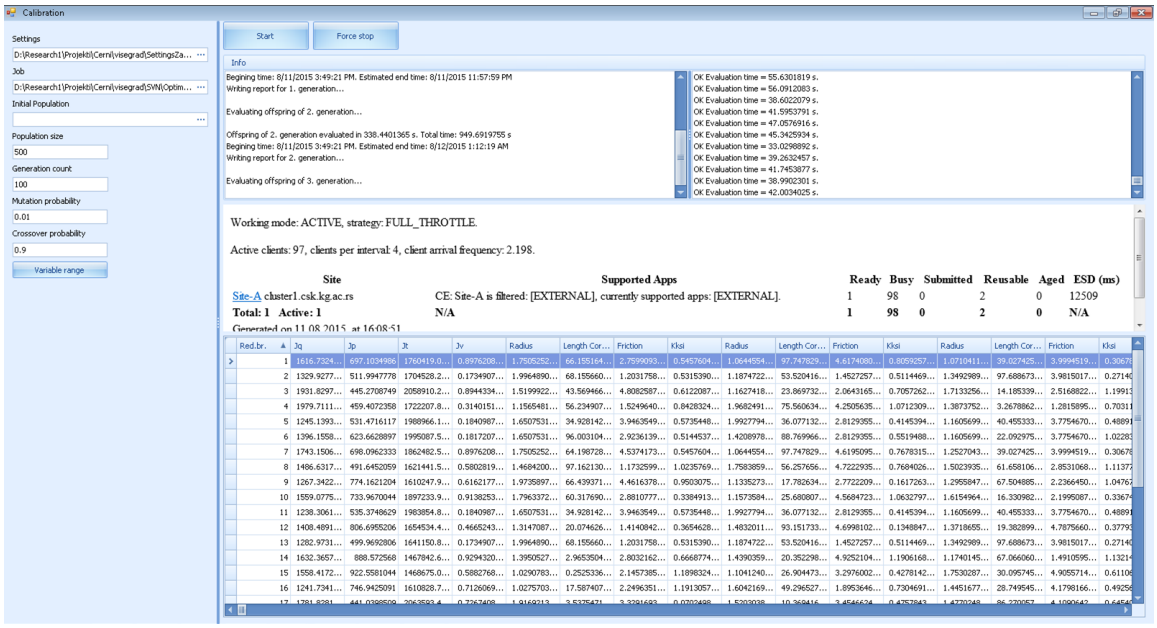
Пакет за евалуацију који је у току рада WoBinGO оквира прослеђиван на радне чворове садржао је конзолну апликацију *WorkerJob* која је задужена да од клијента преко *WB* проксија прими прво број параметара који сачињавају јединку која се оцењује, а потом и саме параметре један по један. Она је даље покретала евалуацију, а по њеном завршетку, клијенту истим путем враћала израчунате вредности циљних функција. Само одређивање ових вредности вршено је коришћењем C #/.NET библиотеке класа *VisegradEvaluation* (слика 7.9) која је такође била део пакета за



Слика 7.9: Дијаграм класа библиотеке VisegradEvaluation

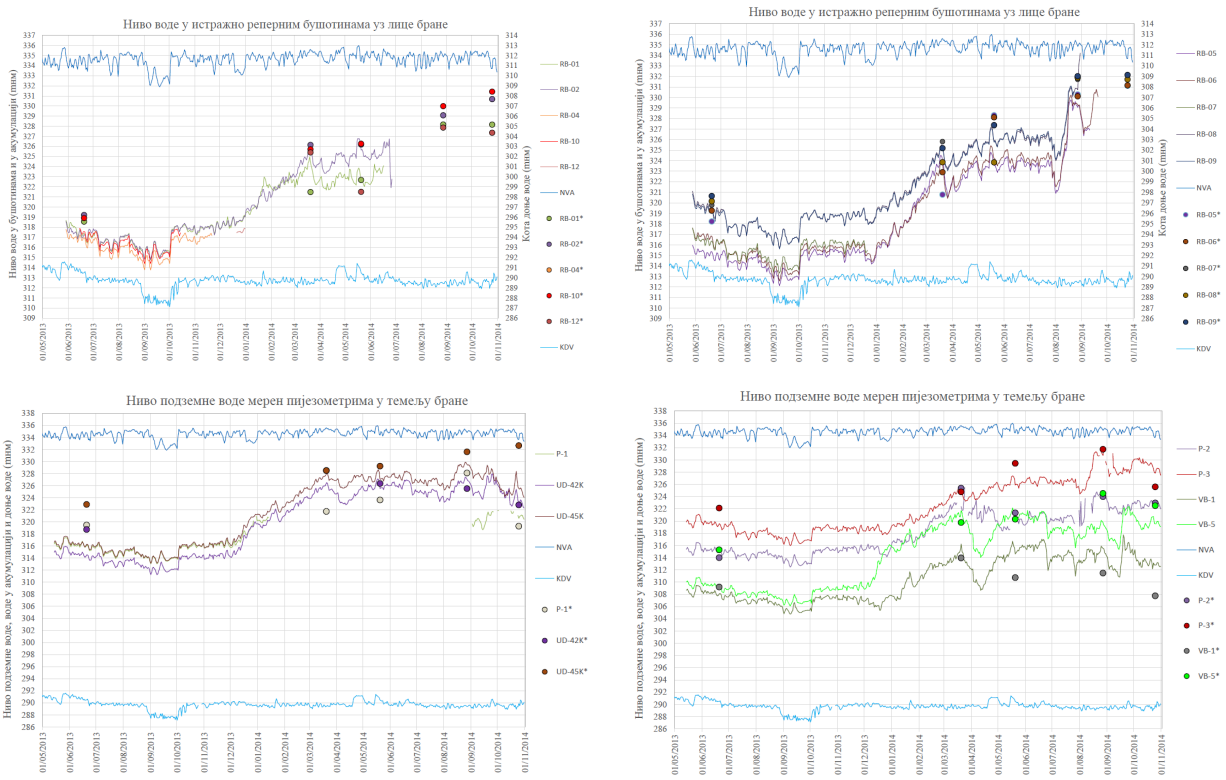
евалуацију. Основна класа ове библиотеке *Calibration* садржи скуп параметара модела које је потребно естимирати, заједно са њиховим границама (колекција објеката

класе *PropertySet*). Поред тога, ова класа садржи и колекцију критеријума за оцењивање јединки, груписаних унутар класе *Evaluator*. Критеријуми за оцену пијезометарских нивоа, брзина, протока и трасера су реализовани кроз класе *PiezoCriteria*, *VelocityCriteria*, *DischargeCriteria* и *SaltCriteria*, које су изведене из основне класе *BaseCriteria*. Сам прорачун хидраулике и седиментације развијен у виду посебне *C++* конзолне апликације назване *Pipes*. *Pipes* користи методу коначних елемената за симулацију течења кроз математички модел при задатим граничним условима. Поред прорачуна потенцијала и протока у систему подземних канала, ова апликација врши и прорачун транспорта гранулисаног материјала и интензитета његовог таложења у подземним каналима. Симулација транспорта материје кроз модел вршена је методом коначних разлика коришћењем C#/.NET библиотеке класа под називом *PaintFiniteDifference*. Апликација *Pipes* и библиотека *PaintFiniteDifference* су се такође налазиле у пакету за евалуацију, који је додатно садржао и комплетно *Mono* окружење, које није достуно локално на радним чворовима, а било је неоходно ради извршавања модула развијених у C#/.NET језику на *Linux* оперативном систему.



Слика 7.10: Изглед корисничког интерфејса за покретање и праћење процеса калибрације модела XE Вишеград

Покретање калибрације модела текло је преко корисничког интерфејса (слика 7.10) који омогућава одабир конфигурационих датотека неопходних за рад WoBinGO-а, величине популације и броја генерација, вероватноће укрштања и вероватноће мутације. Интерфејс нуди и опцију да се као почетна популација изабере нека генерација јединки добијена при ранијим калибрацијама модела. Кориснички интерфејс омогућава праћење статуса оптимизационог процеса, и даје увид у последњи израчунати Парето фронт. Додатно се приказује и веб портал на коме се очитава статус *pool*-а *WB* сервиса, дајући кориснику податке о томе на које *CE*-ове су дистрибуирани послови који врше евалуацију јединки, као и колико послова је тренутно у



Слика 7.11: Поређење прорачунских и осмотрених пијезометарских нивоа

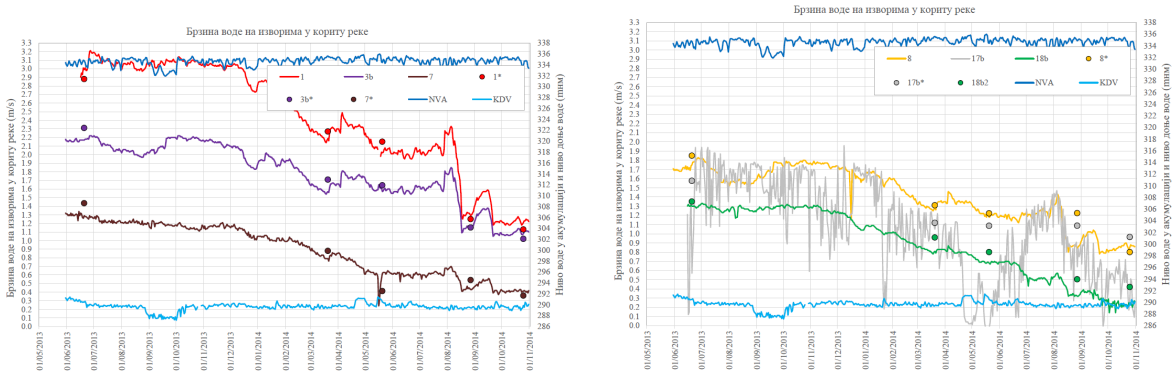
Submitted, *Ready*, *Busy*, или *Aged* статусу.

7.5 Резултати калибрације математичког модела

Коришћењем калибрисаног математичког модела добијени су задовољавајући резултати који су приказани на наредним дијаграмима. На слици 7.11 упоредо су приказане прорачунске и осмотрене вредности пијезометарских нивоа на различим локацијама. Прорачунске и осмотрене вредности брзина на изворима дате су на слици 7.12. Ради веће прегледности, на сваком дијаграму су приказани резултати за по три извора. На обе слике осморене вредности су дате за комплетан период санације (пуне линије), док су прорачунске вредности приказане у карактеристичним конфигурацијама (серије означене са "*"), и то у циљу што бољег сагледавања резултата.

На слици 7.13 су дати упоредни дијаграми за осмотрене и прорачунске вредности промене електропроводљивости на репрезентативном извору. Дијаграми су формирани за опите солима и то за све значајне конфигурације.

Током санације редовно је вршен прорачун процењеног укупног провирања, што је на слици 7.14 приказано плавом линијом. Уз то, позната је динамика уградње инертног материјала, па је било могуће одредити и количине материјала које су се



Слика 7.12: Поређење прорачунских и осмотрених брзина извирања

задржале у систему карстних канала. Ове количине се могу одредити само у тренуцима који одговарају дефинисаним конфигурацијама, и на претходној слици су обележене црвеним круговима. Почетна вредност процењеног провирања је износила 14,03 m3/s. Након санације, ова вредност износи 4,47 m3/s што упућује на значај извршених радова. Смањење провирања је последица задржавања агрегата у систему канала испод бране, па је на слици дата и процена задржаног агрегата на основу прорачуна.

Наведени прикази осмотрених и прорачунских вредности релевантних величина указују да је поузданост добијених резултата на задовољавајућем нивоу и да се коришћени математички модел показао као добра основа за пружање подршке у доношењу одлука.

