УНИВЕРЗИТЕТ У БЕОГРАДУ
ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

Милош Д. Даниловић

УНАПРЕЂЕЊЕ КОНСТРУКТИВНИХ ХЕУРИСТИКА ЗА ПРОБЛЕМЕ КОМБИНАТОРНЕ ОПТИМИЗАЦИЈЕ У ОПЕРАЦИОНОМ МЕНАЏМЕНТУ

Докторска дисертација

Београд, 2017.

UNIVERSITY OF BELGRADE
FACULTY OF ORGANIZATIONAL SCIENCES

Miloš D. Danilović

IMPROVEMENT OF CONSTRUCTIVE HEURISTICS FOR COMBINATORIAL OPTIMISATION PROBLEMS IN OPERATIONS MANAGEMENT

Doctoral Disertation

Belgrade, 2017.

МЕНТОР:
Др Оливер Илић, редовни професор
Факултет организационих наука, Универзитет у Београду

ЧЛАНОВИ КОМИСИЈЕ:
Др Мирјана Чангаловић, редовни професор у пензији
Факултет организационих наука, Универзитет у Београду

Др Мирко Вујошевић, редовни професор
Факултет организационих наука, Универзитет у Београду

Др Драган Васиљевић, редовни професор
Факултет организационих наука, Универзитет у Београду

Др Обрад Бабић, редовни професор
Саобраћајни факултет, Универзитет у Београду

Датум одбране:_____2017. године.

Пре свега, захваљујем се проф. др Оливеру Илићу што ме је увео у предиван свет истраживачког рада и што је својом стручношћу и ауторитетом одредио моје професионално опредељење ка правцима који за мене сада немају алтернативу.

Захваљујем се проф. др Драгану Васиљевићу на поверењу и подршци без којих би све било много теже.

Захваљујем се др Биљани Цветић чији су пријатељска подршка и конструктивни савети помогли да постанем део једног посебног тима.

Хвала Вани и Дачи на бескрајној родитељској љубави и подршци.

Тинина љубав и разумевање заокружују листу која је суштински утицала на израду ове дисертације.

УНАПРЕЂЕЊЕ КОНСТРУКТИВНИХ ХЕУРИСТИКА ЗА ПРОБЛЕМЕ КОМБИНАТОРНЕ ОПТИМИЗАЦИЈЕ У ОПЕРАЦИОНОМ МЕНАЏМЕНТУ

Резиме: Операциони менаџер користи скуп поступака чији је циљ да се послови ураде брже, јефтиније и квалитетније. Научници из области операционог менаџмента имају задатак да ови поступци буду изводљиви и практични. Скоро увек, менаџери покушавају да нешто оптимизују – или је то минимизација трошкова и потрошње енергије, или пак, максимизација профита, резултата, перформанси и ефикасности. Међутим, није увек могуће пронаћи оптимална решења. У пракси, менаџер мора да се задовољи решењима која можда нису оптимална, али су допустива, задовољавајућа, робустна, и достижна у разумном времену. Оваква решења се добијају применама хеуристика, које могу бити конструктивне, побољшавајуће или хибридне. Област истраживања у докторској дисертацији су конструктивне хеуристике за проблеме комбинаторне оптимизације у операционом менаџменту који припадају класи сложености НП. Представљен је нови генерализовани конструктивни алгоритам који омогућава да се разноврсне хеуристике формирају избором његових аргумената. Такође је уведено опште окружење за генерисање пермутација, које формира везу између енумерације пермутација и корака у конструктивним хеуристикама уметања. Предложен је скуп аргумената генерализованог алгоритма који омогућује паралелно праћење више парцијалних решења за време извршавања алгоритма. Могућности и предности генерализованог алгоритма су представљене кроз његову примену на проблем формирања ћелија у производним системима, проблем распореда производних ћелија и проблем редоследа послова у линији. Нови приступ даје решења која на испитиваним примерима надмашују најбоље познате резултате из литературе.

Кључне речи: НП-комплетни проблеми; пермутације; партиције; проблем формирања производних ћелија; проблем распореда производних ћелија; проблем редоследа послова у линији

Научна област: Операциони менаџмент

Ужа научна област: Рачунарски интегрисана производња и логистика

УДК број: 658.5:004.382

IMPROVEMENT OF CONSTRUCTIVE HEURISTICS FOR COMBINATORIAL OPTIMISATION PROBLEMS IN OPERATIONS MANAGEMENT

Abstract: Operations manager deals with a collection of methods for getting things done more quickly, more cheaply or to a higher standard of quality. It is the job of the management scientist to make sure that these methods are practical and relevant. Almost always managers try to optimize something - whether to minimize the cost and energy consumption, or to maximize the profit, output, performance and efficiency. Subsequently, it is not always possible to find the optimal solutions. In practice, managers have to settle for suboptimal solutions or even feasible ones that are satisfactory, robust, and practically achievable in a reasonable time scale. These kind of solutions are obtained with heuristics, which can be constructive, improvement heuristics or hybrid. The field of research in the doctoral thesis are constructive heuristics for NP-hard combinatorial optimization problems in operations management. A new generalized constructive algorithm is presented which makes it possible to select a wide variety of heuristics just by the selection of its arguments values. A general framework for generating permutations of integers is presented. This framework forms a link between the numbering of permutations and steps in the insertion-based heuristics. A number of arguments controlling the operation of the generalized algorithm tracking multiple partial solutions, are identified. Features and benefits of the generalized algorithm are presented through the implemetations to the Cell Formation Problem, the Quadratic Assignment Problem and the Permutation Flowshop Problem. The new approach produces solutions that outperform, on the tested instances, the best known results from literature.

Keywords: NP-complete problems; Permutations; Partitions; Cell Formation Problem; Quadratic Assignment Problem; Permutation Flowshop Problem

Academic Expertise: Operations management
Major in: Computer integrated manufacturing and logistics

UDK: 658.5:004.382

Садржај

Списак Табела

Списак Слика

1. УВОД

Главни задатак менаџера је доношење најбољих одлука које доприносе унапређењу пословања. Овај задатак укључује стратегијске и тактичке одлуке везане за концепт циљева компаније, за тржиште, за производе и средства, али и за оперативне одлуке којима се контролишу постојећи логистички процеси. Менаџмент је суочен, нарочито у областима производње и логистике, са сложеним одлукама које се заснивају на решењима постављених сложених проблема. Често, ови проблеми могу бити представљени математичким моделима који намећу кључно питање: како структуирати процес решавања проблема представљеног тим математичким моделом.

Сложени проблеми подразумевају широк спектар разноврсних поступака, тј. алгоритама за њихово решавање. Различити алгоритми дају различит квалитет решења и захтевају различите ресурсе за њихово израчунавање. У великом броју случајева се захтева да решења буду оптимална у односу на неку дефинисану меру квалитета решења. Да би се дошло до оптималног решења, алгоритам треба да претражи допустива решења и да одабере најбоље. Обим претраге допустивог скупа дефинише кључну карактеристику проблема и алгоритама: сложеност.

Теорија сложености оперише са класама, односно пребројивим скупом инстанци проблема. За дати алгоритам или математички модел, сложеност је карактерисана зависношћу времена односно меморијског заузећа, потребних да се реши проблем за дату инстанцу у функцији величине те инстанце. Због аналитичке доследности, временска и просторна сложеност се анализирају у смислу асимптотског понашања у најгорем случају. Веза између сложености проблема и сложености алгоритама је дефинисана тако што се сложеност проблема поистовећује са сложеношћу најбољег познатог алгоритма за тај проблем. Пошто је у општем случају временска сложеност рестриктивнији фактор од просторне сложености, проблеми се најчешће категоризују у класе према њиховој асимптотској временској сложености.

Како је функција сложености проблема везана за сложеност тренутно најбољег познатог алгоритама за решавање тог проблема, јасно је да математичка

формулација ове функције садржи субјективан елемент: тренутно стање истраживања на том проблему. Због овога, прецизно класификовање сложености проблема није нимало једноставно. Да би се на неки начин превазишла ова суштинска некохерентност, класификација сложености се спроводи на следећи начин: прва класификација проблема је заснована на чињеници да ли за тај проблем постоји познат полиномијални алгоритам. Уколико такав алгоритам постоји, проблем припада класи П и без обзира на будуће алгоритме и развој хардвера, увек ће припадати класи П. Унапређење ефикасности алгоритама за решавање ових проблема се постиже смањењем реда полинома.

Ситуација је много сложенија са проблемима који тренутно нису у П. Ови проблеми нису у П из једног од два разлога: први разлог може бити да још није пронађен алгоритам који овај проблем решава у полиномијалном времену, а други, да је проблем такав да не може постојати алгоритам који га решава у полиномијалном времену. Како до данас ни за један проблем није доказано да није решив у полиномијалном времену, било је неопходно да се уведу критеријуми који би омогућили да се разврстају проблеми за које још увек не постоје полиномијални алгоритми. Ове критеријуме је дефинисала теорија сложености израчунавања чије основе су постављене у раду Кука, објављеног 1971. године под насловом “The Complexity of Theorem Prooving Procedures” (Cook, 1971). Предмет истраживања у дисертацији ће бити проблеми из класе НП комплетних проблема. Фокус истраживања у дисертацији ће бити поступци за решавање неких комбинаторних оптимизационих проблема.

Према (Cvetkovic, Čangalović, Dugošija, Kovačević-Vujčić, Simić, & Vuleta, 1996), комбинаторна оптимизација је математичка дисциплина која проучава проблеме налажења екстремних вредности фунцкије дефинисане на коначном или пребројиво бесконачном, дискретном скупу. Додавање услова дискретности чини многе проблеме оптимизације неупоредиво тежим (Vujošević, 2012). Када за решавање ових проблема не постоје егзактни полиномијални алгоритми, аутори предлажу **хеуристике**, чији је циљ да у задовољавајућем времену дају што је могуће квалитетнија решења. У хеуристикама за решавање проблема комбинаторне оптимизације се операције изводе над пермутацијама, комбинацијама или варијацијама почетног скупа објеката. Како се и комбинације и варијације могу да

формулишу помоћу пермутација, акценат истраживања у дисертацији је усмерен на хеуристике за решавање **пермутационих оптимизационих проблема**. Све ове хеуристике се међусобно разликују само по начину на који генеришу те пермутације и по критеријуму за избор жељених пермутација из скупа свих могућих пермутација. Према начину на који се генеришу пермутације, ове хеуристике могу да се поделе на конструктивне хеуристике, хеуристике побољшања и хибридне хеуристике. У конструктивним хеуристикама се, полазећи од једног одабраног објекта, у свакој следећој итерацији број одабраних објеката увећева за један. Хеуристике побољшања пермутују делимичан или комплетан скуп објеката у потрази за оптималном пермутацијом. Заједничка карактеристика већине конструктивних хеуристика је њихова изузетна временска ефикасност, тј. оне имају занемарљив утрошак CPU времена у поређењу са хеуристикама побољшања на истој инстанци проблема. У поступку конструкције решења је могуће симултано генерисати више од једног решења која имају међусобно једнаку вредност циљне функције или вредности циљне функције унутар одређеног опсега вредности. У таквим случајевима се добијене пермутације могу да користе као почетна популација за хеуристике побољшања.

Конструктивне хеуристике представљају саставни део великог броја познатих хеуристика. У великом броју поступака за решавање проблема комбинаторне оптимизације се на неки начин имплементира одређени конструктивни поступак за добијање скупа почетних решења. Ефикасност и ефективност ових хеуристика има значајну улогу у укупној вредности поступака којима се решавају најразноврснији проблеми комбинаторне оптимизације. Задатак конструктивне хеуристике је да из допустивог скупа решења датог проблема издвоји што је могуће квалитетнији подскуп решења у смислу вредности циљне функције. У складу са тим, конструктивне хеуристике се међусобно пореде на основу добијених вредности циљне функције и сложености поступака који те хеуристике спроводе.

Иако постоји огроман број различитих конструктивних хеуристика за примену у решавању најразноврснијих проблема, највећи број тих хеуристика има веома сличну структуру. Слична структура ових хеуристика намеће потребу да се поступци генерисања и избора жељених пермутација генерализују. Генерализовани

поступак би омогућавао да се велики број постојећих хеуристика представи једним алгоритмом са различитим вредностима својих аргумената.

Иако је очигледно да су могуће предности генерализације конструктивних хеуристика огромне, у литератури није било покушаја да се дефинише било каква генерализација. Чак и без дубље анализе могућности које отвара постојање генерализованог поступка, неке предности могу да се наведу *a priori*. Као прво, генерализација алгоритама повећава објективност евалуације, јер се уместо комплексних алгоритама, евалуација врши на мањим, међусобно независним деловима алгоритама. Као друго, отвара се могућност побољшања постојећих хеуристика изменом вредности аргумената генерализованог поступка.

У литератури постоје хеуристике које се декларишу као конструктивне, у којима се, током извршавања поступка, већ одабрани објекти избацују из оформљене пермутације и накнадно, у следећим корацима поново уносе у њу. Суштински, ове хеуристике нису праве конструктивне хеуристике, већ неки хибрид конструктивних хеуристика и хеуристика побољшања. Праве конструктивне хеуристике се по ефикасности и ефективности суштински разликују од ових хибридних хеуристика и хеуристика побољшања. У дисертацији се разматрају **праве конструктивне хеуристике**, које ће бити означене као ПКХ.

Кључна предност ПКХ је њихова полиномијална временска сложеност. Ово је најважнија особина која проистиче из саме дефиниције типа хеуристике. У следећим поглављима ће ова тема бити прецизно елаборирана. Евалуација поступака за решавање проблема комбинаторне оптимизације се спроводи поређењем добијених експерименталних резултата предложеног алгоритма са најбољим познатим експерименталним резултатима у литератури на истим тест инстанцама. Пореде се одступања вредности циљних функција добијених различитим алгоритмима од најбољих познатих одговарајућих вредности као и времена рада рачунара утрошених на добијање решења различитим алгоритмима. За разлику од објективне евалуације одступања добијених вредности, поређење временске ефикасности је у огромном броју случајева крајње провизорно и субјективно, тако да евалуација временске ефикасности предложених алгоритама представља, можда и најслабију карику у поступцима презентације нових алгоритама. По правилу, евалуација

временске ефикасности за хеуристике побољшања се спроводи тако што аутор испрограмира све хеуристике са којима се пореди и измери времена рада тих хеуристика на тест инстанцама. Јасно је да цео поступак увелико зависи од способности и објективности програмера. Насупрот овоме, евалуација полиномијалних алгоритама је много прецизнија: ред полинома је апсолутни фактор квалитета хеуристике. Само у случајевима када два алгоритма имају исти ред полиномијалне временске зависности има смисла експериментално поређење тих алгоритама.

Јасно је да је тражење компромиса (у наставку означен као *trade-off*) полиномијалне ефикасности ПКХ плаћен слабијим резултатима у погледу вредности циљне функције у односу на хеуристике побољшања. Циљ дисертације је да истражи могућности за побољшање већ постојећих конструктивних хеуристика уз задржавање њихове оригиналне полиномијалне временске сложености. Правац тог побољшања, који се природно намеће, је формализација вишеструке примене исте конструктивне хеуристике и усвајање најбољег од добијених решења. Вишеструка примена не мења ред полинома временске сложености и изузетно је подесна за паралелно програмирање. Ова побољшања су значајна из следећих разлога:

1. За велики број проблема комбинаторне оптимизације је показано у литератури да су динамичке природе, тј. да је изузетно битно да се решење нађе уз што мањи утрошак CPU времена;
2. Полиномијална ефикасност алгоритма омогућава да се унапред прецизно контролише *trade-off* између утрошеног CPU времена и квалитета решења, што код хеуристика побољшања често није могуће;
3. У великом броју случајева из реалног живота вредност циљне функције се израчунава на основу мерења чија је прецизност у неком реалном интервалу поузданости. За такве случајеве је непотребно мерење одступања циљне функције у интервалима ужим од тих интервала поузданости. На пример, растојања између градова дефинишу величине које најчешће имају прецизност мању од једног километра. У неким радовима у којима се пореде алгоритми за најкраће путање на тест инстанцама путних мрежа у Америци, поређења се спроводе на резултатима који се разликују у метрима, што је

потпуно непотребно. У тим случајевима ПКХ добијају на значају уколико њихова одступања у односу на хеуристику побољшања упадају у ове интервале;

4. Квалитетне ПКХ могу да дају решења која ће да послуже као почетна популација хеуристикама побољшања ради ефикаснијег добијања квалитетнијих решења. Овде треба нагласити једну важну чињеницу. Наиме, познато је да у великом броју случајева хеуристике побољшања дају боља решења уколико имају случајно одабрану почетну популацију у односу на неку софистицирану почетну популацију. Ова констатација је тачна у великом броју случајева, осим у једном. Наиме, хеуристике побољшања, по правилу, раде најбоље уколико је почетна популација у уској околини оптималног решења. Значи, квалитетне ПКХ, које су по квалитету решења близу хеуристикама побољшања су итекако погодне за добијање почетне популације.

Најзад, посебан правац истраживања у дисертацији је везан за главни недостатак свих конструктивних хеуристика, заглављивање поступка у локалном оптимуму. Овај проблем је уочен и решен у хеуристикама побољшања и представља суштинску предност хеуристика побољшања у односу на ПКХ. Посебно квалитетан поступак удаљавања од локалног оптимума је дефинисан хеуристиком симулираног каљења, прецизније, хеуристика симулираног каљења *de facto* представља поступак избегавања локалног оптимума. Паралелна обрада више пермутација генерализованим конструктивним алгоритмом отвара непосредну могућност примене поступка симулираног каљења на ПКХ уз одржавање исте временске сложености алгоритма.

1.1. Предмет, проблеми, циљ и хипотезе докторске дисертације

Предмет истраживања дисертације су ПКХ и поступци за енумерацију пермутација, док су проблеми истраживања:

- дефинисање релација које постоје између поступака за генерисање пермутација и корака ПКХ;
- генерализација ПКХ са циљем формализације паралелне обраде;

- примена генерализованог конструктивног алгоритма за решавање одабраних проблема операционог менаџмента;
- примена ПКХ у новом хибридном алгоритму за проблем формирања производних ћелија који користи поступке за сужавање допустивог скупа у циљу унапређења познатих хеуристика за решавање овог проблема.
- нова ПКХ за решавање пермутационог проблема редоследа послова у проточној радионици (у наставку означен као *flowshop* проблем), која користи паралелну обраду парцијалних секвенци у циљу побољшања најбољих постојећих конструктивних хеуристика за тај проблем.

Истраживања, спроведена при изради дисертације, су заснована на експерименталној провери хипотеза, постављених у приступном раду. Разматрана је могућност примене генерализованог конструктивног алгоритма за решавање проблема распореда производних ћелија. Такође, конструисан је хибридни алгоритам за проблем формирања производних ћелија у коме је као једина хеуристика примењен генерализовани конструктивни алгоритам. Циљ експерименталне анализе у дисертацији је да потврди предности овог хибридног алгоритма у односу на најбоље познате хеуристике из ове области. Као проблем, за који ће бити детаљно приказан поступак формирања алгоритма који је бољи од познатих конструктивних алгоритама, одабран је пермутациони *flowshop* проблем и то из три разлога: прво, овај проблем представља један од најпроучаванијих проблема комбинаторне оптимизације, те постоји огроман број радова у којима се предлажу конструктивни алгоритми за његово решавање; друго, у великом броју радова из ове области се користи чувена НЕХ конструктивна хеуристика (Nawaz, Enscore Jr, & Ham, 1983) која представља прототип ПКХ; најзад, за евалуацију алгоритама из ове области постоје опште прихваћене тест инстанце, што је од великог значаја за објективну евалуацију алгоритама.

У дисертацији треба да се докажу или оспоре три опште хипотезе. За прву општу хипотезу треба да се докажу три посебне хипотезе, за другу општу хипотезу две посебне хипотезе и за трећу општу хипотезу шест посебних хипотеза.

1.1.1 *Опште хипотезе*

Х(1) ПКХ за решавање пермутационих проблема комбинаторне оптимизације имају сличну структуру, погодну за генерализацију.

Х(2) Могуће је дефинисати поступак за генерисање пермутација који је у узајамној једнозначној кореспонденцији са корацима конструктивног алгоритма.

Х(3) Генерализовани конструктивни алгоритам отвара значајне могућности побољшавања постојећих ПКХ и генерисање нових за решавање пермутационих проблема комбинаторне оптимизације.

1.1.2 *Посебне хипотезе*

Х(11) Све ПКХ се састоје од две фазе, фазе иницијализације и фазе уметања.

Х(12) Све ПКХ користе три критеријума: критеријум селекције почетног објекта; критеријум избора објекта који се умеће и критеријум избора позиције на коју се одабрани објекат умеће.

Х(13) Све ПКХ се међусобно разликују само у дефиницији три критеријума селекције.

Х(21) Постојећи поступци за енумерацију пермутација немају никакву кореспонденцију са корацима конструктивних алгоритама.

Х(22) Постојећи поступци за енумерацију пермутација, засновани на лексикографском уређењу пермутација, нису подесни за ефикасно генерисање пермутација.

Х(31) Велики број постојећих конструктивних хеуристика има непрецизну формулацију, јер се детаљно објашњавају кораци који су заједнички за све конструктивне хеуристике, а површно дефинишу кораци по којима се те хеуристике разликују од осталих.

Х(32) Данашњи ниво објективности евалуација хеуристика је веома низак.

Х(33) Генерализовани конструктивни алгоритам подиже ниво прецизности формулације конструктивних алгоритама и суштински поправља објективност евалуације конструктивних хеуристика.

Х(34) Генерализовани конструктивни алгоритам проширује могућности прилагођавања поступка конкретним инстанцама на које се примењује.

Х(35) Генерализовани конструктивни алгоритам омогућава унапређење квалитета постојећих алгоритама подешавањем вредности својих аргумената.

Х(36) Генерализовани конструктивни алгоритам омогућава једноставно генерисање нових алгоритама изменом вредности својих аргумената.

1.2. Структура докторске дисертације

У уводном делу докторске дисертације описан је проблем истраживања, као и начин на који ће се анализирати постављени проблем истраживања. Дефинисани су предмет, циљ и хипотезе, које су касније у раду разматране са аспекта прегледа и анализе постојећих знања и литературе из области релевантних за тему рада. Уводни део дисертације у кратким цртама приказује и описује структуру рада, кратким прегледом сваког поглавља.

Класе сложености проблема су разматране у другом поглављу дисертације. Циљ овог разматрање је да се прецизно дефинишу појмови недетерминистичке машине и псеудо-полиномијалних алгоритама. Генерализација конструктивних алгоритама, предложена у овом раду, симулира функцију недетерминистичке машине применом паралелне обраде више парцијалних решења. Псеудо-полиномијални алгоритми, представљени у овом поглављу, представљају циљни облик хеуристика које се предлажу у дисертацији. Главни алат који усмерава нове хеуристике ка псеудополиномијалним алгоритмима је дефинисање поступака за сужавање допустивог скупа решења.

Трећи део разматра структуру конструктивних хеуристика и дефинише суштинску разлику између правих и хибридних конструктивних хеуристика. На основу ових разматрања дефинише се прототип ПКХ. Временска ефикасност представља кључни

параметар свих хеуристика. У четвртом поглављу се хеуристике пореде по временској ефикасности и погодности да се имплементирају у паралелном програмирању.

Недостаци најпознатијих ПКХ објављених у стручној литератури су разматрани у петом делу рада у коме су приказане ПКХ за решавње најразноврснијих проблема операционог менаџмента. Обухваћени су поступци из најновијих радова у најугледнијим светским часописима. И поред чињенице да се ради о радовима најуваженијих аутора разматраних области, могу да се уоче значајни недостаци у формулацијама и евалуацији ових алгоритама. Разматрани су редудантност, непрецизност и вишезначност формулације као и необјективност поређења вредности циљне функције и необјективност евалуације временске ефикасности.

У шестом поглављу је представљена генерализација конструктивних хеуристика која је заснована на вишеструкој обради парцијалних решења, док седми део рада уводи нови поступак за енумерацију пермутација и нови поступак за генерисање пермутација, базиран на структури ПКХ.

У циљу верификације постављених хипотеза на конкретним проблемима операционог менаџмента одабрани су проблем формирања производних ћелија, проблем распореда производних ћелија и проблем редоследа послова у линији. Примена новог приступа на ове проблеме је дата, редом у поглављима 8, 9 и 10. Треба посебно истаћи да је поступак за дефинисање нових алгоритама за ове проблеме усмераван ка структурама недетерминистичког алгоритма и псеудо-полиномијалног алгоритма. Тако, паралелно праћење више парцијалних решења проблема редоследа послова у линији симулира избор који спроводи недетерминистичка машина, док дискретизација решења за проблем формирања производних ћелија усмерава поступак ка псеудо-полиномијалној структури.

Експериментални резултати, представљени у поглављима 8, 9 и 10 су показали да је проблем формирања производних ћелија изузетно погодан за предложени приступ и да нови алгоритам даје значајно боље резултате од најбољих објављених резултата. Затим је показано да пермутациона верзија проблема распореда производних ћелија може ефикасно да се прилагоди за примену генерализованог алгоритма. Најзад,

проблем редоследа послова у линији је искоришћен да се на њему покажу разноврсне могућности генерализованог алгоритма за експериментисање у циљу креирања нових, квалитетнијих алгоритама. Експериментални резултати су показали да предложени алгоритам у полиномском времену даје резултате који су у рангу најбољих објављених хеуристика за овај проблем.

За разлику од проблема формирања производних ћелија и проблема редоследа послова у линији, за које су у дисертацији експериментална поређења обухватила све најбоље алгоритме, експериментални резултати за проблем распореда производних ћелија имају другачију намену, и то из два разлога. Први разлог је обимност: сва три обрађивана проблема представљају изузетно значајне проблеме за које постоји огроман број радова у литератури. Добијени резултати за три изабрана проблема су толико обимни и разноврсни да је било неопходно, због обима дисертације, да се скуп резултата сузи на минималан подскуп којим се експериментално верификују све хипотезе дисертације. Због тога је одабрано да се експерименталним резултатима за проблем формирања производних ћелија покаже да је нови алгоритам бољи од свих постојећих алгоритама, а да се на проблему редоследа послова у линији покаже како се врши експериментално подешавање вредности алгоритама у циљу формирања полиномијалног алгоритма чији су резултати у рангу најбољих резултата за тај проблем. Други разлог је специфичност проблема распореда производних ћелија. Сви експериментални резултати за проблем распореда производних ћелија су у литератури поређени са резултатима који су добијени **решавачима** (у литератури познати као *solver*-и) за предложени математички модел. У свим тим поређењима су резултати добијени *solver*-има били или најбољи или у групи најбољих резултата. Најбољи *solver* за проблем распореда производних ћелија, према препоруци аутора опште прихваћене библиотеке инстанци за проблем квадратне асигнације, *Quadratic Assignment Problem Library*, QAPLIB (Burkard, Karisch, & Rendl, 1997), је GAMS (*General Algebraic Modeling System, GAMS Development Corporation, Washington, DC, USA, http://www.gams.com*). У дисертацији је нови алгоритам за проблем распореда производних ћелија поређен са резултатима, добијеним GAMS-ом на 15 одабраних QAPLIB инстанци да би се показало да и једноставна имплементација генерализованог приступа даје изузетно квалитетне резултате у полиномијалном времену рада рачунара.

Последња два дела дисертације су усмерена на будуће правце истраживања и закључна разматрања. Показано је да је циљ истраживања остварен и да су постављене хипотезе доказане.

2. СЛОЖЕНОСТ ПРОБЛЕМА

Григориј Перелман, рођен 13. јуна 1966. године у Лењинграду осваја 1982. године, златну медаљу на Математичкој Олимпијади за средњешколце, а 1990. године докторира на чувеној Школи за Математику и Механику Лењинградског државног универзитета. Када је 1994. на Беркли универзитету у Калифорнији доказао Соул хипотезу, понуђен му је посао на најпрестижнијим универзитетима у САД, укључујући Принстон и Станфорд. Одбија их и враћа се у Петроград где се повлачи из јавног живота. Појављује се 2003. године са доказом хипотезе Тхурстонове геометризације, што као последицу доказује Поенкареову хипотезу, постављену 1904. године.

У каквој је вези изузетна биографија Григорија Перелмана са предметом ове дисертације? Поенкареова хипотеза је једна од седам најважнијих нерешених математичких проблема за које је, 2000 године, Математички Институт Clay понудио по милион долара за њихово решавање:

1. Birch-ова и Swinnerton-Dyer-ова хипотеза
2. Hodge-ова хипотеза
3. Navier-Stokes-ове једначине
4. Однос П према НП
5. Поенкареова хипотеза
6. Риманова хипотеза
7. Yang-Mills-ова теорија

Поенкареова хипотеза је једини проблем који је решен до данас. Четврти проблем са ове листе, однос П према НП представља суштину алгоритама комбинаторне оптимизације, који су предмет ове дисертације. Шта више, евентуални доказ да је П = НП би суштински променио свет, почевши од тога да би за последицу имао аутоматско проверавање преосталих пет отворених проблема са претходне листе.

Познато је да је П подскуп од НП, али доказ да НП није подскуп од П је итекако сложен и за сада непознат. Независно од ових доказа, познавање класа П, НП, псеудо П, НП комплетних и НП тврдих проблема је од суштинске важности за све оне који се на било који начин баве алгоритмима комбинаторне оптимизације.

Истраживања у овој дисертацији су заснована на могућности унапређења конструктивних хеуристика коришћењем специфичности проблема који решавају. Крајњи циљ је да се, захваљујући тим специфичностима проблем третира поступком који припада „најслабијем“ подскупу НП проблема, тзв. „псеудо НП“. Због овога ће, у овом делу рада, бити прецизно дефинисане класе сложености и назначен њихов утицај на свеукупни будући развој алгоритама.

2.1. Проблеми, алгоритми и сложеност

Приближно половина светског становништва користи паметне телефоне, моћније рачунаре од суперрачунара од пре три декаде. Рачунари нам стављају на располагање информације, које за нас претражују и сортирају невероватним брзинама. Рачунари нам омогућују комуникацију независну од физичког удаљења. Рачунари могу да обаве невероватна израчунавања, од симулације космичких догађаја до планирања комплексних авионских рута. Рачунари могу да препознају наше гласове, наша лица, наше покрете. Рачунари могу да науче наше преференце и предлажу нам књиге, музику или филмове које бисмо волели да видимо и чујемо. Рачунари могу да управљају возилом уместо нас. Све упућује да не постоје границе у могућностима шта све рачунар може да уради.

Да ли је то заиста тако? У овом поглављу се разматрају проблеми које можда никада неће бити могуће лако решити. Разматрање подразумева најважнији изазов рачунарске науке, ако не и комплетне математике, оскудно назван **проблемом односа П и НП**. Однос П и НП није само математички изазов за чије решење су расписане милионске награде, однос П и НП је много више од тога.

Уколико би доказали да је П = НП, тада би било могуће да се нађе решење за било који проблем који желимо да решимо. Тада би друштво какво познајемо доживело драматичне промене са тренутним, задивљујућим напретком у медицини, науци, забави и аутоматизацији скоро сваког посла.

Уколико је пак, П ≠ НП, тада постоје проблеми који вероватно неће моћи да буду решени у задовољавајућем времену. То није крај приче пошто можемо да креирамо технике које нам помажу да нападнемо те проблеме са различитих позиција. П ≠ НП

значи да не постоји аутоматски начин да се реше ови проблеми, али да усавршавањем разноврсних техника можемо да добијамо све квалитетнија приближна решења ових проблема.

Како је временска сложеност алгоритама кључна величина у вредновању алгоритама обрађених у овој дисертацији, у наставку ће класе сложености бити прецизно формално дефинисане. Ове дефиниције ће јасно сугерисати могуће будуће правце унапређивања тих и сродних алгоритама.

2.2. Проблеми одлучивања, језици и шеме шифрирања

Из разлога прецизности формулације, теорија сложености израчунавања је дизајнирана за примену само на **проблеме одлучивања**. Овакви проблеми, као што је претходно поменуто, имају само два могућа решења, или је одговор "**да**" или је одговор "**не**". Формално, проблем одлучивања **П** се састоји од скупа инстанци $D_{\text{П}}$ и подскупа $Y_{\text{П}} \subseteq D_{\text{П}}$ инстанци које дају одговор "**да**". Стандардни формат за специфицирање проблема одлучивања се састоји из два дела, првим се специфицира генеричка инстанца проблема коришћењем разноврсних компоненти као што су скупови, графови, функције, бројеви, итд, док се другим поставља да-не питање у односу на генеричку инстанцу. На пример, проблем одлучивања везан за проблем трговачког путника, могао би бити формулисан као:

ИНСТАНЦА: Коначан скуп $C = \{c_1, c_2, ..., c_m\}$ „градова", затим „растојање" $d = (c_i, c_j) \in Z^+$ и граница $B \in Z^+$.

ПИТАЊЕ: Да ли постоји путања кроз све градове у C чија укупна дужина није већа од B?

Овај пример може да послужи као илустрација како проблем одлучивања може да се изведе из оптимизационог проблема. Ако оптимизациони проблем тражи структуру одређеног типа која има минималну „цену" између свих одговарајућих структура, тада можемо да придружимо том проблему проблем одлучивања који укључује нумеричку границу B и да поставимо питање да ли постоји одговарајућа структура чија цена **није већа** *од* B. На аналоган начин, проблем одлучивања може да се изведе из проблема максимизације заменом „**највише**" са „**није мање**".

Кључна тачка у овоме је да, докле год функција цене може релативно лако да се израчуна, проблем одлучивања није тежи од одговарајућег оптимизационог проблема. Конкретно, ако смо у стању да у полиномијалном времену одредимо путању најмање дужине за проблем трговачког путника, тада можемо у полиномијалном времену да решимо и одговарајући проблем одлучивања. Довољно је да упоредимо дужину ове путање најмање дужине са датом границом *B*. Према томе, уколико можемо да докажемо да је проблем трговачког путника, дефинисан као проблем одлучивања НП-комплетан, можемо исто да тврдимо и за оптимизациони проблем трговачког путника. На тај начин, иако је теорија НП-комплетности ограничена на проблеме одлучивања, можемо да проширимо ову теорију и на оптимизационе проблеме.

Разлог за ограничавање на проблеме одлучивања лежи у чињеници да ови могу природно, формално да буду представљени преко објекта који је погодан за проучавање у математички прецизној рачунарској теорији. Овај објекат се назива „језик“ и дефинише се на следећи начин: за произвољан, коначан скуп симбола $\sum$, означимо са $\sum^*$ скуп свих речи коначне дужине, састављене од тих симбола. Ако је *L* подскуп од $\sum^*$, тада је *L* **језик** над алфабетом $\sum$. На пример, ако је $\sum = \{0,1\}$, тада се $\sum^*$ састоји од празног стринга и речи 0, 1, 00, 01, 10, 11, 000, 001, и свих осталих речи коначне дужине, састављених од нула и јединица.

Веза између проблема одлучивања и језика се остварује преко шема шифрирања које користимо да би специфицирали инстанце проблема. Шема енкодовања *e* за проблем **П** дефинише начин на који се описује свака инстанца од **П**, коначним низом симбола над датим алфабетом $\sum$. Према томе, проблем **П** и шема енкодовања *e* деле скуп $\sum^*$ на три партиције: у првој су речи које не представљају шифру инстанци од **П**, у другој су речи које енкодују оне инстанце из **П** за које је одговор „не“, док су у трећој оне речи које енкодују инстанце из **П** за које је одговор „да“. Ова трећа партиција речи представља језик који придружујемо проблему **П** и енкодовању *e* :

$L[\mathbf{П}, e] = \{x \in \sum^* : \sum$ је алфабет коришћен у *e*, *x* је енкод инстанце $I \in Y_{\text{П}}\}$.

Ова формализација подразумева да ако резултат важи за језик $L[\mathbf{\Pi}, e]$, тада важи и за проблем **П** под шемом енкодовања *e*.

Приказана формализација омогућује прецизно дефинисање параметра у односу на који се израчунава комплексност алгоритма. Сваком проблему одлучивања се придружује функција *Length*: $D_{\Pi} \to Z^{+}$, независна од примењеног енкодовања. На пример, за проблем трговачког путника можемо да применимо:

$$Length\ [I] = m + \lceil \log_2 B \rceil + \max\{\ \lceil \log_2 d(c_i, c_j) \rceil : c_i, c_j \in C\ \}.$$

Логаритам за основу два подразумева да су нумеричке величине предствљене у бинарном бројном систему. Јасно је да уместо ове, могу да се користе друге *Length* функције и да ће, уколико су дефинисане на правилан начин увек резултовати истим типом комплексности алгоритма.

Посебно треба уочити логаритамске сабирке функције *Length* на овом примеру. Уколико би дужина речи која представља решење проблема била, нпр. линеарно зависна од *B,* поступак не би припадао класи П, јер *B* има експоненцијалну зависност у односу на улазну дужину $\lceil \log_2 B \rceil$, $B = 2^{\log_2 B}$. Ово запажање илуструје једну изузетно важну класу сложености, класу **псеудо П**, о којој ће детаљније бити речи у потпоглављу 2.6.

2.3. Детерминистичка машина и класа П

Формализација, представљена у претходном одељку утврђује апстракцију проблема и инстанци, која омогућује да енкодовање инстанци не утиче на припадност класи сложености. Ради комплетирања математичког формализма алгоритма, неопходно је да се фиксира одређени рачунарски модел на коме се извршавају операције предвиђене алгоритмом. Као модел користимо **детерминистичку Тјурингову машину са једном траком** (DTM), која је шематски приказана на Слици 2-1.

Машина се састоји од траке, састављене од бесконачног низа поља, обележених са ...,-3,-2,-1,0,1,2,3,...; главе за читање и писање и контроле коначних стања. Улаз за DTM је реч $x \in \sum^{*}$ која се уписује у поља траке, почев од позиције 1, један симбол

по пољу. Сва остала поља садрже празан симбол *b*. Контрола коначних стања дефинише функцију транзиције δ за скуп *Q* стања $\{q_0, q_1, ..., q_m\}$, где је q_0 почетно стање. Поред ових *m* стања, постоје два посебна стања q_Y и q_N. Уколико је алфабет $\{0,1\}$, δ је дефинисан за свако од стања из *Q* преко три триплета, један за 0, један за 1 и један за *b*. Триплет има облик (q_i,0/1,+1/-1).

Слика 2-1. Детерминистичка Тјурингова машина

Извршавање програма је једноставно: Програм стартује из стања q_0, а глава чита симбол уписан у поље 1. У зависности да ли је учитани симбол 0, 1 или *b*, функција δ примењује одговарајући триплет: уписује у посматрано поље 0/1, помера главу за једно поље удесно/улево на основу +1/-1, и прелази на стање q_i. Уколико је q_i једно од стања q_Y или q_N, извршавање програма се зауставља, а одговор је „да“ у случају q_Y а „не“ у случају q_N.

Веза између „препознавања“ језика и „решавања“ проблема одлучивања је очигледна. Кажемо да програм M за DTM **решава** проблем одлучивања **П** под шемом енкодовања *e*, уколико се M зауставља после коначног броја корака за све улазне речи из улазног алфабета и важи $L_M = L[\mathbf{П}, e]$.

Сада је могућа формална дефиниција "временске сложености“. **Време** потрошено на имплементацију програма M за улаз *x* је број корака главе за писање до заустављања. За програм M, који се зауставља за све улазе $x \in \sum^*$, **функција временске сложености** $T_M : Z^+ \to Z^+$ је дата са $T_M(n) = \max\{m\}$, при чему је $|x| = n$

и M се завршава после m корака за улаз x. Програм M се назива **полиномијални DTM програм** уколико постоји полином p такав да за свако $n \in Z^+$, $T_M(n) \leq p(n)$.

Најзад, претходни формализам омогућује и формалну дефиницију класе П. Кажемо да проблем одлучивања **П** припада класи П под шемом енкодовања e уколико $L[\mathbf{\Pi}, e] \in \Pi$, односно постоји полиномијални DTM програм који „решава“ **П** под шемом енкодовања e. На основу изведеног закључка у претходном потпоглављу о равноправности различитих шема енкодовања, можемо да изоставимо енкодовање из дефиниције и једноставно кажемо да тада проблем **П** припада класи П.

Чак и површни познавалац програмирања одмах може да уочи да је DTM изузетно неефикасна машина и да је писање иоле сложенијих програма за ову машину крајње мукотрпан посао. Зашто је онда концепт НП-комплетности заснован на овом рачунарском моделу? Одговор је директан и састоји се из два дела. Прво, овај модел потпуно огољује зависност броја корака од величине енкода улазне инстанце. Постављањем улазног стринга на почетне позиције траке, број корака може једноставно да се изрази преко дужине овог стринга, што код сложенијих рачунарских модела није тако експлицитно изражено. Ово даље омогућује стриктну математичку формализацију класе П, што иначе није ни мало једноставан посао.

Друго, иако је за извршење DTM програма потребно знатно више корака него за извршавање одговарајућег програма писаног за неки сложенији рачунарски модел, увек је полиномијални програм за DTM истовремено и полиномијални програм за било који други рачунарски модел и обрнуто. Тако, за програм који се извршава на савременом RAM моделу у времену $T(n)$, емулација извршења овог програма на DTM има временску сложеност $O(T^3(n))$.

2.4. Недетерминистичка машина и класа НП

За проблем трговачког путника, дефинисан у претходном потпоглављу не постоји познат полиномијални алгоритам. Претпоставимо, међутим, да, за одређену инстанцу овог проблема неко тврди да она даје одговор „да“. Да бисмо проверили ово тврђење, довољно је да израчунамо дужину путање дефинисане том инстанцом

и да је упоредимо са *B*. Очигледно је да ову проверу можемо да обавимо у полиномијалном времену. Класа НП управо настоји да изолује ову полиномијалну „*проверљивост*“ решења. Јасно је да полиномијална проверљивост не подразумева полиномијалну решивост проблема. У потврди „да“ одговора инстанце проблема трговачког путника не рачунамо време које је неко провео претражујући експоненцијалан број могућих рута трговачког путника. Ми једноставно прихватамо да, за дату руту за одређену инстанцу *I*, можемо, у полиномијалном времену да проверимо да ли је одговор „да“.

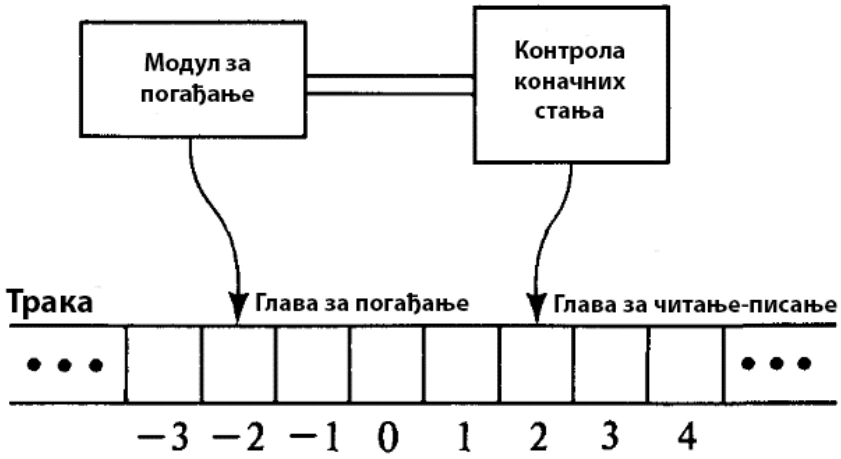
На основу овога, НП може неформално да се дефинише преко појма који је назван „**недетерминистички алгоритам**“. Овај алгоритам може да се посматра као програм који се састоји од две одвојене фазе: прве, **фазе погађања** и друге, **фазе провере**. За дату инстанцу *I* проблема, прва фаза „погађа“ неку структуру *S*. Тада прослеђујемо *I* и *S* као улаз за фазу провере, којом се у детерминистичком маниру примењује програм који може да се заустави са добијеним одговором „да“, или са одговором „не“ или да се никада не заустави.

Недетерминистички алгоритам који решава проблем одлучивања **П** је полиномијални уколико постоји полином *p* такав, да за сваку инстанцу $I \in Y_{\text{П}}$ постоји структура *S* која може да се провери у времену $p(Length[I])$ у фази провере. Класа НП се неформално дефинише као класа свих проблема одлучивања **П** таквих, да се под разумним шемама енкодовања могу да реше у полиномијалном времену применом недетерминистичких алгоритама. Употреба термина „реше“ у овој неформалној дефиницији треба да се узме условно. **Полиномијални недетерминистички алгоритам** је у суштини апстрактан појам за утврђивање степена полиномијалне проверљивости, а није реалан поступак за решавање проблема одлучивања. Уместо једне обраде за дати улаз, он спроводи вишеструку обраду, по једну за сваку могућу структуру *S*.

Постоји још једна значајна разлика између решења проблема одлучивања добијеног недетерминистичким алгоритмом и решења детерминистичког алгоритма: у првом случају не постоји симетрија између „да“ и „не“. Уколико проблем „да ли је *X* тачно за дато *I*„ може да се реши у полиномијалном времену детерминистичким

алгоритмом, тада може и комплементарни проблем „да ли је *X* нетачно за дато *I* “. Ово важи јер се детерминистички алгоритам зауставља за све улазе, тако да је само потребно међусобно заменити одговоре „да“ и „не“. Ово не важи увек за недетерминистички алгоритам. Посматрајмо на пример комплемент проблема трговачког путника: „да ли је тачно да ни једна рута нема дужину *B* или мању“. Не постоји познати начин да се провери „да“ одговор за овај проблем, осим да се испитају све могуће туре. Другим речима, не постоји полиномијални недетерминистички алгоритам за овај комплементарни проблем.

Формализација недетерминистичког алгоритма може да се добије увођењем **недетерминистичке Тјурингове машине са једном траком**, (NDTM), која се добија када се у DTM дода модул за погађање који у поља -1,-2,-3,… уписује структуре *S*. Шематски приказ NDTM је дат на Слици 2-2.



Слика 2-2. Недетерминистичка Тјурингова машина

Недетерминистички рачунар је хипотетички, концептуални рачунар који у ситуацијама када има више могућих следећих корака у алгоритму, увек бира исправан, односно најбољи корак. Избор овог корака није случајан или неодређен. Прецизније, овај рачунар поседује суперрачунарске карактеристике које му омогућавају бирање оптималног корака. Овај рачунар може да се конструише као детерминистички рачунар са неограниченим бројем паралелних процесора. Сваки пут, када постоји више од једног следећег корака у алгоритму, сви ови кораци се симултано обрађују на рачунару. Класа НП садржи проблеме који могу да се реше недетерминистичким рачунаром у полиномијалном времену. Према томе, Н у

ознаци НП не означава да проблем није полиномијалан, већ да је полиномијално решив на недетерминистичком рачунару. Јасно је да је сваки проблем из П такође садржан и у НП пошто се свака детерминистичка операција може да емулира на недетерминистичком рачунару. С друге стране, питање да ли је $НП \subseteq П$, што би даље подразумевало и НП = П је отворен проблем рачунарске науке. Велики број најзначајнијих проблема из реалног живота су у НП и вероватно не у П. За ове, рачунарски тешке проблеме, најбољи до сада познати алгоритми имају у најгорем случају експоненцијалну временску сложеност. Због тога, и поред фасцинантног развоја рачунарског хардвера, повећање величине обрађиване инстанце доводи до немогућности добијања решења у разумном времену.

2.5. Полиномијалне трансформације и НП-комплетност

У свом сажетом, али елегантном раду (Cook, 1971) Кук је поставио темеље НП-комплетности (*NP-complete*) и утврдио више кључних чињеница. Прво, он је уочио значај „полиномијалне редуцибилности“, тј. редукција које могу да се изврше применом полиномијалног алгоритма. Постојање полиномијалног алгоритма за редукцију из једног проблема у други обезбеђује да било који полиномијални алгоритам за решавање једног проблема може да се конвертује у одговарајући полиномијални алгоритам за решавање другог проблема.

Друго, он је фокус усмерио на проблеме одлучивања који нису у П, а могу да се реше у полиномијалном времену применом недетерминистичког рачунара.

Треће, Кук је уочио да је велики број проблема из НП међусобно полиномијално редуцибилан. Због тога је формирана поткласа НП проблема, тзв. класа НП-комплетних проблема, таквих, да је за сваки од тих проблема познат полиномијални алгоритам редукције на неки проблем из те класе. Уколико би се за било који проблем из ове класе доказало да припада класи П, тиме би била доказана и припадност свих осталих проблема из ове класе класи П. С друге стране, проблем који је у најмању руку тежак као било који проблем из НП у смислу да се било који проблем из НП може да сведе на њега у полиномијалном времену, припада класи НП-тврдих (*NP-hard*) проблема. На тај начин, НП-тврди проблеми могу, на неки

начин, да се сматрају за проблеме, који су у најмању руку тешки као и било који проблем из НП, али не морају обавезно да припадају класи НП пошто могу да имају и вишу временску сложеност или да немају полиномску проверљивост решења.

Полиномијална трансформација из језика $L_1 \subseteq \sum_1^*$ у језик $L_2 \subseteq \sum_2^*$ је функција из језика $f : \sum_1^* \to \sum_2^*$ која задовољава следећа два услова:

1. Постоји полиномијални DTM који израчунава *f*.
2. За свако $x \in \sum_1^*$, $x \in L_1$ ако и само ако је $f(x) \in L_2$.

Уколико постоји полиномијална трансформација из L_1 у L_2, пишемо $L_1 \propto L_2$.

Значај полиномијалне трансформације је истакнут следећом Лемом:

Лема 2.1. Ако је $L_1 \propto L_2$, тада из $L_2 \in \Pi$ следи $L_1 \in \Pi$, или, еквивалентно, из $L_1 \notin \Pi$ следи $L_2 \notin \Pi$.

Привилегија да буде први НП-комплетан проблем припала је проблему одлучивања из Булове логике, назван проблемом задовољивости (*Satisfiability problem, SAT*). SAT проблем је проблем одлучивања, који се састоји из буловских израза записаних само помоћу оператора И, ИЛИ, НЕ, променљивих, и заграда. Питање је: ако је дат израз, да ли постоји нека додела вредности ТАЧНО и НЕТАЧНО променљивима, таква да цео израз има вредност ТАЧНО? Ако је тако, каже се да је формула задовољива. Чувена Кукова теорема доказује да је SAT проблем НП-комплетан. Значај ове теореме лежи у чињеници да је пре ње скуп НП-комплетних проблема био празан те је требало доказати да за све језике *L* из НП важи $L \propto L_{SAT}$.

После Кукове теореме листа се убрзано проширивала јер је било довољно само доказати полиномијалну трансформацију на неки од проблема који су већ у скупу НП-комплетних проблема. Данас постоје хиљаде НП-комплетних проблема из најразноврснијих области, као што су математичко програмирање, теорија графова и хиперграфова, формалних језика, обрада стрингова, теорије игара и других.

2.6. Псеудо-полиномијални алгоритми и јака НП-комплетност

Недетерминистичка машина, описана у претходном потпоглављу сугерише један од праваца решавања НП-комплетних проблема. Када би технолошки развој хардвера омогућио паралелну обраду свих равноправних ситуација у поступку решавања проблема комбинаторне оптимизације, НП-комплетни проблеми би били решиви у полиномијалном времену. Ово је, са становишта тренутног технолошког развоја, утопија у општем случају. Међутим, уколико би се, за конкретан проблем који се решава, искористиле његове специфичности у циљу смањења броја могућих равноправних ситуација, сваким новим технолошким унапређењем би се повећавао број инстанци које би могле да се обаве у разумном времену рада рачунара. Другим речима, проблем би се „нападао“ са две стране. Са једне стране би се откривале нове и нове законитости које би сужавале допустиви скуп равноправних решења, а са друге би технолошки напредак омогућавао да се што више таквих равноправних могућности обраде у разумном времену. Овај приступ представља основну идеју над којом су предложена побољшања алгоритама комбинаторне оптимизације.

Друга идеја је везана за потклasу псеудо НП. Не постоји ни једна област за коју је анализа НП-комплетних проблема важнија од проблема који као улазне инстанце укључују и нумеричке вредности. Оптимизација решења оваквих проблема је од кључне важности за најразноврсније научне дисциплине, а посебно за решавање проблема операционог менаџмента. У овом делу ће бити показано да овакви проблеми могу да припадају подскупу НП-комплетних проблема за које постоји највећа вероватноћа да могу да буду решени полиномијалним алгоритмом.

Специфичност нумеричких проблема ће бити илустрована поступком за решавање НП-комплетног проблема партиционисања (*Partition problem*). Овај проблем, који је један од првих шест проблема за које је доказано да припадају НП-комплетној класи, има једноставну формулацију:

ИНСТАНЦА: Коначан скуп A, $|A|=n$ и бројеви $s(a) \in Z^+$, $\forall a \in A$.

ПИТАЊЕ: Да ли постоји подскуп $A' \subset A$ такав да је $\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$?

Нека је $A = \{a_1, a_2, a_3, a_4, a_5\} = \{3,7,1,8,5\}$. Збир бројева скупа је $B = 24$, те сума бројева у подскуповима мора да буде по 12. Поступак попуњава матрицу 5x13 $T = \|t(i, j)\|$, редом по врстама логичком вредношћу 1 на следећи начин. У првом реду је $t(1,0) = 1 \wedge t(1, s(a_1)) = 1$. Сваки следећи ред се попуњава са 1 на основу елемената из претходног реда ако и само ако је или $t(i-1, j) = 1$ или $t(i-1, j - s(a_i)) = 1$. Јасно је да сваки елемент $t(i, j)$ задовољава следећи став: „$t(i, j) = 1$ ако и само ако постоји подскуп од A чија је сума једнака j“. Према томе, када се попуни комплетна матрица, проблем је решен ако и само ако је $t(5,12) = 1$. Матрица T је приказана у Табели 2-1. Полазећи од било ког елемента последње колоне који је једнак јединици оформљују се подскупови од A чија је сума 12.

Табела 2-1. Матрица решења проблема партиционисања

	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1			1									
2	1			1				1			1		
3	1	1		1	1			1	1		1	1	
4	1	1		1	1			1	1	1	1	1	1
5	1	1		1	1	1	1	1	1	1	1	1	1

Јасно је да је временска сложеност овог поступка ограничена полиномом по броју елемената матрице T, тј. да је сложеност $O(nB)$. На први поглед, имамо полиномијални алгоритам за проблем партиционисања и тиме и доказ да је П = НП. Свакако, ово није тачно. Разлог за ово је прецизно дефинисан у потпоглављу 2.2, чиме је објашњено да B има експоненцијалну зависност у односу на улазну дужину $\lceil \log_2 B \rceil$, $B = 2^{\log_2 B}$. Према томе, проблем партиционисања припада класи НП-комплетних проблема, али, захваљујући описаном алгоритму, истовремено и потпкласи псеудо-полиномијалних проблема.

Кључни закључак је да НП-комплетност псеудо-полиномијалних проблема строго зависи од чињенице да су теоретски дозвољене екстремно велике вредности улазних бројева. Уколико би се унапред поставиле горње границе ових бројева, чак и ако би те границе биле полиноми по *Length*[*I*], тај алгоритам би био полиномијални за овако редуковани проблем. Ово отвара нову димензију за решавање читавог низа проблема који се јављају у разноврсним практичним апликацијама.

На пример, у проблемима редоследа послова, где нумеричке вредности представљају трајање послова, ретко могу да се појаве екстремно велики бројеви. Ово је зато што је суштина да ти послови морају да буду обављени, те је бесмислено да било који од тих послова захтева екстремно дугачак интервал времена. Такође, у проблемима где нумеричке вредности представљају емпиријски измерене величине, граница прецизности мерења има ефекат ограничавања опсега вредности бројева за које се алгоритам примењује. Шта више, псеудо-полиномијални алгоритам може да буде користан чак и ако не постоји природна граница величине улазних вредности. Овај алгоритам ће показивати „експоненцијално понашање“ само када се заиста обрађују инстанце које садрже „експоненцијално велике бројеве“, а такве инстанце су веома ретке у скоро свим практичним апликацијама. У свим овим случајевима, овај тип алгоритама може да послужи циљевима исто тако добро као и прави полиномијални алгоритам.

Постојање потклассе псеудо-полиномијалних алгоритама намеће потребу за прецизном формализацијом ове потклассе у теорији НП-комплетности. Због тога се, поред описане функције *Length*[*I*], уводи и функција *Max* са циљем да повеже произвољну инстанцу *I* са целим бројем *Max*[*I*] који представља величину највеће вредности из *I*. Тада, алгоритам спада у подскуп псеудо-полиномијалних алгоритама ако је његова функција временске сложености ограничена са горње стране полиномском функцијом од две променљиве, *Length*[*I*] и *Max*[*I*]. По овој дефиницији, било који полиномијални алгоритам је истовремено и псеудо-полиномијалан, пошто је његова функција сложености ограничена само полиномом од *Length*[*I*].

