УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Анђела Р. Вучинић

МАТХЕУРИСТИЧКИ ПРИСТУП ЗА РЕШАВАЊЕ ПРОБЛЕМА ПРАВОВРЕМЕНОГ РАСПОРЕЂИВАЊА ПОСЛОВА ПО МАШИНАМА У ВИШЕФАЗНОЈ ПРОИЗВОДЊИ

мастер рад

Београд, 2024.

Ментор:

проф. др Зорица Дражић, ванредни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

проф. др Зорица Станимировић, редовни професор
Универзитет у Београду, Математички факултет

проф. др Александар Савић, ванредни професор
Универзитет у Београду, Математички факултет

Датум одбране: Септембар 2024.

Овај рад посвећујем својој породици...
Највећу захвалност дугујем мојој драгој менторки на несебичној подршци и стрпљењу.

Наслов мастер рада: Матхеуристички приступ за решавање проблема правовременог распоређивања послова по машинама у вишефазној производњи

Резиме: Проблеми распоређивања спадају у групу неких од најраспрострањенијих проблема комбинаторне оптимизације. Једну њихову класу чине проблеми распоређивања извршавања n послова на m машина при одређеним условима и ограничењима, који уједно имају и велику практичну примену, а често се могу срести као саставни део процеса оперативног планирања производње, нарочито у многобројним индустријским постројењима за производњу различитих сировина. Овај рад се бави конкретним проблемом из ове класе - проблемом правовременог распоређивања послова по машинама у вишефазној производњи (*engl. JUST-IN-TIME JOB-SHOP SCHEDULING PROBLEM (JIT-JSSP)*). Он подразумева да се сваки посао састоји од уређеног скупа различитих операција које се морају извршити по унапред задатом редоследу на одговарајућим машинама, а да притом свака машина може извршавати највише једну операцију истовремено. Поред тога, за сваку операцију задато је време потребно да се она реализује, као и временски рок када је најбоље да буде завршена. Свако одступање од идеалног времена завршетка резултира казном. Укупна казна једнака је збиру казни за превремени завршетак или за кашњење за све операције свих послова. Задатак је да се одреди такав редослед извршавања операција свих послова на датим машинама да укупна казна буде минимална. JIT-JSSP је решаван матхеуристиком која представља хибрид између методе променљивих околина (*engl. VARIABLE NEIGHBORHOOD SEARCH - VNS*) и егзактног CPLEX решавача. Главна идеја ове хибридизације је декомпозиција полазног проблема на потпроблеме који се потом решавају независно, и то тако што се најпре одреди редослед извршавања свих операција за све послове применом VNS методе, а затим, за такав редослед, применом егзактног решавача одређује се време завршетка за сваку операцију појединачно, са циљем да укупна казна буде што је могуће мања. Предложена матхеуристика је тестирана на 72 инстанце различитих димензија, а добијени резултати упоређени су са резултатима доступним у литератури.

Кључне речи: JIT-JSSP, проблем распоређивања, правовремено распоређивање, вишефазна производња, метода променљивих околина, VNS, CPLEX, матхеуристика

Master's thesis title: Matheuristic Approach to Solving Just-In-Time Job-Shop Scheduling Problem

Abstract: Scheduling problems are some of the most common combinatorial optimization problems. One class of them comprises problems of scheduling the performance of n jobs on m machines under specific conditions and limitations, which at the same time have wide application in practice and are often integral to the operational production planning process, especially in industrial raw material production plants. This paper addresses a specific problem from this class - *the Just-In-Time Job-Shop Scheduling Problem (JIT-JSSP)*. It implies that each job consists of an ordered set of different operations that must be performed in a preset sequence on adequate machines, with each machine performing one operation at a time at most. In addition, the time needed for each operation to be performed and the most favorable due date to complete the operation is set. Any deviation from the due date results in a penalty. The total penalty is the sum of earliness or tardiness penalties for all job operations. The task is to determine the sequence of operations for all jobs on given machines that would minimize the total penalty. JIT-JSSP has been solved by a matheuristic algorithm, a hybrid of Variable Neighborhood Search (VNS) and the exact CPLEX solver methods. The main idea of this hybridization is to decompose the initial problem into subproblems that are then solved independently, first by determining the sequence of all operations for all jobs by applying the VNS method and then, for such a sequence, determining the completion time for each operation individually by using the exact solver with the aim of making the total penalty as low as possible. The proposed matheuristic approach was tested on 72 instances of different dimensions, and the obtained results were compared with those available in the literature.

Keywords: JIT-JSSP, scheduling problem, just-in-time scheduling, multiphase production, variable neighborhood search, VNS, CPLEX, matheuristics

Садржај

1 Увод **1**

1.1 Проблем оптимизације 1

1.2 Класификација проблема оптимизације 3

1.3 Методе за решавање проблема оптимизације 5

1.4 Метахеуристике 7

1.4.1 Метода променљивих околина 15

1.5 Матхеуристике 21

2 Проблем правовременог распоређивања послова по машинама у вишефазној производњи **25**

2.1 Преглед и дефиниција JIT-JSSP 27

2.2 Математичка формулација JIT-JSSP 28

2.3 Теоријске и рачунске границе за JIT-JSSP 31

2.4 Илустративни пример проблема 35

2.5 Преглед релевантне литературе 37

3 Матхеуристика за решавање JIT-JSSP **41**

3.1 Репрезентација решења 41

3.2 Метода променљивих околина 43

3.2.1 Иницијализација почетног решења 44

3.2.2 Дефиниција околина 45

3.2.3 Фаза размрдавања 46

3.2.4 Фаза локалне претраге 48

3.3 Решавање потпроблема CPLEX решавачем 52

3.4 Структура алгоритма 53

4 Експериментални резултати **55**

4.1 Тест примери - опис 55
4.2 Презентација добијених резултата 57
4.3 Поређење са резултатима из литературе 66

5 Закључак **73**

Библиографија **75**

Глава 1

Увод

Да је математика учитељица живота може се чути често, на најразличитијим местима и у најразличитијим ситуацијама. Поред тога што је понудила одговоре на прегршт недоумица из многих научних дисциплина, уколико се њени потенцијали искористе на прави начин, свакоме може бити водиља и кроз свакодневицу. Доношење најбољих одлука у економији, правилна конструкција бродова, авиона, најоптималније распоређивање семафора, стабилна изградња мостова и још много сличних проблема који су од немерљиве битности за функционисање глобалног система у ком живимо темеље се на математичким принципима и своја решења траже у њеним законима. Стога, можемо рећи да је математика апстракција живота.

Животни проблеми описани математичким језиком представљају математичке моделе. Један модел није еквивалент једном проблему, већ целој класи њему сличних. Тако на пример, систем пошта Србије тежи да све пошиљаке испоручи на одговарајуће адресе са минималним утрошком ресурса, или многе компаније желе да распореде своје запослене тако да од њих добију највећу могућу продуктивност. С друге стране, мрави увек траже најкраћи пут до извора хране, па и многи физички системи теже да дођу у такво стање које ће им захтевати минималну енергију. Све описане ситуације спадају у проблеме оптимизације, а детаљније о њима изложено је у [48, 93].

1.1 Проблем оптимизације

Задатак оптимизације представља проналажење најбољег решења неког проблема, а да притом буду задовољени одређени услови. У зависности од тога да

ли је потребно одредити максималну или минималну вредност задате функције, једнокритеријумски оптимизациони проблеми се деле на проблеме максимизације и проблеме минимизације.

Дефиниција 1.1. *Нека је дата функција $f : S \to R$ и нека је $X \subseteq S$. Тада:*

- *Тачка $x^* \in X$ је **тачка глобалног минимума** функције f на скупу X ако важи да је $f(x^*) \leq f(x)$, за свако $x \in X$. Вредност $f(x^*)$ је глобални минимум фукције f на скупу X.*
- *Тачка $x^* \in X$ је **тачка глобалног максимума** функције f на скупу X ако важи да је $f(x^*) \geq f(x)$, за свако $x \in X$. Вредност $f(x^*)$ је глобални максимум фукције f на скупу X.*
- *Тачка $x^{'} \in X$ је **тачка локалног минимума** функције f на скупу X ако постоји $\varepsilon > 0$ такво да је $f(x^*) \leq f(x)$, за свако $x \in X \cap N_\varepsilon(x^{'})$, где је $N_\varepsilon(x^{'}) = \{x \in S \mid d(x^{'}, x) < \varepsilon\}$ ε-околина тачке $x^{'}$ у некој метрици d задатој у простору X. Вредност $f(x^{'})$ је локални минимум функције f на скупу X.*
- *Тачка $x^{'} \in X$ је **тачка локалног максимума** функције f на скупу X ако постоји $\varepsilon > 0$ такво да је $f(x^*) \geq f(x)$, за свако $x \in X \cap N_\varepsilon(x^{'})$, где је $N_\varepsilon(x^{'}) = \{x \in S \mid d(x^{'}, x) < \varepsilon\}$ ε-околина тачке $x^{'}$ у некој метрици d задатој у простору X. Вредност $f(x^{'})$ је локални максимум функције f на скупу X.*

Глобални минимум и максимум се једним именом називају глобални оптимум, респективно важи и за локални оптимум. Сваки глобални оптимум је уједно и локални.

Проблем минимизације се може дефинисати на следећи начин:

$$\min f(x) \tag{1.1}$$

при ограничењима:

$$x \in X,\ X \subseteq S, \tag{1.2}$$

где је S простор претраге, $X \subseteq S$ простор допустивих решења кога чине они елементи из S који задовољавају дати скуп услова, а f функција циља. Елементи скупа X зову се допустива решења.

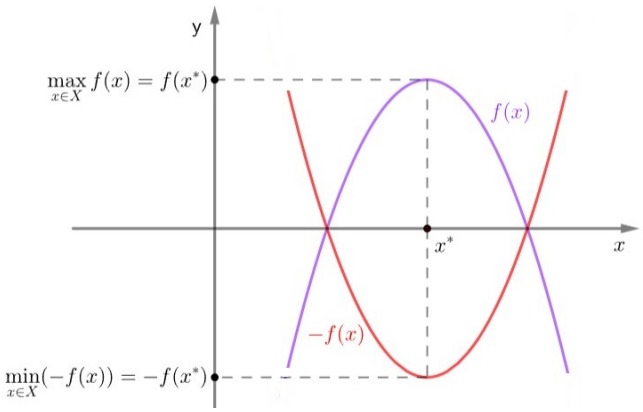
Оваква дефиниција проблема (1.1)-(1.2), изложена у [11, 109], захтева проналажење глобалног минимума које се још назива и оптимално решење. Глобални

минимум не мора бити јединствен. У пракси налажење свих глобалних минимума може бити веома тежак задатак, те је често довољно пронаћи само једно од њих.

Проблем максимизације се обично може свести на проблем минимизације, тј.

$$\max_{x \in X} f(x) = -\min_{x \in X}(-f(x)), \tag{1.3}$$

те је углавном довољно разматрати само поступак одређивања минимума функције при задатим условима.



Слика 1.1: Илустрација тврђења $\max_{x \in X} f(x) = -\min_{x \in X}(-f(x))$ на примеру квадратне функције

1.2 Класификација проблема оптимизације

У зависности од природе скупа S, оптимизациони проблеми се могу поделити на следећи начин:

- Ако је S коначан или бесконачно пребројив скуп, онда проблем (1.1)-(1.2) припада класи **комбинаторне**, односно **дискретне оптимизације**. Више о овој групи проблема може се наћи у [94].
- Ако је S непребројив скуп, онда проблем (1.1)-(1.2) припада класи **континуалне оптимизације**. Више о овој групи проблема дато је у [103].

У средини велике поделе оптимизационих проблема на оне који припадају класи дискретне или класи континуалне оптимизације, налази се класа проблема **мрежне оптимизације**. Ова група проблема има велику практичну примену у савременом друштву, а опширније о њој може се прочитати у [18].

На основу скупа X сви проблеми оптимизације могу се поделити у две групе:

- Уколико је $X = S$, тј. простор претраге S је уједно и простор допустивих решења, онда (1.1)-(1.2) дефинише **проблем безусловне оптимизације**.
- Иначе, ако $X \neq S$, тј.$X \subset S$, онда (1.1)-(1.2) дефинише **проблем условне оптимизације**.

Више о овим проблемима може се наћи у [61].

У односу на својства и тип функције циља f и задатих ограничења, проблеми оптимизације се деле у две групе:

- Ако су функција циља f и сва ограничења задата у линеарном облику, онда се проблем (1.1)-(1.2) назива **проблемом линеарног програмирања** или **линеарне оптимизације**.
- Иначе, ради се о **проблему нелинеарног програмирања** или **нелинеарне оптимизације**. Више о овој класи проблема може се наћи у [110].

Проблем линеарног програмирања (*engl. LINEAR PROGRAMMING - LP*) ком је додат услов да све координате вектора $x \in X$ морају бити целобројне назива се **проблем целобројног линеарног програмирања** (*engl. INTEGER LINEAR PROGRAMMING - ILP*). Уколико услов целобројности није задат за све координате вектора $x \in X$, већ само за неке од њих, онда је задат **проблем мешовитог целобројног линеарног програмирања** (*engl. MIXED INTEGER LINEAR PROGRAMMING - MILP*). Опширније о овој теми може се прочитати у [102].

Алгоритми за решавање оптимизационих проблема могу се поделити на основу времена које им је потребно да се изврше, и то:

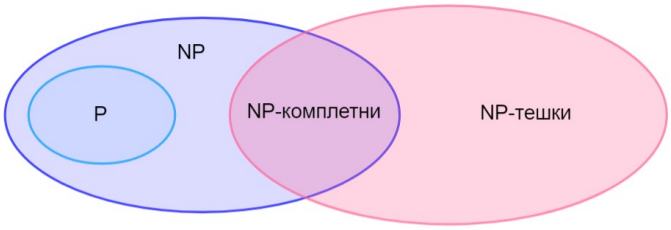
- Алгоритам који се завршава за не више од $p(n)$ корака, где је p нека полиномска функција, а n димензија проблема[1], је **полиномске сложености**, а за посматрани проблем каже се да је решив у полиномском времену.
- Иначе, ради се о алгоритму који је **експоненцијалне сложености**, а за посматрани проблем каже се да је решив у експоненцијалном времену.

Проблеми за чије решавање су развијени алгоритми полиномске сложености сматрају се релативно лаким за решавање и сврставају се у тзв. **P** класу сложености. **NP** класу сложености чини скуп проблема одлучивања[2] који се недетер-

[1]Време извршавања алгоритма и обим улазних података за одговарајући проблем описани су једним природним бројем n, који се назива димензија проблема.

[2]Проблем одлучивања представља произвољно питање у неком формалном систему које враћа одговор „ДА" или „НЕ".

министичким алгоритмом[3] могу решити у полиномском времену. Проблем који припада NP класи сложености је **NP-комплетан** ако се сви остали проблеми из NP класе сложености у полиномском времену могу свести на њега. Ако постоји детерминистички алгоритам[4] којим се у полиномском времену може решити NP-комплетан проблем, онда се и сви остали проблеми из NP класе сложености, такође, могу решити у полиномском времену. Проблем оптимизације је **NP-тежак** ако је њему одговарајући проблем одлучивања NP-комплетан. Већина оптимизационих проблема који се могу срести у пракси припадају класи NP-тешких и до сада није доказано да постоји ефикасан алгоритам за њихово решавање, већ се најчешће решавају алгоритмима експоненцијалне сложености. Опширније о овим класама може се наћи у [10, 23, 73, 80].



Слика 1.2: Класе сложености - Венов дијаграм

1.3 Методе за решавање проблема оптимизације

Неке од најзаступљенијих техника којима се могу решити различити проблеми оптимизације су:

- Егзактне методе (*engl. EXACT METHODS*);
- Приближне методе (*engl. APPROXIMATE METHODS*).

Задатак егзактне или тачне методе јесте да пронађе оптимално решење, уз које мора да обезбеди доказ оптималности, или да покаже да оно не постоји. Најчешће се користе код проблема малих димензија, као и код оних за чије решавање су развијени алгоритми полиномске сложености. Различита ограничења, као што су расположиво време за које је потребно решити проблем,

[3]Недетерминистички алгоритам за исти улаз пратећи дефинисан низ инструкција може дати различите резултате у различитим покретањима. То се најчешће постиже коришћењем технике случајног одабира или техником погађања.

[4]Детерминистички алгоритам за исти улаз пратећи прецизно дефинисан низ инструкција даје исти резултат ма колико пута био покренут.

меморијски капацитети рачунара, то што за посматрани проблем не постоји математички модел или је превише сложен, и слично, онемогућавају њихову ширу примену. Неке од најпознатијих метода из ове групе су:

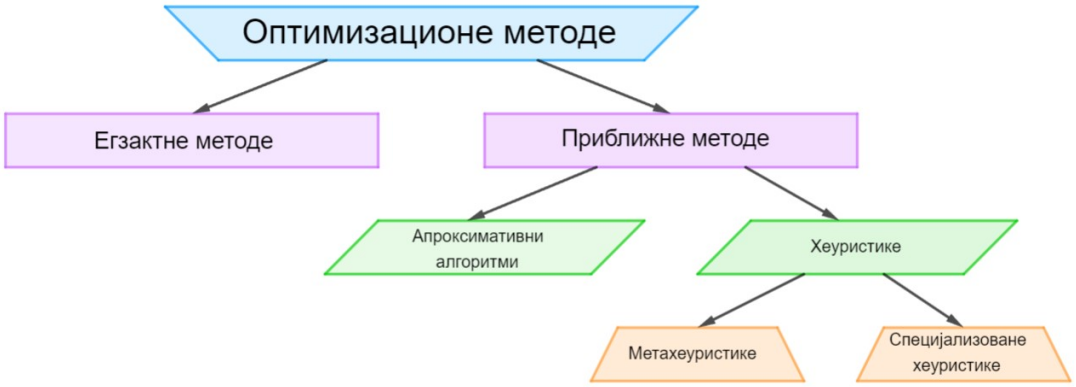
- Симплекс метода (*engl. SIMPLEX METHOD*);
- Метода гранања и ограничавања (*engl. BRANCH AND BOUND*);
- Метода имплицитног пребројавања (*engl. IMPLICIT FILTERING METHOD*);
- Метода гранања и одсецања (*engl. BRANCH AND CUT*);
- Гоморијева метода одсецања равни (*engl. GOMORY'S CUTTING-PLANE METHOD*);
- Метода гранања и вредновања (*engl. BRANCH AND PRICE*).

Многе егзактне методе су имплементиране унутар различитих математичких софтвера као што су: *CPLEX*, *Lingo*, *Gurobi* и други. Детаљније о егзактним методама оптимизације може се наћи у [111].

Проблеми који се не могу решити егзактним методама, углавном се решавају приближним методама. Ове методе се могу поделити у две класе и то: на апроксимативне алгоритме (*engl. APPROXIMATION ALGORITHMS*) и на хеуристичке (*engl. HEURISTIC ALGORITHMS*). Апроксимативни алгоритми се у највећој мери користе за одређивање што ближих доњих и горњих граница оптималног решења, на основу којих се касније може доказано проценити квалитет решења. Поред тога, обезбеђују доказиве границе времена извршавања алгоритма, а и дају увид у сложеност проблема, што омогућава креирање ефикасних хеуристика за њихово решавање. Једна од мана им је то што немају општу примену, већ се имплементација прилагођава конкретном проблему, и у већини случајева се не може користити за решавање неког другог проблема. Са друге стране, у пракси се показало да вредности које се добију апроксимативним алгоритмима често могу бити доста далеко од оптималних решења.

Хеуристике, за разлику од апроксимативних алгоритама, не нуде доказ о ваљаности решења, а за разлику од егзактних метода, не гарантују оптималност, али у пракси се често дешава да дају решења високог квалитета, неретко и оптимална. Оне су релативно брзе, потребни су им мањи меморијски капацитети и погодне су за решавања проблема различитих димензија, а и сложености. Једна од особина која их препоручује за коришћење јесте то што праве баланс између квалитета решења и времена извршавања алгоритма. Настале су сре-

дином осамдесетних година, а до данас, достигле су велику популарност што је интензивирало теоријска истраживања којима се оправдава њихова ефикасност. Могу се поделити у две групе: на специјализоване хеуристике (*engl. PROBLEM SPECIFIC HEURISTICS*) и на метахеуристике (*engl. METAHEURISTICS*). Специјализоване хеуристике немају универзалну примену, већ се дизајнирају према карактеристикама конкретаног проблема, и најчешће се не могу користити за решавање неког другог. За разлику од њих, метахеуристике су алгоритми опште намене и могу се применити за решавање широког спектра оптимизационих проблема. На слици 1.3 приказана је једна од подела метода којима се решавају проблеми оптимизације, а више о њима може се наћи у [23, 106].



Слика 1.3: Методе за решавање оптимизационих проблема

1.4 Метахеуристике

Многобројни проблеми из свакодневног живота припадају класи NP-тешких, а за њихово решавање углавном се бирају метахеуристике. Оне су настале пре више од тридесет година, али су одржале своју атрактивност, те се све већи број научника бави истраживањем ове групе метода. Користе се код проблема различитих димензија, а и сложености. Главна предност им је то што су засноване на апстрактним универзалним правилима, па се са мало адаптације могу применити за решавање великог броја проблема оптимизације. Извршавају се итеративно, све до испуњења унапред задатог критеријума заустављања. Ослањају се на два концепта: на диверзификацију која има задатак да претрагу рашири што је могуће више унутар простора претраживања и на интензификацију која тежи да унутар простора претраге детаљно истражи мање околине око текућег решења. Једна од најчешће коришћених подела је на метахеу-

ристике које су засноване на итеративном побољшању једног решења, тзв. **S-метахеуристике** (*engl. SINGLE-SOLUTION BASED*) и оне засноване на итеративном побољшању скупа решења (популације), а зову се **P-метахеуристике** (*engl. POPULATION BASED*).

Неке од најпознатијих S-метахеуристика су:

- Локална претрага;
- Похлепна стохастичко-адаптивна процедура претраге;
- Табу претрага;
- Симулирано каљење;
- Метода промељивих околина.

Неке од најпознатијих P-метахеуристика су:

- Генетски алгоритам;
- Оптимизацијаројем честица;
- Оптимизација колонијом пчела;
- Оптимизација мрављом колонијом.

Поред ове поделе, метахеуристике се могу класификовати и на основу других критеријума: алгоритми инспирисани процесима из природе или они који нису инспирисани њима, метахеуристике које користе меморију за складиштење добијених резултата током претраге или оне које је не користе, итеративни или похлепни (конструктивни) алгоритми, као и детерминистички или стохастички. Детаљније о овој групи метода може се наћи у [23, 29], а у наставку биће презентован кратак преглед неких од најпознатијих њених представника.