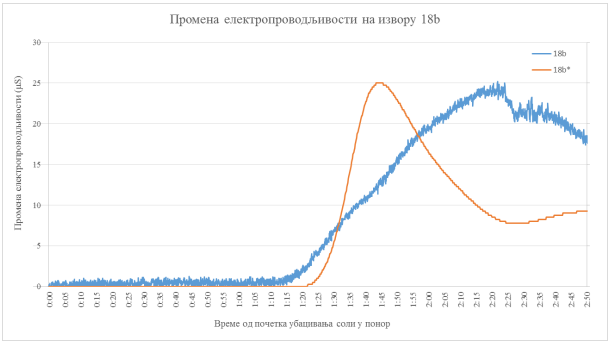
7.6 Ефикасност WoBinGO софтверског оквира у решавању сложеног реалног оптимизационог проблема

У делу 6.1 је кроз експерименте у којима је решаван вештачки креиран тест проблем показано да се коришћењем WoBinGO оквира за проблеме са дуготрајним евалуацијама процес решавања проблема може значајно убрзати. С друге стране, у делу 4.6.2 се као једна од кључних карактеристика WoBinGO-а наводи његова способност да еластично заузима ресурсе и тиме их ефикасно употребљава. У овом делу рада ће претходно наведене тврдње бити додатно потврђене кроз резултате експеримената спроведених у случају употребе WoBinGO софтверског оквира за естимацију параметара сложеног математичког модела мреже подземних токова.


Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у понор


Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у RB-10


Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у RB-15


Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у понор

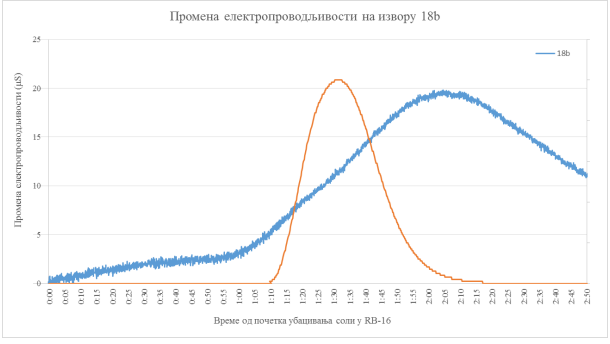

Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у RB-10

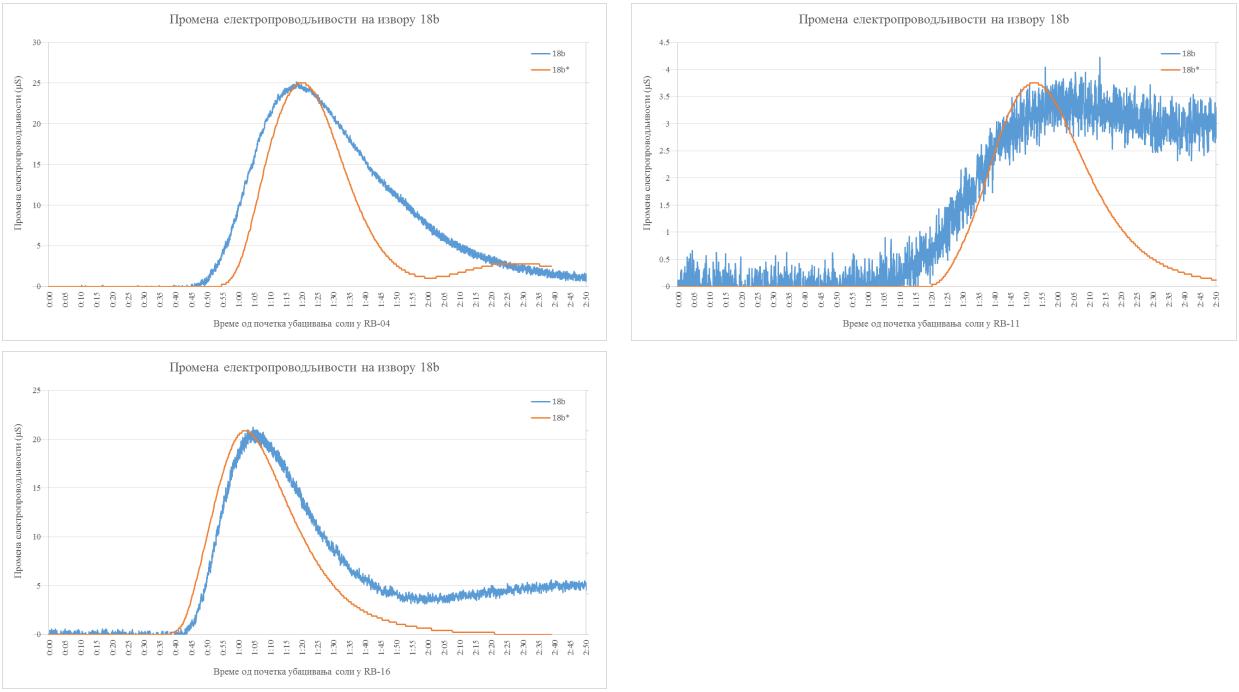

Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у RB-15


Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у RB-10

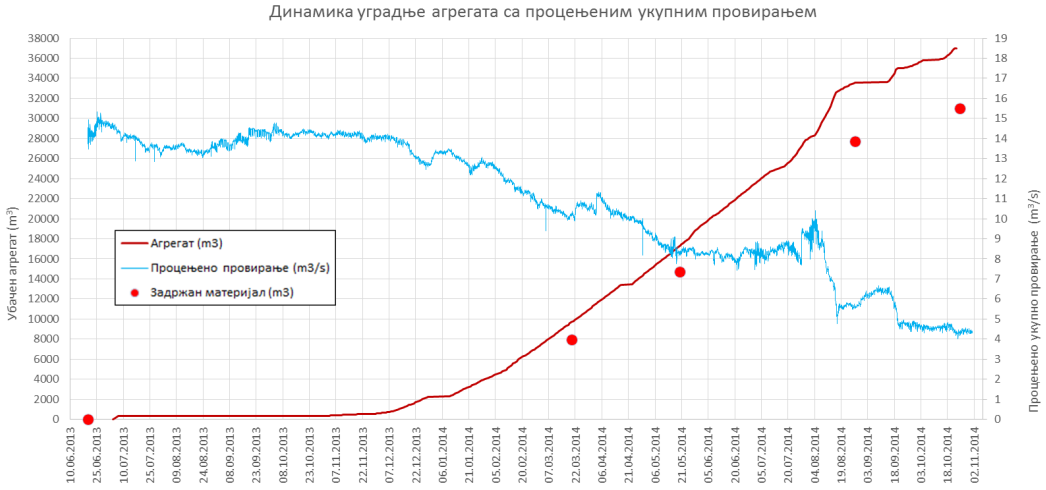

Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
18b*
Време од почетка убацивања соли у RB-11


Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
Време од почетка убацивања соли у RB-15


Промена електропроводљивости на извору 18b
Промена електропроводљивости (μS)
18b
Време од почетка убацивања соли у RB-16



Слика 7.13: Поређење прорачунске и осмотрене промене електропроводљивости



Слика 7.14: Дијаграми уградње инертног материјала и процењеног провирања са процењеним количинама задржаног материјала у појединачним конфигурацијама

7.6.1 Експериментална подешавања

Као што је раније наведено за решавање проблема вишекритеријумске калибрације модела процуривања коришћен је *NSGA-II* алгоритам. Параметри алгоритма су имали следеће вредности: величина популације је била 500 јединки, максимални број генерација је био 500, коришћено је симулирано бинарно укрштање са вероватноћом од 0.9 и индексом расподеле 20, и полиномна мутација са вероватноћом $1/l$, где је l дужина хромозома. Сви резултати су добијени из 10 независних

Сајт	**Чворови**	**CPU тип**	**RAM/чвор**	**OS/LRMS**	$\mathbf{t_{eval}}$**(s)**
AEGIS04-KG	5	2x16 core Opt. 6276@2.3GHz	96 GB	SL6.4 x86_64 PBS Torque	68.12
AEGIS01-PHY-SCL	88	2x4 core Xeon E5345@2.33GHz	8 GB	SL6.4 x86_64 PBS Torque	66.65
AEGIS03-ELEF-LEDA	8	2x4 core Xeon E5420@2.5GHz	4 GB	SL6.4 x86_64 PBS Torque	65.77

Табела 7.1: Карактеристике кластера коришћених кроз извршене тестове

извршавања експеримента.

Експерименти су извршени у Грид окружењу које се састоји од три кластера представљених одговарајућим CE-овима. Њихове карактеристике су обједињене у табели 7.1. Последња колона показује просечно време извршавања једне евалуације циљних функција за сваки од кластера. Иако разлике између кластера нису велике, ипак су приликом израчунавања T_{ser} коришћена наведена просечна времена потребна за једну евалуацију на сваком од кластера. Отуда је употребљена следећа формула:

$$T_{ser} = \lambda(n) + n \sum_{i=1}^{3} \frac{\bar{p}_i}{\bar{p}} t^i_{eval}, \tag{7.6}$$

где је n величина популације; t^i_{eval} означава просечно време потребно да се израчунају вредности циљних функција једне јединке на i-том рачунарском кластеру; $\frac{\bar{p}_i}{\bar{p}}$ је удео просечног броја процесора који извршавају евалуације на i-том кластеру у укупном просечном броју процесора на сва три кластера.

7.6.2 Резултати и дискусија

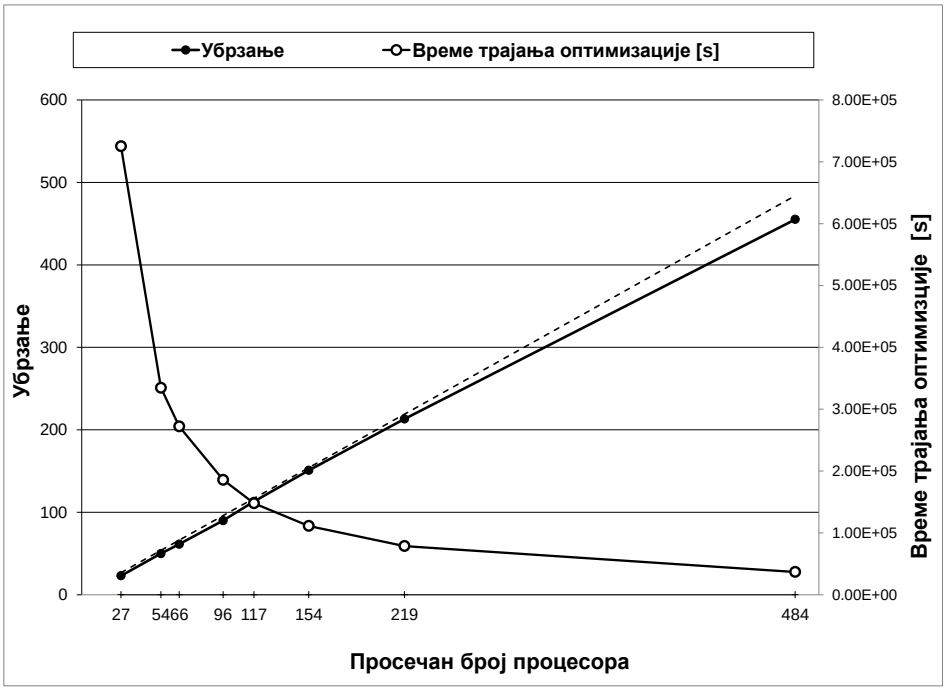
Табела 7.2 сумира резултате добијене експериментима. Услед еластичног понашања WB, вредност T_{ser} се мења у складу са једначином (7.6). T_{par} је експериментално одређено време потребно да популација од 500 јединки еволуира кроз 500 генерација при чему је оцењивање јединки паралелизовано.