Многи НП-комплетни проблеми, уколико је П ≠ НП, по дефиницији, не могу да припадају потклассе псеудо-полиномијалних проблема. Ови проблеми имају особину да је сам број *Max*[*I*] ограничен полиномијалном функцијом од *Length*[*I*] и за ове проблеме не постоји разлика између полиномијалних и псеудо-полиномијалних проблема. На пример, једина вредност која се појављује у проблему **клика** је величина клике, а она не може бити већа од броја чворова датог графа. Одавде се намеће очигледан закључак да, под претпоставком да П ≠ НП, само нумерички проблеми из класе НП-комплетних проблема могу бити потенцијални кандидати да

се реше псеудо-полиномијалним алгоритмом. Проблеми из класе НП-комплетних проблема који не могу да се реше псеудо-полиномијалним алгоритмом формирају класу **јаких НП-комплетних проблема**.

Изложена формализација НП-комплетности представља основу која је одредила правац истраживања у дисертацији. Унапређење алгоритама се усмерава ка поступцима који имају највише шанси да, што је могуће више, редукују експоненцијалност НП-комплетних проблема. Због овога ће се, као прво, уградити паралелизам поступака „погађања“, аналогно NDTM у формулацију генерализованог приступа. Као друго, псеудо-полиномијални алгоритми имају сложеност која најприближније одговара конструктивним алгоритмима. Приказни пример псеудо-полиномијалног алгоритма за решавање проблема партиционисања је управо ПКХ. Ово је такође искоришћено за прецизно дефинисање сложености генерализованог приступа.

3. СТРУКТУРА КОНСТРУКТИВНИХ ХЕУРИСТИКА

Детерминистички проблем комбинаторне оптимизације може да се дефинише као четворка (I, F, C, g), где је:

- I дата инстанца проблема комбинаторне оптимизације;
- $F(I)$ коначан скуп допустивих решења за дату инстанцу I;
- $C(x)$, $x \in F(I)$, реална циљна функција која представља меру допустивог решења x;
- g, тип оптимизације ("min" или "max").

Циљ је да се одреди, за дату инстанцу I, **оптимално решење**, које је, допустиво решење уз:

$$C(x) = g\{C(x') \mid x' \in F(I)\}. \quad (1)$$

У значајном броју проблема комбинаторне оптимизације (проблеми распореда, проблеми рутирања, проблеми асигнације) $I = O^n = \{o_1,...,o_n\}$ је скуп од n објеката, од којих сваки поседује дефинисана својстава и $F(O^n)$ може да се формализује различитим уређеним низовима тих објеката. Избор објеката и њихов редослед у низу одређује одговарајућу C вредност. Нека $PR^n = \{pr_1,...,pr_n\}$ означава скуп својстава објеката (pr_i су својства објекта o_i). Скуп датих објеката O^n може формално да се дефинише, без губитка општости, као бијекција скупа $S^n=\{1, 2,..., n\}$ на O^n. На тај начин, I може да буде представљен помоћу PR^n, а $F(O^n)$ може да буде представљен пермутацијама π^n од S^n. Хеуристике за решавање ових проблема се међусобно разликују само по начину на који формирају пермутације и критеријуму за селекцију жељених пермутација.

У овој дисертацији се проучавају ПКХ које на специфичан начин формирају скуп пермутација као решење постављеног проблема комбинаторне оптимизације. У овом одељку су прво дефинисани основни појмови постављеног истраживања, а затим истакнуте разлике које постоје између типова конструктивних хеуристика. На крају одељка је дефинисан прототип ПКХ.

3.1. Дефиниције појмова

У наставку су представљене ознаке и дефиниције појмова везаних за операције са пермутацијама које омогућавају једнозначну формализацију конструктивних алгоритама. У даљем тексту термин **скуп** се користи за неуређени скуп чији су елементи представљени унутар великих заграда, док се термин **секвенца** односи на уређене скупове чији су елементи представљени унутар малих заграда. Према томе, S^m=(1, 2,..., m) означава секвенцу првих m позитивних природних бројева у растућем редоследу, док A^m={a_1,...,a_m} означава скуп од m различитих позитивних природних бројева. Нека је:

- π - произвољна секвенца различитих, позитивних природних бројева;
- $\boldsymbol{\Pi(A^m)}$ - секвенца свих пермутација елемената из A^m. Ова дефиниција подразумева да је почетно уређење елемената из A^m познато. Међусобно уређење секвенци унутар $\boldsymbol{\Pi(A^m)}$ зависи од примењеног начина генерисања пермутација;
- $\pi^m_{A^m}$ - произвољна секвенца из $\boldsymbol{\Pi(A^m)}$. Слово m из суперскрипта се односи на дужину секвенце π док субскрипт A^m означава да је π генеричка секвенца од A^m. Како је слово m у овој ознаци редундантно, ознака може да се пише у скраћеном облику $\pi^m_{A^m} = \pi^m_A$ кад год ово скраћење не проузрокује конфузију;
- $\pi^{m,i}_A$ - i-та секвенца из $\boldsymbol{\Pi(A^m)}$, i =1,..., m!;
- $\pi^m_A(j)$ - природни број на j-тој позицији у π^m_A;
- $\rho^k[\pi]$ - произвољна под-секвенца од π са k елемената. Важно је да се напомене да је уређење елемената у $\rho^k[\pi]$ исто као и у π ;
- $\rho^k[\pi](j)$ - природан број на j-тој позицији у $\rho^k[\pi]$;
- $\mathbf{ext}^k[\pi]$ - под-секвенца од π, добијена када се сви елементи већи од k уклоне из π (на пример, $\mathbf{ext}^4[(3,7,2,8,4,9,1,6,5)] = (3,2,4,1)$).

Уколико је у претходним ознакама слово A изостављено, пермутације се односе на S, тако да скраћена ознака постаје, нпр. $\Pi^m = \Pi(S^m)$; $\pi^m = \pi^m_S$.

Дефиниција 1: **Индекс** од k у π, **indx**(π,k), је позиција броја k у π (**indx**((5,1,3,7,2),7) = 4).

Дефиниција 2: **Композиција пермутација** π_A^a и π^m је $\pi_A^a \circ \pi^m$, при чему је:

$$\pi_A^a \circ \pi^m = (\pi_A^a(\pi^m(1)),...,\pi_A^a(\pi^m(m))),\ a \geq m.$$

На пример, π_A^5 = (7,1,5,2,3), π^3 = (2,3,1), $\pi_A^5 \circ \pi^3$ = (1,5,7).

Дефиниција 3: **Индексна композиција пермутација** $\pi^{m,1}$ и $\pi^{m,2}$ је $\pi^{m,1} \bullet \pi^{m,2}$, при чему је:

$$\pi^{m,1} \bullet \pi^{m,2} = (\mathbf{indx}(\pi^{m,2},\pi^{m,1}(1)),...,\mathbf{indx}(\pi^{m,2},\pi^{m,1}(m))).$$

На пример, $\pi^{5,1}$ = (4,1,5,2,3), $\pi^{5,2}$ = (2,3,1,5,4), $\pi^{5,1} \bullet \pi^{5,2}$ = (5,3,4,1,2).

Дефиниција 4: ***k*-партиција** скупа A^m,

$$PRT^k(A^m) = (PRT_1^{m_1}(A^m),...,PRT_k^{m_k}(A^m)) = (A_1^{m_1},...,A_k^{m_k})$$

је секвенца од k непразних подскупова, $A_i^{m_i}$, од A^m таквих да је сваки елеменат од A^m у једном и само једном од тих подскупова (тј., A^m је дисјунктна унија тих подскупова). Дефиниција важи и у случајевима када је било који од скупова $A^m, PRT_1^{m_1}(A^m),...,PRT_k^{m_k}(A^m)$ секвенца.

Дефиниција 5: ρ_1 и ρ_2 су **дисјунктне** секвенце уколико немају заједничких елемената.

Дефиниција 6: **Конкатенација дисјунктних секвенци** $\rho_1^{a_1},...,\rho_k^{a_k}$ је секвенца $(\rho_1^{a_1},...,\rho_k^{a_k})$, добијена додавањем сваке од секвенци, једне за другом:

$$(\rho_1^{a_1},...,\rho_k^{a_k}) = (\rho_1^{a_1}(1),...,\rho_1^{a_1}(a_1),...,\rho_k^{a_k}(1),...,\rho_k^{a_k}(a_k)).$$

Дефиниција 7: **Потез уметања** $v(j,l)$ умеће природан број j на позицију l секвенце ρ^k, где је $j \notin \rho^k; l \leq k+1$. Нека $\mathbf{ins}(\rho^k, j, l)$ означава резултујућу секвенцу од (k + 1) елемената. Према томе, потез $v(j,l)$ формира секвенцу ρ^{k+1} из ρ^k на следећи начин:

$$\rho^k \xrightarrow{v(j,l)} \mathbf{ins}(\rho^k, j, l) = \rho^{k+1} = (\rho^k(1),...,\rho^k(l-1), j, \rho^k(l),...,\rho^k(k)). \quad (2)$$

3.2. Праве и хибридне конструктивне хеуристике

Конструктивне технике граде решење корак по корак, проверавајући допустивост решења и вредност циљне функције. У сваком кораку се један нови објекат умеће у тренутно парцијално решење, те се у литератури овакве хеуристике често називају и **хеуристике уметања** (*insertion based constructive heuristics*). Ове хеуристике се, по правилу, састоје из **фазе иницијализације** и циклуса у **фази уметања**. У фази иницијализације, функција селекције $f_1(PR^n)$ одређује природан број e_1 из S^n, који ће бити придодат парцијалном решењу x. Фаза уметања има n-1 итерацију, тако да је, на почетку k-те итерације, S^n подељен на два подскупа:

$$PRT^2(S^n) = (PRT_1^k(S^n), PRT_2^{n-k}(S^n)) = (A_1^k, A_2^{n-k}).$$

У свакој k-тој итерацији фазе уметања спроводе се два типа селекције: селекција $f_2(PR^n, k, A_2^{n-k})$ броја e_{k+1} из A_2^{n-k} који се уклања из A_2^{n-k} и селекција $f_3(PR^n, k, e_{k+1}, \rho^k[A_1^k])$ позиције l у $\rho^k[A_1^k]$ на коју ће e_{k+1} бити уметнут. Спроведено уметање једнозначно одређује уређење бројева у $\rho^{k+1}[A_1^{k+1}]$.

Већина конструктивних хеуристика користи потезе уметања за конструкцију коначног решења. Ипак, одређене хеуристике спроводе поновна уметања већ уметнутих објеката у одређеним корацима конструкције коначног решења. Ове хеуристике су, *de facto*, мешавина конструктивних хеуристика и хеуристика побољшања. Поновна уметања, по правилу, кваре временску комплексност, те ће се у даљем тексту правити оштра дистинкција између хеуристика са поновним уметањима (биће третиране као хибридне хеуристике) и ПКХ. Евидентна је јасна разлика између различитих типова хеуристика у односу на квалитет решења и временску комплексност поступка. Хибридне хеуристике постижу боља решења по цену лошије временске ефикасности и комплексности поступка. Насупрот њима, ПКХ, по правилу, имају полиномијалну комплексност нижег степена. Ово је кључна разлика којој се не поклања адекватна пажња у литератури. Наиме, за значајан број примена проблема комбинаторне оптимизације, у стварном животу, је утврђено да су ти проблеми динамичке природе, тј. да је брзина решавања проблема најважнији фактор у вредновању одговарајућих хеуристика. Хеуристике побољшања, по правилу, уводе вештачка временска ограничења да би се извршавање програма

завршило у разумном времену. Хибридне конструктивне хеуристике најчешће задржавају полиномијалну сложеност, али са увећаним степеном. Веома често се процентуално мала побољшања плаћају вишеструким повећањем реда комплексности поступка.

Још једна важна карактеристика ПКХ је да захваљујући својој полиномијалној зависности често могу да се пореде са егзактним алгоритмима. Шта више, неки најпознатији егзактне конструктивне методе имају структуру ПКХ. На пример, чувени Дајкстрин алгоритам за најкраће путање у мрежи и Џонсонов алгоритам за *flow-shop* проблем са две машине представљају ПКХ.

Из изложеног може да се закључи да су једноставност и полиномијална зависност две кључне особине ПКХ. Према томе, било какво побољшавање постојећих ПКХ, које се предлаже у литератури, треба обавезно да се одреди према чињеници колико се тим побољшањем кваре ове две кључне особине. Циљ дисертације је управо да истражи могућности побољшања постојећих конструктивних хеуристика уз непромењену временску сложеност поступка.

3.3. Прототип праве конструктивне хеуристике

На основу поставки, изложених у претходном потпоглављу, прототип ПКХ може да се прикаже кроз следеће кораке:

Прототип ПКХ

Function $GCA(PR^n, F, C, g)$

1 *фаза иницијализације*: $e_1 \leftarrow f_1(PR^n)$; $\rho^1[A_1^1] = (e_1)$; $A_2^{n-1} = S^n \setminus e_1$

2 $k \leftarrow 1$

3 **repeat** (*фаза уметања*)

4 $e_{k+1} \leftarrow f_2(PR^n, k, A_2^{n-k})$; $A_2^{n-k-1} = A_2^{n-k} \setminus e_{k+1}$

5 $l \leftarrow f_3(PR^n, k, e_{k+1}, \rho^k[A_1^k])$

6 $\rho^{k+1}[A_1^{k+1}] \leftarrow \mathbf{ins}(\rho^k[A_1^k], e_{k+1}, l)$

7 $k \leftarrow k + 1$

until $k = n$

return $x \leftarrow \rho^n[A_1^n]$

У фази иницијализације, функција f_1 одређује први елемент e_1 који се уклања из скупа A_2^{n-1} и формира секвенцу $\rho^1[A_1^1]$. У фази уметања, у свакој итерацији, функција f_2 одређује нови елемент e_{k+1} који се уклања из скупа A_2^{n-k}, док функција f_3 одређује позицију l секвенце $\rho^k[A_1^k]$ на коју ће бити уметнут овај елемент.

Из овог псеудокода може да се уочи да функције селекције f_1, f_2 и f_3 једнозначно дефинишу прототип ПКХ. Разлике између алгоритама потичу само од разлика у дефиницијама ових функција. У већини објављених радова, свака од ових функција враћа природан број, при чему f_1 и f_2 овај број добијају случајним избором. Функцијски алгоритам, *Generalized Constructive Algorithm*, GCA, (Danilovic & Ilic, 2016), може да се дефинише као функција:

$$x \leftarrow GCA(PR^n, F, C, g, f_1, f_2, f_3) \tag{3}$$

У неким хеуристикама f_1 одређује почетну пермутацију πp^n од S^n, означену као **приоритетно уређење**, и враћа први елеменат од πp^n. У тим хеуристикама, у *k*-тој итерацији, f_2 враћа $\pi p^n(k+1)$. У циљу једноставности излагања, у даљем тексту ће се усвојити ови облици функција f_1 и f_2, тј.

$$\pi p^n(1) \leftarrow f_1(PR^n) \text{ и } \pi p^n(k+1) \leftarrow f_2(PR^n, k, A_2^{n-k}).$$

Ове претпоставке не умањују општост GCA, пошто било који други метод селекције броја e_k може једноставно да се конвертује у овај тип GCA формулације. С друге стране, уведене претпоставке омогућују да се почетно πp^n замени са S^n. Конкретно, произвољна секвенца π^n може да се преслика на S^n преко: $S^n = (\mathbf{indx}(\pi^n, \pi^n(1)), ..., \mathbf{indx}(\pi^n, \pi^n(n)))$.

Према овоме, уместо операција над елементима секвенце, оперише се са позицијама тих елемената у секвенци. Било која пермутација π^n елемената из πp^n се једнозначно пресликава на пермутацију $\pi^n \bullet \pi p^n$ елемената из S^n. На пример, за $\pi p^5 = (3,1,4,5,2)$ и $\pi^5 = (2,4,1,3,5)$, пресликана пермутација из S^n је:

$$\pi^5 \bullet \pi p^5 = (\mathbf{indx}(\pi p^5, 2), \mathbf{indx}(\pi p^5, 4), \mathbf{indx}(\pi p^5, 1), \mathbf{indx}(\pi p^5, 3), \mathbf{indx}(\pi p^5, 5)) = (5,3,2,1,4).$$

Предложене претпоставке поједностављују партицију скупа S^n на:

$$PRT^2(S^n) = (\pi^k, (k+1,..., n)), \quad (4)$$

и прецизирају да је $e_k = k$. На тај начин πp^n и f_3 једнозначно дефинишу GCA. Ово омогућује да се формализација GCA представи кроз формализацију функције f_3. Другим речима, прототип ПКХ је представљен кроз формализацију функције f_3. У општем случају, за дати проблем P = (F,C,g), GCA одређује, према изразу (1), решење x за инстанцу PR^n, тј.

$$x \leftarrow GCA(PR^n, P, f_1, f_2, f_3). \quad (5)$$

4. СЛОЖЕНОСТ ХЕУРИСТИКА

Хеуристике за решавање проблема комбинаторне оптимизације могу, у односу на начине добијања решења, да се поделе на конструктивне хеуристике, хеуристике побољшања и хибридне хеуристике. У конструктивним хеуристикама одабрани објекат представља почетно парцијално решење, а затим се, у свакој следећој итерацији по један нови објекат уноси у парцијално решење. Комплетан скуп објеката решења постоји тек по завршетку рада алгоритма. Хеуристике побољшања мењају комплетна решења проблема у потрази за најбољим решењем. Хибридне хеуристике се састоје од више алгоритама, најчешће и од конструктивних хеуристика и од хеуристика побољшања. Разлика у поступцима ових хеуристика представља и суштинску разлику у њиховој временској сложености.

У овом делу дисертације се разматра временска сложеност ових хеуристика. Такође су анализиране могућности примене ових хеуристика у паралелној обради.

4.1. Конструктивне хеуристике

На основу кода Алгоритма 1, прецизно може да се одреди број операција ПКХ у функцији од величине инстанце, n. У алгоритму постоји само један циклус који се понавља $n - 1$ пут. У сваком циклусу, поред три елементарне операције, одређују се и функције $f_2(PR^n, k, A_2^{n-k})$ и $f_3(PR^n, k, e_{k+1}, \rho^k[A_1^k])$. Према томе, број елементарних операција алгоритма је $(3 + O(f_2) + O(f_3))(n - 1)$. Уколико се функције f_2 и f_3 одређују у полиномијалном времену, алгоритам има полиномијалну временску зависност. Шта више, број елементарних операција, потребних за одређивање функција f_2 и f_3 једнозначно и прецизно одређује укупан број елементарних операција алгоритма. Очигледно је које значење има ова особина ПКХ за објективност евалуације временске ефикасности. Никаква експериментална тестирања дужине рада оваквих алгоритама није потребна, довољно је да се пореде полиноми који одговарају тестираним алгоритмима.

Временску сложеност хибридних конструктивних хеуристика сачињавају компоненте хеуристика које сачињавају хибрид, а удео сваке од њих зависи од

начина на који је овај хибрид конструисан. Као илустрација може да се размотри побољшања чувене НЕХ хеуристике (Nawaz, Enscore Jr, & Ham, 1983). Радови који по општој процени представљају најквалитетнија побољшања НЕХ хеуристике су радови (Fernandez-Vigas & Framinan, 2014), (Dong, Chen, Huang, & Nowak, 2013), (Ruiz & Stutzle, 2007) и (Rad, Ruiz, & Boroojerdian, 2009). Побољшања вредности циљне функције, остварена у овим алгоритмима, су највећа међу свим осталим побољшањима до данас. Нажалост, ова побољшања су добијена на штету ефикасности поступка уз значајно компликованију формулацију самог алгоритма. У фази уметања се врши промена позиција већ уметнутих послова, што вишеструко повећава ред сложености оригиналног алгоритма. Табела 4-1 приказује времена извршења два најбоља алгоритма, представљена у (Rad, Ruiz, & Boroojerdian, 2009), означена са ФРБ3 и ФРБ5. Ставке у колонама приказују колико пута посматрани алгоритам на одређеној инстанци дуже ради у односу на време извршавања истог алгоритма на најмањој инстанци. Ова времена су поређена са временом извршавања НЕХ алгоритма са Тајллардовим убрзањем, означена са НЕХТ (Taillard, 1990). Треба напоменути да су побољшања вредности циљне функције ових алгоритама у односу на НЕХ мања од 3%.

Табела 4-1. Поређење временских сложености алгоритама

n	m	НЕХТ	ФРБ3	ФРБ5
20	5	1.00	1.00	1.00
20	10	2.00	1.00	1.31
20	20	4.00	2.83	3.00
50	5	6.25	12.50	8.69
50	10	12.50	24.67	21.15
50	20	25.00	48.00	47.85
100	5	25.00	91.33	55.77
100	10	50.00	178.83	142.08
100	20	100.00	370.00	378.38
200	10	200.00	1,389.83	951.46
200	20	400.00	2,845.50	2,929.23
500	20	2,500.00	43,613.00	40,503.38

Овај пример јасно истиче супериорност НЕХТ алгоритма, односно ПКХ. Број елементарних операција НЕХТ алгоритма је пропорционалан са $n^2 \cdot m$ и ова зависност је стриктна за све тестиране инстанце. За остала два алгоритма не може да се утврди ни приближно нека зависност, али је очигледно да се повећава стрмина зависности са повећањем величине инстанце. Инстанце малих величина често могу

да се обраде и егзактним алгоритмима пошто у разумном времену може да се обради значајан део допустивог скупа. Код великих инстанци је ред комплексности од суштинског значаја, посебно у применама у којима су посматрани проблеми динамичке природе.

Најзад, структура ПКХ, представљена прототипом ПКХ у поглављу 2, недвосмислено указује на могућност примене паралелног програмирања на овај тип хеуристика. На пример, може се паралелно обрађивати више вредности $l \leftarrow f_3(PR^n, k, e_{k+1}, \rho^k[A_1^k])$. Кључни фактор који ово омогућује је то што, на основу дефиниције поступка, сваки од процесора обрађује међусобно потпуно различите делове допустивог скупа.

4.2. Хеуристике побољшања

Практично, не постоји ни један проблем комбинаторне оптимизације из класе НП за који не постоји алгоритам за његово решавање применом неке од метахеуристика побољшања. По правилу, ове хеуристике дају најбоље резултате за вредности циљних функција. Свакодневно се објављују радови у којима се на неки НП проблем комбинаторне оптимизације примењује нови скуп метахеуристика са посебно подешеним вредностима својих параметара за инстанце на које се те хеуристике примењују. Сигурно ће проћи још доста времена док се неки нови приступ не наметне као компетентна алтернатива хеуристикама побољшања у општем случају.

Управо због несумњиве супериорности метахеуристика побољшања у погледу вредности циљне функције, савремени аутори нових алгоритама приступају проблему једнострано, са јединим циљем да покажу да њихов алгоритам постиже боље резултате од конкуренције на општеприхваћеним тест инстанцама. Једноставан рецепт за објављивање таквих радова, чак и у најеминентнијим часописима, је да се приложи што више експерименталних података који потврђују супериорност предложеног алгоритма у односу на најбоље одговарајуће алгоритме. При овоме аутори најчешће занемарују две веома важне чињенице: 1. пропуштају да схвате и објасне зашто њихов алгоритам ради боље од осталих и 2. недовољно

користе специфичности конкретног проблема и конкретне инстанце тог проблема. Последица овога је да су разлике у вредностима објављених резултата све мање, и да се све ређе објављује неки квалитативни продор у решавању неког проблема. Један од најзначајнијих фактора који се занемарује у презентацијама хеуристика побољшања је прецизна и објективна анализа временске сложености предложених алгоритама.

Метахеуристике дефинишу поступке којима се текуће решење проблема поправља кроз итерације. У тој претрази допустивог скупа наилази се на два суштинска проблема: заглављивање у локалном оптимуму и обраду већ обрађених решења. Први проблем се решава **диверсификацијом**, удаљавањем од тренутног оптимума, док се други решава прављењем тзв. табу-листа или се уопште не разматра. Овај поједностављени приказ метахеуристика има за циљ да истакне три чиниоца који опредељују њихову временску сложеност.

Побољшање вредности циљне функције се, без изузетка у свим метахеуристикама обавља тако што се у случајевима појаве побољшања, на побољшано решење примењују предвиђени поступци од почетка. Ово је на веома једноставан начин формулисано у VNS (*Variable Neighborhood Search*) метахеуристици (Mladenović & Hansen, 1997) у поступку за промену околине.

Algorithm 1: Neighborhood change

Function NeighborhoodChange (x, x', k)
if $f(x') < f(x)$ **then**
    $x \leftarrow x'$ // Make a move
    $k \leftarrow 1$ // Initial neighborhood
else
    $k \leftarrow k + 1$ // Next neighborhood
return x, k

У случају када је у кораку 1 добијено побољшање решења, у кораку 3 се k враћа на јединицу, односно на почетну околину. За структуиране алгоритме, какав је и овај алгоритам, временска сложеност може да се одреди на основу структуре циклуса, при чему није неопходно да се познаје значење промењљивих у алгоритму. У

алгоритму *Basic VNS*, поступак *Neighborhood change* се налази у циклусу (кораци 3-6)

Algorithm 2: Basic VNS

Function BVNS(x, k_{max}, t_{max})
1 **repeat**
2 $k \leftarrow 1$
3 **repeat**
4 $x' \leftarrow$ Shake(x, k)
5 $x'' \leftarrow$ BestImprovement(x') // Local search
6 $x, k \leftarrow$ NeighborhoodChange (x, x'', k)
until $k = k_{max}$
7 $t \leftarrow$ CpuTime()
until $t > t_{max}$
return x

Из структуре алгоритма се види да, уколико је $f(x'') < f(x)$, вредност за k се враћа на један, тако да може да наступи случај када је $k < k_{max}$ током комплетеног извршења алгоритма без обзира колика је вредност промењиве k_{max}. Уколико не би било вештачког ограничења t_{max}, наступио би један од два случаја: 1. алгоритам би претражио комплетан допустиви скуп или 2. алгоритам се никад не би зауставио ако би упао у петљу обраде већ обрађених решења. Временска сложеност алгоритма се рачуна за најгори случај, другим речима, за било који проблем и за било који алгоритам који га решава може да се конструише таква инстанца која ће у свакој итерацији k враћати на један. Према томе, временска сложеност овог алгоритма је пропорционална величини допустивог скупа који алгоритам претражује у најгорем случају, а то је експоненцијална временска зависност. Ограничење t_{max} једноставно прекида извршавање алгоритма, али не мења његову временску сложеност. Друга је ствар што у том, вештачки ограниченом времену, метахеуристике постижу најбоље резултте.

Други чинилац који одређује временску сложеност метахеуристика је диверсификација. У највећем броју случајева се удаљавање од локалног оптимума обавља на случајан начин. Најпрецизнији поступак диверсификације је уствари сама метахеуристика симулираног каљења у којој се распон удаљавања смањује кроз итерације алгоритма по унапред дефинисаном закону смањења температуре.

Међутим, и код ове, као и код осталих метахеуристика, постоји фактор случајности. Због овога, већина метахеуристика, по правилу не даје исте резултате када се више пута примени на исту инстанцу. Многи аутори, и у најеминентнијим часописима, за поређење наводе најбоље резултате које добијају вишеструким применама алгоритма на истој инстанци. Да би задовољили форму, прилажу статистичке резултате о опсезима добијених резултата. Ово је, наравно, бесмислено, јер су ти алгоритми намењени за примену у реалном животу у коме се обрада обавља једном за сваку инстанцу. Једино одговарајуће поређење је поређење најгорег случаја, односно поређење са њиховим најлошијим добијеним резултатима. Овоме треба додати још једну важну чињеницу: већина метахеуристика има параметре, чије се вредности прилагођавају тестираним инстанцама. То такође уноси значајну необјективност у експериментално поређење, јер се некад огромно време потрошено на одређивање тих параметара не рачуна приликом поређења.

Најзад, управо због диверсификације, хеуристике побољшања имају озбиљан проблем са израженом могућношћу да се током извршења више пута обрађује исти део допустивог скупа. Ово, даље, може да успостави петљу, која поразно делује на временску ефикасност поступка. Грамзиве хеуристике немају тај проблем, јер је вредност циљне функције у свакој итерацији све ближа локалном оптимуму. Једини начин за отклањање ове опасности су табу листе, али је њихова примена веома ограничена. Наиме, листе памте само обрађена решења за неколико последњих итерација, јер би иначе сама претрага тих листа имала крајње неповољну временску сложеност. Због овога се често дешава да неке хеуристике побољшања раде дуже и постижу лошије резултате од тоталне претраге на одређеним инстанцама. Ово је показано у поглављу 8 у експерименталном поређењу алгоритама за решавање проблема формирања производних ћелија.

Хеуристике побољшања нису подесне, у општем случају, за паралелну обраду. Разлог за ово лежи управо у проблемима који су претходно побројани. Паралелно програмирање подразумева поделу претраге допустивог скупа, при чему се унапред дефинише правило по коме се тај скуп дели. Диверсификација по правилу усмерава претрагу на случајно одабрани део допустивог скупа, тако да је немогуће на једноставан начин контролисати који део допустивог скупа се додељује ком процесору.

4.3. Хибридне хеуристике

Хибридне хеуристике су састављене од различитих типова алгоритама. Могу да садрже и егзактне алгоритме, али и најмање једну хеуристику. Хибридне хеуристике најчешће прво примењују конструктивни алгоритам за добијање почетне популације, а затим неком од хеуристика побољшања одређују коначно решење. Сложеност ових хеуристика дефинише онај алгоритам чија је сложеност највећа. Уколико је конструктивна хеуристика ПКХ, сложеност хибридне хеуристике одређује сложеност хеуристике побољшања.

5. НЕДОСТАЦИ ПОЗНАТИХ ПРАВИХ КОНСТРУКТИВНИХ ХЕУРИСТИКА

Брзе технолошке промене непрекидно отварају нове могућности и изазове, што неминовно проширује могућности пословања кроз нова окружења, технике и процесе. Због овога, управљање операционим (производним и услужним) системима постаје практично најважнији изазов глобалног компетитивног окружења. Операциони менаџмент је постао кључни елемент у побољшању продуктивности пословања свуда у свету. Истраживачи свакодневно проналазе начине како би се унапредиле постојеће технике и дефинисале нове. Велика разноврсност проблематике је довела до публиковања небројено радова у којима се предлажу унапређења постојећих техника и дефинисања нових. Међу овим радовима, значајно место заузимају поступци за решавање проблема комбинаторне оптимизације. Саставни део великог броја ових поступака садржи ПКХ. Циљ овог дела рада је да истакне разноврсност проблема за чије се решавање користи ПКХ и укаже на недостатке у презентацији и евалуацији ових хеуристика.

Критеријум за избор репрезентативних ПКХ је да су оне објављене у најугледнијим часописима из своје области и да су ти радови значајно референцирани у другим радовима. Са изузетком неких радова који су изабрани због свог значаја за област коју обрађују, скоро сви радови су објављени 2014, 2015. и 2016. године. Табела 5-1 даје преглед ових хеуристика са назнаком који проблем решавају, референцом и бројем странице. Проблеми су представљени у оригиналу, на енглеском, јер само тако могу једнозначно да буду референцирани.

Табела 5-1. Репрезентативне хеуристике за проблеме операционог менаџмента

Бр.	Проблем	Бр.	Алгоритам	стр.
1	The vehicle routing and scheduling problems (Campbell & Savelsbergh, 2004)	1	Algorithm 1. Insertion Heuristic	370
2	The earthwork allocation, sequencing and routing (Burdett & Kozan, 2014)	2	Algorithm 8. Make Assignment Greedy (S,G, c, f)	749
3	The two-dimensional bin packing (Fleszar, 2013)	3	First-fit and best-fit insertion heuristics	467
		4	Critical-fit insertion heuristic	468

4	The response time variability problem, (Garcia-Villoria, Salhi, Corominas, & Pastor, 2011)	5	Gr1 - Gr4	161
		6	A priority-based rule heuristic (Gr5)	162
		7	A contribution-based rule heuristic (Gr6)	162
		8	A crude constructive hyper-heuristic (CHH-1)	163
		9	A learning-based constructive hyper-heuristic (CHH-2)	163
5	The capacitated arc routing problem (Kirlik & Sipahiogly, 2012)	10	Algorithm 1. Algorithm CPP(G)	2382
		11	Algorithm 2. Algorithm RPP(G,R)	2382
		12	Algorithm 3. Shorten(G,R,T)	2383
		13	Algorithm 4. ConstructTour(G,R)	2384
		14	Algorithm 5. Mod. Ulusoy algorithm	2385
6	The dynamic facility layout problem (McKendall & Hakobyan, 2010)	15	BSH algorithm	176
7	The selective maintenance problem (Lust, Roux, & Riane, 2009)	16	Construction heuristic	1170
8	The set covering problem (Lan, DePuy, & Whitehouse, 2007)	17	Procedure Meta-RaPs-SCP-Construction (I, J, priority, restriction)	1391
9	The pickup and delivery problem with time windows (Lu & Dessouky, 2006)	18	Flow chart of the overall construction heuristic	682 Fig. 7.
10	The undirected rural postman problem (Ghiani, Lagana, & Musmanno, 2006)	19	Outline of the insertion procedure	3452 Fig 2.
11	The asymmetric TSP (Glover, Gutin, Yeo, & Zverovich, 2001)	20	Modified Karp-Steele patching heuristic	557
12	Allocation and scheduling of rescue units (Wex, Schryen, Feuerriegel, & Neumann, 2014)	21	Greedy heuristic	701
		22-24	Shed 1 - 3	701
		25-28	Shed 4 - 7	702
13	Container loading problem (Araya & Riff, 2014)	29	Algorithm 2. BeamSearch (in: Γ, C, B, w, in-out: sbest).	102
14	Location-arc routing problem (Lopes, Plastri, Ferreira, & Santos, 2014)	30	The augment-merge algorithm	311
15	Switch allocation in electrical distribution networks (Benavides, Ritt, Buriol, & Franc, 2013)	31	Semi-greedy construction algorithm	26
		32	Sample construction algorithm	27
16	Design of hub networks with multiple lines (Sáa, Contrerasb, & Cordea, 2015)	33	Algorithm 1. A deterministic constructive heuristic.	190
		34	Algorithm 2. A modified FW algorithm	191

17	Dispersion problems (Aringhieria, Cordoneb, & Grosso, 2015)	35	Algorithm 1. BuildDP(N,d,m,DP).	23
18	Green scheduling of a two-machine flowshop (Mansourila, Aktasb, & Besikci, 2016)	36	Algorithm 1. The schedule development heuristic SDH	115
19	The multiple container loading problem with preference (Tiana, Zhub, Limc, & Wei, 2016)	37	Algorithm 1. Enumerate all rows	88
20	Network repair crew scheduling and routing for emergency relief distribution problem (Duquea, Dolinskaya, & Sörensen, 2016)	38	Algorithm 1. Construction Phase	278
21	Minimize makespan for parallel batch machines with arbitrary job sizes (ZH & Leung, 2015)	39	Algorithm 1. The RLOA Algorithm	658
22	Rectangle packing area minimization problem (Hea, Jia, & Li, 2015)	40	Algorithm 1. The basic algorithm LIF0	677
		41	Algorithm 2. Enhanced algorithm LIF1	678
23	Risk-constrained cash-in-transitve vehicle routing problem (Talarico, Sörensen, & Springae, 2015)	42	Modified Clarke and Wright algorithm with greedy selection mechanism CWg	461
		43	Nearest neighbour with greedy selection mechanism NNG	462
		44	TSP nearest neighbor with greedy randomized selection mechanism plus splitting (TNNg)	462
		45	TSP Lin–Kerninghan plus splitting (TLK)	463
24	Static rebalancing problem (Erdogana, Battarrab, & Calvo, 2015)	46	Constructive Heuristic	674
25	The multiple vehicle pickup and delivery problem with LIFO constraints (Benaventa, Landeteb, Motaa, & Tirado, 2015)	47	Clarke and Wright	757
		48	Random solution with consecutive pickups and deliveries	757
		49	Random solution with pickups before deliveries	757
26	The two-dimensional bin-packing problem (Cui, Cui, & Tang, 2015)	50	GetPattern	46
27	Assembly line balancing problem (Akpinar & Oglu, 2014)	51	Solution encoding mechanism used while generating initial colonies	453
		52	Solution encoding mechanism used by the neighborhood structure	454

28	Total flowtime min in a no-waitflowshop with sequence - dependent setup times (Nagano, Miyata, & Araújo, 2014)	53	BAH algorithm	2
		54	BIH algorithm	2
		55	TRIPS heuristic	3
29	2-dimensional binpacking problems with irregular pieces and guillotine cuts (Sykora, Valdes, Bennell, & Tamarit, 2015)	56	Algorithm 1. Constructive algorithm structure	23-24
30	Test assembly design Problem (Pereira & Vilà, 2015)	57	Algorithm 1. Greedy heuristic	4810
31	Manufacturing Cell Formation problem (Danilovic & Ilic, 2016)	58	GCA	
32	Parking slot assignment for urban distribution (Riu, Fernández, & Estrada, 2015)	59	Algorithm 1. Constructive phase	11
33	The shift minimisation personnel task scheduling problem (Smet, Wauters, Mihaylov, & Berghe, 2014)	60	Algorithm 1. First fit constructive heuristic	66
		61	Algorithm 2. Constructive mat heuristic	67
34	The two-dimensional vector packing problem with piecewise linear cost function (Hua, Lim, & Zhu, 2015)	62	Algorithm 1. Greedy-on-weight-rangeheuristic GWR(I, SR, DR, L)	46

У Табели 5-1 је наведено 62 ПКХ за решавање 34 разноврсних проблема операционог менаџмента. Ове хеуристике представљају репрезентативни узорак небројеног скупа различитих проблема. У овим радовима, ПКХ су представљене на најразноврсније начине, укључујући псеудо кодове, дијаграме тока и текстуалне описе. Иако распон сложености ових ПКХ иде од најједноставнијих до изузетно комплексних алгоритама, сви ови алгоритми се састоје од фазе иницијализације и фазе уметања и сви могу да се представе преко три GCA функције. Једина суштинска разлика је у дефиницијама ове три функције.

Структура сваке од наведених хеуристика може да се прецизно разматра без улажења у проблематику коју решава. Ово је веома битно са становишта идеје која се обрађује у дисертацији. Наиме, циљ рада је да покаже могућност генерализације структуре ПКХ. Ово може да се покаже на примеру *Algorithm 1*: *Insertion Heuristic* за проблеме редоследа и рутирања (Campbell & Savelsbergh, 2004) (проблем 1 из Табеле 5-1).

Algorithm 1: Insertion Heuristic

1. N = скуп нераспоређених клијената
2. R = скуп путања; увек садржи празну путању, на почетку се састоји само од празне путање.
3. **while** $N \neq \Phi$ **do**
4. $p* = -\infty$
5. **for** $j \in N$ **do**
6. **for** $r \in R$ **do**
7. **for** $(i-1, i) \in r$ **do**
8. **if** *Feasible*(i,j) and *Profit*$(i,j) > p*$ **then**
9. $r* = r$
10. $i* = i$
11. $j* = j$
12. $p* = Profit(i, j)$
13. **end if**
14. **end for**
15. **end for**
16. **end for**
17. $Insert(i*, j*)$
18. $N = N \setminus j*$
19. $Update(r*)$
20. **end while**

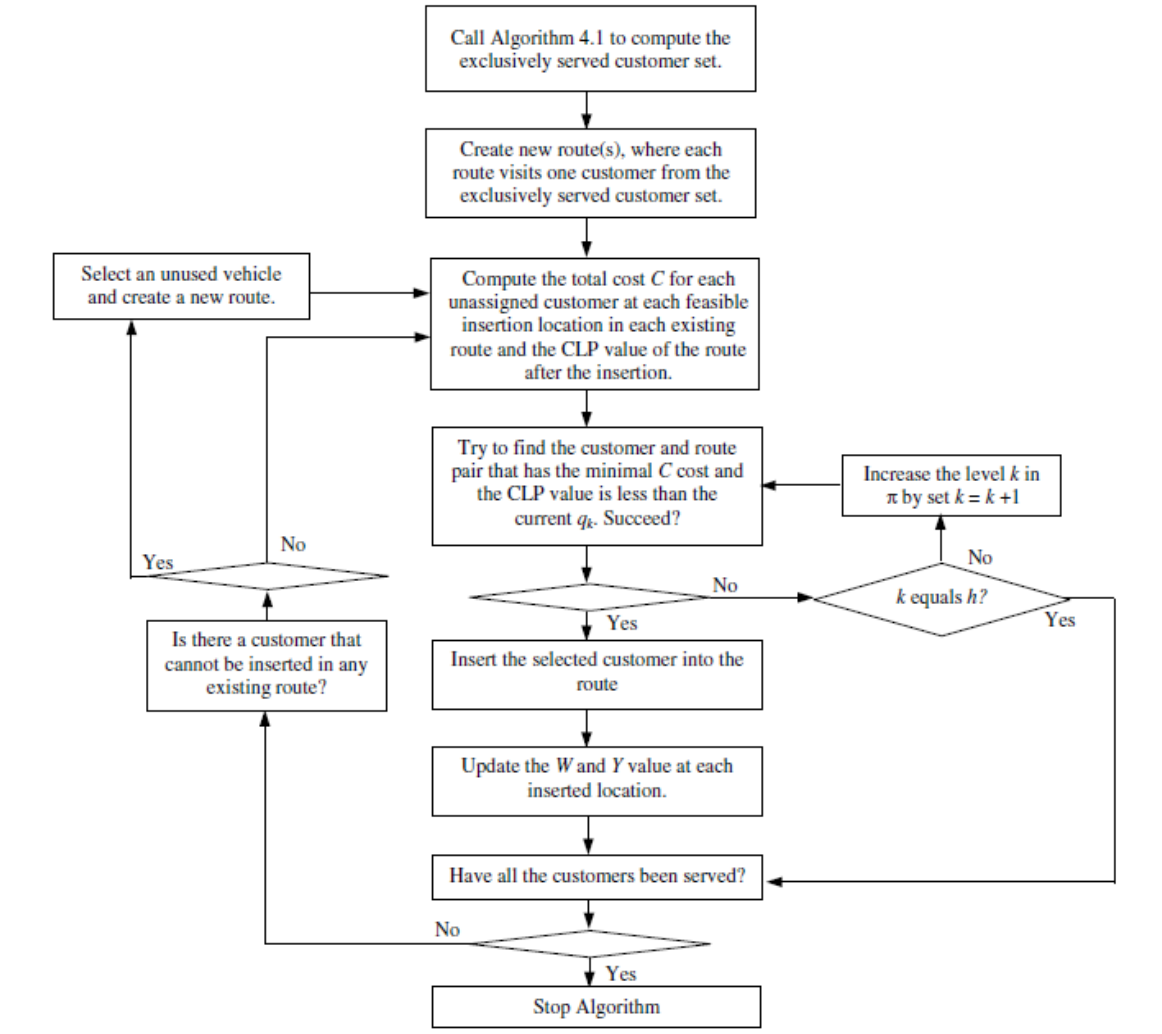
Лако може да се уочи да спољња *while* петља обухвата N итерација, по једну, за сваког од N клијената и да се у свакој итерацији распоређује $j*$-ти клијент. Такође је очигледно да N уствари представља скуп A_2^{n-k}, а *for* циклус између корака 5 и 16 поступак за одређивање функције $f_2(PR^n, k, A_2^{n-k})$. У свакој итерацији се редом претражује сваки елемент из A_2^{n-k} и бира најбоље $j*$. Јасно је такође, да је функција f_3 у ствари процедура која се обавља у корацима 17 и 19, *Insert*$(i*, j*)$ и *Update*$(r*)$.

Уместо да се презентира цео псеудокод, у GCA формулацији би било потребно само прецизно дефинисати f_2, који је иначе овде потпуно непрецизно формулисан. Наиме, циклус између корака 6 и 15 одређује вредност циљне функције C за сваки елемент из A_2^{n-k}. Формулација тог израчунавања је недопустиво непрецизна. У следећим корацима се одређује:

7. **for** $(i-1, i) \in r$ **do**
8. **if** *Feasible*(i,j) and *Profit*$(i,j) > p*$ **then**
9. $r* = r$
10. $i* = i$
11. $j* = j$
12. $p* = Profit(i,j)$
13. **end if**
14. **end for**

клијент, i , из тренутног скупа рута, такав, да постоји веза између њега и посматраног клијента j (*Feasible*(i, j)) и да његово увођење у руте повећава профит (*Profit*(i, j)). На пример, потпуно је нејасан избор клијената i. Из израза ***for*** $(i - 1, i) \in r$ је јасно да су то клијенти са суседним индексима, што је очигледно потпуно погрешно. У ствари, аутори су хтели да кажу да су ти клијенти суседни у рути, при чему је нејасно како се те ознаке додељују клијентима. Недвосмислени закључак је да је формулација једне једноставне ПКХ преопширна, нејасна и непрецизна.

Слични закључци могу да се изведу и из осталих начина презентација ПКХ. На Слици 5-1 је приказан дијаграм тока који описује хеуристику за проблем трговачког путника у неусмереној путној мрежи (проблем 10 из Табеле 5-1), објављен у раду (Ghiani, Lagana, & Musmanno, 2006). Дијаграм је представљен у оригиналу, јер за уочавање проблема са овом формулацијом није потребно залазити у детаље поступка. Прва два блока представљају класичну фазу иницијализације у којој се одређују „*exclusively served customer set*“ (ово је подскуп чворова мреже који образују максималну клику) и „*route(s)*“ (скуп рута које обухватају све чворове из претходног подскупа чворова). Остатак дијаграма представља класичну фазу уметања у којој се, у свакој итерацији, додаје по један неискоришћени чвор, односно „*customer*“. Селекција новог чвора (f_2 из GCA) се врши на основу вредности циљне функције („*minimal C cost*“) под ограничењем („*CLP value is less than the current* q_k“). Најзад, f_3 из GCA дефинише позицију у рути у коју ће бити уметнут нови чвор („*Insert the selected customer into the route*“). Уколико то није могуће („*Succeed?*“), формира се нова рута („*Select an unused vehicle and create a new route*“).



Слика 5-1. ПКХ формулисан преко дијаграма тока **(Ghiani, Lagana, & Musmanno, 2006)**

У наставку су разматрани недостаци конструктивних хеуристика који су евидентни у већини објављених радова. Узрок ових недостатака лежи у начинима формулација тих хеуристика и најчешће су манифестовани кроз:

- редундантну формулацију опште познатих и заједничких делова алгоритама;
- непрецизну формулацију битних делова алгоритама по којима се тај алгоритам разликује од осталих, као последицу претходног недостатка;
- могућност да се одређени поступци у алгоритму могу да спроведу на више различитих начина, као директну последицу непрецизне формулације, што даље повлачи добијање различитих вредности циљне функције при примени истог алгоритма на истим тест инстанцама; и

- необјективну евалуацију алгоритама, јер свака непрецизност и вишезначност омогућује ауторима да повољне варијанте вишезначности искористе на свом алгоритму, а неповољне на алгоритму са којим се пореде.

У наставку су дати примери редундантних, непрецизних и вишезначних формулација ПКХ, објављених у еминентним часописима. Такође је на примерима показана необјективност евалуације познатих ПКХ.

5.1. Редундантност

Псеудокод за два од седам ПКХ за проблем реаговања у ванредним ситуацијама у условима природних катастрофа (Wex, Schryen, Feuerriegel, & Neumann, 2014) су приказани као алгоритми *GREEDY* и *SCHED*1. Оба алгоритма имају базичну ПКХ структуру. Дванаест корака су идентични у оба алгоритма, а једина разлика је у кораку 1.

GREEDY

1 Sort incidents in decreasing order of severity,
$w_1 \geq w_2 \geq ... \geq w_n$ and set $C \leftarrow \{w_1,...,w_n\}$
2 Initialize the current completion time of each rescue unit, rescue units to start at the depot, the ordered list of incidents assigned to unit, i.e.
$C_k \leftarrow 0, \quad \alpha_k \leftarrow 0, \quad \sigma_k \leftarrow 0, \quad \forall k \in K$
3 **for** $l = 1$ **to** n **do**
4 Select incident $i \leftarrow l$ to be processed
5 $K^* \leftarrow \{k \in K \mid cap_{ki} = 1\}$ are all units capable of processing incident
6 **if** $K^* \neq 0$ **then**
7 $unit \leftarrow \arg\min_{k \in K^*} \tau_k + s^k_{\alpha_k,i}$ chooses unit with lowest start time.
8 **else**
9 **return** unsuccessfully (no feasible assignment).
10 **end if**
11 Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit},i} + p^{unit}_i, \quad \alpha_{unit} \leftarrow i, \quad \sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$
12 **end for**
13 **return** $\sigma \leftarrow (\sigma_1,...,\sigma_m)$ being the list of schedules.

SCHED1

1 Sort incidents
$\frac{\overline{p}_1}{w_1} \geq \frac{\overline{p}_2}{w_2} \geq ... \geq \frac{\overline{p}_n}{w_n}$ with $\overline{p}_i \leftarrow \frac{1}{m} \sum_{k \in \{K | cap_{ki}=1\}} p_i^k$ being the average process. time of i
and set $C \leftarrow \left\{ \frac{\overline{p}_1}{w_1}, ..., \frac{\overline{p}_n}{w_n} \right\}$

2 Initialize the current completion time of each rescue unit, rescue units to start at the depot, the ordered list of incidents assigned to unit, i.e.
$C_k \leftarrow 0, \quad \alpha_k \leftarrow 0, \quad \sigma_k \leftarrow 0, \quad \forall k \in K$

3 **for** $l = 1$ **to** n **do**

4 Select incident $i \leftarrow l$ to be processed

5 $K^* \leftarrow \{k \in K \mid cap_{ki} = 1\}$ are all units capable of processing incident

6 **if** $K^* \neq 0$ **then**

7 $unit \leftarrow \arg\min_{k \in K^*} \tau_k + s^k_{\alpha_k, i}$ chooses unit with lowest start time.

8 **else**

9 **return** unsuccessfully (no feasible assignment).

10 **end if**

11 Update $\tau_{unit} \leftarrow \tau_{unit} + s^{unit}_{\alpha_{unit}, i} + p^{unit}_i, \quad \alpha_{unit} \leftarrow i, \quad \sigma_{unit} \leftarrow \sigma_{unit} \cup \{i\}$

12 **end for**

13 **return** $\sigma \leftarrow (\sigma_1, ..., \sigma_m)$ being the list of schedules.

И без познавања суштине проблема, јасно је да се у оба алгоритма у кораку 1 одређује приоритетно уређење πp, да је f_2 дефинисано у кораку 4 и да се селекција f_3 обавља у кораку 7. Алгоритми се разликују само у начину формирања почетног уређења, πp. Како је f_2 тривијална функција (у *к*-тој итерацији се бира број *к*), формулација оба алгоритма захтева само прецизну формулацију функције f_3 и поступак израчунавања вредности циљне функције. Ово није прецизно дефинисано у претходној формулацији и поред толико утрошеног простора.

У литератури постоји велики број сличних примера понављања редундантних ПКХ корака. Јасно је да овакав начин формулисања поступака замагљује јасноћу и прецизност саме формулације. С друге стране, уз формализацију стандардних ПКХ корака, фокус читаоца може да се усмери на суштинске разлике које презентирану хеуристику чине јединственом и различитом од осталих.

5.2. Непрецизност

У великом броју објављених радова се даје псеудокод предложене хеуристике који по структури одговара ПКХ структури, али то из формулације тог псеудокода веома тешко може да се установи. Другим речима, аутори, уместо да само прецизно формулишу функције предложене ПКХ, приказују комплетан код, али непрецизно, тако да непотребно збуњују читаоца. Као пример наводимо псеудокод хеуристике *Algorithm 8: Make Assignment Greedy (S,G, c, f)*:

Algorithm 8: *MakeAssignment_Greedy*(S, G, c, f)

Begin

Jumble (c); // Уреди на случајан начин листу блокова

for $(i = 1,...|V|)$ $req_i = |\{b \mid b \in B^+, (x_b, y_b) = (x_i, y_i)\}|$; // Initialise fill requirements

for (i = 1,…,$|c|$) // For each source of cut, choose an accompanying fill location

begin

$b = c_i$; v = *FindNode*(G, x_b, y_b); // Specify current cut block and associated node

if(*haul_from_bottom*) *UpdateSurface*(G, b, cut); // Simulate cut and update surface

ShortestPath (G, v, *score*); // Evaluate candidate fills using shortest path algorithm

best = 1; j = *NULL*; // Initialise for comparison

for (i = 1,..,$|V|$): **if** (req_i > 0 and $score_i$ < *best*) *best* = $score_i$; $j = i$; // Choose best

req_j -- ; // Decrement fill requirement at chosen grid location

b' = *FindBlock*$(x_j, y_j, z_j + 1)$; // Find correct block at specified grid location

S_i =(b; b'); // Record ith allocation as (b, b')

f.remove(b'); // Remove fill block from list

if (*haul_from_top*) *UpdateSurface* (G, b, cut); // Simulate cut and update surface

UpdateSurface (G, b' , fill); // Simulate fill and update surface

$status_b$ = empty; $status_b 0$ = not empty; // Change status of blocks

end

End.

Када се детаљно проучи рад и сам алгоритам, може да са закључи да се заиста ради о ПКХ и да се итерације у фази уметања спроводе у ***for*** циклусу. Нажалост, ни из рада ни из псеудокода се не може тачно одредити у ком кораку кода се завршава тај циклус. Такође, формулација потпуно збуњује читаоца на основу ознака променљивих које се у псеудокоду користе: бројач циклуса је означен словом *i*, а у циклусу се претражује оно *i* за које *scorei* има најмању вредност. Када се одреди та најмања вредност, одговарајуће *i* се препише у индекс *j* (*j = i*; // *Choose best*) и у следећа два корака користи тај индекс: међутим, у следећем кораку се поново користи *i* које увек на тој позицији псеудокода има вредност |*V*| (***for*** (*i* = 1,..,|*V*|)). Шта су аутори стварно хтели да кажу, остаје читаоцу само да нагађа.

5.3. Вишезначност

Иако по правилу конструктивни алгоритми имају једноставну структуру, у великом броју радова формулација тих алгоритама оставља могућност за различите начине њихове имплементације. Ово има као последицу да се за различита кодирања истог алгоритма добијају различити резултати. Нажалост, аутори тога нису уопште свесни, тако да чак у истом раду у различитим ситуацијама примењују различито кодирање. Као последица, читалац је збуњен када нађе, у најзначајнијим радовима, различите референтне вредности за исту проблематику на истим тест инстанцама. Као пример, Табела 5-2 даје релативна одступања вредности циљне функције HEX алгоритма, која су објављена у три најзначајнија рада из ове области: (Fernandez-Vigas & Framinan, 2014), означен са FF, (Dong, Huang, & Chen, 2008), означен са D и (Ruiz & Stutzle, 2007), означен са RS. Треба да се напомене да су вредности у Табели 5-2 усредњене за групе тест инстанци и да су разлике у резултатима за појединачне инстанце још веће.

Табела 5-2. Некохерентни резултати у различитим алгоритмима

m	5	10	20	5	10	20	5	10	20	10	20	20
n	20	20	20	50	50	50	100	100	100	200	200	500
FF	3.3	4.6	3.73	0.73	5.07	6.65	0.53	2.22	5.34	1.26	4.41	2.07
D	3.09	5.02	3.66	0.78	4.22	5.22	0.38	2.28	3.68	1.08	2.51	1.26
RS	3.35	5.02	3.73	0.84	5.12	6.26	0.46	2.13	5.23	1.43	4.41	2.24

Некохерентност резултата из Табеле 5-2 је последица некоректне формулације алгоритма у сваком од ових радова. У сваком раду је дата комплетна формулација свих корака конструктивног алгоритма, али ни у једном није формулисано оно по чему се те имплементације НЕХ алгоритма међусобно разликују: начина уређења приоритетног редоследа послова по машинама, π_p. Изузетно лоша последица овакве некохерентности је да су овако добијене различите вредности за НЕХ алгоритам коришћене за поређење побољшања која се у радовима предлажу. Како су релативна побољшања у свим случајевима мања од 3%, јасно је колико ове некохерентне вредности утичу на необјективност тих поређења.

Један од важних задатака истраживања у дисертацији је да се квантификује ниво некохерентности у анализираним хеуристикама и покаже како GCA утиче на онемогућавање вишезначних формулација алгоритама.