Локална претрага (*engl. LOCAL SEARCH - LS*) спада у најстарије и по својој структури најједноставније метахеуристике. Њена главна идеја се огледа у итеративном претраживању различитих околина допустивог, са циљем проналажења бољег, по могућству оптималног решења. На почетку, непоходно је одабрати репрезентацију решења као и начин на који ће се креирати његове околине. Алгоритам започиње генерисањем иницијалног решења x, најчешће на случајан начин, и његовим постављањем за најбоље, тј. $x^* := x$, $f^* := f(x)$. Потом, поступак се итеративно понавља до испуњења унапред задатог критеријума заустављања, тако што се у сваком кораку претражује околина текућег решења $N(x^*)$ и разматра један од два могућа случаја. Први, када има побољ-

шања, тј. када претрага дође до $x^{'} \in N(x^{*})$ за које, код проблема минимизације, важи да је $f(x^{'}) < f^{*}$. Тада се најбоље решење ажурира, тј. $x^{*} := x^{'}$, $f^{*} := f(x^{'})$, и прелази се у следећу итерацију. Друга могућност је да претрага не нађе $x^{'} \in N(x^{*})$ за које важи да је $f(x^{'}) < f^{*}$. Тада се тренутно најбоље решење узима за апроксимацију оптималног, а локална претрага се прекида.

Недостатак ове методе је то што се неретко дешава да дође до заглављивања у локалном оптимуму, што отежава или потпуно онемогућава долазак до глобалног. Из тог разлога, све чешће се користи као део неке друге метахеуристике ради појачавања диверзификације унутар саме претраге. С циљем побољшања перформанси, развијене су и неке њене модификације, као што су: мултистартна локална претрага (*engl. MULTISTART LOCAL SEARCH - MLS*), итеративна локална претрага (*engl. ITERATED LOCAL SEARCH - ILS*), и друге. Детаљније о овој методи може се наћи у [21].

Похлепна стохастичко-адаптивна процедура претраге (*engl. GREEDY RANDOMIZED-ADAPTIVE SEARCH PROCEDURE - GRASP*) спада у групу конструктивних метахеуристика. За генерисање решења поред локалне претраге користи се и насумично-похлепни алгоритам (*engl. GREEDY-RANDOMIZED CONSTRUCTION - GRC*) за чије извршавање је неопходно дефинисати листу кандидата (*engl. CANDIDATE LIST - CL*) у којој су смештени елементи који у одговарајућој итерацији могу постати део решења, као и редуковану листу кадидата (*engl. RESTRICTIVE CANDIDATE LIST - RCL*) која је подскуп листе кандидата и настаје тако што се елементима листе кандидата доделе оцене $o(i)$, $i \in CL$, на основу којих се мери њихов утицај на вредност функције циља и доноси се одлука да ли могу постати део редуковане листе или не. Поред тога, за потребе локалне претраге неопходно је дефинисати околине $N(x) \subseteq X$ допустивог решења x.

На самом почетку GRC алгоритмом се конструише почетно решење на претходно одабран начин. У свакој GRC итерацији бира се по један елемент који ће постати део решења, све док се не конструише комплетно решење. Након сваког избора елемента, ажурирају се CL и RCL. Решење које је конструисано у GRC фази подлеже процедури локалног претраживања. Достигнути локални оптимуми из сваке итерације се чувају. Поступак се завршава када се испуни унапред изабрани критеријум заустављања, а оптимум који има најбољу вредност функције циља бира се за апроксимацију оптималног решења. Више о овој методи дато је у [69, 99].

Табу претрага (*engl. TABU SEARCH - TS*) је први пут предложена у [28] и спада у групу метахеуристика које су засноване на локалном претраживању и користе меморију. Специфичност ове методе огледа се у креирању тзв. табу листе, која је најчешће константне, а може бити и променљиве дужине. У њој се краткорочно чувају информације о достигнутим оптимумима, са циљем да се избегне заглављивање алгоритма у локалном оптимуму, што представља главнуману локалне претраге.

Алгоритам почиње генерисањем иницијалног решења x претходно одабраном процедуром, које се уједно проглашава за најбоље, тј. $x^* := x$, $f^* := f(x)$. Затим, на унапред дефинисан начин врши се локална претрага околине $N(x) \subseteq X$ која као резултат даје ново допустиво решење $x^{'}$, а оно се потом додаје у до тада празну табу листу и на тај начин у наредних неколико итерација бива искључено из околине која се претражује. Проверава се квалитет добијеног решења, тј. код проблема минимизације, проверава се да ли важи да је $f(x^{'}) < f^*$. Уколико је ова неједнакост задовољена, чува се најбоље нађено решење, тј. $x^* := x^{'}$, $f^* := f(x^{'})$. Затим $x := x^{'}$, а локална претрага модификоване околине се наставља, табу листа се ажурира, а решења која се налазе у њој након извесног броја итерација, тзв. табу времена, поново бивају враћена у процес претраге. Поступак се понавља док се не испуни већ одабрани критеријум заустављања. Уколико је потребно смањити меморијске захтеве, онда се, често, уместо целокупног решења чувају подаци о поступцима који до њега доводе. Развијене су многе модификације ове методе, а у некима од њих се поред краткорочне користи и средњорочна и дугорочна меморија.

Симулирано каљење (*engl. SIMULATED ANNEALING - SA*) спада у групу метахеуристика које су инспирисане процесима из природе. Засновано је на локалном претраживању, а мотивација за развој ове методе пронађена је у алгоритмима статичке термодинамике развијеним за опонашање процеса каљења челика. Специфичност која је разликује од већине других метахеуристика јесте то што са извесном вероватноћом прихвата и лошија решења од већ нађених, а са циљем доласка до глобалног оптимума.

На самом почетку врши се генерисање иницијалног решења x претходно одабраном процедуром, и проглашава се за најбоље, тј. $x^* := x$, $f^* := f(x)$. Потом, на случајан начин бира се $x^{'}$ из унапред дефинисане околине текућег решења $N(x)$ и проверава се његов квалитет. Ако је $x^{'}$ боље од x^*, тј. код проблема минимизације, ако важи да је $f(x^{'}) < f^*$, онда се $x^{'}$ прихвата као

најбоље решење и ажурирају се потребне вредности, тј. $x^* := x^{'}$, $f^* := f(x^{'})$, $x := x^{'}$. Уколико ова неједнакост није испуњена, тада се $x^{'}$ прихвата као текуће решење ($x := x^{'}$) са одређеном вероватноћом p_n, која може бити задата на различите начине, а један од најчешћих је онај који зависи од температуре T_n и степена деградације лошијег решења δ_n, где је n редни број итерације, а рачуна се по формули:

$$p_n = e^{\delta_n / T_n},\ \delta_n = f(x_n^{'}) - f(x^*). \tag{1.4}$$

Температура има задатак да контролише процес претраживања, тако што изборoм веће почетне вредности овог параметра постиже се већи степен диверзификације и избегава се заглављивање у локалном оптимуму, а како алгоритам напредује, тако се и вредност температуре смањује, тј. важи да је $T_1 \geq T_2 \geq ..., \lim_{n\to\infty} T_n = 0$, те је и вероватноћа прихватања лошијег решења све мања, док се при крају готово прихватају само побољшања. Поступак се итеративно понавља све док се не испуни унапред одабрани критеријум заустављања. Више о овој методи може се наћи у [17].

Генетски алгоритам (*engl. GENETIC ALGORITHM - GA*) спада у метахеуристике које опонашају процесе из природе. Мотивација за развој ове методе лежи у имитацији еволуције, а сама идеја датира још од средине 20. века. Први пут у актуелном облику помиње се 1975. године у књизи Џона Холанда [45].

Ова метахеуристика заснована је на итеративном побољшању коначног скупа решења, тј. популације. Популација представља један подскуп простора кодираних решења, а простор кодираних решења представља скуп кодова свих допустивих решења. Допустива решења у овој метахеуристици су представљена низовима коначних дужина над неком коначном азбуком симбола, и ти низови се називају генетским кодовима тих решења. Појединачни елементи генетског кода зову се гени. Допустива решења се зову још и јединке, а њима додељени кодови хромозоми тих јединки.

Главни градивни елементи ове методе су: *оператор селекције*, који може бити дефинисан на више начина, а неки од најчешћих су пропорцијонална, рулет и турнирска селекција, *оператор укрштања*, које може бити једнопозиционо, двопозиционо, вишепозиционо, униформно и слично, и *оператор мутације*, који се такође може одабрати на више начина, а неке од најзаступљенијих су проста мутација, нормална или биномна расподела и друге. На ефикасност ове методе поред имплементације набројаних елемената, тзв. генетских оператора, утиче и начин кодирања решења. Најчешће се користи бинарно, али ако

природа проблема захтева другачије, може се одабрати и целобројно или кодирање реалним бројевима. Потом, треба дефинисати функцију прилагођености којом се оцењује квалитет јединке. То може бити вредност функције циља, скалирана вредност функције циља или нека друга особина по којој ће се јединке процењивати. На почетку врши се генерисање иницијалне популације унапред одабраном процедуром, а потом, итеративно, узастопном применом генетских оператора креирају се нове генерације јединки све док се не испуни претходно задати критеријум заустављања. Иако се очекује да потомци, наслеђујући увек најбоље од родитеља, воде ка што бољем решењу, некада се то не догоди, него се кроз процес селекције и укрштања квалитет решења изгуби. Тада се прибегава тзв. елитизму, тј. најбоље јединке из претходне генерације пуштају се непромењене у нову.

Оптимизацијаројем честица (*engl. PARTICLE SWARM OPTIMIZATION - PSO*) је метахеуристика заснована на интелигенцији групе и први пут је изложена у [47]. Главна идеја јесте да се међусобном интеракцијом јединки (честица) унутар популације најбоља искуства појединачних чланова пренесу на читав рој и тиме формира тзв. колективна интелигенција која треба претрагу да усмери ка што бољем решењу. Свака честица d-димензионог простора претраге представља једно допустиво решење и са собом носи следеће податке:

- $x_i = (x_{i_1}, x_{i_2}, ..., x_{i_d}) \leftarrow$ тренутна позиција честице;
- $p_i = (p_{i_1}, p_{i_2}, ..., p_{i_d}) \leftarrow$ најбоља позиција честице;
- $v_i = (v_{i_1}, v_{i_2}, ..., v_{i_d}) \leftarrow$ брзина кретања честице, тј. правац у ком би се кретала честица да нема других утицаја са стране;
- $p_q = (p_{q_1}, p_{q_2}, ..., p_{q_d}) \leftarrow$ најбоља позиција роја.

Алгоритам започиње генерисањем иницијалне популације и почетних вектора брзине, најчешће на случајан начин. Потом, итеративно, све док се не испуни унапред задати критеријум заустављања, врши се:

- ажурирање вектора брзине, најчешће по формули:

$$v_{ik}^j = \omega v_{ik}^{j-1} + c_1 r_1 (p_{ik}^j - x_{ik}^j) + c_2 r_2 (p_{qk}^j - x_{ik}^j),\ k = 1, ..., d \tag{1.5}$$

где је i редни број честице, j редни број итерације, ω параметар инерције који контролише утицај претходне брзине честице на садашњу, те избором већих

вредности појачава се степен диверзификације, а избором мањих врши се интензивирање претраге, параметар c_1 који контролише утицај претходног искуства честице (когнитивно учење), c_2 који контролише утицај околних честица унутар роја (социјално учење), док су r_1 и r_2 произвољне константе из опсега $[0, 1]$;

- ажурирање тренутне позиције честице, по формули:

$$x_{ik}^{j+1} = x_{ik}^{j} + v_{ik}^{j},\ k = 1, ..., d \tag{1.6}$$

где се вредност v_{ik}^{j} ограничава на сегмент $[V_{min}, V_{max}]$ како новокреирана честица не би изашла из простора допустивих решења;

- ажурирање најбоље позиције честица, као и најбоље позиције читавог роја, уколико је дошло до побољшања.

Оптимизација колонијом пчела (*engl. BEE COLONY OPTIMIZATION - BCO*) је конструктивна метахеуристика која је заснована на интелигенцији групе, а инспирисана појавама из природе. Ова метода симулира понашање пчела у потрази за храном, што представља један веома добро организован процес. Вештачке пчеле представљају допустива решења унутар BCO алгоритма који је 2001. године предложен од стране српских математичара Панте Лучића и Душана Теодоровића у [83].

Нека је са B означен број пчела, а са NC број интеракција сваке пчеле са остатком роја. Након иницијализације почетне колоније у којој су сва решења празна, наизменично се у свакој итерацији смењују два основна поступка: лет унапред, током којег на сваком кораку u, $u = 1, ..., NC$, свака пчела b, $b = 1, ..., B$, најчешће правилом рулетске селекције, доноси одлуку о свом следећем потезу, притом фаворизујући оне који воде ка бољем решењу. И лет уназад, у ком се процењују добијена решења и доноси одлука да ли ће му пчела остати лојална и добити статус регрутера или ће га напустити и остати неопредељена. Вероватноћа да пчела остане лојална свом решењу рачуна се по формули:

$$p_b^{u+1} = e^{-\frac{O_{max} - O_b}{u}}. \tag{1.7}$$

O_{max} је максимална нормализована вредност функције циља свих решења, O_b представља нормализовану вредност функције циља решења представљеног пчелом b, и за проблем минимизације рачуна се на следећи начин:

$$O_b = \frac{C_{max} - C_b}{C_{max} - C_{min}},\ b = 1, ..., B, \tag{1.8}$$

где је C_b вредност функције циља решења које одговара пчели b, а C_{min} и C_{max} минимална и максимална вредност функције циља целог роја. Уколико напусти своје решење, пчела мора да изабере једног од R регрутера кога ће пратити и то најчешће чини користећи правило рулетске селекције, а по формули:

$$p_b = \frac{O_b}{\sum_{k=1}^{R} O_k}. \tag{1.9}$$

Након направљених NC летова унапред и уназад, врши се ажурирање најбољег решења, а поступак се итеративно понавља до испуњења задатог критеријума заустављања. Детаљније о овој методи може се наћи у [84].

Оптимизација мрављом колонијом (*engl. ANT COLONY OPTIMIZATION - ACO*) спада у групу конструктивних метахеуристика и први пут је предложена 1994. године у [64]. Инспирација за њен развој нађена је у опонашању колоније мрава у потрази за храном. При кретању мрави луче хемијску супстанцу која се назива феромон и која им служи за међусобну комуникацију. Најјачи феромонски траг углавном остаје на најкраћем путу до извора хране, те тако опредељује већи део заједнице да га следи.

Имитирајући економично понашање мрављих колонија, могуће је решити многе проблеме комбинаторне оптимизације. На почетку неопходно је одредити број вештачких мрава, где сваки од њих представља једно допустиво решење, и генерисати матрицу феронома $L = [\lambda_{ij}]$ чији су елементи унапред задате иницијалне вредности феромона за сваку компоненту решења. Вредност λ_{ij} осликава ваљаност решења ако се i-та компонента нађе на j-тој позицији. Сам алгоритам пролази кроз две кључне фазе: конструкција решења и ажурирање трагова феромона. Сваки мрав се понаша као једна стохастичко похлепна процедура којом се конструише решење итеративним додавањем компоненти на основу тренутне матрице феромона. Боље решење има јачи траг феромона, те је и вероватноћа његовог избора већа. Трагови феромона се ажурирају кроз две фазе: фазу евапорације и фазу појачавања. Фаза евапорације имитира испаравање феромона, односно снижавање вредности истог, чиме се појачава степен диверзификације и утиче на разноврсност решења. То се постиже множењем сваке компоненте матрице L вредношћу $(1-\rho)$, тј. $\lambda_{ij} = (1-\rho)\lambda_{ij},\ \rho \in (0, 1]$. Фаза појачавања еквивалентна је освежавању феромонског трага, односно повећавању вредности истог, чиме се фаворизују боља решења. У имплементацији ACO хеуристике то се постиже додавањем сваком члану матрице L параментар δ који директно зависи од квалитета решења, било парцијалног или комплетног,

тј. $\lambda_{ij} = \lambda_{ij} + \delta$. Цео поступак се понавља до испуњења унапред задатог критеријума заустављања. Додатно о овој метахеуристици може се наћи у [34, 75].

1.4.1 Метода променљивих околина

Метода променљивих околина (*engl. VARIABLE NEIGHBORHOOD SEARCH - VNS*) је једна од најпопуларнијих и најефикаснијих метахеуристика заснованих на локалном претраживању. Користи се за решавање великог броја проблема како комбинаторне, тако и континуалне оптимизације. Предложена је 1997. године од стране српског математичара Ненада Младеновића и канадског математичара Пјера Хансена у [77].

VNS метода се ослања на три чињенице везане за различите типове околина и локалне оптимуме унутар њих, и то:

1. локални оптимум у једној околини не мора бити локални оптимум и у некој другој околини;
2. глобални оптимум је уједно и локални оптимум у свим околинама;
3. код великог броја проблема, локални оптимуми нађени у различитим околинама су релативно близу једни другима.

VNS метода се реализује узастопном применом две фазе, фаза размрдавања (*engl. SHAKING*) у којој се креирају различите околине текућег решења и фаза локалне претраге (*engl. LOCAL SEARCH*) која има задатак да детаљно претражи околине текућег решења, уз доношење одлуке о евентуалном преласку у ново решење (*MOVE OR NOT* корак), са циљем да се достигне оптимално решење. Међутим, постоје имплементације које користе само једну од њих две.

Ова метода ужива велику популарност, те се често користи за решавање различитих проблема оптимизације. Током више од двадесет година после њене промоције развијено је доста модификација, о чему се може прочитати у [78, 89]. Неке од најпознатијих и највише примењиваних VNS варијанти су:

- Метода променљивог спуста;
- Основна метода променљивих околина;
- Општа метода променљивих околина;
- Редукована метода променљивих околина;
- Искошена метода променљивих околина.

Метода промењивог спуста

Метода промењивог спуста (*engl. VARIABLE NEIGHBORHOOD DESCENT - VND*) користи прву од три чињенице на које се ослања VNS метахеуристика. Не задовољава се проналаском локалног оптимума унутар појединачних околина, већ даје задатак да се нађе локални оптимум у односу на све унапред изабране околине. Њена специфичност је изостављање фазе размрдавања.

VND метода започиње избором околина $N_k,\ k = k_{min}, ..., k_{max}$, иницијализацијом почетног решења x на унапред изабран начин, и његовим проглашавањем за најбоље, тј. $x^* := x,\ f^* := f(x)$. Свака итерација креће локалним претраживањем околине $N_k(x^*)$, са циљем проналаска најбољег суседа из ње, у ознаци $x^{'}$, и провером квалитета нађеног решења. У случају проблема минимизације, проверава се да ли је испуњен услов $f(x^{'}) < f^*$. Ако јесте испуњен, врши се ажурирање најбољег решења, тј. $x^* := x^{'},\ f^* := f(x^{'})$, а вредност параметра k подешава се на почетну ($k := k_{min}$) и наставља се са претраживањем околина новодобијеног решења. Ако неједнакост није задовољена, онда $k := k + 1$ и наставља се претрага наредне околине текућег решења. Поступак се завршава ако након претраживања свих околина текућег решења нема побољшања. Псеудокод за методу промењивог спуста приказан је Алгоритмом 1.

Алгоритам 1 : Метода промењивог спуста (VND)

```
Одабир околина N_k, k = k_min, ..., k_max
Генерисање почетног решења x
Ажурирање: x* := x, f* := f(x), k := k_min
while k ≤ k_max do
    x' := LocalSearch(x*, k)          ▷ најбоље решење из околине која се претражује
    if f(x') < f* then
        x* := x'
        f* := f(x')
        k := k_min
    else
        k := k + 1
    end if
end while
return x*
```

Основна метода промењивих околина

Основна метода промењивих околина (*engl. BASIC VARIABLE NEIGHBORHOOD SEARCH - BVNS*) је једна од најкоришћенијих варијанти ове метахеуристике. Садржи и фазу размрдавања која јој обезбеђује висок ниво ди-

верзификације, као и фазу локалне претраге која доприноси интензификацији. На самом почетку неопходно је одабрати начин репрезентације решења, дефинисати алгоритам по ком ће се креирати околине $N_k,\ k = k_{min}, ..., k_{max}$, као и алгоритам по ком ће се вршити локална претрага креираних околина.

Након иницијализације почетног решења x, било избором произвољног елемента из допустивог скупа или неком другом процедуром, његовог постављања за најбоље, тј. $x^* := x,\ f^* := f(x)$ и подешавања параметра $k := k_{min}$, врши се фаза размрдавања. Овом фазом започиње свака итерација BVNS методе. У њој се на случајан начин бира допустиво решење $x^{'}$ из околине $N_k(x^*)$. Потом, као резултат одабране локалне претраге која претражује околину од $x^{'}$ добија се локални оптимум $x^{''}$. Проверава се квалитет добијеног решења, тј. у случају проблема минимизације, проверава се да ли је испуњен услов $f(x^{''}) < f^*$. Уколико јесте испуњен, врши се ажурирање најбољег решења, тј. $x^* := x^{''},\ f^* := f(x^{''})$, а вредност параметра k подешава се на почетну ($k := k_{min}$), и наставља се са претраживањем околина новодобијеног решења. Ако неједнакост није задовољена, онда $k := k + 1$ и наставља се претрага наредне околине текућег решења. Када је $k = k_{max}$, његова вредност се ажурира на $k := k_{min}$. Поступак се понавља до испуњења унапред задатог критеријума заустављања. Псеудокод за основну методу променљивих околина приказан је Алгоритмом 2.

Алгоритам 2 : Основна метода променљивих околина (BVNS)

```
Одабир околина N_k, k = k_min, ..., k_max
Генерисање почетног решења x
Ажурирање: x* := x, f* := f(x), k := k_min
while није испуњен критеријум заустављања do
    x' := Shaking(x*, k)                          ▷ фаза размрдавања
    x'' := LocalSearch(x')                        ▷ фаза локалне претраге
    if f(x'') < f* then
        x* := x''
        f* := f(x'')
        k := k_min
    else if k = k_max then
        k := k_min
    else
        k := k + 1
    end if
end while
return x*
```

Општа метода променљивих околина

Општа метода променљивих околина (*engl. GENERAL VARIABLE NEIGHBORHOOD SEARCH - GVNS*) настаје комбиновањем основне методе променљивих околина (BVNS) и методе променљивог спуста (VND) тако што се у алгоритму BVNS фаза локалне претраге замени VND методом. Околине над којима ради VND не морају бити исте као оне које се користе у фази размрдавања. Псеудокод за општу методу променљивих околина приказан је Алгоритмом 3.

Алгоритам 3 : Општа метода променљивих околина (GVNS)

Одабир околина $N_k,\ k = k_{min}, ..., k_{max}$
Одабир околина $N_l,\ l = 1, ..., l_{max}$ за VND
Генерисање почетног решења x
Ажурирање: $x^* := x$, $f^* := f(x)$, $k := k_{min}$
while није испуњен критеријум заустављања **do**
    $x^{'} := Shaking(x^*, k)$ ▷ фаза размрдавања
    $x^{''} := VND(x^{'})$ ▷ VND уместо фазе локалне претраге
    if $f(x^{''}) < f^*$ **then**
        $x^* := x^{''}$
        $f^* := f(x^{''})$
        $k := k_{min}$
    else if $k = k_{max}$ **then**
        $k := k_{min}$
    else
        $k := k + 1$
    end if
end while
return x^*

Редукована метода променљивих околина

Редукована метода променљивих околина (*engl. REDUCED VARIABLE NEIGHBORHOOD SEARCH - RVNS*) је модификација VNS методе која је временски најмање захтевна, обзиром да је карактерише одсуство локалне претраге, која уме да буде временски најзахтевнија компонента ове класе метахеуристика.