Слика 7.15 даје визуелни приказ убрзања које је достигнуто приликом решавања проблема калибрације модела. Додатно је на графикону приказано и укупно време трајања ове оптимизације. Ради веће прегледности, време трајања процеса калибрације на једном процесору изостављено је са слике, јер се за неколико редова величине разликује од приказаних времена. С обзиром на то да просечно време потребно да се на једном процесору оцени једна јединка износи више од 66 секунди, евалуација генерације од 500 јединки би на једном процесору трајала готово 23 дана. Дакле, трајање читаве калибрације на једном процесору би износило преко 6 месеци, за разлику од нешто више од 10 сати на просечно 484 процесора уз употребу

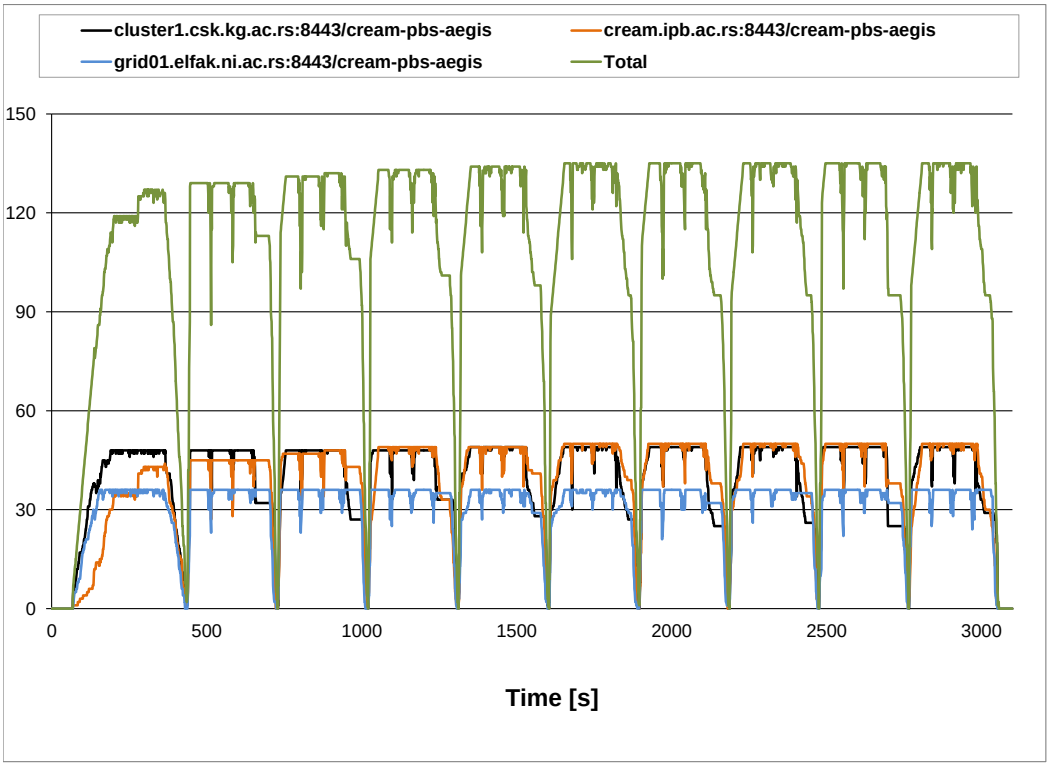
cluster1.csk.kg.ac.rs	cream.ipb.ac.rs	grid01.elfak.ni.ac.rs	$\bar{p}$	T_{ser}	T_{par}	T_{ser}/T_{par}
$\bar{p}_1$	$\bar{p}_2$	$\bar{p}_3$				
8	10	9	27	1.67E+07	7.25E+05	23.02
17	18	19	54	1.67E+07	3.35E+05	49.91
20	23	23	66	1.67E+07	2.72E+05	61.35
29	35	32	96	1.67E+07	1.86E+05	89.90
41	43	32	117	1.67E+07	1.48E+05	113.18
58	88	8	154	1.68E+07	1.11E+05	150.87
78	119	22	219	1.68E+07	7.87E+04	213.16
123	302	59	484	1.67E+07	3.68E+04	455.23

Табела 7.2: Емпиријски резултати добијени решавањем реалног проблема калибрације модела процуривања на ХЕ "Вишеград" коришћењем WoBinGO оквира и *NSGA-II* алгоритма (време је дато у секундама).

WoBinGO оквира. Управо је изузетно убрзање које се добија паралелизацијом калибрације модела путем WoBinGO-а омогућило коришћење система за подршку одлучивању у готово реалном времену и последично обезбедило доношење исправних одлука током процеса санације.



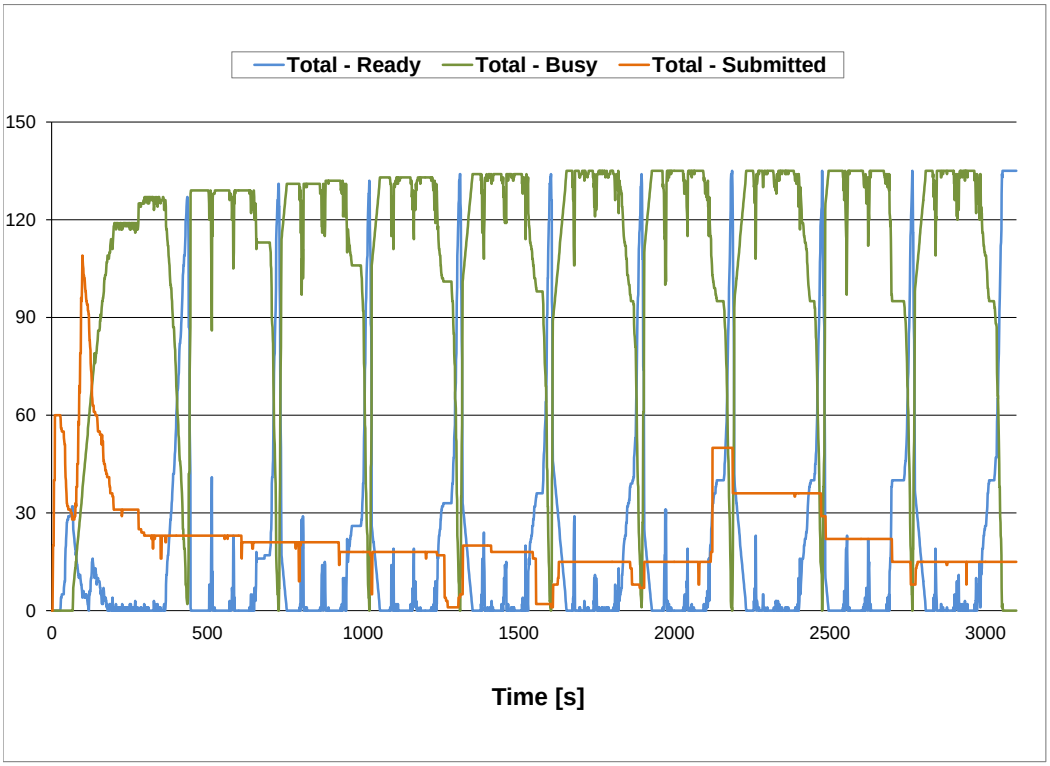
Слика 7.15: Просечно време трајања калибрације модела процуривања и достигнуто убрзање



Слика 7.16: Број упослених процесора на сваком од кластера појединачно, као и укупно на сва три кластера

Главна предност WoBinGO оквира у односу на друга слична решења огледа се у његовој способности да еластично алоцира послове на Гриду у складу са захтевима клијената. Са циљем да се анализира еластично понашање WoBinGO оквира, извршен је експеримент у коме је бележен број *busy*, *ready*, и *submitted* послова на сваком од три упослена кластера током десет генерација. WoBinGO оквир је био конфигурисан тако да сума *busy*, *ready*, и *submitted* послова буде 50 по кластеру, док су остала експериментална подешавања иста као и она наведена на почетку овог дела рада. Добијени резултати су приказани на сликама 7.16 и 7.17. Слика 7.16 приказује како се кроз време мења број упослених процесора на сваком од кластера појединачно, као и укупно на сва три кластера. На слици 7.17 се може видети број *ready*, *busy* и *submitted* послова на сва три кластера током времена.

Док нема клијентских захтева, WoBinGO одржава број спремних послова на предефинисаном минимуму, и на тај начин не врши беспотребно оптерећење Грид инфраструктуре. Оног момента када клијенти почну да се обраћају *WB* сервису са захтевима за евалуацију, спремни послови постају упослени оцењивањем јединки. Како је број спремних послова био веома мали, нови послови бивају одмах предати на извршење, и ускоро постају *busy* испуњавајући клијентске захтеве. Током евалуације генерације, број *busy* послова остаје висок. На крају прве генерације, клијент-



Слика 7.17: Укупан број *ready*, *busy* и *submitted* послова на сва три кластера

ски захтеви за пословима на тренутак престају, тако да број *busy* послова пада на нулу. Када више нису *busy*, ови послови одмах постају *ready*, и ускоро поново бивају упослени евалуацијом наредне генерације. Са слике се такође може видети да број *submitted* послова никада није једнак нули. Ово је последица покушаја WoBinGO оквира да на сваком кластеру достигне максимално предефинисано оптерећење (50 радника), што није увек могуће услед оптерећености кластера од стране других корисника и/или њихове конфигурације.

Оваква еластичност у заузимању рачунарских ресурса представља главну новину коју доноси WoBinGO оквир у односу на слична софтверска решења. Сви приказани резултати у овој студији случаја, јасно говоре да и уз еластични приступ, који је од изузетне важности при коришћењу дељених ресурса, WoBinGO оквир пружа значајно убрзање када се користи за оптимизационе проблеме са рачунски захтевним евалуацијама. Узимајући у обзир и успешну санацију процуривања испод бране ХЕ "Вишеград", резултати изложени у овом поглављу несумљиво представљају доказ полазних хипотеза дисертације.

Закључак

С обзиром на то да већина реалних проблема захтева проналажење решења која треба да задовоље неколико, често супротстављених критеријума, методе вишекритеријумске оптимизације имају велику важност у пракси. Вештачко свођење проблема ВКО на проблеме једнокритеријумске оптимизације и решавање одговарајућим методама има следеће недостатке: решење проблема зависи од приоритета које је доносилац одлука на основу свог знања о проблему и тренутних потреба доделио функцијама циља; алгоритми за једнокритеријумску оптимизацију дају једно решење док проблеми ВКО, уместо једног оптималног решења, имају низ компромисних оптималних решења (познатих као Парето-оптимална решења). Решавање проблема ВКО вишекритеријумским генетским алгоритмима даје вишеструка Парето оптимална решења истовремено, што доносиоцима одлука пружа могућност да одаберу оно решење које је у датој ситуацији најбоље.

Генетски алгоритми припадају класи еволуционих алгоритама код којих се одређивање решења оптимизационих проблема заснива на природној селекцији или опстанку најспособнијих. Јединке које се у датим условима покажу као способније од осталих имају веће шансе да створе потомство и пренесу свој генетски материјал, чиме се добре особине умножавају и популација еволуира ка оптимуму. Извршавање ГА подразумева да се, из генерације у генерацију, одређује степен прилагођености сваке јединке у популацији по сваком од критеријума оптимизације. Стотине евалуација јединки у популацији кроз стотине генерација, за комплексне проблеме често чине да извршавање ГА траје неприхватљиво дуго. Овај недостатак ГА се превазилази њиховом паралелизацијом и имплементацијом на дистрибуираним рачунарским платформама.

Грид рачунарство обезбеђује неопходну инфраструктуру за имплементацију паралелних генетских алгоритама. Међутим, писање паралелних апликација које ће се извршавати на Гриду изискује значајан напор и експертско знање. Додатно, приликом сваког извршавања неке апликације на Гриду потребно је одабрати ресурсе и често незанемарљиво дуго чекати да захтев за тим ресурсима буде процесиран. Ту су и обавезна припрема, отпремање, надгледање и окончавање Грид послова, који се код различитих мидлвера могу обављати на различите начине. Све претходно набројано одвраћа истраживаче од веће употребе Грида у научне сврхе, па се развој алата који скривају комплексност Грида и олакшавају развој Грид апликација, намеће као

решење које би довело до веће употребе Грида за научне и инжењерске апликације.

Преглед литературе, који је извршен у трећем поглављу ове дисертацјие показује да је много тога већ урађено на пољу развоја система са пилот пословима који побољшавају искоришћеност Грида. Такође, развијани су и софтверски оквири који омогућавају оптимизацију коришћењем паралелних ГА на дистрибуираним рачунарским ресурсима. Ипак, може се уочити да не постоји ниједан софтверски оквир за паралелне метахеуристике, који еластично резервишући ресурсе, динамички користи пилот послове. Отуда је циљ ове дисертације био развијање софтверског оквира за ВКГА са аутоматском адаптивном алокацијом послова на Гриду који би паралелно вршили евалуације јединки.