5.4. Необјективност евалуације временске ефикасности

Евалуација временске ефикасности хеуристика је вероватно најслабија карика у поступцима презентација хеуристика у случајевима када време извршења алгоритма није полиномијална функција од величине обрађиване инстанце. У тим случајевима, једини начин евалуације је експериментално поређење резултата на познатим тест инстанцама. По правилу, аутори кодирају свој алгоритам и алгоритме са којима се пореде и у табелама приказују поређења времена рада и квалитета вредности циљне функције. За разлику од поређења вредности циљне функције, које читалац може да провери тако што сам кодира све поређене алгоритме, поређење временске ефикасности није ни мало тако једноставно. Иако најчешће аутори наводе податке о коришћеном хардверу и софтверу, дужина рада алгоритма у великој мери зависи од умешности програмера да на најбољи начин кодира алгоритам. Светска литература је препуна најразноврснијих експерименталних поређења временске ефикасности у којима нема никаквих гарантованих параметара који би указивали да су та поређења објективна. Необјективности евалуације, поред овога доприноси и чињеница да аутори тримују вредности параметара у метахеуристикама тако да добију најбоље резултате за поређење. Ово тримовање може да траје и данима а при томе нико не може да гарантује да су тако подешени параметри оптимални за неки други скуп

инстанци. Најзад, диверсификација, којом се случајно бира део допустивог скупа за претрагу, значајно доприноси необјективности евалуације. Наиме, аутори обрађују исте инстанце више пута својим алгоритмом, а затим пореде најбоље добијене вредности са вредностима добијеним алгоритмима са којима се пореде. Једина објективна евалуација за овакве случајеве је да се за поређење узме најлошији резултат, сходно опште прихваћеној, *worst case* анализи комплексности.

Као пример колико је евалуација временске сложености произвољна и необјективна, чак и у најеминентнијим радовима, може да послужи рад (Dong, Chen, Huang, & Nowak, 2013). Аутори су приказали податке о утрошку времена, већ поменуте, чувене HEX хеуристике за различите величине тест инстанци (Табела 5-3). Као што је и показано у Табели 4-1, HEX алгоритам је једноставна ПКХ чија је временска сложеност стриктно $O(n^2 \cdot m)$. Значи, не анализира се утрошено време неке хеуристике побољшања, већ једноставног полиномијалног алгоритма, крајње једноставног по формулацији и за кодирање.

Табела 5-3. Поређење времена рада алгоритама DONG и HEXT

m	5	10	20	5	10	20	5	10	20	10	20	20
n	20	20	20	50	50	50	100	100	100	200	200	500
DONG	2	3	6	5	9	16	10	18	32	37	70	208
HEXT	0	0	0	0	0.6	1.2	1.3	2.6	5.3	11	22	121

Добијени резултати намећу интересантне закључке. Сложеност O (n^2m) је код поступка HEXT очигледна. У алгоритму DONG могу да се уоче нелогичности, као што су однос (100, 20) : (200, 10) = 32 : 37, затим однос (200, 20) : (500, 20) = 70 : 208, као и однос (50, 20) : (100, 10) = 16 : 18, итд, из којих би могло да се закључи да је сложеност $O(n^k m)$, где је k мање од 2, што је очигледно немогуће. Чак и да аутори нису били вични кодирању, и направили верзију која ради дуже него што је потребно, остаје потпуно нејасно како временска зависност може да буде таква каквом су је приказали. Јасно је шта може да се очекује у таквим околностима од сложених алгоритама код којих није позната временска сложеност поступка.

6. ГЕНЕРАЛИЗОВАНИ КОНСТРУКТИВНИ АЛГОРИТАМ

ПКХ може једноставно да се прошири додавањем нових аргумената у листу дефинисану у (3). У дисертацији је разматран један облик генерализације која се односи на паралелно праћење више тренутних решења проблема. Ово је у складу са основном идејом да се начин претраге што је могуће више приближи недетерминистичкој машини. Разликоваћемо случајеве у којима се кроз итерације прати једна или више секвенци π^k из Π^k.

6.1. Једна секвенца у итерацији

Међусобно уређење елемената из π^k, према (4) остаје непромењено до краја извршења алгоритма, тј. $\mathbf{ext}^k[x] = \pi^k, k = 1,...,n-1$, где је *x* коначно решење. У *k*-тој итерацији, цео број, *k*+1, се умеће на позиције од π^k. Уметање може да се спроведе на *k*+1 различитих позиција, што резултује стварањем *k*+1 нових секвенци, дужине *k*+1. Функција ξ одређује позицију *l*, која задовољава:

$$C(\mathbf{ins}(\pi^k, k+1, l)) = \xi\{C(\mathbf{ins}(\pi^k, k+1, l')) \mid l' = 1,...,k+1\}. \quad (6)$$

Према томе, на почетку *k*+1-те итерације партиција је:

$$PRT^2(S^n) = (\mathbf{ins}(\pi^k, k+1, l), (k+2,...,n)).$$

У случајевима када више различитих вредности *l* задовољава (6) (у раду су такви случајеви означени као **равноправни случајеви**), редослед уметања постаје веома важан. Разматрани су редоследи *ASC* = 1,2,...,*k*+1 и *DSC* = *k*+1,k,...,1. У оба случаја усваја се прва позиција која задовољава (6).

Алтернативно једначини (6), ξ може да се примени на комплетну секвенцу, дужине *n*; конкретно, низ $(k+2,...,n)$ се додаје на секвенцу дужине *k*+1 из (6):

$$C((\mathbf{ins}(\pi^k, k+1, l)), (k+2,...,n)) = \xi\{C((\mathbf{ins}(\pi^k, k+1, l')), (k+2,...,n)) \mid l' = 1,...,k+1\}. \quad (7)$$

На пример, ако је $PRT^2(S^n) = ((2,3,1),(4,5,6,7))$, према (6), ξ се израчунава за секвенце (4,2,3,1), (2,4,3,1), (2,3,4,1) и (2,3,1,4). Према (7), ξ се израчунава за секвенце (4,2,3,1,5,6,7), (2,4,3,1,5,6,7), (2,3,4,1,5,6,7) и (2,3,1,4,5,6,7).

6.2. Више секвенци у итерацији

У овом случају се дефиниција критеријума селекције ξ проширује на $\xi(R)$ да би се омогућила селекција више секвенци. Конкретно, уколико су секвенце рангиране на основу критеријума ξ, тада R дефинише број прворангираних секвенци за селекцију. Због равноправних ситуација, предуслов за једнозначно рангирање секвенци је једнозначно уређење тих секвенци. На пример, ако је ξ = ***max*** и пет секвенци имају максималну C вредност, рангирање тих пет секвенци се врши на основу њиховог међусобног уређења. Показаће се да дефинисани редослед уметања једнозначно одређује уређење секвенци.

Нека $P^k(\xi(R))$ означава секвенцу $P^k(\xi(R)) = \{\rho_i^k, i = 1,..., w_k\}$, састављену од свих w_k секвенци које задовољавају критеријум селекције у k-1-вој итерацији. Према томе, у k-тој итерацији, број k+1 се умеће на различите позиције у сваки елемент од $P^k(\xi(R))$. Добијене секвенце $P_i^{k+1} = \{\mathbf{ins}(\rho_i^k, k+1, l), l = 1,..., k+1\}$, формирају секвенцу PP^{k+1} састављену од секвенци $(P_1^{k+1},..., P_{w_k}^{k+1})$, кардиналности k+1. Секвенца $P^{k+1}(\xi(R)) \subseteq PP^{k+1}$ се добија када се примени критеријум селекције $\xi(R)$ на секвенце из PP^{k+1}. $P^{k+1}(\xi(R))$ је почетна секвенца у k+1-вој итерацији. Рангирање може да се обави на два начина:

- $\xi(R)$ се примењује на PP^{k+1} као целини; и (8)
- $\xi(R)$ се примењује одвојено за сваки P_i^{k+1}. (9)

Из претходног је јасно да је уређење у PP^k једнозначно дефинисано редоследом потеза уметања. Због тога се првих w_k секвенци селектује на јединствен начин. Опције за R могу да се сврстају у пет група (C_{best} означава меру за најбоље рангиране секвенце у односу на критеријум ξ):

- R_1 – у равноправним ситуацијама, селектују се све секвенце са C_{best};
- $R_2(p_2)$ – селектује се првих p_2 рангираних секвенци;
- $R_3(p_3)$ – дозвољено одступање је p_3%, тј. одступање у односу на C_{best} је $\frac{C_{best}\, p_3}{100}$;

- $R_4(f_4(k))$ – опсег одступања је дефинисан функцијом f_4, где је k број итерације. Селектују се секвенце у опсегу $(1+ f_4(k))C_{best}$;
- $R_5(p_5)$ – поред w_k одабраних секвенци, такође се селектује и следећих p_5 рангираних секвенци. Ово је опциони, додатни критеријум за критеријуме R_1, R_3 и R_4 (напомиње се да је истовремено коришћење R_2 и R_5 тривијално, и резултује у $p_2 + p_5$ селектованих секвенци).

Уколико је број селектованих секвенци већи од предефинисане горње границе, селекција се ограничава на овај гранични број секвенци.

6.3. Промењљив број секвенци у итерацији

Једнозначно уређење од PP^k омогућује генерализацију процедура које паралелно прате различит број секвенци у итерацијама. Број праћених секвенци по итерацијама може бити постављен унапред, или динамички одређиван у току извршења алгоритма. У дисертацији се разматра поједностављена опција: цео број *delay* дефинише да се у првих (*delay* – 1) итерација селектује само по једна секвенца, а да се секвенце из *R* опсега селектују од *delay*-те итерације.

GCA, представљен као прототип ПКХ и дефинисан у (3) може, на основу претходних разматрања, да се дефинише као функција $x \leftarrow GCA(PR^n, F, C, g, \pi p^n, \xi, ord)$. Подразумеване вредности аргумената ове функције су: $F = \Pi^n$; $g = \xi = \min$; πp^n је случајна секвенца из Π^n; $ord = ASC$. Ова формализација омогућује креирање проширења GCA једноставним додавањем опционих аргумената у листу аргумената од GCA. Нека ESARG (*Extended Set of ARGuments*) означава проширени скуп аргумената, $ESARG = \{PR^n, F, C, g, \pi p^n, \xi, ord, ARG\}$, где ARG означава скуп опционих аргумената. У дисертацији користимо скуп од седам опционих аргумената, $ARG = \{\boldsymbol{arg_1}, \ldots, \boldsymbol{arg_7}\}$. Аргумент $\boldsymbol{arg_1}$ генерализује примену функције f_2, док аргумент $\boldsymbol{arg_4}$ дефинише број и начин селекције паралелно праћених секвенци. У случајевима када се паралелно обрађује више од једне секвенце, аргументи $\boldsymbol{arg_2}$, $\boldsymbol{arg_3}$ и $\boldsymbol{arg_5}$ су неопходни јер дефинишу начин симултане обраде секвенци. Аргументи $\boldsymbol{arg_6}$ и $\boldsymbol{arg_7}$ су опциони. Аргументи из *ARG* су:

***arg*$_1$** – функција селекције f_2, подразумевано: $\pi p^n(k+1) \leftarrow f_2(PR^n, k, A_2^{n-k})$;

***arg*$_2$** – максималан број праћених секвенци, подразумевана вредност је 1;

***arg*$_3$** – за селекцију се примењује:

$$\begin{cases} 0, \ једн.(6)\ (подразумевано) \\ 1, \ једн.(7) \\ 2, \ (7)\ само\ у\ случајевима\ када\ (6)\ даје\ равноправне\ ситуације \end{cases};$$

arg*$_4$** - R из $\{R_1, R_2(p_2), R_3(p_3), R_4(f_4)\}$, подразумевано је R_2(arg*$_2$**)

(треба напоменути да је $R_5(p_5)$ дефинисан одвојено као ***arg*$_6$** пошто представља опциони додатни параметар аргументу ***arg*$_4$**);

$$\textbf{\textit{arg}}_5 = \begin{cases} 0\ ако\ је\ \xi(R)\ примењено\ према\ (8)\ (подразумевано) \\ 1\ ако\ је\ \xi(R_1)\ примењено\ према\ (9) \end{cases};$$

***arg*$_6$** - p_5 у $R_5(p_5)$, подразумевана вредност је 0;

***arg*$_7$** - *delay*, подразумевана вредност је 1.

Проширење GCA омогућује креирање различитих хеуристика избором различитих вредности аргумената. Из разлога концизности, уколико аргумент има подразумевану вредност, не приказује се у листи аргумената. Треба напоменути да је алгоритам код кога сви аргументи имају подразумеване вредности идентичан алгоритму који се добија када се *ARG* уклони из листе аргумената.

6.4. Секвенцијална примена различитих GCA

Још једна предност предложеног приступа представља могућност да се формализује секвенцијална примена више алгоритама A_1, ..., A_z који имају различите вредности аргумената. Идеја је да се ови алгоритми примене један за другим и да се **најбоље решење** усвоји као коначно. Најбоље решење подразумева допустиво решење алгоритма, означено као $x(A)$ које задовољава:

$$C(x(A)) = g\{C(x'(A')) \mid x'(A') \in \{x_1(A_1), ..., x_z(A_z)\}.$$

Multi-start верзија конструктивног приступа може бити представљена као алгоритам MULTIGCA(A_1, ..., A_z, t_{max}), где је t_{max} опциони параметар који ограничава утрошак CPU времена. Вредности аргумената сваког од алгоритама могу да буду дефинисане унапред или пак, динамички измењене током извршавања алгоритама. Овај други

приступ захтева постојање поступка за избор вредности аргумената током извршавања алгоритама. Најједноставнији начин је да се дефинише процедура ARGCHOICE којом се прави избор између аргумената два алгоритма.

ARGCHOICE: Избор вредности аргумената

Procedure ARGCHOICE ($ESARG^c$, $ESARG^1$, $ESARG^2$)

1 $x^1 \leftarrow$ GCA($ESARG^c$, $ESARG^1$)

2 $x^2 \leftarrow$ GCA($ESARG^c$, $ESARG^2$)

3 **if** $C(x^1) = g(C(x^1), C(x^2))$ **then return** x^1, $ESARG^1$

4 **else return** x^2, $ESARG^2$

$ESARG^c$ је подскуп вредности аргумената које су једнаке за оба алгоритма, док су $ESARG^1$ и $ESARG^2$ редом преостале вредности аргумената првог и другог алгоритма. Када су вредности одређеног аргумента природни бројеви, може да се примени специјалан случај у виду сукцесивне примене алгоритама. Вредност тог аргумента се мења од 1 до неке вредности *val*, у квантима од 1 (вредности аргумената $\boldsymbol{arg_2}$, $\boldsymbol{arg_6}$ и $\boldsymbol{arg_7}$ могу да се увећавају на овај начин). У том случају вредност аргумента $\boldsymbol{arg_i}$ се у листи аргумената означава као $\overline{\boldsymbol{arg}}_i = \overline{val}$. Треба приметити да $\overline{val}$ може динамички да се израчунава за време увећавања вредности аргумената. Једна могућност је да се заустави увећавање *val* у тренутку када нема побољшања резултата током *d* сукцесивних промена *val* вредности. У експерименталним истраживањима користиће се минималне вредности $\overline{val}$ које дају најбољу вредност циљне функције. Ова минимална вредност аргумента $\boldsymbol{arg_i}$, означена као $\boldsymbol{arg_i^*}$, зависи од вредности осталих аргумената.

6.5. Диверсификација и Табу листе

Увођење проширеног скупа аргумената омогућава да се у конструктивне хеуристике уведе деверсификација са циљем избегавања заглављивања у локалном оптимуму. Флексибилност аргумената пружа разноврсне могућности да се тип диверсификације направи аналогно било којој метахеуристици. При овоме, овако спроведена диверсификација има значајну предност јер се једноставно може да прати који део допустивог скупа се претражује.

Као пример, избором одговарајућих аргумената може да се направи једноставна имплементација диверсификације у хеуристици симулираног каљења. Суштина ове диверсификације је да је у почетним итерацијама, опсег удаљавања од локалног оптимума велики, и да се овај опсег кроз итерације смањује према дефинисаној функцији смањења температуре. Ако се у GCA усвоји да је $\boldsymbol{arg_4} = R_2(p_2)$ и ако се p_2 мења по функцији која одговара функцији смањења температуре код хеуристике симулираног каљења, постиже се исти ефекат диверсификације. Суштинска разлика је у томе што је допустив скуп унапред расподељен на p_2 подскупова, тако да једноставно може да се имплементира паралелно програмирање. Ово је, наравно, само најједноставнији пример имплементације диверсификације, као једне од незаобилазних метода хеуристика. Из оваквог примера могу да се наслуте огромне разноврсне могућности које генерализација конструктивних хеуристика пружа. Тако на пример, избор аргумента $\boldsymbol{arg_1}$ и функције селекције f_3 могу да симулирају скуп околина, дефинисаних у метахеуристици променљивих околина.

Најзад, примена проширеног скупа аргумената не уводи потребу за дефинисањем табу-листа, као у неким хеуристикама побољшања. Кључна чињеница у овако дефинисаном скупу проширених аргумената је да сваки паралелно обрађивани скуп секвенци води директно ка свом локалном оптимуму. Значи, циљ је да локални оптимум једног од ових скупова секвенци буде и глобални оптимум. Уколико то није случај, резултат ће бити неки квалитетан локални оптимум. Како су секвенце које се паралелно обрађују дисјунктне парцијалне секвенце, не постоји могућност да се током поступка појаве две исте секвенце у различитим скуповима. Према томе, нема ни потребе да се формирају листе већ обрађених решења.

7. МЕТОДЕ ЕНУМЕРАЦИЈЕ ПЕРМУТАЦИЈА

Главна препрека за практичну имплементацију хеуристика комбинаторне оптимизације је огромна кардиналност допустивог скупа $F(I)$, која компликује пребројавање и уређење елемената из $F(I)$. Један начин за представљање пермутација од n елемената је преко целог броја N, где је $0 \leq N < n!$, који је у практичним применама могућ када је n довољно мало да N може да се упише у машинску реч; за 32-битне речи $n \leq 12$, а за 64-битне речи $n \leq 20$. Јасно је да је овај тип пребројавања пермутација бескористан у општем случају.

У овом делу рада је дат приказ познатих поступака за енумерацију пермутација. Уочено је да су кораци GCA у ствари кораци за генерисање и селекцију пермутација, при чему познати начини енумерације пермутација нису ни у каквој вези са корацима GCA. Због тога ће у дисертацији бити приказан нови поступак за генерисање пермутација који следи кораке GCA. На основу овог поступка је предложен поступак за енумерацију пермутација и дефинисана веза која постоји између овог поступка и GCA.

7.1. Познате енумерације пермутација

Енумерација пермутација треба да задовољи услов да уређење ознака у енумерацији одговара уређењу самих пермутација. Ово подразумева да је дефинисано уређење пермутација у скупу свих пермутација. Када је реч о уређењу пермутација, најчешће се подразумева тзв. **лексикографско** уређење где нижа позиција у редоследу пермутација подразумева нижу вредност одговарајућег броја у бројном систему основе n. Наиме, свака пермутација π^n може да се представи бројем чија је основа n, као: $\pi(1) \cdot n^{n-1} + \pi(2) \cdot n^{n-2} + ... + \pi(n) \cdot n^0$. Јасно је да ова формула нема практичну вредност, јер су одговарајући бројеви огромни, већ служи да прецизно дефинише да уређење тих бројева дефинише лексикографско уређење пермутација.

Лексикографско уређење пермутација је уређење које се намеће као најлогичније, јер се скоро сва уређења у практичним апликацијама спроводе лексикографски. Нажалост, постоји значајна разлика између уређења, на пример, имена у

телефонском именику и пермутација, а то је поступак за генерисање уређених објеката, односно пермутација. Разлика је у величини скупа који се уређује. Како се код пермутација ради најчешће о огромним скуповима, неопходно је да постоји алгоритам којим се генеришу пермутације према дефинисаном уређењу. Алгоритам за генерисање лексикографски уређених пермутација није сасвим једноставан зато што се прелаз из одређене пермутације у следећу обавља на различите начине, у зависности од тренутног редоследа бројева у пермутацији. Иако је овај недостатак очигледан, у теорији су познати само поступци енумерације пермутација који одговарају лексикографском уређењу пермутација.

У комбинаторној теорији користи се израз за N у факторијалном бројном систему. Број $N<n!$ се представља низом бројева $D = (d_n, d_{n-1}, ..., d_1)$, где је d_i ненегативни цео број мањи од i, док су основе за суседне цифре $n, n - 1, ..., 2, 1$. Конверзија између D и одговарајуће пермутације се обавља преко **Лехмеровог кода** (Lehmer, 1960) или **табеле инверзије** (Knuth D. E., 2011)**.**

Лехмеров код користи чињеницу да постоји $n!=n\times(n-1)\times...\times2\times1$ пермутација од n. Пермутација $\pi^n = (\pi(1),\ldots,\pi(n))$ се обележава низом од n бројева, тако да се за први елемент бира број из скупа од n бројева, за други елемент број из скупа од n - 1 бројева, и тако до последњег елемента који може имати само једну фиксну дозвољену вредност. У овом коду је i-то обележје једнако броју позиција у пермутацији на којима је $\pi(i) > \pi(j),\ j = i+1,...,n$. Пар индекса (i, j), таквих да је $\pi(i) > \pi(j),\ j > i$ се назива инверзија од π и Лехмерово обележје пребројава инверзије за фиксно i и променљиве j. Такође, постоји могућност да се обележје добије тако што се инверзије пребројавају за фиксно j и променљиве i. Овај начин не мења фундаментално Лехмеров тип кодирања, а добијено обележје представља табелу инверзије.

У Табели 7-1 је дат пример одређивања Лехмеровог кода и табеле инверзије за пермутацију 6 3 8 1 4 9 7 2 5. Попуњава се тзв. Рочеов дијаграм за пермутацију: Ознака X се уноси у поље чији је индекс мањи од посматраног броја у пермутацији, док тачка у пољу означава одговарајући број у пермутацији. Када се једном унесе тачка у поље, сва поља испод тог поља се остављају празна. Лехмеров код

пребројава редом ознаке X у редовима, док табела инверзије пребројава редом ознаке X у колонама. На тај начин се, за Лехмеров код добија 5 2 5 0 1 3 2 0 0, док се за табелу инверзије добија 3 6 1 2 4 0 2 0 0.

Табела 7-1. Лехмеров код и табела инверзије

6 3 8 1 4 9 7 2 5

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Лехмеров код**
1	×	×	×	×	×	•				$d_9 = 5$
2	×	×	•							$d_8 = 2$
3	×	×		×	×		×	•		$d_7 = 5$
4	•									$d_6 = 0$
5		×		•						$d_5 = 1$
6		×			×		×		•	$d_4 = 3$
7		×			×		•			$d_3 = 2$
8		•								$d_2 = 0$
9					•					$d_1 = 0$
табела инверзије	3	6	1	2	4	0	2	0	0	

Оба типа енумерације обезбеђују директну везу између уређења ознака и уређења одговарајућих пермутација. Поступак за декодовање, тј. добијање пермутације из њеног кода је очигледан и овде неће бити посебно представљен.

7.2. Нови поступак за генерисање пермутација

Како је лексикографски начин уређења пермутација неподесан за дефинисање алгоритама за уређење пермутација, истраживања су усмерена ка дефинисању неког новог начина за генерисање пермутација којим би се омогућила једноставнија енумерација пермутација. Ово је од великог значаја за само кодирање свих алгоритама у којима постоје захтеви за рад са пермутацијама. Наиме, таква енумерација би омогућила да се програм пише директно за ознаке пермутација што би значајно поправило ефикасност алгоритма.

Идеја за генератор пермутација је потекла баш од структуре ПКХ. Наиме, ПКХ није ништа друго до алгоритам који на одређени начин генерише пермутације, а затим селектује оне, које у итерацији задовољавају постављене критеријуме. Главни део генератора су процедуре PERMGEN1(Z^k, ord) и PERMGEN2(Z^k, ord, n), које од датог низа пермутација $Z^k = (z^{k,1},..., z^{k,|Z^k|}) \subseteq \Pi^k$ генеришу нови низ пермутација уметањем броја $k + 1$ редом на све могуће позиције у Z^k. Параметар *ord* дефинише редослед уметања: када је *ord = ASC*, уметање се врши од позиције 0 ка вишим позицијама, док се за *ord = DSC* уметање врши од позиције $k + 1$ ка нижим позицијама.

Када је аргумент $Z^k = \Pi^k$, PERMGEN1(Π^k, ord) генерише свих (k+1)! пермутација од Π^{k+1}. Низ Π^{k+1}се састоји од k подскупова од $k + 1$ пермутације, при чему i-ти подскуп одговара i-тој пермутацији у Π^k. Ови подскупови су на јединствен начин уређени у Π^{k+1}, на основу уређења одговарајућих низова у Π^k. Сваки од ових подскупова се састоји од $k + 1$ пермутације код које је број $k + 1$ уметнут на једну од $k + 1$ могуће позиције. Уређење ових секвенци је једнозначно дефинисано, према примењеним *ASC* или *DSC* начинима уметања. Према томе, укупан број могућих различитих секвенци је $1!+2!+...+n!$. Детаљан код поступка PERMGEN1 је:

PERMGEN 1: Конструкција Z^{k+1} од датог Z^k

Function PERMGEN1(Z^k, ord)

1 $i \leftarrow 1$
2 $ii \leftarrow 1$
3 **repeat**
4 **if** $ord =$ "*ASC*" **then** $l \leftarrow 1$ **else** $l \leftarrow k+1$
5 **repeat**
6 $z^{k+1,ii} \leftarrow (z^{k,i} \xrightarrow{v(k+1,l)} \mathbf{ins}(z^{k,i}, k+1, l))$
7 **if** $ord =$ "*ASC*" **then** $l \leftarrow l+1$ **else** $l \leftarrow l-1$
8 $ii \leftarrow ii+1$
if $ord =$ "*ASC*" **then (until** $l > k+1$**) else (until** $l < 1$**)**
9 $i \leftarrow i+1$
until $i > |Z^k|$
return Z^{k+1}

Нека је $k = 4$ и $Z^k = \{(1,3,2,4),(1,4,3,2)\}$. Циклус у кораку 3 пролази кроз сваку од две секвенце у Z^k. Циклус у кораку 5 умеће број 5 редом на позиције, и то ако је *ord* = *ASC*, на : $(5,1,3,2,4),(1,5,3,2,4),(1,3,5,2,4),(1,3,2,5,4),(1,3,2,4,5)$ за прву секвенцу од Z^k, односно $(5,1,4,3,2),(1,5,4,3,2),(1,4,5,3,2),(1,4,3,5,2),(1,4,3,2,5)$ за другу секвенцу од Z^k. Ако је *ord* = *DSC*, 5 се умеће редом на позиције $(1,3,2,4,5),(1,3,2,5,4),$ $(1,3,5,2,4),(1,5,3,2,4),(5,1,3,2,4)$ за прву секвенцу од Z^k, односно $(1,4,3,2,5),$ $(1,4,3,5,2),(1,4,5,3,2),(1,5,4,3,2),(5,1,4,3,2)$ за другу секвенцу од Z^k.

Описани алгоритам генерише пермутације дужине $k + 1$. Суштина овог генератора је да су у свим пермутацијама од n елемената, при чему је n произвољан број већи од k, првих $k!$ пермутација исте и једнако уређене, са једином разликом да је пермутација од n елемената допуњена секвенцом $(k + 1, ..., n)$ иза сваке од секвенци дужине k. Идеја је да поступак PERMGEN2(Z^k, ord, n) генерише овакав тип пермутација, дужине n, допуњавањем секвенци добијених са PERMGEN1. Ако се ово спроведе, може да се уочи да овако добијене секвенце од $1!+2!+...+n!$ пермутације морају да садрже одређене пермутације које се више пута понављају. Јасно је да овај случај наступа када је $\pi^{k+1,i}(k+1) = k+1$. Значи, да би се избегла понављања, сви $\pi^{k+1,i}$ код којих је $\pi^{k+1,i}(k+1) = k+1$ морају бити искључени из Π^{k+1}. Овако редуковани Π^{k+1}, означен као $\overline{\Pi}^{k+1}$, има само $k \cdot k!$ секвенци. Тада се секвенца $(k+2,...,n)$ надовезује на сваку од секвенци из $\overline{\Pi}^{k+1}$, и добијена секвенца $\hat{\Pi}^{k+1,n}$ има $k \cdot k!$ различитих секвенци дужине n. Укупан број различитих секвенци је тада:

$$1+1\cdot 1!+...+(n-1)(n-1)! = (\text{индукцијом}) = (n-1)!+n(n-1)!-(n-1)! = n!.$$

Ово може бити илустровано примером са $n = 7$. У том случају, $\Pi^1 = (1)$, и, дописивање бројева редом до 7 даје (1, 2, 3, 4, 5, 6, 7). Уколико се PERMGEN1 примени на Π^1 добија се $\Pi^2 = \{(2, 1), (1, 2)\}$, и дописивање даје: (2, 1, 3, 4, 5, 6, 7) и (1, 2, 3, 4, 5, 6, 7). Последњи низ је већ садржан у допуњеном Π^1, те мора бити искључен, пошто је $\pi^{2,2}(2) = k = 2$.

Процедура PERMGEN2(Z^k, ord, n) за конструкцију $\hat{Z}^{k+1,n} \subseteq \hat{\Pi}^{k+1,n}$ од Z^k се разликује од PERMGEN1 у:

- линија 4: $l \leftarrow k+1$ се мења у $l \leftarrow k$;
- услов **(until** $l > k+1$) се мења **(until** $l = k+1$);
- **return** Z^{k+1} се мења у **return** $\hat{Z}^{k+1,n}$.

Описани поступци представљају генераторе пермутација који успостављају нови начин уређења пермутација код којих је главна особина да су првих *k*! пермутација исте и једнако уређене, независно од укупне дужине пермутација. Другим речима, пермутација већег броја елемената садржи као почетне пермутације све пермутације мањег броја елемената, увек уређене на исти начин. Показаће се да је ово веома важна особина за повезивање енумерације са корацима ПКХ, као и за ефикасно кодирање алгоритама.

7.3. Нова енумерација

Дефиниција 8 уводи нов тип ознака пермутација који омогућује ефикасно пребројавање пермутација, добијених са PERMGEN1 и PERMGEN2.

Дефиниција 8 (**означавање пермутација**): Ознака у форми низа дужине *v*, позитивних целих бројева $L(\pi^n) = (a_1, a_2, ..., a_k, ..., a_v), v \le n$ се придружује пермутацији $\pi^n \in \Pi^n$, где је $a_1 = n$ и $a_k = \mathbf{indx}(\mathbf{ext}^k[\pi^n], k), k = 2, ..., v.$ Дужина *v* низа $L(\pi^n)$ задовољава $\pi^n(v) \neq v$ и $\pi^n(r) = r \mid v < r \le n$. У овом раду, *v* се назива *дубина пермутације*.

На пример, ознака пермутације: $\pi^9 = 5, 3, 1, 4, 2, 6, 7, 8, 9$ је $L(\pi^9) = 9, 2, 1, 3, 1$

$(a_2 = \mathbf{indx}((1,2),2) = 2;\ a_3 = \mathbf{indx}((3,1,2),3) = 1;\ ...\ \pi^9(5) \neq 5; \pi^9(6) = 6; ...\ \pi^9(9) = 9;\ \Rightarrow v = 5).$

Обрнуто, низ π^n се добија од $L(\pi^n)$ спровођењем следеће процедуре:

Корак 1: **Set** $n = a_1, \rho^1(1) = 1, k = 1;$

Корак 2: **Until** $k < v$ **do**: $\{k = k + 1, \rho^k = \mathbf{ins}(\rho^{k-1}, k, a_k)\};$

Корак 3: $\pi^n = (\rho^v, (v+1, ..., n))$.

За претходни пример, π^9 се добија из $L(\pi^9)$ кроз: (1), (1,2), (3,1,2), (3,1,4,2), (5,3,1,4,2), (5,3,1,4,2,6,7,8,9). Може да се уочи да се π^9 састоји од два низа, раздвојена индексом *v*; бројеви на позицијама које су веће од *v* су на истим позицијама као у S^9. Другим речима, позиција *v* је највиша позиција за коју је $a_k < k$.

Дефиниција 9 (***h*-проширење ознаке**): $L(\pi^n, h) = (a_1, a_2, ..., a_v, v+1, ..., h)$ се добија из $L(\pi^n)$ додајући на крај низа редом бројеве од $v + 1$ до h. За горњи пример, $L(\pi^9, 7) = 9, 2, 1, 3, 1, 6, 7.$

Предложена означавања пермутација представљају формализацију процедура PERMGEN1 и PERMGEN2. Према томе, кодирање ПКХ може да се спроведе преко операција на уведеним ознакама. Кроз итерације алгоритма, селекција ξ_2 из f_2 редукује скуп тренутних секвенци, тј. скуп одговарајућих ознака. Ако Λ^r означава скуп ознака свих пермутација чија је дубина једнака r, тада су редуковани скупови ознака кроз итерације: $\Lambda^1(\xi), \Lambda^2(\xi), ..., \Lambda^r(\xi)$. Означимо са $\Lambda_h^r(\xi)$, $h \le r$, скуп r-проширења ознака $\Lambda^h(\xi)$. Коначно, означимо са $\overline{\Lambda}^r(\xi)$ унију свих r-проширења ознака генерисаних до r-те итерације:
$\overline{\Lambda}^r(\xi) = \Lambda_1^r(\xi) \cup ... \cup \Lambda_{r-1}^r(\xi) \cup \Lambda^r(\xi)$ (може да се уочи да важи $\Lambda_r^r(\xi) = \Lambda^r(\xi)$).

На пример, за $\Lambda^1(\xi)$={(9)}, $\Lambda^2(\xi)$={(9,1)}, $\Lambda^3(\xi)$={(9,1,2), (9,2,1)} и $\Lambda^4(\xi)$= {(9,2,1,2), (9,2,1,3)}, важи $\overline{\Lambda}^4(\xi)$={(9,2,3,4), (9,1,3,4), (9,1,2,4), (9,2,1,4), (9,2,1,2), (9,2,1,3)}.

Јасно је да $\overline{\Lambda}^r(\xi)$ формира везу између уређења ознака и јединственог уређења секвенци кроз итерације конструктивних алгоритама. Према описаном, потез уметања у пермутације чије су ознаке из скупа $\overline{\Lambda}^{k-1}(\xi)$ се одиграва у k-тој итерацији; односно, k ознака се формирају од сваке од ознака из тог скупа додавањем једног од бројева 1,..., k на крај тих ознака. Тада се $\overline{\Lambda}^k(\xi)$ формира применом селекције ξ и преласком на следећу итерацију.

Ово је значајна предност за кодирање поступка: ни у једном тренутку се не умеће број у низ који представља ознаку, већ се додавања искључиво врше на крају низа у ознаци.

7.4. Веза између нове енумерације пермутација и поступка PERMGEN

Јасно је да предложени поступак за означавање пермутација омогућује једноставан начин њихове обраде. Иако два PERMGEN поступка имплицирају различите бројеве генерисаних пермутација, дефиниција $\overline{\Lambda}^r(\xi)$ омогућује једнозначно уређење у оба случаја. Веза између уређења пермутација и енумерације директно имплицира једнозначно лексикографско уређење од $L(\pi^n)$:

- уређење $ASC(L(\pi^n))$: прво по растућим вредностима *v*; затим према растућим a_k, при чему позиције са мањим индексима имају виши степен важности у уређењу;
- уређење $DSC(L(\pi^n))$: прво по растућим вредностима *v*; затим према опадајућим a_k, при чему позиције са мањим индексима имају виши степен важности у уређењу.

Према томе, $ASC(L(\pi^n))$ представља формализацију за *ASC* редослед потеза уметања док $DSC(L(\pi^n))$ представља *DSC* редослед потеза уметања. На пример, $ASC(L(\pi^9))$: (9), (9,1), (9,1,1), (9,1,2), (9,2,1), (9,2,2), (9,1,1,1), (9,1,1,2), (9,1,1,3), (9,1,2,1),..., док $DSC(L(\pi^9))$: (9), (9,1), (9,2,2), (9,2,1), (9,1,2), (9,1,1), (9,2,3,3), (9,2,3,2), (9,2,3,1), (9,2,2,3), и тако даље.

Уведене ознаке и њихово лексикографско уређење омогућују прецизно и једноставно дефинисање различитих поступака уметања у конструктивним хеуристикама. Такође, предложена енумерација омогућује једноставно мониторисање дубина селектованих пермутација кроз итерације алгоритма. На веома једноставан начин може да се из ознаке пермутације утврди како је секвенца била формирана кроз итерације. Ово може да се покаже на примеру оптималне секвенце, Тајлардове тест инстанце ta_{21} (Taillard, Scheduling instances, 2004):

- πp^{20} = 13, 11, 2, 12, 9, 8, 20, 16, 14, 15, 10, 6, 17, 7, 1, 18, 4, 5, 3, 19;
- $x = \pi_{opt}^{20}$ = 16, 18, 14, 7, 13, 8, 15, 9, 6, 20, 17, 12, 10, 11, 5, 1, 2, 4, 3, 19;
- $\pi_{opt}^{20} \bullet \pi p^{20} = \pi^{20}$ = 8, 16, 9, 14, 1, 6, 10, 5, 12, 7, 13, 4, 11, 2, 18, 15, 3, 17, 19, 20;
- $L(\pi^{20})$ = 20, 2, 3, 2, 2, 2, 4, 1, 2, 5, 9, 7, 9, 3, 14, 2, 17, 15.

Приоритетно уређење за ову инстанцу је πp^{20}. Оптимална секвенца је $x = \pi_{opt}^{20}$. Примењена је операција $\pi_{opt}^{20} \bullet \pi p^{20}$ да би се добило оптимално уређење индекса. За ову секвенцу, ознака $L(\pi^{20})$ указује следеће:

- потез уметања је први пут коришћен у четвртој итерацији (ознаке на позицијама 2 и 3 су једнаке својим индексима);
- у задње две итерације није дошло до промене секвенце ($L(\pi^{20})$ се састоји од 18 од могућих 20 елемената);
- није било промена у 17-тој итерацији (17 је на позицији 17);
- у већини итерација су се дешавале значајне измене у почетној секвенци πp (бројеви у ознакама имају у просеку мале вредности).

8. ПРОБЛЕМ ФОРМИРАЊА ПРОИЗВОДНИХ ЋЕЛИЈА

Групна технологија је приступ који идентификује оне атрибуте популације који су погодни да се њени чланови групишу у групе, односно фамилије (Ilić, Računarski integrisana proizvodnja, 2015). Чланови сваке поједине групе поседују атрибуте чије су вредности међусобно сличне. Делови из исте фамилије имају сличне производне захтеве. Према томе, проблем формирања производних ћелија, CFP (*Cell Formation Problem*) може да се дефинише као проблем груписања машина у групе, тако да се у истој групи обрађују делови из исте фамилије, а да се минимизира број делова који се обрађују на машинама из различитих група.

CFP припада класи НП (Ballakur & Steudel, 1987). У последње три декаде истраживања су развијени различити CF приступи. Ови приступи могу да се сврстају у следећих пет категорија: обрада бинарних секвенци, кластеровање, примена графова, математичко програмирање, и хеуристике. Свеобухватни прегледи ових приступа су представљени у (King & Nakornchai, 1982), (Greene & Sadowski, 1984), (Singh N. , 1993), (Kamraniand, Parsaie, & Chaudhry, 1993), (Offodile, Mehrez, & Grznar, 1994), (Agarwal & Sarkis, 1997), (Selim, Askin, & Vakharia, 1998), (Mansouri, Moattar-Husseini, & Newman, 2000), (Yin & Yasuda, 2006), (Ghosh, Dan, Sengupta, & Chattopadhyay, 2010), (Ghosh, Sengupta, Chattopadhyay, & Dan, 2010), (Papaioannou & Wilson, 2010) и (Paydar & Saidi-Mehrabad, 2013). У новије време одређене мета-хеуристике постижу најбоље резултате у решавању CFP. Ове хеуристике укључују генетске алгоритме (Solimanpur, Vrat, & Shankar, 2004), (Goncalves & Resende, 2004), (Rajagopalan & Batra, 2006), (Stawowy, 2006), (Saeedi, Solimanpur, Mahdavi, & Javadian, 2010), (Arkat, Hosseini, & Farahani, 2011), (Yin & Khoo, 2011), (Banerjee & Das, 2012), (Ozcelik & Sarac, 2012) и (Saraç & Ozcelik, 2012), групне генетске алгоритме (Brown & Sumichrast, 2001), (Yasuda, Hu, & Yin, 2005) и (James, Brown, & Keeling, 2007), табу претраге (Wu, Low, & Wu, 2004), симулирано каљење (Baykasoglu, 2004), (Wu, Chang, & Yeh, 2009), (Lin, Ying, & Lee, 2010), (Paydar, Mahdavi, Sharafuddin, & Solimanpur, 2010) и (Kia, Baboli, Javadian, Tavakkoli-Moghaddam, Kazemi, & Khorrami, 2012)), мрављe колоније (Islier, 2005), (Kao & Li, 2008) и (Megala, Rajendran, & Gopalan, 2008) као и хибридне хеуристике (Rezaeian, Javadian, Tavakkoli-Moghaddam, & Jolai, 2011), (Ghezavati & Saidi-

Mehrabad, 2011), (Elbenani & Ferland, 2012), (Rafiei & Ghodsi, 2013), (Paydar & Saidi-Mehrabad, 2013) и (Dalfard, 2013).

Према резултатима који су представљени у литератури, најбољи резултати за CFP су добијени коришћењем метахеуристика. Ови алгоритми припадају огромном броју покушаја да се "добију бољи резултати". Да би постигли овај циљ, аутори конструишу алгоритме као мешавину познатих процедура и пореде своје резултате са најбољим резултатима из литературе. Уобичајени сценарио је следећи. Прво, аутори подешавају вредности параметара хеуристика тако да се на тестираним инстанцама добију најбољи резултати. Затим, да би могли да пореде временске ефикасности, кодују све поступке са којима се пореде или уопште не приказују временску ефикасност поступака. Тада прилажу експерименталне резултате који потврђују супериорност предложених алгоритама. Међутим, главни проблем је што стварна иновација у истраживањима не потиче само од још једног новог поступка који даје боље резултате од осталих, ако није добро схваћено и објашњено зашто тај поступак даје добре резултате. Дубља анализа CFP показује постојање природних ограничења у CFP која драматично сужавају допустиви скуп. Заједнички недостатак значајног броја објављених алгоритама за CFP је чињеница да се претрага обавља на комплетном, или недовољно суженом допустивом скупу.

Кардиналност допустивог скупа је дата у (Wang, 1999):

$$\left(\left(\sum_{i=1}^{w}\frac{(-1)^{w-i}i^{r}}{i!(w-i)!}\right)*\left(\sum_{i=1}^{w}\frac{(-1)^{w-i}i^{q}}{i!(w-i)!}\right)\right), \tag{10}$$

где је w број ћелија, q број машина и r број делова. С обзиром да ова величина постаје рачунарски необрадива, чак и за релативно мале бројеве машина и делова, у новије време је истраживање усмерено ка хеуристичким моделима.

У односу на неке најпознатије комбинаторне проблеме из класе НП, CFP показује значајне разлике у смислу погодности за сужавање допустивог скупа решења. На пример, у проблему редоследа послова у линији није могуће, осим у неким тривијалним случајевима, да се на основу улазних података делови допустивог скупа уклоне из даљег разматрања. Увек може да се одреди таква позиција у редоследу новог посла која би противречила претходним закључцима. Насупрот овоме, CFP имплицира скуп разноврсних природних ограничења, која значајно

редукују допустиви скуп. Уколико се на пример, установи јака корелација између две машине, те машине ће припадати истој ћелији независно од операција на осталим машинама. Ове редукције могу да се класификују као:

1. ограничење могућих комбинација, одвојено за машине и делове;
2. ограничење могућих вредности циљне функције;
3. праг квалитета и глобална ограничења;
4. корелација између машина и делова;
5. идентитети, опозити и ћелијски потомци;
6. јединичне ћелије, резидуали и екстреми

Циљ је да се искористе специфичности CFP ради сужавања допустивог скупа и да се предложи поступак који ће бити ефикаснији од постојећих алгоритама за овај проблем, при чему ће свака од активности усмеравати претрагу ка глобалном оптимуму. Показаће се да на свим тест инстанцама које су коришћене у литератури, нови алгоритам даје најбоље познате резултате уз значајно мањи утрошак CPU времена. Примена GCA на CFP има више значајних предности. Као прво, паралелно праћење више решења може веома једноставно да се имплементира као алгоритам паралелног програмирања. Друго, како се алгоритмом у сваком тренутку прати тренутна величина допустивог скупа, у сваком тренутку може да се контролише *trade-off* између квалитета решења и трајања рада програма. Најзад, алгоритам нема потребу за диверсификацијом и табу листама као метахеуристике јер директно циља ка глобалном оптимуму.

8.1. Дефиниције појмова

За дати скуп машина $M = \{m_1,...,m_i,...,m_q\}$ и скуп делова $P = \{p_1,...,p_j,...,p_r\}$, дефинише се *q*x*r* матрица суседства $A = \|a_{i,j}\|$ тако да је

$$a_{i,j} = \begin{cases} 1 \textit{ ако се } p_j \textit{ обрађује на } m_i, \\ 0 \textit{ у осталим случајевима} \end{cases}$$

Случајеви у којима се одређени део више пута обрађује на истој машини нису разматрани. Циљ формирања ћелија је да се конструишу ћелије машина и фамилије делова, а затим да се фамилије делова доделе ћелијама тако да се оптимизује одабрана мера квалитета. Све мере укључују две променљиве: **изузетке** и **празнине**.

Машина m_i и део p_j образују изузетак ако је $a_{i,j} = 1$, али су m_i и p_j распоређени у различите ћелије; m_i и p_j образују празнину ако су m_i и p_j са $a_{i,j} = 0$ распоређени у исту ћелију.

Ефикасност груписања (Kumar & Chandrasekharan, 1990) је опште прихваћена у свим новијим радовима као једна од мера циљне функције за CFP:

$$\eta = \frac{n-e}{n+v} = \frac{o}{n+v}, \tag{11}$$

где је n укупан број операција, e број изузетака, v број празнина и o укупан број операција унутар ћелија које су у раду назване као **интерне операције**.

У скоро свим радовима се процедуре спроводе над матрицом A. Са становишта рачунарске ефикасности, претраживање ове матрице је у највећем броју случајева неекономично, због димензије q x r. Ефикасније претраге се спроводе над *операционом листом*, чија је димензија једнака n. **Операциона листа машина** је секвенца скупова $L^M = (l_1^M,...,l_i^M,...,l_q^M)$ где $l_i^M = \{p^i(1),..., p^i(d_i^M)\}$ означава скуп свих делова који се обрађују на m_i док је d_i^M укупан број операција машине m_i, $(d_1^M + ... + d_i^M + ... + d_q^M = n)$. Дефиниција **операционе листе делова** је аналогна. Подесно је да се користе сва три облика улазних података због јефтиног меморијског простора и значаја смањења потрошње CPU времена.

У дисертацији се користе следеће ознаке:

q – број машина,

r – број делова,

$\rho = \frac{n}{q \cdot r}$ – **укупна густина операција**,

d_i^M и d_j^P – број операција од m_i и p_j респективно,

$\rho_i^M = \frac{d_i^M}{r}$ и $\rho_j^P = \frac{d_j^P}{q}$ – **густина операција** на m_i и p_j респективно,

$C = \{c_1,...,c_k,...,c_w\}$ – скуп ћелија, где је w број ћелија,

n_k^M и n_k^P - број машина у ћелији k и број делова у ћелији k,

d^C – укупна димензија ћелија, $d^C = d_1^C + ... + d_k^C + ... + d_w^C = \sum_{k=1}^{w} (n_k^M \cdot n_k^P)$,

o_i^M и o_j^P – број интерних операција на m_i и p_j респективно,

e_i^M и e_j^P – број изузетака од m_i и p_j респективно,

v_i^M и v_j^P – број празнина од m_i и p_j респективно,

o – укупан број интерних операција, $o = \sum_{k=1}^{w} o_k = \sum_{i=1}^{q} o_i^M = \sum_{j=1}^{r} o_j^P$

v – укупан број празнина, $v = \sum_{k=1}^{w} v_k = \sum_{k=1}^{w} d_k^C - \sum_{i=1}^{q} o_i^M = \sum_{k=1}^{w} d_k^C - \sum_{j=1}^{r} o_j^P$

e – укупан број изузетака, $e = \sum_{k=1}^{w} e_k = q \cdot r - \sum_{i=1}^{q} o_i^M = q \cdot r - \sum_{j=1}^{r} o_j^P$

Ефикасност груписања може да се изрази као:

$$\eta = \frac{o}{n + d^C - o} = \frac{\sum_{k=1}^{w} o_k}{n + \sum_{k=1}^{w} (n_k^M \cdot n_k^P) - \sum_{k=1}^{w} o_k} \quad . \tag{12}$$

Овај израз истиче најважнија ограничења која сужавају допустиви скуп. За дате димензије ћелија, максимизација једне промењиве, o, директно проузрокује максимизацију ефикасности груписања. Према томе, ограничење могућих димензија ћелија проузрокује да значајан део скупа, датог у (10) није допустив.

8.2. Сужавање допустивог скупа

Проблеми комбинаторне оптимизације из класе НП садрже огромне допустиве скупове већ за мале величине инстанци. Како је тотална претрага таквих скупова немогућа у разумном времену рада рачунара, хеуристике претражују само део допустивог скупа, покушавајући да усмере претрагу у правцу оптималне вредности циљне функције. Веома често се у ту сврху примењују опште метахеуристике, прилагођене конкретном проблему. Заједнички недостатак већине оваквих алгоритама је да не користе довољно специфичности конкретног проблема у циљу повећања ефикасности поступка. Док су за неке проблеме комбинаторне оптимизације овакве могућности незнатне, CFP садржи разноврсна ограничења, која ће у овом делу дисертације бити детаљно разматрана. Резултати оваквих

истраживања су искоришћени за генерисање новог алгоритма за CFP. Међутим, резултати ових истраживања могу да се примене на било коју постојећу хеуристику за CFP, што би сигурно резултовало у побољшању квалитета тих хеуристика, било у погледу вредности циљне функције, било у погледу временске ефикасности.

8.2.1 *Генерисање скупа партиција*

Прва ствар коју је потребно познавати приликом покушаја да се конструишу нови алгоритми комбинаторне оптимизације је величина допустивог скупа у функцији величине проблема. Без познавања ове зависности, настају чести случајеви у литератури да се спроводи много дужа и неквалитетнија обрада над инстанцама које је у одређеном тренутку рада програма могуће обрадити тоталном претрагом. У скоро свим радовима везаним за CFP постоје инстанце за које су предложеним алгоритмима добијени лошији резултати уз дуже време рада рачунара него да је на те инстанце примењена тотална претрага. Циљ приступа који је предложен у овом раду је да се током извршења алгоритма допустиви скуп смањује и да се у тренутку када се кардиналност допустивог скупа спусти на одговарајућу вредност, примени тотална претрага. Као референтан податак може да послужи да се на сасвим просечном рачунару (Intel Core i5-5200U, 6GB RAM), циклус, у коме се 10^9 пута уписује нека вредност у променљиву, обавља за 2.3*sec*.

Формирање ћелија са машинама и фамилија делова представља познат **проблем партиционисања скупа** (*set partitioning problem*). **Партиције скупа** представљају начине да се тај скуп третира као унија непразних, дисјунктних подскупова названих **блокови** (Knuth D. E., 2011). На пример, пет суштински различитих партиција скупа {1,2,3} може да се напише у облику 123, 12|3, 13|2, 1|23, 1|2|3, користећи вертикалну линију за раздвајање једног блока од другог. У овој листи, елементи блокова, као и сами блокови, могу да буду написани у било ком редоследу, пошто 13|2 и 31|2 и 2|13 и 2|31 представљају исту партицију. У овом примеру је презентација стандардизована усвајањем да се елементи сваког блока ређају у растућем поретку и да су блокови међусобно уређени према растућем поретку њихових најмањих елемената. Усвајајући ову конвенцију, може да се формулише једноставан генератор свих партиција, PARTGEN, скупа {1,2,…,*k*} ако су дате све партиције скупа {1,2,…,*k*-1}:

PARTGEN:

За сваку партицију скупа {1,2,…,*k*-1}:

1. постави *k* као последњи елемент блокова, по једном за сваки блок;

2. додај нови блок "|k" на крају посматране партиције.

Следећа партиција.

Користећи PARTGEN, може итеративно да се конструише партиција било ког датог скупа:

{1} – 1

{1,2} – 12 1|2

{1,2,3} – 123 12|3 13|2 1|23 1|2|3

{1,2,3,4} – 1234 123|4 124|3 12|34 12|3|4 134|2 13|24 13|2|4 14|23 1|234 1|23|4 14|2|3 1|24|3 1|2|34 1|2|3|4

На основу овог једноставног генератора може да се конструише троугаона матрица $K = \left\| k_{i,j} \right\|$ где $k_{i,j}$ садржи број различитих партиција скупа од *i* елемената са тачно *j* блокова. Елементи $k_{i,j}$ задовољавају: $k_{i,1} = k_{i,i} = 1$; $k_{i,j} = k_{i-1,j-1} + j \cdot k_{i-1,j}$. Матрица K, за скупове до 15 елемената је приказана у Табели 8-1.

Табела 8-1. Број партиција за дате кардиналност скупа и број блокова

	1	2	3	4	5	6	7	8	9	10
1	1									
2	1	1								
3	1	3	1							
4	1	7	6	1						
5	1	15	25	10	1					
6	1	31	90	65	15	1				
7	1	63	301	350	140	21	1			
8	1	127	966	1,701	1,050	266	28	1		
9	1	255	3,025	7,770	6,951	2,646	462	36	1	
10	1	511	9,330	34,105	42,525	22,827	5,880	750	45	1
11	1	1,023	28,501	145,750	246,730	179,487	63,987	11,880	1,155	55
12	1	2,047	86,526	611,501	1,379,400	1,323,652	627,396	159,027	22,275	1,705
13	1	4,095	261,625	2,532,530	7,508,501	9,321,312	5,715,424	1,899,612	359,502	39,325
14	1	8,191	788,970	10,391,745	40,075,035	63,436,373	49,329,280	20,912,320	5,135,130	752,752
15	1	16,383	2,375,101	42,355,950	210,766,920	420,693,273	408,741,333	216,627,840	67,128,490	12,662,650

Било које ограничење по питању партиционисања смањује вредности елемената у табели. На пример, уколико блок треба да садржи бар два елемента, бришу се све партиције у којима постоји блок од само једног елемента. Тако на пример, у том случају $k_{6,3}$ са 90 пада на 15. Подаци из Табеле 8-1, као и саме партиције могу бити

запамћени у бази података, тако да њихова обрада не оптерећује време извршавања самог алгоритма.

На први поглед, према Табели 8-1, укупан број комбинација се добија множењем одговарајућег елемента матрице за машине са одговарајућим елементом матрице за делове. На пример, за случај од 6 машина и 10 делова за три формиране ћелије, треба помножити 90 и 9330. Међутим, захваљујући израженој корелацији између партиционисања делова и машина, која је обрађена у одељку 8.2.4, обрађивани број комбинација ће бити само 90, јер тај запис одговара мањем од бројева машина и делова. И поред тога, како је 90 * 9330 = 839700, новим алгоритмом ће се обавити тотална претрага, јер се овај број операција практично тренутно извршава на рачунару.

8.2.2 *Дискретан скуп могућих вредности циљне функције*

Једначине (11) и (12) јасно истичу чињеницу да је скуп вредности циљне функције дискретан и редак. Могуће вредности променљиве o су $\{1,2,\ldots,n\}$ (при томе, вредности $o \le \frac{n}{2}$ подразумевају неквалитетно груписање у ћелије). Следеће тврђење дефинише довољан услов да би се поправила вредност ефикасности груписања ако се o и v промене, редом, за Δo и Δv.

Став 1. Ефикасност груписања η се повећава када је:

1. $\frac{\Delta o}{\Delta v} \ge \eta$ за $\Delta o > 0 \wedge \Delta v > 0$
2. $\frac{\Delta o}{\Delta v} < \eta$ за $\Delta o \le 0 \wedge \Delta v < 0$

Доказ: $\frac{o+\Delta o}{n+v+\Delta v} \ge \frac{o}{n+v} = \eta \quad \Rightarrow \quad \Delta o \cdot (n+v) \ge \Delta v \cdot o \quad \Rightarrow \quad \Delta o \ge \eta \cdot \Delta v.$

У тривијалном случају када је $\Delta v = 0$, било које $\Delta o > 0$ поправља ефикасност. Ово комплетира доказ. Важан закључак је да количник $\frac{\Delta o}{\Delta v}$ поставља меру квалитета побољшања.

Претпоставимо да је извршавањем програма добијено партиционосање са вредностима *o* и *v*. Било који резултат који поправња вредност циљне функције подразумева пар вредности $(\Delta o, \Delta v)$ које задовољавају услове Става 1. Сви овакви парови одређују скуп од *c* могућих вредности циљне функције, $\mathrm{H} = \{\eta_1, ..., \eta_i, ..., \eta_c\}$. Како је, према (12), $d^C = o \cdot \frac{\eta + 1}{\eta} - n$, d^C има само *c* могућих вредности. Уколико су d^C вредности сачуване у бази података, довољно је да се претраже само решења која одговарају овим вредностима да би се добило оптимално решење.