На самом почетку поред репрезентације решења, неопходно је дефинисати и алгоритам по ком ће се креирати околине $N_k,\ k = k_{min}, ..., k_{max}$. Затим, врши се иницијализација почетног решења x унапред одабраном процедуром, ажурира се најбоље решење на почетно, тј. $x^* := x,\ f^* := f(x)$ и подешава се параметар $k := k_{min}$. У свакој итерацији на случајан начин се бира допустиво решење $x^{'}$ из околине $N_k(x^*)$. Потом се проверава његов квалитет, тј. у случају проблема минимизације, проверава се да ли је испуњен услов $f(x^{'}) < f^*$. Уколико јесте

испуњен, врши се ажурирање најбољег решења, тј. $x^* := x^{'}$, $f^* := f(x^{'})$, а вредност параметра k подешава се на почетну ($k := k_{min}$), и наставља се са претраживањем околина новодобијеног решења. Ако неједнакост није задовољена, онда $k := k+1$ и наставља се претрага наредне околине текућег решења. Када је $k = k_{max}$, његова вредност се ажурира на $k := k_{min}$. Поступак се понавља до испуњења унапред задатог критеријума заустављања. Псеудокод за редуковану методу променљивих околина приказан је Алгоритмом 4.

Алгоритам 4 : Редукована метода променљивих околина (RVNS)

```
Одабир околина N_k, k = k_min, ..., k_max
Генерисање почетног решења x
Ажурирање: x* := x, f* := f(x), k := k_min
while није испуњен критеријум заустављања do
    x' := Shaking(x*, k)                          ▷ фаза размрдавања
    if f(x') < f* then
        x* := x'
        f* := f(x')
        k := k_min
    else if k = k_max then
        k := k_min
    else
        k := k + 1
    end if
end while
return x*
```

Искошена метода променљивих околина

Искошена метода променљивих околина (*engl. SKEWED VARIABLE NEIGHBORHOOD SEARCH - SVNS*) је најпогоднија за коришћење код проблема код којих се локални оптимуми налазе далеко један од другог. Разлог је то што под одређеним условима допушта прихватање и лошијег решења од већ нађеног, како би повећала степен диверзификације и омогућила истаживање удаљених делова простора претраге.

На самом почетку поред избора репрезентације решења, дефинисања алгоритма по ком ће се креирати околине N_k, $k = k_{min}, ..., k_{max}$, неопходно је и задати метрику $d(\cdot, \cdot)$ којом ће се дефинисати растојање између два допустива решења, као и одабрати вредност параметра α на основу ког ће се одлучивати о прихватању лошијег решења.

Након генерисања почетног решења x на унапред одабран начин, његовог постављања за најбоље, тј. $x^* := x$, $f^* := f(x)$ и подешавања параметра

$k := k_{min}$, врши се фаза размрдавања. Овом фазом започиње свака итерација SVNS методе. У њој се на случајан начин бира допустиво решење $x^{'}$ из околине $N_k(x)$. Потом, као резултат одабране локалне претраге која претражује околину од $x^{'}$ добија се локални оптимум $x^{''}$. Разматра се квалитет добијеног решења, тј. у случају проблема минимизације, проверава се да ли је испуњен услов $f(x^{''}) < f^*$. Уколико јесте испуњен, врши се ажурирање најбољег решења, тј. $x^* := x^{''}$, $f^* := f(x^{''})$, као и текућег $x := x^{''}$, вредност параметра k подешава се на почетну ($k := k_{min}$), и наставља се са претраживањем околина новодобијеног решења. Ако неједнакост није задовољена, тј. ако је $f(x^{''}) \geq f^*$, онда се на основу удаљености $x^{''}$ од x^* која се одређује изабраном метриком $d(x^*, x^{''})$, а контролисано парамтром α, одлучује да ли се решење прихвата као текуће, и то на следећи начин: ако је $f(x^{''}) - \alpha \cdot d(x^*, x^{''}) < f^*$, решење се прихвата као текуће ($x := x^{''}$), ажурира се $k := k_{min}$ и наставља се са претраживањем његових околина. У супротном, поставља се $k := k + 1$ и наставља се претрага наредне околине (непромењеног) текућег решења. Када је $k = k_{max}$, његова вредност се ажурира на $k := k_{min}$. Поступак се понавља до испуњења унапред задатог критеријума заустављања. Псеудокод за искошену методу променљивих околина приказан је Алгоритмом 5.

Алгоритам 5 : Искошена метода променљивих околина (SVNS)

Одабир околина N_k, $k = k_{min}, ..., k_{max}$ и одабир параметра α
Генерисање почетног решења x
Ажурирање: $x^* := x$, $f^* := f(x)$, $k := k_{min}$
while није испуњен критеријум заустављања **do**
    $x^{'} := Shaking(x, k)$ ▷ фаза размрдавања
    $x^{''} := LocalSearch(x^{'})$ ▷ фаза локалне претраге
    if $f(x^{''}) < f^*$ **then**
        $x^* := x^{''}$
        $f^* := f(x^{''})$
        $x := x^{''}$
        $k := k_{min}$
    else if $f(x^{''}) - \alpha \cdot d(x^*, x^{''}) < f^*$ **then** ▷ прихватање лошијег решења за текуће
        $x := x^{''}$
        $k := k_{min}$
    else if $k = k_{max}$ **then**
        $k := k_{min}$
    else
        $k := k + 1$
    end if
end while
return x^*

1.5 Матхеуристике

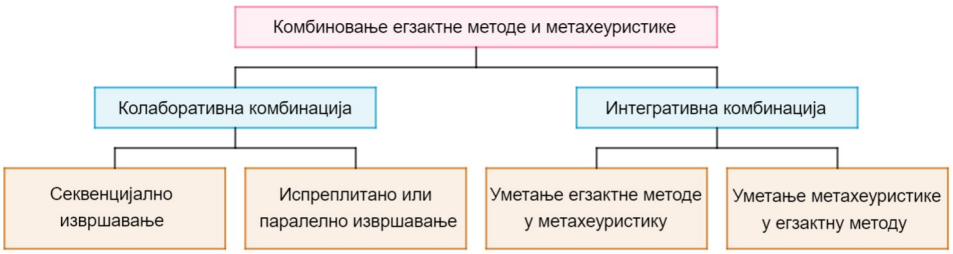
Реч „матхеуристика" (*engl. MATHEURISTIC*) први пут је употребљена 2006. године на првој међународној радионици одржаној у Бертинороу у Италији, на којој се разматрала употреба математичких алата за креирање хеуристика. Како јој само име каже, представља симбиозу егзактне методе (методе математичког програмирања) и метахеуристике. Обзиром да спада у групу хеуристика, неопходно је да приликом имплементације хибридизације између тачне и приближне методе од којих је сачињена, буде очувана општост, тј. да уз мале измене може да се користи за решавање великог броја различитих проблема.

Како се овај приступ у решавању различитих проблема комбинаторне оптимизације (посебно оних из свакодневног живота) показао углавном најефикаснијим, тако су многа истраживања довела до развоја великог броја различитих техника које спадају у групу матхеуристика. Оне се могу класификовати на више начина, а један од њих јесте и подела према начину на који се реализује хибридизација између егзактне методе и метахеуристике, и то на интегративну комбинацију и колаборативну. Интегративна комбинација подразумева да је један од ова два типа метода уграђен унутар алгоритма овог другог, док код колаборативне комбинације врши се размена информација између њих, али нису део један другог, већ се извршавају независно (секвенцијално, испреплитано или паралелно). Више о томе дато је у [50, 62].

Прва истраживања углавном су била усмерена на технике које настају уметањем егзактних метода унутар алгоритма метахеуристике са циљем да се побољшају перформансе саме метахеуристике, или да се добије решење неког мањег потпроблема, које ће се даље користити у алгоритму метахеуристике. Тачније, егзактне методе се могу користити за генерисање квалитетног почетног решења које се прослеђује метахеуристици. Могу бити од велике користи и код унапређивања претраге околина различитих величина, поправљања целокупног решења или неких локалних елемената унутар решења, код спајања решења уколико се оно добија из више компонената које се одвојено рачунају, код декодирања решења у еволутивним метахеуристикама, и слично. Више о овом приступу може се наћи у [6, 49, 82, 101]. С друге стране, и метахеуристике могу бити уметнуте унутар алгоритма егзактне методе са идејом да обезбеде нека допустива решења или неке постојеће границе, да генеришу колоне или резове, да допринесу стратешком вођењу егзактне претраге, да унесу

дух метахеуристике у егзактну методу тако што ће детаљније истражити околине текућег решења пре него што се оно искористи од стране егзактне методе, и слично. Више о овој групи матхеуристика дато је у [8, 20, 25, 51].

Секвенцијално извршавање подразумева или да се тачна метода извршава као нека врста уводне, припремне радње пре метахеуристике, или обрнуто. Понекад је тешко рећи да ли се прва техника користи као иницијализација друге, или је друга накнадна обрада решења које је генерисала прва. Опширније о овом начину повезивања тачних метода и метахеуристика дато је у [36, 41, 72, 101]. Оно што је мање испитано, а што би могло да доведе до мале револуције у решавању оптимизационих проблема матхеуристикама, су методе код којих се метахеуристике и тачне методе преплићу или се извршавају паралелно. Такав приступ омогућава истовремено решавање више потпроблема, што би могло да доведе до веће ефикасности ове групе метода у решавању оних проблема за које су погодне. Једна од важних имплементационих одлука јесте каква се врста информација размењује између ове две методе и када. Више о овој класи метода може се наћи у [2, 43].



Слика 1.4: Основна класификација начина за креирања матхеуристика

Први од претходно наведених начина креирања матхеуристике је до сада, можда и највише истражен, а један од најчешћих приступа у њеној имплементацији јесте ситуација у којој се локална претрага унутар одабране метахеуристике унапреди применом неке од егзактних метода (више у [38, 39]), а посебно неком од метода гранања, што представља популаран правац истраживања и у литератури се назива локално гранање (*engl. LOCAL BRANCHING*), а које је детаљно објашњено у [66]. Задржавајући основу ове идеје, развијене су различине модификације засноване на неком од математичких алата, као што су: динамичко програмирање, релаксација, дуалност линеарног програмирања, Лагранжова релаксација, сурогатна релаксација или неке друге уско повезане методе, а у којима се користе напреднији начини креирања и претраге околина,

најчешће оних које су великих димензија, неретко и експоненцијалне величине, а неке од најпознатијих су:

- Претрага веома великих околина (*engl. VERY LARGE-SCALE NAIGHBOURHOOD SEARCH - VLSNS*) - [49];
- Метод коридора (*engl. CORRIDOR METHOD - CM*) - [97];
- Метода динамичке претраге (*engl. DYNASEARCH METHOD - DM*) - [15];
- Хеуристика рођења (*engl. DIVING HEURISTICS - DH*) - [25, 26];
- Претрага зрака (*engl. BEAM SEARCH - BS*) - [70, 85].

Поред наведених техника, развијене су и матхеуристике које спадају у тзв. декомпозиционе методе (*engl. DECOMPOSITION METHODS*). Главна идеја ових метода је подела тежих проблема на мање и лакше потпроблеме, који се могу решавати одвојено, а са циљем да се добије целокупно решење прекомпоновањем појединачних решења сваког потпроблема. Они проблеми код којих је декомпозицију могуће извршити у једном кораку називају се (блоковски) одвојиви или тривијално паралелизовани и чине мању и једноставнију групу. Много чешћи су они проблеми код којих постоје везе између потпроблема, што може онемогућити њихово самостално решавање, међутим, даља истраживања довела су до развоја различитих техника за превазилажење оваквих ситуација. Сама идеја декомпозиције је стара, а први записи о њеним применама могу се наћи у [33, 44] редом као Данциг–Вулфова декомпозиција (*engl. DANTZIG-WOLFE DECOMPOSITION*) [4, 58] и Бендерсова декомпозиција (*engl. BENDERS DECOMPOSITION*) [57], чијом генерализацијом, а на основу теорије дуалности презентованој у [53], су развијене две нове методе, декомпозиција ограничења (*engl. CONSTRAINT DECOMPOSITION*) и декомпозиција променљивих (*engl. VARIABLE DECOMPOSITION*), чија је примена веома широка. Симетрија која је уочена између ове две методе објашњава се успостављањем релације дуалности између њих, а показало се да су и друге технике разлагања обухваћене истим тим општим оквиром који је описан у [79]. Још неке од познатих метода декомпозиције су: Лагранжова хеуристика (*engl. LAGRANGIAN HEURISTIC*) [55], Хеуристика сурогатне релаксације (*engl. SURROGATE RELAXATION HEURISTIC*) [27], и друге.

Како овај приступ има богату историју када је у питању решавање најразличитијих проблема оптимизације, тако је и литература која се овом темом

(a) Диаграм Бендерсове декомпозиције

(b) Диаграм Данциг–Вулфове декомпозиције (генерисање колона)

бави садржајна, а детаљније о теоријским и практичним основама метода декомпозиције може се наћи у [5, 16, 35, 46, 104].

Целокупна прича о матхеуристикама, углавном је усмерена на решавање различитих проблема линеарног програмирања. Међутим, досадашња сазнања указују на то да би у будућности већу позорност требало дати и њиховој имплементацији за решавање проблема нелинеарног програмирања, обзиром да тачне методе развијене за њихово решавање не дају тако добре резултате, те постоје назнаке да би у комбинацији са метахеуристиком могле да достигну много већу ефикасност, а један од примера дат је у [30]. Детаљније о матхеуристикама може се наћи у [3, 59, 60, 74].

Глава 2

Проблем правовременог распоређивања послова по машинама у вишефазној производњи

Са појавом дигиталне, а касније и четврте индустријске револуције проблеми комбинаторне оптимизације постајали су све популарнији. Многи од њих, као што су проблеми у управљању производњом, распоређивање ресурса у простору и времену, транспортни проблеми и слично, су врло применљиви у пракси, али ни они који су теоријске природе, као што су проблеми из теорије графова или теорије мрежа, не заостају у својој атрактивности.

Не постоји општи образац по ком би се сви проблеми комбинаторне оптимизације могли решити, већ се сваком приступа индивидулано. Стога је њихово изучавање интензивније, а коришћење креираних модела све присутније у пракси. Репрезентативни пример тога јесу и тзв. проблеми распоређивања (*engl. SCHEDULING PROBLEMS*). Њихове основне варијанте формулисао је Грејем 1966. године у [87], а данас представљају једну од популарнијих области комбинаторне оптимизације. Задатак ових проблема је просторно или временско распоређивање одређеног скупа ресурса или послова на одговарајући скуп позиција или машина са циљем постизања најкраћег времена рада, најмањих укупних трошкова, највеће могуће добити, најмање укупне казне и слично. Више информација о овим проблемима, и методама за њихово решавање може се наћи у [12, 71, 81]. У зависности од појединачних елемената разликује се

више класа ових проблема, а детаљнија подела дата је у [7]. Једну од најпознатијих међу њима чине проблеми распоређивања послова по машинама.

Проблеми распоређивања послова по машинама се веома често могу срести у свету индустрије. Многа производна предузећа теже да са што мање губитака остваре свој циљ, те је планирање производње неизоставан део процеса. У зависности од захтева који су постављени постоји неколико варијанти овог проблема, а оне се разликују по начину организације производње, зацртаним циљевима компаније, расположивим временским и просторним капацитетима и слично. Додатно, квалитет машина, степен аутоматизације, расположивост потребних ресурса и многи други фактори, намећу додатна ограничења која је неопходно уврстити у модел који се креира.

Проблеми распоређивања n послова на m машина код којих се сваки посао састоји из n_i, $i = 1, 2, ..., n$ операција може се класификовати на више начина, а овде ће бити изложене неке од најпознатијих варијанти:

- Проблеми распоређивања послова на једној машини (*engl. SINGLE MACHINE SCHEDULING - SMS*) - сви послови извршавају се на једној машини ($m = 1$) [91];

- Проблеми распоређивања послова на паралелним машинама (*engl. PARALLEL MACHINES SCHEDULING - PMS*) - све машине су исте и не мора се водити рачуна која ће се операција извршавати на којој машини [105];

- Проблеми распоређивања послова на вишенаменским машинама (*engl. MULTI - PURPOSE MACHINES SCHEDULING - MPMS*) - све машине су различите и мора се водити рачуна која ће се операција извршавати на којој машини [63];

- Проблеми распоређивања послова у вишефазној производњи (*engl. FLOW SHOP SCHEDULING - FSS*) - сваки посао састоји се од тачно m операција ($n_i = m,\ i = 1, ..., n$) које су задате у уређеном поретку који се мора поштовати, i-та операција за сваки посао извршава се на i-тој машини, а циљ је минимизовати укупно време извршавања свих операција [92];

- Проблеми распоређивања послова у флексибилној вишефазној производњи (*engl. FLEXIBLE FLOW SHOP SCHEDULING - FFSS*) - уопштење FSS проблема, могуће је дуплирање одређеног броја машина, па је самим тим

повећана и флексибилност у одређивању распореда извршавања операција на машинама [52];

- Проблеми распоређивања послова по машинама у вишефазној производњи (*engl. JOB SHOP SCHEDULING - JSS*) -сваки посао се састоји од уређеног скупа операција које се морају извршавати редом, на унапред изабраним машинама, где се тачно зна која се операција извршава на којој машини, али не нужно i-та операција на i-тој машини, при чему једна машина у истом тренутку може обрађивати највише једну операцију [24, 108];

- Проблеми распоређивања послова по машинама у отвореној вишефазној производњи (*engl. OPEN SHOP SCHEDULING - OSS*) - сваки посао је задат преко скупа операција које се могу извршавати по произвољном редоследу на унапред одређеним машинама. Једна машина у истом тренутку може обрађивати највише једну операцију [96, 100].

Свака од наведених класа има своје потклассе које се проучавају засебно. У наставку биће разматран проблем правовременог распоређивања послова по машинама (*engl. JUST-IN-TIME JOB-SHOP SCHEDULING PROBLEM (JIT-JSSP)*) који захтева да се пронађе оптималан редослед извршавања свих операција за све послове, тако да укупна казна за ранији завршетак или кашњење одговарајућих операција буде минимална [37, 42, 56, 76, 90, 95, 107]. Ова група проблема најчешће се јавља у две варијанте. Прва, када се казна односи на време завршетка посла, тј. само се мери време завршетка задње операције за сваки посао, док је казна за остале операције једнака 0. Недостатак овог модела је у томе што се првих $n_i - 1$ операција изврши што је могуће раније, а онда долази до дугог чекања на извршавање последње операције, што може довести до додатних трошкова, као што су, нпр. трошкови складиштења. Друга варијанта овог проблема подразумева да је задато идеално време завршетка за сваку операцију појединачно. Овај модел биће разматран у наставку рада.

2.1 Преглед и дефиниција JIT-JSSP

Нека је дат скуп од n послова и одговарајући скуп од m различитих машина. Нека је сваки посао подељен на низ операција, тако што се i-ти посао састоји од укупно $n_i \leq m,\ i = 1, 2, ..., n$ операција које се морају извршити по унапред задатом редоследу. Наредна операција може кренути са реализацијом

тек након завршетка претходне. Извршавање операција се не може прекидати, већ по завршетку једне, може почети извршавање друге. Појединачно, за сваку операцију одговарајућег посла, унапред је одабрана машина из задатог скупа на којој она мора да се изврши. Притом, једна машина одговара највише једној операцији за посматрани посао. Задато је и време трајања операције, које може бити различито за сваку од њих, без обзира на којој се машини извршава, као и најбољи временски тренутак за завршетак исте. Свака машина у истом тренутку може извршавати највише једну операцију, што код одабраног редоследа извршавања може довести до ранијег или каснијег завршетка од предвиђеног и резултирати казном. Задатак је пронаћи такав редослед извршавања операција на машинама да укупна казна буде минимална. Овако описани проблем у литератури се назива „**Проблем правовременог распоређивања послова по машинама у вишефазној производњи**" (*engl. JUST-IN-TIME JOB-SHOP SCHEDULING PROBELM (JIT-JSSP)*) и припада проблемима целобројног линеарног програмирања.

JIT-JSSP се може срести у различитим гранама индустрије. Уколико се посао не заврши на време, конзументи производа губе стрпљење што нарушава репутацију фирме и одвлачи потенцијалне нове купце. Неретко се дешава да се због кашњења исплаћују и новчани пенали, што је додатни трошак који свака озбиљна компанија жели да избегне. С друге стране, уколико дође до ранијег завршетка, произведену робу је неопходно негде складиштити, те се на тај начин јављају додатни трошкови залиха. У циљу превазилажења оваквих ситуација, тежи се проналажењу таквог редоследа извршавања операција на машинама, да укупна казна буде минимална. Овакви проблеми спадају у НП-тешке [19] и често су веома компликовани за решавање.

2.2 Математичка формулација JIT-JSSP

Математичка формулација проблема правовременог распоређивања послова по машинама у вишефазној производњи са укљученим казнама за превремени завршетак и кашњење за сваку операцију, први пут је изложена у [68] и дата је коришћењем следеће нотације:

Скупови:

- $\boldsymbol{N = \{1, 2, ..., n\}}$ - скуп послова;

- $\boldsymbol{M = \{1, 2, ..., m\}}$ - скуп машина;
- $\boldsymbol{O_i = \{1, 2, ..., n_i\}}$, $n_i \leq m$, $i \in N$ - скуп редних бројева операција за сваки посао $i \in N$;
- $\boldsymbol{O(M_j)}$, $j \in M$ - скуп операција које се извршавају на машини $j \in M$.

Индекси:

- $\boldsymbol{i \in N}$- редни број посла;
- $\boldsymbol{j \in M}$- редни број машине;
- $\boldsymbol{k \in O_i}$, $i \in N$ - редни број операције за посао $i \in N$;
- $\boldsymbol{o_i^k}$, $k \in O_i$, $i \in N$ - k-та операција за i-ти посао;
- $\boldsymbol{M(o_i^k) \in M}$, $k \in O_i$, $i \in N$ - редни број машине на којој се мора извршити операција o_i^k.

Параметри:

- $\boldsymbol{d_i^k > 0}$, $k \in O_i$, $i \in N$ - идеално време завршетка (правовремени завршетак) операције o_i^k, мерено од почетка рада свих машина;
- $\boldsymbol{p_i^k \geq 0}$, $k \in O_i$, $i \in N$ - време трајања операције o_i^k;
- $\boldsymbol{\alpha_i^k \geq 0}$, $k \in O_i$, $i \in N$ - казнени коефицијент за превремени завршетак операције o_i^k;
- $\boldsymbol{\beta_i^k \geq 0}$, $k \in O_i$, $i \in N$ - казнени коефицијент за кашњење операције o_i^k.

Променљиве:

- $\boldsymbol{c_i^k \geq 0}$, $k \in O_i$, $i \in N$ - време завршетка операције o_i^k;
- $\boldsymbol{E_i^k = \max\{0, d_i^k - c_i^k\}}$, $E_i^k \geq 0$, $k \in O_i$, $i \in N$ - разлика између правовременог и превременог завршетка операције o_i^k;
  За $E_i^k = 0$ - није дошло до ранијег завршетка операције o_i^k, те нема казне за њен превремени завршетак;
  За $E_i^k > 0$ - дошло је до ранијег завршетка операције o_i^k, што резултира одговарајућом казном $\alpha_i^k \cdot E_i^k$;
- $\boldsymbol{T_i^k = \max\{c_i^k - d_i^k, 0\}}$, $T_i^k \geq 0$, $k \in O_i$, $i \in N$ - кашњење операције o_i^k;

За $T_i^k = 0$ - није дошло до кашњења у реализацији операције o_i^k, те нема казне за њен закаснели завршетак;

За $T_i^k > 0$ - дошло је до кашњења у реализацији операције o_i^k, што резултира одговарајућом казном $\beta_i^k \cdot T_i^k$.