У дисертацији су представљена четири софтверска оквира: вишенитни, *pull*, *push*, и WoBinGO, током чијег развоја се тежило испуњењу следећих захтева:

1. Паралелно извршавање евалуације јединки на дистрибуираним рачунарским ресурсима у процесу ВКГА.
2. Паралелно извршавање више инстанци паралелизоване оптимизације.
3. Једноставно додавање нових оптимизационих алгоритама, чији развој није везан за развој евалуатора јединки.
4. Евалуатори могу бити написани на било ком компајлерском или скрипт језику подржаном од стране оперативног система или окружења на коме ће се извршавати.
5. Корисници су потпуно ослобођени бриге о специфичним детаљима дистрибуираног рачунарства и обезбеђен им је брз приступ дистрибуираним ресурсима.
6. Аутоматска, еластична алокација ресурса обезбеђује кориснику софтверског оквира да не чека на ослобађање тачно одређених капацитета, већ да користи оно што је тренутно доступно, али и да заузима додатне ресурсе чим они постану слободни.
7. Еластични приступ ресурсима, са временски ограниченим резервисањем, омогућава осталим корисницима тих ресурса бржи приступ.

Оно што је заједничко за сва четири оквира јесте да обезбеђују паралелно извршавање једнокритеријумске и вишекритеријумске оптимизације засноване на ГА и да примењују господар-слуга модел паралелизације. Овај модел паралелног ГА је једноставан за имплементацију, а процесори слуге се могу додавати и уклањати динамички, што је у складу са идејом аутоматске еластичне алокације ресурса. Сви софтверски оквири обезбеђују да развој оптимизационих алгоритама не зависи од проблема који ће се тим алгоритмима решавати и да евалуатор јединки може бити дат у виду "црне кутије" којој се прослеђују параметри за евалуацију и од које се добијају оцене тих параметара.

Глобално гледано, архитектуру сва четири оквира сачињавају модул за ВКО, који извршава главни ток ГА, диспечер и радници. Диспечер има улогу посредника између господара и радника. Он јединке послате на евалуацију од стране господара расподељује радницима, који ће ту евалуацију обавити. Особа или тим који развија алгоритам за оптимизацију мора да раздвоји евалуациони корак од остатка алгоритма, али не мора да брине о детаљима дистрибуирања индивидуа и враћања резултата евалуације. На диспечеру је да јединке ефикасно расподели радницима и да прикупи повратне резултате.

Сви развијени оквири користе исти модул за ВКО, који представља флексибилну и надоградиву колекцију еволуционих алгоритама и пратећих компоненти неопходних за њихову реализацију. Евалуација решења ван главне нити ГА омогућена је апстрактном класом *EvaluationPool* која служи као основа за извођење различитих конкретних класа за различите начине дистрибуције јединки радницима.

Вишенитни оквир омогућава паралелизацију евалуације јединки у више нити на једном рачунару са вишејезгарним процесором. Са *push* и *pull* софтверским оквиром јединке се могу оцењивати на радним чворовима рачунарског кластера. Код *push* оквира диспечер има улогу клијента који захтева од радника, имплеметираних у виду *WCF* веб сервиса, да оцене јединке. Улоге су код *pull* оквира измењене, тако да је једини веб сервис менаџер, који прима захтеве и од господара и од радника. Господар шаље индивидуе диспечеру где оне чекају да буду оцењене. Са друге стране, радници диспечеру шаљу захтеве за јединкама које је потребно евалуирати, извршавају евалуацију и диспечеру враћају резултате. На захтев господара, диспечер му враћа натраг евалуиране индивидуе.

Потпуно остварење постављених захтева постигнуто је кроз WoBinGO софтверски оквир који пружа нов, еластичан приступ у искоришћењу рачунарских ресурса, пратећи динамику корисничких захтева за евалуацијама јединки. Непотребно заузимање ресурса је избегнуто захваљујући брзој, флексибилној алокацији и ограниченом трајању послова, што је обезбеђено инкорпорирањем *WB* сервиса у овај оквир. Комплексност Грид инфраструктуре је скривена од корисника, те је он потпуно ослобођен обавезе прибављања слободних ресурса и бављења мидлвером. Аутоматска еластична алокација ресурса кориснику обезбеђује да не чека на ослобађање тачно одређених капацитета, већ да користи оно што је тренутно доступно, али и да заузима додатне ресурсе, чим они постану слободни. Адаптивно заузимање ресурса омогућава осталим корисницима Грида да им брже приступе, што је посебан квалитет WoBinGO оквира.

У циљу процене ефикасности развијених софтверских оквира извршена је теоријска анализа максималног убрзања које се њиховим коришћењем може постићи и установљени практични услови који треба да буду испуњени како би убрзање било остварено. Спроведена емпиријска истраживања омогућила су компаративну анализу перформанси оквира на вишепроцесорском рачунару и кластеру, из које се може закључити да се сваки софтверски оквир може успешно применити за убрза-

вање процеса евалуације јединки за одређену врсту оптимизационих проблема и на одређеној архитектури. Са WoBinGO оквиром је постигнут баланс између добијеног убрзања и степена флексибилности. Иако је достигнуто убрзање нешто мање него код осталих оквира, остварена је флексибилност у складу са захтевима које поставља Грид окружење, у коме се ресурси морају користити рационално, јер их дели велики број корисника.

Додатно емпиријско истраживање над WoBinGO оквиром уз коришћење вештачког тест проблема обављено је са циљем да се утврде перформансе овог оквира када је у питању ефикасно искоришћење Грид инфраструктуре, али и убрзање које се при том достиже. Резултати су показали да се WoBinGO посебно издваја по значајном смањењу времена чекања послова осталих корисника Грида. У поређењу са статичким инфраструктурама заснованим на пилот пословима, WoBinGO омогућава да послови осталих корисника Грида почну да се извршавају више од 10 пута брже. Поред тога, оквир пружа и значајно убрзање поступка оптимизације, посебно у случају дужих евалуација. Тиме је потврђена хипотеза ове дисертације да еластично заузимање ресурса може бити подједнако ефикасно као и код статичких инфраструктура пилот послова које су коришћене у раније развијаним оквирима, али уз кључну предност да се Грид ресурси заузимају само онда када је то заиста потребно.

Коначно, WoBinGO софтверски оквир је искоришћен и у решавању захтевног проблема хидроинформатике - калибрације модела процуривања на хидроелектрани Вишеград. На основу резултата спроведених експеримената анализиране су перформансе које WoBinGO испољава при решавању реалног проблема са нагласком на испољену еластичност у заузимању рачунарских ресурса. Приказани резултати показују да се употребом паралелних ГА за решавање конкретног проблема вишекритеријумске оптимизације у области хидроинформатике знатно убрзава одређивање оптималних решења, чиме је потврђена и друга хипотеза ове дисертације.

Један правац даљег развоја WoBinGO софтверског оквира је његово прилагођавање којим би се омогућила дистрибуирана евалуација јединки у Облаку. Облак омогућава приступ удаљеним рачунарским ресурсима на захтев, за оне клијенте којима је потребно да привремено и у кратком року увећају своје рачунарске капацитете. За оне институције којима су додатни рачунарски ресурси потребни периодично, Облак представља исплативију солуцију од рачунарског кластера, који захтева велика почетна улагања и ангажовање око инсталације и каснијег одржавања. Корисници Облака плаћају услуге провајдерима ресурса само онда када желе да те ресурсе искористе. За разлику од Грида где ресурсе обезбеђују махом научне установе и који се пре свега користе у научне сврхе и потребе индустрије која је у спрези са истраживачим институцијама, Облак је доступан свима. Због свега наведеног, Облак је потенцијално боље решење од Грида за институције којима је повремено потребно да упосле WoBinGO оквир у циљу решавања неког оптимизационог проблема. Отуда је искоришћење WoBinGO еластичности за заузимање инстанци у оквиру услуге *IaaS* (*Infrastructure as a Service*) следећи корак његовог развоја. При-

лагођавање WoBinGO-а за коришћење ресурса у оквиру услуге *IaaS* захтевало би промене одговарајућих конфигурационих датотека *WB* сервиса, као и скриптова за алокацију и покретање послова.

Други могући правац развоја би подразумевао формирање додатног модула WoBinGO софтверског оквира за потребе визуелизације компромисних решења са Парето фронта у случајевима када простор циљних функција има више од три димензије. Док је разумевање добитака и губитака који прате избор појединих решења са Парето фронта у случају дводимензионалног простора циљних функција тривијално, анализа компромисних решења већ код три, а посебно за вишедимензионе просторе бива јако компликована. Један од начина визуелизације Парето решења јесте коришћење самоорганизујућих мапа - COM (*Self-Organizing Map, SOM*). Самоорганизујуће мапе су посебна врста неуронских мрежа које се користе за визуелизацију и интерпретацију скупова високо-димензионих података. COM се састоји од најчешће дводимензионе правилне мреже неурона и користи компетитивно обучавање које подразумева да само један неурон, или један неурон по групи, буде активан на основу тренутног улаза у мрежу. Током процеса обучавања неурони бивају селективно побуђени од стране различитих улазних вектора - Парето решења. Локације побуђених неурона постепено се уређују тако да одговарају дистрибуцији Парето решења, стварајући дводимезиону слику вишедимензионог Парето фронта. Пресликавање вишедимензионог у дводимензиони простор коришћењем COM-а чува топологију простора који се пресликава, тако да суседни неурони COM-а одговарају суседним тачкама из улазног простора.

Поред изнетих идеја за унапређење WoBinGO-а, његова већ присутна интензивна примена у пракси, свакако ће наметнути потребу за додатним развојем како би се испратили захтеви које постављају сложени оптимизациони проблеми хидроинформатике.

Додатак

Додатак А

NSGA-II алгоритам

У овом делу рада ће бити детаљно описан елитистички *NSGA-II* [28] алгоритам који је коришћен у претходним студијама случајева. *NSGA-II* је скраћеница за *Non-dominated Sorting Genetic Algorithm*, то јест генетски алгоритам заснован на сортирању на основу недоминираности решења, док број II означава да је реч о модификацији *NSGA* алгоритма који је предложен у [110]. *NSGA-II* алгоритам је осмишљен са циљем да се превазиђу три особине његовог претходника које су биле критиковане: (а) $\mathcal{O}(MN^3)$ временска сложеност (где је M број функција циља, а N величина популације), (б) неелитистички приступ и (в) потреба да корисник одређује параметар σ_{share}.

Пре описа начина рада *NSGA-II* алгоритма биће приказане специфичне процедуре за сортирање јединки и одржавање разноликости решења које се у алгоритму користе.

А.1 Сортирање популације по скуповима недоминираних решења

NSGA-II алгоритам у раду користи брзо сортирање популације по недоминираним фронтовима. Даље ће бити описано проналажење недоминираног скупа решења примењено у NSGA-II.

Брзим сортирањем по недоминираним фронтовима читава популација се класификује на различите нивое (рангове) недоминираности. За свако решење $\mathbf{x}_i$ у популацији прво се одређују: (а) број решења која доминирају решење $\mathbf{x}_i$ - број доминација, у ознаци n_i и (б) скуп решења која су доминирана решењем $\mathbf{x}_i$, у ознаци S_i. Сва решења за која је $n_i = 0$ биће решења првог нивоа недоминираности и припадаће првом недоминираном фронту у ознаци F_1. Даље се за свако $\mathbf{x}_i$ решење из

F_1, пролази кроз скуп решења S_i и сваком решењу $\mathbf{y}_j$ из S_i се његов број доминација n_j умањује за 1. У случају да је број доминација n_j постао 0, решење $\mathbf{y}_j$ бива издвојено у други недоминирани фронт F_2. Када се ова процедура изврши за сва решења из F_1, процедура се понавља над F_2 ради утврђивања трећег недоминираног фронта. Овај поступак се наставља док сви чланови популације не буду класификовани у неки ниво недоминираности. Важно је нагласити да су решења првог нивоа недоминираности боља од решења другог нивоа, и тако редом.