Предложени приступ може да се покаже на примеру инстанце из литературе. То је инстанца *No.1* (King & Nakornchai, 1982) са 5 машина и 7 делова, из опште прихваћеног скупа од 35 инстанци из литературе. У скоро свим објављеним радовима најбоља партиција даје вредност циљне функције 73.68% (процентуална представа вредности 0.7368). Ова партиције је приказана у Табели 8-2.

Табела 8-2. Партиција за инстанцу *No.1*

	2	4	5	6	1	3	7
1	1	1	1	1			
4	1	1		1			
2					1	1	
3				1	1	1	1
5			1		1		1

За ову партицију је *n* = 16, *o* = 14 и *v* = 3. Најближе веће вредности ефикасности груписања се добијају за $\eta(\Delta o, \Delta v) = \eta(-2,-3) = 75\%$; $\eta(-1,-2) = 76.47\%$ и $\eta(0,-1) = 77.77\%$. За прво побољшање, $d^C(-2,-3) = 12$. За d^C = 12, постоје две могуће партиције са димензијама ћелија 2*3 + 2*2 + 1*2 и 2*4 + 2*1 + 1*2. Значи, партиције имају три ћелије, без празнина. Избор се сужава на једну могућност, која је приказана у Табели 8-3. За већа побољшања, $d^C(-1,-2) = 14$ и $d^C(0,-1) = 16$ и не постоје партиције са овим вредностима. Само у (Pailla, Trindade, Parada, & Ochi, 2010), је добијена партиција са вредношћу 75% , која је истовремено и оптимална вредност за ову инстанцу.

Табела 8-3. Побољшана партиција за инстанцу *No.1*

	2	4	6	1	3	5	7
1	1	1	1			1	
4	1	1	1				
2				1	1		
3			1	1	1		1
5				1		1	1

Кардиналности блокова у партицијама одговарају елементима матрице K, која је приказана у Табели 8-1. Сваком елементу $k_{i,j}$ матрице K одговара скуп секвенци $S_{i,j}$, у облику $(ns, b_1, ..., b_l, ..., b_i)$ где b_l означава број блокова дужине l у посматраној партицији, док ns представља број таквих секвенци. Јасно је да важи: $b_1 + ... + b_l + ... + b_i = j$. Табела 8-4 садржи ове податке за скупове до 6 елемената.

Табела 8-4. Секвенце кардиналности блокова

	1	2	3	4	5	6
1	1 1					
2	1 0 1	1 0 2				
3	1 0 0 1	3 1 1 0	1 3 0 0			
4	1 0 0 0 1	4 1 0 1 0	6 2 1 0 0	1 4 0 0 0		
		3 0 2 0 0				
5	1 0 0 0 0 1	5 1 0 0 1 0	1 0 2 0 1 0 0	1 0 3 1 0 0 0	1 5 0 0 0 0	
		1 0 0 1 1 0 0	1 5 1 2 0 0 0			
6	1 0 0 0 0 0 1	6 1 0 0 0 1 0	1 5 2 0 0 1 0 0	2 0 3 0 1 0 0 0	1 5 4 1 0 0 0 0	1 6 0 0 0 0 0
		1 5 0 1 0 1 0 0	6 0 1 1 1 0 0 0	4 5 2 2 0 0 0 0		
		1 0 0 0 2 0 0 0	1 5 0 3 0 0 0 0			

Поступак за формирање секвенци је једноставан и аналоган поступку за израчунавање вредности $k_{i,j}$ из матрице K. За формирање секвенци које одговарају елементу $k_{i,j}$, модификују се елементи који одговарају елементима $k_{i-1,j-1}$ и $k_{i-1,j}$. Од сваке секвенце која одговара елементу $k_{i-1,j-1}$, формира се по једна секвенца која одговара елементу $k_{i,j}$ на следећи начин: b_1 се повећава за један, а нула се додаје на крај секвенце. Секвенце које одговарају елементу $k_{i-1,j}$ се сукцесивно модификују мењањем $b_l > 0$ на следећи начин: за сваки $b_l > 0$, формира се b_l секвенци од $k_{i,j}$ са $b_l' = b_l - 1$ и $b_{l+1}' = b_{l+1} + 1$. Такође, на крају сваке секвенце се додаје нула.

На пример, секвенце (5,3) се формирају као:

4 x 1 0 1 0 → 4 x 2 0 1 0 0

3 x 0 2 0 0 → 3 x 1 2 0 0 0

6 x 2 1 0 0 → 6 x 2 x 1 2 0 0 0

6 x 2 1 0 0 → 6 x 1 x 2 0 1 0 0.

Груписањем бројева истих инстанци се добијају две секвенце: (10 2 0 1 0 0) и (15 1 2 0 0 0).

Иако је поступак формирања ових секвенци елементаран и може да се извршава током спровођења алгоритма, ефикасније је да се ови подаци израчунају једном и сместе у базу података. Могућности савремених база података, чак и на најскромнијим рачунарима су огромне, и увек је неспорно повећавати временску ефикасност на рачун јефтиног меморијског заузећа. У следећим одељцима ће се разматрати разноврсне могућности сужавања допустивог скупа које ће омогућити да за све тест инстанце из литературе, скромна база података на просечном рачунару, практично тренутно враћа све тражене податке.

8.2.3 *Праг квалитета и глобална ограничења*

За разлику од великог броја разноврсних проблема комбинаторне оптимизације, за CFP може унапред да се постави праг за ефикасност груписања испод кога нема сврхе формирања ћелија. На пример, процес обраде делова на машинама може да буде такав да није могуће груписати ћелије тако да се већи број операција обавља унутар ћелија него изван ћелија. Јасно је да је добит од оваквог груписања мала. Такође, може да се ограничи број празнина у односу на вредност d^C. Ове границе дефинишу вредности ефикасности груписања испод којих се не разматра формирање ћелија. Уколико би се усвојио праг испод кога се формирање ћелија не разматра, аутоматски би се из обраде избациле тзв. "најтеже инстанце". Наиме, што је оптимална вредност циљне функције мања, то је проблем тежи, јер подразумева непостојање јасно изражених блокова. Ово је вероватно јединствена погодност за решавање CFP у односу на друге проблеме комбинаторне оптимизације.

Са становишта операционог менаџмента, менаџер не може да се задовољи информацијом да улазни подаци не омогућавају квалитетно формирање ћелија. У

тим случајевима треба применити другачије приступе. Једна од могућности је да се из разматрања уклоне машине и делови који су неподесни за груписања па да се поступак формирања ћелија примени на преостале машине и делове. У случајевима када је густина операција велика, корисно је размотрити мултипликацију појединих машина. Најзад, веома користан приступ је да се машине међусобно уреде тако да суседне машине обрађују што је могуће више заједничких делова. Овај приступ је приказан у потпоглављу 8.4.

Густина операција одређује дозвољене вредности од d^C. Уколико је праг ефикасности груписања $\hat{\eta}$, границе од d^C се добијају из (12):

$$\hat{\eta} \le \frac{n-e}{d^C+e} \le 1 \quad \Rightarrow \quad n-2e \le d^C \le \frac{n-(\hat{\eta}+1)\cdot e}{\hat{\eta}}. \tag{13}$$

Према томе, ако је на пример $\hat{\eta} = 0.5$, тада d^C може имати само $n - e$ различитих вредности. Ово ограничење даље сужава скуп могућих вредности димензија ћелија. Ако се ћелије које садрже само једну машину или један део не разматрају, најмања димензија ћелија је 2x2. У том случају је максималан број ћелија $w = \left\lfloor \frac{q}{2} \right\rfloor$, подразумевајући, без губитка општости да је број машина мањи или једнак броју делова. Пошто, за дато w, максимална вредност d^C подразумева (w-1) ћелија са димензијом 2x2, а минимално d^C подразумева ћелије једнаких димензија, следи да је:

$$\frac{q\cdot r}{w} \le d^c \le 4(w-1)+[q-2(w-1)][r-2(w-1)] \tag{14}$$

Неједначине (13) и (14) дефинишу ограничења која значајно сужавају избор могућих вредности из Табеле 8-1 и Табеле 8-4.

Слични закључци могу да се изведу и за случајеве када су ћелије од једног елемента дозвољене. У раду (Chandrasekharan & Rajagopalan, 1986), је дата горња граница независних ћелија за тај случај:

$$w \le 1 + \left\lfloor \frac{(q+r-1)-\sqrt{[(q+r-1)^2-4(qr-n)]}}{2} \right\rfloor.$$

8.2.4 *Корелација између машина и делова*

Број комбинација у тоталној претрази за CFP се добија као производ елемената матрица K, за машине и делове. Ова нелинеарност смањује величину инстанци које могу да се обраде у разумном времену рада рачунара. На срећу, специфичност CFP омогућује да се делови групишу на основу датог груписаења машина и обрнуто. Јасно је да се због овога, тотална претрага треба да спроведе на ентитету мање величине. У наставку ће се сматрати, без губитка општости, да је број машина мањи од броја делова.

Корелација између машина и делова је једна од малог броја специфичности CFP, које су разматране у литератури. У раду (Li, Baki, & Aneja, 2010), доказана је важна Теорема да је, за дату партицију машина, сваки део p_j оптимално распоређен у ћелију c_k ако важи:

$$k = \arg\min\{e_{j,t}^{P} + \eta * v_{j,t}^{P}\}, t = 1,..., w. \quad (15)$$

где је $e_{j,t}^{P}$ број изузетака који су формирани распоређивањем p_j у c_k док је $v_{j,t}^{P}$ број празнина које су формиране распоређивањем p_j у c_k. У радовима који се баве корелацијом између машина и делова, правила за распоређивање се дефинишу у случајевима када тренутно груписање није оптимално. Због овога је, $\eta *$ тренутна, а не оптимална вредност, што даље подразумева да распоређивање делова зависи од распоређивања осталих делова. Јасно је да због овога распоређивање мора да се понавља све док се не добије стационаран случај. Због овога временска сложеност оваквог поступка не може унапред да се дефинише, док је у најгорем случају експоненцијална.

Следеће тврђење уводи јачи услов за распоређивање делова:

Став 2. Довољан услов**, з**а дато груписање машина**,** да је p_j оптимално распоређен у c_k је:

$$o_{j,t}^{P} - v_{j,t}^{P} - e_{j,t}^{P} > 0. \quad (16)$$

Ово распоређивање не зависи од распоређивања осталих делова.

Доказ: Најгори случај за распоређивање p_j у c_k је када постоји ћелија која садржи само машине које нису у c_k а обрађују p_j. Ефикасност груписања у том случају је

$\eta = \frac{o + \Delta e}{n + v}$. Према томе, довољан услов да се p_j оптимално распореди у c_k је $\frac{o + \Delta o}{n + v + \Delta v} \geq \frac{o + \Delta e}{n + v} = \eta \leq 1 \quad \Rightarrow \quad \Delta o \geq \Delta v + \Delta e$. Овај услов не зависи од η, па распоређивање p_j у c_k не зависи од распоређивања осталих делова. Ово комплетира доказ.

Најважнија предност предложеног услова је да распоређивање у том случају не зависи од тренутне вредности ефикасности груписања. Као последица, распоређивање одређеног дела не мења претходно груписање делова. Ова разматрања нам омогућују да дефинишемо процедуру ASSIGNMENT којом се делови оптимално распоређују у ћелије ако је дато груписање машина. Овај поступак не захтева никакво почетно распоређивање делова.

ASSIGNMENT: $\Pi \leftarrow ASSIGNMENT(\pi)$

1. За дато груписање машина, π, распореди све делове који задовољавају (16).
2. Нераспоређене делове распореди у ћелије користећи (15).
3. **излаз:** груписање у ћелије Π.

8.2.5 *Идентитети, опозити и ћелијски потомци*

У оперативним листама и матрици A су скривене разноврсне везе које постоје између машина односно делова, које одређују њихово груписње у ћелије. У овом одељку ће се размотрити могућности да се ове везе уоче и квантификују, па да се на основу њих редукује допустиви скуп. Без губитка општости ће се разматрати скуп машина, подразумевајући да машина има мање од делова. Предложени скуп поступака за редукцију скупа машина није коначан, сигурно могу постојати нови критеријуми који ће ове скупове даље сужавати. Главни циљ рада је да предложи један потпуно нови приступ за CFP који ће бити ефикаснији од постојећих и који ће омогућити потпуну контролу над обрадом. Овај скуп поступака је сасвим довољан за ефикасну примену на све тест инстанце које су до сада коришћене у литератури.

У литератури су коришћени коефицијенти сличности, који су изузетно погодна величина за ову сврху. Нажалост, могућности ових коефицијената нису ни

приближно довољно искоришћене. Мера коефицијента сличности између машина m_i и m_j се дефинише као $s_{i,j}^M = \frac{đ_{i,j}^P}{đ_{i,j}^P + ć_{i,j}^P + č_{i,j}^P}, 0 \le s_{i,j}^M \le 1$, где $đ_{i,j}^P$ представља број делова који се обрађују и на m_i и на m_j; док је $ć_{i,j}^P$ број делова који се обрађују на m_i, али не и на m_j, а $č_{i,j}^P$ број делова који се обрађују на m_j, али не и на m_i (аналогна дефиниција важи за меру сличности између делова). Коефицијент сличности је моћна мера, али количник у њеном изразу крије неке значајне информације. Из тог разлога ће уз $s_{i,j}$, такође бити коришћени и $đ_{i,j}^P$, $ć_{i,j}^P$, $č_{i,j}^P$, d_i^M, d_j^P, ρ_i^M и ρ_j^P.

Граничне вредности ове мере означавају: **идентитете** за $s_{i,j} = 1$ (два идентична реда односно колоне у матрици A) и **опозите** за $s_{i,j} = 0$ (логичка операција ИЛИ између бинарног записа ова два реда, односно колоне матрице A даје вредност FALSE). Гранични случајеви директно могу да се примене на смањење реда проблема. Нека посматрана инстанца има скуп од l машина између којих постоји међусобни идентитет. Из почетних података може, без икаквог информационог губитка да се уклони $l - 1$ ових машина поступком EXSTRACT.

EXSTRACT:

1. оформи подскупове идентитета, $T = \{T_1,\ldots, T_i,\ldots, T_g\}$. Запамти кардиналност $\{|T_1|,\ldots, |T_i|,\ldots, |T_g|\}$ ових подскупова;
2. уклони произвољних $|T_i| - 1$, $i = 1,\ldots, g$ машина из сваког T_i. Нека су преостале машине $t_1^*,\ldots,t_i^*,\ldots,t_g^*$;
3. у релацијама за израчунавање ефикасности груписања, сабирци који одговарају t_i^* се множе са $|T_i|$.

На овај начин се директно смањује величина проблема. По завршетку оптимизације, избачене машине се дописују у ћелију у којој се налази њихов неизбрисани представник. Када су вредности коефицијената сличности мањи од јединице, машине се не могу са сигурношћу груписати као у случају идентитета. Међутим, што је коефицијент сличности између две машине већи, већа је и вероватноћа да те машине буду у истој ћелији. У великом броју радова се коефицијенти сличности користе за хијерархијско кластеровање машина, али најчешће над матрицом A, обједињено, за машине и делове. Поступак GROUPING има улазне параметре $S_{\min}$,

минималну дозвољену вредност коефицијента сличности између две машине и $comb_{max}$, максималан дозвољен број комбинација.

GROUPING(S_{min}, $comb_{max}$):

1. Нађи највећи неискоришћен $s_{i,j}$. У случају када више парова машина имају ту вредност, обради их према опадајућим вредностим од $n_i + n_j$. **Ако је** $s_{i,j} < S_{min}$ **тада изађи из програма, иначе, ако** су m_i и m_j у различитим ћелијама и **ако**
   - m_i и m_j не припадају ни једној ћелији, оформи нову ћелију од ових машина;
   - једна од машина, нпр. m_i не припада ни једној ћелији, додели је ћелији c_k која садржи m_j **ако и само ако** је број делова који се обрађују на m_i и нису у c_k мањи од броја делова који се обрађују на m_j и нису у c_k;
   - m_i и m_j припадају различитим ћелијама, спој те две ћелије **ако и само ако** су сви коефицијенти сличности у овако спојеној ћелији већи од S_{min}.
2. На основу кардиналности нових ћелија, израчунај број пермутација тоталне претраге. Уколико је тај број мањи од $comb_{max}$, **тада изађи из програма**, **иначе, иди на** 1.

Треба да се напомене важна чињеница. Избор S_{min} одређује квалитет крајње вредности циљне функције. Ова вредност је сигурно оптимална само када је $S_{min} = 1$. Као и у осталим хеуристикама, и у овом случају мора да постоји *trade-off* између утрошеног времена и квалитета решења. Суштинска разлика је да се у предложеном приступу овај *trade-off* може динамички да контролише те да се претраге спроводе на суженом допустивом скупу. За боље резултате се бира већа вредност за S_{min} и главна обрада се препушта хеуристици. За ово може да се користи произвољна хеуристика побољшања, уз напомену да ће свака од ових хеуристика бити ефикаснија на суженом допустивом скупу. Такође може да се користи и адаптивни S_{min} чија вредност се одређује на основу глобалних параметара. Нови алгоритам, предложен у дисертацији, користи $S_{min} = 0.6$.

Уколико су два опозита лоцирана у исту ћелију, све интерне операције једног опозита су празнине другог. Оптимално груписање може да има два опозита у истој

ћелији само у посебним случајевима, на пример када постоји машина која обрађује скоро све делове оба опозита или када је сума одговарајућих интерних операција мала у поређењу са просечном густином операција. Опозити могу ефикасно да се користе за смањење допустивог скупа на следећи начин: дефинише се праг у односу на густину операција, па ако је сума густина операција два опозита изнад те границе, ови опозити се додељују различитим ћелијама које не могу бити спојене пре тоталне претраге.

Став 2 може да се искористи да се предложени поступак ASSIGMENT прошири на прерасподелу машина и делова. У ту сврху уводимо дефиницију **ћелијских потомака**.

Дефиниција 10. Ћелијиски потомак је део или машина који задовољавају $x_{i,k} = o_{i,k} - v_{i,k} - e_{i,k} > 0$. Вредност $x_{i,k}$ је **тежина ћелијског потомка**. Ћелија c_k је **доминатор** за своје потомке.

Процедура SUCCESSOR додељује потомке својим доминаторима.
SUCCESSOR:

1. из скупа нераспоређених машина, одреди m_i са највећим $x_{i,k}$. Уколико више од једне машине има највеће $x_{i,k}$, изабери ону са највећим $o_{i,k}$. Распореди m_i у c_k;
2. **користећи** ASSIGNMENT прерасподели делове у c_k;
3. **иди у** 1.

8.2.6 *Јединичне ћелије, резидуали и екстреми*

Јединичне ћелије (*singletons*) су ћелије које имају мање од два дела или мање од две машине (Wu, Chang, & Yeh, 2009). Ова дефиниција укључује: 1. ћелије са једним делом или једном машином и 2. ћелије у којима су само машине или само делови. Случај 2 се у одређеним радовима (Li, Baki, & Aneja, 2010) означава као **резидуал** (*residual*). Неки радови не дозвољавају постојање јединичних ћелија и резидуала. Иако ова ограничења сужавају допустив скуп, ово није оправдано, осим у неким специфичним захтевима. Циљ CFP је да побољша процес производње и смањи трошкове, те је неоправдано да се машине или делови придружују ћелијама уколико

то придруживање смањује вредност циљне функције. На пример, оптимална партиција са два изузетка и четири празнине за инстанцу *No.4* (Kusiak & Cho, 1992) има вредност циљне функције 76.92% (Табела 8-5). Лако може да се уочи да p_2 учествује у вредности циљне функције са једним изузетком, две празнине и једном интерном операцијом.

Табела 8-5. Груписања за инстанцу *No.4*

	4	7	1	2	3	5	6	8
1	1	1		1				
4	1	1						
6	1	1						
2		1	1	1	1	1	1	1
3			■	■	1	■	1	1
5			1	■	1	1	1	1

Јасно је да би уклањање p_2 из ћелија повећало вредност циљне функције. Нова партиција има три изузетка и две празнине, резидуал са једним делом, вредност циљне функције 79.17% и приказана је у Табели 8-6.

Табела 8-6. Инстанца *No.4* са резидуалом

	4	7	1	3	5	6	8	2
1	1	1						1
4	1	1						
6	1	1						
2		1	1	1	1	1	1	1
3			■	1	■	1	1	
5			1	1	1	1	1	

За јединичне ћелије може једноставно да се квантификује учешће у вредности циљне функције на основу чињенице да одређени параметри оваквих ћелија не зависе од композиције осталих блокова у партицији.

Став 3. Уколико је c_k јединична ћелија са машином m_i, тада је $n_k^M = 1$, $d_k^C = n_k^P$, $v_k = 0$ и удео, Δe те ћелије у укупном e је $\Delta e = d_i^M - n_k^P + \sum_{p_j \in c_k} (d_j^P - 1)$.

Доказ: Све претходне релације произилазе из чињенице да јединичне ћелије немају празнине. Да би се то доказало, претпоставља се да је $p_j \in c_k \mid a_{i,j} = 0$. Овај део, p_j

учествује у укупној ефикасности груписања са једном празнином и d_j^P изузетака. Уколико се овај део уклони из c_k и прогласи за резидуал, он учествује у укупној ефикасности груписања само са d_j^P изузетака. На основу става 2, овај други случај има бољу ефикасност груписања, што комплетира доказ овог става.

Насупрот јединичним ћелијама и резидуалима, **екстреми** (*outliers*) су машине или делови са скоро свим одговарајућим јединицама у матрици *A*. Јасно је да је, у екстремним случајевима, када машина обрађује све делове, или је део обрађиван на свим машинама, информациона вредност оваквог екстрема у CF поступку једнака нули. Оваква машина или део се уклањају из CF поступка. После комплетирања поступка, екстрем се придружује најповољнијој ћелији. Треба да се напомене да је, за разлику од случаја са јединичним ћелијама и резидуалима, неопходно да се провере могућности спајања одговарајућих ћелија или мултиплицирања екстремних машина. У експерименталној евалуацији у дисертацији је усвојено да су екстреми машине или делови чија је густина операција већа од 0.85. Овај праг је на свим инстанцама давао исправан резултат.

Претходна разматрања сугеришу критеријуме за избор потенцијалних јединичних ћелија и резидуала. Идеја је да се они уклоне из обраде, чиме се смањује величина проблема, и да се они по завршетку рада алгоритма придруже одговарајућој ћелији или оставе изван ћелија као резидуали. Насупрот груписањима предложеним у претходним одељцима, избор потенцијалних јединичних ћелија, резидуала и екстрема на почетку обраде није критично због њихове мале информационе вредности у поступку формирања ћелија.

На основу вредности $s_{i,j}, d_i^M, \rho_i^M, d_j^P$ и ρ_j^P, може да се постави критеријум за одређивање потенцијалних резидуала. Мање вредности од $\sum_{j=1}^{r} s_{i,j}$, d_i^M и ρ_i^M имплицирају већу вероватноћу да је m_i резидуал (исто важи за делове). Поред значаја за смањење допустивог скупа, уклањање свих потенцијалних резидуала је важно из следећег разлога. За разлику од јединичних ћелија, резидуали не постоје у скупу свих могућих партиција, дефинисаних у потпоглављу 3.1. Алгоритам за генерисање партиција не пружа могућност да елемент не припада ни једном блоку.

Наравно ово може да се измени, али не постоји потреба за тим јер је присуство резидуала корисно за редукцију допустивог скупа. Према томе, најједноставнији начин је да се декларише предефинисани проценат машина (односно делова) са минималним $\sum_{j=1}^{r} s_{i,j}$, d_i^M и ρ_i^M за потенцијалне резидуале.

Став 3 дефинише критеријум за избор јединичних ћелија. Уколико је величина тренутног допустивог скупа критична, потенцијалне јединичне ћелије се уклањају из разматрања пре тоталне претраге, а ако није, процедура за генерисање партиција ће их конструисати на исти начин као и остале блокове.

8.3. Нови алгоритам

На основу претходних разматрања предложен је нови хибридни алгоритам за CFP. Као хеуристика, у овом алгоритму се користи GCA, означена као GCACFP којом се побољшава вредност циљне функције променом почетне партиције датог скупа машина. *ARG* садржи само један аргумент, $\boldsymbol{arg_4} = R_1$ који одређује да се у равноправним ситуацијама прате све партиције које резултују у тренутно оптималној вредности циљне функције. Ова дефиниција омогућује једноставну имплементацију паралелног програмирања, пошто се индивидуе у почетној популацији могу да обрађују независно једна од друге. Прецизна дефиниција GCACFP-а захтева само дефиниције функција f_1, f_2, f_3, које су у случају CFP веома једноставне:

GCACFP

---

f_1: $\pi \leftarrow f_1(PR^n)$ – уређује блокове у датој партицији према опадајућој вредности њихове кардиналности.

---

f_2: $b_{k+1} \leftarrow f_2(k, B^{n-k})$ – бира последњи блок b_{k+1} из B^{n-k}.

$f_3 : \pi, B^*, B^{n-k} \leftarrow f_3(\pi, k, b_{k+1})$:

1. $\Pi \leftarrow ASSIGNMENT(\pi)$; $b_{opt} \leftarrow null$; $\eta_{opt} \leftarrow null$
2. **for each** block b in $\pi \neq b_{k+1}$
   i. **if not** b = **op**(b_{k+1}) **then** $\pi^* \leftarrow concenate(b, b_{k+1})$
   ii. $\Pi^* \leftarrow ASSIGNMENT^*(\pi^*)$

iii. **if** ($\eta(\Pi^*) > \eta_{opt}$) **then** $b_{opt} \leftarrow b$; $\eta_{opt} \leftarrow \eta(\Pi^*)$

3. if not $b_{opt} = null$ **then** $\pi \leftarrow concenate(b_{opt}, b_{k+1})$ **else** $B^* \xleftarrow{last\ element} b_{k+1}$

4. $B^{n-k} = B^{n-k} \setminus b_{k+1}$

Израз b = **op**(b_{k+1}) означава да је неки елемент блока b_1 опозит неком елементу из b_2. Звездица у ASSIGNMENT* означава да се узимају у обзир само делови који се обрађују на машинама у b и у b_{k+1}. Функција $concenate(b, b_{k+1})$ спаја блокове b и b_{k+1}. Функција f_3 итеративно бира најбољи блок коме ће се придружити тренутно одабрани део или други блок.

Нови хибридни алгоритам, CFOPT може да се представи низом корака, који се извршавају један за другим.

CFOPT:

1. примени EXSTRACT на машине и делове;
2. израчунај глобалне параметре за редуковани проблем, па на основу њих одреди ограничења према (13) и (14);
3. на основу оригиналних коефицијената сличности, уклони резидуале и екстреме машина и делова;
4. одреди забрањена груписања на основу опозита;
5. примени GROUPING на машине;
6. примени SUCCESSOR за машине;
7. примени GCACFP;
8. претражи најближе боље вредности циљне функције на основу разматрања у одељку 8.2.2;
9. у зависности од величине допустивог скупа, уклони јединичне ћелије;
10. уколико редуковани допустиви скуп то дозвољава, примени тоталну претрагу, иначе, примени неку хеуристику побољшања;
11. придружи јединичне ћелије оптималним блоковима у посматраној партицији;
12. нађи најбоље придруживање екстрема тренутној партицији;
13. нађи најбоље придруживање резидуала тренутној партицији или их остави као резидуале.

Сви кораци овог алгоритма су већ детаљно објашњени, изузимајући кораке 11-13, који подразумевају тривијалну претрагу свих могућих комбинација.

Функционисање алгоритма може да се објасни на примеру инстанце *No.4*, приказане у одељку 8.2.6 (Табела 8-5 и Табела 8-6). Инстанца има 6 машина и 8 делова, а укупна густина операција је 45.83. Матрице сличности и густине операција за машине и делове су приказане, редом, у Табели 8-7 и Табели 8-8.

Табела 8-7. Матрица сличности и густине опарација за машине

	1	2	3	4	5	6	Густина
1		0.25	0	0.67	0	0.67	0.375
2			0.43	0.13	0.71	0.13	0.875
3				0	0.6	0	0.375
4					0	1	0.25
5						0	0.625
6							0.25

Табела 8-8. Матрица сличности и густине опарација за делове

	1	2	3	4	5	6	7	8	Густина
1		0.33	0.67	0	1	0.67	0.2	0.67	0.33
2			0.25	0.25	0.33	0.25	0.5	0.25	0.33
3				0	0.67	1	0.17	1	0.5
4					0	0	0.75	0	0.5
5						0.67	0.2	0.67	0.33
6							0.17	1	0.5
7								0.17	0.67
8									0.5

У кораку 1 алгоритма CFOPT, процедура EXTRACT редукује величину проблема на 5x5, пошто машина 6 и делови 5, 6 и 8 могу да се избаце из даљег разматрања. Иако ова димензија проблема може, скоро тренутно, да се обради тоталном претрагом, биће показан утицај осталих параметара. Машина 2 има густину операција 0.875 и представља несумњиви екстрем. Део 2 је потенцијални резидуал пошто нема ни један коефицијент сличности изнад прага, а при томе има најмању густину операција. Већ на основу ових додатних информација, оптимално решење може да се добије и без употребе рачунара. Независно од овога, оптимално решење може да се добије и на основу података о опозитима. Међу машинама, опозити су (1, 3), (1, 5), (3, 4), (3, 6), (4, 5) и (5, 6) са одговарајућим паровима густина операција (0.375,

0.375), (0.375, 0.625), (0.375, 0.25), (0.375, 0.25), (0.25, 0.625) и (0.625, 0.25). Ово имплицира једнозначно груписање у (1, 4, 6), (3, 5).

8.4. Проширење проблема

За дизајн ћелијског производног система је, у одређеним ситуацијама, важно уређење ентитета унутар ћелија. Ово уређење добија на значају и у ситуацијама када није могуће обавити квалитетно формирање ћелија, као што је објашњено у одељку 8.2.3. Речено је да су то случајеви када је вредност циљне функције оптималног CF испод прага квалитета који је дефинисао менаџер. У том случају, уместо да се врши груписање у ћелије, потребно је да се машине, односно делови, међусобно уреде тако да суседне машине обрађују што је могуће више истих делова, односно да се суседни делови обрађују на што је могуће више истих машина. У том случају је кључно да се обезбеди циљна функција која је прилагођена овим захтевима и при томе је подесна за оптимизацију. У раду (Ilić, 2014) и (Ilić & Cvetić, 2014) се предлаже максимизација $\sum_{i=1}^{q-1} s^{M}_{\pi_i,\pi_{i+1}}$ и $\sum_{i=1}^{r-1} s^{P}_{\pi_i,\pi_{i+1}}$, где π означава тренутну пермутацију машина или делова, а $\pi(i)$ означава машину или део који је на *i*-тој позицији у π. Предложена циљна функција узима у обзир само суседне елементе у пермутацији машина, односно делова. Овако дефинисана циљна функција отвара велике могућности за избор оптимизационог поступка. Ова оптимизација може у потпуности да се спроведе коришћењем претходно описаних процедура, уз једину измену да се уместо скупова партиција користе скупови пермутација (Danilovic and Ilić, 2016).

Оптимизациони поступак за машине одређује пермутацију $(\pi_1,...,\pi_m)$ ознака машина (1,..,*m*), такву да максимизира вредност $\sum_{i=1}^{m-1} s^{M}_{\pi_i,\pi_{i+1}}$, где је $s^{M}_{i,j}$ коефицијент сличности између машине *i* и машине *j*. Оптимизациони поступак за делове одређује пермутацију $(\Pi_1,...,\Pi_n)$ ознака делова (1,..,*n*), такву да максимизира вредност $\sum_{k=1}^{n-1} s^{P}_{\Pi_k,\Pi_{k+1}}$, где је $s^{P}_{k,l}$ коефицијент сличности између дела *k* и дела *l*.

Циљ оптимизације може да се појасни на примеру који је обрађен у (Ilić, 2014) и (Ilić & Cvetić, 2014). Резултати добијени у овим радовима су поређени са резултатима из (Irani & Huang, 2005), (Irani, PFAST, 2012) и (Irani, Zhang, Zhou, Huang, Udai, & Subramanian, 2000), добијеним применом програмског пакета PFAST. Табела 8-9 приказује процесне путање за 19 делова на 12 машина.

Табела 8-9. Процесне путање

1	1	4	8	9			
2	1	4	7	4	8	7	
3	1	2	4	7	8	9	
4	1	4	7	9			
5	1	6	10	7	9		
6	6	10	7	8	9		
7	6	4	8	9			
8	3	5	2	6	4	8	9
9	3	5	6	4	8	9	
10	4	7	4	8			
11	6						
12	11	7	12				
13	11	12					
14	11	7	10				
15	1	7	11	10	11	12	
16	1	7	11	10	11	12	
17	11	7	12				
18	6	7	10				
19	12						

На основу ових улазних података су конструисане матрице сличности за машине и делове, приказане у Табелама 8-10 и 8-11. У овим табелама су црвеном бојом означени дијагонални елементи матрица. Сума ових елемената представља вредност циљне функције. Јасно је да је циљ оптимизације да преуреди елементе тако да по дијагонали буду елементи са највећом вредношћу коефицијента сличности. У табелама су жутом бојом означени елементи матрице који су изнад постављеног прага, који је на овим сликама 0.7 и за машине и за делове. Вредност циљних функција за овакав распоред машина и делова је 3.496 за машине, и 8.632 за делове.

Табела 8-10. Матрица сличности за машине

	1	2	3	4	5	6	7	8	9	10	11	12
01		0.125	0	0.364	0	0.077	0.462	0.25	0.364	0.3	0.182	0.182
02			0.333	0.25	0.333	0.125	0.077	0.25	0.25	0	0	0
03				0.25	1	0.286	0	0.25	0.25	0	0	0
04					0.25	0.25	0.25	0.778	0.6	0	0	0
05						0.286	0	0.25	0.25	0	0	0
06							0.188	0.364	0.5	0.3	0	0
07								0.25	0.25	0.5	0.385	0.286
08									0.6	0.077	0	0
▶ 09									0	0.167	0	0
10											0.333	0.2
11												0.714
12												

Табела 8-11. Матрица сличности за делове

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
▶ 01	0																		
02	0.6																		
03	0.667	0.667																	
04	0.6	0.6	0.667																
05	0.286	0.286	0.375	0.5															
06	0.286	0.286	0.375	0.286	0.667														
07	0.6	0.333	0.429	0.333	0.286	0.5													
08	0.375	0.222	0.444	0.222	0.2	0.333	0.571												
09	0.429	0.25	0.333	0.25	0.222	0.375	0.667	0.857											
10	0.4	0.75	0.5	0.4	0.143	0.333	0.4	0.25	0.286										
11	0	0	0	0	0.2	0.2	0.25	0.143	0.167	0									
12	0	0.167	0.125	0.167	0.143	0.143	0	0	0	0.2	0								
13	0	0	0	0	0	0	0	0	0	0	0	0.667							
14	0	0.167	0.125	0.167	0.333	0.333	0	0	0	0.2	0	0.5	0.25						
15	0.125	0.286	0.222	0.286	0.429	0.25	0	0	0	0.143	0	0.6	0.4	0.6					
16	0.125	0.286	0.222	0.286	0.429	0.25	0	0	0	0.143	0	0.6	0.4	0.6	1				
17	0	0.167	0.125	0.167	0.143	0.143	0	0	0	0.2	0	1	0.667	0.5	0.6	0.6			
18	0	0.167	0.125	0.167	0.6	0.6	0.167	0.111	0.125	0.2	0.333	0.2	0	0.5	0.333	0.333	0.2		
19	0	0	0	0	0	0	0	0	0	0	0	0.333	0.5	0	0.2	0.2	0.333	0	

Прва фаза GCA је дефинисање почетног уређења објеката, у овом случају машина, односно делова. Како постоји директна аналогија између оптимизације уређења машина и оптимизације уређења делова, у наставку рада је описан само оптимизациони поступак за уређење машина. Дефиниција циљне функције омогућава сужење допустивог скупа решења на основу следећих Лема.

Лема 1: Уколико је $s_{i,j}^{M}=1$, машина *i* се у оптималној пермутацији машина налази поред машине *j*.

Лема 2: Уколико је $s_{ij}^{M}=1$ и $s_{j,k}^{M}=1$, тада важи $s_{i,k}^{M}=1$.

Лема 3: Уколико више машина има међусобни коефицијент сличности једнак јединици, свака пермутација ових машина унутар укупне пермутације машина резултира у истој вредности циљне функције.

Доказ ових лема може да се изведе на следећи начин: Вредност циљне функције се добија сумирањем само коефицијената сличности суседних машина у пермутацији. Претпоставимо, без губитка општости, да само машине *i* и *j* имају коефицијент сличности један, односно да су сви остали коефицијенти мањи од један. Нека је у првом случају оптимални парцијални редослед машина *k i j l*, а у другом *k i q j l*. Коефицијент сличности између машина *i* и *q* је мањи или једнак од коефицијента сличности између машина *i* и *k*, иначе *k i j l* не би била оптимална пермутација. На исти начин је и коефицијент сличности између машина *j* и *q* мањи или једнак од коефицијента сличности између машина *j* и *l*. Према томе, сабирак, чија је вредност јединица се у другом случају замењује сабирком мањим од један и додаје се нови сабирак који је у најбољем случају једнак сабирку у првом случају. Овим је прва Лема доказана. Доказ за друге две Леме је тривијалан.

Закључци ових Лема могу да се прошире и на случајеве када коефицијенти сличности нису једнаки јединици:

Лема 4: Уколико више машина има међусобни коефицијент сличности већи од *x*, и уколико су сви остали коефицијенти сличности мањи од *x*, у оптималној пермутацији ће све ове машине да представљају компактну групу. Међусобни распоред машина унутар ове групе ће одговарати оној пермутацији ове групе која има највећу парцијалну вредност циљне функције.

Доказ ове Леме је аналоган претходном доказу. Јасно је да добијени закључци могу да дефинишу поступке којима се значајно редукује допустиви скуп решења. Број од *m*! могућих пермутација се смањује на (*w*-*v*+1)! + (*m*-*w*)!, где је *w* кардиналност

скупа машина са коефицијентом сличности већим од осталих, а v број машина чији је међусобни коефицијент сличности једнак јединици. Ово смањење допустивог скупа може да буде толико значајно да за одређене инстанце омогући примену егзактног алгоритма.

Следећи ову идеју, закључци могу да се генерализују и на примене када је ипак, због величине редукованог проблема, неопходно да се примени нека од хеуристика. Усвојена циљна функција и унапред дефинисани ниво кластеровања представљају идеалне предуслове да се дефинише почетно уређење које обезбеђује изузетно ефикасан GCA:

Корак 1. Свака од група машина у којој су сви коефицијенти сличности једнаки јединици се замени са само једном, произвољно одабраном машином из те групе.

Корак 2. Дефинише се параметар $p \leq 1$.

Корак 3. Примени се кластер алгоритам којим се групишу машине у кластере, тако да је најмањи коефицијент сличности у сваком од кластера већи или једнак са p. Означимо најмањи коефицијент сличности у q-том кластеру са y_q (уколико машина не припада ни једном кластеру, одговарајућа вредност јој је једнака највећем од њених коефицијената сличности са другим машинама).

Корак 4. Кластери се међусобно уреде према нерастућим вредностима y_q.

Корак 5. У сваком кластеру се одреде пермутације машина из кластера које дају највећу вредност циљне функције.

Описани кораци представљају теоријску основу над којом се дефинише почетно уређење, односно фаза иницијализације GCA. Свака комбинација добијених оптималних парцијалних пермутација корацима претходног поступка одређује једно почетно уређење на које се затим примењује GCA. Фаза уметања се примењује тако да се свака од кластерованих група третира као један елемент у пермутацији. Када се у κ-тој итерацији одреди пермутација дужине κ, на свакој позицији ове пермутације која одговара неком кластеру се препишу елементи тог кластера по редоследу из почетног уређења.

Дефиниција почетног уређења π_P^n је довољна за прецизну дефиницију комплетног алгоритма. Наиме, у фази уметања се бира први слободни елемент из π_P^n и умеће на најбољу позицију у тренутну секвенцу. Пошто алгоритмом могу да се симултано прате више почетних пермутација ово омогућава тривијалну примену паралелног програмирања, јер се јединке почетне популације обрађују потпуно независно једна од друге.

И пре експерименталне верификације предложеног приступа је јасно које предности он ставља на располагање корисницима. Иако предложени поступак не гарантује оптималну вредност циљне функције он омогућује ванредно ефикасну обраду уз битну чињеницу: број корака алгоритма је унапред дефинисан са само две променљиве, *m* и *p*, те корисник прецизно добија информацију колика ће бити потрошња CPU времена. На тај начин корисник може да прилагоди прецизност претраге расположивом времену рачунара, тј. да бира што је могуће већу вредност за *p*. За постављени праг 0.7 постоје три кластера за машине и четири кластера за делове. Кластери за машине су {3,5},{4,8} и {11,12}, док су за делове {15,16},{12,17},{8,9} и {2,10}. Ови кластери смањују величину допустивог скупа решења, и то, за машине са 12! на 2!2!2!9! и за делове са 19! на 2!2!2!2!15!. Јасно је да за машине може да се примени тотална претрага, док за делове треба да се примени GCA. Применом тоталне претраге за машине се добијају четири оптимална редоследа за које је вредност циљне функције 5.87. Ови редоследи су приказани у Табели 8-12.

Табела 8-12. Оптимални редоследи машина

	1	2	3	4	5	6	7	8	9	10	11	12
▶ 1	12	11	10	7	1	4	8	9	6	3	5	2
2	2	3	5	6	9	8	4	1	7	10	11	12
3	12	11	10	7	1	4	8	9	6	5	3	2
4	2	5	3	6	9	8	4	1	7	10	11	12

Применом GCA за делове се добија 16 оптималних редоследа за које је вредност циљне функције 11.418. Ови редоследи су приказани у Табели 8-13.

Табела 8-13. Оптимални редоследи делова

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
▶ 01	11	8	9	7	1	3	4	2	10	6	5	18	14	15	16	12	17	13	19
02	11	8	9	7	1	4	3	2	10	6	5	18	14	15	16	12	17	13	19
03	11	8	9	7	1	3	4	2	10	6	5	18	14	16	15	12	17	13	19
04	11	8	9	7	1	4	3	2	10	6	5	18	14	16	15	12	17	13	19
05	19	13	12	17	15	16	14	18	5	6	10	2	3	4	1	7	9	8	11
06	19	13	12	17	15	16	14	18	5	6	10	2	4	3	1	7	9	8	11
07	19	13	12	17	16	15	14	18	5	6	10	2	3	4	1	7	9	8	11
08	19	13	12	17	16	15	14	18	5	6	10	2	4	3	1	7	9	8	11
09	19	13	17	12	15	16	14	18	5	6	10	2	3	4	1	7	9	8	11
10	19	13	17	12	15	16	14	18	5	6	10	2	4	3	1	7	9	8	11
11	19	13	17	12	16	15	14	18	5	6	10	2	3	4	1	7	9	8	11
12	19	13	17	12	16	15	14	18	5	6	10	2	4	3	1	7	9	8	11
13	11	8	9	7	1	3	4	2	10	6	5	18	14	15	16	17	12	13	19
14	11	8	9	7	1	4	3	2	10	6	5	18	14	15	16	17	12	13	19
15	11	8	9	7	1	3	4	2	10	6	5	18	14	16	15	17	12	13	19
16	11	8	9	7	1	4	3	2	10	6	5	18	14	16	15	17	12	13	19

Може да се уочи да постоје парови симетричних решења (за машине, решења (1,3) су симетрична са решењима (2,4), док је за делове ова симетрија (1,2,3,4,13,14,15,16) према (12,11,10,9,8,7,6,5)).

За прве распореде машина и делова је формирана 0-1 матрица делова-машина и приказана у Табели 8-14.

На основу овако преуређених низова машина и делова и преуређене 0-1 матрице се формирају производне ћелије. Уколико се одреди нижи праг прецизности претраге, време рада се смањује уз могућност лошијих вредности циљних функција. Како је за машине из посматраног примера и праг од 0.7 довео до могућности примене тоталне претраге, експеримент је спроведен у поступку за оптимизацију редоследа делова. Усвојен је праг 0.66. Кластеровањем делова уз праг одсецања изнад 0.66 добијају се 5 кластера, и то: {14,15}, {16,17,18}, {2,3,4}, {5,6,7,8,9} и {10,11}. Ово груписање смањује величину допустивог скупа решења са 19! на 2!3!3!5!2!9!. Занимљиво је да и поред значајног смањења допустивог скупа решења, односно драстичне уштеде времена рада рачунара, GCA даје оптималне резултате идентичне претходном експерименту када је праг био 0.7. Табеле 8-15 и 8-16 приказују преуређене матрице

сличности машина, односно делова на основу оптималних резултата добијених применом GCA.

Табела 8-14. Матрица делова-машина

	12	11	10	7	1	4	8	9	6	3	5	2
▶ 11	0	0	0	0	0	0	0	0	1	0	0	0
08	0	0	0	0	0	1	1	1	1	1	1	1
09	0	0	0	0	0	1	1	1	1	1	1	0
07	0	0	0	0	0	1	1	1	1	0	0	0
01	0	0	0	0	1	1	1	1	0	0	0	0
03	0	0	0	1	1	1	1	1	0	0	0	1
04	0	0	0	1	1	1	0	1	0	0	0	0
02	0	0	0	1	1	1	1	0	0	0	0	0
10	0	0	0	1	0	1	1	0	0	0	0	0
06	0	0	1	1	0	0	1	1	1	0	0	0
05	0	0	1	1	1	0	0	1	1	0	0	0
18	0	0	1	1	0	0	0	0	1	0	0	0
14	0	1	1	1	0	0	0	0	0	0	0	0
15	1	1	1	1	1	0	0	0	0	0	0	0
16	1	1	1	1	1	0	0	0	0	0	0	0
12	1	1	0	1	0	0	0	0	0	0	0	0
17	1	1	0	1	0	0	0	0	0	0	0	0
13	1	1	0	0	0	0	0	0	0	0	0	0
19	1	0	0	0	0	0	0	0	0	0	0	0

Табела 8-15. Преуређена матрица сличности машина

	1	2	3	4	5	6	7	8	9	10	11	12
▶ 01	0	0.714	0.2	0.286	0.182	0	0	0	0	0	0	0
02			0.333	0.385	0.182	0	0	0	0	0	0	0
03				0.5	0.3	0	0.077	0.167	0.3	0	0	0
04					0.462	0.25	0.25	0.25	0.188	0	0	0.077
05						0.364	0.25	0.364	0.077	0	0	0.125
06							0.778	0.6	0.25	0.25	0.25	0.25
07								0.6	0.364	0.25	0.25	0.25
08									0.5	0.25	0.25	0.25
09										0.286	0.286	0.125
10											1	0.333
11												0.333
12												

Уочава се да су применом GCA све вредности изнад прага (обојене жутом бојом) померене на дијагоналу или у непосредну околину дијагонале.

Табела 8-16. Преуређена матрица сличности делова

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
01	0																		
02	0.143																		
03	0.167	0.857																	
04	0.25	0.571	0.667																
05	0	0.375	0.429	0.6															
06	0	0.444	0.333	0.429	0.667														
07	0	0.222	0.25	0.333	0.6	0.667													
08	0	0.222	0.25	0.333	0.6	0.667	0.6												
09	0	0.25	0.286	0.4	0.4	0.5	0.4	0.75											
10	0.2	0.333	0.375	0.5	0.286	0.375	0.286	0.286	0.333										
11	0.2	0.2	0.222	0.286	0.286	0.375	0.5	0.286	0.143	0.667									
12	0.333	0.111	0.125	0.167	0	0.125	0.167	0.167	0.2	0.6	0.6								
13	0	0	0	0	0	0.125	0.167	0.167	0.2	0.333	0.333	0.5							
14	0	0	0	0	0.125	0.222	0.286	0.286	0.143	0.25	0.429	0.333	0.6						
15	0	0	0	0	0.125	0.222	0.286	0.286	0.143	0.25	0.429	0.333	0.6	1					
16	0	0	0	0	0	0.125	0.167	0.167	0.2	0.143	0.143	0.2	0.5	0.6	0.6				
17	0	0	0	0	0	0.125	0.167	0.167	0.2	0.143	0.143	0.2	0.5	0.6	0.6	1			
18	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0.4	0.4	0.667	0.667		
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0.2	0.333	0.333	0.5	

8.5. Експериментални резултати

Ради евалуације предложеног алгоритма, извршена је експериментална провера на 35 инстанци, добијених са адресе http://mauricio.resende.info/data/cell-formation/. Треба да се нагласи да су сви релевантни алгоритми за CFP тестирани на овим инстанцама. Захваљујући периоду од три деценије у коме су ове инстанце тестиране, најбољи познати резултати су истовремено и оптимални резултати за већину ових инстанци. Величина инстанце се означава бројем машина и бројем делова. Опсег димензија инстанци је од 5x7 до 40x100 и инстанце садрже како неструктуиране, тако и добро-структуиране матрице. Величине инстанци и изворни радови где су објављене се приказују у Табели 8-17.

Табела 8-17. Референтне инстанце из литературе

Бр.	*q*	*r*	Референца
1	5	7	(King & Nakornchai, 1982)
1a	5	7	(King & Nakornchai, 1982)*
2	5	7	(Waghodekar & Sahu, 1984)
3	5	18	(Seifoddini H. , 1989)
4	6	8	(Kusiak & Cho, 1992)
5	7	11	(Kusiak & Chow, 1987)
6	7	11	(Boctor, 1991)
7	8	12	(Seifoddini & Wolfe, 1986)
8	8	20	(Chandrasekharan & Rajagopalan, 1986)
9	8	20	(Chandrasekharan & Rajagopalan, 1986a)
10	10	10	(Mosier & Taube, 1985)
11	10	15	(Chan & Milner, 1982)
12	14	23	(Askin & Subramanian, 1987)
13	14	24	(Stanfel, 1985)
14	16	23	(McCormick, Schweitzer, & White, 1972)
15	16	30	(Srinivasan, Narendran, & Mahadevan,
16	16	43	(King J. , 1980)
17	18	24	(Carrie, 1973)
18	20	20	(Mosier & Taube, 1985a)
19	20	23	(Kumar, Kusiak, & Vannelli, 1986)
20	20	35	(Carrie, 1973)
21	20	35	(Boe & Cheng, 1991)
22	24	40	(Chandrasekharan & Rajagopalan, 1989)
23	24	40	(Chandrasekharan & Rajagopalan, 1989)
24	24	40	(Chandrasekharan & Rajagopalan, 1989)
25	24	40	(Chandrasekharan & Rajagopalan, 1989)
26	24	40	(Chandrasekharan & Rajagopalan, 1989)
27	23	40	(Chandrasekharan & Rajagopalan, 1989)
28	27	27	(McCormick, Schweitzer, & White, 1972)
29	28	46	(Carrie, 1973)
30	30	41	(Kumar & Vannelli, 1987)
31	30	50	(Stanfel, 1985)
32	30	50	(Stanfel, 1985)
33	30	90	(King & Nakornchai, 1982)
34	37	53	(McCormick, Schweitzer, & White, 1972)
35	40	100	(Chandrasekharan & Rajagopalan, 1987)

Нови алгоритам је програмиран у језику C# и примењен на лаптоп рачунару са процесором Intel Core 2 Duo CPU T6600, 2.2GHz, и 6 GB инсталисане меморије, на оперативном систему Microsofts Windows 7. Резултати, добијени за сваку од инстанци су поређени са најбољим резултатима за овај проблем. Ови резултати су добијени применом 19 поступака приказаних у Табели 8-18.

Табела 8-18. Најбољи поступци за CFP

Бр.	Метода	Алгоритам	Референца
1	ZODIAC	кластер	(Chandrasekharan & Rajagopalan, 1987)
2	GRAFICS	кластер	(Srinivasan & Narendran, 1991)
3	GATSP	генетски	(Cheng, Gupta, Lee, & Wong, 1998)
4	GA	генетски	(Onwubolu & Mutingi, 2001)
5	EA	еволуциони	(Goncalves & Resende, 2004)
6	ES	еволуциона стратегија	(Stawowy, 2006)
7	HGGA	генетски алгоритам груписања	(James, Brown, & Keeling, 2007)
8	SACF	симулирано каљење	(Wu, Chang, & Chung, 2008)
9	HGA	хибридни генетски	(Tariq, Hussain, & Ghafoor, 2009)
10	GAA	генетски	(Mahdavi, Paydar, Solimanpur, & Heidarzade, 2009)
11	HHA	хибридна хеуристика	(Wu, Chang, & Yeh, 2009)
12	EnGGA	проширени генетски алгоритам груписања	(Tunnukij & Hicks, 2009)
13	WFA	алгоритам протока воде	(Wu, Chung, & Chang, 2010)
14	HGDE	дифернцијални еволуциони	(Noktehdan, Karimi, & Husseinzadeh Kashan, 2010)
15	ACO	оптимизација мрављих колонија	(Li, Baki, & Aneja, 2010)
16	SA	симулирано каљење	(Pailla, Trindade, Parada, & Ochi, 2010)
17	SAYLL	симулирано каљење са претрагом променљивих околина	(Ying, Lin, & Lu, 2011)
18	GRASP	GRASP хеуристика	(Dıaz, Luna, & Luna, 2012)
19	GAVNS	генетски алгоритам са претрагом променљивих околина	(Paydar & Saidi-Mehrabad, 2013)

Табела 8-19 садржи поређење времена рада процесора између егзактног алгоритма, CPLEX (Elbenani & Ferland, 2012), алгоритма GAVNS и новог алгоритма, CFOPT, за 35 референтних инстанци. Логични избор хеуристика за поређење временске ефикасности је да се временска ефикасност нове хеуристике пореди са ефикасношћу процедуре која даје најбоље објављене резултате. Према томе, Табела 8-19 приказује поређење времена рада процесора за GAVNS и CFOPT. Треба да се напомене да је HHA једина процедура која је ефикаснија од GAVNS-a, али уз специфичну комбинацију критеријума заустављања, који, када се примене, дају значајно лошије резултате. Подаци за егзактан алгоритам су приложени као референце. За веће проблеме, (16, 18, 21, и 25 - 34), је временско ограничење од 86,400 секунди достигнуто пре добијања решења. Овакве ситуације су означене звездицом у Табели 8-19. За остале веће проблеме (14, 19, и 35), оптимално решење је добијено, али време рада премашује 86,400 секунди (такође означено звездицом у Табели 8-19). За

инстанце (3, 4, 12 – 16, 18, 20, 21, и 25 - 34) је добијена вредност ефикасности груписања испод оптималне вредности, што се објашњава тиме да егзактни алгоритам не продукује резидуале. Такође, за инстанце (12, 14 – 16, 18, 20, 21, 25 – 30 и 32 – 34) просечна ефикасност добијена методом GAVNS је испод најбољих вредности.

Табела 8-19. Поређење CPLEX, GAVNS и CFOPT поступака

Бр.	*q*	*r*	CPLEX		GAVNS		CFOPT	
			Max.	CPU	Av.	CPU	Вредност	CPU
1	5	7	82.35	0.33	82.35	0.21	82.35	0
2	5	7	69.57	0.35	69.57	0.16	69.57	0
3	5	18	79.59	0.47	80.85	0.11	80.85	0
4	6	8	76.92	0.37	79.17	0.14	79.17	0
5	7	11	60.87	1.67	60.87	0.89	60.87	0
6	7	11	70.83	1.17	70.83	0.74	70.83	0
7	8	12	69.44	1.69	69.44	0.85	69.44	0
8	8	20	85.25	2	85.25	0.93	85.25	0
9	8	20	58.72	5.03	58.72	0.96	58.72	0
10	10	10	75	1.82	75	1.24	75	0
11	10	15	92	1.72	92	1.54	92	0
12	14	24	72.06	78.11	71.83	5.65	74.24	0.07
13	14	24	71.83	103.68	72.86	8.76	72.86	0.09
14	16	24	53.26	*101,783.88	53.33	12.34	53.85	0.08
15	16	30	69.53	752.01	69.92	15.4	70.76	0.09
16	16	43	57.23	*	57.42	17.42	57.98	0.55
17	18	24	57.73	14,909.77	57.73	16.65	57.73	0.14
18	20	20	39.66	*	43.18	19.89	43.97	0.13
19	20	23	50.81	*869,103	50.81	24.23	50.81	0.12
20	20	35	77.91	144.63	77.78	31.75	79.38	0.43
21	20	35	55.49	*	57.61	43.57	58.79	0.22
22	24	40	100	108.58	100	56.43	100	0.16
23	24	40	85.11	496.95	85.11	71.69	85.11	0.43
24	24	40	73.51	6,556.99	73.51	82.5	73.51	0.21
25	24	40	47.95	*	53.27	74.19	53.29	0.66
26	24	40	39.39	*	46.66	53.29	48.95	0.87
27	24	40	41.84	*	47.16	83.65	47.26	0.75
28	27	27	50.57	*	54.31	98.81	54.82	0.02
29	28	46	35.68	*	46.06	95.47	47.68	0.9
30	30	41	59.7	*	63.04	118.21	63.31	1.1
31	30	50	49.69	*	60.12	151.72	60.12	1.74
32	30	50	41.42	*	50.68	167.23	50.83	1.93
33	36	90	43.77	*	46.03	276.22	47.93	3.56
34	37	53	57.47	*	60.57	245.68	61.16	0.22
35	40	100	84.03	*1,572,184.5	84.03	210.43	84.03	3.11

Табела 8-20 садржи резултате на 35 тест инстанци, добијених применом 19 одабраних метода и новим алгоритмом CFOPT. За две инстанце (14 и 15) групна ефикасност добијена алгоритмом CFOPT је изнад најбољих објављених вредности. Поређење резултата у Табели 8-19 и Табели 8-20 намеће једнозначан и једноставан закључак: CFOPT је на скупу тестираних инстанци квалитетнији и ефикаснији од

најбољих објављених резултата за CFP. Нови алгоритам је више него за ред величине ефикаснији од најбољих постојећих алгоритама. Ни за једну од 35 инстанци није било потребно да се примени нека хеуристика побољшања и при томе је за сваку инстанцу добијен резултат који је бољи или једнак најбољим објављеним резултатима.