Ако је $E_i^k = 0$ и $T_i^k = 0$, нађено је идеално време завршетка операције o_i^k.

Математичка формулација

Математички модел проблема правовременог распоређивања n послова на m машинама у вишефазној производњи са укљученим казнама за превремени завршетак и кашњење за сваку операцију, презентован у наставку овог рада, преузет је из [9] и у даљем тексту биће означен са ILP1.

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k E_i^k + \beta_i^k T_i^k \tag{2.1}$$

При ограничењима:

$$c_i^1 - p_i^1 \geq 0, \quad i \in N \tag{2.2}$$

$$c_i^k \leq c_i^{k+1} - p_i^{k+1}, \quad k \in O_i/\{n_i\},\ i \in N \tag{2.3}$$

$$c_i^k \geq c_l^h + p_i^k \ \vee \ c_l^h \geq c_i^k + p_l^h, \tag{2.4}$$

$$o_i^k, o_l^h \in O(M_j),\ j \in M,\ k \in O_i,\ h \in O_l,\ i \neq l,\ i, l \in N$$

$$E_i^k \geq d_i^k - c_i^k, \quad k \in O_i,\ i \in N \tag{2.5}$$

$$T_i^k \geq c_i^k - d_i^k, \quad k \in O_i,\ i \in N \tag{2.6}$$

$$E_i^k,\ T_i^k \geq 0, \quad k \in O_i,\ i \in N. \tag{2.7}$$

Функција циља дефинисана је као збир казни, тј. као сума свих разлика између правовременог и превременог завршетка операција, као и сума свих кашњења, скалирано одговарајућим казненим коефицијентима. Задатак је пронаћи минимум функције циља, а да притом буду испоштована задата ограничења која имају следећа значења:

- Услови (2.2) обезбеђују да извршавање прве операције за сваки посао не може почети пре нултог момента.

- Услови (2.3) обезбеђују да за сваки посао $i \in N$ и сваке две његове узастопне операције o_i^k и o_i^{k+1}, извршавање операције o_i^{k+1} не може почети пре него што се заврши извршавање операције o_i^k. Ови услови се називају и *ограничења приоритета.*

- Услови (2.4) обезбеђују да се на једној машини истовремено може извршавати само једна операција. Ови услови се називају и *ограничења ресурса.*

- Услови (2.5) и (2.6) задају начине рачунања вредности за превремени завршетак рада и кашњење, редом.

- Услови (2.7) означавају ненегативност променљивих које чувају информацију о ранијем или каснијем завршетку.

Изложена математичка формулација проблема, због дисјункције отежава решавање проблема егзактним решавачима, па се зато уводи нови модел.

Други математички модел за JIT-JSSP, који је преузет из [88], а у наставку ће бити означен са ILP2, користи исту нотацију као и први, али за разлику од њега подразумева да је редослед извршавања операција на машинама унапред познат. Услови (2.4) су замењени условима:

$$c_i^k \geq c_l^h + p_i^k, \text{ ако је } M(o_i^k) = M(o_l^h) \text{ и операција } o_i^k \text{ се појављује после операције } o_l^h \text{ у вектору решења, } k \in O_i,\ h \in O_l,\ i \neq l,\ i, l \in N \tag{2.8}$$

који заједно са условима (2.2) - (2.3) и (2.5) - (2.7) дефинишу нову математичку формулацију проблема.

- Услови (2.8) захтевају да ако се две операције извршавају на истој машини, онда она која се појављује касније у вектору решења мора почети након завршетка претходне.

Овај модел је програмабилан и лако се може имплементирати у жељени егзактни решавач.

2.3 Теоријске и рачунске границе за JIT-JSSP

Проблеми распоређивања послова по машинама у вишефазној производњи спадају у групу новијих НП-тешких проблема. Многи од њих могу бити веома захтевни за решавање, па поред алгоритама развијених за одређивање што

квалитетнијих решења, важну улогу у њиховом проучавању имају и теоријски докази доњих и горњих граница решења о којима ће бити речи у овом поглављу. За одређивање граница коришћена је Лагранжова релаксација одговарајућег модела.

Лагранжова релаксација (*engl. LAGRANGIAN RELAXATION*) представља моћну оптимизациону технику којом се полазни проблем апроксимира проблемом једноставнијим за решавање. Главна идеја се огледа у подели ограничења на две групе, и то на „мека" и „тврда", где се потом „тврда" ограничења искључују из скупа ограничења и помножена одговарајућим Лагранжовим коефицијентима λ додају функцији циља, док „мека" остају непромењена. Новодобијени проблем најчешће је лакши за решавање, његове особине у многоме могу помоћи у анализи почетног проблема, а решења су му блиска решењима полазног проблема. Ова врста релаксације неретко даје квалитетне горње границе решења, чијом се претрагом, може доћи до висококвалитетних решења првобитног проблема. Детаљније о овој методи може се наћи у [67].

Истражујући проблеме распоређивања послова по машинама, као и сложеније варијанте проблема планирања, извесни број научника користио је Лагранжове релаксације као помоћ у њиховом решавању. Проучаване су две варијанте ових релаксација, а детаљније о њима, као и о добијеним резултатима може се наћи у [13, 22, 32] и [14, 37].

Вођени закључцима изведеним о квалитету Лагранжових релаксација за проблеме распоређивања послова по машинама, аутори рада [68] презентовали су две технике за добијање релаксације JIT-JSSP. Прва се заснива на ублажавању ограничења приоритета која је мотивисана резултатима изложеним у [31] и у наставку ће бити детаљно презентована. Друга се заснива на ублажавању ограничења ресурса, мотивисана је садржајем рада [37] и урађена на основу математичке формулације изложене у [65].

- **Ублажавање ограничења приоритета**

Лагранжова релаксација је добијена ублажавањем ограничења приоритета, а изведена је из математичког модела ILP1 и користи исту нотацију.

За сваку операцију o_i^k уводи се време реализације $r_i^k = \sum_{1 \leq h \leq k-1} p_i^h$, и модификацијом почетног модела добија се:

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k E_i^k + \beta_i^k T_i^k \tag{2.9}$$

При ограничењима:

$$c_i^k - p_i^k \geq r_i^k, \quad k \in O_i,\ i \in N \tag{2.10}$$

$$c_i^k \leq c_i^{k+1} - p_i^{k+1}, \quad k \in O_i/\{n_i\},\ i \in N \tag{2.11}$$

$$c_i^k \geq c_l^h + p_i^k \ \vee \ c_l^h \geq c_i^k + p_l^h, \quad o_i^k, o_l^h \in O(M_j),\ j \in M \tag{2.12}$$

$$E_i^k = \max\{0, d_i^k - c_i^k\}, \quad k \in O_i,\ i \in N \tag{2.13}$$

$$T_i^k = \max\{0, c_i^k - d_i^k\}, \quad k \in O_i,\ i \in N. \tag{2.14}$$

Нека је са $c = [c_i^k],\ k \in O_i,\ i \in N$ означен вектор који садржи време завршетка сваке операције o_i^k, а са $\lambda = [\lambda_i^k],\ k \in O_i,\ i \in N$ вектор који садржи одговарајуће Лагранжове коефицијенте. Приоритет у извршавању операција o_i^k и o_i^{k+1} регулише се Лагранжовим коефицијентом $\lambda_i^k \geq 0$, а Лагранжова функција L(c,λ) је следећег облика:

$$L(c, \lambda) = \sum_{i=1}^{n} \sum_{k=1}^{n_i} (\alpha_i^k E_i^k + \beta_i^k T_i^k) + \sum_{i=1}^{n} \sum_{k=1}^{n_i - 1} \lambda_i^k (c_i^k - c_i^{k+1} + p_i^{k+1}). \tag{2.15}$$

Под претпоставком да је $\lambda_i^0 = \lambda_i^{n_i} = 0$, Лагранжова функција (2.15) се може записати у следећем облику:

$$L(c, \lambda) = \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k(\lambda) E_i^k + \beta_i^k(\lambda) T_i^k - K(\lambda), \tag{2.16}$$

где је:

$$K(\lambda) = \sum_{i=1}^{n} \sum_{k=1}^{n_i} \lambda_i^k (p_i^{k+1} + d_i^k - d_i^{k+1}) \tag{2.17}$$

$$\alpha_i^k(\lambda) = \alpha_i^k - \lambda_i^k + \lambda_i^{k+1} \tag{2.18}$$

$$\beta_i^k(\lambda) = \beta_i^k + \lambda_i^k - \lambda_i^{k+1}. \tag{2.19}$$

У наредном кораку фиксирају се вредности вектора λ и решава се проблем минимизације функције L(c,λ) по непознатој c, те ће се L(c,λ) надаље посматра као функција једне променљиве L(λ):

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n_i} \alpha_i^k(\lambda) E_i^k + \beta_i^k(\lambda) T_i^k - K(\lambda) \tag{2.20}$$

При ограничењима:

$$c_i^k - p_i^k \geq r_i^k, \quad k \in O_i,\ i \in N \tag{2.21}$$

$$c_i^k \geq c_l^h + p_i^k \ \vee \ c_l^h \geq c_i^k + p_l^h, \quad o_i^k, o_l^h \in O(M_j), \ j \in M \tag{2.22}$$

$$E_i^k = \max\{0, d_i^k - c_i^k\}, \quad k \in O_i, \ i \in N \tag{2.23}$$

$$T_i^k = \max\{0, c_i^k - d_i^k\}, \quad k \in O_i, \ i \in N. \tag{2.24}$$

Овако дефинисан проблем може се поделити на m потпроблема у којима се сви послови извршавају на једној машини (*engl. SINGLE MACHINE SCHEDULING*), а задатак је распоредити их тако да укупна казна за превремени завршетак или кашњење буде минимална. У том случају Лагранжова функција је облика:

$$L(\lambda) = \sum_{u=1}^{m} L_u(\lambda) - K(\lambda), \tag{2.25}$$

где $L_u(\lambda), \ u = 1, 2, ..., m$ представља одговарајући потпроблем у коме се сви послови извршавају на једној машини, и дефинисан је на следећи начин:

$$\min \sum_{o_i^k \in O(M_u)} \alpha_i^k(\lambda) E_i^k + \beta_i^k(\lambda) T_i^k \tag{2.26}$$

При ограничењима:

$$c_i^k - p_i^k \geq r_i^k, \quad o_i^k \in O(M_u) \tag{2.27}$$

$$c_i^k \geq c_l^h + p_i^k \ \vee \ c_l^h \geq c_i^k + p_l^h, \quad o_i^k, o_l^h \in O(M_u) \tag{2.28}$$

$$E_i^k = \max\{0, d_i^k - c_i^k\}, \quad o_i^k \in O(M_u) \tag{2.29}$$

$$T_i^k = \max\{0, c_i^k - d_i^k\}, \quad o_i^k \in O(M_u) \tag{2.30}$$

Овако дефинисан проблем познат је у литератури као проблем правовременог распоређивања послова на једној машини где су укључене казне и за ранији завршетак и за кашњење. Иако спада у класу НП-тешких проблема, развијен је механизам за његово ефикасно решавање, а изложен је у [31]. Након решавања свих m потпроблема, потребно је одредити максимум функције L(λ) по променљивој λ.

За горње границе решења презентоване у [68] изабране су најбоље вредности добијене или решавањем Лагранжових релаксација помоћу једне од две различите имплементиране хеуристике или помоћу CPLEX решавача, док су за доње границе изложене вредности добијене на основу обе Лагранжове релаксације као и оне које је дао CPLEX.

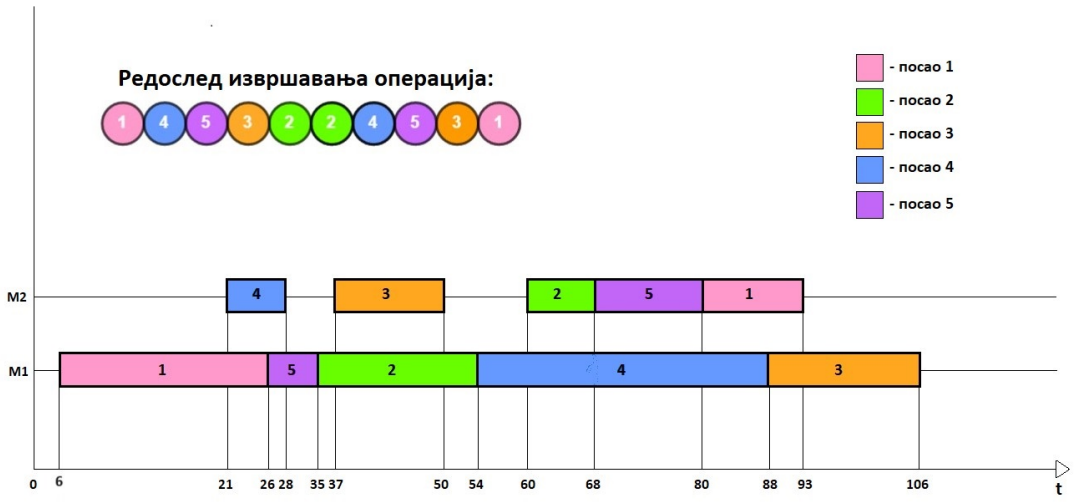
2.4 Илустративни пример проблема

Проблем правовременог распоређивања n послова на m машинама у вишефазној производњи биће презентован кроз пример инстанце која садржи 5 послова ($n = 5$) и 2 машине ($m = 2$). У табели 2.1 изложени су улазни подаци. За сваку операцију дате су следеће информације: редни број машине на којој се одговарајућа операција мора извршити, трајање операције (*engl. PROCESSING TIME* - p), идеално време завршетка операције (*engl. DUE DATE* - d), казнени коефицијент за превремени завршетак операције (α), као и за кашњење (β). На пример, за посао 1 прва операција (o_1^1) се мора извршити на машини 1, време трајања јој је 20 јединица времена, идеално време завршетка је 44. јединица времена, а казнени коефицијенти за превремени завршетак и кашњење су редом 0, 96 и 0, 46 по јединици одступања времена завршетка од идеалног времена завршетка операције.

Табела 2.1: Пример инстанце $n = 5$, $m = 2$

	Операција 1					Операција 2				
Посао	Машина	p	d	α	β	Машина	p	d	α	β
1	1	20	44	0.96	0.46	2	13	75	0.70	0.14
2	1	19	54	0.77	0.53	2	8	68	0.86	0.91
3	2	13	50	0.72	0.74	1	18	87	0.52	0.77
4	2	7	28	0.62	0.62	1	34	70	0.18	0.95
5	1	9	44	0.70	0.93	2	12	74	0.97	0.30

Један од оптималних распореда извршавања операција на машинама презентован је на слици 2.1.



Слика 2.1: Редослед извршавања операција на машинама за инстанцу $n = 5$, $m = 2$

Посматрајући дати редослед послова чије се одговарајуће операције извршавају на машини 1, може се уочити да се прво извршава прва операција за посао 1, тј. o_1^1, која почиње у 6. јединици времена, а завршава се у 26. То резултира превременим завршетком од $E_1^1 = d_1^1 - c_1^1 = 44 - 26 = 18$ јединица времена и казном која износи $\alpha_1^1 \cdot E_1^1 = 0.96 \cdot 18 = 17.28$. Потом, следи извршавање прве операција за посао 5, тј. o_5^1, која почиње у 26. јединици времена, а завршава се у 35, што резултира казном за превремени завршетак која износи $0.70 \cdot (44 - 35) = 0.70 \cdot 9 = 6.30$. Потом, прва операција за посао 2, тј. o_2^1, која почиње у 35. јединици времена, а завршава се у 54, што представља идеално време завршетка операције o_2^1, те нема одступања, већ је $E_2^1 = T_2^1 = 0$, па самим тим нема ни казне. Затим се на машини 1 извршава друга операција за посао 4, тј. o_4^2, која почиње у 54. јединици времена, а завршава се у 88. То резултира кашњењем од $T_4^2 = c_4^2 - d_4^2 = 88 - 70 = 18$ јединица времена и казном која износи $\beta_4^2 \cdot T_4^2 = 0.95 \cdot 18 = 17.1$. Исто тумачење приказаног оптималног решења се наставља и даље за преостале послове, као и за редослед одабран за извршавање на машини 2. Укупна казна износи $0.96 \cdot 18 + 0.7 \cdot 9 + 0.14 \cdot 18 + 0.77 \cdot 19 + 0.95 \cdot 18 + 0.3 \cdot 6 = 59.63$.

Анализом илустрације редоследа извршавања операција на машинама, може се уочити да се на машини 2 прво извршава прва операција за посао 4, тј. o_4^1 (плави правоугаоник), и да њено извршавање почиње у 21, а завршава се у 28. јединици времена, што уједно представља и идеално време завршетка ове операције. Ако би њено извршавање почело пре/после 21. јединице времена, завршила би се пре/после 28, што би довело до превременог завршетка/кашњења и резултирало одговарајућом казном. Исто тумачење важи и за другу по реду операцију која се извршава на машини 2, тј. за операцију o_3^1 (наранџасти правоугаоник), која почиње у 37. јединица времена, а завршава се у 50, што уједно и јесте њено идеално време завршетка. Ако би извршавање почело пре/после 37. јединице времена, завршило би се пре/после 50, и резултирало би казном за превремени завршетак/кашњење, те из тог разлога машина 2 мирује између 28. и 37. јединице времена. Исто важи и за мировање поменуте машине у интервалу између 50. и 60. јединице времена.

2.5 Преглед релевантне литературе

У овом поглављу биће изложени различити приступи у решавању JIT-JSSP који су доступни у литератури.

Поред тога што је први пут дата формулација JIT-JSSP означена као ILP1, у раду [68] објављеном 2008. године, изложене су и две Лагранжове релаксације овог проблема. У једној су ублажена ограничења ресурса, а у другој ограничења приоритета. Тестирање је обављено на скупу од 72 инстанце које су први пут изложене баш у раду [68]. Лагранжова релаксација добијена ублажавањем ограничења ресурса дала је вредности доњих граница решења за свих 72 тест примера, док она добијена ублажавањем ограничења приоритета ове информације нуди за њих 67. Још, показало се да су вредности добијене првом од поменуте две релаксације бољег квалитета у односу на оне добијене другом. Изложене су и доње границе решења за све инстанце, добијене егзактим CPLEX 9.1 решавачем коме је време извршавања ограничено на 10 минута. Међутим, ове вредности су лошијег квалитета од оних добијених Лагранжовим релаксацијама. Презентоване су и горње границе решења за свих 72 тест примера, за коју су изабране најбоље вредности добијене или CPLEX-ом или локалном претрагом решења добијених хеуристиком примењеном на некој од поменутих релаксација.

У раду [1] објављеном 2008. године, JIT-JSSP је решаван на два начина, генетским алгоритмом и његовом хибридизацијом са локалном претрагом. Она је имплементирана тако што се решење добијено генетским алгоритмом прослеђује као почетно за локалну претрагу којом се претражују околине на начин који је први пут изложен у раду [1]. Током извршавања алгоритма локалне претраге не чува се само најбољи сусед, већ се креира ограничена листа потенцијалних добрих решења тако што сваки пут кад се наиђе на боље, оно јој се дода, потом се листа сортира по квалитету решења и наставља се са претрагом околине најбољег решења у њој. Не задаје се критеријум заустављања, већ алгоритам стаје када суседи свих решења буду размотрени и више јој се ниједно потенцијално решење не додаје. У раду [1] нису презентоване добијене вредности за тестиране инстанце, већ процене њихових горњих граница добијене у односу на одговарајуће горње границе које су претходно изложене у [68]. Генетским алгоритмом боља горња граница добијена је за 43, а хибридизацијом за 47 од укупно 72 тестиране инстанце. У просеку 4.9% бољих вредности за

горњу границу дала је хибридизација у односу на генетски алгоритам.

Један од првих радова у ком је коришћена хибридизације између метахеуристика и егзактне методе за решавање JIT-JSSP јесте [88] објављен 2010. године. Комбиновне су три различите методе, и то: еволутивни алгоритам (*engl. EVOLUTIONARY ALGORITHM - EA*) који је уопшење генетског алгоритма [86], локална претрага (LS) и метода математичког програмирања (MP). Презентоване су две групе решења за тестиране инстанце. Прво, имплементирана је хибридизација EA+MP тако што је EA добио задатак да генерише решење, тј. да изабере редослед извршавања свих операција за све послове, а MP да одреди време завршетка за сваку од њих, и то на основу математичког модела ILP2. У овом случају боље вредности за горње границе од оних које су изложене у [68] достигнуте су за 50 од укупно 72 инстанце. Након ове хибридизације имплементирана је још једна код које се решење добијено хибридизацијом EA+MP шаље као почетно локалној претрази која је дефинисана на исти начин као у [1] са циљем да се пронађе побољшање. Добијена хибридизација, означена са EA+MP+LS, добија боље вредности за горње границе од оних које су изложене у [68], за 56 од укупно 72 инстанце. Побољшање након укључивања локалне претраге добијено је за 44 инстанце у односу на резултате добијене алгоритмом EA+MP.

Решавање JIT-JSSP помоћу унапређеног генетског алгоритма (*engl. IMPROVED GENETIC ALGORITHM - IGA*) презентовано је у раду [54] објављеном 2012. године. Као највећи бенефит ове методе, аутори наводе имплементацију тростепеног механизма декодирања решења које има задатак да хромозоме пребаци у одговарајући редослед операција по ком ће се оне извршавати. На првом нивоу користи се полуактивни метод декодирања који има задатак да обезбеди свеобухватност решења и прошири простор претраге. На другом користи се похлепни механизам којим се операције које касне убацују у редослед извршавања на одговарајућој машини тако што се слободан ход те машине додели одговарајућој операцији, поштујући притом сва задата ограничења. На овај начин смањује се број кашњења, а самим тим се смањује и укупна казна за кашњење што директно утиче на квалитет решења. На трећем нивоу користи се похлепни механизам којим се регулише ранији завршетак операције и то у три корака. Први корак подразумева одређивање два подскупа, где један садржи операције које се завршавају пре идеалног времена завршетка, а други оне које се завршавају након идеалног времена завршетка. У другом кораку се редослед

операција које касне чува непромењен, а у трећем се помоћу похлепне процедуре врши убацивање операција са ранијим завршетком на места која се бирају на унапред дефинисан начин. Овај ниво обезбеђује смањење укупне казне за превремени завршетак. Тестирано је 48 од укупно 72 инстанце, а побољшање у односу на до тада позната решења реализовало се код њих 18. Инстанце које садрже 20 послова нису тестиране. Према ауторима рада [54], мана предложеног EA приступа јесте то што може доћи до заглављивања алгоритма у локалном минимуму, те се решење не може додатно побољшати.

У раду [98] објављеном 2014. године, JIT-JSSP решаван је хибридизацијом методе променљивих околина (VNS) и методе математичког програмирања (MP) (коришћен је математички модел ILP2). Два основна питања на којима се темељи ова имплементација јесу како истражити огроман простор претраге и како убацити време мировања за машине. Ради повећања ефикасности одлучено је да у свакој итерацији VNS методе, која има задатак да претражи простор решења и одреди редослед извршавања свих операција за све послове, буде уграђена MP која треба да убаци време мировања за машине како би оптимизовала време завршетка за сваку операцију. Тестиране су 72 инстанце и резултати су упоређени са резултатима презентованим у [88]. Боље решење пронађено је за 40 тест примера, и то за 20 инстанци са 10 послова, 12 инстанци са 15 послова и 8 инстанци са 20 послова.