Алгоритам за брзо сортирање популације по недоминираним фронтовима:

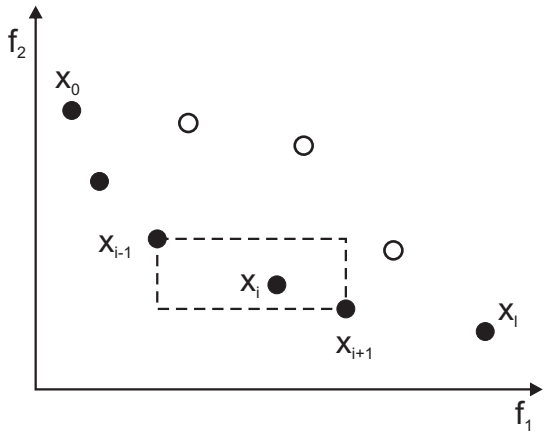
```
1  Za svako resenje x_i ∈ P
2  {
3     n_i := 0,  S_i := ∅
4
5     Za svako resenje x_j ∈ P,  j ≠ i
6     {
7        Ako (x_i ⪯ x_j) onda
8           S_i := S_i ∪ x_j
9        U suprotnom
10          n_i := n_i + 1
11    }
12    Ako (n_i = 0) onda
13       F_1 := F_1 ∪ x_i
14 }
15 k := 1   //brojac frontova
16
17 Dok je F_k ≠ ∅
18 {
19    P' := ∅
20    za svako x_i ∈ F_k
21       za svako x_j ∈ S_i,  j ≠ i
22          n_j := n_j − 1
23          ako (n_j = 0) onda
24             P' := P' ∪ x_j
25    k := k + 1
26    F_k := P'
27 }
```

С обзиром на то да се у делу алгоритма којим се проналазе решења у првом недоминираном фронту свако решење из популације од N решења пореди са свим осталим решењима, и то по свих M цильних функција, овај део алгоритма има сложеност $\mathcal{O}(MN^2)$. Број доминација решења у другом или вишим фронтовима доминације, може бити максимално $N-1$. Дакле, свако од ових решења ће бити разматрано највише $N-1$ пута пре него што број његових доминација постане једнак 0 и оно буде сврстано у неки фронт доминције. Како ових решења која не припадају првом фронту доминације има највише $N-1$, то ће сложеност другог дела алгоритма бити $\mathcal{O}(N^2)$. Следи да је укупна сложеност алгоритма $\mathcal{O}(MN^2)+\mathcal{O}(N^2)$, то јест $\mathcal{O}(MN^2)$.

А.2 Растојање до нагомилавања (*Crowding distance*)

Раније је у делу дисертације 1.4.3 уопштено објашњен метод очувања разноликости решења који је први пут уведен у *NSGA-II* алгоритму. У овом делу ће бити детаљно приказана процедура одређивања растојања од нагомилавања за јединке у популацији. Такође, ће бити представљен поредбени оператор који се заснива на овом растојању, а користи се у сврху сортирања јединки у популацији по густини њиховог окружења.

Да би се одредила густина решења која окружују одређено решење $\mathbf{x}_i$ у популацији, одређује се просечно растојање два најближа решења са сваке стране решења $\mathbf{x}_i$ по свакој функцији циља. Ова величина се назива растојање до нагомилавања, у ознаци d_i. На слици А.1 је дат пример растојања до нагомилавања у дводимензионом случају.



Слика А.1: Приказ израчунавања растојања до нагомилавања у дводимензионом случају

Следећи алгоритам се користи за одређивање растојања до нагомилавања за сваку тачку у скупу F:

```
1 l = |F|   //broj resenja u skupu F
2 Za svako resenje x_i ∈ F
3 {
4    d_i = 0, i = 1,l   //inicijalizovati rastojanja svih resenja
5 }
6 Za svaku funkciju cilja f_m, m = 1,M
7 {
8    I_m = sort(F, f_m, ascending)   /* Skup F sortirati u rastuci poredak po
9 svakoj funkciji cilja, formirajuci tako vektore indeksa I^m, gde ce
      svaki
10 vektor sadrzati indekse svih resenja iz F u redosledu koji odgovara
      rastucem
```

```
redosledu vrednosti funkcije cilja f_m za resenja iz F */
  d_{I_1^m} = d_{I_l^m} = ∞  /*Resenju ciji je indeks prvi clan vektora
I_m kao i resenju ciji je indeks poslednji clan vektora I_m dodeliti
    najvece
rastojanje do nagomilavanja*/
  za j = 2 do l − 1
  {
     d_{I_j^m} = d_{I_j^m} + (f_m^{(I_{j+1}^m)} − f_m^{(I_{j−1}^m)}) / (f_m^max − f_m^min)
  }
}
```

Индекс I_j^m означава индекс j-тог решења у вектору I_m. Отуда, за било коју функцију циља, I_1 и I_l означавају индексе решења са најмањом, односно са највећом вредношћу те функције циља, респективно. Именилац разломка на десној страни последње једнакости представља разлику вредности циљних функција два решења, која су једно с десна, а друго с лева, први суседи решења са индексом I_j. f_{max} и f_{min} су максимална и минимална вредност m-те функције циља у популацији. Дакле, читав разломак је половина обима правоугаоника чија су два дијагонално постављена темена најближи суседи решења са индексом I_j.

Предност приступа са израчунавањем растојања до нагомилавања огледа се у томе што се за одређивање густине популације око одређеног решења не захтева кориснички дефинисан параметар као што је то случај са σ_{share} параметром у *NSGA* методи. Сложеност алгоритма за одређивање растојања до нагомилавања је $\mathcal{O}(MN \log N)$.

А.3 Поредбени оператор заснован на мери растојања до нагомилавања (Crowded Comparison Operator, $<_c$)

Поредбени оператор пореди два решења и враћа боље решење и то коришћењем следећих особина сваког решења $\mathbf{x}_i$:

- ранга недоминираности решења у популацији, у ознаци r_i
- растојања решења до нагомилавања у популацији, у ознаци d_i

Поредбени оператор заснован на мери растојања до нагомилавања се дефинише на следећи начин [25]:

Дефиниција 4 *Решење $\mathbf{x}_i$ је победник турнирске селекције над решењем $\mathbf{x}_j$ ако важи било који од следећа два услова:*

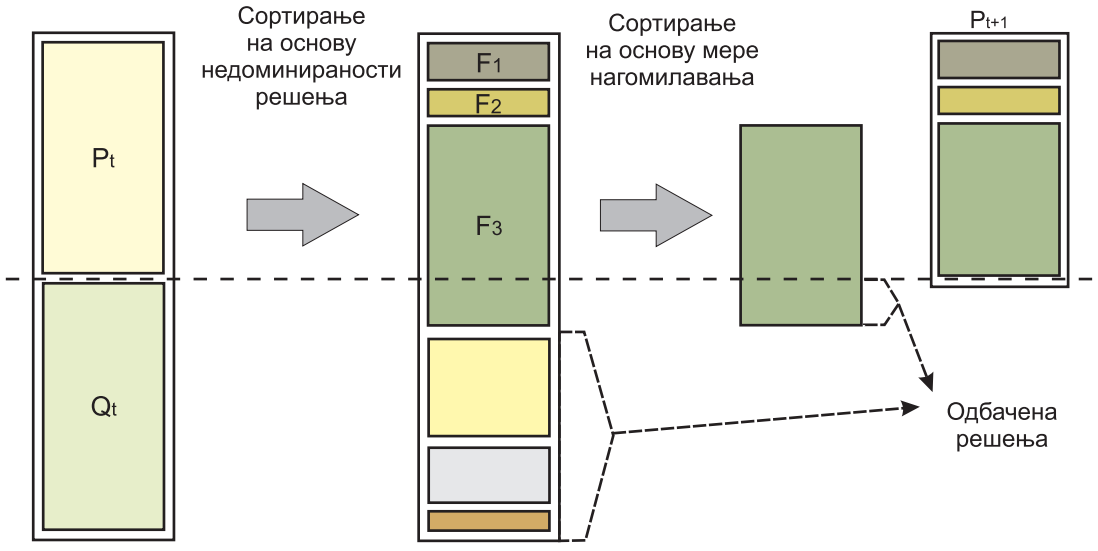
1. *Решење $\mathbf{x}_i$ је боље рангирано од решења $\mathbf{x}_j$, то јест $r_i < r_j$*

2. *Оба решења су истог ранга али решење* $\mathbf{x}_i$ *има веће растојање до нагомилавања у популацији од решења* $\mathbf{x}_j$*, то јест* $r_i = r_j$ *и* $d_i > d_j$*.*

Први услов у дефиницији обезбеђује да одабрано решење припада бољем недоминираном фронту од другог решења. У случају да оба решења припадају истом фронту, други услов се стара да буде одабрано оно решење које се налази у области са мањом густином решења.

А.4 Ток *NSGA-II* алгоритма

NSGA-II користи фиксирану величину популације N. На почетку се креира популација случајно одабраних родитеља P_0 и сортира по недоминираним фронтовима. Сваком решењу се додељује вредност степена прилагођености који је једнак нивоу недоминације тог решења. Под претпоставком да треба минимизовати функције циља, ниво 1 је најбољи и њему припадају решења која нису доминирана, и којима се додељује степен прилагођености 1, нивоу 2 припадају решења која доминира једно решење, и тако редом. Да би се формирала популација потомака Q_0 величине N, користе се оператори турнирске селекције, симулираног бинарног укрштања и полиномне мутације. Елитизам се уводи кроз поређење тренутне популације са претходно пронађеним најбољим недоминираним решењима, тако да се након почетне популације процедура проналажења наредних генерација разликује.



Слика А.2: Скица *NSGA-II* алгоритма

У генерацији t од родитељске популације P_t креира се популација потомака Q_t величине N и у популацији $R_t = P_t \cup Q_t$, величине $2N$, се за свако решење одређује ранг који је једнак нивоу недоминације датог решења. С обзиром на то да се рангирање врши над родитељском и популацијом потомака заједно, елитна решења

из родитељске популације се преносе и у наредну генерацију. Након одређивања недоминираних фронтова $F_1, F_2, \ldots, F_R$ прелази се на формирање наредне популације P_{t+1}. Она се прво попуњава решењима из фронта F_1, затим из фронта F_2 и тако редом. Пошто популација R_t садржи $2N$ решења, у популацију P_{t+1} се не могу пренети решења из свих фронтова. Нека је F_{l-1} последњи фронт из кога су сва решења постала део популације P_{t+1}, и нека би додавањем свих решења из фронта F_l у популацију P_{t+1} била премашена фиксирана величина популације N. Тада се на фронт F_l примењује сортирање по растојању до нагомилавања у опадајућем поретку и оно-лико најбољих решења добијених овим поретком, колико је потребно да би будућа популација имала N јединки, укључује се у будућу популацију. Остала решења из R_t која нису могла бити пренета у наредну генерацију, бивају обрисана. Овај поступак је илустрован на слици А.2. Над популацијом P_{t+1} се даље врши селекција, укрштање јединки и мутација како би била добијена нова популација потомака Q_{t+1}.