Ово свакако није најзначајнија предност новог приступа. Најважнија предност је што је за проблем комбинаторне оптимизације предложен скуп поступака који користе специфичности самог проблема и сваки од њих, понаособ, директно усмерава правац претраживања ка глобалном оптимуму. Свакако, повољна страна евалуације овог приступа је што су тестиране инстанце, и по величини, и по саставу изузетно погодне за овај приступ. Како су ове инстанце конструисане током 30-тогодишњег објављивања радова за CFP, оне су једини расположиви скуп инстанци које могу да се употребе за објективну евалуацију хеуристика.

Изложени приступ предствља покушај да се проблем комбинаторне оптимизације реши на начин који се суштински разликује од приступа већине објављених радова у новије време. Свакодневно се предлажу нови алгоритми који поправљају познате резултате, како по ефикасности тако и по квалитету решења. Ови резултати се, по правилу, постижу коришћењем техника побољшања на скуп почетних решења. Међутим, ограничавање истраживања само на овакав приступ је нелогично када структура проблема омогућава да се специфичности искористе за редуковање величине проблема. Предложеним приступом се добија једна значајна предност: могуће је објаснити зашто нови алгоритам ради добро. Наиме, нови поступак представља разматрање суштине састава посматране инстанце и користи разноврсне алате који му стоје на располагању да проблем разложи на скуп мањих проблема. Ово даље упућује на могућност свођења проблема класе НП на проблем из класе псеудо НП. Наиме, дискретност скупа могућих вредности циљне функције омогућује да се у одређеном тренутку извршавања програма дефинише коначан скуп могућих вредности циљне функције и да се тада примени поступак чија је временска зависност функција од овог броја. На тај начин се укупна сложеност таквог алгоритма састоји од два сабирка. Први представља полиномијалну зависност у функцији величине проблема а други полиномијалну зависност у

функцији броја могућих вредности циљне функције. Ово је уједно и најзначајнија предност новог приступа.

Будући рад на овом проблему може да иде у смеру који је усвојен овим приступом. Потребно је дефинисати боље и јаче везе између параметара и ограничења у циљу даљег повећања ефикасности поступка. Такође је потребно прецизно дефинисати псеудо-полиномијални алгоритам који би прецизно одредио услове његове примене. Најзад, ради свеобухватније експерименталне евалуације алгоритама је потребно конструисати нове, „теже“ инстанце пошто је општеприхваћени скуп 35 инстанци очигледно исувише једноставан за евалуацију поступака који би користили приступ предложен у дисертацији.

Табела 8-20. Поређење резултата 19 метода са резултатима добијеним применом CFOPT

Бр.	ZODIAC	GRAFICS	GATSP	GA	EA	ES	HGGA	SACF	HGA	GAA	HHA	EnGGA	GRASP	WFA	HGDE	ACO	SA	SAYLL	GAVNS	CFOPT
1	73.68	73.68			73.68	73.68	82.35		73.68			82.35	73.68	73.68	82.35	82.35	75	82.35	82.35	82.35
2	56.22	60.87	68.00	62.50	62.50	60.87	69.57	69.57	69.57	69.57	69.57	69.57	62.50	62.50	69.57	69.57	69.57	69.57	69.57	69.57
3			77.36	77.36	79.59	79.59	79.59	79.59	79.59	79.59	79.59	79.59	79.59	79.59	79.59	79.59	80.85	79.59	80.85	80.85
4			76.92	76.92	76.92		76.92	76.92	76.92	76.92	76.92	76.92	76.92	76.92	76.92	76.92	79.17	76.92	79.17	79.17
5	39.13	53.12	46.88	50.00	53.13	53.13	60.87	60.87	58.62	60.87	60.87	60.87	53.13	53.13	60.87	60.87	60.00	60.87	60.87	60.87
6			70.37	70.37	70.37	70.37	70.83	70.83	70.37	70.83	70.83	70.83	70.37	70.37	70.83	70.83	70.83	70.83	70.83	70.83
7	68.30	68.30			68.29	68.29	69.44		68.30			69.44	68.29	68.29	69.44	69.44	69.44	69.44	69.44	69.44
8	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25	85.25
9	58.33	58.13	58.33	55.91	58.72	58.72	58.72	58.41	58.72	58.72	58.72	58.72	58.72	58.72	58.72	58.72	58.72	58.72	58.72	58.72
10	70.59	70.59	70.59	72.79	70.59	70.59	75.00	75.00	70.59	75.00	75.00		70.59	70.59	75.00	75.00	75.00	75.00	75.00	75.00
11	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00		92.00	92.00	92.00	92.00	92.00	92.00	92.00	92.00
12	64.36	64:36			69.86	69.34	72.06		70.83				69.86	69.86	72.06	72.06	74.24	72.06	71.83	74.24
13	65.55	65.55	67.44	63.48	69.33	69.24	71.83	71.21	70.51	71.83	71.83		69.33	69.33	71.83	71.83	72.86	71.83	72.86	72.86
14	32.09	45.52			52.58	51.96	52.75		51.96			53.26	51.96	51.96	53.41	52.75	53.33	53.26	53.33	53.85
15	67.83	67.83			67.83	67.83	68.99		67.83			68.99	67.83	67.83	68.99	68.99	69.92	68.99	69.92	70.76
16	53.76	54.39	53.89	86.25a	54.86	54.86	57.53	52.44	54.86	56.13	56.38	57.53	56.52	55.90	57.53	57.53	57.98	57.53	57.42	57.98
17	41.84	48.91			54.46	54.46	57.73		54.95			57.73	54.46	54.46	57.73	57.73	57.73	57.73	57.73	57.73
18	21.63	38.26	37.12	34.16	42.94	42.96	43.18	41.04	43.45	42.94	43.26		42.96	42.96	43.45	43.45	43.97	43.45	43.18	43.97
19	38.96	49.36	46.62	39.02	49.65	49.65	50.81	50.81	49.65		50.81		49.65	49.61	50.81	50.81	50.81	50.81	50.81	50.81
20	75.14	75.14	75.28	66.30	76.22	76.14	77.91	78.40	76.14	77.91	78.40	77.91	76.54	76.54	77.91	77.91	79.38	77.91	77.78	79.38
21			55.14	44.44	58.07	58.06	57.98	56.04	58.38		57.61	57.98	58.15	58.15	57.98	57.98	58.79	57.98	57.61	58.79
22	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
23	85.11	85.10	85.10	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11	85.11
24	37.85	73.51	73.51	73.03	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51	73.51
25	20.42	43.27	43.27	37.62	51.88	51.85	53.29	52.44	52.50	52.87	53.29	53.29	51.97	51.97	53.29	53.29	53.29	53.29	53.27	53.29
26	18.23	44.51	44.67	34.76	46.69	46.50	48.95	47.13	46.84	48.95	48.63	48.95	47.37	47.37	48.95	48.95	48.57	48.95	46.66	48.95
27	17.61	41.67	42.50	34.06	44.75	44.85	47.26	44.64	44.85	47.26	46.15	46.58	44.87	44.87	47.26	47.26	46.00	47.26	47.16	47.26
28	52.14	47.37			54.27	54.27	54.02		54.31			54.82	54.27	54.27		54.82	54.82	54.82	54.31	54.82
29	33.01	32.86			44.37	43.85	46.91		46.43				46.06	46.06		47.08	47.68	47.23	46.06	47.68
30	33.46	55.43	53.80	40.96	58.11	57.69	63.31	62.42	60.74		62.59	63.31	59.52	59.52	63.31	63.31	62.86	63.31	63.04	63.31
31	46.06	56.32	56.61	48.28	59.21	59.43	59.77	60.12	59.66	60.12	60.12		60.00	60.00	59.77	59.77	59.66	59.77	60.12	60.12
32	21.11	47.96	45.93	37.55	50.48	50.51	50.83	50.51	50.51	50.83	50.83		50.51	50.51		50.83	50.55	50.83	50.68	50.83
33c	32.73	39.41			42.12	41.71	46.35		44.67				45.93	46.15		47.11	47.93	47.14	46.03	47.93
34	52.11	52.21			56.42	56.14	60.64		59.60			60.64	59.85	59.85	60.64	60.64	61.16	60.64	60.57	61.16
35	83.92	83.92	84.03	83.90	84.03	84.03	84.03	84.03	84.03	84.03	84.03		84.03	84.03		84.03	84.03	84.03	84.03	84.03

9. ПРОБЛЕМ РАСПОРЕДА ПРОИЗВОДНИХ ЋЕЛИЈА

Један од најинтересантнијих проблема у рачунарски интегрисаној производњи је распоред ћелија у флексибилним производним системима (Ilić, Računarski integrisana proizvodnja, 2015). Примери из праксе овако дефинисаног проблема могу да буду разноврсни, од распоређивања машина у производним халама до планирања комплетних *lay-out*-а фабрика или система производних хала. Сви проблеми овог типа имају заједнички назив **распоред објеката** (*facility layout* - FLA). Проблем распореда производних ћелија може да се сведе на један од најинтересантнијих проблема комбинаторне оптимизације, проблем квадратне асигнације који ће се у даљем тексту означавати као ПКА. Свођење на ПКА је могуће у случајевима:

- када су димензије површине на коју се распоређују ћелије знатно веће од димензија ћелија,
- када су димензије ћелија међусобно једнаке, или
- када се димензија површине на коју се распоређују ћелије не узима у обзир.

Истраживања у дисертацији су усмерена на проблем распореда ћелија који могу да се сведу на ПКА.

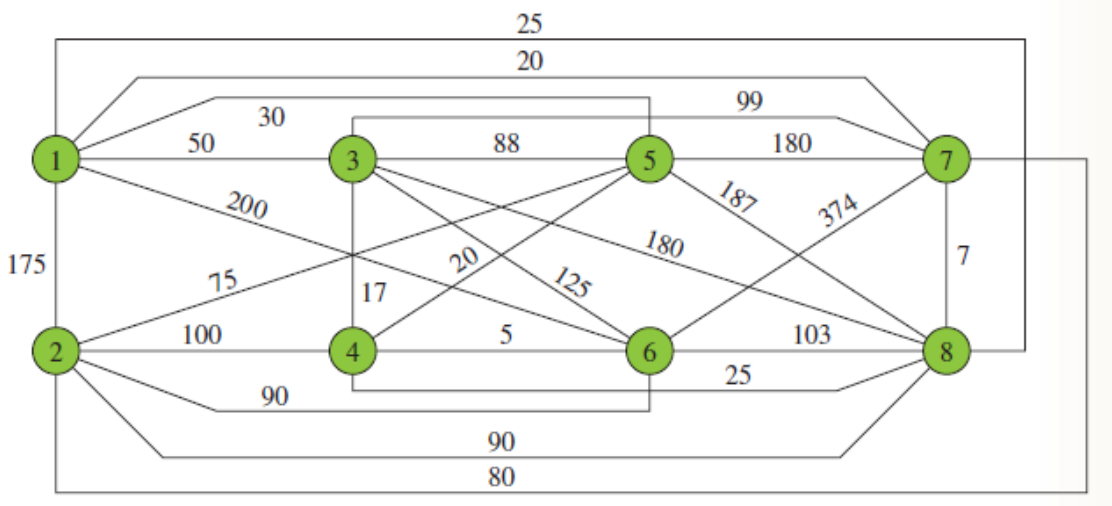
Проблем распореда ћелија се састоји од поделе дате површине познатих димензија на одељења датих (фиксних) површина и распореда ћелија у та одељења тако да се минимизира тотални трошак услед интеракције између ћелија. Овај трошак најчешће представља транспортни трошак. Уколико ширина и дужина одељења може да варира, онда је и одређивање оптималних димензија одељења такође део проблема. Ово је сложенији проблем од ПКА**,** јер не може да се дозволи преклапање одељења. Када су димензије одељења фиксне, па се оптимизација врши над коначним скупом могућих локација, проблем се директно своди на ПКА**.**

У интегрисаној производњи FLA се дели на више група у зависности од типа протока који се обавља између ћелија или одељења, али грубо, сви они могу да се сврстају у три основне групе: процесни распоред, распоред по производима и распоред са фиксним позицијама. Процесни распоред, који се често назива и *job*-

shop или *functional layout*, је формат у коме је груписана слична опрема или операције. Део који се обрађује путује, према дефинисаном редоследу операција, од области до области у којима су лоциране одговарајуће машине за те операције. На пример, одељења у фабрици играчака би могла да буду: пријем материјала, пластична обрада, метална обрада, операције на текстилу, склапање малих играчака, склапање великих играчака, фарбара, уградња механичких делова. Уколико ова одељења обележимо бројевима од 1 до 8, проток између одељења представљамо горњом троугаоном матрицом, приказаном на Слици 9-1, односно графом протока, приказаним на Слици 9-2.

	1	2	3	4	5	6	7	8	
1		175	50	0	30	200	20	25	1
2			0	100	75	90	80	90	2
3				17	88	125	99	180	3
4					20	5	0	25	4
5						0	180	187	5
6							374	103	6
7								7	7
8									8

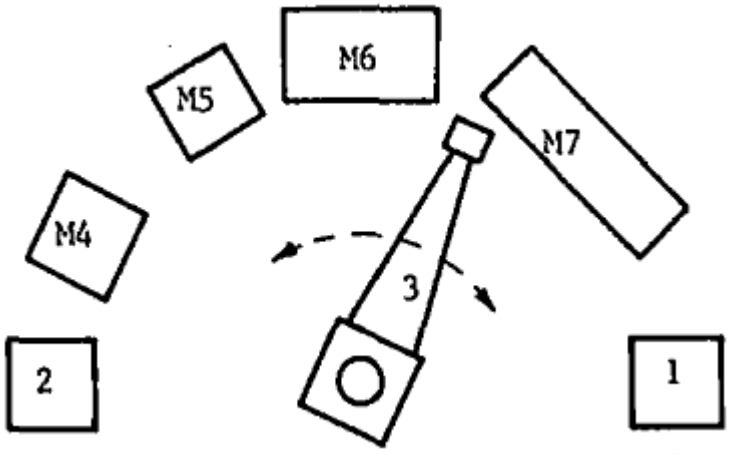
Слика 9-1. Проток између одељења у фабрици играчака



Слика 9-2. Граф протока између одељења у фабрици играчака

Распоред према производима (*product layout*) често се назива *flow-shop layout* и представља распоред у коме су опрема или процеси уређени према прогресивним корацима по којима се прави производ. Путања за сваки део је најчешће права линија. Суштинска разлика између овог распореда и процесног распореда је тип протока. У процесном распореду материјал за обраду може више пута да се допреми у исто одељење у току производног циклуса, док су у *flow-shop* распореду опрема или одељења директно распоређена за одређени корак циклуса, тако да се често опрема може и да дуплира, да не би било повратка на претходну локацију. Овакав распоред има смисла када су производне количине производа велике.

Фиксни распоред (*fixed-position layout*) је карактерисан релативно малим бројем производних јединица у поређењу са претходним распоредима. У оваквом распореду обрађивани производ смештамо у центар точка, на чијем су ободу материјали и алати који се по дефинисаном распореду употребљавају у производњи (Слика 9-3).



Слика 9-3. Фиксна локација обрађиваног дела

Сви ови распореди могу да се пројектују коришћењем поступака рачунарске оптимизације, с тим што се у линијским распоредима ПКА поједностављује и примењује линеарни поступак одређивања линијског распореда.

ПКА је класични комбинаторни проблем и представља један од најизазовнијих проблема у светској литератури. Оваква заинтересованост за овај проблем потиче из три разлога.

- Прво, број проблема из реалног живота који математички могу да се моделирају као ПКА континуално расте, док је разноврсност поља којима ти проблеми припадају заиста задивљујућа. Ту спадају проблем просторног и временског распоређивања, пројектовање VLSI (*very large scale integration*) кола, статистичка анализа података као и паралелно и дистрибуирано програмирање.
- Друго, велики број других, веома познатих комбинаторних оптимизационих проблема могу да се формулишу као ПКА. Типични примери су проблем трговачког путника, као и велики број проблема теорије графова, као што су одређивање максималних клика, проблем партиционисања чворова графа и одређивања минималног скупа грана графа.
- Најзад, у погледу рачунарске сложености, ПКА је један од најтежих проблема, тако да до данас није пронађено опште решење у виду алгоритма који овај проблем решава у разумном времену рада рачунара за величине проблема изнад $n = 16$.

ПКА су први представили Koopmans и Beckmann (Koopmans & Beckmann, 1957) као математички модел за локацију скупа недељивих економских активности**.** Посматран је проблем распоређивања скупа објеката на скуп локација, при чему је цена функција растојања и протока између објеката, увећана за цену која се плаћа за постављање објекта на одређену локацију. Задатак је да се сваком објекту додели једна локација тако да укупна цена буде минимална. Koopmans и Beckmann су ПКА формулисали као проблем одређивања оне пермутације π^n објеката, која минимизира функцију:

$$Z = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{i,j} d_{\pi(i),\pi(j)} + \sum_{i=1}^{n} c_{i,\pi(i)}\,, \tag{17}$$

где $f_{i,j}$ представља проток између објекта i и објекта j**,** $d_{i,j}$ је растојање између локације i и локације j, а $c_{i,j}$ је цена постављања објекта i на локацију j. Овако дефинисани ПКА са улазним матрицама F, D і C садржи квадратни и линеарни

члан који су дати у потпоглављу 9.2 (стр. 121). Оптимизација линеарног члана, иако такође НП-тежак проблем, представља проблем нижег реда у односу на оптимизацију квадратног члана, те се у највећем броју објављених поступака спроводи само минимизација квадратног члана.

Општију дефиницију ПКА је увео (Lawler, 1963) у којој се користи четвородимензионални низ $B = (b_{i,j,k,l})$ коефицијената:

$$Z = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{i,j,\pi(i),\pi(j)} + \sum_{i=1}^{n} c_{i,\pi(i)} \qquad (18)$$

Lawler-ова верзија ПКА може да се формулише као Koopmans – Beckmann-ова, сменом $b_{i,j,k,l} = f_{i,j}\, d_{k,l}$ за све *i, j, k, l,* за које је *i* различито од *j* или *k* различито од *l* и $b_{i,i,k,k} = f_{i,i}\ d_{k,k} + c_{i,k}$ у осталим случајевима. Суштински, уколико нема линеарног члана, ова дефиниција не уводи ништа ново, сем што уместо две матрице F и D користи њихов Хадамаров скаларни производ.

Постоји више приступа решавању овог проблема, али сви поступци могу да се разврстају у две основне групе. У првој групи се на основу модела математичког програмирања примењују линеарно целобројно или мешовито програмирање или, веома ретко нелинеарно програмирање. Друга група садржи алгоритме који директно оперишу са пермутацијама и на различите начине покушавају да смање број пермутација и у неком разумном времену добију егзактно решење или решење које се налази у границама прихватљивости.

Gilmore (Gilmore, 1962) је 1962. године предложио конструктивну процедуру која гради пермутацију, односно допустиво решење, кроз итерације алгоритма. У овом поступку, уводе се скупови A и L, где се први односи на лоциране објекте, а други на заузете локације. Оба скупа су на почетку поступка празна. Конструкција пермутације се обавља у свакој итерацији тако да се, по одређеном критеријуму, изабрани нерспоређени објекат распоређује на изабрану незаузету локацију. Поступак се наставља док се сви објекти не распореде на све локације. Ово је класичан пример ПКХ. Објављен је велики број ПКХ за решавање ПКА, од којих су најзначајнији (Armour & Buffa, 1963), (Buffa, Armour, & Vollmann, 1964),

(Sarker, Wilhelm, Hogg, & Han, 1995), (Sarker, Wilhelm, & Hogg, 1998), (Tansel & Bilen, 1998), (Burkard, Locations with spatial interactions: The quadratic assignment problem, 1991), (Arkin, Hassin, & Sviridenko, 2001), (Gutin & Yeo, 2002) и (Yu & Sarker, 2003). Све ове хеуристике су давале коначно допустиво решење, док су од ПКХ које су коришћење за добијање почетних решења најзначајније хеуристике које су предложили (Misevicius, 1997), (Fleurent & Glover, 1999), (Misevicius & Riskus, 1999), (Punnen, Sripratak, & Karapetyan, 2015) и (Punnen & Wang, 2016).

У овом поглављу је предложен једноставан GCA за ПКА којим се паралелно прате равноправне секвенце. Циљ је да се прикаже једноставност имплементације генерализованог приступа за ПКА и да се резултати добијени овим новим алгоритмом упореде са резултатима из литературе. Прецизније, циљ је да се покаже како се у полиномијалном времену рада, са једноставном GCA имплементацијом, могу да добију изузетно квалитетни резултати.

9.1. Сложеност проблема и асимптотско понашање

ПКА је НП-тежак проблем, шта више, одређивање епсилон апроксимативног решења је такође у класи НП. У раду (Queyranne, 1986) је изведен значајан резултат по питању сложености који потврђује зависност сложености ПКА са другим тешким комбинаторним проблемима. Показано је да је проблем трговачког путника, ПТП, специјалан случај ПКА. ПТП за n градова може да се формулише као ПКА(F,D), где је F матрица растојања између градова, а D матрица суседства Хамилтонове петље од n чворова. Најпознатији комбинаторни проблеми могу да се представе као посебан случај од ПКА. На пример:

- проблем партиције чворова графа: подела чворова графа у два подскупа, тако да гране између чворова ова два подскупа имају најмању тежину;
- проблем одређивања максималних клика: Проблем одређивања максималних потпуних подграфова датог графа.

Постоји неколико специјалних случајева ПКА који имају полиномијалну зависност времена извршења. Показано је да, уколико су и матрица растојања и

матрица протока тежинске матрице суседстава стабла, проблем може да се реши динамичким програмирањем у полиномијалном времену. Међутим, уколико једна од ове две матрице не може да се представи као тежинска матрица суседстава стабла, проблем остаје у класи НП. Полиномијално време извршења има и проблем у коме је једна од матрица тежинска матрица суседства двоструке звезде. Такође, када су обе матрице тежинске матрице суседства серијско-паралелног графа који не садржи бипартитни граф $K_{2,2}$, одговарајући ПКА може да се реши у полиномијалном времену.

Најновији резултати дају делимичан одговор на једно од отворених питања које је постављено у (Queyranne, 1986). Шта се дешава ако су локације правилно распоређене на линији. Овај специјални случај ПКА-а је назван **проблем линеарног уређења**, који је добро проучени проблем класе НП. У раду (Arora, Frieze, & Kaplan, 1996) је дата и апроксимативна шема општег случаја која има полиномијалну зависност извршења од величине проблема. Ова апроксимација је заснована на одређеним условима које треба да задовољава матрица A.

ПКА поседује једну веома интересантну особину. Наиме, релативна разлика између најгорег решења и оптималног решења тежи нули са вероватноћом која тежи јединици када величина проблема тежи бесконачности. Математички, за дато произвољно мало ε, ε > 0 важи:

$$P\left(\frac{F^{+}}{F^{-}} < 1+\varepsilon\right) \geq 1 - 2n!e^{-\alpha n^2}.$$

Како $2n!\,e^{-\alpha n^2}$ тежи нули када n тежи бесконачности, из израза се види да за велико n вероватноћа израза у загради тежи јединици. F^{+} и F^{-} су максимална односно минимална вредност циљне функције за ПКА, те је из израза такође очигледно да за довољно мало ε и довољно велико n ове две вредности теже једна другој.

Због свега горе изложеног правци развоја решавања проблема ПКА су се усмерили на:

- линеаризацију проблема,

- одређивање доњих граница одступања и дефинисања процентуалног одступања од оптималне вредности,
- развијање егзактних алгоритама за посебне класе проблема и
- развијање хеуристичких алгоритама којима се одређују решења која нису оптимална али су у допустивим границама одступања.

9.2. Математички модели са пермутационим матрицама

За многе комбинаторне оптимизационе проблеме постоје различите, али еквивалентне математичке формулације, које истичу различите структурне карактеристике проблема, које могу да доведу до различитих приступа решавању. Најчешћа формулација је заснована на чињеници да свака пермутација π скупа N = {1, 2, ..., n} може да буде представљена n x n матрицом $X = (x_{i,j})$, таквом да је увек:

$$x_{i,j} = \begin{cases} 1\,, \pi(i) = j \\ 0\,, \pi(i) \neq j \end{cases}.$$

Матрица X се зове пермутациона матрица и карактерисана је ограничењима:

$$\sum_{i=1}^{n} x_{i,j} = 1\,,\ j = 1,...,n \text{ и } \sum_{j=1}^{n} x_{i,j} = 1\,,\ i = 1,...,n\,.$$

Веза између пермутације и пермутационе матрице следи из саме дефиниције пермутационе матрице. Нека је пермутација π од m елемената дата следећом формом:

$$\begin{pmatrix} 1 & 2 & \cdots & m \\ \pi(1) & \pi(2) & \cdots & \pi(m) \end{pmatrix},$$

тада је одговарајућа пермутациона матрица, $m \times m$ матрица X_π чији су сви елементи нуле, осим што је у сваком реду i елемент $\pi(i)$ једнак 1. Може да се напише да је:

$$\mathbf{X}_{\pi} = \begin{bmatrix} \mathbf{e}_{\pi(1)} \\ \mathbf{e}_{\pi(2)} \\ \vdots \\ \mathbf{e}_{\pi(m)} \end{bmatrix},$$

где e_j означава вектор реда, дужине m са јединицом на j-тој позицији и нулама на свим осталим позицијама.

Пермутациона матрица омогућава да се дефинише квалитативно другачији тип модела од оног који је дат изразима (17) и (18). Наиме, у овим изразима функција пермутације π није алгебарска, те овакав модел не може да се примени у стандардним оптимизационим *solver*-има. Користећи пермутациону матрицу уместо пермутација, ПКА може да буде формулисан као целобројни програм са квадратном оптимизујућом функцијом:

$$(\min) Z = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{t=1}^{n} b_{i,j,k,t} x_{i,k} x_{j,t} + \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} x_{i,j}$$

п.о.

$$\sum_{i=1}^{n} x_{i,j} = 1,\ j = 1,...,n$$

$$\sum_{j=1}^{n} x_{i,j} = 1,\ i = 1,...,n$$

$$x_{i,j} = 0 \vee 1 \quad i = 1,...,n;\ j = 1,...,n.$$

У овом моделу, $b_{i,j,k,t}$ има различиту форму од $b_{i,j,\pi(i),\pi(j)}$ у пермутационом моделу. Док у пермутационом моделу члан b може да се представи преко $n!$ матрица B реда nxn, у овом моделу b се представља једном матрицом реда n^2xn^2 која може да се дефинише преко Кронекеровог производа B = F⊗D:

$$F \otimes D = \begin{bmatrix} f_{1,1}D & \cdots & f_{1,n}D \\ \vdots & \ddots & \vdots \\ f_{n,1}D & \cdots & f_{n,n}D \end{bmatrix}$$

У овако дефинисаној матрици, $b_{i,j,k,t}$ се налази у $(i-1)n+k$-том реду и $(j-1)n+t$-тој колони. Чланови $x_{i,k}$ $x_{j,t}$ означавају да је истовремено објекат i постављен на локацију k, а објекат j на локацију t. Ови чланови такође могу да буду

представљени матрицом реда $n^2 x n^2$, која се означава са X и у којој распоред елемената одговара распореду елемената $b_{i,j,k,t}$ у матрици B. Сада се визуелно могу да повежу ове две матрице тако да се у поља уписују елементи матрице B, а у ознаке редова и колона уписују елементи матрице X.

9.3. Нови алгоритам

Нови алгоритам за проблем распореда производних ћелија је:

$$x \leftarrow GCA(PR^n, F, C, g = \min, \pi p^n, \xi, ord, \boldsymbol{arg_2}, \boldsymbol{arg_3} = 1, \boldsymbol{arg_4} = R_2).$$

Аргумент $\boldsymbol{arg_2}$ одређује максималан број паралелно праћених секвенци, $\boldsymbol{arg_3}$ = 1 дефинише да се у итерацијама, критеријум селекције ξ примењује на комплетну секвенцу, а $\boldsymbol{arg_4} = R_2$ означава да се кроз итерације паралелно прати $\boldsymbol{arg_2}$ секвенци. Решење x представља оптималну пермутацију скупа локација при фиксној пермутацији скупа објеката, πp^n одређује почетно уређење локација и трајно уређење објеката, а f_2 бира прву слободну локацију у уређеном низу локација који ће се обрађивати у итерацији. Критеријум ξ одређује парцијално оптимално додељивање објеката локацијама.

Уређење πp^n се спроводи аналогно за објекте и локације. Одреде се два објекта са највећим збировима протока ка свим осталим објектима. Њихово међусобно уређење је произвољно. Трећи објекат у уређеном низу је објекат са највећим протоком ка претходно одабрана два објекта. Поступак се наставља на исти начин, тако да се од нераспоређених објеката бира онај са највећим збиром протока ка распоређеним објектима. Почетно уређење локација је аналогно, с тим да су прве две локације у низу оне са најмањим збиром растојања према свим осталим локацијама, и да се од нераспоређених локација бира она са најмањим збиром растојања од распоређених локација.

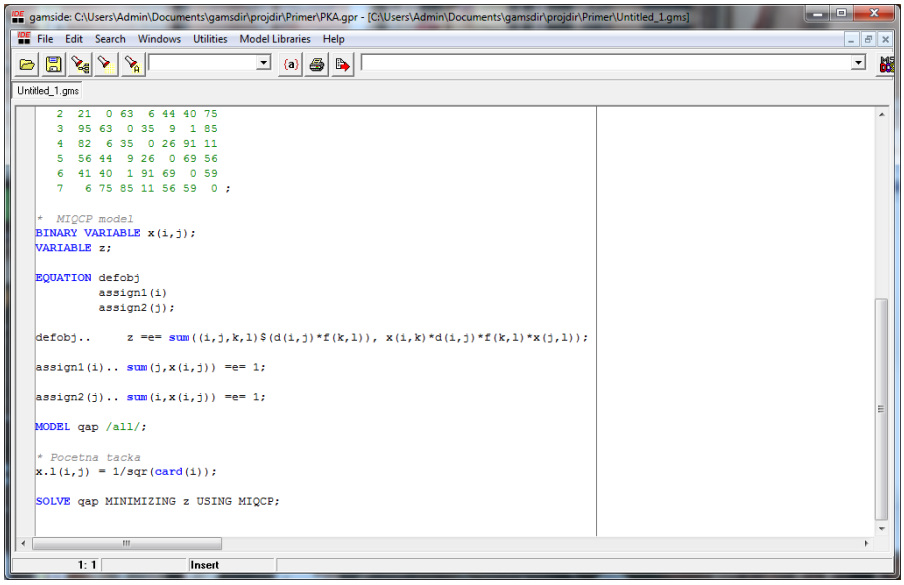
Критеријум ξ функционише на следећи начин. Локацији која је одабрана критеријумом f_2, додељује се објекат који резултује најмањом вредношћу циљне функције. Ако тај објекат није био додељен некој локацији, прелази се на следећу итерацију. Уколико је тај објекат био додељен одређеној локацији, са овом

локацијом се поступак понавља, све док се не дође до објекта који није био додељен ни једној локацији.

Претходна формулација једнозначно дефинише све кораке алгоритма. Пажљивом анализом може да се закључи да је довољно по претходном поступку одабрати само прва два објекта у низу, јер уређење осталих објеката не утиче на крајњи резултат. Међутим, усвојено је да се одради комплетно уређење објеката, јер је временска сложеност овог поступка само $n\log n$ и, захваљујући овом уређењу, ξ ефикасније одређује парцијално оптимално додељивање објеката локацијама.

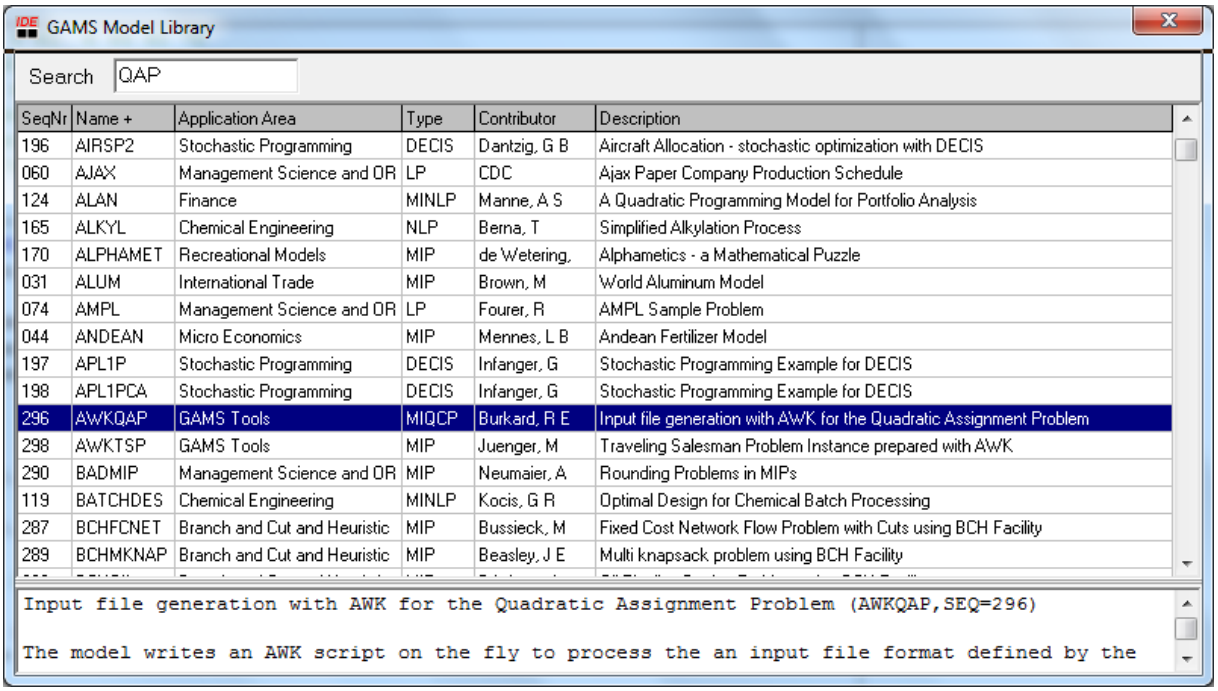
9.4. Неки експериментални резултати

За експерименталну проверу је коришћен *solver* GAMS, компаније *GAMS Development Corporation, Washington, DC, USA*. Аутори референтне QAPLIB (Burkard, Karisch, & Rendl, 1997) датотеке решених и нерешених инстанци ПКА директно сарађују са овом компанијом. Начин рада овог програма је приказан на примеру величине 7 који је узет из QAPLIB датотеке и за који је модел направио један од њених оснивача Burckard. Интегрисано развојно окружење програма GAMS је приказано на Слици 9-4.



Слика 9-4. Интегрисано развојно окружење програма GAMS

Када се GAMS покрене први пут, програм тражи од корисника да креира пројекат, који служи да би се запамтила подешавања која су специфична за тај пројекат. Пројекат не садржи никакав код. Као што се види на Слици 9-4 главни прозор је организован у облику панела, тако да се избором одговарајућег језичка бира прозор који се приказује.

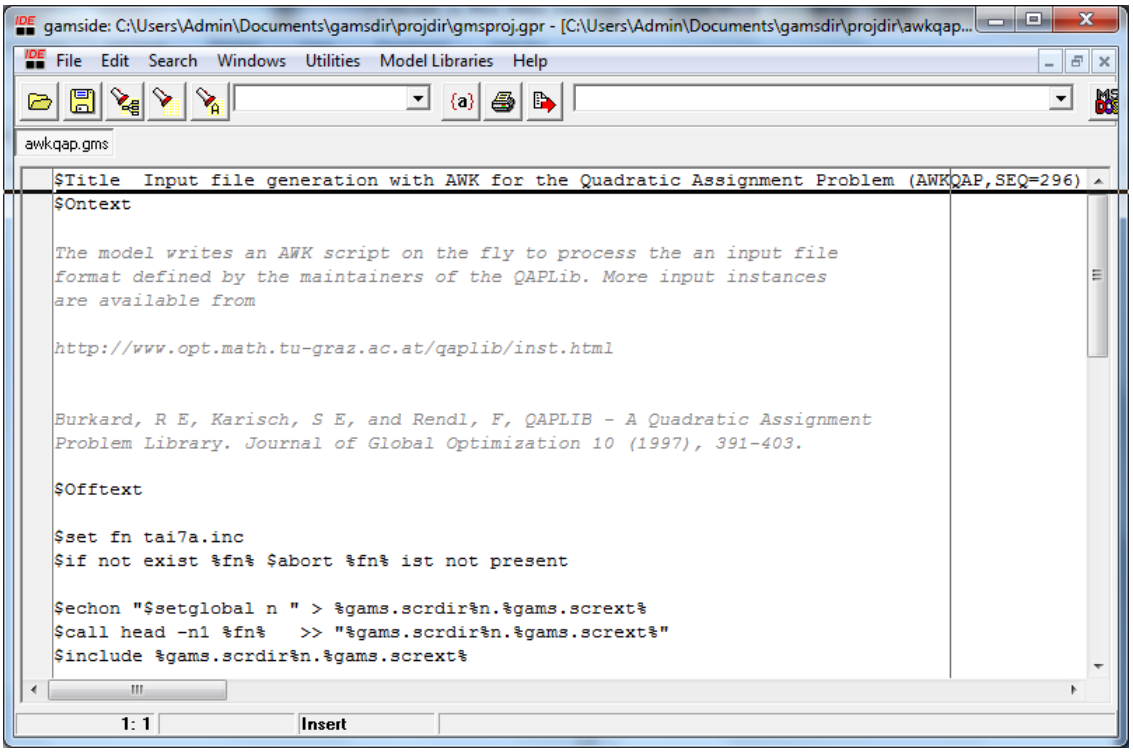


Слика 9-5. Избор модела

Корисник може да сам унесе текст који дефинише посматрани модел, или да из GAMS библиотеке модела одабере неки од постојећих модела. Избором менија *File | Model library | Open GAMS model library* отвара се прозор у коме су излистани сви доступни модели. На Слици 9-5 је приказан овај прозор на коме је селектован модел проблема квадратне асигнације. Избором одговарајућег модела, у главном прозору се приказује текст програма који одговара том моделу (Слика 9-6).

Притиском на дугме на коме је уцртана црвена стрелица покреће се програм којим се оптимизује циљна функција за дати скуп улазних података. Као резултат се добијају разноврсни извештаји који садрже добијене резултате, све метаподатке о току оптимизације, као и тражене дијаграме. Уколико понуђени модели не

задовољавају потребе, корисник може сам да конструише модел у језику који је дефинисан GAMS програмом. Језик је једноставан, а могућности су такве да њиме може да се опише практично сваки модел.



Слика 9-6. Приказ модела у главном прозору

Конструисан је модел за решавање ПКА у језику програма GAMS са улазним подацима из QAPLIB библиотеке проблема:

```
$Title  Model za Problem Kvadratne Asignacije
$Ontext
Ovaj model predstavlja problem Kvadratne Asignacije
$Offtext
set   i / 1*7 /;  alias (i,j,k,l);

Table  d(i,j)  rastojanja izmedju lokacija
     1  2   3   4   5   6   7
 1   0 71  11  29  82  82   6
 2  71  0  74  30  89  76 76
 3  11 74   0   1  50   4  36
 4  29 30   1   0   1  15  11
 5  82 89 50   1   0  21  61
 6  82 76   4 15  21   0  76
 7   6  76  36 11  61 76   0    ;
```

```
Table  f(i,j)  tezinski faktori
     1  2   3   4   5   6   7
  1  0 21  95  82  56 41   6
  2 21  0  63   6  44 40  75
  3 95 63   0 35   9   1  85
  4 82  6  35   0 26 91  11
  5 56 44   9 26   0 69  56
  6 41 40   1 91 69   0  59
  7  6 75  85 11 56  59   0   ;

*  MIQCP model
BINARY VARIABLE   x(i,j);
VARIABLE   z;

EQUATION defobj
       assign1(i)
       assign2(j);

defobj..    z =e= sum((i,j,k,l)$(d(i,j)*f(k,l)), x(i,k)*d(i,j)*f(k,l)*x(j,l));

assign1(i).. sum(j,x(i,j)) =e= 1;
assign2(j).. sum(i,x(i,j)) =e= 1;

MODEL qap /all/;

* Pocetna tacka
x.l(i,j) = 1/sqr(card(i));
SOLVE qap MINIMIZING z USING MIQCP;
```

Добијена вредност циљне функције је 63,804 за пермутацију 3 2 1 6 4 5 7. Овај резултат објективно може да се пореди са резултатом добијеним новим алгоритмом на следећи начин. Време које је утрошио *solver* на добијање овог резултата је 0.729sec. Нови алгоритам са $\boldsymbol{arg_2}$ = 4,800 потроши такође 0.729sec и добије 4,800 секвенци од којих су најбољих 6 приказане у Табели 9-1.

Из ових резултата може да се уочи да је вредност, добијена GAMS *solver*-ом тек пети резултат у низу. Најбоља пермутација је 6 7 3 5 2 4 1, којом се добија вредност циљне функције 60,690 што је за 3,114 бољи резултат, тј. девијација GAMS резултата је 5.13%.

Табела 9-1. Шест најбољих пермутација добијених новим алгоритмом

Бр.	Пермутација	C	Одступање
1	6 7 3 5 2 4 1	60,690	1.0000
2	6 1 3 5 2 4 7	61,196	1.0083
3	1 5 3 7 2 4 6	62,508	1.0300
4	3 5 6 1 2 7 4	62,786	1.0345
5	3 2 1 6 4 5 7	63,804	1.0513
6	1 6 3 7 2 4 5	64,162	1.0572

Објашњење за овако драстичну супериорност новог алгоритма је тривијално. *Solver* троши значајно време на припрему, што има као последицу да је утрошак времена за мале инстанце несразмерно велики. За то време нови алгоритам може да одради 4,800 паралелних секвенци, што је скоро комплетан допустиви скуп, тако да се као резултат добија сигурно оптимално решење и сва остала решења у низу. Наравно, оваква супериорност не важи у општем случају, али су експериментални резултати показали да нови алгоритам даје боља решења и у случајевима када не може да се претражи комплетан допустиви скуп. Резултати добијени новим алгоритмом су поређени са резултатима добијеним *solver*-ом GAMS на 15 инстанци из QAPLIB библиотеке. Усвојено је $\boldsymbol{arg_2} = 250$, што је резултовало утрошком времена испод једне секунде на свим инстанцама. У следећем списку инстанци, имена аутора су праћена скраћеним именом инстанце, величином проблема, оптималном вредношћу циљне функције и оптималном пермутацијом:

N. Christofides and E. Benavent
Chr12a 12 9552 (OPT) (7,5,12,2,1,3,9,11,10,6,8,4)
Chr12b 12 9742 (OPT) (5,7,1,10,11,3,4,2,9,6,12,8)
Chr12c 12 11156 (OPT) (7,5,1,3,10,4,8,6,9,11,2,12)
Chr15a 15 9896 (OPT) (5,10,8,13,12,11,14,2,4,6,7,15,3,1,9)
Chr15b 15 7990 (OPT) (4,13,15,1,9,2,5,12,6,14,7,3,10,11,8)
Chr15c 15 9504 (OPT) (13,2,5,7,8,1,14,6,4,3,15,9,12,11,10)
S.W. Hadley, F. Rendl and H. Wolkowicz
Had12 12 1652 (OPT) (3,10,11,2,12,5,6,7,8,1,4,9)
Had14 14 2724 (OPT) (8,13,10,5,12,11,2,14,3,6,7,1,9,4)
C.E. Nugent, T.E. Vollmann and J. Ruml
Nug12 12 578 (OPT) (12,7,9,3,4,8,11,1,5,6,10,2)
Nug14 14 1014 (OPT) (9,8,13,2,1,11,7,14,3,4,12,5,6,10)
Nug15 15 1150 (OPT) (1,2,13,8,9,4,3,14,7,11,10,15,6,5,12)
E.D. Taillard
Tai12a 12 224416 (OPT) (8,1,6,2,11,10,3,5,9,7,12,4)
Tai12b 12 39464925 (OPT) (9,4,6,3,11,7,12,2,8,10,1,5)
Tai15a 15 388214 (OPT) (5,10,4,13,2,9,1,11,12,14,7,15,3,8,6)
Tai15b 15 51765268 (OPT) (1,9,4,6,8,15,7,11,3,5,2,14,13,12,10)

У Табели 9-2 су приказане оптималне вредности, OPT, резултати добијени *solver*-ом GAMS и резултати добијени новим алгоритмом, GCA.

Табела 9-2. Поређење резултата добијених *solver*-ом GAMS и GCA

Бр.	Инстанца	Величина	OPT	GAMS		GCA	
				C	Одст.	C	Одст.
1	Chr12a	12	9,552	10,106	5.80	9,903	3.67
2	Chr12b	12	9,742	10,295	5.67	9,950	2.14
3	Chr12c	12	11,156	11,710	4.97	11,515	3.22
4	Chr15a	15	9,896	10,601	7.12	10,215	3.22
5	Chr15b	15	7,990	8,498	6.36	8,223	2.92
6	Chr15c	15	9,504	10,336	8.75	9,817	3.29
7	Had12	12	1,652	1,748	5.83	1,710	3.51
8	Had14	14	2,724	2,891	6.12	2,811	3.19
9	Nug12	12	578	611	5.67	603	4.33
10	Nug14	14	1,014	1,077	6.18	1,035	2.07
11	Nug15	15	1,150	1,255	9.09	1,201	4.43
12	Tai12a	12	224,416	233,682	4.13	228,111	1.65
13	Tai12b	12	39,464,925	41,351,348	4.78	40,352,379	2.25
14	Tai15a	15	388,214	419,698	8.11	401,623	3.45
15	Tai15b	15	51,765,268	54,990,244	6.23	53,782,654	3.90

Несумњиво бољи резултати новог алгоритма треба да се прихвате условно. Наиме, као што је речено, циљ експерименталног поступка је да покаже како једноставна имплементација GCA на проблем распореда производних ћелија може да да резултате који су равноправни са објављеним резултатима. Како су резултати добијени применом GAMS референтни за велики број објављених резултата у литератури, представљени експериментални резултати недвосмислено потврђују хипотезу да је примена GCA на проблем распореда производних ћелија оправдана.

10. ПРОБЛЕМ РЕДОСЛЕДА ПОСЛОВА У ЛИНИЈИ

Проблем редоследа послова у линији, у литератури познат као *flow shop* проблем, је најчешће проучаван проблем у теорији распоређивања и планирања. Овај проблем може да се формулише на следећи начин. Сваки од n послова из скупа $j = \{1,2,\ldots,n\}$ је расположив у почетном нултом тренутку и треба да буде обрађен на m машина $M_1,M_2,\ldots,M_m$ по том редоследу. Ни једна машина не може да обрађује више од једног посла истовремено, ни један посао не може да се истовремено обрађује на више од једне машине, када започне да се обрађује, посао се комплетира без прекидања (*no-preemption*), сва припремна времена су укључена у време обраде посла и нема ограничења складиштења између машина. Проблем, који се најчешће означава као $F|prmu|C_{max}$ је одређивање редоследа послова који треба да се обави на свакој машини, тако да *makespan*, C_{max}, односно време завршетка последњег посла на M_m буде минимално. Уколико је m фиксирано, одговарајући проблем се означава са $Fm|prmu|C_{max}$. Пермутациона (*prmu*) верзија *flow shop* проблема, PFSP (*Permutation Flow Shop Problem*), подразумева да је на свакој машини усвојени редослед послова исти. У општем случају исти редослед послова на свим машинама не даје увек оптималан редослед, тако да је $F|prmu|C_{max}$ у општем случају једноставнији проблем од генералног, $F||C_{max}$.

Чувени алгоритам Џонсона (Johnson, 1954) и (Johnson, 1959) решава $F2|prmu|C_{max}$, који је истовремено и генерални $F2||C_{max}$ проблем. $Fm|prmu|C_{max}$, као и $F|prmu|C_{max}$ су строго НП-комплетни за m > 3 (Garey & Johnson, 1979). Како због овога егзактни алгоритми не могу да дају решење у задовољавајућем времену рада рачунара, у литератури постоје два приступа за решавање овог проблема и то коришћењем полиномијалних апроксимационих шема, PTAS (*Polynomial Time Approximation Scheme*), као и хеуристичких алгоритама. PTAS су апроксимације проблема, и то улазних података, ограничења и излазних података у циљу добијања проблема који је решив у полиномијалном времену и који би дао приближно оптималан редослед. Постоје PTAS (Hall, 1998) за $Fm|prmu|C_{max}$ и $Fm||C_{max}$, али не и за $F||C_{max}$, односно $F|prmu|C_{max}$. Објављен је огроман број радова у којима се предлажу, како конструктивне хеуристике, тако и хеуристике

побољшања. Од конструктивних метода, НЕХ хеуристика (Nawaz, Enscore Jr, & Ham, 1983) је опште прихваћена као најбоља до сада објављена. Скоро све познате методе метахеуристика као што су методе локалне претраге у (Ilić, 2003) и (Stutzle, 1998), гранања и ограничавања у (Companys & Mateo, 2007) и (Ladhari & Haouari, 2005), симулираног каљења у (Framinan, Gupta, & Leisten, 2004) и (Osman & Potts, 1989), табу претраге у (Ekskioglu, Ekskioglu, & Jain, 2008) и (Grabowski, Pempera, & ., 2001), генетских алгоритама у (Etiler, Toklu, & Wilson, 2004), (Chen, Vempati, & Aljaber, 1995) и (Ruiz, Maroto, & Alcaraz, 2006), система мрављих колонија у (Ying & Liao, 2004) и (Chandrasekharan & Ziegler, 2004), *particle swarm* оптимизације у (Tasgetiren, Sevkl, & Gencyilmaz, 2007) и друге, користе и НЕХ хеуристику за добијање почетног решења.

Без обзира који од приступа за решавање PFSP је примењен, у сваком се израчунавање *makespan*-а обавља велики број пута, прецизније, по једном у сваком циклусу за добијени привремени редослед послова. Због овога је веома важно да поступак за израчунавање *makespan*-а буде што је могуће ефикаснији. Укупна ефикасност поступка одређена је са *O*(*a***O*(*b*)), где се *O*(*b*) односи на одређивање *makespan*-а, а *a* представља број различитих редоследа који се алгоритмом третирају.

10.1. Дефиниције основних појмова

За дати низ послова $\pi := (\pi(1),...,\pi(n))$, нека $p_{i,\pi(j)}$ означава време обраде $\pi(j)$-тог посла на *i*-тој машини, $i = 1,...,m;\ j \in \{1,...,n\}$, и нека $C(i,\pi(j))$ означава време, протекло од почетног тренутка *t* = 0 до завршетка $\pi(j)$-тог посла на *i*-тој машини. Ова нотификација може да се, у случајевима који не стварају забуну, скрати на $p_{i,j}$ *и* $C(i,j)$. Одговарајуће *m* x *n* матрице су улазна матрица $\boldsymbol{P} = \left\| p_{i,j} \right\|$ и *makespan* матрица за дато π, $\boldsymbol{C}[\pi] = \left\| C(i,\pi(j)) \right\| \xrightarrow{скраћено} \boldsymbol{C} = \left\| C_{i,j} \right\|$. *Makespan* је тада $C_{\max}[\pi] = C_{m,\pi(n)}$, при чему се $C_{i,j}$ одређује из система једначина:

$$C_{1,1} = p_{1,1} \tag{22}$$

$$C_{1,j} = C_{1,j-1} + p_{1,j} \quad j = 2,...,n \tag{23}$$

$$C_{i,1} = C_{i-1,1} + p_{i,1} \quad i = 2,...,m \tag{24}$$

$$C_{i,j} = \max\{C_{i,j-1}, C_{i-1,j}\} + p_{i,j} \quad i = 2,...,m; j = 2,...,n \tag{25}$$

Једначине (23) и (24) могу да се представе и у апсолутном облику, а не у итеративном:

$$C_{1,j} = C_{1,j-1} + p_{1,j} = C_{1,j-2} + p_{1,j-1} + p_{1,j} = \sum_{k=1}^{j} p_{1,k} \quad j = 2,...,n \tag{26}$$

$$C_{i,1} = C_{i-1,1} + p_{i,1} = C_{i-2,1} + p_{i-1,1} + p_{i,1} = \sum_{k=1}^{i} p_{k,1} \quad i = 2,...,m \tag{27}$$

Из итеративне формуле (25) је јасно да је за одређивање $C_{m,n}$ неопходно одредити свих $n*m$ вредности $C_{i,j}$. Прва итерација даје:

$$C_{\max} = C_{m,n} = \max\{C_{m-1,n}, C_{m,n-1}\} + p_{m,n}.$$

Друга итерација примене рекурентне формуле (25) даје:

$$C_{\max} = \max\{\max\{C_{m-2,n}, C_{m-1,n-1}\} + p_{m-1,n}, \max\{C_{m-1,n-1}, C_{m,n-2}\} + p_{m,n-1}\} + p_{m,n}$$

$$C_{\max} = \max\{C_{m-2,n} + p_{m-1,n}, C_{m-1,n-1} + p_{m-1,n}, C_{m-1,n-1} + p_{m,n-1}, C_{m,n-2} + p_{m,n-1}\} + p_{m,n}.$$

Лако може да се установи правило по коме се образују формуле у следећим итерацијама: збирови индекса уз C су $m + n - k$, а уз p редом $m + n - (k - 1)$, $m + n - (k - 2)$,..., $m + n$, где је k број итерације. Уколико је $m - k = 1$ примењује се (26), а уколико је $n - k = 1$ primenjuje se (27).

На основу изложеног, могу да се прегрупишу суме и преуреде сабирци у сумама тако да се добије нова, веома значајна формулација *makespan*-а:

$$C_{\max} = \max_{1 \le k_1 \le .. \le k_{m-1} \le n} [R], \tag{28}$$

при чему је:

$$R = \sum_{j=1}^{k_1} p_{1,j} + \sum_{j=k_1}^{k_2} p_{2,j} + ... + \sum_{j=k_{m-2}}^{k_{m-1}} p_{m-1,j} + \sum_{j=k_{m-1}}^{n} p_{m,j}.$$

R се састоји од *m* парцијалних сума у којима су фиксиране вредности за *i*, редни број машине. Суме могу да имају најмање један, а највише *n* сабирака. Ова формулација је основ за већину познатих поступака за одређивање *makespan*-а.

10.2. Поступци за одређивање укупног времена производње

На основу претходних дефиниција могу да се изведу различити поступци за израчунавање *makespan*-а. У наставку ће *makespan* подразумевати укупно време обраде свих делова на свим машинама, односно време извршења свих послова на свим машинама. Израчунавање *makespan*-а ће бити приказано на примеру у коме је матрица ***P*** дата у Табели 10-1.

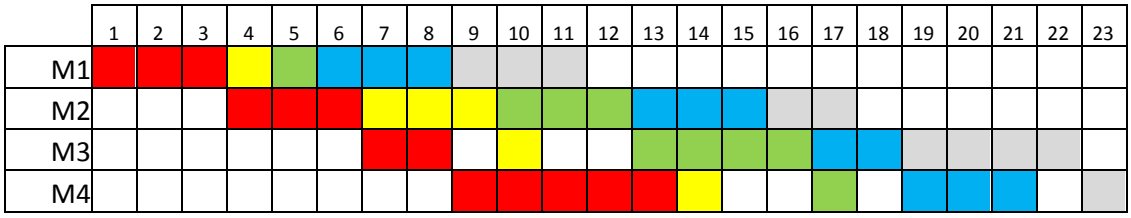
Табела 10-1. Улазна матрица за одређивање *makespan*-а

P[4,5]=

3	**1**	**1**	**3**	**3**
3	**3**	**3**	**3**	**2**
2	**1**	**4**	**2**	**4**
5	**1**	**1**	**3**	**1**

10.2.1 *Графички поступак*

На Слици 10-1 је приказан графички поступак одређивања *makespan*-а.