До сада, последњи приступ у решавању JIT-JSSP изложен је у раду [9] објављеном 2021. године. Имплементирана је метода промељивих околина (VNS) по математичком моделу ILP1. Идеја на основу које је креиран VNS алгоритам обухвата декомпоновање полазног проблема на потпроблеме тако што након одабира редоследа извршавања свих операција, потребно је одредити и оптимално време завршетка за сваку од њих. Већ је напоменуто да JIT-JSSP спада у групу НП-тешких проблема, али налажење оптималног времена завршетка за сваку операцију, уколико је познат редослед извршавања истих, може се реализовати у полиномском времену. Спицифичност ове VNS методе је у дефиницији фазе размрдавања иплементиране преко четири мини-процедуре, од којих су две први пут представљене у [9]. Такав приступ обезбедио јој је велику ефикасност. Осим VNS методом, проблем је решаван и егзактним решавачем CPLEX 12.8.0 коме је време рада ограничено на 30 минута. Тестиране су 72 инстанце, свака од њих покренута је по 5 пута. Како би поређење са познатим резултатима из литературе било што прецизније, на истом рачунару на ком су

тестирани VNS и CPLEX, извршено је и тестирање алгоритама изложених у радовима [88] и [98]. Имплементирана VNS метода изложена у [9] дала је 41 ново најбоље решење од укупно 72 тест примера, што износи 57%. За инстанце са 20 послова побољшање се јавило код чак 80% њих, са 15 послова код 54%, а са 10 послова код 38%. Као критеријум заустављања задат је максималан број итерација који износи $\lfloor \frac{n \cdot m}{4} \rfloor$ (цео део броја $\frac{n \cdot m}{4}$) или, уколико након 3 итерације нема побољшања решења, алгоритам такође стаје.

Као што је раније већ напоменуто, у овом раду биће презентована матхеуристика за решавање JIT-JSSP, добијена хибридизацијом VNS методе и егзактног CPLEX решавача. Мотивација за овакав одабир методе пронађена је, пре свега, у томе што је JIT-JSSP лако поделити на мање потпроблеме који се могу решавати независно, и то тако што матхеуристика повери потпроблем одређивања редоследа извршавања свих операција за све послове одабраној метахеуристици, а потпроблем одређивања времена завршетака за сваку операцију појединачно, повери егзактном решавачу. Узимајући у обзир резултате добијене у ранијим решавањима JIT-JSSP и ефикасност коју је показала VNS метода у [9], одлучено је да се редослед извршавања свих операција за све послове одређује управо VNS методом. Уколико је редослед извршавања свих операција за све послове већ познат, проблем одређивања времена завршетака за сваку од њих је полиномске сложености [71], те га је могуће решавати егзактним решавачем. У овом случају одабран је CPLEX због својих перформанси. Детаљније о предложеној матхеуристици изложено је у наредном поглављу.

Глава 3

Матхеуристика за решавање проблема правовременог распоређивања послова по машинама у вишефазној производњи

Матхеуристика имплементирана за решавање проблема правовременог распоређивања послова по машинама у вишефазној производњи представља хибридизацију методе променљивих околина (VNS) и егзактног CPLEX решавача. Главна идеја огледа се у декомпозицији полазног проблема на потпроблеме тако што се, поштујући правила коришћене метахеуристике, утврђује редослед извршавања свих операција за све послове на одговарајућим машинама, а решавање потпроблема које подразумева одређивање времена завршетка за сваку операцију појединачно, као и проналажење вредности функције циља, задатак је егзактног решавача.

3.1 Репрезентација решења

Правовремено распоређивање n послова на m машина у вишефазној производњи спада у групу оптимизационих проблема минимизације. Стога, први корак у решавању јесте правилно дефинисање скупа допустивих решења X, тј.

потребно је одабрати такву репрезентацију за свако $z \in X$ која ће омогућити лакшу манипулацију подацима унутар алгоритма имплементиране матхеуристике.

У овом раду, као и у [68], разматран је случај JIT-JSSP у ком се сваки посао састоји од тачно m операција, тј. $n_i = m$, за свако $i \in N$, па су одговарајући скупови који садрже редне бројеве операција истог облика и то $O_i = \{1,\ 2,\ 3,\ ...,\ m\}$, за свако $i \in N$. Стога је за скуп допустивих решења X изабран скуп свих вектора дужине $n \cdot m$ који се могу добити као произвољна пермутација са понављањем целобројних вредности из скупа $\{1,\ ...,\ 1,\ 2,\ ...,\ 2,\ ...,\ n,\ ...,\ n\}$, где се сваки цео број $i = 1,\ 2,\ ...,\ n$ појављује тачно m пута. На тај начин дат је јединствен редослед извршавања свих операција за сваки посао, и то тако што k-то појављивање редног броја посла у вектору решења подразумева извршавање k-те операције тог посла на одговарајућој машини. На основу улазних података, оваква репрезентација решења омогућава лаку реконструкцију редоследа извршавања операција на машинама, обзиром да је свакој од m машина додељена тачно по једна операција за сваки посао.

Пример 3.1. *Код већ разматраног примера описаног у поглављу 2.4 где је* $n = 5$ *и* $m = 2$ *једно допустиво решење је облика* $z = [1,\ 4,\ 5,\ 3,\ 2,\ 2,\ 4,\ 5,\ 3,\ 1]$, *што значи да прво појављивање броја 1 (2, 3, 4 или 5) представља прву операцију за посао 1 (2, 3, 4 или 5). Друго појављивање одговарајућег броја даје информацију о извршавању друге операције за одговарајући посао, и тако редом. Редни број машине на којој се одговарајућа операција извршава дефинисан је улазним подацима и дат је у табели 2.1. С друге стране, и вектор облика* $z = [1,\ 2,\ 1,\ 3,\ 3,\ 4,\ 5,\ 2,\ 4,\ 5]$ *такође представља једно допустиво решење овог проблема. Сада, прво појављивање броја 1 представља прву операцију за посао 1* (o_1^1), *која се извршава на машини 1. Број 2 указује на прву операцију за посао 2* (o_2^1), *која се извршава на машини 1, број 1 означава да се извршава друга операција за посао 1* (o_1^2), *на машини 2, и тако редом.*

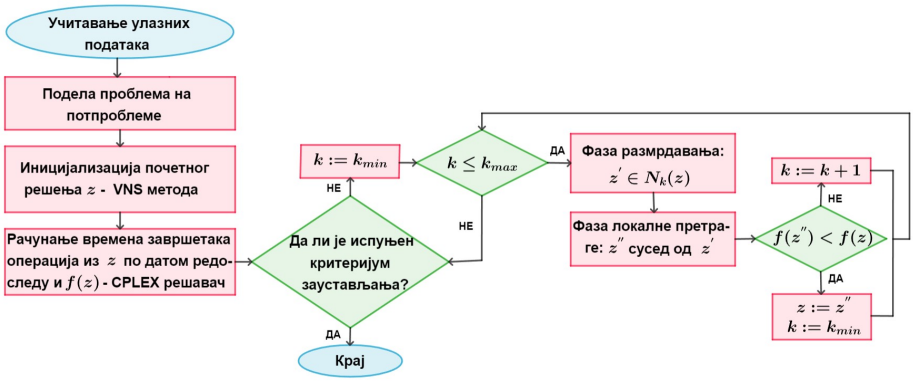
Дакле, решење може бити било која пермутација са понављањем целобројних вредности из скупа $\{1,\ ...,\ 1,\ 2,\ ...,\ 2,\ ...,\ n,\ ...,\ n\}$, где се сваки цео број $i = 1,\ 2,\ ...,\ n$ појављује тачно m пута, а не само произвољне пермутације унутар блоковских подскупова $\{1,\ 2, ...,\ n\}$.

Оваква репрезентација решења за JIT-JSSP први пут је предложена 1994. године у [108] и врло је погодна, с обзиром да након фазе размрдавања и фазе

локалне претраге унутар VNS методе не мора се проверавати допустивост потенцијалног решења, тј. свака измена решења у оквиру ових фаза чува допустивост новодобијеног решења.

3.2 Метода променљивих околина

Метода променљивих околина (*engl. VARIABLE NEIGHBORHOOD SEARCH - VNS*) представља костур имплементиране матхеуристике. Матхеуристика најпре почетни проблем дели на мање потпроблеме који су лакши за решавање, и то тако што VNS метода одабере редослед извршавања свих операција за све послове, који се, затим проследи егзактном CPLEX решавачу. Он, без могућности промене датог редоследа извршавања операција, а на основу задатих ограничења, рачуна оптимално време завршетка сваке операције појединачно. Међутим, како је на једној машини у истом тренутку могуће извршавање само једне операције, а за један посао све њему одговарајуће операције се морају извршавати редом, а и извршавање наредне операције за одговарајући посао не може почети пре завршетка претходне, оптимална времена завршетка операција која одрђује CPLEX могу се разликовати од одговарајућих идеалних времена завршетака, што доводи до ранијих или каснијих завршетака и резултира казнама. На основу тих казни CPLEX рачуна вредност функције циља за тако одабрани редослед извршавања операција и прослеђује је VNS методи. Она, потом проверава да ли је испуњен унапред одабрани критеријум заустављања, било да је то задато време извршавања програма, број итерација, постигнута одређена тачност решења или нешто друго. Ако је изабрани критеријум заустављања испуњен, алгоритам стаје и одабрани редослед извршавања операција на одговарајућим машинама се узима за апроксимацију решења полазног проблема. Уколико изабрани критеријум заустављања није испуњен, VNS метода кроз фазу размрдавања и фазу локалне претраге бира нови редослед извршавања операција, а описани поступак се итеративно понавља до испуњења задатог критеријума заустављања.



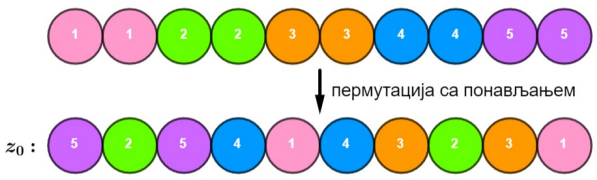
Слика 3.1: Диаграм матхеуристике предложене за решавање JIT-JSSP

Поред добро имплементиране хибридизације егзактне и апроксимативне методе, на ефикасност матхеуристике значајно утиче ефикасност одабране метахеуристике. Стога, поред паметно изабране репрезентације решења, критеријума заустављања и иницијализације почетног решења, код VNS методе велики утицај на ефикасност има начин на који су дефинисане околине решења које ће се претраживати, као и фаза размрдавања и фаза локалне претраге које су уједно и главни градивни елементи ове метахеуристике.

3.2.1 Иницијализација почетног решења

Почетно решење z_0 се бира на случајан начин из скупа допустивих решења X. Конкретније, добија се као једна, произвољна пермутација са понављањем елемената из скупа $\{\underbrace{1, \ldots, 1}_{m}, \underbrace{2, \ldots, 2}_{m}, \ldots \underbrace{n, \ldots, n}_{m}\}$.

Пример 3.2. *На примеру описаном у поглављу 2.4 код кога је $n = 5$ и $m = 2$ за почетно решење се бира произвољна пермутација са понављањем елеманата из скупа $\{\underbrace{1, 1}_{2}, \underbrace{2, 2}_{2}, \underbrace{3, 3}_{2}, \underbrace{4, 4}_{2}, \underbrace{5, 5}_{2}\}$, па један од могућих избора за почетно решење је $z_0 = [5, 2, 5, 4, 1, 4, 3, 2, 3, 1]$, што је приказано на слици 3.2.*



Слика 3.2: Иницијализација почетног решења за инстанцу $n = 5,\ m = 2$

3.2.2 Дефиниција околина

Као што је раније у овом поглављу наведено, на ефикасност VNS методе велики утицај има и начин на који су дефинисане околине које се претражују. У овој имплементацији VNS методе која је део матхеуристике одабране за решавање проблема правовремено распоређивање n послова на m машина у вишефазној производњи, претражују се три различита типа околина означених са N_k, N_1^{ls} и N_2^{ls}, а које су одабране екперименталним путем. Елементи околине допустивог решења z називају се суседима од z.

Дефиниција 3.1. *Околина $N_k(z) \subseteq S$, $k \in \{2, 3, ..., n \cdot m\}$ допустивог решења z, дефинише се као скуп решења која се добијају трансформацијом вектора z, тако што се на случајан начин одабере k позиција из скупа $\{1,\ 2, ...,\ n \cdot m\}$. Потом се одабране позиције сортирају у растућем поретку. Вредности које се налазе на одабраним позицијама циклично се померају за једно место у лево, тј. прва вредност, она која се налази на изабраној позицији са најмањим редним бројем, се помера на последње место, тј. на изабрану позицију са највећим редним бројем, а остале вредности се померају за једно место у лево, на наредну изабрану позицију. Вредности на позицијама које нису изабране, у вектору решења, остају непромењене. Додатно, уколико је $k \leq m$ потребно је проверити да ли су све вредности на изабраним позицијама исте. Уколико јесу, онда се бира нових k позиција из скупа $\{1,\ 2,\ ...,\ n \cdot m\}$. Провера се поново спроводи и поступак се понавља док се бар на једној од изабраних позиција не појави вредност која је различита.*

Овако дефинисана околина $N_k(z)$ представља скуп свих допустивих решења која се од z могу разликовати за највише k позиција.

Пример 3.3. *Нека је за инстанцу код које је $n = 3$ и $m = 2$ једно допустиво решење $z = [1,\ 3,\ 3,\ 2,\ 1,\ 2]$, и нека је $k = 3$. Нека су из скупа $\{1,\ 2,\ 3,\ 4,\ 5,\ 6\}$ на случајан начин одабране вредности $4,\ 1$ и 3 које представљају позиције на којима ће се дешавати промене унутар вектора z. Након одабира потребно је у растућем поретку сортирати вектор $[4,\ 1,\ 3]$ у који су смештене одабране позиције. Након сортирања добија се нови редослед и то $[1,\ 3,\ 4]$. Елементи који се налазе на сортираним позицијама су редом $1,\ 3$ и 2 и међусобно су различити. Вектор $[1,\ 3,\ 2]$ се ротира за једно место у лево и добија се $[3,\ 2,\ 1]$. Потом, ротирани елементи се враћају редом на позиције $1,\ 3$ и 4,*

док елементи на осталим позицијама остају непромењени. Добија се вектор $z' = [\mathbf{3},\ 3,\ \mathbf{2},\ \mathbf{1},\ 1,\ 2]$. *Допустиво решење* z' *је један сусед од* z *у околини* $N_3(z)$.

Дефиниција 3.2. *Околина* $N_1^{ls}(z) \subseteq S$ *допустивог решења* z *се састоји од укупно* $n \cdot m - 1$ *елемената добијених тако што се унутар вектора* z *елементи који се налазе на позицијама од* 1 *до* $i+1$, $i = 1, ..., n \cdot m - 1$ *циклично померају за једно место у десно, тј.* $N_1^{ls}(z) = \{z_i' \in X \mid z_i'$ *се добија тако што се унутар вектора* z *првих* i, $i = 2, ..., n \cdot m$ *елемената ротирано за једно место у десно*$\}$.

Пример 3.4. *Нека је* $z = [1,\ 3,\ 3,\ 2,\ 1,\ 2]$ *допустиво решења проблема као у примеру 3.3.* $N_1^{ls}(z)$ *је скуп од* 5 *чланова. Први члан се добија за* $i = 1$, *цикличним померањем елемената на позицијама* 1 *и* 2 *за једно место у десно, тј. добија се* $z_1 = [\mathbf{3},\ \mathbf{1},\ 3,\ 2,\ 1,\ 2]$. *Други члан се добија за* $i = 2$, *цикличним померањем елемената од* 1. *до* 3. *позиције за једно место у десно, тј. добија се* $z_2 = [\mathbf{3},\ \mathbf{1},\ \mathbf{3},\ 2,\ 1,\ 2]$. *И остали чланови околине* $N_1^{ls}(z)$ *се добијају на исти начин, повећањем вредности параметра* $i \leq 5$. *Чланови околине* $N_1^{ls}(z)$ *називају се суседима од* z.

Дефиниција 3.3. *Околина* $N_2^{ls}(z) \subseteq S$ *допустивог решења* z *садржи сва допустива решења која се могу добити од* z *заменом свака два његова узастопна елемента, тј.* $N_2^{ls}(z) = \{z_i' \in X \mid z_i'$ *је добијено тако што се унутар вектора* z *замене места елементима на позицијама* i *и* $i+1$, $i = 1, ..., n \cdot m - 1\}$.

Пример 3.5. *Нека је* $z = [1,\ 3,\ 3,\ 2,\ 1,\ 2]$ *допустиво решења проблема као у примеру 3.3.* $N_2^{ls}(z)$ *је скуп од* 5 *чланова. Први члан се добија за* $i = 1$, *заменом места елементима на узастопним позицијама* 1 *и* 2, *тј. добија се* $z_1 = [\mathbf{3},\ \mathbf{1},\ 3,\ 2,\ 1,\ 2]$. *Други члан се добија за* $i = 2$, *заменом места елементима на позицијама* 2 *и* 3, *тј. добија се* $z_2 = [1,\ \mathbf{3},\ \mathbf{3},\ 2,\ 1,\ 2]$. *Вектор* z_2 *је једнак вектору* z, *што је у дефиницији ове околине, као и околине* $N_1^{ls}(z)$ *дозвољено, за разлику од дефиниције околине* $N_k(z)$, *у којој се строго захтева да се сусед мора разликовати од текућег решења. И остали чланови околине* $N_2^{ls}(z)$ *се добијају на исти начин, повећањем вредности параметра* $i \leq 5$. *Чланови околине* $N_2^{ls}(z)$ *називају се суседима од* z.

3.2.3 Фаза размрдавања

Фаза размрдавања представља прву од две кључне компоненте VNS методе. Њена улога јесте да повећа диверзификацију у циљу спречавања завршетка

алгоритма у локалном минимуму. То се постиже тако што се унутар простора претраге S креирају околине $N_k(z)$ допустивог решења z, дефинисане у потпоглављу 3.2.2, а које су различитих величина у зависности од вредности параметра $k_{min} \leq k \leq k_{max}$, где је $2 \leq k_{min} \leq k_{max} \leq n \cdot m$. На тај начин се омогућава померање унутар простора претраге до допустивог решења које за веће околине може бити релативно далеко од тренутног оптимума.

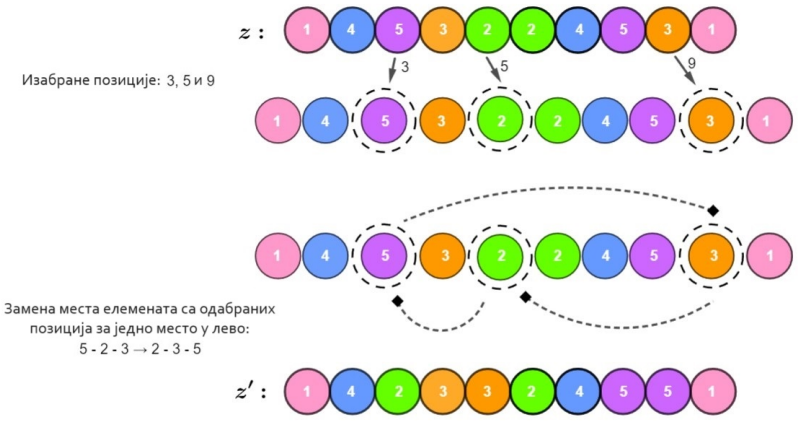
Свака итерација VNS методе почиње фазом размрдавања, у којој се на случајан начин бира ново допустиво решење $z^{'} \in N_k(z),\ z^{'} \neq z$. Услов да је $z^{'} \neq z$ обезбеђен је дефиницијом околине $N_k(z)$, а допустивост решења $z^{'}$ се не мора доказивати, обзиром на то да је $z^{'}$ једна пермутација елемената из скупа $\{\underbrace{1,\ ...,\ 1}_{m},\ \underbrace{2,\ ...,\ 2}_{m},\ ...,\ \underbrace{n,\ ...\ n}_{m}\}$, те се на тај начин добија нови редослед извршавања свих операција за све послове. Оваква имплементације фазе размрдавања с једне стране чува редослед извршавања операција, а са друге стране допушта да се унутар простора претраге удаљи довољно далеко од оптимума и да се испитају и друга допустива решења. Стога се цикличним обртањем редоследа елемената на изабраним позицијама постиже јача диверзификација, док се њиховим спајањем са непромењеним елементима добија нижи степен диверзификације. Псеудокод фазе размрдавања приказан је Алгоритмом 6.

Алгоритам 6 : *Shaking(z,k)*

```
Улазни подаци: z, k
z' := z
if k ≤ m then
    do
        pozicije ← на случајан начин изабрати k природних бројева од 1 до n · m
    while сви елементи на изабраним позицијама у вектору z имају исту вредност
else
    pozicije ← на случајан начин изабрати k природних бројева од 1 до n · m
end if
вектор pozicije сортирати у растућем поретку
for i := 1 to k do
    elementi[i] := z[pozicije[i]]
end for
ротирати вектор elementi за једно место у лево
for i := 1 to k do
    z'[pozicije[i]] := elementi[i]
end for
return z'
```

Пример 3.6. *Нека је* $z = [1,\ 4,\ 5,\ 3,\ 2,\ 2,\ 4,\ 5,\ 3,\ 1]$, *и нека је* $k = 3$. *Нека су из скупа* $\{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10\}$ *на случајан начин одабране вредности*

9, 3 и 5 *које представљају позиције на којима ће се дешавати промене унутар вектора* z. *Након одабира потребно је у растућем поретку сортирати вектор* [9, 3, 5] *у који су смештене одабране позиције. Након сортирања добија се нови редослед и то* [3, 5, 9]. *Елементи који се налазе на сортираним позицијама су редом* 5, 2 *и* 3 *и међусобно су различити. Вектор* [5, 2, 3] *се ротира за једно место у лево и добија се* [2, 3, 5]. *Потом ротирани елементи се враћају редом на позиције* 3, 5 *и* 9, *док елементи на осталим позицијама остају непромењени. Резултат фазе размрдавања је вектор* $z' = [1, 4, \mathbf{2}, 3, \mathbf{3}, 2, 4, 5, \mathbf{5}, 1]$. *Описана фаза размрдавања илустрована је на слици 3.3.*



Слика 3.3: Фаза размрдавања за инстанцу $n = 5,\ m = 2$

3.2.4 Фаза локалне претраге

Будући да је основна улога фазе размрдавања да омогући претраживање различитих делова простора претраге и спречи заглављивање алгоритма у локалном минимуму, не постоји гаранција да је допустиво решење z' које је резултат њеног извршавања локални оптимум у одговарајућој околини, штавише, често се дешава да није. Стога се VNS метода састоји и из друге фазе која следи након фазе размрдавања и назива се фаза локалне претраге. Ова фаза може бити имплементирана на више начина. Задатак јој је да детаљно претражи унапред дефинисане околине допустивог решења z' са циљем да пронађе што је могуће квалитетније решење. Околине које се сада претражују могу се разликовати од околина коришћених у фази размрдавања.