NSGA-II алгоритам:

```
1  t := 0, P_0 :=napravi_pocetnu_populaciju , |P_0| = N
2  Q_0 :=napravi_populaciju_potomaka(P_0), |Q_0| = N
3  Sve dok (!kriterijum_za_zaustavljanje)
4  {
5     R_t = P_t ∪ Q_t
6     (F_1, F_2, ..., F_k) :=brzo_sortiranje_po_nedominiranim_frontovima(R_t)
7     P_{t+1} := ∅, i = 0
8     Dok je |P_{t+1}| + |F_i| < N
9     {
10       P_{t+1} := P_{t+1} ∪ F_i
11       i := i + 1
12    }
13    Sort(F_i, <_c)
14    P_{t+1} := P_{t+1} ∪ F_i[0 : N − |P_{t+1}|]   //u populaciju P_{t+1} dodati
15 prvih N − |P_{t+1}| resenja iz sortiranog fronta
16    Q_{t+1} := napravi_populaciju_potomaka (P_{t+1})   //Primeniti binarnu
17 turnirsku selekciju baziranu na operatoru <_c, ukrstanje i mutaciju i
18 formirati populaciju potomaka
19    t = t + 1
20 }
21 return P_t
```

NSGA–II је $\mathcal{O}(MN^2)$ временски сложен. Резултати симулација над бројним тешким тест проблемима показују да *NSGA–II* за већину проблема може да пронађе много бољи опсег решења и решења која боље конвергирају ка правом Парето фронту у односу на PAES [69] и SPEA [127] (два елитистичка алгоритма која посебну пажњу посвећују формирању разноврсног Парето фронта).

Библиографија

[1] Michael B Abbott. Hydroinformatics: a copernican revolution in hydraulics. *Journal of Hydraulic Research*, 32(S1):3–13, 1994.

[2] Michael B Abbott и др. *Hydroinformatics: information technology and the aquatic environment*. Avebury Technical, 1991.

[3] Enrique Alba, Francisco Almeida, M Blesa, J Cabeza, Carlos Cotta, Manuel Díaz, Isabel Dorta, Joaquim Gabarró, Coromoto León, J Luna и др. MALLBA: A library of skeletons for combinatorial optimisation. In *Euro-Par 2002 Parallel Processing*, pages 927–932. Springer, 2002.

[4] Enrique Alba, Gabriel Luque. Evaluation of parallel metaheuristics. *Lecture Notes in Computer Science*, 4193:9–14, 2006.

[5] Enrique Alba, Gabriel Luque, Sergio Nesmachnow. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research*, 20(1):1–48, 2013.

[6] Enrique Alba, Marco Tomassini. Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462, 2002.

[7] Md Atiquzzaman, Shie-Yui Liong, Xinying Yu. Alternative decision making in water distribution network with NSGA-II. *Journal of water resources planning and management*, 132(2):122–126, 2006.

[8] Antun Balaz, Ognjen Prnjat, Dusan Vudragovic, Vladimir Slavnic, Ioannis Liabotis, Emanouil I. Atanassov, Boro Jakimovski, Mihajlo Savic. Development of grid e-infrastructure in south-eastern europe. *CoRR*, abs/1112.4303, 2011.

[9] Wilmer Barreto, Zoran Vojinovic, Roland Price, Dimitri Solomatine. Multiobjective evolutionary approach to rehabilitation of urban drainage systems. *Journal of Water Resources Planning and Management*, 136(5):547–554, 2009.

[10] Elias G Bekele, John W Nicklow. Multi-objective automatic calibration of SWAT using NSGA-II. *Journal of Hydrology*, 341(3):165–176, 2007.

[11] Andrés Bernal, Mauricio A Ramirez, Harold Castro, Jose L Walteros, Andrés L Medaglia. JG^{2}A: A grid-enabled object-oriented framework for developing genetic algorithms. In *Proceedings of the Systems and Information Engineering Design Symposium (SIEDS'09.)*, pages 67–72, 2009.

[12] Christian Blum, Jakob Puchinger, Günther R. Raidl, Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135 – 4151, 2011.

[13] Christian Blum, Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[14] Sébastien Cahon, Nordine Melab, E-G Talbi. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[15] Colin Campbell, Ralph Johnson, Ade Miller, Stephen Toub. *Parallel Programming with Microsoft .NET: Design Patterns for Decomposition and Coordination on Multicore Architectures*. Microsoft Press, 2010.

[16] Erick Cantú-Paz, David E. Goldberg. Efficient parallel genetic algorithms: theory and practice. *Computer Methods in Applied Mechanics and Engineering*, 186(2–4):221–238, 2000.

[17] Min-Rong Chen, Yong-Zai Lu. A novel elitist multiobjective optimization algorithm: Multiobjective extremal optimization. *European Journal of Operational Research*, 188(3):637–651, 2008.

[18] Chun-Tian Cheng, Xin-Yu Wu, KW Chau. Multiple criteria rainfall–runoff model calibration using a parallel genetic algorithm in a cluster of computers. *Hydrological sciences journal*, 50(6), 2005.

[19] Pablo Cibraro, Kurt Claeys, Fabio Cozzolino, Johann Grabner. *Professional WCF 4: Windows Communication Foundation with .NET 4*. Wrox Press Ltd., Birmingham, UK, UK, 2010.

[20] Condor home page. `http://research.cs.wisc.edu/htcondor/`.

[21] Remegio B Confesor, Gerald W Whittaker. Automatic calibration of hydrologic models with multi-objective evolutionary algorithm and pareto optimization. *JAWRA Journal of the American Water Resources Association*, 43(4):981–989, 2007.

[22] David Corne, Joshua D Knowles, Martin J Oates. The pareto envelope-based selection algorithm for multi-objective optimisation. In *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 839–848. Springer-Verlag, 2000.

[23] David W Corne, Nick R Jerram, Joshua D Knowles, Martin J Oates и др. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, CA, 2001. Citeseer.

[24] N.L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proc. of the First International Conference on Genetic Algorithms and their Applications*, pages 183—-187, Carnegie-Mellon University, Pittsburgh, PA, USA, 1985. Spartan Books, Washington.

[25] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[26] Kalyanmoy Deb. Multi-objective optimization. In Edmund K Burke, Graham Kendall, editors, *Search methodologies*, pages 273–316. Springer, 2005.

[27] Kalyanmoy Deb, Ram B Agrawal. Simulated binary crossover for continuous search space. *Complex Systems*, 9(3):1–15, 1994.

[28] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[29] M Dimkic, V Rankovic, N Filipovic, B Stojanovic, V Isailovic, M Pusic, M Kojic. Modeling of radial well lateral screens using 1D finite elements. *Journal of Hydroinformatics*, 15(2):405–415, 2013.

[30] D Divac, Nikola Milivojević, N Grujović, B Stojanović, Zoran Simić. A procedure for state updating of SWAT-based distributed hydrological model for operational runoff forecasting. *Journal of the Serbian Society for Computational Mechanics*, 3(1):298–326, 2009.

[31] Dejan Divac, Dusan Prodanovic, Nikola Milivojevic. Hydro-information systems and management of hydropower resources in serbia. *Journal of the Serbian Society for Computational Mechanics*, 3(1):1–37, 2009.

[32] Marco Dorigo, Thomas Stutzle. Ant colony optimization: Overview and recent advances. In Michel Gendreau, Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 227–263. Springer US, 2010.

[33] Bernabé Dorronsoro, Daniel Arias, Francisco Luna, Antonio J Nebro, Enrique Alba. A grid-based hybrid cellular genetic algorithm for very large scale instances of the CVRP. In *High Performance Computing & Simulation Conference (HPCS)*, pages 759–765, 2007.

[34] Kathryn A. Dowsland. Simulated annealing. In Colin R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 20–69. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[35] Marc Dubreuil, Christian Gagné, Marc Parizeau. Analysis of a master-slave architecture for distributed evolutionary computations. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(1):229–235, 2006.

[36] Juan J. Durillo, Antonio J. Nebro. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[37] Raziyeh Farmani, Godfrey Walters, Dragan Savic. Evolutionary multi-objective optimization of the design and operation of water distribution network: total cost vs. reliability vs. water quality. *Journal of Hydroinformatics*, 8(3):165–179, 2006.

[38] Tiziana Ferrari, Luciano Gaido. Resources and services of the EGEE production infrastructure. *Journal of Grid Computing*, 9(2):119–133, June 2011.

[39] Michael J Flynn, Kevin W Rudd. Parallel architectures. *ACM Computing Surveys (CSUR)*, 28(1):67–70, 1996.

[40] L. J. Fogel, A. J. Owens, M. J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, 1966.

[41] Carlos M Fonseca, Peter J Fleming и др. Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, volume 93, pages 416–423, 1993.

[42] Ian Foster. *Designing and building parallel programs*. Addison-Wesley Reading, 1995.

[43] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

[44] Ian Foster, Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kauffman, San Francisco, 1999.

[45] Ian Foster, Carl Kesselman, Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[46] Christian Gagné, Marc Parizeau. Open BEAGLE: A new versatile C++ framework for evolutionary computation. In *GECCO Late Breaking Papers*, pages 161–168, 2002.

[47] Christian Gagné, Marc Parizeau, Marc Dubreuil. Distributed BEAGLE: An environment for parallel and distributed evolutionary computations. In *Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS)*, volume 2003, pages 201–208. NRC Research Press, 2003.

[48] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533 – 549, 1986.

[49] David E Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison Wesley, 1989.

[50] D Gorgan, V Bacu, D Mihon, D Rodila, K Abbaspour, E Rouholahnejad, N Rebora, H Astsatryan. Grid based calibration of swat hydrological models. *Natural Hazards & Earth System Sciences*, 12(7), 2012.

[51] J-P Goux, Sanjeev Kulkarni, Jeff Linderoth, Michael Yoder. An enabling framework for master-worker applications on the computational grid. In *The Ninth International Symposium on High-Performance Distributed Computing*, pages 43–50, 2000.

[52] Leanne Guy, Peter Kunszt, Erwin Laure, Heinz Stockinger, Kurt Stockinger. Replica management in data grids. In *Global Grid Forum*, volume 5, pages 278–280, 2002.

[53] Mihael Hategan, Justin Wozniak, Ketan Maheshwari. Coasters: uniform resource provisioning and access for clouds and grids. In *Fourth IEEE International Conference on Utility and Cloud Computing (UCC)*, pages 114–121, 2011.

[54] Kejing He, Li Zheng, Shoubin Dong, Liqun Tang, Jianfeng Wu, Chunmiao Zheng. PGO: A parallel computing platform for global optimization based on genetic algorithm. *Computers & Geosciences*, 33(3):357–366, 2007.

[55] Onur Hınçal, A Burcu Altan-Sakarya, A Metin Ger. Optimization of multireservoir systems by genetic algorithm. *Water resources management*, 25(5):1465–1487, 2011.

[56] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor., 1975.

[57] Jeffrey Horn, Nicholas Nafpliotis, David E Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 82–87. Ieee, 1994.

[58] Boye Annfelt Høverstad. Simdist: a distribution system for easy parallelization of evolutionary computation. *Genetic Programming and Evolvable Machines*, 11(2):185–203, 2010.

[59] Hiroaki Imade, Ryohei Morishita, Isao Ono, Norihiko Ono, Masahiro Okamoto. A grid-oriented genetic algorithm for estimating genetic networks by s-systems. In *SICE 2003 Annual Conference*, volume 3, pages 2750–2755, 2003.

[60] Hiroaki Imade, Ryohei Morishita, Isao Ono, Norihiko Ono, Masahiro Okamoto. A grid-oriented genetic algorithm framework for bioinformatics. *New Generation Computing*, 22(2):177–186, 2004.

[61] Milos Ivanovic, Ana Kaplarevic-Malisic, Visnja Simic, Boban Stojanovic. Design and comparison of two web service based frameworks for parallel evaluation of the population in genetic algorithms. *Facta Universitatis, Series: Mathematics and Informatics*, 29(2):155–171, 2014.

[62] Milos Ivanovic, Visnja Simic, Boban Stojanovic, Ana Kaplarevic-Malisic, Branko Marovic. Elastic grid resource provisioning with WoBinGO: A parallel framework for genetic algorithm based optimization. *Future Generation Computer Systems*, 42(0):44 – 54, 2015.

[63] Miloš Ivanović, Boban Stojanović, Nikola Milivojević, Dejan Divac. Dot Net platform for distributed evolutionary algorithms with application in hydroinformatics. In Marijana Despotovic-Zrakic, Veljko Milutinovic, Aleksandar Belic, editors, *Handbook of Research on High Performance and Cloud Computing in Scientific Research and Education*, pages 362–386. IGI Global, 2014.