Слика 10-1. Графичка метода израчунавања *makespan*-а

Први посао је означен црвеном бојом, други жутом, трећи зеленом, четврти плавом и пети сивом. Сваки од четири реда представља редослед послова на једној од четири машина. Колоне представљају временске јединице у којима се обављају послови на машинама. Временска јединица започињања одређеног посла на одређеној машини је условљена завршетком тог посла на претходној машини и завршетком претходног посла на тој машини. *Makespan* одговара колони у којој се

завршава последњи посао у последњем реду графика, тј. вредност *makespan*-а је за дати пример 23.

10.2.2 *Матрични поступак*

Овај тип поступака за одређивање C_{max} је заснован на формулацији (25)-(27). Овим поступком се одређује матрица $C[m,n] = \|C_{i,j}\|$, при чему је, по завршетку алгоритма, $C_{max} = c_{mn}$. Овај тип алгоритама је представљен као Алгоритам 10-1.

Алгоритам 10-1: Матрични поступак за израчунавање C_{max}

```
1    INPUT: P[m,n]
2    c_{1,1} = p_{1,1}
3    for j = 2 to n do
4       c_{1,j} = c_{1,j-1} + p_{i,j}
5    for i = 2 to m do
6       c_{i,1} = max(c_{i-1,1} + p_{i,1}, c_{i-1,2})
7       for j = 2 to n-1 do
8          c_{i,j} = max(c_{i,j-1} + p_{i,j}, c_{i-1,j+1})
9       c_{i,n} = c_{i,n-1} + p_{i,n}
10   OUTPUT: C[m,n] ; C_max = c_{m,n}
```

Алгоритам 10-1 у првој петљи 3-4 попуњава прву врсту матрице C сабирајући претходна времена на првој машини са трајањем тренутно посматраног посла j. Петља 5-9 и у њој садржана петља 7-9 попуњавају преостала поља матрице A према формулама (25) до (27). Први елементи у свакој врсти се попуњавају другачије од осталих елемената матрице.

За претходни пример, поступак решавања је дат на Слици 10-2:

3	4	5	8	11
6				
8				
13				

3	4	5	8	11
6	9	12	15	17
8				
13				

3	4	5	8	11
6	9	12	15	17
8	10	16	18	22
13				

3	4	5	8	11
6	9	12	15	17
8	10	16	18	22
13	14	17	21	23

Слика 10-2. Пример матричне методе

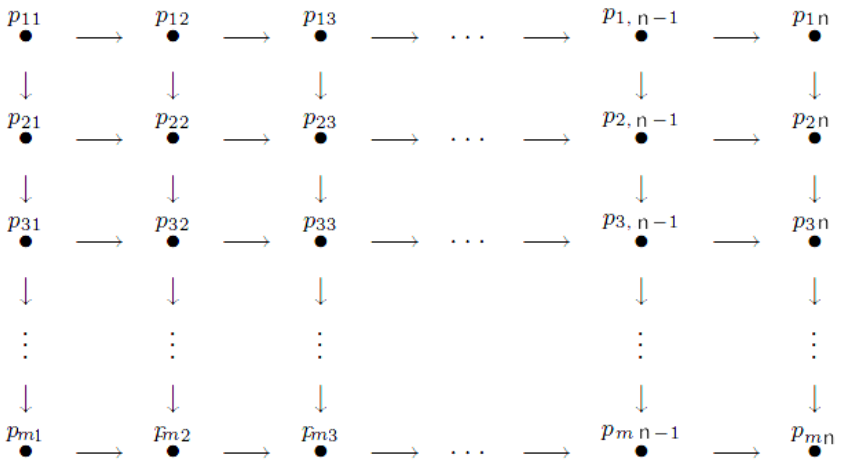
10.2.3 *Поступак са графовима*

Формула (27), с друге стране, дефинише скуп могућих путања којима се може, полазећи од позиције $p_{1,1}$, која репрезентује обављање првог посла на првој машини, стићи до позиције $p_{m,n}$, која означава обављање n-тог посла на m-тој машини. Овакве формулације представљају директну аналогију са графовима, односно мрежама и поступцима за одређивање путања у њима.

Одговарајућа мрежа има $n*m$ чворова од којих сваки представља обављање одређеног посла на одређеној машини, конкретно, сваки чвор репрезентује по једну позицију $p_{i,j}$. Ради прегледности, усвојено је да чвор $v_{i,j}$ одговара позицији $p_{i,j}$, а реални број $w(v_{i,j}) = p_{i,j}$.

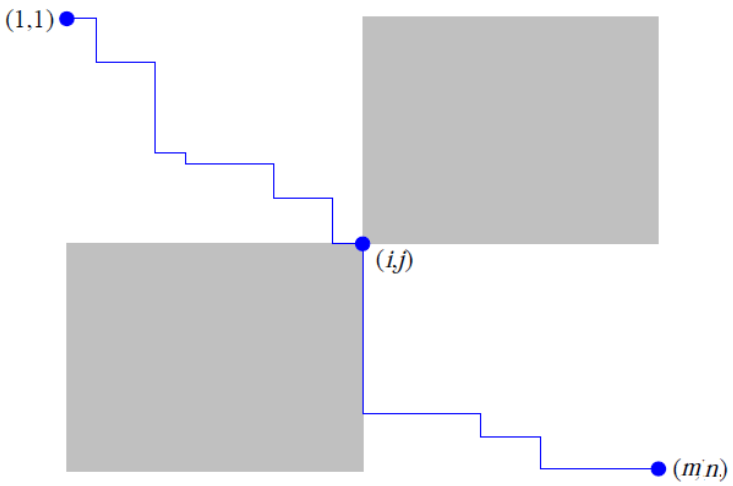
На основу дефиниције мреже која представља посматрани проблем одређивања C_{max}, решење проблема представља одређивање најдуже путање у посматраној

мрежи која полази из v_{11} и завршава у $v_{m,n}$. Формула (10) дефинише правила која треба да задовољавају путање мреже да би одговарале посматраном проблему. На Слици 10-3 је приказана мрежа чије су гране означене стрелицама које указују на дозвољене правце кретања по путањи.



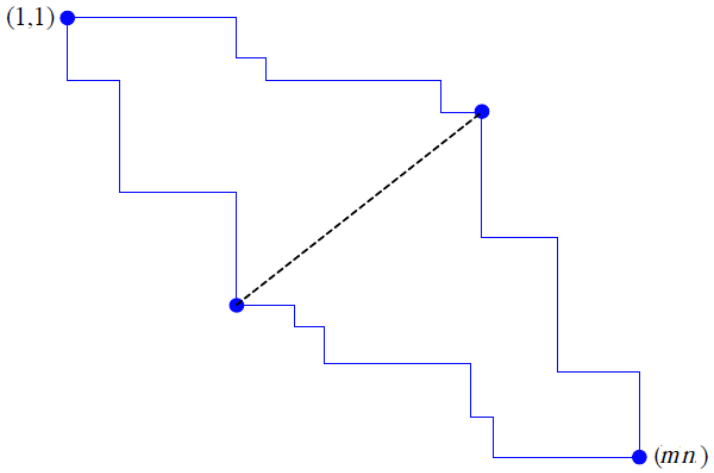
Слика 10-3. Мрежа за графовско одређивање *makespan*-а

Са Слике 10-3 се уочава да су усмерене гране мреже такве да, уколико путања пролази кроз чвор $p_{i,j}$, тада та путања не може да садржи чвор $p_{a,b}$ такав да је или $i < a \wedge b < j$ или $i > a \wedge b > j$. На Слици 10-4 сиви правоугаоници означавају недозвољене чворове у односу на чвор v_{ij}.



Слика 10-4. Забрањени чворови за чвор $v_{i,j}$

На основу дозвољених праваца путања, додатно може да се закључи да уколико су дате две путање, трећа путања не може да пређе из једне путање у другу уколико је нагиб прелазне путање позитиван. На Слици 10-5 је такав прелаз приказан испрекиданом линијом.



Слика 10-5. Недозвољени прелаз између путања

Закључак је да је потребно одредити све путање од $p_{1,1}$ до $p_{m,n}$ крећући се или на десно или на доле између ових чворова. Постоји више алгоритама који су засновани на овом приступу, од којих је најпознатији (Rad, Ruiz, & Boroojerdian, 2009) који користи методу критичне путање (*Critical Path Method*). Суштински, сви алгоритми овог типа одређују све дозвољене путање и дужину најдуже усвајају за коначно решење. За дати пример, најдужа путања је приказана на Слици 10-6.

1	2	3	4	5
3	1	1	3	3
3	3	3	3	2
2	1	4	2	4
5	1	1	3	1

1	2	3	4	5
3	4	5	8	11
6	9	12	15	17
8	10	16	18	22
13	14	17	21	23

Слика 10-6. Најдужа путања за дати пример

10.2.4 *Дуални поступак*

Taillard (Taillard, 1990) је искористио познату чињеницу да се *makespan* не мења уколико су редоследи послова и редоследи машина обрнути у односу на одговарајуће редоследе у оригиналном проблему. Другим речима, за сваку инстанцу, њена **дуална** инстанца има исту вредност *makespan*-а. Ово омогућује израчунавање *makespan*-а на начин, другачији од (22) - (27), увођењем **репа** (*tail*),

$Q(i,\pi(j)) \xrightarrow{\text{скраћено}} Q_{i,j}$:

$$Q_{m,n} = p_{m,n} \tag{29}$$

$$Q_{m,j} = Q_{m,j+1} + p_{m,j} \quad j = n-1,\ldots,1 \tag{30}$$

$$Q_{i,n} = Q_{i+1,n} + p_{i,n} \quad i = m-1,\ldots,1 \tag{31}$$

$$Q_{i,j} = \max\{Q_{i,j+1}, Q_{i+1,j}\} + p_{i,j} \quad i = m-1,\ldots,1;\ j = n-1,\ldots,1 \tag{32}$$

$C_{\max}[\pi] = Q(1,\pi(1))$ и одговарајућа матрица је $\boldsymbol{Q}[\pi] \xrightarrow{\text{скраћено}} \boldsymbol{Q}$.

Веза између три групе формула за одређивање *makespan*-а може да се уочи на Слици 10-7.

1	2	3	4	5
3	4	5	8	11
6	9	12	15	17
8	10	16	18	22
13	14	17	21	23

1	2	3	4	5
23	18	15	13	10
20	17	14	10	7
14	12	11	7	5
11	6	5	4	1

1	2	3	4	5	5
23	21	19	18	18	11
20	23	23	22	22	17
14	20	21	23	23	22
11	19	19	21	22	23

Слика 10-7. Матрице *C*, *Q* и *X* за дати пример

на Слици 10-7 су приказане матрице ***C***, ***Q*** и ***X***, где се елементи матрице ***X*** добијају сабирањем *i*-те колоне матрице ***C*** са (*i*+1)-ом колоном матрице ***Q***. Елементи матрице са максималним вредностима у ***X*** једнозначно одређују *k* вредности у једначини (28), при чему је та максимална вредност уједно и вредност *makespan*-а. Ове максималне вредности одређују путању од (1,1) до (*m*,*n*), која је на Слици 10-6 означена зеленом бојом. У случају да има више оваквих путања, све су једнозначно одређене ***X*** матрицом. Свака од ових путања дефинише структуру гант дијаграма која се односи, како на оригиналну, тако и на дуалну инстанцу.

X матрица има велики значај у дефинисању и програмирању Тајлардовог убразања за HEX алгоритам, о чему ће бити речи у следећем потпоглављу.

10.2.5 *PI / IP поступци*

Израчунавање *makespan*-а се у PFSP алгоритмима обавља за различите редоследе послова, да би се редослед који даје најмању вредност *makespan*-а усвојио као коначно решење. У раду (Petronijević & Ilić, 1994) се уводе две формулације проблема које инкорпорирају редослед послова у поступак израчунавања *makespan*-а. Прва формулација обухвата n^2 бинарних променљивих одлучивања, $x_{j,k},\ j=1,...,n\,;\,k=1,...,n$, које су дефинисане као:

$$x_{j,k}=\begin{cases}1, \textit{ ако се } j-\textit{ти посао извршава } k-\textit{ти по реду},\\ 0, \textit{ у противном}\end{cases}$$

Makespan се израчунава као: $C_{m,n}=\max\{f_{m,n-1},f_{m-1,n}\}+\sum_{k=1}^{n}p_{m,k}x_{n,k}$ где су:

$$f_{i,j}=\max\{f_{i,j-1},f_{i-1,j}\}+\sum_{k=1}^{n}p_{i,k}x_{j,k}\quad i=2,...,m;\ j=2,...,n$$

$$f_{1,j}=\sum_{k=1}^{j}\sum_{j=1}^{n}p_{1,j}x_{j,k}\quad j=1,...,n$$

$$f_{i,1}=\sum_{k=1}^{i}\sum_{j=1}^{n}p_{k,j}x_{1,j}\quad i=2,...,m$$

Овај модел је формулисан од стране Петронијевића и Илића као PROGRAM-PI који минимизира функцију $C_{m,n} = f_{m,n}(x_{1,1},...,x_{1,n},x_{2,1},...,x_{2,n},...,x_{n,1},...,x_{n,n})$ уз ограничење да бинарне променљиве $x_{j,k}$ одговарају елементима одговарајуће пермутационе матрице.

За дати пример из Табеле 5-1 је $f_{1,1} = \sum_{k=1}^{1}\sum_{j=1}^{n} p_{1,j}x_{j,k}$, $j = 1,...,n$, односно:

$$f_{1,1} = p_{1,1}x_{1,1} + p_{1,2}x_{2,1} + p_{1,3}x_{3,1} + p_{1,4}x_{4,1} + p_{1,5}x_{5,1}$$

$$f_{1,2} = f_{1,1} + p_{1,1}x_{1,2} + p_{1,2}x_{2,2} + p_{1,3}x_{3,2} + p_{1,4}x_{4,2} + p_{1,5}x_{5,2}$$

$$f_{1,3} = f_{1,2} + p_{1,1}x_{1,3} + p_{1,2}x_{2,3} + p_{1,3}x_{3,3} + p_{1,4}x_{4,3} + p_{1,5}x_{5,3}$$

$$f_{1,4} = f_{1,3} + p_{1,1}x_{1,4} + p_{1,2}x_{2,4} + p_{1,3}x_{3,4} + p_{1,4}x_{4,4} + p_{1,5}x_{5,4}$$

$$f_{1,5} = f_{1,4} + p_{1,1}x_{1,5} + p_{1,2}x_{2,5} + p_{1,3}x_{3,5} + p_{1,4}x_{4,5} + p_{1,5}x_{5,5}$$

У сваком реду је једна и само једна променљива $x_{j,k}$ различита од нуле. Претпоставимо, без губитка општости, да је:

$$x_{j,k} = \begin{cases} 1, & j = k \\ 0, & j \neq k \end{cases}$$

Тада је $f_{1,1} = p_{1,1}; f_{1,2} = p_{1,1} + p_{1,2}; f_{1,3} = p_{1,1} + p_{1,2} + p_{1,3}$....., што одговара вредностима у примерима претходних поступака. Аналогно се добијају и остале вредности за дати пример.

Друга формулација користи n целобројних променљивих одлучивања, $x_k, k = 1,...,n$, где x_k означава редни број посла који се налази на k-тој позицији. Одговарајуће функције f су:

$$f_{i,j} = \max\{f_{i,j-1}, f_{i-1,j}\} + p_{i,x_j} \quad i = 2,...,m; j = 2,...,n;$$

$$f_{1,j} = \sum_{k=1}^{j} p_{1,x_k} \quad j = 1,...,n;$$

$$f_{i,1} = \sum_{k=1}^{i} p_{x_k,1} \quad i = 2,...,m.$$

Овај модел је формулисан од стране Илића као PROGRAM-IP који минимизира функцију $C_{m,n} = f_{m,n}(x_1, x_2, ..., x_n)$ без ограничења.

10.3. HEX алгоритам за проблем редоследа послова у линији

Кључни продор у напорима за решавање пермутационог *flow shop* проблема је направљен објављивањем HEX хеуристике од аутора Nawaz, Enscore и Ham (Nawaz, Enscore Jr, & Ham, 1983), која је опште прихваћена као најбоља хеуристика за решавање $F|prmu|C_{max}$ (Fernandez-Vigas & Framinan, 2014), (Kalczynski & Kamburowski, 2007), (Ribas, Companys, & Tort-Martorell, 2010) и (Dong, Huang, & Chen, 2008). HEX хеуристика одређује почетне секвенце за све најбоље метахеуристике (Agraval, Colak, & Eryarsoy, 2006); (Ekskioglu, Ekskioglu, & Jain, 2008); (Grabowski & Wodecki, 2004); (Haq, Saravanan, Vivekraj, & Prasad, 2007); (Jarboui, Ibrahim, Siarry, & Rebai); (Kalczynski & Kamburowski, 2008); (Laha & Mandal, 2007); (Liao, Tseng, & Luarn, 2007); (Liu, Wang, & Jin, 2007); (Nowicki & Smutnicki, 1996); (Onwubolu & Davendra, 2006); (Pan, Tasgetiren, & Liang, 2007); (Ruiz & Stutzle, 2007); (Tasgetiren, Sevkl, & Gencyilmaz, 2007), (Ribas, Companys, & Tort-Martorell, 2010), (Fernandez-Vigas & Framinan, 2014), (Dong, Huang, & Chen, 2006), (Dong, Huang, & Chen, 2008) и (Dong, Chen, Huang, & Nowak, 2013).

Нека $T_j = \sum_{i=1}^{m} p_{i,j}$ означава укупно време извршења посла j. HEX хеуристика се састоји од два једноставна корака:

Корак 1: Уредити послове по нерастућим T_j у приоритетно уређење πp^n.

Set $\rho^1 = (\pi p^n(1))$ и $k = 1$;

Корак 2: While $k < n$ **do**: for $l = 1, ..., k+1$ **compute** $C_{\max}[\rho^k(\pi p^n(k+1), l)]$ и одреди прву подсеквенцу ρ^{k+1} са минималним $C_{\max}$. **Set** $k = k+1$ **Continue**.

На крају поступка, $\rho^n = \pi^n$ је коначна секвенца. Међусобни редослед бројева у секвенци ρ^{k+1}, која је одређена у *к*-тој итерацији, остаје исти кроз све даље итерације алгоритма.

НЕХ хеуристика је прототип ПКХ. У фази иницијализације НЕХ одређује приоритетни редослед уређењем послова према нерастућим вредностима њихових укупних времена рада. У фази уметања, први нераспоређени посао се умеће на најбољу позицију тренутно уређених послова. *Makespan* циљна функција за PFSP је посебно интересантна за истраживаче пошто је у раду (Taillard, 1990) предложено убрзање које омогућује да се комплексност фазе уметања редукује са $O(n^2m)$ на $O(nm)$. На тај начин, укупна комплексност НЕХ хеуристике се редукује на $O(n^2m)$.

Често, приликом извршавања НЕХ-а, у уређеним секвенцама могу постојати елементи који имају међусобно једнаке вредности, тј. уређење није стриктно једнозначно. Овакви случајеви се називају **равноправни** (*ties*) и могу да наступе: у приоритетном редоследу - послови са једнаким укупним временима рада; у фази уметања равноправни случајеви наступају у различитим подсеквенцама са једнаким најбољим парцијалним *makespan*-ом. У литератури је много више пажње фокусирано на овај други тип равноправних случајева и предлажу се тзв. **правила раскидања равноправности** (*tie-breaking rules*) приликом уметања. НЕХ приоритетно уређење је разматрао Framinan *и ост.* (Framinan, Leisten, Rajendran, & ., 2003) и показао да је оно супериорно у односу на свих 177 различитих испитиваних уређења. На основу ове анализе је објављен велики број радова у којима се сугерише неко ново приоритетно уређење и уводе различита правила раскидања равноправности у фази уметања. У свом познатом раду, (Kalczynski & Kamburowski, 2008) сугеришу боље приоритетно уређење у комбинацији са једноставним правилом раскидања равноправности којима поправљају укупне перформансе и гарантују оптималност за случај две машине. У раду (Dong, Huang, & Chen, 2006) је предложено почетно уређење засновано на средњој вредности и варијанси времена послова, у комбинацији са посебним механизмом за раскидање равноправности у фази уметања. Према (Kalczynski & Kamburowski, 2011),

механизам за раскидање равноправности, предложен у овом раду представља најбоље резултате, тестиране на Тајлардовим инстанцама. У раду (Fernandez-Vigas & Framinan, 2014) предлаже се нови механизам за раскидање равноправности заснован на предвиђању времена док машине не раде.

Сви објављени радови везани за НЕХ могу да се разврстају у две категорије: у првој се настоји да се поправи вредност циљне функције уз задржавање изузетне ефикасности оригиналне процедуре; радови из друге категорије предлажу метахеуристичка побољшања којима се поправљају почетни редоследи, добијени НЕХ алгоритмом уз што је могуће мањи утрошак додатног CPU времена.

Пре било какве анализе везане за НЕХ, неопходно је да се истакну четири најважније НЕХ предности, које је класификују у најефективнију хеуристику, без премца у одговарајућем савременом пољу истраживања:

1. НЕХ формулација је изузетно једноставна, што представља значајну предност за објективно експериментално тестирање;
2. изузетна ефикасност; на просечним рачунарима НЕХ захтева, за највеће Тајлардове инстанце, свега 12 стотих делова секунде. Истраживања су показала да је у реалним апликацијама PFSP динамички проблем, што указује да је временска ефикасност изузетно битан параметар;
3. CPU временска зависност НЕХ алгоритма директно зависи само од n и m; она не зависи од расподеле дужина рада послова на машинама. Ово омогућује да се CPU време рада НЕХ алгоритма усвоји као еталон за поређење са осталим алгоритмима;
4. у овом, изузетно кратком временском интервалу, НЕХ одређује редослед послова чија циљна функција у најгорем случају само 6% одступа од оптималних вредности на Тајлардовим инстанцама, док су на Вотсоновим инстанцама ова одступања знатно нижа. Треба напоменути да су на највећим Тајлардовим инстанцама ова одступања мања од 3%. Према томе, сви објављени радови везани за овај проблем уствари покушавају да направе у просеку 3% *makespan* побољшања, ризикујући да покваре три претходне кључне предности НЕХ процедуре.

Претходно побројане предности НЕХ хеуристике намећу стриктне критеријуме који треба да буду испуњени приликом покушаја да се поправе НЕХ резултати. Како су сва побољшања покушаји да се поправе *makespan* вредности, веома је важно да се аутори одреде према чињеници колико та побољшања кваре прве три предности НЕХ процедуре. Посебност овог проблема, да је опсег могућих побољшања веома мали, имплицира веома прецизну и објективну евалуацију. Нажалост, скоро сви објављени радови везани за ову проблематику садрже одређену субјективност и непрецизност:

- **неунифицирано кодирање НЕХ процедуре:** у различитим радовима, пријављена CPU времена за НЕХ процедуру на истим инстанцама могу да се разликују и за два реда величине на сличном хардверу. На пример, просечно CPU време у (Jin, Song, & Wu, 2007) на 500 x 20 Тајлард тест инстанцама је 13.21*sec*, док је одговарајуће време на истом хардверу за код који се користи у овом раду 12 стотих делова секунде. Овакве огромне разлике су последица неадекватног кодирања Тајлардовог убрзања, што као последицу има понављање примене лошег кода у свакој итерацији. Да би ствари биле горе, овако добијена CPU времена се користе за извођење закључака, па у истом раду време од 13.21*sec* аутори користе да би потврдили своје убрзање поступка. Као прилог смањењу ове некохерентности, у наставку је приказан прецизан код НЕХ алгоритма са Тајлардовим убрзањем. Овај код је једноставан, и према расположивим подацима резултује у најкраћем времену рада од свих познатих објављених резултата;
- **непрецизна дефиниција саме НЕХ процедуре:** приоритетно НЕХ уређење, π_p, подразумева нерастући низ укупних времена рада послова. У равноправним случајевима, π_p зависи од примењеног алгоритма за сортирање. Различити π_p обично резултују у различитим *makespan* вредностима, добијених НЕХ процедуром. Како се вредност *makespan*-а користи као основ за поређење са НЕХ поступком, непрецизност и необјективност су очигледне последице. Треба напоменути да неки новији радови усвајају π_p у коме су равноправни случајеви међусобно уређени

према растућим вредностима ознака одговарајућих послова. Ово ипак не доприноси даљој објективности, јер се најчешће референтне вредности, добијене на овај начин пореде са побољшањима која примењују неку другу процедуру за сортирање. Због овога је неопходно, када се у раду презентирају експериментални резултати, јасно навести тип сортирања, а поређења вршити са резултатима добијеним коришћењем исте процедуре сортирања. У дисертацији ће бити показано да су одступања *makespan* вредности добијених када се у НЕХ процедури примене различита сортирања у рангу најбољих НЕХ побољшања;

– **различите референтне вредности за исте тест инстанце:** за Тајлардове инстанце се свакодневно на јавним сајтовима ажурирају најбољи резултати. Различити аутори користе различите референтне вредности, које су или застареле или нетачне.

10.3.1 *Тајлардово убрзање*

Прво треба увести неке додатне ознаке и дефиниције у циљу једноставнијег приказивања Тајлардовог убрзања. Свакој операцији O, придодаје се величина **no**(O), број елементарних математичких операција, потребних да се та операција изврши. У зависности од рачунарског модела, постоје различите листе елементарних операција, нпр. у РАМ моделу, јединичне операције су сабирање, одузимање, поређење и померај, тј. множење и дељење са степеном броја два. У наставку се подразумева да су димензије матрица m x n и да је m димензија колона.

- $A_j := (A_j(1),\ldots, A_j(m))$ - колона j дате матрице $\boldsymbol{A}$; $\boldsymbol{A}$ се у том случају означава са $\boldsymbol{A} := (A_1,\ldots, A_m)$;
- **max**(A_j) - максимална вредност елемената у A_j; **no**(**max**(A_j)) = m-1 ;
- $A_j + B_k$, колона са збиром одговарајућих елемената у A_j и B_k; **no**(**max**($A_j + B_k$)) = m ;

- $\downarrow A_j$ - операција која формира нову колону R са елементима: $R(1) = A_j(1)$, $R(i) = R(i-1) + A_j(i) \quad i = 2,...,m$; $\mathbf{no}(\downarrow A_j) = m$;
- $\uparrow A_j$ - операција која формира нову колону R са елементима: $R(m) = A_j(m)$, $R(i) = R(i+1) + A_j(i) \quad i = m-1,...,1$; $\mathbf{no}(\uparrow A_j) = m$;
- $A_j \circ B_k$ - операција која формира нову колону R са елементима:

  $R(1) = A_j(1) + B_k(1), \quad R(i) = \max\{R(i-1), A_j(i)\} + B_k(i) \quad i = 2,...,m$; $\mathbf{no}(A_j \circ B_k) = 2m\text{-}1$;
- $A_j \bullet B_k$ - операција која формира нову колону R са елементима:

  $R(m) = A_j(m) + B_k(m), \quad R(i) = \max\{R(i+1), A_j(i)\} + B_k(i) \quad i = m-1,...,1$;

  $\mathbf{no}(A_j \bullet B_k) = 2m\text{-}1$;
- $\underset{indx}{\mathbf{min}}(k, x_1,..., x_n)$: нека низ ($x_1$,…, x_n) има l најмањих вредности; ако је $k < l$ тада $\underset{indx}{\mathbf{min}}(k, x_1,..., x_n)$ одређује позицију у (x_1,…, x_n) k-тог елемента који има најмању вредност, иначе, одређује позицију у (x_1,…, x_n) l-тог елемента који има минималну вредност; $\mathbf{no}(\underset{indx}{\mathbf{min}}(k, x_1,..., x_n)) = n\text{-}1$;
- $\underset{indx}{\mathbf{minL}}(k, x_1,..., x_n)$: нека низ ($x_1$,…, x_n) има l најмањих вредности; ако је $k < l$ тада $\underset{indx}{\mathbf{minL}}(k, x_1,..., x_n)$ одређује позицију у (x_1,…, x_n) (l-k+1)- тог елемента који има најмању вредност, иначе, одређује позицију у (x_1,…, x_n) првог елемента који има минималну вредност; $\mathbf{no}(\underset{indx}{\mathbf{minL}}(k, x_1,..., x_n)) = n\text{-}1$;
- **rspan**(***A***,***B***,k,j): m x j матрица ***R*** := (R_1,…, R_j) :

$$R = \begin{cases} ((\downarrow B_1), (R_1 \circ B_2),..., (R_{j-1} \circ B_j)) & , k = 1 \\ (A_1,..., A_{k-1}, (R_{k-1} \circ B_k), (R_k \circ B_{k+1}),..., (R_{j-1} \circ B_j)) & , k > 1 \end{cases}$$

  где је: $\mathbf{no}(\mathbf{rspan}(\boldsymbol{A},\boldsymbol{B},k,j)) = (j\text{-}k\text{+}1)(2m\text{-}1)$;
- **lspan**(***A***,***B***,k,j): m x j матрица ***R*** := (R_1,…, R_j) :

$$R = \begin{cases} ((R_2 \bullet B_1),...,(R_j \bullet B_{j-1}),(\uparrow B_j)) & ,k = j \\ ((R_2 \bullet B_1),...,(R_{k+1} \bullet B_k), A_k,..., A_{j-1}) & ,k < j \end{cases}$$

где је: **no**(**rspan**(***A***,***B***,*k*,*j*)) = *k*(2*m*-1) ;

Важно је да се напомене да је **no**(**rspan**(***A***,***B***,*k*,*j*) + **lspan**(***A***,***B***,*k*,*j*)) = (*j*+1)(2*m*-1), тј. не зависи од *k*.

- **ins** (π, j, k): умеће *j* на позицију *k* у π ;

  ins $(\pi, j, k) = (\pi(1),..., \pi(k-1), j, \pi(k),..., \pi(n))$; **no**(**ins** (π, j, k)) = 1.

Детаљни код HEX процедуре са Тајлардовим убрзањем, означен са HEXT је:

HEXT(*P* = *P*$[\pi_p]$, *n* , *m*)

1 $A \leftarrow (\downarrow P_1)$; $B \leftarrow (\uparrow P_1)$; $D \leftarrow A \circ P_2$; $F \leftarrow B \bullet P_2$

2 **if** (*D* (*m*) < *F* (1)) {***C*** ← (*A*, *D*) ; ***Q*** ← $((\uparrow P_2) \bullet P_1, (\uparrow P_2))$; $\pi \leftarrow (1,2)$ }

else {***C*** ← $((\downarrow P_2), (\downarrow P_2) \circ P_1)$; ***Q*** ← (*F*, *B*) ; $\pi \leftarrow (2,1)$ }

3 $j \leftarrow 3$

4 **repeat**

5 $A \leftarrow Q_1 \bullet P[\pi]_j$; $B \leftarrow C_{j-1} \circ P[\pi]_j$

6 $\boldsymbol{X} \leftarrow ((C_1 \circ P[\pi]_j + Q_2),..., (C_{j-2} \circ P[\pi]_j + Q_{j-1}))$

7 $k \leftarrow \min_{indx}(1, A(1), \max(X_1),..., \max(X_{j-2}), B(m))$

8 $\pi \leftarrow$ **ins**(π, j, k)

9 ***C***←**rspan**(***C***,***P***[π], *k*, *j*)) ; ***Q***← **lspan**(***C***,***P***[π], *k*, *j*))

10 $j \leftarrow j + 1$

until *j* = *n*

return π , *C*(*m*,*n*).

Подразумева се да је *n* > 2 и да је, без губитка општости, улазно ***P*** уређено према π_p, тако да је почетно π после извршења корака 2 (1,2) или (2,1). Циклус 4-10 иде од *j* = 3 до *j* = *n*. Позиција уметања новог посла је непосредно одређена у кораку 7 као позиција елемента са најмањом вредношћу у поређењу са *A*(1), затим

са максималном вредности у колонама од $\boldsymbol{X}$ и са $B(m)$. У кораку 8 се умеће j на позицију k у π. Током извршавања алгоритма, колоне од $\boldsymbol{P}$ су уређене према текућем π. Приказани код подразумева да се, коришћењем показивача низова, комплетно извршавање обавља без икаквог пребрисавања података у колонама и векторима. Захваљујући овоме, операција **ins** је елементарна аритметичка операција, док је време потребно да се приступи било ком елементу од $\boldsymbol{P}[\pi]$ исто као и време потребно да се приступи елементу оригиналне матрице $\boldsymbol{P}$.

Овај код омогућује две значајне предности. Прво, укупан број елементарних аритметичких операција може да се једноставно преброји и прикаже као егзактна функција од n и m. Ово затим успоставља зависност између утрошка CPU времена и димензија инстанце. Друга предност кода се односи на могућност модификације оригиналне процедуре. Како је избор позиције уметања у фази уметања формализован у једном јединому кораку, кораку 7, једноставним изменама овог корака могу да се промене правила уметања. Шта више, може да се омогући једноставно истовремено праћење више парцијалних секвенци.

10.4. Евалуација алгоритама за проблем редоследа послова у линији

Евалуација свих објављених процедура се спроводи поређењем са најбољим резултатима на двема групама тест инстанци, које су поставили Taillard (Taillard, 1993) (Taillard, 2004) и Watson *и ост.* (Watson, Whitley, & Howe, 2002). Тајлардове тест инстанце су у потпуности неструктуиране: времена рада послова се генеришу на случајан начин из униформне расподеле над бројевима 1,2,...,99. Одабран је подскуп ових инстанци на основу различитих критеријума, као што су нпр. тежина проблема, мерена варијансом квалитета решења кроз вишеструке примене *tabu search* алгоритма. На тај начин је одабран подскуп од 120 најтежих инстанци. Watson *и ост.* (Watson, Whitley, & Howe, 2002) су увели поступак за генерисање структуираних *flow-shop* инстанци, који су моделовани према примерима из неких стварних производних окружења. Већина објављених алгоритама даје, на структуираним тест инстанцама, или оптимална решења, или решења у блиској околини оптималних решења (прегледи у (Framinan, Gupta, & Leisten, 2004),

(Gupta & Stafford Jr, 2006), (Ruiz & Maroto, 2005), (Framinan, Leisten, & Ruiz-Usano, 2005) и (Hejazi & Saghafian, 2005)). Такође су, у скоро свим објављеним радовима, разлике у добијеним вредностима *makespan*-а на најтежим тест инстанцама, у опсегу мањем од 10%. С друге стране, измерена CPU времена у тим радовима значајно укључују субјективност индивидуалног програмирања и могу значајно да се разликују.

У дисертацији се користе Тајлардове тест инстанце, као најтеже, подразумевајући да ће алгоритам који добија добре резултате на овим инстанцама, добијати такође добре резултате и на осталим тест инстанцама. Треба посебно напоменути да одговарајући закључак за лоше резултате не мора да важи. Тајлард је проучавао инстанце за следећих 12 парова (*m*, *n*): (5, 20), (10, 20), (20, 20), (5, 50), (10, 50), (20, 50), (5, 100), (10, 100), (20, 100), (10, 200), (20, 200) и (20, 500) и, за сваки пар одабрао по Q = 10 најтежих инстанци. Табела 10-2 приказује доње и горње граничне вредности *makespan*-а (LB-UB, *lower bound-upper bound*) за свих 120 инстанци. Јасно је да је ако су доња и горња гранична вредност исте, та вредност је једнака оптималној вредности за ту инстанцу.

Евалуација алгоритама се врши на основу процентуалног увећања, $\Delta[\pi]$ у односу на горњу граничну вредност из табеле, $C_{\max}[\pi^*]$:

$$\Delta[\pi] = 100\frac{C_{\max}[\pi] - C_{\max}[\pi^*]}{C_{\max}[\pi^*]}\ \%\,, \tag{33}$$

где π представља секвенцу послова добијену тестираним алгоритмом. Ако са $impr(A_1, A_2)$ означимо релативно побољшање алгоритма A_1 у односу на алгоритам A_2, тада је:

$$impr(A_1, A_2) = 100\frac{\Delta^{A_2} - \Delta^{A_1}}{\Delta^{A_2}}\ \%, \tag{34}$$

где Δ^A означава Δ за алгоритам *A*. На основу једначине (33):

$$impr(A_1, A_2) = 100\frac{C_{\max}[\pi^{A_2}] - C_{\max}[\pi^{A_1}]}{C_{\max}[\pi^{A_2}] - C_{\max}[\pi^*]}\ \%,$$

где π^A означава секвенцу добијену алгоритмом *A*.

Табела 10-2. Тајлардове инстанце

20 x 5	LB-UB	**20 x 10**	LB-UB	**20 x 20**	LB-UB	**50 x 5**	LB-UB
ta 1	1278-1278	ta 11	1582-1582	ta 21	2297-2297	ta 31	2724-2724
ta 2	1359-1359	ta 12	1659-1659	ta 22	2099-2099	ta 32	2834-2834
ta 3	1081-1081	ta 13	1496-1496	ta 23	2326-2326	ta 33	2621-2621
ta 4	1293-1293	ta 14	1377-1377	ta 24	2223-2223	ta 34	2751-2751
ta 5	1235-1235	ta 15	1419-1419	ta 25	2291-2291	ta 35	2863-2863
ta 6	1195-1195	ta 16	1397-1397	ta 26	2226-2226	ta 36	2829-2829
ta 7	1234-1234	ta 17	1484-1484	ta 27	2273-2273	ta 37	2725-2725
ta 8	1206-1206	ta 18	1538-1538	ta 28	2200-2200	ta 38	2683-2683
ta 9	1230-1230	ta 19	1593-1593	ta 29	2237-2237	ta 39	2552-2552
ta 10	1108-1108	ta 20	1591-1591	ta 30	2178-2178	ta 40	2782-2782

50 x 10	LB-UB	**50 x 20**	LB-UB	**100 x 5**	LB-UB	**100 x 10**	LB-UB
ta 41	2991-2991	ta 51	3771-3850	ta 61	5493-5493	ta 71	5770-5770
ta 42	2867-2867	ta 52	3668-3704	ta 62	5268-5268	ta 72	5349-5349
ta 43	2839-2839	ta 53	3591-3640	ta 63	5175-5175	ta 73	5676-5676
ta 44	3063-3063	ta 54	3635-3723	ta 64	5014-5014	ta 74	5781-5781
ta 45	2976-2976	ta 55	3553-3611	ta 65	5250-5250	ta 75	5467-5467
ta 46	3006-3006	ta 56	3667-3681	ta 66	5135-5135	ta 76	5303-5303
ta 47	3093-3093	ta 57	3672-3704	ta 67	5246-5246	ta 77	5595-5595
ta 48	3037-3037	ta 58	3627-3691	ta 68	5094-5094	ta 78	5617-5617
ta 49	2897-2897	ta 59	3645-3743	ta 69	5448-5448	ta 79	5871-5871
ta 50	3065-3065	ta 60	3696-3756	ta 70	5322-5322	ta 80	5845-5845

100 x 20	LB-UB	**200 x 10**	LB-UB	**200 x 20**	LB-UB	**500 x 20**	LB-UB
ta 81	6106-6202	ta 91	10862-10862	ta 101	11152-11195	ta 111	26040-26059
ta 82	6183-6183	ta 92	10480-10480	ta 102	11143-11203	ta 112	26500-26520
ta 83	6252-6271	ta 93	10922-10922	ta 103	11281-11281	ta 113	26371-26371
ta 84	6254-6269	ta 94	10889-10889	ta 104	11275-11275	ta 114	26456-26456
ta 85	6262-6314	ta 95	10524-10524	ta 105	11259-11259	ta 115	26334-26334
ta 86	6302-6364	ta 96	10329-10329	ta 106	11176-11176	ta 116	26469-26477
ta 87	6184-6268	ta 97	10854-10854	ta 107	11337-11360	ta 117	26389-26389
ta 88	6315-6401	ta 98	10730-10730	ta 108	11301-11334	ta 118	26560-26560
ta 89	6204-6275	ta 99	10438-10438	ta 109	11145-11192	ta 119	26005-26005
ta 90	6404-6434	ta 100	10675-10675	ta 110	11284-11288	ta 120	26457-26457

10.5. Равноправне ситуације

Дефиниција НЕХ хеуристике није једнозначна у случајевима равноправних ситуација у почетном уређењу. У неким новијим радовима је усвојено π_p у коме су равноправни послови међусобно уређени према растућим вредностима својих ознака. У свим објављеним и будућим радовима који се баве побољшањима НЕХ хеуристике је важно да аутори прецизно дефинишу уређење секвенце π_p, или да наведу тип примењеног поступка за сортирање. У овом делу дисертације је разматран утицај на квалитет решења типа сортирања који је примењен за уређење секвенце π_p.

Спроведена је експериментална анализа са циљем да се дефинишу опсези вредности циљне функције за различите поступке сортирања секвенце π_p. У случајевима када постоји више равноправних послова испитују се све могуће пермутације у оквиру целе секвенце. Како је на неким Тајлардовим тест инстанцама укупан број пермутација које треба обрадити на овај начин изузетно велики, уведено је ограничење за максималан број овако обрађиваних пермутација. Уведен је праг од 4700 пермутација; испод овог прага се обрађују све секвенце, изнад овог прага се примењује претрага променљивих околина (VNS). VNS је подешен на максимално време од 60*sec*. Експериментална студија је проширена за још један скуп резултата који квантификују утицај равноправних ситуација у фази уметања. Конкретно, поред резултата за оригиналну НЕХ хеуристику, приложени су резултати који се добијају на дуалним инстанцама проблема, чиме је уствари експериментално квантификован утицај примене *DESC* уређења на квалитет решења. Ови резултати су у табелама означени са НЕХЛ. Табела 10-3 приказује детаљне резултате за свих 120 Тајлардових инстанци. Резултати нису усредњивани по групама инстанци пошто појединачне вредности могу да послуже као референца за будуће радове.

Табела 10-3. Опсези вредности *makespan*-а

	HEX		HEXЛ									
	min	*max*	*min*	*max*	*Cmpl*	*Gr.*	*Cnt*	*Rng*	*RngL*	*RngO*	Δ	*t*
20 x 5												
ta 1	1286	1286	1299	1299	*T*	0	1	0.00	0.00	1.01	0.63	0
ta 2	1365	1365	1365	1365	*T*	3	8	0.00	0.00	0.00	0.44	0
ta 3	1140	1159	1132	1132	*T*	1	2	1.67	0.00	2.39	4.72	0
ta 4	1325	1340	1329	1329	*T*	1	2	1.13	0.00	1.13	2.47	0
ta 5	1305	1305	1305	1305	*T*	0	1	0.00	0.00	0.00	5.67	0
ta 6	1228	1228	1251	1251	*T*	0	1	0.00	0.00	1.87	2.76	0
ta 7	1278	1279	1251	1251	*T*	1	2	0.08	0.00	2.24	1.38	0
ta 8	1223	1235	1215	1227	*T*	2	4	0.98	0.99	1.65	0.75	0
ta 9	1291	1291	1284	1284	*T*	0	1	0.00	0.00	0.55	4.39	0
ta 10	1151	1151	1127	1127	*T*	0	1	0.00	0.00	2.13	1.71	0
Av.								0.39	0.10	1.30	2.49	0
20 x 10												
ta 11	1680	1680	1681	1681	*T*	0	1	0.00	0.00	0.06	6.19	0
ta 12	1729	1786	1747	1766	*T*	2	4	3.30	1.09	3.30	4.22	0
ta 13	1557	1557	1562	1562	*T*	0	1	0.00	0.00	0.32	4.08	0
ta 14	1439	1450	1416	1428	*T*	1	2	0.76	0.85	2.40	2.83	0
ta 15	1502	1502	1502	1502	*T*	0	1	0.00	0.00	0.00	5.85	0
ta 16	1453	1453	1456	1456	*T*	0	1	0.00	0.00	0.21	4.01	0
ta 17	1562	1562	1531	1531	*T*	0	1	0.00	0.00	2.02	3.17	0
ta 18	1609	1609	1626	1626	*T*	0	1	0.00	0.00	1.06	4.62	0
ta 19	1647	1647	1639	1639	*T*	0	1	0.00	0.00	0.49	2.89	0
ta 20	1653	1653	1656	1656	*T*	1	6	0.00	0.00	0.18	3.90	0
Av.								0.41	0.19	1.00	4.17	0
20 x 20												
ta 21	2410	2410	2443	2443	*T*	0	1	0.00	0.00	1.37	4.92	0
ta 22	2150	2150	2134	2134	*T*	0	1	0.00	0.00	0.75	1.67	0
ta 23	2411	2429	2414	2432	*T*	1	2	0.75	0.75	0.87	3.65	0
ta 24	2262	2262	2257	2257	*T*	0	1	0.00	0.00	0.22	1.53	0
ta 25	2397	2397	2370	2370	*T*	0	1	0.00	0.00	1.14	3.45	0
ta 26	2349	2349	2349	2349	*T*	0	1	0.00	0.00	0.00	5.53	0
ta 27	2362	2362	2383	2383	*T*	1	2	0.00	0.00	0.89	3.92	0
ta 28	2249	2249	2249	2249	*T*	0	1	0.00	0.00	0.00	2.23	0
ta 29	2306	2320	2306	2306	*T*	2	4	0.61	0.00	0.61	3.08	0
ta 30	2277	2277	2257	2257	*T*	1	2	0.00	0.00	0.89	3.63	0
Av.								0.14	0.07	0.67	3.36	0
50 x 5												
ta 31	2733	2733	2729	2743	*T*	4	16	0.00	0.51	0.51	0.18	0
ta 32	2843	2882	2882	2882	*T*	4	16	1.37	0.00	1.37	0.32	0
ta 33	2625	2643	2650	2650	*T*	4	16	0.69	0.00	0.95	0.15	0
ta 34	2782	2810	2782	2785	*T*	7	384	1.01	0.11	1.01	1.13	120
ta 35	2868	2868	2868	2876	*T*	4	16	0.00	0.28	0.28	0.17	3
ta 36	2835	2853	2835	2850	*T*	5	96	0.63	0.53	0.63	0.21	29
ta 37	2732	2782	2736	2850	*T*	8	2304	1.83	4.17	4.32	0.26	743
ta 38	2690	2721	2700	2700	*T*	6	64	1.15	0.00	1.15	0.26	20
ta 39	2571	2576	2567	2606	*T*	6	64	0.19	1.52	1.52	0.59	20
ta 40	2786	2822	2786	2801	*T*	6	64	1.29	0.54	1.29	0.14	20
Av.								0.82	0.77	1.30	0.34	95
50 x 10												
ta 41	3135	3168	3125	3184	*T*	4	16	1.05	1.89	1.89	4.48	9
ta 42	3021	3032	3026	3073	*T*	2	4	0.36	1.55	1.72	5.37	0
ta 43	2952	3042	2958	3068	*T*	5	32	3.05	3.72	3.93	3.98	20
ta 44	3183	3198	3153	3218	*T*	2	4	0.47	2.06	2.06	2.94	0
ta 45	3128	3188	3145	3186	*T*	4	16	1.92	1.30	1.92	5.11	9
ta 46	3158	3178	3136	3148	*T*	2	4	0.63	0.38	1.34	4.32	0
ta 47	3277	3277	3277	3277	*T*	4	48	0.00	0.00	0.00	5.95	31
ta 48	3123	3193	3145	3172	*T*	2	4	2.24	0.86	2.24	2.83	0
ta 49	3002	3015	3025	3033	*T*	4	16	0.43	0.26	1.03	3.62	9
ta 50	3257	3272	3236	3269	*T*	3	8	0.46	1.02	1.11	5.58	3
Av.								1.06	1.31	1.72	4.42	8

50 x 20												
ta 51	4013	4082	4006	4098	*T*	3	8	1.72	2.30	2.30	4.05	9
ta 52	3921	3921	3958	3958	*T*	0	1	0.00	0.00	0.94	5.86	0
ta 53	3890	3927	3861	3887	*T*	3	8	0.95	0.67	1.71	6.07	9
ta 54	3926	3987	3907	3953	*T*	4	16	1.55	1.18	2.05	4.94	20
ta 55	3822	3835	3825	3874	*T*	2	4	0.34	1.28	1.36	5.84	3
ta 56	3914	3940	3857	3871	*T*	2	4	0.66	0.36	2.15	4.78	3
ta 57	3952	3952	3927	3927	*T*	1	2	0.00	0.00	0.64	6.02	0
ta 58	3916	3974	3910	3968	*T*	6	64	1.48	1.48	1.64	5.93	80
ta 59	3952	3952	3970	3970	*T*	0	1	0.00	0.00	0.46	5.58	0
ta 60	4016	4079	4036	4036	*T*	1	2	1.57	0.00	1.57	6.92	0
Av.								0.83	0.73	1.48	5.60	12
100 x 5												
ta 61	5514	5524	5514	5514	*F*	7	4608	0.18	0.00	0.18	0.38	6060
ta 62	5284	5348	5284	5300	*F*	9	1536	1.21	0.30	1.21	0.30	2069
ta 63	5204	5246	5206	5246	*F*	10	1024	0.81	0.77	0.81	0.56	1366
ta 64	5023	5037	5023	5023	*F*	9	1536	0.28	0.00	0.28	0.18	2083
ta 65	5261	5277	5255	5261	*F*	12	4096	0.30	0.11	0.42	0.10	5234
ta 66	5139	5141	5139	5162	*F*	9	512	0.04	0.45	0.45	0.08	691
ta 67	5257	5280	5277	5297	*F*	7	384	0.44	0.38	0.76	0.21	514
ta 68	5105	5132	5096	5148	*T*	9	512	0.53	1.02	1.02	0.04	657
ta 69	5487	5517	5489	5489	*F*	10	3072	0.55	0.00	0.55	0.72	3960
ta 70	5322	5354	5322	5355	*F*	10	3072	0.60	0.62	0.62	0.00	4109
Av.								0.49	0.37	0.63	0.26	2674
100 x 10												
ta 71	5797	5967	5823	6001	*F*	12	4096	2.93	3.06	3.52	0.47	10917
ta 72	5397	5552	5395	5522	*F*	7	1152	2.87	2.35	2.91	0.86	2963
ta 73	5710	5856	5693	5831	*T*	7	384	2.56	2.42	2.86	0.30	1020
ta 74	5890	6103	5875	6111	*F*	9	4608	3.62	4.02	4.02	1.63	12031
ta 75	5577	5771	5603	5736	*F*	10	3072	3.48	2.37	3.48	2.01	7980
ta 76	5344	5466	5334	5465	*F*	7	3456	2.28	2.46	2.47	0.58	9117
ta 77	5664	5769	5678	5776	*F*	9	4608	1.85	1.73	1.98	1.23	12506
ta 78	5694	5821	5694	5816	*T*	12	4096	2.23	2.14	2.23	1.37	10766
ta 79	5961	6052	5968	6077	*F*	9	4608	1.53	1.83	1.95	1.53	12551
ta 80	5903	5994	5900	6041	*F*	10	3072	1.54	2.39	2.39	0.94	8140
Av.								2.49	2.48	2.78	1.09	8799
100 x 20												
ta 81	6531	6684	6468	6682	*T*	11	2048	2.34	3.31	3.34	4.29	10931
ta 82	6446	6565	6489	6550	*T*	4	16	1.85	0.94	1.85	4.25	80
ta 83	6507	6691	6559	6721	*T*	9	512	2.83	2.47	3.29	3.76	2683
ta 84	6516	6696	6492	6642	*F*	10	3072	2.76	2.31	3.14	3.56	16266
ta 85	6612	6738	6605	6753	*T*	10	1024	1.91	2.24	2.24	4.61	5357
ta 86	6631	6833	6631	6837	*T*	9	512	3.05	3.11	3.11	4.20	2691
ta 87	6573	6715	6557	6652	*T*	4	48	2.16	1.45	2.41	4.61	251
ta 88	6736	6884	6682	6901	*T*	10	3072	2.20	3.28	3.28	4.39	16363
ta 89	6582	6726	6604	6746	*T*	6	192	2.19	2.15	2.49	4.89	989
ta 90	6662	6750	6635	6725	*T*	5	32	1.32	1.36	1.73	3.12	169
Av.								2.26	2.26	2.69	4.17	5578
200 x 10												
ta 91	10916	11013	10919	11072	*F*	7	3456	0.89	1.40	1.43	0.50	37609
ta 92	10591	10777	10594	10772	*F*	8	2304	1.76	1.68	1.76	1.06	23883
ta 93	11025	11123	11025	11169	*F*	10	3072	0.89	1.31	1.31	0.94	32931
ta 94	11057	11057	11057	11057	*F*	9	4608	0.00	0.00	0.00	1.54	50309
ta 95	10575	10721	10575	10789	*F*	10	3072	1.38	2.02	2.02	0.48	32160
ta 96	10407	10577	10390	10550	*F*	8	3072	1.63	1.54	1.80	0.59	32280
ta 97	10917	11051	10924	11056	*F*	10	1024	1.23	1.21	1.27	0.58	10637
ta 98	10793	10903	10804	10921	*F*	11	2048	1.02	1.08	1.19	0.59	21569
ta 99	10538	10716	10522	10635	*F*	7	384	1.69	1.07	1.84	0.80	3906
ta 100	10758	10884	10768	10855	*F*	9	512	1.17	0.81	1.17	0.78	5409
Av.								1.17	1.21	1.38	0.79	25069

200 x 20												
ta 101	11503	11754	11511	11757	*F*	9	2735	2.18	2.14	2.21	2.75	60163
ta 102	11615	11844	11638	11879	*F*	8	768	1.97	2.07	2.27	3.68	15246
ta 103	11685	12045	11681	11956	*F*	9	2748	3.08	2.35	3.12	3.55	60449
ta 104	11631	11879	11656	11920	*F*	8	2720	2.13	2.26	2.48	3.16	59774
ta 105	11575	11783	11573	11813	*F*	9	1536	1.80	2.07	2.07	2.79	29991
ta 106	11567	11821	11580	11852	*F*	7	2734	2.20	2.35	2.46	3.50	60157
ta 107	11701	11967	11664	11995	*F*	10	2738	2.27	2.84	2.84	2.68	60243
ta 108	11717	11939	11745	11924	*F*	10	1024	1.89	1.52	1.89	3.38	20069
ta 109	11541	11818	11547	11849	*F*	7	2734	2.40	2.62	2.67	3.12	60157
ta 110	11693	11959	11670	11941	*F*	12	2732	2.27	2.32	2.48	3.38	60120
Av.								2.22	2.25	2.45	3.20	48637
500 x 20												
ta 111	26614	26880	26604	26878	*F*	7	384	1.00	1.03	1.04	2.09	46371
ta 112	27116	27347	27084	27339	*F*	8	256	0.85	0.94	0.97	2.13	30643
ta 113	26799	27108	26793	27074	*F*	10	498	1.15	1.05	1.18	1.60	60286
ta 114	26822	27220	26882	27222	*F*	4	503	1.48	1.26	1.49	1.38	60429
ta 115	26707	26960	26682	27031	*F*	9	500	0.95	1.31	1.31	1.32	60054
ta 116	26904	27175	26906	27225	*F*	10	501	1.01	1.19	1.19	1.61	60071
ta 117	26658	26933	26700	26961	*F*	7	500	1.03	0.98	1.14	1.02	60043
ta 118	27052	27287	27081	27260	*F*	7	384	0.87	0.66	0.87	1.85	47351
ta 119	26491	26846	26457	26740	*F*	9	501	1.34	1.07	1.47	1.74	60083
ta 120	26851	27118	26900	27189	*F*	10	501	0.99	1.07	1.26	1.49	60114
Av.			1.46	2.25				1.07	1.06	1.19	1.62	54545

Колона *Cmpl* садржи *T* уколико су обрађене све пермутације; у колону *Gr* је уписан број група равноправних секвенци, а у колону *Cnt* број тестираних секвенци. У колонама *Rne, RngL* и *RngO* су уписаниа релативна повећања $C_{\max}[\pi \max]$ у односу на $C_{\max}[\pi \min]$ за НЕХ, НЕХЛ и укупно повећање и за НЕХ и за НЕХЛ. Колона Δ садржи $\Delta[\pi]$ из (33) као релативно повећање добијене доње границе у односу на најбољи познати резултат. Утрошена времена рада процесора су приказана у колони *t.* Са Av. су означене просечне вредности у колонама. Подаци у *Cmpl, Gr* и *t* представљају резултате који се односе на НЕХ, пошто се одговарајући резултати за НЕХЛ од њих незнатно разликују.