На ефикасност VNS методе велики утицај има начин на који је имплементирана фаза локалне претраге. Поред добро изабраних околина које ће се претра-

живати, неопходно је одабрати и стратегију која ће дефинисати ток алгоритма локалне претраге. Постоји више начина за то, а неки од основних су:

- **најбоље побољшање** (*engl. BEST IMPROVEMENT - BI*) - бира се најбољи сусед из околине која се претражује. Израчуна се вредност функције циља сваког суседа и изабере се онај са најбољом вредношћу исте;
- **прво побољшање** (*engl. FIRST IMPROVEMENT - FI*) - бира се први сусед са бољом вредношћу функције циља. Потом се претражује његова околина и поступак се понавља све док има побољшања;
- **произвољна селекција** (*engl. RANDOM SELECTION*) - бира се произвољан сусед са бољом вредношћу функције циља.

За решавање JIT-JSSP фаза локалне претраге је имплементирана као модификација VND методе у којој се претражују два типа околина N_1^{ls} и N_2^{ls} дефинисаних у потпоглављу 3.2.2. Приметимо да се у овом случају, околине у фази локалне претраге разликују од околина коришћених у фази размрдавања.

Фаза локалне претраге започиње претраживањем околине $N_1^{ls}(z^{'})$ у којој се за избор новог суседа користи принцип најбољег побољшања. Вектор $z^{'}$ представља допустиво решење добијено као резултат фазе размрдавања. Након завршетка ове претраге, добија се решење $z^{''}$ чији се кватитет потом проверава. Тачније, проверава се да ли је испуњен услов $f(z^{''}) < f(z^{'})$. Ако јесте испуњен, онда $z^{'} := z^{''}$ и наставља се са претрагом околине $N_1^{ls}(z^{'})$, а поступак се понавља све док има побољшања решења. Овај поступак је коначан, зато што се претрагом околине текућег решења увек бира сусед са најмањом вредношћу функције циља из те околине, те ће се на тај начин доћи до решења чији су сви суседи већ истражени, а које има најмању вредност функције циља, па поступак ту престаје са понављањем. Уколико услов $f(z^{''}) < f(z^{'})$ није испуњен, алгоритам се наставља претрагом околине $N_2^{ls}(z^{'})$ у којој се за избор новог суседа користи принцип првог побољшања. Ако се у $N_2^{ls}(z^{'})$ наиђе на решење $z^{''}$ боље од $z^{'}$, врши се ажурирање $z^{'} := z^{''}$ и поново се претражује околина $N_2^{ls}(z^{'})$ новодобијеног текућег решења. Претрага се завршава када се решење више не може поправити. Без обзира да ли се претрагом околине N_1^{ls} дошло до побољшања решења или није, претрага околине N_2^{ls} се свакако извршава, и она као резултат даје решење $z^{'}$. Како је то уједно и последњи корак у извршавању

фазе локалне претраге, добијено решење $z^{'}$ представља коначан резултат ове фазе. Псеудокод фазе локалне претраге приказан је Алгоритмом 7.

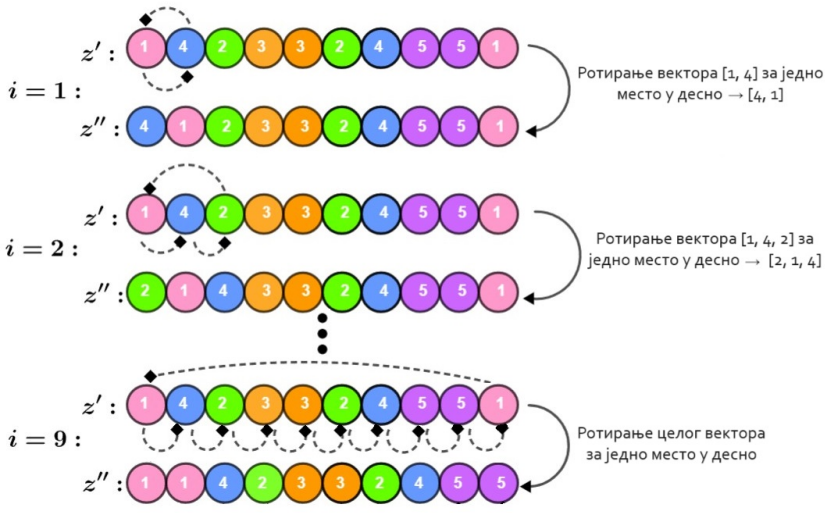
Алгоритам 7 : *LokalSearch*($z^{'}$)

Улазни подаци: $z^{'}$
do
  improvement := false ▷ нема побољшања
  $z^{''} := z^{'}$ % претраживање околине N_1^{ls}
  $i := 1$
  while $i \leq n \cdot m - 1$ **do**
    $z^{'''} := z^{'}$
    ротирати за једно место у десно елементе вектора $z^{'''}$ од позиције 1 до позиције $i+1$
    if $f(z^{'''}) < f(z^{''})$ **then**
      improvement := true ▷ дошло је до побољшања
      $z^{''} := z^{'''}$
    end if
    $i := i + 1$ ▷ best improvement
  end while
  $z^{'} := z^{''}$
while improvement = true
$i := 1$ % претраживање околине N_2^{ls}
while $i \leq n \cdot m - 1$ **do**
  $z^{''} := z^{'}$
  заменити места елементима вектора $z^{''}$ на позицијама i и $i+1$
  if $f(z^{''}) < f(z^{'})$ **then**
    $z^{'} := z^{''}$
    $i := 1$ ▷ first improvement
  else
    $i := i + 1$
  end if
end while
return $z^{'}$

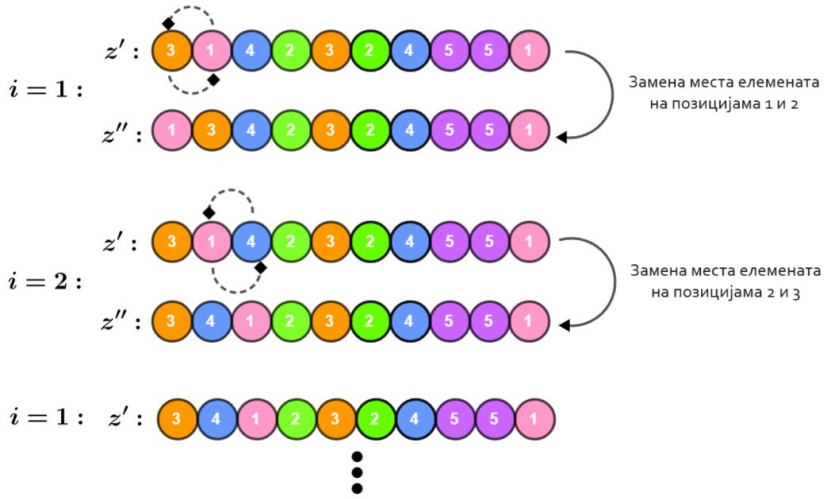
Пример 3.7. *Нека је вектор* $z^{'} = [1,\ 4,\ 2,\ 3,\ 3,\ 2,\ 4,\ 5,\ 5,\ 1]$ *добијен као резултат фазе размрдавања. Као такав узима се за почетно решење у фази локалне претраге и проглашава се за најбоље нађено решење, тј.* $z^{''} := z^{'}$. *На почетку алгоритма врши се претрага околине* $N_1^{ls}(z^{'})$. *Она започиње првом итерацијом* ($i = 1$) *у којој се елементи вектора* $z^{'}$ *циклично померају за једно место у десно, и то на позицијама* 1 *и* 2, *тј. добија се вектор* $z^{'''} = [\mathbf{4},\ \mathbf{1},\ 2,\ 3,\ 3,\ 2,\ 4,\ 5,\ 5,\ 1]$. *Потом се рачуна вредност функције циља тако добијеног решења, и проверава се да ли је дошло до побољшања, тј. проверава се да ли је* $f(z^{'''}) < f(z^{''})$. *Ако јесте, врши се ажурирање најбољег решења, тј.* $z^{''} := z^{'''}$, *иначе се решење занемарује. У другој итерацији* ($i = 2$) *ротирају се елементи вектора* $z^{'}$ *од* 1. *до* 3. *позиције за једно место у десно и добија се* $z^{'''} = [\mathbf{2},\ \mathbf{1},\ \mathbf{4},\ 3,\ 3,\ 2,\ 4,\ 5,\ 5,\ 1]$. *Затим се рачуна вредност функције циља*

добијеног решења, проверава се да ли има побољшања и врши се ажурирање z'' *ако је то потребно. Поступак се понавља све док се не дође до последње итерације* ($i = 9$)*, када се ротира цео вектор* z' *за једно место у десно и добија се* $z''' = [1,\ 1,\ 4,\ 2,\ 3,\ 3,\ 2,\ 4,\ 5,\ 5]$*. Рачуна се вредност функције циља добијеног решења, проверава његов квалитет и врше се потребна ажурирања. Након провере свих елемената из околине* $N_1^{ls}(z')$*, уколико је дошло до побољшања, тј. ако важи да је* $f(z'') < f(z')$*, онда* $z' := z''$*, а описани поступак се понавља у околини* $N_1^{ls}(z')$ *новодобијеног решења. С друге стране, уколико ниједно решење из околине* $N_1^{ls}(z')$ *није боље од текућег, тј. ако је* $f(z'') \geq f(z')$*, претрага се завршава. Нека је резултирајући вектор ове фазе локалне претраге* $z' = [3,\ 1,\ 4,\ 2,\ 3,\ 2,\ 4,\ 5,\ 5,\ 1]$.

Даље, алгоритам се наставља претраживањем околине $N_2^{ls}(z')$*. У првој итерацији ове претраге* ($i = 1$) *врши се замена места елементима на узастопним позицијама* 1 *и* 2 *и добија се* $z'' = [\mathbf{1},\ \mathbf{3},\ 4,\ 2,\ 3,\ 2,\ 4,\ 5,\ 5,\ 1,]$*. Рачуна се вредност функције циља* $f(z'')$ *и проверава се да ли је* $f(z'') < f(z')$*. Претпоставимо да није. Прелази се на другу итерацију* ($i = 2$) *и мењају се места елементима на позицијама* 2 *и* 3*. Добија се* $z'' = [3,\ \mathbf{4},\ \mathbf{1},\ 2,\ 3,\ 2,\ 4,\ 5,\ 5,\ 1]$*. Нека сада важи да је* $f(z'') < f(z')$*. У том случају* $z' := z''$ *и поступак се наставља претрагом околине* $N_2^{ls}(z')$ *новодобијеног решења. Алгоритам завршава са радом уколико се претрагом целе околине* $N_2^{ls}(z')$ *не дође до побољшања. Као резултат добија се решење* z'*. Поступак претраживања околине* $N_1^{ls}(z')$ *приказан је на слици 3.4, а околине* $N_2^{ls}(z')$ *на слици 3.5.*



Слика 3.4: Претрага околине N_1^{ls} на примеру инстанце $n = 5,\ m = 2$



Слика 3.5: Претрага околине N_2^{ls} на примеру инстанце $n = 5,\ m = 2$

3.3 Решавање потпроблема CPLEX решавачем

За сваки редослед извршавања операција који је добијен у корацима VNS методе потребно је одабрати начин на који ће се одредити време завршетка за сваку операцију појединачно, као и вредност функције циља. Како JIT-JSSP спада у проблеме линеарног програмирања, тај посао препуштен је CPLEX 22.1.0 решавачу [40]. Имплементација је урађена на основу математичког модела ILP2, с напоменом да је у овом раду разматран случај када се сваки посао $i \in N$ састоји тачно од m операција.

IBM ILOG CPLEX Optimizer је математички софтвер креиран од стране IBM компаније који је врло ефикасан за решавање различитих проблема линеарног програмирања. Развијен је пре више од тридесет година, али и данас га многе компаније користе за решавање различитих оптимизационих проблема како би смањиле оперативне трошкове и побољшале начин пословања.

За рад софтвера неопходно је унети почетне податке, задата ограничења, као и формулу за рачунање вредности функције циља. Потребно је означити о којој врсти проблема из класе проблема линеарног програмирања се ради, као и задати време рада самог софтвера. За проблеме малих димензија је веома ефикасан и релативно брзо даје оптимална решења. Међутим, код НП-тешких проблема, а нарочито оних већих димензија, што због временских, што због меморијских ограничења, CPLEX често није у могућности да достигне оптимално решење. Зато су у овом раду искоришћене његове перформансе за решавање

потпроблема, код којих је за задати редослед извршавања свих операција за све послове, добијен VNS методом, потребно одредити време завршетка за сваку операцију појединачно (што спада у проблеме полиномске сложености [71]), као и вредност функције циља.

Као резултат CPLEX даје вредност функције циља $f(z)$ за задати редослед операција, односно за допустиво решење z, као и времена завршетака за сваку операцију појединачно смештених у вектору c дужине $n \cdot m$, где се на првих m позиција, редом, налазе времена завршетака одговарајућих операција за први посао, на других m позиција за други посао и тако редом.

Пример 3.8. *Код примера описаног у поглављу 2.4, CPLEX је за задати редослед операција* $z = [1,\ 4,\ 5,\ 3,\ 2,\ 2,\ 4,\ 5,\ 3,\ 1]$ *дао следеће податке:*
Вредност функције циља: $f(z) = 59.63$*;*
Времена завршетка операција: $c = [26,\ 93,\ 54,\ 68,\ 50,\ 106,\ 28,\ 88,\ 35,\ 80]$.
То значи да је укупна казна за ранији или каснији завршетак за све операције једнака 59.63. Још, знамо да се прва операција за први посао завршава у 26. јединици времена, друга операција за први посао у 93, прва операција за други посао у 54. и тако редом. Редослед извршавања операција на машинама приказан је на слици 2.1.

3.4 Структура алгоритма

Како је већ наведено у тексту изнад, костур алгоритма имплементиране матхеуристике чини алгоритам VNS методе. Након одабира репрезентације решења и учитавања улазних података на основу којих се врше сва потребна рачунања, генерише се почетно решење z_0 и узима се за апроксимацију тренутно најбољег решења, тј. $z^* := z_0$, а њему одговарајућа вредност функције циља (добијена CPLEX-ом) за најбољу, тј. $f^* := f(z_0)$.

Свака итерација алгоритма матхеуристике започиње фазом размрдавања, која као резултат даје допустиво решење $z^{'}$ из околине текућег најбољег $N_k(z^*)$. Оно се потом прослеђује као почетно решење у фази локалне претраге, где се претраживањем његових околина на већ описани начин добија ново допустиво решење $z^{''}$. Потом се упоређују вредности функције циља новодобијеног и најбољег решења (добијене CPLEX-ом), тј. проверава се да ли је испуњен услов $f(z^{''}) < f^*$.

- Уколико је услов испуњен, онда се врши ажурирање најбољег решења $z^* := z^{''}$, најбоље вредности функције циља $f^* := f(z^{''})$ и параметра k на најмању вредност $k := k_{min}$. Потом алгоритам прелази у своју наредну итерацију истражујући прву околину $N_{k_{min}}(z^*)$ новодобијеног најбољег решења.

- Уколико дата неједнакост није испуњена, онда се k повећава за 1 и прелази се у следећу итерацију алгоритма фазом размрдавања у наредној околини текућег најбољег решења $N_{k+1}(z^*)$.

Када k достигне максималну вредност k_{max} ажурира се на минималну, тј. $k := k_{min}$ и алгоритам се наставља у наредној итерацији фазом размрдавања у околини $N_{k_{min}}(z^*)$.

Поступак се понавља до испуњења задатог критеријума заустављања, у овом случају до достизања унапред задатог броја итерација.

Начини репрезентације решења и рачунања вредности функције циља изложени су у поглављима 3.1 и 3.3, а иницијализација почетног решења, фаза размрдавања и фаза локалне претраге описани су у потпоглављима 3.2.1, 3.2.3 и 3.2.4 редом. Псеудокод имплементиране матхеуристике за решавање JIT-JSSP приказан је Алгоритмом 8.

Алгоритам 8 : Матхеуристика за решавање JIT-JSSP

```
Учитавање инстанце
Генерисање почетног решења z_0
Ажурирање: z* := z_0, f* := f(z_0)          ▷ вредност f(z_0) се добија CPLEX решавачем
Ажурирање: k := k_min, brojIteracija := 1
while brojIteracija ≤ maxIter do
    z' := Shaking(z*, k)                                  ▷ фаза размрдавања
    z'' := LocalSearch(z')                                ▷ фаза локалне претраге
    if f(z'') < f* then                  ▷ вредност f(z'') се добија CPLEX решавачем
        z* := z''
        f* := f(z'')
        k := k_min
    else if k = k_max then
        k := k_min
    else
        k := k + 1
    end if
    brojIteracija := brojIteracija + 1
end while
Исписивање добијених резултата
```

Глава 4

Експериментални резултати

У овом одељку биће описани тест примери који су коришћени за тестирање матхеуристике, презентовани добијени резултати као и њихово поређење са резултатима доступним у литератури.

4.1 Тест примери - опис

Имплементирана матхеуристика тестирана је на укупно 72 инстанце које су доступне у литератури [68]. Креиране су тако што је за свако $n \in \{10,\ 15,\ 20\}$, које представља број послова и свако $m \in \{2,\ 5,\ 10\}$, које представља број машина генерисано по 8 тест примера на начин који ће бити описан у даљем тексту. Инстанце су формиране и први пут презентоване у литератури у [68] из 2008. године. Назив инстанце је облика „***dd-w_ id_ n*×*m***", при чему само име описује неке њене карактеристике и то:

- „**dd**"- садржи информације о времену завршетка узастопних операција за исти посао, и то на следећи начин :
  - „$dd = tight$(чврсто)"- време завршетка тренутне операције поклапа се са почетком наредне;
  - „$dd = loose$(лабаво)"- време почетка наредне операције једнако је времену завршетка тренутне плус произвољна вредност из опсега $[0, 10]$;
- „**w**" - садржи информације о односу казнених коефицијената за превремени завршетак (α) и кашњење (β) за сваку операцију, на следећи начин:

  - „w = *equal*(једнако)"- када су α и β произвољно изабрани из интервала [0.1, 1];
  - „w = *tard*(касно)"- када је α изабрано из интервала [0.1, 0.3], а β из интервала [0.1, 1], где се фаворизује превремени завршетак у односу на кашњење, што одговара већем броју ситуација из свакодневног живота;

- „**id ∈ {1, 2}**"- за сваку комбинацију параметара генерисана су по два тест примера.

Тест примери су задати у формату који је приказан на слици 4.1.

```
5 2
0 20 44 0.96 0.46                1 13 75 0.70 0.14
0 19 54 0.77 0.53                1 8 68 0.86 0.91
1 13 50 0.72 0.74                0 18 87 0.52 0.77
1 7 28 0.62 0.62                 0 34 70 0.18 0.95
0 9 44 0.70 0.93                 1 12 74 0.97 0.30
```

Слика 4.1: Формат тест примера

У првом реду инстанце налазе се две вредности. Прва од њих се односи на број послова (n), а друга на број машина (m). Потом, сваки од наредних n редова односи се на по један посао где су подаци за њега груписани у m блокова од по пет вредности. Сваки блок описује по једну операцију одговарајућег посла, почевши од прве, редом све до m-те. За сваку операцију дати су следећи подаци: на првом месту се налази редни број машине (почевши од нула) на којој се операција извршава, на другом месту је дато време трајања операције, на трећем презентовано је идеално време завршетка операције, на четвртом задат је казнени коефицијент за превремени завршетак, а на петом казнени коефицијент за кашњење операције.

Пример 4.1. *Посматрајући инстанцу презентовану на слици 4.1 из података датих у првом реду може се закључити да се ради о тест примеру са* $n = 5$ *послова и* $m = 2$ *машине. Стога, остале информације ће бити груписане у пет редова са по два блока од по пет вредности. Подаци за први посао налазе се у другом реду инстанце. На првом месту налази се информација да се операција* o_1^1 *извршава на нулотој машини, да траје 20 јединица времена, да јој је*

идеално време завршетка 44. јединица времена, да је казнени коефицијент за превремени завршетак једнак 0.96, а за кашњење 0.46 по јединици одступања времена завршетка од идеалног времена завршетка операције. Операција o_1^2 се извршава на првој машини, траје 13 јединица времена, идеално време завршетка јој је у 75. јединици времена, казнени коефицијент за превремени завршетак износи 0.7, а за кашњење 0.14 по јединици одступања времена завршетка од идеалног времена завршетка операције. У трећем реду инстанце налазе се одговарајући подаци за други посао и тако редом.

Додатно, на основу презентованих података може се уочити да за исти посао време завршетка прве операције се не поклапа са временом почетка наредне. На пример, идеално време завршетка операције o_1^1 је у 44, а операције o_1^2 у 75. јединици времена. Операција o_1^2 траје 13 јединица времена, што би значило да њено извршавање треба да започне у (75-13=)62. јединици времена, а 62≠44, те ова инстанца има карактеристику „dd = loose". Потом, α и β имају произвољно изабране вредности из интервала [0.1, 1], те јој је друга карактеристика „w = equal". Ово је прва, од две генерисане инстанце са истим параметрима, па је „id = 1". Из наведених чињеница закључује се да је назив ове инстанце „loose-equal_1_5×2".

4.2 Презентација добијених резултата

Имплементација матхеуристике која се састоји из VNS алгоритма у хибридизацији са егзактним CPLEX решавачем извршена је у програмском језику C++. Тестирања су спроведена на Windows 11 Pro оперативном систему на лап-топ рачунару са процесором ADM Ryzen 5 5500U од 2.10GHz и 16GB RAM меморије. За сваки тест пример програм је покренут по 20 пута за различите вредности на основу којих се генеришу случајни бројеви, тј. за различите вредности тзв. SEED-ова.

У табелама 4.1, 4.2 и 4.3 редом су презентовани добијени резултати за тестиране инстанце са 10, 15 и 20 послова. Свака од њих садржи следеће податке:

- ***Инстанце*** - име инстанце;
- $\boldsymbol{best_{sol}}$ - вредност функције циља оптималног решења (ако је познато) или вредност функције циља најбољег решења познатог у литератури. Уколико је оптималност решења доказана, поред наведене нумеричке вред-

ности стоји ознака „(опт)". Позната оптимална решења из литературе добијена су CPLEX решивачем коме је време рада било ограничено на 30 минута и презентоване су у [9];

- ***Литература*** - референце на радове у којима су презентоване одговарајуће $best_{sol}$ вредности;
- $\boldsymbol{z_{best}}$(r)- вредност функције циља најбољег решења добијеног имплементираном матхеуристиком у 20 извршавања, где вредност r у загради даје информацију колико пута се z_{best} достигло у тих 20 извршавања;
- $\boldsymbol{z_{avg}}$ - просечна вредност функције циља решења добијених у 20 извршавања;
- $\boldsymbol{t_{best}}$ - просечно време извршавања за које се први пут добило најбоље решење у 20 покретања (у минутима);
- $\boldsymbol{t_{tot}}$ - просечна вредност укупног времена извршавања у 20 покретања (у минутима);
- $\boldsymbol{iter_{best}}$ - просечна вредност броја итерација за који се први пут добило најбоље решење у 20 покретања;
- $\boldsymbol{agap\ \%}$ - просечно процентуално одступање вредности функције циља нађених решења у односу на оптимално решење (уколико је познато) или најбоље познато решење, израчунато по формули:
  $\boldsymbol{agap = \frac{1}{20}\sum_{i=1}^{20} gap_i}$, где је $\boldsymbol{gap_i = 100 \cdot \frac{|c_i - best_{sol}|}{|best_{sol}|}}$;
- $\boldsymbol{\sigma}$ % - стандардна девијација просечног одступања вредности функције циља нађених решења (у процентима), израчуната по формули:
  $\boldsymbol{\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}}$.