[64] Anil K. Jain, Lin Hong, Sharath Pankanti. Soil and water assessment tool theoretical documentation. Technical report, USDA Agricultural Research Service and Texas A&M Blackland Research Center, Temple, Texas, 2005.

[65] The jMetal Web Site. `http://jmetal.sourceforge.net/`.

[66] jmetal.NET. a framework for multi-objective optimization (.NET version). `http://jmetalnet.sourceforge.net/`.

[67] Dylan F Jones, S Keyvan Mirrazavi, Mehrdad Tamiz. Multi-objective meta-heuristics: An overview of the current state-of-the-art. *European journal of operational research*, 137(1):1–9, 2002.

[68] Taesoon Kim, Jun-Haeng Heo, Chang-Sam Jeong. Multireservoir system optimization in the Han River basin using multi-objective genetic algorithms. *Hydrological Processes*, 20(9):2057–2075, 2006.

[69] Joshua Knowles, David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1. IEEE, 1999.

[70] M Kojic, N Filipovic, B Stojanovic, V Rankovic, L Krstic, M Ivanovic, M Nedeljkovic, M Dimkic, M Trickovic, M Pusic и др. Finite element modeling of underground water flow with ranney wells. *Water Science & Technology: Water Supply*, 7(3):41–50, 2007.

[71] Joshua B. Kollat, Patrick M. Reed. The value of online adaptive search: A performance comparison of NSGAII, ϵ-NSGAII and ϵMOEA. In CarlosA. Coello Coello, Arturo Hernandez Aguirre, Eckart Zitzler, editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science*, pages 386–398. Springer Berlin Heidelberg, 2005.

[72] Abdullah Konak, David W Coit, Alice E Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

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

[74] S Krcevinac, M Cangalovic, V Kovacevic-Vujicic, M Martic. *Operaciona istraživanja*. Fakultet Organizacionih Nauka–Beograd, 2004.

[75] John W Labadie. Optimal operation of multireservoir systems: State-of-the-art review. *Journal of water resources planning and management*, 130(2):93–111, 2004.

[76] G Lecca, Monique Petitdidier, L Hluchy, M Ivanovic, N Kussul, Nicolas Ray, V Thieron. Grid computing technology for hydrological applications. *Journal of Hydrology*, 403(1):186–199, 2011.

[77] Dudy Lim, Yew-Soon Ong, Yaochu Jin, Bernhard Sendhoff, Bu-Sung Lee. Efficient hierarchical parallel genetic algorithms using grid computing. *Future Generation Computer Systems*, 23(4):658 – 670, 2007.

[78] Steffen Limmer, Dietmar Fey. Framework for distributed evolutionary algorithms in computational grids. In *Advances in Computation and Intelligence*, pages 170–180. Springer, 2010.

[79] Juval Lowy. *Programming WCF services*. O'Reilly Media, 2007.

[80] André Luckow, Lukasz Lacinski, Shantenu Jha. SAGA BigJob: An extensible and interoperable pilot-job abstraction for distributed applications and systems. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 135–144, 2010.

[81] Aristotelis Mantoglou, George Kourakos. Optimal groundwater remediation under uncertainty using multi-objective optimization. *Water resources management*, 21(5):835–847, 2007.

[82] Branko Marović, Milan Potočnik, Branislav Čukanović. Multi-application bag of jobs for interactive and on-demand computing. *Scalable Computing: Practice and Experience*, 10(4), 2001.

[83] AL Medaglia, E Gutiérrez. JGA: An object-oriented framework for rapid development of genetic algorithms. In Jean-Philippe Rennard, editor, *Handbook of Research on Nature Inspired Computing for Economics and Management*, pages 608–624. IDEA Publishing, Hershey, 2006.

[84] Nouredine Melab, Sébastien Cahon, El-Ghazali Talbi. Grid computing for parallel bioinspired algorithms. *Journal of Parallel and Distributed Computing*, 66(8):1052–1061, 2006.

[85] J Quirm Michael. Parallel programming in C with MPI and OpenMP. *Dubuque, IA: McGraw-Hill*, 2004.

[86] Nikola Milivojevic. *Optimizacione metode u simulaciji i upravljanju hidroenergetskim sistemima.* PhD thesis, Univerzitet u Kragujevcu, 2008.

[87] Nikola Milivojević, D Divac, D Vukosavić, Dejan Vučković, Vladimir Milivojević. Computer-aided optimization in operation planning of hydropower plants–algorithms and examples. *Journal of the Serbian Society for Computational Mechanics*, 3(1):273–297, 2009.

[88] Nikola Milivojević, Dejan Divac, Zdravko Stojanović. Računarski podržana optimizacija rada hidroelektrana. In Dejan Divac, Dušan Prodanović, Nikola Milivojević, editors, *Hidroinformacioni Sistemi za Upravljanje Hidroenergetskim Resursima u Srbiji*, pages 335–358. Institut za vodoprivredu "Jaroslav Černi", 2009.

[89] Nikola Milivojević, Zoran Simić, A Orlić, Vladimir Milivojević, B Stojanović. Parameter estimation and validation of the proposed SWAT based rainfall-runoff model–methods and outcomes. *Journal of the Serbian Society for Computational Mechanics*, 3(1):86–110, 2009.

[90] Jakub T Moscicki. Diane - distributed analysis environment for grid-enabled simulation and analysis of physics data. In *IEEE Nuclear Science Symposium Conference Record*, volume 3, pages 1617–1620, 2003.

[91] J.T. Moscicki, F. Brochu, J. Ebke, U. Egede, J. Elmsheuser, K. Harrison, R.W.L. Jones, H.C. Lee, D. Liko, A. Maier, A. Muraru, G.N. Patrick, K. Pajchel, W. Reece, B.H. Samset, M.W. Slater, A. Soroko, C.L. Tan, D.C. van der Ster, M. Williams. Ganga: A tool for computational-task management and easy access to grid resources. *Computer Physics Communications*, 180(11):2303 – 2316, 2009.

[92] A. Munawar, M. Wahib, M. Munetomo, K. Akama. A survey: Genetic algorithms and the fast evolving world of parallel computing. In *10th IEEE International*

Conference on High Performance Computing and Communications (HPCC '08.), pages 897–902, 2008.

[93] Asim Munawar, Mohamed Wahib, Masaharu Munetomo, Kiyoshi Akama. The design, usage, and performance of GridUFO: A grid based unified framework for optimization. *Future Generation Computer Systems*, 26(4):633–644, 2010.

[94] Tadahiko Murata, Hisao Ishibuchi, Hideo Tanaka. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Computers & Industrial Engineering*, 30(4):957–968, 1996.

[95] Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner. *Professional C# 4.0 and .NET 4*. John Wiley & Sons, 2010.

[96] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, Enrique Alba. MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7):726–746, 2009.

[97] Antonio J Nebro, Gabriel Luque, Francisco Luna, Enrique Alba. DNA fragment assembly using a grid-based genetic algorithm. *Computers & Operations Research*, 35(9):2776–2790, 2008.

[98] José Antonio Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, Pablo Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16(3):527–561, 2012.

[99] RK Price, K Ahmad, P Holz. Hydroinformatics concepts. In *Hydroinformatics Tools for Planning, Design, Operation and Rehabilitation of Sewer Systems*, pages 47–76. Springer, 1998.

[100] Ioan Raicu, Yong Zhao, Catalin Dumitrescu, Ian Foster, Mike Wilde. Falkon: a Fast and Light-weight tasK executiON framework. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, 2007.

[101] I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann−Holzboog (Stuttgart), 1973.

[102] Patrick Reed, Barbara Minsker, David Goldberg. A multiobjective approach to cost effective long-term groundwater monitoring using an elitist nondominated sorted genetic algorithm with historical data. *Journal of Hydroinformatics*, 3:71–89, 2001.

[103] Patrick M Reed, David Hadka, Jonathan D Herman, Joseph R Kasprzyk, Joshua B Kollat. Evolutionary multiobjective optimization in water resources: The past, present, and future. *Advances in Water Resources*, 51:438–456, 2013.

[104] E Rouholahnejad, KC Abbaspour, M Vejdani, R Srinivasan, R Schulin, Anthony Lehmann. A parallelization framework for calibration of hydrological models. *Environmental Modelling & Software*, 31:28–36, 2012.

[105] J David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st international Conference on Genetic Algorithms*, pages 93–100. L. Erlbaum Associates Inc., 1985.

[106] Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung.* PhD thesis, TU Berlin, Germany, 1975.

[107] Igor Sfiligoi. glideinWMS - a generic pilot-based workload management system. In *Journal of Physics: Conference Series*, volume 119. IOP Publishing, 2008.

[108] Igor Sfiligoi, Daniel C Bradley, Burt Holzman, Parag Mhashilkar, Sanjay Padhi, Frank Wurthwein. The pilot way to grid resources using glideinwms. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, volume 2, pages 428–432. IEEE, 2009.

[109] Alex Shenfield, Peter J Fleming. A service oriented architecture for decision making in engineering design. In *Advances in Grid Computing-EGC 2005*, pages 334–343. Springer, 2005.

[110] Nidamarthi Srinivas, Kalyanmoy Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248, 1994.

[111] Burke Stephen, Campana Simone, Lanciotti Elisa, Litmaath Maarten, Mendez Lorenzo Patricia, Miccio Vincenzo, Nater Christopher, Santinelli Roberto, Sciaba Andrea. gLite 3.2 user guide. `https://edms.cern.ch/file/722398/ /gLite-3-UserGuide.pdf`, 2009.

[112] B Stojanović, D Divac, Nikola Milivojević, N Grujović, Zdravko Stojanović. State variables updating algorithm for open-channel and reservoir flow simulation model. *Journal of the Serbian Society for Computational Mechanics*, 3(1):327–346, 2009.

[113] B Stojanovic, M Milivojevic, M Ivanovic, N Milivojevic, D Divac. Adaptive system for dam behavior modeling based on linear regression and genetic algorithms. *Advances in Engineering Software*, 65:182–190, 2013.

[114] Boban Stojanovic, Visnja Simic, Milos Ivanovic, Ana Kaplarevic-Malisic, Adam Stanojevic. WCF platform for distributed evaluation in evolutionary algorithms. In *Proceedings of the 4th International Conference Science and Higher Education in Function of Sustainable Development SED 2011*, pages pp. 2:8–13, October 2011.

[115] Grid storage. `https://grid.sara.nl/wiki/index.php/Using_the_Grid/Grid_storage`.

[116] Yong Tang, PM Reed, JB Kollat. Parallelization strategies for rapid and robust evolutionary multiobjective optimization in water resources applications. *Advances in Water Resources*, 30(3):335–353, 2007.

[117] Thread Pooling (C# and Visual Basic). msdn.microsoft.com/en-us/library/n4732ks0.aspx.

[118] Task Parallelism (Task Parallel Library). msdn.microsoft.com/en-us/library/dd537609(v=vs.110).aspx.

[119] David H Wolpert, William G Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[120] S Yalew, A Van Griensven, Nicolas Ray, L Kokoszkiewicz, GD Betrie. Distributed computation of large scale swat models on the grid. *Environmental Modelling & Software*, 41:223–230, 2013.

[121] C. Yann, Siarry. P. *Multiobjective Optimization: Principles and Case Studies*. Springer-Verlag: Berlin, Germany, 2004.

[122] Gary G Yen, Haiming Lu. Dynamic multiobjective evolutionary algorithm: adaptive cell-based rank and density estimation. *Evolutionary Computation, IEEE Transactions on*, 7(3):253–274, 2003.

[123] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32 – 49, 2011.

[124] Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[125] Eckart Zitzler, Marco Laumanns, Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. In *Metaheuristics for multiobjective optimisation*, pages 3–37. Springer, 2004.

[126] Eckart Zitzler, Marco Laumanns, Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. In K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papailiou, T. Fogarty, editors, *Evolutionary Methods for Design, Optimisation, and Control*, pages 19–26. Barcelona, Spain, 2002.

[127] Eckart Zitzler, Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *Evolutionary Computation, IEEE Transactions on*, 3(4):257–271, 1999.