Приказани резултати недвосмислено истичу значај информације која се односи на тип сортирања у поступку одређивања почетног уређења. И поред уведеног ограничења за број пермутација тоталне претраге и ограничење рада VNS-а, добијене су значајне промене вредности *makespan*-а у односу на оригинални НЕХ алгоритам. Закључак је више него охрабрујући: у већини случајева је добијени опсег решења на нивоу побољшања која су у литератури добијена применом много сложенијих алгоритама. Ови закључци ће бити саставни део резултата који су добијени у наставку овог поглавља и који су искоришћени за конструкцију

новог полиномијалног алгоритма који надмашује сва позната побољшања НЕХ хеуристике.

10.6. GCA за проблем редоследа послова у линији

Проблем редоследа послова у линији је одабран да би се на њему испитао утицај избора аргумената и вредности тих аргумената на перформансе GCA за овај проблем. Циљ истраживања је дефинисање алгоритма којим се добијају ниже вредности *makespan*-а у односу на НЕХ хеуристику уз очување временске сложености и једноставности НЕХ поступка. НЕХ је одабран за поређење пошто његова изузетна ефикасност даје посебан кредибилитет добијеним побољшањима.

За израчунавање πp^n, коришћен је познати *insertion sort method*. $C_{\max}$ се израчунава коришћењем Тајлардовог убрзања (Taillard, 1990). Као јединица мере за одређивање временске ефикасности је коришћено НЕХ CPU време. На овај начин је омогућено поређење које не зависи од субјективности кодирања и примењеног хардвера. Према томе, за објективно поређење није потребан детаљан опис кода за израчунавање вредности $C_{\max}$. Конкретно, НЕХ алгоритам је једноставан; сложеност O (n^2m) подразумева да је број елементарних операција, $EO = k \cdot n^2m + o$, где је o број припремних операција изван циклуса. Оно што је најважније је да k не зависи од осталих улазних параметара (расподеле времена послова). За било које исправно кодирање НЕХ алгоритма, o може да се занемари (за код примењен у овом раду је у случајевима када је $n^2m > 200$, степен поклапања са кривом $EO = k \cdot n^2m$ већи од 97%; инстанце код којих је $n^2m < 200$ могу да се обрађују тоталном претрагом). На основу овога, време обраде НЕХ алгоритмом, означено са $T^{n,m}$, је узето за јединицу мере приликом одређивања временске сложености алгоритама заснованих на предложеном приступу. Ако је, за било коју (n_1, m_1) инстанцу време T^{n_1,m_1} познато, тада може да се израчуна T^{n_2,m_2} за било коју другу, (n_2, m_2) инстанцу, обрађивану на истом хардверу:

$$T^{n_2,m_2} = \frac{n_2^2 m_2}{n_1^2 m_1} T^{n_1,m_1}. \qquad (35)$$

Ово потпоглавље је подељено у три одељка. У првом одељку су приказана три једноставна НЕХ побољшања. У другом одељку је испитан утицај вредности аргумената на квалитет решења. Најзад, на основу ових испитивања предложена је нова хеуристика за PFSP. У циљу истицања разлике нових алгоритама у односу на НЕХ, у листи аргумената су приказани само они аргументи чије се вредности разликују од НЕХ аргумената.

10.6.1 *Три побољшања НЕХ алгоритма*

Три побољшања НЕХ хеуристике се постижу само променом вредности *ord*, $\boldsymbol{arg_2}$, $\boldsymbol{arg_3}$ и $\boldsymbol{arg_5}$. Коришћено време CPU, означено као $t^{n,m}$, пореди се са $T^{n,m}$. Табела 10-4 приказује поређење НЕХ и GCA (*ord* = *DSC*). Може се уочити да GCA (*DSC*) резултати надмашују НЕХ резултате у 50% тестираних инстанци. Време CPU коришћено за GCA (*ord* = *DSC*) је исто као и време CPU коришћено за НЕХ на истој инстанци: $t^{n,m} = T^{n,m}$.

Табела 10-4. Поређење НЕХ и GCA на Тајлардовим инстанцама

m	5	10	20	5	10	20	5	10	20	10	20	20	
n	20	20	20	50	50	50	100	100	100	200	200	500	*Укупно*
НЕХ бољи	3	5	3	3	5	6	4	4	6	4	1	3	47
Равноправно	2	1	3	2	1	0	2	0	0	1	1	0	13
GCA бољи	5	4	4	5	4	4	4	6	4	5	8	7	60
Укупно	10	10	10	10	10	10	10	10	10	10	10	10	120

Објашњење ових резултата је очигледно. Показано је, (Pinedo, 2010), да се $C_{\max}$ не мења ако послови обилазе проточну радионицу у супротном правцу и по обрнутом редоследу машина. Према томе, може се извући врло једноставан закључак да је резултат за GCA (*DSC*) исти као и резултат добијен када се НЕХ примени на тако дефинисану обрнуту инстанцу. Ово омогућује једноставан доказ да је вероватноћа да GCA (*DSC*) надмашује НЕХ једнака вероватноћи да НЕХ надмашује GCA (*DSC*). Значи, примене НЕХ и GCA (*DSC*) могу да резултују у различитим коначним секвенцама увек када има више равноправних кандидата за уређење. Конкретно, НЕХ бира прву, а GCA (*DSC*) последњу од равноправних опција.

Табела 10-5 представља поређење NEH и GCA(***arg*₃**=1). Поново, модификовани NEH надмашује NEH резултате у скоро 50% тестираних инстанци. За овај пример је $t^{n,m} > T^{n,m}$. приликом израчунавања $C_{\max}(\pi^k, k+1,...,n)$, те GCA(***arg*₃** = 1) захтева дуже CPU време (у спроведеним експерименталним резултатима, горња граница је била $t^{n,m} < 2T^{n,m}$ за све тестиране инстанце, али детаљан опис примењеног кода није предмет ове дисертације). Треба да се напомене да је поменути недостатак у примени (***arg*₃** = 1) специфичан за PFSP због Тајлардове редукције са $O(n^3m)$ на $O(n^2m)$.

Табела 10-5. Поређење HEX и GCA ($arg_3 = 1$)

m	5	10	20	5	10	20	5	10	20	10	20	20	
n	20	20	20	50	50	50	100	100	100	200	200	500	*Укупно*
HEX бољи	1	0	2	3	0	2	2	3	5	5	6	4	33
Равноправно	7	6	7	3	3	0	5	0	0	3	0	0	34
GCA бољи	2	4	1	4	7	8	3	7	5	2	4	6	53
Укупно	10	10	10	10	10	10	10	10	10	10	10	10	120

Коначно, Табела 10-6 приказује поређење HEX и GCA(***arg*₂**=15, ***arg*₅**=1). Може се уочити овим поређењем да HEX никада није бољи од GCA(***arg*₂**=15, ***arg*₅**=1). Овај закључак је очекиван, јер је прва од праћених секвенци иста као HEX секвенца. Наиме, ***arg*₅**=1 претпоставља да се прво бирају ***arg*₂** секвенце и HEX бира прву од тих секвенци. GCA(***arg*₂**=15, ***arg*₅**=1) надмашује HEX у 93% тестираних инстанци. $t^{n,m} = 15T^{n,m}$, што представља веома ефикасно побољшање.

Табела 10-6. Поређење HEX и GCA(arg_2=15, arg_5=1)

m	5	10	20	5	10	20	5	10	20	10	20	20	
n	20	20	20	50	50	50	100	100	100	200	200	500	*Укупно*
HEX бољи	0	0	0	0	0	0	0	0	0	0	0	0	0
Равноправно	2	0	0	2	0	0	3	0	0	1	0	0	8
GCA бољи	8	10	10	8	10	10	7	10	10	9	10	10	112
Укупно	10	10	10	10	10	10	10	10	10	10	10	10	120

Приказани резултати дају могућност за примену MULTIGCA са HEX и презентованим алгоритмом. На пример, MULTIGCA(HEX, GCA(*ord = DSC*)), обезбеђују могућност од 50% за добијање бољих резултата од NEH за $t^{n,m} = 2T^{n,m}$, тј. за линеарно повећање потрошеног CPU времена.

10.6.2 *Утицај вредности аргумената на квалитет решења*

Циљ истраживања у овом поглављу није поређење добијених резултата са резултатима из литературе, већ да се демонстрира један нов начин истраживања: проучавање предности и ограничења вишеструке примене HEX алгоритма са различитим вредностима аргумената. Закључци који су изведени из овог истраживања су искоришћени за креирање побољшаног HEX алгоритма, приказаног у потпоглављу 7.3.

Прво је постављена горња граница времена извршавања програма, у функцији од n и m. У већини метахеуристика за PFSP, време извршења се поставља на $60n(m/2)$ у милисекундама. За $n = 200$ и $m = 20$ ово време је $120sec$. Објављене вредности за $T^{200,20}$ у познатом, референтном раду (Rad, Ruiz, & Boroojerdian, 2009) је $0.0127sec$ (ова вредност је узета као референтна, пошто вредности за $T^{n,m}$, објављене у овом раду најприближније прате функцију $O(n^2m)$, у односу на друге објављене резултате). На основу овога, за $120sec$ HEX може да се примени 9,448 пута на инстанцу ($n = 200$, $m = 20$). За све групе Тајлардових тест инстанци, одговарајући број примене HEX алгоритма варира од 3,779 (на инстанци $n = 500$, $m = 20$) до 94,488 (на инстанци $n = 20$, $m = 20$). Конкретно, за инстанцу ($n = 20$, $m = 20$) се број примена добија на следећи начин:

$$60n(m/2)=12sec;\ T^{20,20}=\frac{20^2\cdot 20}{200^2\cdot 20}T^{200,20}=\frac{1}{100}T^{200,20}=1.27\cdot 10^{-4}\,\text{sec};\ \frac{12}{1.27\cdot 10^{-4}}=94{,}488$$

Истраживања у овом поглављу су фокусирана на побољшања *makespan*-а која могу да се добију са ограничењем $t^{n,m} < 200T^{n,m}$ (0.2% - 5% у односу на претходне бројеве).

Као следеће, одабране су тест инстанце које су прихваћене у скоро свим радовима које се баве са PFSP. У литератури су тестови најчешће спроведени на Тајлардовим тест инстанцама, које се сматрају најтежим од расположивих тест инстанци. У раду (Watson, Whitley, & Howe, 2002) је закључено да супериорни резултати на Тајлардовим инстанцама не подразумевају увек супериорне резулате на проблемима из реалног живота. Предност GCA лежи у способности да се

вредности аргумената могу да прилагоде специфичним особинама обрађиваних инстанци. Ипак, тестирања у овом поглављу су спроведена на Тајлардовим инстанцама пошто су сви релевантни алгоритми такође тестирани на овим инстанцама.

Следећи корак је избор типа рангирања, R. Један од логичких избора је R_1, којим се селектују равноправне секвенце. Избором ове опције није потребно сортирати секвенце, те је $t^{n,m} \leq \boldsymbol{arg}_2 \cdot T^{n,m}$. У случајевима када је број равноправних секвенци већи од $\boldsymbol{arg_2}$, бира се првих $\boldsymbol{arg_2}$ равноправних секвенци. Опције $R_2(p_2)$, $R_3(p_3)$ и $R_4(f_4(k))$ дефинишу опсег одступања од тренутно најбоље вредности $C_{\max}$. У односу на ефикасност, $R_3(p_3)$ и $R_4(f_4(k))$, не захтевају сортирање секвенци те је $t^{n,m} \leq \boldsymbol{arg}_2 \cdot T^{n,m}$. Када је број секвенци већи од $\boldsymbol{arg_2}$, тада се бира произвољних $\boldsymbol{arg_2}$ секвенци чије су одговарајуће вредности циљне функције у постављеном опсегу. Израчунавања у опцији $R_2(p_2)$ укључују сортирања p_2 бројева, тако да је ова опција атрактивна за мале вредности p_2.

Потребно је истаћи нека додатна опажања у вези паралелног праћења више секвенци. Број секвенци које су кандидати за селекцију расте експоненцијално кроз итерације. Као последица, опсег вредности циљне функције првих w рангираних секвенци опада кроз итерације. С друге стране, у почетним итерацијама одлуке се доносе на основу малог броја испитиваних позиција. Другим речима, избор је заснован на малом узорку популације. Како међусобно уређење селектованих позиција остаје непромењено до краја извршења алгоритма, погрешан избор у почетним итерацијама има значајне последице на коначно решење. Према томе, корисна стратегија би била да се у почетним итерацијама дозволи избор из ширег опсега вредности циљне функције. Ово може да се оствари на више начина:

- $R_2(p_2)$: фиксан број селектованих секвенци у итерацијама обезбеђује да ова селекција не зависи од опсега вредности циљне функције. За мало p, ово представља добру стратегију;
- $R_4(f_4(k))$: омогућује дефинисање различитих опсега вредности у итерацијама. Функција $f = tp/k^2$, где је tp параметар подешавања, је атрактивна опција за

F|prmu|C$_{max}$. Ова функција је одабрана да би се добили ужи опсези у вишим итерацијама. Најбољи резултати на Тајлардовим инстанцама су добијени за вредност *tp* = 1.7;

– $R_5(p_5)$: може да се користи истовремено са другим *R* ради додатног подешавања вредности селектованог опсега. Када селектовани опсег вредности циљне функције задовољава мање од $\boldsymbol{arg_2}$ секвенци, додаје се још p_5 секвенци. На овај начин се постиже жељено проширење селектованог опсега.

У експерименталној студији су поређени резултати у случајевима када је $\boldsymbol{arg_4} = R_1$ и $\boldsymbol{arg_4} = R_4(1.7/k^2)$.

На крају су испитане остале вредности аргумената: *ord* = *DSC*; $\boldsymbol{arg_2} > 1$ и $\boldsymbol{arg_7} > 1$. Подразумевана вредност, $\boldsymbol{arg_5} = 0$, је коришћена током свих експеримената. Важно је да се истакне разлика између $\boldsymbol{arg_5} = 0$ и $\boldsymbol{arg_5} = 1$. Резултати у Табели 10-7 потврђују да HEX не може бити бољи од GCA($\boldsymbol{arg_5} = 1$). Ово не важи када је $\boldsymbol{arg_5} = 0$. У том случају најбоље секвенце се бирају међу свим тренутно запамћеним секвенцама. Због тога, у одређеном тренутку извршавања програма, HEX секвенца може бити изостављена из даље обраде. Последица је да вредност *makespan*-а добијена алгоритмом HEX може бити увећана иако се повећава број праћених секвенци. Да би се ово избегло, у случајевима када је $\boldsymbol{arg_5} = 0$, увек треба да се уместо $val(\boldsymbol{arg_2})$ бира $\overline{val}(\boldsymbol{arg_2})$. На пример, GCA($\boldsymbol{arg_2} = \overline{5}$) се прво извршава као алгоритам HEX, затим као GCA($\boldsymbol{arg_2} = 2$) и тако све до GCA ($\boldsymbol{arg_2} = 5$). На крају се бира најбоља од свих добијених секвенци. За усвојено ограничење $t^{n,m} < 200T^{n,m}$, горња граница за $\overline{val}(\boldsymbol{arg_2})$ је 19 (1+...+19 = 190 < 200).

Циљ спроведених експеримената је добијање одгавора на следећа питања:

1. који је бољи избор: $\boldsymbol{arg_4} = R_1$ или $\boldsymbol{arg_4} = R_4(1.7/k^2)$, када је $\boldsymbol{arg_2} > 1$?
2. колика је разлика у побољшањима у случају када је $\boldsymbol{arg_2} < \overline{20}$ и када је $\boldsymbol{arg_2} = \overline{130}$ (130 је довољно велико да би закључци могли да се уопште)?

3. које су предности од $\boldsymbol{arg_7} > 1$, када је $\boldsymbol{arg_2} < \overline{20}$?
4. која је разлика у побољшању резултата применом MULTIGCA(*ASC*, *DSC*,..., A_z, $t_{\max}$) у односу на MULTIGCA(*ASC*,..., A_z, $t_{\max}$), када је $\boldsymbol{arg_2} < \overline{20}$?

Одговори на прво и друго питање. су добијени применом алгоритама:

– A1: $GCA(\boldsymbol{arg_2} = \overline{130}, \boldsymbol{arg_4} = R_1)$;

– A2: $GCA(\boldsymbol{arg_2} = \overline{130}, \boldsymbol{arg_4} = R_4(1.7/k^2))$.

Табела 10-7 сумира резултате за HEX, A1 и A2. Ознаке *min*, *av* и *max* означавају најмање, просечне и највеће вредности од $\Delta[\pi]$ из (33) за групе инстанци. Најбољи резултати су осенчени.

Табела 10-7. Поређење HEX, A1 и A2

Инстанце	HEX			A1			A2		
n x m	*min*	*av*	*max*	*min*	*av*	*max*	*min*	*av*	*max*
20 x 5	0.44	3.31	7.22	0.00	2.17	5.67	0.00	1.02	1.87
20 x 10	3.39	4.94	7.66	1.38	3.14	5.12	0.61	1.53	1.99
20 x 20	1.75	3.75	5.53	1.66	2.77	4.15	0.28	1.04	1.53
50 x 5	0.17	0.83	1.69	0.15	0.38	1.13	0.00	0.27	1.13
50 x 10	3.92	5.45	6.52	1.53	3.54	4.76	1.45	2.42	2.88
50 x 20	4.88	6.38	7.88	4.05	5.04	5.84	3.09	3.84	4.26
100 x 5	0.08	0.58	1.39	0.08	0.29	0.75	0.08	0.26	0.75
100 x 10	1.02	2.43	4.06	0.39	1.20	2.12	0.50	1.17	1.96
100 x 20	4.32	5.41	6.30	2.49	4.11	4.92	2.85	3.59	4.08
200 x 10	0.80	1.26	1.61	0.28	0.70	1.54	0.39	0.72	1.54
200 x 20	3.97	4.47	5.01	2.58	3.25	4.00	2.27	2.91	3.63
500 x 20	1.64	2.25	2.93	0.86	1.46	1.80	0.93	1.44	1.97
min	0.08			0.00			0.00		
av	2.20	3.42	4.82	1.29	2.34	3.48	1.04	1.69	2.30
max			7.88			5.84			4.26

Закључци су: A1 и A2 су бољи од алгоритма HEX на свим инстанцама и такође важи да је A2 бољи од A1 на скоро свим инстанцама (за просечне вредности, *impr*(*A*1, NEH) = 31.6%; *impr*(*A*2, NEH) = 50.6%; *impr*(*A*2, *A*1) = 27.8%). Према томе, $\boldsymbol{arg_4} = R_4(1.7/k^2)$ је бољи избор, што уједно представља и одговор на прво питање.

Да би се добио одговор на друго питање, проучен је однос између Δ и вредности аргумента $\boldsymbol{arg_2}$. У првом експерименту, инстанца *ta*-116 је обрађивана алгоритмима:

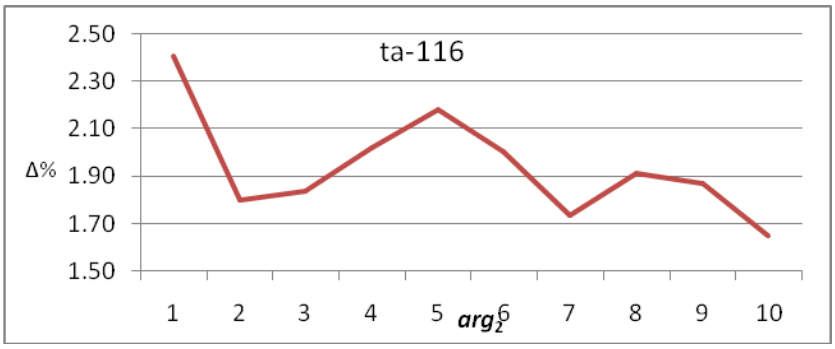
$GCA(\boldsymbol{arg_2} = 1, \boldsymbol{arg_4} = R_4(1.7/k^2))$,

$GCA(\boldsymbol{arg_2} = 2, \boldsymbol{arg_4} = R_4(1.7/k^2))$,

.....

$GCA(\boldsymbol{arg_2} = 10, \boldsymbol{arg_4} = R_4(1.7/k^2))$,

да би се добила релација $\Delta(\boldsymbol{arg_2})$ за $\boldsymbol{arg_2}$ = {1,2,...,10}(Слика 10-8).



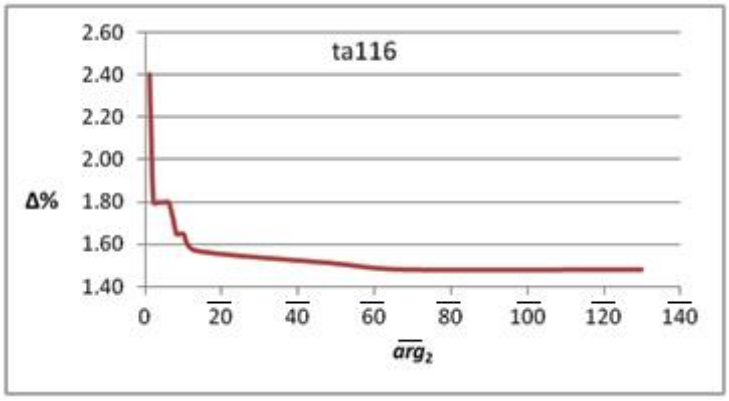
Слика 10-8. $\Delta(\boldsymbol{arg_2})$ график за *ta* -116

Могу да се изведу два значајна закључка: прво, евидентан је глобални опадајући тренд и друго, веће $\boldsymbol{arg_2}$ може да резултује у већој вредности Δ. Примењена статистичка анализа је показала да је опадајући тренд присутан у 115 од 120 инстанци у случајевима када су примењени наведени алгоритми (овај тренд не постоји само код инстанци *ta*-2, *ta*-5, *ta*-22, *ta*-28 и *ta*-35). Важна чињеница је да је овај тренд присутан код свих већих инстанци (*ta*36 - *ta*120). С друге стране, Слика 10-9 приказује исту релацију с тим што је примењен $\overline{\boldsymbol{arg_2}}$ уместо $\boldsymbol{arg_2}$ у целом опсегу од 130 вредности. У ствари, график на Слици 10-9 је изведен из графика на Слици 10-8 када су најмање вредности од Δ у опсегу [1, $\boldsymbol{arg_2}$] исцртане као Δ вредности за $\overline{\boldsymbol{arg_2}}$.

Јасно је да је ова функција монотоно нерастућа за све тестиране инстанце. Важно питање (у односу на постављено ограничење $t^{n,m} < 200T^{n,m}$, тј. $\overline{\boldsymbol{arg_2}} < \overline{20}$) је нагиб одговарајуће криве. Примењена статистичка анализа је показала да:

– најмање вредности за Δ су добијене у оквиру $\overline{\boldsymbol{arg_2}} < \overline{20}$ за 78 од 120 инстанци;

- најмање вредности за Δ су добијене у оквиру $\overline{\boldsymbol{arg}_2} < \overline{10}$ за 68 од 120 инстанци;
- за 98% тестираних инстанци, релативно побољшање вредности Δ, када је $\overline{\boldsymbol{arg}_2}$ у опсегу $[\overline{20}, \overline{30}]$, је мање од 6%;
- општи закључак је да се $\overline{\boldsymbol{arg}_2} = \overline{10}$ и $\overline{\boldsymbol{arg}_2} = \overline{20}$ разликују у вредностима Δ у опсегу од 2% за све тестиране инстанце.



Слика 10-9. Типичан облик функције $\Delta(\overline{\boldsymbol{arg}_2})$

Према томе, одговор на друго питање је да се најзначајнија побољшања добијају до $\overline{\boldsymbol{arg}_2} < \overline{20}$. Примена аргумента $\overline{\boldsymbol{arg}_2} < \overline{10}$ незнатно квари побољшања у односу на $\overline{\boldsymbol{arg}_2} < \overline{20}$, али даје уштеду у CPU времену са $190T^{n,m}$ на $45T^{n,m}$. Ово отвара могућност примене MULTIGCA са GCA($\overline{\boldsymbol{arg}_2} \le \overline{10}$) и још неким додатним алгоритмом.

За одговор на треће питање тестиран је алгоритам A3 са следећим вредностима параметара:

A3: $GCA(\overline{\boldsymbol{arg}_2} = \overline{50}, \boldsymbol{arg}_4 = R_4(1.7/k^2), \overline{\boldsymbol{arg}_7} = \overline{32})$.

Када се у истом GCA користе више различитих $\overline{val}$ вредности, разматрају се све комбинације ових вредности. Ово подразумева да је за A3,

$t^{n,m} = \frac{50 \cdot 51}{2} \cdot 32T^{n,m} = 40{,}800T^{n,m}$.

Наравно, A3 је конструисан само за потребе изведених експеримената.

У Табелама 10-8, 10-9, 10-10 и 10-11, поређени су резултати за A2 и A3. У овим табелама, у колони v_1 је уписан $\boldsymbol{arg}_2^*$ за A2. Табеле приказују резултате за групе највећих Тајлардових инстанци, али сви закључци важе и за преостале Тајлардове инстанце. За A3, колона v_2 садржи $\boldsymbol{arg}_2^*$ у односу на свих 32 вредности од $\boldsymbol{arg}_7$. Колона v_3 приказује $\boldsymbol{arg}_7^*$ када је примењен v_2. Крајња десна колона, означена са LB-UB, приказује најбоље доње и горње границе, објављене до 14.7.2015. год.

Табела 10-8. Поређење HEX, A2 и A3 за инстанце 100 x 20

Instance	C_{max}			Δ			v_1	v_2	v_3	
n x m	HEX	A2	A3	HEX	A2	A3	A2	A3	A3	LB-UB
100 x 20										
ta 81	6593	6436	6436	6.30	3.77	3.77	25	25	1	6106-6202
ta 82	6523	6418	6378	5.50	3.80	3.15	15	9	9	6183-6183
ta 83	6627	6503	6486	5.68	3.70	3.43	28	9	7	6252-6271
ta 84	6541	6500	6437	4.34	3.68	2.68	22	17	10	6254-6269
ta 85	6662	6494	6501	5.51	2.85	2.96	84	15	11	6262-6314
ta 86	6722	6594	6582	5.63	3.61	3.43	4	9	26	6302-6364
ta 87	6620	6477	6477	5.62	3.33	3.33	37	37	1	6184-6268
ta 88	6739	6662	6658	5.28	4.08	4.01	20	9	13	6315-6401
ta 89	6644	6512	6505	5.88	3.78	3.67	51	16	32	6204-6275
ta 90	6712	6635	6579	4.32	3.12	2.25	124	12	12	6404-6434
Av.				5.41	3.59	3.27	41	16	12	

Табела 10-9. Поређење HEX, A2 и A3 за инстанце 200 x 10

	C_{max}			Δ			v_1	v_2	v_3	
n x m	HEX	A2	A3	HEX	A2	A3	A2	A3	A3	LB-UB
200 x 10										
ta 91	10992	10916	10916	1.20	0.50	0.50	129	7	8	10862-10862
ta 92	10649	10574	10602	1.61	0.90	1.16	119	17	8	10480-10480
ta 93	11050	11025	11017	1.17	0.94	0.87	14	10	3	10922-10922
ta 94	11057	11057	11057	1.54	1.54	1.54	1	1	1	10889-10889
ta 95	10656	10565	10565	1.25	0.39	0.39	8	8	1	10524-10524
ta 96	10412	10388	10388	0.80	0.57	0.57	18	18	1	10329-10329
ta 97	10966	10913	10919	1.03	0.54	0.60	72	17	2	10854-10854
ta 98	10887	10788	10777	1.46	0.54	0.44	32	2	28	10730-10730
ta 99	10583	10494	10486	1.39	0.54	0.46	30	3	24	10438-10438
ta 100	10801	10758	10758	1.18	0.78	0.78	8	8	1	10675-10675
Av.				1.26	0.72	0.73	43	9	8	

Табела 10-10. Поређење HEX, A2 и A3 за инстанце 200 x 20

	C_{max}			Δ			v_1	v_2	v_3	
n x m	HEX	A2	A3	HEX	A2	A3	A2	A3	A3	LB-UB
200 x 20										
ta 101	11639	11450	11461	3.97	2.28	2.38	76	16	20	11152-11195
ta 102	11741	11610	11568	4.80	3.63	3.26	19	16	11	11143-11203
ta 103	11804	11671	11651	4.64	3.46	3.28	6	11	13	11281-11281
ta 104	11770	11615	11590	4.39	3.02	2.79	75	14	6	11275-11275
ta 105	11729	11515	11500	4.17	2.27	2.14	35	11	18	11259-11259
ta 106	11660	11533	11508	4.33	3.19	2.97	20	10	11	11176-11176
ta 107	11832	11659	11655	4.15	2.60	2.60	34	34	1	11337-11360
ta 108	11828	11630	11630	4.36	2.61	2.61	50	50	1	11301-11334
ta 109	11753	11523	11517	5.01	2.96	2.90	15	7	31	11145-11192
ta 110	11841	11636	11587	4.90	3.08	2.65	15	23	4	11284-11288
Av.				4.47	2.91	2.76	34	19	12	

Табела 10-11. Поређење HEX, A2 и A3 за инстанце 500 x 20

	C_{max}			Δ			v_1	v_2	v_3	
n x m	HEX	A2	A3	HEX	A2	A3	A2	A3	A3	LB-UB
500 x 20										
ta 111	26733	26530	26532	2.59	1.81	1.82	73	10	16	26040-26059
ta 112	27297	27042	26962	2.93	1.97	1.67	81	39	2	26500-26520
ta 113	26959	26743	26741	2.23	1.41	1.40	76	44	2	26371-26371
ta 114	26984	26845	26760	2.00	1.47	1.15	87	9	9	26456-26456
ta 115	26861	26627	26624	2.00	1.11	1.10	85	23	2	26334-26334
ta 116	27114	26869	26850	2.41	1.48	1.41	68	41	9	26469-26477
ta 117	26822	26634	26607	1.64	0.93	0.83	87	14	27	26389-26389
ta 118	27182	26975	26951	2.34	1.56	1.47	84	20	2	26560-26560
ta 119	26689	26414	26411	2.63	1.57	1.56	87	26	2	26005-26005
ta 120	26796	26751	26767	1.28	1.11	1.17	74	12	29	26457-26457
Av.				2.20	1.44	1.36	80	24	10	

A2 надмашује A3 у само шест од 40 приказаних инстанци. Ове табеле такође показују да су најбољи резултати за A3 добијени за ниже вредности $\boldsymbol{arg_2}$ (колона v_2), у поређењу са одговарајућим вредностима $\boldsymbol{arg_2}$ за A2 (колона v_1), када је $\boldsymbol{arg_7} > 1$.

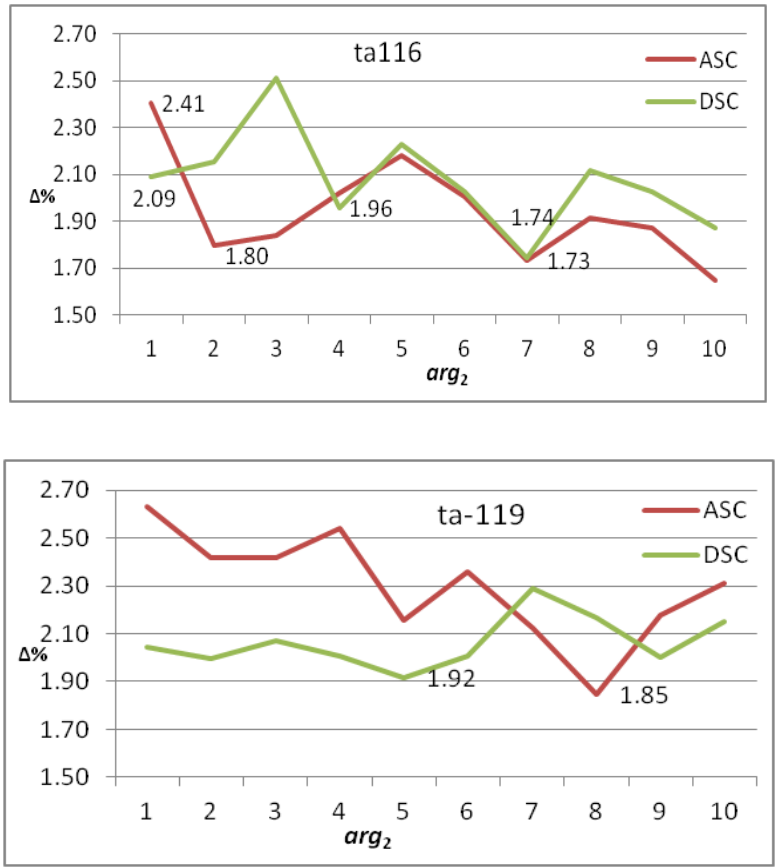
Табела 10-12 приказује функцију $\Delta(\boldsymbol{arg}_2, \boldsymbol{arg}_7)$ за инстанцу *ta*-45. Што су резултати бољи, осенченост им је у табели већа.

Табела 10-12. Δ у функцији од $\boldsymbol{arg_2}$ и $\boldsymbol{arg_7}$ за *ta*-45

$\boldsymbol{arg_7}$ / $\boldsymbol{arg_2}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52	6.52
2	5.38	5.38	6.62	6.62	6.62	6.62	5.07	5.28	5.28	5.75	5.75	7.19	7.19	7.19	7.19	6.92	5.04	6.92
3	5.11	5.11	4.44	4.44	4.44	4.44	5.71	5.28	5.28	5.75	5.75	6.65	6.65	6.65	6.65	6.92	6.12	6.92
4	5.11	5.11	5.28	5.28	5.28	5.28	5.28	5.34	5.34	6.08	6.08	6.15	6.15	6.15	6.15	6.12	6.12	5.04
5	6.72	4.57	4.84	4.84	4.84	4.84	5.04	6.28	6.28	5.44	5.44	6.08	6.08	6.08	6.08	5.44	5.44	5.14
6	4.54	4.27	5.28	5.28	5.28	5.28	5.04	6.28	6.28	5.01	5.01	6.49	6.49	6.49	6.49	5.98	5.98	4.77
7	5.54	5.54	5.28	5.28	5.28	5.28	4.84	5.44	5.44	5.61	5.61	5.28	5.28	5.28	5.28	5.21	5.21	4.77
8	5.54	5.28	5.95	5.95	5.95	5.95	4.23	5.44	5.44	5.54	5.54	5.61	5.61	5.61	5.61	5.24	5.24	4.74
9	5.48	4.87	5.24	5.24	5.24	5.24	4.67	5.44	5.44	5.54	5.54	5.61	5.61	5.61	5.61	5.24	5.24	4.74
10	4.30	4.87	6.08	6.08	6.08	6.08	3.73	5.44	5.44	5.54	5.54	5.61	5.61	5.61	5.61	5.24	5.24	4.74
11	5.21	4.87	5.04	5.04	5.04	5.04	5.04	5.44	5.44	5.61	5.61	5.21	5.21	5.21	5.21	5.17	5.17	4.74
12	5.91	5.14	5.28	5.28	5.28	5.28	3.76	4.40	4.40	4.27	4.27	5.21	5.21	5.21	5.21	5.17	5.17	4.74
13	5.11	5.14	4.74	4.74	4.74	4.74	4.33	4.67	4.67	4.10	4.10	5.21	5.21	5.21	5.21	6.32	6.32	4.74
14	5.11	5.14	4.84	4.84	4.84	4.84	4.64	4.67	5.95	4.10	4.10	5.21	5.21	5.21	5.21	5.17	5.17	4.74
15	5.91	5.14	4.84	4.84	4.84	4.84	4.94	4.07	5.95	4.10	4.10	5.21	5.21	5.21	5.21	5.17	5.17	4.74
16	5.91	3.83	4.74	4.74	4.74	4.74	4.77	4.07	5.95	4.50	4.50	5.28	5.28	5.28	5.28	5.17	5.17	4.74
17	5.91	3.83	4.50	4.50	4.50	4.50	4.77	4.07	5.95	4.44	4.44	6.05	6.05	6.05	6.05	5.17	5.17	4.74
18	5.91	3.83	4.50	4.50	4.00	4.50	4.77	4.07	5.95	5.17	5.17	5.71	5.71	5.71	5.71	5.17	5.17	4.74
19	5.91	3.83	4.84	4.84	4.00	4.84	4.57	4.70	5.95	4.67	4.67	5.71	5.71	5.71	5.71	5.17	5.17	4.74
20	5.91	3.83	4.50	4.50	4.50	4.50	4.57	4.70	5.95	4.67	4.67	5.38	5.38	5.38	5.38	5.17	5.17	4.74
21	5.91	3.83	4.50	4.40	4.40	4.40	5.34	4.70	5.44	4.67	4.67	5.38	5.38	5.38	5.38	5.17	5.17	4.74
22	3.86	3.90	4.50	4.30	4.30	4.30	5.34	5.44	5.44	4.67	4.67	5.38	5.38	5.38	5.38	5.31	5.31	4.74
23	4.27	4.27	4.50	4.40	4.40	4.40	5.34	4.27	4.27	4.67	4.67	5.91	5.91	5.91	5.91	5.41	5.41	4.74
24	4.47	4.47	4.50	4.40	4.40	4.40	4.17	4.27	4.27	4.67	4.67	5.91	5.91	5.91	5.91	5.41	5.41	4.84
25	4.47	4.47	4.50	4.40	4.40	4.40	4.81	5.28	5.28	4.67	4.67	5.91	5.91	5.91	5.91	5.41	5.41	4.84

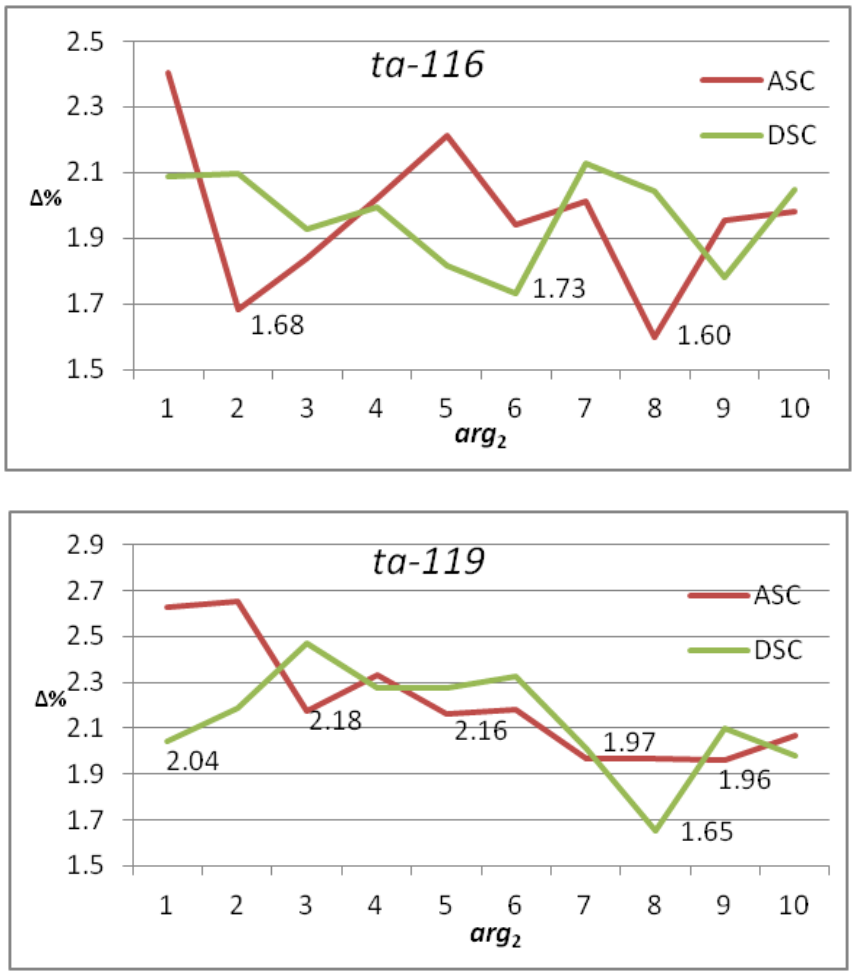
Најбољи резултат за ову инстанцу је $\Delta = 3.73$, добијен за вредности $\boldsymbol{arg_2}$ = 10 и $\boldsymbol{arg_7}$ = 7. CPU време, $t^{n,m}$ за $GCA(\overline{\boldsymbol{arg_2}} = \overline{10}, \overline{\boldsymbol{arg_7}} = \overline{7})$ је $385T^{n,m}$, што је изнад

усвојене горње границе. Резултат $\Delta = 4.27$ је добијен за вредности $\boldsymbol{arg_2} = 6$ и $\boldsymbol{arg_7} = 2$ и за тај случај је $t^{n,m} = 42T^{n,m}$. Релативно побољшање, (34) је: $impr(GCA(\overline{\boldsymbol{arg_2}} = \overline{6}, \overline{\boldsymbol{arg_7}} = \overline{2}), NEH) = 34.5\%$. Сви резултати за $\boldsymbol{arg_7} > 11$ су лошији од 4.27. У односу на усвојена временска ограничења, дозвољени парови вредности су ограничени на: $(\overline{\boldsymbol{arg_2}} < \overline{20}, \boldsymbol{arg_7} = 1)$, $(\overline{\boldsymbol{arg_2}} < \overline{14}, \overline{\boldsymbol{arg_7}} = \overline{2})$, $(\overline{\boldsymbol{arg_2}} < \overline{12}, \overline{\boldsymbol{arg_7}} = \overline{3})$, $(\overline{\boldsymbol{arg_2}} < \overline{10}, \overline{\boldsymbol{arg_7}} = \overline{4})$ итд. За 91% Тајлардових инстанци, $(\overline{\boldsymbol{arg_2}} < \overline{14}, \overline{\boldsymbol{arg_7}} = \overline{2})$ је најбољи избор. На основу овога је овај пар усвојен као одговор на треће питање. Најзад, да би се добио одговор на четврто питање, испитане су предности примене *DSC*. Када су примењене обе вредности, *ASC* и *DSC* у MULTIGCA, претходно усвојен $(\overline{\boldsymbol{arg_2}} < \overline{14}, \overline{\boldsymbol{arg_7}} = \overline{2})$, мора да се замени са $(ASC, \overline{\boldsymbol{arg_2}} < \overline{10}, \overline{\boldsymbol{arg_7}} = \overline{2})$ и $(DSC, \overline{\boldsymbol{arg_2}} < \overline{10}, \overline{\boldsymbol{arg_7}} = \overline{2})$ да би било задовољено постављено временско ограничење. Слика 10-10 приказује зависност између $val(\boldsymbol{arg_2})$ и Δ за инстанце *ta*-116 и *ta*-119, када је $\boldsymbol{arg_7} = 1$.



Слика 10-10. Поређење *ASC* и *DSC* на *ta*-116 и *ta*-119

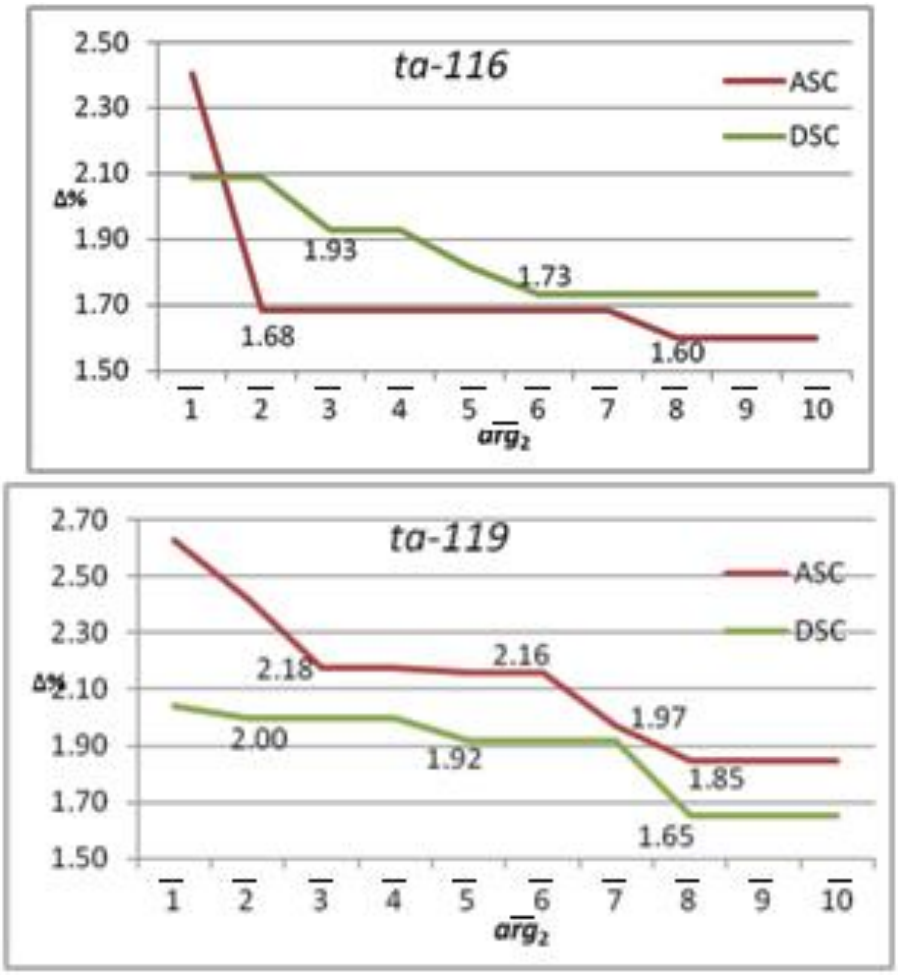
Слика 10-11 приказује исту зависност када је arg_7 = 2. Ознаке на дијаграмима обележавају тренутне најбоље вредности од Δ током повећавања вредности од arg_2. За *ta*-116, *ASC* увек даје боље вредности када је $arg_2 > 1$. За инстанцу *ta*-119, *DSC* увек даје боље вредности када је $arg_7 = 2$.



Слика 10-11. Поређење *ASC* и *DSC* на *ta*-116 и *ta*-119 са $arg_7 = 2$

Главни закључак је да побољшање може са сигурношћу да се постигне само ако се примене и *ASC* и *DSC*. Такође, због укрштених графика приказаних релација, закључак је да предност било којег од ова два уређења за ниже вредности од arg_2 не гарантује исту предност и за више вредности од arg_2.

Слика 10-12 приказује зависност $\Delta(\overline{arg_2})$ за инстанце *ta*-116 и *ta*-119 када је $\overline{arg_7} = \overline{2}$ (примењени $arg_7 = 1$ и $arg_7 = 2$).



Слика 10-12. Релација $\Delta(\overline{\boldsymbol{arg}_2})$ за $\overline{\boldsymbol{arg}_7} = \bar{2}$

За инстанцу *ta*-116, *impr*((*ASC*, $\overline{\boldsymbol{arg}_2} = \bar{9}, \overline{\boldsymbol{arg}_7} = \bar{2}$,), (*DSC*, $\overline{\boldsymbol{arg}_2} = \bar{9}, \overline{\boldsymbol{arg}_7} = \bar{2}$) = 7.5% а за инстанцу *ta*-119, *impr*((*DSC*, $\overline{\boldsymbol{arg}_2} = \bar{9}, \overline{\boldsymbol{arg}_7} = \bar{2}$), (*ASC*, $\overline{\boldsymbol{arg}_2} = \bar{9}, \overline{\boldsymbol{arg}_7} = \bar{2}$)) = 11%. Просечно релативно побољшање за цео скуп инстанци је 10.5%, при чему је *DSC* бољи у 46% тих инстанци. Ови резулати оправдавају увођење *DSC* у MULTIGCA.

10.6.3 *Нови алгоритам*

Добијени закључци су омогућили да се предложи нов алгоритам HEXE:

HEXE = MULTIGCA(A_4, A_5)

$A_4 = GCA(\overline{\boldsymbol{arg}_2} = \bar{9}, \boldsymbol{arg}_4 = R_4(1.7/k^2), \overline{\boldsymbol{arg}_7} = \bar{2})$

$$A_5 = GCA(DSC, \overline{\boldsymbol{arg_2}} = \bar{9}, \boldsymbol{arg_4} = R_4(1.7/k^2), \overline{\boldsymbol{arg_7}} = \bar{2})$$

Приказани су само аргументи који се разликују од НЕХ аргумената. Временска зависност је $t^{n,m} = 2\frac{9\cdot 10}{2}\cdot 2T^{n,m} = 180T^{n,m}$.

Табела 10-13 приказује поређење алгоритма НЕХ и А2 са новим алгоритмом НЕХЕ на групама Тајлардових инстанци.

Табела 10-13. Поређење НЕХ, А2 и НЕХЕ

Instance	HEX			A2			HEXE		
n x m	*min*	*av*	*max*	*min*	*av*	*max*	*min*	*av*	*max*
20 x 5	0.44	3.31	7.22	0.00	1.02	1.87	0.00	1.02	1.87
20 x 10	3.39	4.94	7.66	0.61	1.53	1.99	1.08	1.63	2.29
20 x 20	1.75	3.75	5.53	0.28	1.04	1.53	0.28	1.32	3.04
50 x 5	0.17	0.83	1.69	0.00	0.27	1.13	0.08	0.28	1.13
50 x 10	3.92	5.45	6.52	1.45	2.42	2.88	1.53	2.66	3.27
50 x 20	4.88	6.38	7.88	3.09	3.84	4.26	3.17	3.66	4.39
100 x 5	0.08	0.58	1.39	0.08	0.26	0.75	0.08	0.25	0.75
100 x 10	1.02	2.43	4.06	0.50	1.17	1.96	0.56	1.11	1.88
100 x 20	4.32	5.41	6.30	2.85	3.59	4.08	2.73	3.36	4.06
200 x 10	0.80	1.26	1.61	0.39	0.72	1.54	0.29	0.70	1.54
200 x 20	3.97	4.47	5.01	2.27	2.91	3.63	2.26	2.85	3.25
500 x 20	1.64	2.25	2.93	0.93	1.44	1.97	0.86	1.41	1.86
min	0.08			0.00			0.00		
av	2.20	3.42	4.82	1.04	1.69	2.30	1.08	1.69	2.44
max			7.88			4.26			4.39

Из приказаних резултата могу да се издвоје следећа опажања:

- сви НЕХЕ резултати су бољи од НЕХ резултата са јединим изузетком *min*(100 x 5), који је већ веома низак;
- релативна побољшања $impr(\text{NEHE}, \text{NEH})$ су 51% за *min*, 50.6% за *av* и 49% за *max*;
- иако је $8{,}515T^{n,m}$ време обраде за А2, а $180T^{n,m}$ за НЕХЕ, резултати за НЕХЕ су бољи за све инстанце код којих је $n \geq 100$ (изузимајући *min* за инстанцу 100x10). Исти закључци важе и за негруписане инстанце, али презентација ових резултата би захтевала додатни велики простор у дисертацији;
- А2 и НЕХЕ имају исти укупни просек, што непобитно доказује да правилан избор параметара може да да квалитетније резултате уз мањи утрошак CPU времена;
- максималне девијације за НЕХЕ су у оквиру 4.5%, уз напомену да су за највеће инстанце ове девијације 1.54% (200 x 10), 3.25% (200 x 20) и 1.86%

(500 x 20). Ово је изузетно важан резултат, јер се квалитет решења малих инстанци лако може да поправи праћењем већег броја секвенци због кратког времена које је потребно за њихову обраду.

Повећање меморијског простора због паралелног праћења већег броја секвенци је занемарљиво. Потребно је само резервисати додатни простор за 9 x *n* матрицу у коју се уписују паралелне секвенце.

11. ЗАКЉУЧАК

У дисертацији је спроведена опширна анализа ПКХ. Главна идеја која је усмеравала цео ток истраживања је покушај да се обједине три суштинске карактеристике хеуристика: прецизна полиномијална временска сложеност ПКХ, јасно дефинисан део допустивог скупа који се обрађује са ПКХ и изузетно висок ниво квалитета решења која се добијају применама хеуристика побољшања. Остварење овако дефинисаног циља је тражено у аналогији са два кључна појма теорије сложености: недетерминистичком Тјуринговом машином и структуром псеудо-полиномијалних алгоритама. Аналогно недетерминистичкој машини, којом се у сваком кораку доноси исправан избор од више могућих одлука, у приступу ове дисертације се предлаже паралелно праћење више таквих одлука. Како недетерминистичка машина има суперрачунарске способности да тренутно направи прави избор од огромног броја опција, истраживања у дисертацији су усмеравана ка дефинисању услова који ће омогућити да се тај избор сузи на допустиву величину. На тај начин се, усавршавањем хардвера који би омогућавао паралелну обраду све већег броја различитих одлука, проблеми из класе НП нападају из два различита правца. Ово даље омогућава да се сви механизми који се користе у хеуристикама побољшања, примене и на ПКХ уз значајну предност, једнозначно дефинисану област допустивог скупа која се претражује алгоритмом и прецизно дефинисану полиномијалну ефикасност поступка.

Сужавање допустивог скупа на основу специфичности посматраног проблема односно инстанце, у неким случајевима омогућује одређивање скупа вредности циљне функције. Уколико величина овако добијеног дискретног скупа није експоненцијално зависна од величине улаза, предложени приступ омогућује конструисање псеудо-полиномијалне хибридне хеуристике, којом би ПКХ у полиномијалном времену дао оптимално решење.

Да би се остварио постављени циљ било је неопходно да се детаљно анализирају постојеће хеуристике за решавање пермутационих проблема из класе НП и да се уочене предности усвоје, а недостаци исправе у предложеном потпуно новом

приступу генерализованог конструктивног алгоритма. У наставку су сумирани резултати који су проистекли из овакве студије.

11.1. Остварени доприноси

Свеобухватна анализа постојећих ПКХ је показала да постоје значајни проблеми у формулацијама ових хеуристика који као последицу имају редундантност, непрецизност, вишезначност и необјективну евалуацију. У складу са овим, прва група доприноса које доноси примена GCA је управо везана за отклањање ових недостатака:

1. Прототип ПКХ, представљен у поглављу 2, омогућује систематску и свеобухватну репрезентацију ПКХ. Показано је да три функције, f_1, f_2 и f_3 представљају суштину ПКХ и да се хеуристике међусобно разликују само по начинима одређивања ове три функције.
2. Формулације хеуристика GREEDY и SCHED1, приказане у поглављу 5 су примери редундантне презентације хеуристика. Понављање делова алгоритма који су заједнички за све ПКХ одвлачи фокус са делова алгоритма који су јединствени за презентовани алгоритам и по којима се тај алгоритам разликује од осталих. С друге стране, формулација GCA усмерава фокус управо на дефинисање функција f_1, f_2 и f_3.
3. Опште усвојена формулација HEX алгоритма, представљена у потпоглављу 10.3, је један од многих примера непрецизне формулације хеуристика. У истом поглављу је дата GCA формулација HEX алгоритма, а фокус усмерен на две суштински битне ствари у алгоритму: приоритетно уређење и поступак за програмирање HEX алгоритма са Тајлардовим убрзањем. Показано је како двосмисленост формулације поразно утиче на објављене резултате.
4. Недвосмислена формулација алгоритама ограничава двосмисленост експерименталних резултата. У дисертацији је посебно разматран опсег непрецизности експерименталних резултата услед непрецизности формулације HEX поступка.

5. Како је основна структура ПКХ дефинисана преко GCA, ово омогућује да се усвоји одређени GCA код као јединица мере у експерименталном тестирању временске ефикасности ПКХ. То редукује експерименталну евалуацију ефикасности алгоритама на мерење ефикасности примењених функција. Ово даље значајно редукује некохерентност која је евидентна у скоро свим експерименталним тестирањима хеуристика. Према томе, у дисертацији се показало како примена GCA повећава објективност процеса евалуације ПКХ.
6. Такође, у дисертацији је показано како нова енумерација скупа допустивих решења ПКХ омогућује ефикасно кодирање GCA. Дефинисана веза између уведене енумерације и GCA корака омогућује такође истраживачки рад на поправљању постојећих хеуристика. Наиме, за познате оптималне секвенце тест инстанци може да се утврди шта се дешавало са пермутацијом кроз кораке алгоритма на основу предложене ознаке. Ово може да усмери правац даљих побољшања алгоритама.

Друга, можда и важнија група предности GCA се односи на могућности проширења дефиниције ПКХ увођењем проширеног скупа аргумената. На овај начин се отварају огромне могућности побољшања постојећих алгоритама и формирање нових. У дисертацији су ове могућности посебно анализиране, при чему је фокус усмерен на посебну врсту проширења: могућност вишеструке примене истог алгоритма са измењеним вредностима аргумената. Ово је важно због једне суштинске чињенице: вишеструка примена истог алгоритма задржава временску ефикасност алгоритма непромењеном. На овај начин се резултати могу значајно да побољшају уз задржавање свих побројаних предности ПКХ. Ово је значајна алтернатива алгоритмима побољшања.