Параметри које треба задати VNS методи су k_{min}, k_{max} и max_{iter}, док је CPLEX-у неопходно ограничити време рада, тј. потребно је задати $time_{cplex}$. Ове вредности одређене су експерименталним путем у зависности од укупног броја операција за све послове (број послова n помножен бројем операција за сваки посао $m \ \rightarrow \ n \cdot m$), те код посматраних инстанци креће се између 20 и 200. Треба напоменути да се број машина поклапа са бројем операција које је потребно извршити за један посао, тј. свакој операцији за посматрани посао одговара тачно једна машина из задатог скупа.

- $\boldsymbol{k_{min} = m}$ - најмањи ред околине једнак је броју операција за један посао. Одабир овог параметра је у директној вези са изабраном репрезентацијом решења;
- $\boldsymbol{k_{max} = \lfloor 2.5 \cdot m \rfloor}$ - највећи ред околине једнак је целом делу броја добијеног тако што се број операција за један посао помножи са $\frac{5}{2}$;
- $\boldsymbol{max_{iter}}$ - максималан број итерација зависи од укупног броја операција које је потребно извршити, тј. зависи од вредности $n \cdot m$ и задат је на следећи начин:
  - за $n \cdot m < 30$, $\boldsymbol{max_{iter} = 500}$
  - за $30 \leq n \cdot m < 40$, $\boldsymbol{max_{iter} = 1000}$
  - за $n \cdot m \geq 40$, $\boldsymbol{max_{iter} = 1750}$;
- $\boldsymbol{time_{cplex} = 1s}$ - време рада CPLEX-а ограничено је на 1 секунду.

Инстанце са 10 послова

Анализом резултата добијених тестирањем 24 инстанце са 10 послова изложених у табели 4.1 може се уочити да:

- За оне тест примере код којих је број машина једнак 2 у литератури је познато оптимално решење за 5 од укупно 8 инстанци. Исте те вредности добијене су и имплементираном матхеуристиком. За инстанце „loose-equal_2_10×2", „tight-equal_1_10×2" и „tight-tard_2_10×2" нема информација о оптималном решењу, али вредности које је дала матхеуристика поклапају се са најбољим решењима из литературе. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег или оптималног решења добијеног изложеном методом за 8 тест примера креће се између 102.7 и 236.4.

Вредности *agap* и σ процењују квалитет добијених решења и ако су одступања за *agap* мања од 5%, онда се може сматрати да су добијени резултати квалитетни. За 5 од 8 инстанци најбоље решење добијено је у најмање 17 покретања програма, па су и вредности за *agap* и σ мање од 0.1% , односно 0.48%. За инстанцу „tight-tard_2_10×2" обе вредности једнаке су 0%, што значи да је у свих 20 тестирања добијена иста (најбоља) вредност. За преостале 3 инстанце, најбоље решење добијено је више од 11 пута, али у осталим извршавањима алгоритма јавила су се мања одступања што утиче на вредности *agap* и σ, које су ниже од 0.83% , односно 2.01% .

- Ни за једну од 8 инстанци код којих је број машина једнак 5 у литератури нема података о оптималном решењу. Имплементирана матхеуристика је достигла најбоље познато решење за 7 од укупно 8 тест примера. Код инстанце „tight-equal_2_10×5" добијена вредност је за 0.1% већа од најбоље у литератури. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног изложеном методом за 8 тест примера креће се између 847.05 и 1320.6.

Вредности за *agap* и σ варирају у зависности од тест примера. Најмање су забележене код инстанце „loose-equal_2_10×5" код које се 11 пута достигла најбоља вредност, и износе 0.57%, односно 1.37%, а највеће код инстанце „loose-tard_2_10×5" и то 7.11% , односно 5.34%. За преостале тест примере вредност *agap* не прелази 3.95%, а σ није већа од 3.26%.

- За 8 инстанци код којих је број машина једнак 10 у литератури, такође нема информација о оптималним решењима. Имплементирана матхеуристика је до најбољег познатог решења дошла само за инстанцу „loose-tard_2_10×10", и то у 2 од 20 покретања програма. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног изложеном методом за 8 тест примера креће се између 1151 и 1390.6.

Вредности за *agap* и σ варирају у зависности од тест примера и крећу се у распону од 4.8% до 13.7%, односно од 3.68% до 6.87%.

Табела 4.1: Резултати тестирања за инстанце са 10 послова (n = 10)

Инстанце	$best_{sol}$	*Литература*	z_{best}	z_{avg}	t_{best}	t_{tot}	$iter_{best}$	$agap(\%)$	$\sigma(\%)$
loose-equal_1_10×2	**224.84**$^{(onm)}$	[88, 9, 98]	**224.84**(18)	224.93	1.27	3.48	175.8	0.04	0.24
loose-equal_2_10×2	**319.37**	[9, 98]	**319.37**(11)	322.07	1.38	3.08	221.1	0.83	1.36
loose-tard_1_10×2	**416.44**$^{(onm)}$	[88, 9, 98]	**416.44**(12)	417.98	1.24	3.32	182.6	0.37	0.75
loose-tard_2_10×2	**137.94**$^{(onm)}$	[88, 9]	**137.94**(16)	138.91	1.69	3.57	236.4	0.66	2.01
tight-equal_1_10×2	**461.96**	[88, 9, 98]	**461.96**(18)	462.27	1.16	2.89	194.4	0.07	0.34
tight-equal_2_10×2	**448.32**$^{(onm)}$	[88, 9, 98]	**448.32**(17)	448.79	1.06	2.95	178.9	0.1	0.48
tight-tard_1_10×2	**179.64**$^{(onm)}$	[88, 9, 98]	**179.64**(18)	179.49	0.98	3.49	140.5	0.02	0.14
tight-tard_2_10×2	**145.37**	[88, 9, 98]	**145.37**(20)	145.37	0.79	3.60	102.7	0	0
loose-equal_1_10×5	**1719.63**	[9]	**1719.63**(1)	1783.9	23	43.3	933.1	3.53	3.23
loose-equal_2_10×5	**967.73**	[9, 98]	**967.73**(11)	973.38	23.18	47.59	847.05	0.57	1.37
loose-tard_1_10×5	**175.08**	[88, 9, 98]	**175.08**(1)	181.41	61.68	93.36	1173.6	3.44	2.83
loose-tard_2_10×5	**485.06**	[9]	**485.06**(1)	523.54	29.14	42.73	1193.7	7.11	5.34
tight-equal_1_10×5	**689.11**	[88, 9, 98]	**689.11**(1)	718.01	34.03	52.7	1150.8	3.95	3.26
tight-equal_2_10×5	**763.24**	[9]	764.03(1)	786.29	30.35	40.46	1320.6	2.91	2.26
tight-tard_1_10×5	**379.19**	[9]	**379.19**(5)	384.8	55.49	78.13	1238.7	1.43	1.91
tight-tard_2_10×5	**627.45**	[88, 9]	**627.45**(7)	636.05	26.55	40.76	1137.9	1.31	2.28
loose-equal_1_10×10	**360.74**	[9]	375.9(1)	408.46	121.83	171.65	1257.2	11.32	6.49
loose-equal_2_10×10	**249.85**	[88, 9]	256.13(1)	277.75	101.57	137.50	1293.7	9.75	5.97
loose-tard_1_10×10	**374.2**	[9]	386.51(1)	409.31	99.23	150.22	1151	8.39	4.96
loose-tard_2_10×10	**144.94**	[88, 9]	**144.94**(2)	154.18	108.65	140.6	1344.8	5.58	6.35
tight-equal_1_10×10	**1276.23**	[88, 9]	1303.24(1)	1429.01	104.21	137.45	1333.4	10.25	6.87
tight-equal_2_10×10	**1866.92**	[9]	1939.47(1)	2121.26	139.09	187.48	1295.7	11.85	4.94
tight-tard_1_10×10	**668.14**	[9]	712.2(1)	777.63	124.88	159.47	1368.9	13.7	6.59
tight-tard_2_10×10	**777.85**	[9]	779.17(1)	817.86	161.73	205.51	1390.6	4.8	3.68

Подебљане вредности у колони z_{best} показују да се имплементираном матхеуристиком дошло до оптималног/најбољег познатог решења.

Инстанце са 15 послова

Анализом презентованих резултата добијених тестирањем 24 инстанце са 15 послова изложених у табели 4.2 може се уочити следеће:

- Ни за једну од 8 инстанци код којих је број машина једнак 2 у литератури нема података о оптималном решењу. Имплементирана матхеуристика дала је најбоља позната решења за свих 8 тест примера. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног овом методом за 8 тест примера креће се између 580.4 и 752.7.

За 5 од 8 инстанци најбоље решење добијено је у најмање 8 покретања програма. Код преосталих резултата јављају се мала одступања, те су и вредности за *agap* и σ ниске, и то мање од 0.27%, ондосно 0.62%. Најмање вредности ових параметара забележене су код инстанце „tight-tard_2_15×2" и износе 0.17%, односно 0.53%. За преостале 3, најбоље решење добијено овом матхеуристиком забележено је у 2 или 6 покретања програма, а вредности за *agap* и σ су ниже од 2.89%, односно 2.63%.

- За групу инстанци код којих је број машина једнак 5 у литератури нема информација о оптималним решењима. Имплементираном матхеуристиком није се дошло до најбољих решења познатих у литератури ни за један од укупно 8 тест примера. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног изложеном методом за 8 тест примера креће се између 1255.5 и 1523.6.

Вредности за *agap* и σ варирају у зависности од тест примера. Најмање су забележене код инстанце „tight-tard_2_15×5" и износе 3.7%, односно 2.3%, а највеће код инстанце „loose-tard_1_15×5" и то 9.86%, односно 4.73%.

- За групу инстанци код којих је број машина једнак 10 у литератури нема информација о оптималним решењима. Матхеуристика није достигла најбоља решења позната у литератури ни за један од укупно 8 тест примера. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног изложеном методом за 8 тест примера креће се између 1326.3 и 1454.6.

Вредности за *agap* су веће од 20%, а за σ све прелазе 5%. Најмање су забележене код инстанце „tight-tard_1_15×10" и износе 13.92%, односно 5.52%, а највеће код инстанце „loose-tard_2_15×10" и то 45.76% , односно 9.76%.

Табела 4.2: Резултати тестирања за инстанце са 15 послова (n = 15)

Инстанце	$best_{sol}$	*Литература*	z_{best}	z_{avg}	t_{best}	t_{tot}	$iter_{best}$	$agap(\%)$	$\sigma(\%)$
loose-equal_1_15×2	**1041.33**	[88, 9, 98]	**1041.33**(9)	1043.55	7.96	12.44	635.8	0.21	0.62
loose-equal_2_15×2	**497.97**	[88, 9]	**497.97**(6)	505.18	8.89	13.33	681.5	1.41	1.68
loose-tard_1_15×2	**654.84**	[88, 9, 98]	**654.84**(10)	656.39	7.96	13.72	580.4	0.24	0.57
loose-tard_2_15×2	**279.71**	[88]	**279.71**(6)	288.14	8.93	13.52	663.4	2.89	2.63
tight-equal_1_15×2	**3344.54**	[88, 9, 98]	**3344.54**(8)	3353.73	8.04	11.8	678	0.27	0.58
tight-equal_2_15×2	**1479.76**	[88, 9, 98]	**1479.76**(9)	1483.02	8.79	11.55	752.7	0.22	0.59
tight-tard_1_15×2	**790.5**	[88, 9]	**790.5**(2)	797.66	10.87	14.85	734.5	0.89	1.19
tight-tard_2_15×2	**905.37**	[88, 9, 98]	**905.37**(11)	906.92	9.36	14.71	634	0.17	0.53
loose-equal_1_15×5	**3220.16**	[88]	3256.73(1)	3448.35	65.95	84.51	1361.9	6.5	4.12
loose-equal_2_15×5	**3260.81**	[9]	3439.79(1)	3621.27	89.89	114.98	1373.9	9.84	4.46
loose-tard_1_15×5	**1281**	[9]	1357.32(1)	1423.35	81.25	102.12	1397.6	9.86	4.73
loose-tard_2_15×5	**328.26**	[9]	340.5(1)	358.06	164	215.5	1334.4	8.26	3.71
tight-equal_1_15×5	**1342.3**	[9]	1399.64(1)	1423.66	80.16	100.46	1392.6	5.7	2.61
tight-equal_2_15×5	**2630.06**	[88]	2707.59(1)	2805.61	73.61	91.26	1408	6.23	3.03
tight-tard_1_15×5	**1359.18**	[9]	1366.67(1)	1427.06	72.59	101.04	1255.5	4.71	3.04
tight-tard_2_15×5	**681.27**	[9]	692.08(1)	707.55	132.74	151.69	1523.6	3.7	2.3
loose-equal_1_15×10	**875.74**	[9]	1196.55(1)	1271.19	523.16	732.94	1338.7	31	6.18
loose-equal_2_15×10	**1487.33**	[9]	2019.65(1)	2218.62	477.52	639.24	1326.3	32.65	7.22
loose-tard_1_15×10	**277**	[9]	334.31(1)	368	493.4	671.68	1326.9	24.6	5.88
loose-tard_2_15×10	**597.93**	[9]	901.96(1)	1122.8	240.41	309	1367.65	45.76	9.76
tight-equal_1_15×10	**6820.87**	[9]	8222.82(1)	8963.41	182.77	228.83	1399.2	23.71	6.12
tight-equal_2_15×10	**4760.48**	[9]	5657.76(1)	6089.26	322.12	387	1454.6	21.66	5.83
tight-tard_1_15×10	**767.26**	[9]	833.11(1)	893.34	1269.47	1723.13	1418.4	13.92	5.52
tight-tard_2_15×10	**1224.82**	[9]	1521.14(1)	1750.25	466.66	569.49	1429.9	29.5	8.08

Подебљане вредности у колони z_{best} показују да се имплементираном матхеуристиком дошло до оптималног/најбољег познатог решења.

Инстанце са 20 послова

На основу резултата изложених у табели 4.3 за 24 тестиране инстанце са 20 послова може се уочити следеће:

- Ни код једног од 8 тест примера код којих је број машина једнак 2 у литератури нема података о оптималном решењу. За инстанце „loose-equal_1_20×2", „loose-tard_1_20×2" и „tight-equal_1_20×2" имплементирана матхеуристика дала је боља решења од до сада најбољих познатих у литератури. За „loose-equal_2_20×2" и „loose-tard_2_20×2" достигла је најбоља позната решења, док за преостале 3 инстанце добијене вредности су веће од најбољих познатих, али су одступања мала и крећу се од 0.12% до 0.3%. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног изложеном методом за 8 тест примера креће се између 1196.2 и 1459.4.

Вредности параметра *agap* су у 7 од 8 случајева ниже од 1%, а σ од 1.2%. Најмање су забележене код инстанце „loose-equal_1_20×2" и износе 0.32%, односно 0.63%, а највеће код инстанце „loose-equal_2_20×2" и то 1.36%, односно 1.92%.

- Код групе инстанци где је број машина једнак 5 у литератури нема информација о оптималним решењима. Имплементираном матхеуристиком се није дошло до најбољих решења познатих у литератури ни за један од укупно 8 тест примера. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног изложеном методом за 8 тест примера креће се између 1364.5 и 1564.8.

Вредности за *agap* и σ варирају у зависности од тест примера. Најмање су забележене код инстанце „tight-tard_1_20×5" и износе 4.51%, односно 2.54%, а највеће код инстанце „loose-tard_1_20×5" и то 13.74%, односно 5.26%.

- За групу инстанци код којих је број машина једнак 10 у литератури нема информација о оптималним решењима. Матхеуристика није достигла најбоље решења познато у литератури ни за један од укупно 8 тест примера. Просечан број итерација у 20 извршавања програма за који се дошло до најбољег решења добијеног изложеном методом за 8 тест примера креће се између 1339.4 и 1528.6.

Вредности за *agap* су веће од 20%, а за σ све прелазе 5%. Најмање су забележене код инстанце „tight-equal_1_20×10" и износе 19.61%, односно 5.59%, а највеће код инстанце „loose-equal_2_20×10" и то 40.9% , односно 7.42%.

Табела 4.3: Резултати тестирања за инстанце са 20 послова (n = 20)

Инстанце	$best_{sol}$	*Литература*	z_{best}	z_{avg}	t_{best}	t_{tot}	$iter_{best}$	$agap(\%)$	$\sigma(\%)$
loose-equal_1_20×2	2548.95	[88, 9]	**2547.68**(1)	2555.9	26.29	34.79	1323.8	0.32	0.63
loose-equal_2_20×2	**3069.13**	[88, 9]	**3069.13**(3)	3112.05	31.76	39.51	1419.5	1.36	1.92
loose-tard_1_20×2	1205.59	[98]	**1205.19**(1)	1210.03	29.11	42.58	1196.2	0.4	0.75
loose-tard_2_20×2	**769.35**	[88, 9]	**769.35**(1)	776.81	31.96	46	1224.5	0.96	1.18
tight-equal_1_20×2	1936.45	[9]	**1936.24**(1)	1943.71	31.12	37.36	1459.4	0.38	0.68
tight-equal_2_20×2	**941.69**	[88]	943.7(2)	949.8	31.18	38.96	1399.7	0.85	1
tight-tard_1_20×2	**1671.87**	[88]	1674(1)	1686.15	31.8	43.17	1294.1	0.84	1.07
tight-tard_2_20×2	**1448.66**	[88]	1452.97(1)	1462.14	28.8	42.68	1223.6	0.92	1.07
loose-equal_1_20×5	**7524.13**	[9]	8099.98(1)	8388.64	136.08	167.64	1415.5	10.24	4.03
loose-equal_2_20×5	**7059.38**	[9]	7500.54(1)	7939.61	146.46	174.59	1462.5	11.03	3.98
loose-tard_1_20×5	**2931.11**	[9]	3211.94(1)	3404.63	155.78	180.01	1515.4	13.74	5.26
loose-tard_2_20×5	**3627.51**	[9]	3775.83(1)	4041.57	134.72	171.03	1364.5	10.14	4.44
tight-equal_1_20×5	**2865.95**	[9]	2998.84(1)	3103.67	244.21	300.08	1420.1	7.63	3.27
tight-equal_2_20×5	**6613.62**	[9]	7013.3(1)	7226.59	136.3	153.13	1564.8	8.45	3.4
tight-tard_1_20×5	**3620.61**	[9]	3703.43(1)	3792.44	140.84	170.6	1437.6	4.51	2.54
tight-tard_2_20×5	**1792.78**	[9]	1861.73(1)	1936.55	215.56	245.26	1535.2	7.38	3.33
loose-equal_1_20×10	**4651.31**	[9]	6132.44(1)	6753.14	803.52	996.01	1421.4	30.8	7.26
loose-equal_2_20×10	**1597.89**	[9]	2425.21(1)	2714.81	760.86	1000.53	1349.6	40.9	7.42
loose-tard_1_20×10	**4817.32**	[9]	6915.78(1)	7386.31	535.14	634.88	1471	34.63	6.64
loose-tard_2_20×10	**1401.28**	[9]	1836.85(1)	1972.16	6700.94	8354.26	1528.6	28.76	6.42
tight-equal_1_20×10	**10378.8**	[9]	12116.2(1)	12934.1	511.23	649.05	1415.2	19.61	5.59
tight-equal_2_20×10	**6839.3**	[9]	8074.47(1)	8658.11	2639.69	4150.27	1339.4	20.85	5.76
tight-tard_1_20×10	**4445.59**	[9]	5800.38(1)	6307.7	623.81	756.71	1436.8	29.37	6.31
tight-tard_2_20×10	**3085.54**	[9]	3713.52(1)	4133.69	5517.11	8379.66	1370	25.17	6.26

Подебљане вредности у колони z_{best} показују да се имплементираном матхеуристиком дошло до оптималног/најбољег познатог решења.

4.3 Поређење са резултатима из литературе

У овом поглављу биће презентовани резултати ранијих решавања JIT-JSSP методама предложеним у литератури, као и њихово поређење са резултатима добијеним имплементираном матхеуристиком.

Једна од метода која је коришћена за решавање JIT-JSSP јесте хибридизација еволутивног алгоритма, локалне претраге и методе математичког програмирања (EA+MP+LS), а њени резултати су изложени у [88]. Друга метода је хибридизација методе променљивих околина и методе математичког програмирања (VNS+MP), чији су резултати, заједно са најбољим доњим и горњим границама решења презентовани у [98]. Потом, JIT-JSSP је решаван методом променљивих околина (VNS), као и CPLEX решавачем, а добијени резултати су изложени у [9]. Алгоритми EA+MP+LS и VNS+MP су поново имплементирани по упутствима датим у [88] и [98], редом. Тестирани су на истом рачунару као и VNS метода и CPLEX решавач (по 5 покретања програма за сваку инстанцу), а добијена решења, такође су презентована у [9].

Резултати добијени матхеуристиком изложеном у овом раду упоређени су са резултатима презентованим у [9]. У табелама 4.4, 4.5 и 4.6 приказане су добијене вредности за инстанце са 10, 15 и 20 послова, редом. Прва колона (*Инстанце*) представља име инстанце. Друга (LB) и трећа (UB), редом, представљају најбоље доње и горње границе решења. У четвртој (z) се налази решење добијено CPLEX-ом коме је време рада ограничено на 30 минута, а у петој ($time_{cplex}$) време за које је CPLEX дошао до тог решења. Потом, организовано по блоковима, за сваку од хеуристика EA+MP+LS, VNS+MP и VNS, редом, изложени су следећи подаци: у првој колони блока (z_{best}) смештено је најбоље решење добијено одговарајућом хеуристиком, док се у другој колони (t_{tot}) налази просечно укупно време извршавања алгоритма у 5 покретања програма. У последњем блоку именованом „Матхеуристика (VNS+CPLEX)" презентовани су резултати добијени у овом раду, као и одговарајућа времена. Задња врста садржи податке о укупном броју инстанци за које је достигнуто оптимално или најбоље познато решење, редом, по методама којима је проблем решаван.

Инстанце са 10 послова

У табели 4.4 дат је преглед резултата добијених тестирањем различитих метода предложених у литератури за 24 инстанце са 10 послова.