Биографија

Вишња Симић рођена је 1978. године у Крагујевцу. Основну школу је завршила у Крагујевцу као носилац Вукове дипломе, а затим Прву крагујевачку гимназију. Природно-математички факултет Универзитета у Крагујевцу, група математика, смер информатика, уписала је академске 1997/98., а дипломирала је у марту 2002. године са просечном оценом 9.28. Током студија је била добитник стипендије фонда "Академик Драгослав Срејовић", као и Владе Краљевине Норвешке "За генерацију која обећава". У периоду од јуна 2002. године до децембра 2002. била је стипендиста Министарства за науку и технолошки развој Републике Србије по програму за стипендирање младих талената за магистарске студије и њихово укључивање у научно-истраживачке и развојне пројекте Министарства.

Током 2002. године била је ангажована као студент-сарадник у извођењу вежби на предмету Вештачка интелигенција и експертни системи на Институту за математику и информатику Природно-математичког факултета у Крагујевцу, као и у Математичкој радионици младих на предмету Програмски језик PASCAL за талентоване ученике основних школа. Од јануара 2003. до јануара 2004. године била је запослена као истраживач-приправник на Математичком институту САНУ. Од 2004. до 2012. године била је ангажована као асистент-приправник за групу предмета из програмирања на Природно-математичком факултету у Крагујевцу. Од 2012. године је ангажована као асистент за ужу научну област програмирање, на истом факултету.

Последипломске-магистарске студије на групи математика, смер рачунарство, уписала је академске 2002/03. Докторске академске студије у Институту за математику и информатику за стицање научног назива доктор наука – рачунарске науке уписала је академске 2007/08. Положила 15 од укупно 16 предмета предвиђених планом и програмом докторских академских студија са просечном оценом 10.0. До сада је била ангажована на реализацији вежби на неколико студијских програма и на неколико предмета и то: Рачунарство 1, Операциона истраживања, Вештачка интелигенција и експертни системи, Методика наставе математике и информатике (на студијском програму за стицање звања Дипломирани математичар-информатичар); Математика 2 (на студијском програму за стицање звања Дипломирани физичар); Софтверски алати 2, Визуелно програмирање, Интелигентни системи 1, Методика наставе информатике, Рачунарске симулације, Интелигентни системи 2 (на предметима студијског програма за стицање стручног/академског звања Информатичар и Дипломирани информатичар – мастер).

Учествовала је у реализацији следећих пројеката Министарства просвете, науке и технолошког развоја: пројекат број 101379, под називом Методе математичке логике за подршку закључивању у реалним ситуацијама, Математички институт САНУ (2002. - 2006.); пројекат број 144013, под називом Репрезентације логичких структура и примене у рачунарству, Математички институт САНУ (2006. – 2010.);

пројекат број ИИИ-44010, под називом Интелигентни системи за развој софтверских производа и подршку пословања засновани на моделима, ФТН Нови Сад, и пројекат број ТР-35021 Развој триболошких микро/нано двокомпонентних и хибридних самоподмазујућих композита, ФИН Крагујевац (2011. -).

Активно се служи енглеским језиком.

Списак објављених радова

Радови објављени у научним часописима међународног значаја:

1. M. Ivanovic, V. Simic, B. Stojanovic, A. Kaplarevic-Malisic, and B. Marovic. Elastic grid resource provisioning with WoBinGO: A parallel framework for genetic algori thm based optimization. *Future Generation Computer Systems*, 42(0):44 – 54, 2015. ISSN: 0167-739X (M21)

2. M. Ivanovic, A. Kaplarevic-Malisic, V. Simic, and B. Stojanovic. Design and comparison of two web service based frameworks for parallel evaluation of the population in genetic algorithms. *Facta Universitatis, Series: Mathematics and Informatics*, 29(2):155-–171, 2014. ISSN: 0352-9665 (M51)

3. M. Petrovic, T. Aleksic, V. Simic, On the least eigenvalue of cacti, *Linear Algebra and its Applications*, Vol. 435 (10), 2357-2364, 2011. ISSN: 0024-3795 (M22)

4. M. Stefanovic, V. Cvijetkovic, M. Matijevic, V. Simic, A LabVIEW-Based Remote Laboratory Experiments for Control Engineering Education, *Computer Applications in Engineering Education*, Vol. 19 (3), 538-549, 2011. ISSN: 1061-3773 (M23)

5. M. Stefanovic, M. Matijevic, V. Cvijetkovic, V. Simic, Web-Based Laboratory for Engineering Education, *Computer Applications in Engineering Education*, Vol. 18 (3), 526-536, 2010. ISSN: 1061-3773 (M23)

6. M. Stefanovic, M. Matijevic, M. Eric, V. Simic, Method of design and specification of web services based on quality system documentation, *Information Systems Frontiers*, Vol. 11 (1), str. 75-86, 2009. ISSN: 1387-3326 (M22)

Радови штампани у целини у зборницима научних скупова међународног значаја:

1. B. Stojanovic, V. Simic, M. Ivanovic, A. Kaplarevic-Malisic, A. Stanojevic, WCF Platform for Distributed Evaluation in Evolutionary Algorithms, *Proceedings of the 4th International Conference "Science and Higher Education in Function of Sustainable Development" SED 2011*, Uzice, Serbia, 7-8 October 2011, pp. 2:8-13, ISBN 978-86-83573-22-6 (M33)

2. M. Milivojevic, B. Stojanovic, V. Simic, The Simulation of Probability Distribution Function in Queuing Theory, *Proceedings of the 4th International Conference "Science*

and Higher Education in Function of Sustainable Development" SED 2011, Uzice, Serbia, 7-8 October 2011, pp. 2:43-46, ISBN 978-86-83573-22-6 (M33)

3. M. Milivojevic, D. Drndarevic, S. Stopic, V. Simic, B. Stojanovic, Modeling Steel Annealing Process Based on BP Neural Network, *Proceedings of 3rd International Conference "Science and Higher Education in Function of Sustainable Development" - SED 2010*, 7-8 October 2010, Uzice, Serbia pp. 2:22-28, ISBN 978-86-83573-18-9 (M33)

4. Korac M., Ognjanovic Z., Simic V., Dugandzic F., PANDORA - Expert system for dating artifacts, *Proceedings of the ECAI06 Workshop on Intelligent Technologies for Cultural Heritage Exploitation*, Riva del Garda, Italy, August 28 – September 1, (2006), 40-44. M(53)

Future Generation Computer Systems 42 (2015) 44–54

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Elastic grid resource provisioning with WoBinGO: A parallel framework for genetic algorithm based optimization

Milos Ivanovic [a,*], Visnja Simic [a], Boban Stojanovic [a], Ana Kaplarevic-Malisic [a], Branko Marovic [b]

[a] Faculty of Science, University of Kragujevac, Radoja Domanovica 12, 34000 Kragujevac, Serbia

[b] School of Electrical Engineering, University of Belgrade, Bulevar kralja Aleksandra 73, 11120 Belgrade, Serbia

HIGHLIGHTS

- Framework for optimization using parallel GA over Grid and HPC resources.
- Provides elastic resource provisioning avoiding unnecessary occupation of resources.
- Automatic adaptive allocation of jobs with limited lifetime.
- Limited job lifetime provides friendliness towards other batching queue users.
- The complexity of underlying Grid infrastructure is hidden from the user.

ARTICLE INFO

Article history:
Received 9 December 2013
Received in revised form
20 March 2014
Accepted 2 September 2014
Available online 16 September 2014

Keywords:
Grid computing
Pilot-job infrastructure
Dynamic resource provisioning
Metaheuristics based optimization framework

ABSTRACT

In this paper, we present the WoBinGO (**Wo**rk **B**inder **G**enetic algorithm based **O**ptimization) framework for solving optimization problems over a Grid. It overcomes the shortcomings of earlier static pilot-job frameworks, by: (1) providing elastic resource provisioning thus avoiding unnecessary occupation of Grid resources; (2) providing friendliness towards other batching queue users thanks to adaptive allocation of jobs with limited lifetime. It hides the complexity of the underlying Grid environment, allowing the users to concentrate on the optimization problems. Theoretical analysis of possible speed-up is presented. An empirical study using an artificial problem, as well as a real-world calibration problem of a leakage model at the Visegrad power plant were performed. The obtained results show that despite WoBinGO's adaptive and frugal allocation of computing resources, it provides significant speed-up when dealing with problems that have computationally expensive evaluations. Moreover, the benchmarks were performed in order to estimate the influence of the limited job lifetime feature on the queuing time of other batching jobs, compared to a static pilot-job infrastructure.

1. Introduction

Grid computing [1] has emerged as an effective environment for the execution of parallel applications that require great computing power. Grid computing consists of a geographically distributed infrastructure gathering computer resources around the world in a transparent way. Users are provided access to enormous computing resources, and enabled to better meet the challenges of science and engineering. One of the most frequently encountered challenges in applied science and engineering, is optimization. Genetic algorithms (GAs) [2] have proven themselves as robust and powerful mechanisms when it comes to solving complex real-world optimization problems. GA is characterized by a large number of function evaluations. Due to the time-consuming fitness evaluation functions found in real-world problems, it may take days and months for the GA to find an acceptable solution. Speeding up the optimization process is achieved by parallelization of GA [3–5] which reduces the resolution times to reasonable levels. Grid computing environments provide the infrastructure for implementing parallel metaheuristics. There, researchers face new difficulties associated with developing and deploying a Grid based application. The fact that Grid resources are distributed, heterogeneous and non-dedicated, makes writing parallel Grid-aware applications very challenging [6]. The development and

* Corresponding author. Tel.: +381 34 336223; fax: +381 34 335040.
E-mail addresses: mivanovic@kg.ac.rs (M. Ivanovic), visnja@kg.ac.rs (V. Simic), bobi@kg.ac.rs (B. Stojanovic), ana@kg.ac.rs (A. Kaplarevic-Malisic), branko.marovic@rcub.bg.ac.rs (B. Marovic).

http://dx.doi.org/10.1016/j.future.2014.09.004
0167-739X/

FACTA UNIVERSITATIS (NIŠ)
SER. MATH. INFORM. Vol. 29, No 2 (2014), 155–171

DESIGN AND COMPARISON OF TWO WEB SERVICE BASED FRAMEWORKS FOR PARALLEL EVALUATION OF THE POPULATION IN GENETIC ALGORITHMS *

Miloš Ivanović, Ana Kaplarević-Mališić, Višnja Simić and Boban Stojanović

Abstract. Genetic algorithms are powerful techniques for optimization of complex systems. These methods require a large number of evaluations of candidate solutions which take huge CPU time. This paper introduces two web service based frameworks for parallel evaluation of the population in genetic algorithm using the master-slave model. Developed frameworks can be easily incorporated into any genetic algorithm, giving a universal mechanism for distribution of individuals and collection of the evaluation results. This concept provides parallelization of genetic algorithms on various distributed architectures, including multiprocessors and computing clusters. Performed tests have shown that proposed frameworks achieve significant speedup, especially when evaluating large-scale problems. In addition, a case study from the field of hydrology is presented.
Key words: Parallel computing, Genetic algorithm, Optimization, Web service, Hydrology

1. Introduction

Genetic algorithms (GAs), as a major class of Evolutionary algorithms (EAs), have proven themselves as a robust and powerful mechanism when it comes to solving challenging optimization problems [1, 2]. They mimic the process of natural evolution, by modifying the set of potential solutions, called population, through selection, crossover and mutation of individuals. In order to select the best candidates for reproduction, one has to evaluate the fitness of each individual in the population. Solving the optimization problem using a genetic algorithm requires evaluation of hundreds of individuals through several tens of generations. Since each evaluation in real-world problems usually requires running a complex, time consuming computer simulation, the whole optimization process can last for hours or even days.

Received March 12, 2014.; Accepted Jun 03, 2014.
2010 *Mathematics Subject Classification.* C.2.4; G.1.6; J.2
*The part of this research is supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia (Grants III41007, OI174028, TR37013, III44010, and TR 14005, and FP7 ICT-2007-2-5.3 (224297) ARTreat project.