Експериментална евалуација предложеног приступа је спроведена на три кључна проблема операционог менаџмента: проблем формирања производних ћелија, проблем распореда производних ћелија и проблем редоследа послова у линији. Она је недвосмислено показала предности овог приступа:

1. Нови приступ решавању проблема формирања производних ћелија је показао изузетну супериорност над свим познатим хеуристикама, како у квалитету решења, тако и у временској ефикасности поступка. Ни за једну од 35 инстанци није било потребно да се примени нека хеуристика побољшања и при томе је за сваку инстанцу добијен резултат који је бољи или једнак најбољим објављеним резултатима. Ако се узме у обзир да је у протеклих тридесет година у међународним часописима објављено преко 100 радова са разноврсним алгоритмима за овај проблем и да ни у једном од ових радова нису сви резултати оптимални и да је утрошено време алгоритма у дисертацији за два реда величине краће од најефикаснијих објављених алгоритама, може да се закључи да је у дисертацији остварен суштински допринос решавању проблема формирања производних ћелија.
2. Показано је да је једноставна имплементација GCA на проблем распореда производних ћелија потпуно равноправна на тестираним инстанцама са најпознатијим солвером за овај проблем.
3. Проблем редоследа послова у линији је искоришћен да се покажу све могућности експериментисања са аргументима и њиховим вредностима. Несумњиви допринос је формирање алата који омогућује да се оптимизују алгоритми и креирају нови. Сваки аргумент GCA има прецизно дефинисану везу са током извршавања алгоритма.
4. Резултат дисертације који анализира утицај вредности аргумената на квалитет решења је нови алгоритам који на свим Тајлардовим инстанцама надмашује чувену HEX хеуристику уз непромењену временску сложеност алгоритма. Треба напоменути да је HEX хеуристика једна од најпроучаванијих хеуристика и да постоји велики број покушаја побољшања ове хеуристике. Нови алгоритам даје најбоље резултате од свих објављених ПКХ којима се предлажу побољшања HEX хеуристике.
5. Сви приказани резултати су добијени на једнопроцесорским рачунарима. Предложени приступ може тренутно да се имплементира на вишепроцесорским рачунарима, јер су праћене секвенце потпуно једнозначно дефинисане и независне једна од друге.

11.2. Правци будућих истраживања

Концепт генерализације отвара више праваца будућих истраживања. Прво, како у општем случају и комбинације и варијације могу да се формулишу помоћу пермутација, интересантно је проверити GCA на комбинаторним проблемима који нису чисто пермутационог карактера. У том смислу је на пример погодан *bin packing* проблем, BP, у коме се објекти различитих запремина или тежина или цена, итд, групишу у коначан број пакета (*bins*) дефинисаних запремина или тежина или цена, итд, тако да се минимизира број потребних пакета. Ово је проблем из класе НП комплетних проблема. Постоји велики број разноврсних апликација овог проблема, као што су пуњење контејнера, попуњавање камиона, прављење *back-up* фајлова на различитим медијима и мапирање технолошких процеса при пројектовању полупроводничких чипова. Проблем 30 у Табели 5-1 је специјалан случај BP проблема, који је дефинисан у (Brusco, Köhn, & Steinley, 2013). Ова формулација је позната као *one-dimensional minimax bin-packing problem with bin size constraints* (MINIMAX_BSC) и разликује се од класичног BP по томе што су објекти претходно груписани, а пакети конструисани тако да се у сваки пакет ставља тачно по један елемент из сваке групе. Овако формулисан проблем се јавља веома често у апликацијама у којима је потребно попунити пакете уз ограничење броја објеката који задовољавају одређене критеријуме. На пример, у области образовања је потребно конструисати тест или више тестова који се састоје од питања или задатака из одређених области, тако да су тестови, у односу на унапред дефинисане критеријуме, што је могуће међусобно приближнији. Једна од могућих метода је позната *minimax* оптимизација и слична је оптимизацији за проблем редоследа послова на паралелним машинама са *makespan* циљном функцијом (Dell' Amico & Martello, 1995).

Други правац истраживања би било проширивање скупа аргумената, којим се проширује област примењивости генерализованог концепта.

Трећи правац може да иде у смеру дефинисања генерализованих хеуристика побољшања. Иако постоји велики број метахеуристика које симулирају

најразноврсније процесе у природи и друштву, суштински, оне спроводе три поступка на различите начине:

- дефинисање околине која се претражује;
- удаљавање од локалног оптимума и
- избегавање обраде већ обрађиваних решења.

Евентуална генерализација метахеуристика би омогућила даље унапређење хеуристика за проблеме комбинаторне оптимизације у операционом менаџменту.

Најзад, GCA може да се користи за добијање почетних популација за различите метахеуристике, са циљем поправљања укупних резултата тих алгоритама.

ЛИТЕРАТУРА

Agarwal, A., & Sarkis, J. (1997). A review and analysis of comparative performance studies on functional and cellular manufacturing layouts. *Computers & Industrial Engineering, 34*(1), 77–89.

Agraval, A., Colak, S., & Eryarsoy, E. (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research, 169*, 801–815.

Akpinar, S., & Oglu, A. (2014). Modeling and solving mixed-model assembly line balancing problem with setups. Part II: A multiple colony hybrid bees algorithm. *Journal of Manufacturing Systems, 33*, 445–461.

Araya, N., & Riff, M. (2014). A beam search approach to the container loading problem. *Computers &Operations Research, 43*, 100–107 .

Aringhieria, R., Cordoneb, R., & Grosso, A. (2015). Construction and improvement algorithms for dispersion problems. *European Journal of Operational Research, 242*, 21–33.

Arkat, J., Hosseini, L., & Farahani, M. (2011). Minimization of Exceptional Elements and Voids in the Cell Formation Problem Using a Multi-Objective Genetic Algorithm. *Expert System with Applications, 38*, 9597-9602.

Arkin, E., Hassin, R., & Sviridenko, M. (2001). Approximating the maximum quadratic assignment problem. *Information Processing Letters, 77*(1), 13–16.

Armour, G., & Buffa, E. (1963). Heuristic algorithm and simulation approach to relative location of facilities. *Management Science, 9*(2), 294-309.

Arora, S., Frieze, A., & Kaplan, X. (1996). A new rounding pro cedure for the assignment problem with applications to dense graph arrangement problems. *37-th Annual IEEE Symp osium on Foundations of Computer Science (FOCS)*, (стр. 21-30).

Askin, R., & Subramanian, S. (1987). A cost-based heuristic for group technology config- uration. *International Journal of Production Research, 25*(1), 101–13.

Ballakur, A., & Steudel, H. (1987). A within–in cell utilization based heuristic for designing cellular manufacturing system. *International Journal of Production Research, 25*, 639–55.

Banerjee, I., & Das, P. (2012). Group Technology Based Adaptive Cell Formation Using Predator-Prey Genetic Algorithm. *Applied Soft Computing, 12*, 559-572.

Baykasoglu, A. (2004). A metaheuristic algorithm to solve quadratic assignment formulations of cell formation problems without presenting number of cells. *Journal of Intelligent Manufacturing, 15*, 753–9.

Benaventa, E., Landeteb, M., Motaa, E., & Tirado, G. (2015). The multiple vehicle pickup and delivery problem with LIFO constraints. *European Journal of Operational Research, 243*, 752–762.

Benavides, A., Ritt, M., Buriol, L., & Franc, P. (2013). An iterated sample construction with path relinking method: Application to switch allocation in electrical distribution networks. *Computers & Operations Research, 40*, 24-32.

Boctor, F. (1991). A linear formulation of the machine–part cell formation problem. *International Journal of Production Research, 29*(2), 343–56.

Boe, W., & Cheng, C. (1991). A close neighbour algorithm for designing cellular manufacturing systems. *International Journal of Production Research, 29*(10), 2097–116.

Brown, E., & Sumichrast, R. (2001). CF-GGA: a grouping genetic algorithm for the cell formation problem. *International Journal of Production Research, 39*, 3651–3669.

Brusco, M., Köhn, H., & Steinley, D. (2013). Exact and approximate methods for a one-dimensional minimax bin-packing problem. *Annals of Operations Research, 206*, 611-626.

Buffa, E., Armour, G., & Vollmann, T. (1964). Allocating facilities with CRAFT . *Harvard Business Review, 42*(2), 136–158.

Burdett, R., & Kozan, E. (2014). An integrated approach for earthwork allocation, sequencing and routing. *European Journal of Operational Research, 238*, 741-759.

Burkard, R. (1991). Locations with spatial interactions: The quadratic assignment problem. In P. Mirchandani, & R. Francis (Eds.), *Discrete Location Theory* (pp. 387–437). John Wiley and Sons.

Burkard, R., Karisch, S., & Rendl, F. (1997). QAPLIB – A Quadratic Assignment Problem Library. *Journal of Global Optimization, 10*, 391-403.

Campbell, A., & Savelsbergh, M. (2004). Efficient Insertion Heuristics for Vehicle Routing and Scheduling Problems. *Transportation Science, 38*(3), 369-370.

Carrie, A. (1973). Numerical taxonomy applied to group technology and plant layout. *International Journal of Production Research, 11*(4), 399–416.

Chan, H., & Milner, D. (1982). Direct clustering algorithm for group formation in cellular manufacture. *Journal of Manufacturing Systems, 1*, 65–75.

Chandrasekharan, M., & Rajagopalan, R. (1986). MODROC: an extension of rank order clustering for group technology. *International Journal of Production Research, 24*(5), 1221–33.

Chandrasekharan, M., & Rajagopalan, R. (1986a). An ideal seed non-hierarchical clustering algorithm for cellular manufacturing. *International Journal of Production Research, 24*(2), 451–64.

Chandrasekharan, M., & Rajagopalan, R. (1987). ZODIAC: an algorithm for concurrent formation of part-families and machine-cells. *International Journal of Production Research, 25*(6), 835–50.

Chandrasekharan, M., & Rajagopalan, R. (1989). GROUPABILITY: an analysis of the properties of binary data matrices for group technology. *International Journal of Production Research, 27*(6), 1035–52.

Chandrasekharan, R., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research, 155*(2), 426–438.

Chen, C., Vempati, V., & Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research, 80*(2), 389–396.

Cheng, C., Gupta, Y., Lee, W., & Wong, K. (1998). A TSP-based heuristic for forming machine groups and part families. *International Journal of Production Research, 36*, 1325–37.

Companys, R., & Mateo, M. (2007). Different behaviour of a double branch-and-bound algorithm on Fm|prmu|Cmax and Fm|block|Cmax problems. *Computers & Operations Research , 34*, 938–953.

Cook, S. A. (1971). The complexity of theorem proving procedures. *Third Annual ACM Symposium on the Theory of Computing* (pp. 151–158). New York: ACM.

Cui, Y., Cui, Y., & Tang, T. (2015). Sequential heuristic for the two-dimensional bin-packing problem. *European Journal of Operational Research, 240*, 43–53.

Cvetkovic, D., Čangalović, M., Dugošija, D., Kovačević-Vujčić, V., Simić, S., & Vuleta, J. (1996). *Kombinatorna optimizacija: matematička teorija i algoritmi* (Biblioteka Operaciona istraživanja i informacioni sistemi изд., Т. 25). (D. Cvetković, Ур.) Beograd: Društvo operacionih istraživača Jugoslavije.

Dalfard, V. (2013). New Mathematical Model for Problem of Dynamic Cell Formation Based on Number and Average Length of Intra and Intercellular Movements. *Applied Mathematical Modelling, 37*, 1884-1896.

Danilovic, M., & Ilic, O. (2016). A generalized constructive algorithm using insertion-based heuristics. *Computers and Operations Research, 66*, 29-43.

Danilovic, M., & Ilic, O. (2016). Primena generalizovanog konstruktivnog algoritma za formiranja proizvodnih ćelija. *YUINFO.* Kopaonik.

Dell' Amico, M., & Martello, S. (1995). Optimal Scheduling of tasks on identical parallel processors. *ORSA Journal of Computing, 2*, 191–200.

Dıaz, J., Luna, D., & Luna, R. (2012). A GRASP heuristic for the manufacturing cell formation problem. *Top, 20*(3), 679–706.

Dong, X., Chen, P., Huang, H., & Nowak, M. (2013). A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers and Operation Research, 40*(2), 627–32.

Dong, X., Huang, H., & Chen, P. (2006). A More Effective Constructive Algorithm for Permutation Flowshop Problem. Intelligent Data Engineering and Automated Learning – IDEAL 2006. *Lecture Notes in Computer Science, 4224*, 25-32. .

Dong, X., Huang, H., & Chen, P. (2008). An improved NEH-based heuristic for the permutation flow shop problem. *Computers and Operation Research, 35*(12), 3962–8.

Duquea, P., Dolinskaya, I., & Sörensen, K. (2016). Network repair crew scheduling and routing for emergency relief distribution problem. *European Journal of Operational Research, 248*, 272–285.

Ekskioglu, B., Ekskioglu, S., & Jain, P. (2008). A tabu search algorithm for the flowshop scheduling problem with changing neighbourhoods. *Computers and Industrial Engineering, 54*, 1–11.

Elbenani, B., & Ferland, J. (2012). An Exact Method for Solving the Manufacturing Cell Formation Problem. *International Journal of Production Research, 50*, 4038-4045.

Erdogana, G., Battarrab, M., & Calvo, R. (2015). An exact algorithm for the static rebalancing problem arising in bicycle sharing systems. *European Journal of Operational Research, 245*, 667–679.

Etiler, O., Toklu, M., & Wilson, J. (2004). A genetic algorithm for flow shop scheduling problem. *Journal of the Operational Research Society, 55*, 830–835.

Fernandez-Vigas, V., & Framinan, J. (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. . *Computers and Operations Research, 45*, 60-67.

Fleszar, K. (2013). Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. *Computers & Operations Research, 40*, 463- 474.

Fleurent, C., & Glover, F. (1999). Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing, 11*, 189–203.

Framinan, J., Gupta, J., & Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society, 55*, 1243–1255.

Framinan, J., Leisten, R., & Ruiz-Usano, R. (2005). Comparison of heuristics for flowtime minimization in permutation flowshops. *Computers and Operations Research, 32*, 1237–54.

Framinan, J., Leisten, R., Rajendran, C., & . (2003). Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *[17] Framinan JM, Leisten R, Rajendran C. Different initial sequences for the heuristic of Nawaz, Enscore and Ham to minimize makespan, International Journal of Production Research, 41*(1), 121–148.

Garcia-Villoria, A., Salhi, S., Corominas, A., & Pastor, R. (2011). Hyper-heuristic approaches for the response time variability problem. *European Journal of Operational Research, 211*, 160-169.

Garey, M., & Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* San Francisco, CA, USA: Freeman.

Ghezavati, V., & Saidi-Mehrabad, M. (2011). An Efficient Hybrid Self-Learning Method for Stochastic Cellular Manufacturing Problem: A Queuing-Based Analysis. *Expert Systems with Applications, 38*, 1326-1335.

Ghiani, G., Lagana, D., & Musmanno, R. (2006). A constructive heuristic for the Undirected Rural Postman Problem. *Computers & Operations Research, 33*, 3450–3457.

Ghosh, T., Dan, P., Sengupta, S., & Chattopadhyay, M. (2010). Genetic rule based techniques in cellular manufacturing (1992–2010): a systematic survey. *International Journal of Engineering, Science and Technology, 2*(5), 198–215.

Ghosh, T., Sengupta, S., Chattopadhyay, M., & Dan, P. (2010). Meta-heuristics in cellular manufacturing: a state-of-the-art review. *International Journal of Industrial Engineering Computations, 2*, 87–122.

Gilmore, P. (1962). Optimal and suboptimal algorithms for the quadratic assignment problem. *SIAM Journal on Applied Mathematics, 10*, 305–313.

Glover, F., Gutin, G., Yeo, A., & Zverovich, A. (2001). Construction heuristics for the asymmetric TSP. *European Journal of Operational Research, 129*, 555-568.

Goncalves, J., & Resende, M. (2004). An evolutionary algorithm for manufacturing cell formation. *Computers & Industrial Engineering, 47*, 247–73.

Grabowski, J., & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion. *Computers and Operations Research, 31*(11), 1891–1909.

Grabowski, J., Pempera, J., & . (2001). New Block Properties for the Permutation Flow Shop Problem with Application in Tabu Search. *The Journal of the Operational Research Society, 52*(2), 210-220.

Greene, T., & Sadowski, R. (1984). A review of cellular manufacturing assumptions, advantages, and design techniques. *Journal of Operations Management, 4*(2), 85–97.

Gupta, Y., & Stafford Jr, E. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research, 169*, 699-711.

Gutin, G., & Yeo, A. (2002). Polynomial approximation algorithms for TSP and QAP with a factorial domination number. *Discrete Applied Mathematics, 119*((1–2)), 107–116.

Hall, L. (1998). Approximability of flow shop scheduling. *Mathematical Programming, 82*, 175–190.

Haq, A., Saravanan, M., Vivekraj, A., & Prasad, T. (2007). A scatter search approach for general flow shop scheduling problem. *International Journal of Advanced Manufacturing Technology, 31*, 751–761.

Hea, K., Jia, P., & Li, C. (2015). Dynamic reduction heuristics for the rectangle packing area minimization problem. *European Journal of Operational Research, 241*, 674–685.

Hejazi, S., & Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research, 43*(14), 2895–2929.

Hua, Q., Lim, A., & Zhu, W. (2015). The two-dimensional vector packing problem with piecewise linear cost function. *Omega, 50*, 43–53.

Ilić, O. (2003). Kompjuterizovana metoda redosleda delova u fleksibilnim transfer linijama. *YU INFO.* Kopaonik.

Ilić, O. (2014). An e-Learning tool considering similarity measures for manufacturing cell formation. *Journal of Intelligent Manufacturing, 25*(3), 617-628.

Ilić, O. (2015). *Računarski integrisana proizvodnja.* Beograd: Fakultet organizacionih nauka, Univerzitet u Beogradu.

Ilić, O., & Cvetić, B. (2014). A comparative case study of e-learning tools for manufacturing cell formation. *Journal of Advanced Mechanical Design, Systems, and Manufacturing, 8*(3), 1-15.

Irani, S. (2012). *PFAST*. Преузето December 10, 2012 са http://pfast.ise.ohio-state.edu/pfast/BuyPFast.html

Irani, S., & Huang, H. (2005). *Hybrid cellular layouts: new ideas for design of flexible and lean layouts for jobshops.* Columbus, USA: Department of Industrial, Welding and Systems Engineering, The Ohio State University.

Irani, S., Zhang, H., Zhou, J., Huang, H., Udai, T., & Subramanian, S. (2000). Production flow analysis and simplification toolkit (PFAST). *International Journal of Production Research, 38*(8), 1855-1874.

Islier, A. (2005). Group technology by an ant system algorithm. *International Journal of Production Research, 43*(5), 913–32.

James, T., Brown, E., & Keeling, K. (2007). A hybrid grouping algorithm for the cell formation problem. *Computers & Operations Research, 34*, 2059–79.

Jarboui, B., Ibrahim, S., Siarry, P., & Rebai, A. (n.d.). A combinatorial particle swarm optimisation for solving permutation flowshop problems. *Computers & Industrial Engineering , 54*, 526–538.

Jin, F., Song, S., & Wu, C. (2007). An improved version of the NEH algorithm and its application to large-scale flow-shop scheduling problems. *IIE Transactions, 39*, 229-234.

Johnson, S. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly, 1*, 61–68.

Johnson, S. (1959). Discussion: Sequencing n jobs on two machines with arbitrary time lags. *Management Science, 5*, 299–303.

Kalczynski, P., & Kamburowski, J. (2007). On the NEH heuristic for minimizing the makespan in permutation flowshops. *Omega-The International Journal of Management Science, 35*, 53–60.

Kalczynski, P., & Kamburowski, J. (2008). An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers and Operations Research, 35*, 3001–3008.

Kalczynski, P., & Kamburowski, J. (2011). On recent modifications and extensions of the NEH heuristic for flow shop sequencing. *Found Comput Decis Sci, 36*(1), 17–34.

Kamraniand, A., Parsaie, H., & Chaudhry, M. (1993). A survey of design methods for manufacturing cells. *Computers & Industrial Engineering, 25*(1– 4), 487–90.

Kao, Y., & Li, Y. (2008). Ant colony recognition systems for part clustering problems. *International Journal of Production Research, 46*(15), 4237–58.

Kia, R., Baboli, A., Javadian, N., Tavakkoli-Moghaddam, R., Kazemi, M., & Khorrami, J. (2012). Solving a Group Layout Design Model of a Dynamic Cellular Manufacturing System with Alternative Process Routings, Lot Splitting and Flexible Reconfiguration by Simulated Annealing. *Computers & Operations Research, 39*, 2642-2658.

King, J. (1980). Machine-component grouping in production flow analysis: an approach using a rank order clustering algorithm. *International Journal of Production Research, 18*, 213–32.

King, R., & Nakornchai, V. (1982). Machine-component group formation in group technology: review and extension. *International Journal of Production Research, 20*(2), 117–33.

Kirlik, G., & Sipahiogly, A. (2012). Capacitated arc routing problem with deadheading demands. *Computers & Operations Research, 39*, 2380-2394.

Knuth, D. (1973). The Art of Computer Programming. Addison–Wesley.

Knuth, D. E. (2011). *The Art of Computer Programming, Volume 4A Combinatorial Algorithms.* Boston, USA: ADDISON-WESLEY, Pearson Education Inc.

Koopmans, T., & Beckmann, M. (1957). Assignment problems and the location of economic activities. *Econometrica, 25*, 53-76.

Kumar, C., & Chandrasekharan, M. (1990). Grouping efficacy: a quantitative criterion for goodness of block diagonal forms of binary matrices in group technology. *International Journal of Production Research, 28*, 233–43.

Kumar, K., & Vannelli, A. (1987). Strategic subcontracting for efficient disaggregated manufacturing. *International Journal of Production Research, 25*(12), 1715–28.

Kumar, K., Kusiak, A., & Vannelli, A. (1986). Grouping of parts and components in flexible manufacturing systems. *European Journal of Operations Research, 24*, 387–97.

Kusiak, A., & Cho, M. (1992). Similarity coefficient algorithm for solving the group technology problem. *International Journal of Production Research, 30*(11), 2633–46.

Kusiak, A., & Chow, W. (1987). Efficient solving of the group technology problem. *Journal of Manufacturing Systems, 6*(2), 117–24.

Ladhari, T., & Haouari, M. (2005). A computational study of the permutation flow shop problem based on a tight lower bound. *Computers & Operations Research, 32*, 1831–1847.

Laha, D., & Mandal, P. (2007). Improved heuristically guided genetic algorithm for the flow shop scheduling problem. *International Journal of Services and Operations Management, 3*, 313–331.

Lan, G., DePuy, G., & Whitehouse, G. (2007). An effective and simple heuristic for the set covering problem. *European Journal of Operational Research, 176*, 1387–1403.

Lawler, E. (1963). The quadratic assignment problem. *Management Science, 9*, 586-599.

Lehmer, D. (1960). Teaching combinatorial tricks to a computer. *Sympos. Appl. Math. Combinatorial Analysis 10* (pp. 179–193). Amer. Math. Soc.

Li, X., Baki, M., & Aneja, Y. (2010). An ant colony optimization metaheuristic for machine-part cell formation problems. *Computers &Operations Research, 37*, 2071–81.

Liao, C., Tseng, C., & Luarn, P. (2007). A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers & Operations Research, 34*, 3099–3111.

Lin, S., Ying, K., & Lee, Z. (2010). Part-Machine Cell Formation in Group Technology Using a Simulated An nealing-Based MetaHeuristic. *International Journal of Production Research, 48*, 3579-3591.

Liu, B., Wang, L., & Jin, Y. (2007). An effective PSO-based memetic algorithm for flow shop scheduling. *IEEE Transactions on Systems, Man and Cybernetics – Part B. Cybernetics , 37*, 18–27.

Lopes, R., Plastri, B., Ferreira, C., & Santos, B. (2014). Location-arc routing problem: Heuristic approaches and test instances. *Computers &OperationsResearch, 43*, 309–317.

Lu, Q., & Dessouky, M. (2006). A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research, 175*, 672–687.

Lust, T., Roux, O., & Riane, F. (2009). Exact and heuristic methods for the selective maintenance problem. *European Journal of Operational Research, 197*, 1166-1177.

Mahdavi, I., Paydar, M., Solimanpur, M., & Heidarzade, A. (2009). Genetic algorithm approach for solving a cell formation problem in cellular manufacturing. *Expert Systems with Applications, 36*, 6598–604.

Mansouri, A., Moattar-Husseini, S., & Newman, S. (2000). A review of the modern approaches to multi-criteria cell design. *International Journal of Production Research, 38*(5), 1201–18.

Mansourila, S., Aktasb, E., & Besikci, U. (2016). Green scheduling of a two-machine flowshop:Trade-off between makespan and energy consumption. *European Journal of Operational Research, 248*(3), 772–788.

Manzini, R., Bindi, F., & Pareschi, A. (2010). The threshold value of group similarity in the formation of cellular manufacturing systems. *International Journal of Production Research, 48*(10), 3029–3060.

McCormick, W., Schweitzer, P., & White, T. (1972). Problem decomposition and data reorganization by a clustering technique. *Operations Research, 20*, 993–1009.

McKendall, J., & Hakobyan, A. (2010). Heuristics for the dynamic facility layout problem with unequal-area departments. *European Journal of Operational Research, 201*, 171-182.

Megala, N., Rajendran, C., & Gopalan, R. (2008). An ant colony algorithm for cell-formation in cellular manufacturing systems. *European Journal of Industrial Engineering, 2*(3), 298–336.

Misevicius, A. (1997). A new algorithm for the quadratic assignment problem. *Information Technology and Control, 5*, 39–44.

Misevicius, A., & Riskus, A. (1999). Multistart threshold accepting: Experiments with the quadratic assignment problem. *Information Technology and Control, 12*, 31–39.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*, 1097–100.

Mosier, C., & Taube, L. (1985). The facets of group technology and their impact on implementation. *OMEGA, 13*(5), 381–91.

Mosier, C., & Taube, L. (1985a). Weighted similarity measure heuristics for the group technology machine clustering problem. *OMEGA, 13*(6), 577–8.

Nagano, M., Miyata, H., & Araújo, D. (2014). A constructive heuristic for total flowtime minimization in a no-waitflowshop with sequence-dependent setup times. *Journal of Manufacturing Systems, 36*, 224–230.

Nawaz, M., Enscore Jr, E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega-The International Journal of Management Science, 11*(1), 91–95.

Noktehdan, A., Karimi, B., & Husseinzadeh Kashan, A. (2010). A differential evolution algorithm for the manufacturing cell formation problem using group based operators. *Expert Systems with Applications, 37*, 4822–9.

Nowicki, E., & Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flowshop problem. *European Journal of Operational Research , 91*, 160–175.

Offodile, O., Mehrez, A., & Grznar, J. (1994). Cellular manufacturing: a taxonomic review framework. *Journal of Manufacturing Systems, 13*(3), 196–220.

Onwubolu, G., & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research, 171*, 674–692.

Onwubolu, G., & Mutingi, M. (2001). A genetic algorithm approach to cellular manufacturing systems. *Computers & Industrial Engineering, 39*(1-2), 125–44.

Osman, I., & Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *OMEGA The International Journal of Management Science, 17*(6), 551–557.

Ozcelik, F., & Sarac, T. (2012). A Genetic Algorithm Extended Modified Sub-Gradient Algorithm for Cell Formation Problem with Alternative Routings. *International Journal of Production Research, 50*, 4025-4037.

Pailla, A., Trindade, A., Parada, V., & Ochi, V. (2010). A numerical comparison between simulated annealing and evolutionary approaches to the cell formation problem. *Expert Systems with Applications, 37*, 5476–83.

Pan, Q., Tasgetiren, M., & Liang, Y. (2007). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Ninth Annual Conference o Genetic and Evolutionary Computation, GECCO '07.* London, UK.

Papaioannou, G., & Wilson, J. (2010). The evolution of cell formation problem methodologies based on recent studies (1997–2008): Review and directions for future research. *European Journal of Operational Research, 206*(3), 509–21.

Paydar, M., & Saidi-Mehrabad, M. (2013). A Hybrid Genetic-Variable Neighborhood cell Formation Problem Based on Group Efficiency. *Computers & Operations Research, 40*, 980-990.

Paydar, M., Mahdavi, I., Sharafuddin, I., & Solimanpur, M. (2010). Applying Simulated Annealing for Designing Cellular Manufacturing Systems Using MDmTSP. *Computers & Industrial Engineering, 59*, 929-936.

Pereira, J., & Vilà, M. (2015). Variable neighborhood search heuristics for a test assembly design problem. *Expert Systems with Applications, 42*, 4805–4817.

Petronijević, B., & Ilić, O. (1994). Egzaktne formulacije problema protocnog redosleda poslova. *XXI YU SYM-OP-IS*, (стр. 752-755). Kotor.

Pinedo, M. (2010). *Scheduling: Theory, Algorithms and Systems* (Fourth изд.). New York: Springer.

Punnen, A., & Wang, Y. (2016). The bipartite quadratic assignment problem and extensions. *European Journal of Operational Research, 250*(3), 715-725.

Punnen, A., Sripratak, P., & Karapetyan, D. (2015). The bipartite unconstrained 0–1 quadratic programming problem: Polynomially solvable cases. *Discrete Applied Mathematics, 193*(1), 1–10.

Queyranne, M. (1986). Performance ratio of heuristics for triangle inequality quadratic assignment problems. *Operations Research Letters, 4*, 231-234.

Rad, S., Ruiz, R., & Boroojerdian, N. (2009). New high performing heuristics for minimizing makespan in permutation flowshops. *Omega , 37*, 331–345.

Rafiei, H., & Ghodsi, R. (2013). A Bi-Objective Mathematical Model toward Dynamic Cell Formation Considering Labor Utilization. *Applied Mathematical Modelling, 37*, 2308-2316.

Rajagopalan, R., & Batra, J. (2006). A genetic algorithm approach for machine cell formation. *Journal of Advance Manufacturing Systems, 5*, 27–44.

Rezaeian, J., Javadian, N., Tavakkoli-Moghaddam, R., & Jolai, F. (2011). A Hybrid Approach Based on the Genetic Algorithm and Neural Network to Design an Incremental Cellular Manufacturing System. *Applied Soft Computing, 11*, 4195-4202.

Ribas, I., Companys, R., & Tort-Martorell, X. (2010). Comparing three-step heuristics for the permutation flow shop problem. *Comput Oper Res, 37*(12), 2062–70.

Riu, M., Fernández, E., & Estrada, M. (2015). Parking slot assignment for urban distribution: Models and formulations. *Omega, 57*(Part B), 157-175.

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research, 165*(2), 479–94.

Ruiz, R., & Stutzle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*, 2033–2049.

Ruiz, R., Maroto, C., & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA the International Journal of Management Science, 34*, 461–476.

Sáa, E., Contrerasb, I., & Cordea, J. (2015). Exact and heuristic algorithms for the design of hub networks with multiple lines. *European Journal of Operational Research, 246*, 186–198.

Saeedi, S., Solimanpur, M., Mahdavi, I., & Javadian, N. (2010). Heuristic Approaches for Cell Formation in Cellular Manufacturing. *Journal of Software Engineering & Applications, 3*, 674-682.

Saraç, T., & Ozcelik, F. (2012). A Genetic Algorithm with Proper Parameters for Manufacturing Cell Formation Problems. *Journal of Intelligent Manufacturing, 23*, 1047-1061.

Sarker, B., Wilhelm, W., & Hogg, G. (1998). One-dimensional machine location problems in a multi-product flowline with equidistant locations. *European Journal of Operational Research, 105*(3), 401–426.

Sarker, B., Wilhelm, W., Hogg, G., & Han, M. (1995). Backtracking of jobs in one-dimensional machine location problems. *European Journal of Operational Research, 85*(3), 593–609.

Seifoddini, H. (1989). A note on the similarity coefficient method and the problem of improper machine assignment in group technology applications. *International Journal of Production Research, 27*(7), 1161–5.

Seifoddini, H., & Wolfe, P. (1986). Application of the similarity coefficient method in group technology. *IIE Transactions, 18*(3), 271–7.

Selim, H., Askin, R., & Vakharia, A. (1998). Cell formation in group technology: review, evaluation and directions for future research. *Computers & Industrial Engineering, 34*(1), 3–20.

Singh, N. (1993). Design of cellular manufacturing systems: an invited review. *European Journal of Operational Research, 69*, 284–91.

Singh, N. (1993). Design of cellular manufacturing systems: An invited review. *European Journal of Operational Research, 69*, 284–291.

Smet, P., Wauters, T., Mihaylov, M., & Berghe, G. (2014). The shift minimisation personnel task scheduling problem: A new hybrid approach and computational insights. *Omega, 46*, 64–73.

Solimanpur, M., Vrat, P., & Shankar, R. (2004). A multi-objective genetic algorithm approach to the design of cellular manufacturing systems. *International Journal of Production Research, 42*(7), 1419–41.

Srinivasan, G., & Narendran, T. (1991). GRAFICS-A nonhierarchical clustering-algorithm for group technology. *International Journal of Production Research, 29*(3), 463–78.

Srinivasan, G., Narendran, T., & Mahadevan, B. (1990). An assignment model for the part- families problem in group technology. *International Journal of Production Research, 28*(1), 145–52.

Stanfel, L. (1985). Machine clustering for economic production. *Engineering Costs and Production Economics, 9*, 73–81.

Stawowy, A. (2006). Evolutionary strategy for manufacturing cell design. *Omega, The International Journal of Management Science, 34*, 1–18.

Stutzle, T. (1998). Applying iterated local search to the permutation flow shop problem. *Technical report, AIDA-98-04, FG Intellektik, FB Informatik*.

Sykora, A., Valdes, R., Bennell, J., & Tamarit, J. (2015). Constructive procedures to solve 2-dimensional bin packing problems with irregular pieces and guillotine cuts. *Omega, 52*, 15–32.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research, 47*(1), 67–74.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research , 64*, 278–85.

Taillard, E. (2004). *Scheduling instances*. Retrieved 7 5, 2015, from http://ina.eivd.ch/collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html

Talarico, L., Sörensen, K., & Springae, J. (2015). Metaheuristics for the risk-constrained cash-in-transitvehicleroutingproblem. *European Journal of Operational Research, 244*, 457–470.

Tansel, B., & Bilen, C. (1998). Move based heuristics for the unidirectional loop network layout problem. *European Journal of Operational Research, 108*(1), 36–48.

Tariq, A., Hussain, I., & Ghafoor, A. (2009). A hybrid genetic algorithm for machine-part grouping. *Computers & Industrial Engineering, 56*, 347–56.

Tasgetiren, M., Sevkl, i. M., & Gencyilmaz, G. (2007). A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research, 177*, 1930–1947.

Tiana, T., Zhub, W., Limc, A., & Wei, L. (2016). The multiple container loading problem with preference . *European Journal of Operational Research, 248*, 84–94.

Tunnukij, T., & Hicks, C. (2009). An enhanced grouping genetic algorithm for solving the cell formation problem. *International Journal of Production Research, 47*(7), 1989–2007.

Vasiljevic, D., & Danilovic, M. (2015). Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems, 35*, 1-9.

Vujošević, M. (2012). *Metode optimizacije u inženjerskom menadžmentu.* Beograd: Akademija inženjerskih nauka Srbije, FON.

Waghodekar, P., & Sahu, S. (1984). Machine-component cell formation in group technology MACE. *International Journal of Production Research, 22*, 937–48.

Wang, J. (1999). A linear assignment clustering algorithm based on the least similar cluster representatives. *IEEE Transactions on Systems Man and Cybernetics Part A – Systems and Humans, 29*(1), 100–104.

Watson, J. B., Whitley, L., & Howe, A. (2002). Contrasting structured and random permutation flowshop scheduling problems: Search space topology and algorithm performance. *ORSA Journal of Computing, 14*(2), 98–123.

Wex, F., Schryen, G., Feuerriegel, S., & Neumann, D. (2014). Emergency Response in Natural Disaster Management: Allocation and Scheduling of Rescue Units. *European Journal of Operational Research, 235*, 697-708.

Wu, T., Chang, C., & Chung, S. (2008). A simulated annealing algorithm for manufacturing cell formation problems. *Expert Systems with Applications, 34*, 1609–17.

Wu, T., Chang, C., & Yeh, J. (2009). A hybrid heuristic algorithm adopting both Boltzmann function and mutation operator for manufacturing cell formation problems. *International Journal of Production Economics, 120*, 669–88.

Wu, T., Chung, S., & Chang, C. (2010). A water flow-like algorithm for manufacturing cell formation problems. *European Journal of Operational Research, 205*, 346–60.

Wu, T., Low, C., & Wu, W. (2004). A tabu search approach to the cell formation problem. *International Journal of Advanced Manufacturing Technology, 23*, 916–24.

Yasuda, K., Hu, L., & Yin, Y. (2005). A grouping genetic algorithm for the multi-objective cell formation problem. *International Journal of Production Research, 43*(4), 829–53.

Yin, X., & Khoo, L. (2011). An Exact Schema Theorem for Adaptive Genetic Algorithm and Its Application to Machine Cell Formation. *Expert Systems with Applications, 38*, 8538-8552.

Yin, Y., & Yasuda, K. (2006). Similarity coefficient methods applied to the cell formation problem: A taxonomy and review. *International Journal of Production Economics, 101*, 329–352.

Ying, K., & Liao, C. (2004). An ant colony system for permutation flow-shop sequencing. *Computers and Operations Research, 31*, 791–801.

Ying, K., Lin, S., & Lu, C. (2011). Cell formation using a simulated annealing algorithm with variable neighbourhood. *European Journal of Industrial Engineering, 5*(1), 22–42.

Yu, J., & Sarker, B. (2003). Directional decomposition heuristic for a linear machine-cell location problem. *European Journal of Operational Research, 149*(1), 142–184.

ZH, J., & Leung, J. (2015). Minimize makespan for parallel batch machines with arbitrary job sizes. *European Journal of Operational Research, 240*, 649–665.

БИОГРАФИЈА

Лични подаци

Датум рођења: 22.04.1985.
Место рођења: Београд, Србија
Држављанство: српско

Образовање

Факултет организационих наука, Универзитет у Београду **2011-**
Ј. Илића 154, 11000 Београд (www.fon.rs)

Докторске студије, изборно подручје МЕНАЏМЕНТ, уписао је школске 2011/2012 године. Положио је следеће испите са оценом 10: Наука о менаџменту код проф. др Мирка Вујошевића, Маркетинг логистика код проф. др Драгана Васиљевића, Интегрисани операциони менаџмент код проф. др Оливера Илића, Одлучивање – изабрана поглавља и Теорија одлучивања код проф. др Бориса Делибашића, Методологија код проф. др Добривоја Михајловића, Теорија алгоритама и Нови трендови у операционим истраживањима код проф. др Мирјане Чангаловић. Испит из предмета Управљање ланцима снабдевања код проф. др Мирка Вујошевића положио је са оценом 9. Приступни рад је одбранио 16. јуна 2016. године.

Факултет организационих наука, Универзитет у Београду **2009 –2011**
Ј. Илића 154, 11000 Београд (www.fon.rs)

Дипломске академске (Мастер) студије из области ИНЖЕЊЕРСКОГ И ОПЕРАЦИОНОГ МЕНАЏМЕНТА, програмско подручје РАЧУНАРСКИ ИНТЕГРИСАНА ПРОИЗВОДЊА И ЛОГИСТИКА је уписао школске 2009/2010. године (буџет), где је положио пет програмом предвиђених испита и одбранио приступни рад са просечном оценом 10.
Завршни – мастер рад под насловом „Проблем квадратне асигнације у рачунарски интегрисаној производњи“ одбранио је пред комисијом у саставу: проф. др Оливер Илић, проф. др Драган Васиљевић и проф. др Мирко Вујошевић, дана 15.03.2011. године, са оценом 10.

Факултет организационих наука, Универзитет у Београду **2004 - 2009**
Ј. Илића 154, 11000 Београд (www.fon.rs)

Прву годину Основних студија уписао школске 2004/2005 године (буџет). Другу годину уписао је школске 2005/2006 на смеру Операциони Менаџмент и определио се за начин студирања по болоњској конвенцији. Апсолвирао је у року, 2008. године. Просечна оцена на студијима је 8.4. Завршни рад је радио из предмета: “Рачунарски интегрисана производња” и “Флексибилни производни системи“ код редовног професора ФОН-а др Оливера Илића под насловом „Проблем квадратне асигнације у рачунарски интегрисаној производњи“ и одбранио га са оценом 10.

Математичка гимназија, Београд **2000 -2004**

Смер за обдарене ученике.
Матурски рад под насловом „Основи статистичке анализе временских низова“ одбранио је са одличном оценом.

ОШ „Михајло Петровић Алас“, Београд **1992 -2000**

Носилац дипломе „Вук Караџић“

Радно искуство

Факултет организационих наука, Универзитет у Београду **2012 -**
Ј. Илића 154, 11000 Београд (www.fon.rs)

Примљен у звање Асистента на катедри РАЧУНАРСКИ ИНТЕГРИСАНА ПРОИЗВОДЊА И ЛОГИСТИКА школске 2012/2013 године.

Поновни избор за Асистента на истој катедри школске 2015/2016 године.

Посебне теме интересовања, на којима кандидат активно ради, су:
Примена комбинаторне оптимизације за решавање проблема из области производње и логистике, *scheduling* проблеми, локацијски проблеми, проблем рутирања возила у ланцима снабдевања, развијање алгоритама за решавање проблема редоследа и распореда у производњи, теорија графова и одређивање најкраћих путања у мрежама.

Био је члан Техничког одбора VIII Скупа привредника и научника СПИН ’11, IX Скупа привредника и научника СПИН ’13, и X Скупа привредника и научника СПИН’15, у организацији Центра за ОМ.

Редовно учествује у припреми “ОМ-Info-day-а” – прилике за информисање потенцијалних (будућих) студената студијске групе ОМ.

Члан је АЛУМНИ асоцијације студената ФОН-овог одсека за ОМ и редовно учествује у организацији њихових скупова.

Држи вежбе и даље из истих десет предмета на основним и мастер студијама.

У анонимним анкетама студената за оцену квалитета вежби оцењен највишим оценама.

Табела 1: Резултати анкете за вредновање рада асистента Милоша Даниловића (оцена на скали од 1 до 5):

ПРЕДМЕТ	ШКОЛСКА ГОДИНА				
	2012/2013	2013/2014	2014/2015	2015/2016	2016/2017
Рачунарски интегрисана производња	4.97	4.92	4.90	4.76	4.99
Логистика	4.88	4.62	4.64	4.87	4.83

Факултет организационих наука, Универзитет у Београду **2010 - 2012**
Ј. Илића 154, 11000 Београд (www.fon.rs)

Примљен као Сарадник у настави на катедри РАЧУНАРСКИ ИНТЕГРИСАНА ПРОИЗВОДЊА И ЛОГИСТИКА школске 2010/2011 године, где је држао вежбе из предмета ЛОГИСТИКА (обавезни предмет на III години основних студија студијске групе ОМ), РАЧУНАРСКИ ИНТЕГРИСАНА ПРОИЗВОДЊА (обавезни предмет на IV години основних студија студијске групе ОМ), из четири изборна предмета студијске групе ОМ: УПРАВЉАЊЕ ОДРЖАВАЊЕМ, МАРКЕТИНГ ЛОГИСТИКА, ФЛЕКСИБИЛНИ ПРОИЗВОДНИ СИСТЕМИ, УПРАВЉАЊЕ ЛАНЦИМА СНАБДЕВАЊА, на мастер студијама на смеру Инжењерски и операциони менаџмент из четири предмета: РАЧУНАРСКИ ИНТЕГРИСАНИ ПРОИЗВОДНИ СИСТЕМИ, ИНТЕГРИСАНИ ЛОГИСТИЧКИ СИСТЕМИ, МАРКЕТИНГ ЛОГИСТИКА 2 и УПРАВЉАЊЕ ЛАНЦИМА СНАБДЕВАЊА 2.

У анонимним анкетама студената за оцену квалитета вежби оцењен највишим оценама:

Табела 2: Резултати анкете за вредновање рада сарадника Милоша Даниловића (оцена на скали од 1 до 5):

ПРЕДМЕТ	ШКОЛСКА ГОДИНА	
	2010/2011	2011/2012
Рачунарски интегрисана производња	4.99	4.81
Логистика	4.91	4.97

Награђен новчаном наградом у јануару 2012 за куповину књига од стране Наставно-научног већа за најбоље постигнуте резултате у анонимној анкети студената за оцену квалитета вежби.

СПИСАК ОБЈАВЉЕНИХ РАДОВА

Милош Даниловић је до сада објавио укупно 21 рад, као аутор и коаутор. Његов научни рад огледа се кроз следећу библиографију радова:

Завршни (дипломски) рад:

Даниловић М., "Проблем квадратне асигнације у рачунарски интегрисаној производњи", (ментор: проф. др О. Илић), Факултет организационих наука, Београд, 2009.

Завршни (мастер) рад:

Даниловић М., "Проблем квадратне асигнације у рачунарски интегрисаној производњи", (ментор: проф. др О. Илић), Факултет организационих наука, Београд, 2011.

Радови објављени у научним часописима међународног значаја (М20)

Радови у врхунским међународним часописима (М21)

1. **Danilovic, M.**, Ilic, O. (2016). A generalized constructive algorithm using insertion-based heuristics. *Computers and Operations Research*, **66**, 29-43, ISSN: 0305-0548 (IF2015: 1.988), DOI: 10.1016/j.cor.2015.07.009
   http://www.sciencedirect.com/science/article/pii/S0305054815001768

2. Vasiljevic, D., **Danilovic**, **M**. (2015). Handling ties in heuristics for the permutation flow shop scheduling problem. *Journal of Manufacturing Systems*, **35**, 1-9, ISSN: 0278-6125 (IF2015: 2.240), DOI: 10.1016/j.jmsy.2014.11.011
   http://www.sciencedirect.com/science/article/pii/S027861251400140X

Рад у истакнутом међународном часопису (М22)

1. **Danilovic, M.**, Vasiljevic, D. (2014). A novel relational approach for assembly system supply planning under environmental uncertainty. *International Journal of Production Research*, **52**(13), 4007-4025, ISSN: 0020-7543 (IF2015: 1.693), DOI: 10.1080/00207543.2014.916429
   http://www.tandfonline.com/doi/abs/10.1080/00207543.2014.916429#.VVvCJc8n_IU

Радови у међународним часописима (М23)

1. Vasiljevic, D., **Danilovic**, **M**. (2013). A Novel Linear Algorithm for Shortest Paths in Networks. *Asia-Pacific Journal of Operational Research*, **30**(2), 1250054-1–1250054-25, ISSN: 0217-5959 (IF2014: 0.346), DOI: 10.1142/S0217595912500546

http://www.worldscientific.com/doi/abs/10.1142/S0217595912500546

2. Vasiljevic, D., Trkulja, Z., **Danilovic**, **M**. (2014). Towards an extended set of production line performance indicators. *Total Quality Management and Business Excellence*, **25**(5-6), 618-634, ISSN: 1478-3363 (IF2014: 1.323), DOI: 10.1080/14783363.2013.850811
http://www.tandfonline.com/doi/abs/10.1080/14783363.2013.850811

Радови објављени у зборницима међународних научних скупова (М30)

Саопштења са међународних скупова штампана у целини (М33)

1. **Danilovic, M**., Ilic, O. (2012). Mathematical Models and Techniques for Quadratic Assignment Problem, *XIII International Symposium SymOrg 2012, "Innovative management and business performance",* Symposium proceedings, 1320-1327, ISBN 978-86-7680-295-1, Book of Abstracts, Faculty of Organizational Sciences, University of Belgrade, Zlatibor, Serbia.
http://www.symorg.fon.bg.ac.rs/download/Program_SymOrg2012.pdf

2. Cvetic, B., Vasiljevic, D., **Danilovic, M.** (2013). DRP Game: New tool to enhance teaching and learning in logistics and supply chain management, *1st Logistics International Conference*, 299-303, University of Belgrade, Faculty of Transport and Traffic Engineering, Belgrade, Serbia.
http://logic.sf.bg.ac.rs/wp-content/uploads/2013/11/LOGIC-2013-Preliminary-program-15.11.2013.pdf

3. **Danilovic, M**., Ilic, O., Cvetic, B. (2014). MRP for Complex Assembly Systems Under Environmental Uncertainty, *XIV International Symposium SymOrg 2014, „New Business Models and Sustainable Competitiveness",* Symposium proceedings, 1223-1230, ISBN 978-86-7680-254-8, Book of Abstracts, Faculty of Organizational Sciences, University of Belgrade, COBISS.SR-ID 191187980, Zlatibor, Serbia.
http://symorg.fon.bg.ac.rs/proceedings/papers/18%20-%20OPERATIONS%20MANAGEMENT.pdf

4. **Danilovic, M**., Ilic, O. (2016). A novel algorithm for combinatorial problem in manufacturing cell formation,, *XV International Symposium SymOrg 2016, „Reshaping the future through sustainable business development and entrepreneurship ",* Symposium proceedings, 954-961, ISBN 978-86-7680-326-2, Book of Abstracts, Faculty of Organizational Sciences, University of Belgrade, COBISS.SR-ID 223988236, Zlatibor, Serbia.
http://symorg.fon.bg.ac.rs/proceedings/2016/papers/OPERATIONS%20MANAGEMENT.pdf

Радови објављени у часописима националног значаја (М50)

Рад у часопису националног значаја (М52)

1. Jaćimović, D., Vasiljević, D., **Danilović, M**., Veković, J. (2013). One approach to risk management modeling: A case study. *Facta Univesitatis: Series Economics and Organization*, **10**(4), 377 - 387, ISSN 0354 – 4699, UDC 005:330.131.7.
   http://facta.junis.ni.ac.rs/eao/eao201304/eao201304-04.pdf

Радови објављени у зборницима скупова националног значаја (М60)

Саопштења са скупова националног значаја штампана у целини (М63)

1. **Даниловић, М.**, Илић, О., (2011). Алгоритми за решавање проблема квадратне асигнације, *YU INFO 2011, Конференција о рачунарским наукама и информационим технологијама*, Копаоник, 06.-09.03.2011. године, Зборник радова, стр. 1-6, ISBN: 978-86-85525-08-7.
   http://www.yuinfo.org/zbornici/2011/html/pdf/024.pdf

2. **Даниловић, М.**, (2011). Примена алгоритама за најкраће путање у мрежама у проблемима вишеетапног процеса управљања, *SYM-OP-IS 2011, XXXVIII Симпозијум о операционим истраживањима*, Златибор, 04.-07.10.2011. године, Зборник радова, стр. 238-241, ISBN: 978-86-403-1168-7

3. **Даниловић, М.**, Илић, О., (2011). Генетски алгоритми за решавање проблема квадратне асигнације, *SPIN 2011, VIII Скуп привредника и научника*, Привредна комора Србије у Београду, 01.-02.11.2011. године, Зборник радова, стр. 539-546, ISBN: 978-86-7680-244-9
   http://www.spin.fon.bg.ac.rs/doc/ret/SPIN%202011/Sekcije/12kvantitativne%20metode%20i%20modeli%20u%20menadzm-pdf/1201_GENETSKI%20ALGORITMI%20ZA%20RE%C5%A0AVANJE%20PROBLEMA%20KVADRATNE%20ASIGNACIJE.pdf

4. Ђорђевић, Л., **Даниловић, М.**, Васиљевић, Д., (2011). Примена софтвера LOGWARE у едукацији менаџера логистике, *SPIN 2011, VIII Скуп привредника и научника*, Привредна комора Србије у Београду, 01.-02.11.2011. године, Зборник радова, стр. 358-365, ISBN: 978-86-7680-244-9
   http://www.spin.fon.bg.ac.rs/doc/ret/SPIN%202011/Sekcije/07logistika%20i%20lanci%20snabdevanja-pdf/705_PRIMENA%20SOFTVERA%20LOGWARE%20U%20EDUKACIJI%20MENAD%C5%BDERA%20LOGISTIKE.pdf

5. **Даниловић, М.**, Илић, О., (2012). Алгоритам прорачуна временског размака производње за проблем редоследа у проточној радионици, *YU INFO 2012, , XVIII Конференција о рачунарским наукама и информационим технологијама*, 29.02.-03.03.2012. године, Копаоник, Зборник радова, стр. 173-178, ISBN: 978-86-85525-09-4 http://www.yuinfo.org/zbornici/2012/html/pdf/376.pdf

6. **Даниловић, М.**, (2012). Генерализација НЕХ хеуристике у пермутационим flowshop проблемима, *SYM-OP-IS 2012, XXXIX Симпозијум о операционим истраживањима*, Тара, 25.-28.09.2012. године, Зборник радова, стр. 311-314, ISBN: 978-86-7488-086-9 http://symopis.vggs.rs/files/PROGRAM_RADA_SYMOPIS_2012.pdf

7. **Даниловић, М.**, Илић, О., (2013). Нова формализација и проширење фазе уметања у НЕХ хеуристици, *YU INFO 2013, XIX Конференција о рачунарским наукама и информационим технологијама*, 03-06. 03. 2013, Копаоник, Зборник радова – ЦД, стр. 304-309, ISBN: 978-86-85525-11-7 http://www.yuinfo.org/zbornici/2013/html/pdf/686.pdf

8. **Даниловић, М.**, Илић, О., (2014). Примена генерализоване конструктивне хеуристике на пермутациони *flowshop* проблем, *YU INFO 2014, XX научно-стручна и бизнис конференција*, 09-13.03.2014, Копаоник, Зборник радова, стр. 195-199, ISBN: 978-86-85525-13-1 http://www.yuinfo.org/YUINFO%202014%20zbornik.pdf

9. **Даниловић, М.**, Илић, О., (2015). Нови приступ решавању пермутационог flowshop проблема, *YU INFO 2015, XXI научно-стручна и бизнис конференција*, 08.-11.03.2015, Копаоник, Зборник радова, стр. 290-295, ISBN: 978-86-85525-15-5 http://www.yuinfo.org/YUINFO%202015%20zbornik.pdf

10. Цветић, Б., Васиљевић, Д., **Даниловић, М.**, (2015). Компетенције менаџера логистике и ланца снабдевања у републици Србији, *SPIN 2015, X Скуп привредника и научника*, 05.-06.11.2015. године, Привредна комора Србије у Београду, Зборник радова, стр. 140-147, ISBN: 978-86-7680-320-0 http://spin.fon.bg.ac.rs/wp-content/uploads/2015/11/SPIN15 Zbornik radova Proceedings.

11. **Даниловић, М.**, Илић, О., (2016). Генерализовани конструктивни алгоритам за формирање производних ћелија, *YU INFO 2016, XXII научно-стручна и бизнис конференција*, 28.02.-02.03.2016. год, Копаоник, Зборник радова, стр. 169-174, ISBN: 978-86-85525-17-9, Друштво за информационе системе и рачунарске мреже, Београд. http://yuinfo.artkey.rs/zbornici/2016/YUINFO2016.pdf

Рецензије и цитираност

У досадашњем раду су успешно обављене рецензије у следећим научним часописима међународног значаја (М20):

- Computers and Operations Research – успешно обављене четири рецензије и добијен цертификат (Recognized Reviewer) у септембру 2015,

- International Journal of Production Research – успешно обављене три рецензије,
- Expert Systems with Applications - успешно обављене три рецензије.

Објављени радови су до сада цитирани укупно 10 пута.

Изјава о ауторству

Име и презиме аутора: **Милош Даниловић**

Број индекса: **5008/2011**

Изјављујем

да је докторска дисертација под насловом

УНАПРЕЂЕЊЕ КОНСТРУКТИВНИХ ХЕУРИСТИКА ЗА ПРОБЛЕМЕ КОМБИНАТОРНЕ ОПТИМИЗАЦИЈЕ У ОПЕРАЦИОНОМ МЕНАЏМЕНТУ

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

Потпис аутора

У Београду, 24.03.2017.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора: **Милош Даниловић**
Број индекса: **5008/2011**
Студијски програм:
Информациони системи и менџмент

Наслов рада:

Унапређење конструктивних хеуристика за проблеме комбинаторне оптимизације у операционом менаџменту

Ментор: Проф. др Оливер Илић

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао ради похрањења у **Дигиталном репозиторијуму Универзитета у Београду.**

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

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

Потпис аутора

У Београду, 24.03.2017.

_____

Изјава о коришћењу

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

Унапређење конструктивних хеуристика за проблеме комбинаторне оптимизације у операционом менаџменту

која је моје ауторско дело.

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

Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио.

1. Ауторство (CC BY)
2. Ауторство – некомерцијално (CC BY-NC)
3. **Ауторство – некомерцијално – без прерада (CC BY-NC-ND)**
4. Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)
5. Ауторство – без прерада (CC BY-ND)
6. Ауторство – делити под истим условима (CC BY-SA)

(Молимо да заокружите само једну од шест понуђених лиценци.
Кратак опис лиценци је саставни део ове изјаве).

Потпис аутора

У Београду, 24.03.2017.

_____

1. **Ауторство**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. **Ауторство – некомерцијално**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. **Ауторство – некомерцијално – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. **Ауторство – некомерцијално – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. **Ауторство – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. **Ауторство – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.