Метода изложена у овом раду достигла је оптимална или најбоља позната решења за 16 од укупно 24 тест примера (66.7%). CPLEX решавач је то постигао за 13 од укупно 24 инстанце (54.2%), од чега је добијено 5 оптималних решења и 8 решења за која оптималност није доказана. Поређењем резултата које је дала имплементирана матхеуристика са резултатима добијеним CPLEX-ом, може се уочити да је она добила боља решења за 5, иста за 12, а лошија за 7 тест примера. Код инстанци са 2 машине, за краће време рада, матхеуристика је дошла до решења истог квалитета као и CPLEX. За оне тест примере код којих је број машина једнак 5, матхеуристика је дала знатно боље резултате, али је и време извршавања програма дуже од максималних 30 минута додељених CPLEX-у. Код инстанци са 10 машина CPLEX је дао боље резултате за краће време рада. Хибридизација EA+LS+MP којој је време рада било ограничено на 1.6 минута, дала је оптимална или најбоља позната решења, такође за 13 од укупно 24 инстанце. У односу на њу, метода изложена у овом раду, за дуже време извршавања, добила је боља решења за 8, иста за 11, а лошија за 5 инстанци. Поређењем са резултатима добијеним хибридизацијом VNS+MP којој је, такође време рада било ограничено на 1.6 минута, а која је достигла оптимална или најбоља позната решења за 11 од 24 инстанце (45.8%), имплементирана матхеуристика, за дуже време рада, дала је боља решења за 7, иста за 12, а лошија за 5 тест примера. Потом, анализирајући резултате добијене VNS методом, а која је достигла оптимална или најбоља позната решења за 23 од укупно 24 инстанце (95.8%), може се уочити да је за краће време рада дала иста решења као и матхеуристика за 16 тест примера, боља за 8 и једино за инстанцу „loose-equal_1_10×10" није успела да достигне најбоље познато решење у литератури, а које је добио једино CPLEX.

Посматрајући доње и горње границе може се уочити да се најбоља нађена решења за 22 тест примера налазе у интервалу (LB, UB), док је код инстанце „loose-tard_1_10×2" изложена горња граница нижа од оптималног решења што указује на грешку, те вредност за ову границу презентовану у [98] треба узети са извесном резервом. Код инстанце „tight-equal_1_10×2" вредност горње границе је такође мања од вредности најбољег познатог решења, али како нема података о оптималној вредности решења за овај тест пример, информације о квалитету решења нису потпуне. Што се резултата добијених матхеуристиком тиче, све вредности, осим две наведене, налазе се у одговарајућем опсегу доњих и горњих граница решења.

Табела 4.4: Преглед свих добијених резултата за инстанце са 10 послова (n = 10)

Инстанце	*LB* [68]	*UB* [68]	*CPLEX* [9]		*EA + MP + LS* [88]		*VNS + MP* [98]		*VNS* [9]		Матхеуристика (VNS+CPLEX)	
			z	$time_{cplex}$	z_{best}	t_{tot}	z_{best}	t_{tot}	z_{best}	t_{tot}	z_{best}	t_{tot}
loose-equal_1_10×2	219	225	**224.84**$^{(onm)}$	26.64	**224.84**(5)	1.6	**224.84**(5)	1.6	**224.84**(5)	0.62	**224.84**(18)	3.48
loose-equal_2_10×2	313	324	**319.37**	30	324.43(5)	1.6	**319.37**(1)	1.6	**319.37**(3)	0.59	**319.37**(11)	3.08
loose-tard_1_10×2	416	416*	**416.44**$^{(onm)}$	4.25	**416.44**(5)	1.6	**416.44**(4)	1.6	**416.44**(5)	0.16	**416.44**(12)	3.32
loose-tard_2_10×2	137	138	**137.94**$^{(onm)}$	3.59	**137.94**(5)	1.6	**137.94**(4)	1.6	**137.94**(5)	0.23	**137.94**(16)	3.57
tight-equal_1_10×2	434	453*	**461.96**	30	**461.96**(5)	1.6	**461.96**(4)	1.6	**461.96**(5)	0.64	**461.96**(18)	2.89
tight-equal_2_10×2	418	458	**448.32**$^{(onm)}$	6.87	**448.32**(5)	1.6	**448.32**(5)	1.6	**448.32**(5)	0.56	**448.32**(17)	2.95
tight-tard_1_10×2	174	195	**179.46**$^{(onm)}$	21	**179.46**(1)	1.6	**179.46**(5)	1.6	**179.46**(5)	0.33	**179.46**(18)	3.49
tight-tard_2_10×2	143	147	**145.37**	30	**145.37**(5)	1.6	**145.37**(2)	1.6	**145.37**(3)	0.75	**145.37**(20)	3.6
loose-equal_1_10×5	1263	1905	1738.05	30	1733.38(1)	1.6	1738.05(2)	1.6	**1719.63**(2)	0.86	**1719.63**(1)	43.3
loose-equal_2_10×5	878	1010	971.96	30	968.89(1)	1.6	**967.73**(2)	1.6	**967.73**(3)	0.83	**967.73**(11)	47.59
loose-tard_1_10×5	168	188	176.83	30	**175.08**(2)	1.6	**175.08**(1)	1.6	**175.08**(5)	0.75	**175.08**(1)	93.36
loose-tard_2_10×5	355	572	492.05	30	525.88(1)	1.6	499.93(1)	1.6	**485.06**(1)	0.87	**485.06**(1)	42.73
tight-equal_1_10×5	660	826	**689.11**	30	**689.11**(1)	1.6	**689.11**(5)	1.6	**689.11**(4)	0.69	**689.11**(1)	52.7
tight-equal_2_10×5	612	848	**763.24**	30	770.49(1)	1.6	764.03(1)	1.6	**763.24**(5)	0.78	764.03(1)	40.46
tight-tard_1_10×5	361	405	**379.19**	30	380.82(1)	1.6	380(1)	1.6	**379.19**(1)	0.98	**379.19**(5)	78.13
tight-tard_2_10×5	461	708	635.93	30	**627.45**(2)	1.6	628.68(1)	1.6	**627.45**(5)	0.87	**627.45**(7)	40.76
loose-equal_1_10×10	331	376	**360.74**	30	366.77(1)	1.6	365.81(1)	1.6	364.39(5)	0.87	375.9(1)	171.65
loose-equal_2_10×10	246	260	251.86	30	**249.85**(1)	1.6	252.99(1)	1.6	**249.85**(5)	1.16	256.13(1)	137.5
loose-tard_1_10×10	356	409	375.71	30	383.84(1)	1.6	380.26(1)	1.6	**374.2**(3)	0.92	386.51(1)	150.22
loose-tard_2_10×10	138	152	**144.94**	30	**144.94**(3)	1.6	144.99(1)	1.6	**144.94**(5)	0.88	**144.94**(2)	140.6
tight-equal_1_10×10	1126	1439	1281.66	30	**1276.23**(1)	1.6	1277.44(1)	1.6	**1276.23**(3)	1.14	1303.24(1)	137.45
tight-equal_2_10×10	1535	2006	1885.25	30	1885.25(1)	1.6	1871.23(1)	1.6	**1866.92**(3)	1.23	1939.47(1)	187.48
tight-tard_1_10×10	574	855	687.65	30	717.85(1)	1.6	746.03(1)	1.6	**668.14**(2)	1.12	712.2(1)	159.47
tight-tard_2_10×10	666	800	779.17	30	780.82(1)	1.6	785.79(1)	1.6	**777.85**(1)	1.02	779.17(1)	205.51
Укупан број инстанци за које је добијено оптимално/најбоље познато решење:			13		13		11		23		16	

Подебљане вредности представљају оптимална или најбоља позната решења.
Ознака * која се налази у горњем десном углу вредности UB указује да је горња граница решења нижа од оптималног/најбољег познатог решења.

Инстанце са 15 послова

У табели 4.5 дат је преглед резултата добијених тестирањем различитих метода предложених у литератури за 24 инстанце са 15 послова.

Имплементирана матхеуристика достигла је оптимална или најбоља решења у литератури код 8 (без доказа оптималности) од укупно 24 тест примера (33.3%). Поређењем са резултатима добијеним CPLEX решавачем, којим је то, такође постигнуто за 8 од 24 инстанце, метода изложена у овом раду дала је боља решења за 8, иста за 5, а лошија за 11 тест примера. Матхеуристика је за краће време рада, код инстанци са 2 машине, дошла до решења истог или бољег квалитета у односу на CPLEX. За оне тест примере код којих је број машина једнак 5, матхеуристика је дала боље резултате, али је и време извршавања програма дуже од максималних 30 минута додељених CPLEX-у. Код инстанци са 10 машина CPLEX је дао боље резултате за краће време рада. Хибридизација EA+LS+MP којој је време рада било ограничено на 5.13 минута, дала је оптимална или најбоља позната решења за 10 од укупно 24 инстанце (41.7%). У односу на њу, метода изложена у овом раду, за дуже време извршавања, добила је боље решење за 1 тест пример, исто за 8, а лошије за 15 инстанци. Поређењем са резултатима добијеним хибридизацијом VNS+MP којој је, такође време рада било ограничено на 5.13 минута, а која је достигла оптимална или најбоља позната решења за 5 од 24 инстанце (20.8%), имплементирана матхеуристика, за дуже време рада, дала је боља решења за 7, иста за 5, а лошија за 12 тест примера. Потом, анализирајући резултате добијене VNS методом, а која је достигла оптимална или најбоља позната решења за 18 од укупно 24 инстанце (75%), може се уочити да је за краће време рада дала иста решења као и матхеуристика за 6 тест примера, боља за 15, а лошија за 3 инстанце.

Посматрајући доње и горње границе може се уочити да се најбоља решења у литератури за свих 24 тест примера налазе у интервалу (LB, UB). Што се резултата добијених матхеуристиком тиче, 22 решења се налазе у одговарајућем опсегу, док су вредности добијене за инстанце „loose-equal_1_15×10” и „loose-tard_2_15×10” веће од вредности одговарајућих горњих граница решења.

Табела 4.5: Преглед свих добијених резултата за инстанце са 15 послова (n = 15)

Инстанце	*LB* [68]	*UB* [68]	*CPLEX* [9]		*EA + MP + LS* [88]		*VNS + MP* [98]		*VNS* [9]		Матхеуристика (VNS+CPLEX)	
			z	$time_{cplex}$	z_{best}	t_{tot}	z_{best}	t_{tot}	z_{best}	t_{tot}	z_{best}	t_{tot}
loose-equal_1_15×2	1032	1142	**1041.33**	30	**1041.33**(3)	5.13	**1041.33**(2)	5.13	**1041.33**(4)	0.95	**1041.33**(9)	12.44
loose-equal_2_15×2	490	520	**497.97**	30	**497.97**(2)	5.13	541.72(1)	5.13	512.54(1)	1.15	**497.97**(6)	13.33
loose-tard_1_15×2	650	730	**654.84**	30	**654.84**(5)	5.13	**654.84**(1)	5.13	**654.84**(5)	1.21	**654.84**(10)	13.72
loose-tard_2_15×2	278	310	293.17	30	**279.71**(2)	5.13	297.54(2)	5.13	285.16(1)	1.23	**279.71**(6)	13.52
tight-equal_1_15×2	3316	3559	3366.66	30	**3344.54**(2)	5.13	**3344.54**(1)	5.13	**3344.54**(5)	1	**3344.54**(8)	11.8
tight-equal_2_15×2	1449	1579	**1479.76**	30	**1479.76**(3)	5.13	**1479.76**(2)	5.13	**1479.76**(5)	1.04	**1479.76**(9)	11.55
tight-tard_1_15×2	786	913	806.92	30	**790.5**(2)	5.13	790.66(1)	5.13	**790.5**(3)	1.25	**790.5**(2)	14.85
tight-tard_2_15×2	886	956	**905.37**	30	**905.37**(5)	5.13	**905.37**(1)	5.13	**905.37**(5)	1.42	**905.37**(11)	14.71
loose-equal_1_15×5	2763	4408	3457.57	30	**3220.16**(1)	5.13	3321.54(1)	5.13	3272.46(1)	2.49	3256.73(1)	84.51
loose-equal_2_15×5	2818	4023	3437.32	30	3328.86(1)	5.13	3310.27(1)	5.13	**3260.81**(1)	2.24	3439.79(1)	114.98
loose-tard_1_15×5	1098	1723	1281.87	30	1320.93(1)	5.13	1315.47(1)	5.13	**1281**(1)	2.22	1357.32(1)	102.12
loose-tard_2_15×5	314	374	335.55	30	329.47(1)	5.13	335.55(1)	5.13	**328.26**(1)	2.78	340.5(1)	215.5
tight-equal_1_15×5	1052	1663	1412.71	30	1369.06(1)	5.13	1345.21(1)	5.13	**1342.3**(1)	1.85	1399.64(1)	100.46
tight-equal_2_15×5	1992	2989	2782.17	30	**2630.06**(1)	5.13	2709.13(1)	5.13	2641.28(2)	1.96	2707.59(1)	91.26
tight-tard_1_15×5	1014	1538	1371.29	30	1371.31(1)	5.13	1426.97(1)	5.13	**1359.18**(1)	2.28	1366.67(1)	101.04
tight-tard_2_15×5	626	843	721.46	30	691.1(1)	5.13	725.72(1)	5.13	**681.27**(1)	2.92	692.08(1)	151.69
loose-equal_1_15×10	758	1109	**875.75**	30	1008.28(1)	5.13	977.43(1)	5.13	906.49(1)	2.8	1196.55(1)	732.94
loose-equal_2_15×10	1242	2256	1587.09	30	1720.18(1)	5.13	1584.44(1)	5.13	**1487.33**(1)	2.35	2019.65(1)	639.24
loose-tard_1_15×10	258	312	**277**	30	297.19(1)	5.13	296.6(1)	5.13	**277**(2)	1.71	334.31(1)	671.68
loose-tard_2_15×10	476	855	**597.93**	30	713.97(1)	5.13	664.13(1)	5.13	601.24(1)	2.95	901.96(1)	309
tight-equal_1_15×10	4389	8381	7167.59	30	7380.76(1)	5.13	8120.43(1)	5.13	**6820.87**(1)	1.99	8222.82(1)	228.83
tight-equal_2_15×10	3539	7039	5407.42	30	4893.44(1)	5.13	4966.34(1)	5.13	**4760.48**(1)	2.7	5657.76(1)	387
tight-tard_1_15×10	649	972	815.03	30	800.87(1)	5.13	832.08(1)	5.13	**767.26**(1)	2.64	833.11(1)	1723.13
tight-tard_2_15×10	955	1656	1267.53	30	1310.04(1)	5.13	1452.9(1)	5.13	**1224.82**(1)	2.77	1521.14(1)	569.49
Укупан број инстанци за које је добијено оптимално/најбоље познато решење:			8		10		5		18		8	

Подебљане вредности представљају оптимална или најбоља позната решења.

Инстанце са 20 послова

У табели 4.6 дат је преглед резултата добијених тестирањем различитих метода предложених у литератури за 24 инстанце са 20 послова.

Имплементирана матхеуристика достигла је оптимална или најбоља решења у литератури код 5 (без доказа оптималности) од укупно 24 тест примера (20.8%). Поређењем са резултатима добијеним CPLEX решавачем који није достигао ниједно најбоље познато решење, метода изложена у овом раду је за дуже време извршавања програма дала боља решења за 18, а лошија за 6 тест примера. Хибридизација EA+LS+MP којој је време рада било ограничено на 10.05 минута, дала је оптимална или најбоља позната решења за 5 од укупно 24 инстанце. У односу на њу, матхеуристка изложена у овом раду је за дуже време извршавања добила боље решење за 3 тест примера, исто за 2, а лошије за 19 инстанци. Ограничено време рада на 10.05 минута, имала је и метода VNS+MP која није достигла најбоље познато решење ни за један од укупно 24 тест примера. У поређењу с њом, имплементирана матхеуристика је за дуже време рада дала боља решења за 10, а лошија за 14 инстанци. Потом, анализирајући резултате добијене VNS методом, а која је достигла најбоља позната решења за 18 од укупно 24 инстанце (75%), може се уочити да је за краће време рада дала иста решења као и матхеуристика за 3 тест примера, боља за 17, а лошија за 4 инстанце. Ова метода показала се најефикаснијом и најквалитетнијом од до сада имплементираних за решавање JIT-JSSP, а које су доступне у литератури.

Посматрајући доње и горње границе може се уочити да се најбоља позната решења за свих 24 тест примера налазе у интервалу (LB, UB). Што се резултата добијених матхеуристиком тиче, 22 решења се налазе у одговарајућем опсегу, док су вредности добијене за инстанце „loose-tard_1_20×10" и „loose-tard_2_20×10" веће од вредности одговарајућих горњих граница решења.

Табела 4.6: Преглед свих добијених резултата за инстанце са 20 послова (n = 20)

Инстанце	*LB* [68]	*UB* [68]	*CPLEX* [9]		*EA + MP + LS* [88]		*VNS + MP* [98]		*VNS* [9]		Матхеуристика (VNS+CPLEX)	
			z	$time_{cplex}$	z_{best}	t_{tot}	z_{best}	t_{tot}	z_{best}	t_{tot}	z_{best}	t_{tot}
loose-equal_1_20×2	2546	2708	2565.51	30	2548.95(1)	10.05	2551.22(1)	10.05	2548.95(1)	2.06	**2547.68**(1)	34.79
loose-equal_2_20×2	3013	3318	3105.34	30	**3069.13**(1)	10.05	3070.16(1)	10.05	**3069.13**(1)	1.99	**3069.13**(3)	39.51
loose-tard_1_20×2	1194	1271	1265.36	30	1206.97(5)	10.05	1205.59(1)	10.05	1206.4(1)	2.32	**1205.19**(1)	29.11
loose-tard_2_20×2	735	857	772.89	30	**769.35**(1)	10.05	790.12(1)	10.05	**769.35**(1)	2.27	**769.35**(1)	46
tight-equal_1_20×2	1901	2008	1940.06	30	1937.49(1)	10.05	1939.11(1)	10.05	1936.45(1)	2.13	**1936.24**(1)	37.36
tight-equal_2_20×2	912	1014	961.44	30	**941.69**(1)	10.05	959.65(1)	10.05	943.7(1)	2.11	943.7(2)	38.96
tight-tard_1_20×2	1515	1913	1675.37	30	**1671.87**(1)	10.05	1682.94(1)	10.05	1674.6(1)	3.14	1674(1)	43.17
tight-tard_2_20×2	1375	1594	1489.65	30	**1448.66**(1)	10.05	1459.57(1)	10.05	1451.61(1)	1.99	1452.97(1)	42.68
loose-equal_1_20×5	6697	9697	8271.34	30	7545.44(1)	10.05	7758.02(1)	10.05	**7524.13**(1)	5.16	8099.98(1)	167.64
loose-equal_2_20×5	6017	8152	7793.45	30	7252.81(1)	10.05	7302.01(1)	10.05	**7059.38**(1)	3.76	7500.54(1)	174.59
loose-tard_1_20×5	2524	3377	3548.14	30	2938.24(1)	10.05	3517.2(1)	10.05	**2931.11**(1)	4.43	3211.94(1)	180.01
loose-tard_2_20×5	3060	5014	4048.77	30	3722.61(1)	10.05	3684.67(1)	10.05	**3627.51**(1)	4.67	3775.83(1)	171.03
tight-equal_1_20×5	2506	3090	3013.79	30	2866.94(1)	10.05	2935.95(1)	10.05	**2865.95**(1)	3.37	2998.84(1)	300.08
tight-equal_2_20×5	5817	7537	7233.35	30	6741.16(1)	10.05	6964.91(1)	10.05	**6613.62**(1)	4.01	7013.3(1)	153.13
tight-tard_1_20×5	3244	4147	3821.29	30	3672.18(1)	10.05	3669.09(1)	10.05	**3620.61**(1)	6.15	3703.43(1)	170.6
tight-tard_2_20×5	1633	1916	1881.02	30	1858.19(1)	10.05	1976.11(1)	10.05	**1792.78**(1)	4.62	1861.73(1)	245.26
loose-equal_1_20×10	3539	6732	5597.4	30	5260.48(1)	10.05	5476.47(1)	10.05	**4651.31**(1)	5	6132.44(1)	996.01
loose-equal_2_20×10	1344	2516	1862.41	30	1987.65(1)	10.05	1718.08(1)	10.05	**1597.89**(1)	5.01	2425.21(1)	1000.53
loose-tard_1_20×10	2462	6237	6494.34	30	5093.89(1)	10.05	5478.21(1)	10.05	**4817.32**(1)	7.02	6915.78(1)	634.88
loose-tard_2_20×10	1226	1830	1545.95	30	1570.64(1)	10.05	1599.55(1)	10.05	**1401.28**(1)	6.63	1836.85(1)	8354.26
tight-equal_1_20×10	6708	12951	11561.7	30	10607.2(1)	10.05	11073.1(1)	10.05	**10378.8**(1)	5.02	12116.2(1)	649.05
tight-equal_2_20×10	5705	9435	8501.42	30	7310.32(1)	10.05	7317.74(1)	10.05	**6839.3**(1)	6.01	8074.47(1)	4150.27
tight-tard_1_20×10	3003	5968	4984.31	30	4853.81(1)	10.05	5536.64(1)	10.05	**4445.59**(1)	6.14	5800.38(1)	756.71
tight-tard_2_20×10	2740	3788	4057.01	30	3358.61(1)	10.05	3245.9(1)	10.05	**3085.54**(1)	5.82	3713.52(1)	8379.66
Укупан број инстанци за које је добијено оптимално/најбоље познато решење:			0		5		0		18		5	

Подебљане вредности представљају оптимална или најбоља познато решења.

Глава 5

Закључак

У овом раду решаван је проблем правовременог распоређивања послова по машинама у вишефазној производњи са укљученим казнама за превремени завршетак и кашњење за сваку операцију. Као улазни подаци за овај проблем дати су скуп послова и скуп машина. Сваки посао састоји се од низа операција које је потребно извршити редом, на унапред изабраним машинима. Потом, за сваку операцију задато је време трајања, идеално време завршетка, као и казнени коефицијенти уколико се она заврши пре или после предвиђеног времена. Задатак се огледа у проналажењу редоследа извршавања свих операција за све послове тако да укупна казна буде минимална. Овај проблем спада у групу НП-тешких, а у пракси има значајну примену, нарочито у производим делатностима, где је ради избегавања додатних трошкова потребно оптимизовати процес производње, тако што ће се одабрати најбољи редослед извршавања производних корака на машинама предвиђеним за њихову реализацију.

Проблем је решаван матхеуристиком добијеном хибридизацијом методе промењљивих околина и егзактног CPLEX решавача. Идеја предложеног алгоритма је да се VNS методом бира редослед извршавања свих операција за све послове, а да се CPLEX-у повери решавање потпроблема у ком је потребно да се за тако задати редослед одредити време завршетка за сваку операцију појединачно. Матхеуристика је тестиран на 72 инстанце, а резултати су упоређени са постојећим резултатима из литературе. Код инстанци са 2 машине, углавном су добијена најбоља позната решења, а чак за 3 тест примера добијено је боље решење од најбољег у литератури. За инстанце код којих је број машина једнак 5, најбоље познато решење достигнуто је за 7 тест примера. Док је код инстанци са 10 машина, само за један тест пример достигнуто најбоље решење

познато у литератури.

Један од већих недостатака ове методе јесте то што инстанце са већим бројем операција захтевају дуго време извршавања. Највећи део времена утроши се на извршавање фазе локалне претраге, која је иначе у већини VNS алгоритама временски најзахтевнија компонента. Како је у алгоритму предложене матхеуристике фаза локалне претраге имплементирана на сложенији начин који изискује велики број рачунања, тако је и време потребно да се она изврши дуго. Унапређивање квалитета ове матхеуристике могуће је извести следећим променама:

- имплементирати бољи начин генерисања почетног решења којим би се на што прецизнији начин распоредиле све операције на одговарајућим машинама, обзиром да је у овом раду дат макималан степен слободе овом кораку, што се није показало превише ефикасним;
- упростити фазу локалне претраге што је могуће више;
- то што ће се упростити фаза локалне претраге, треба надоместити имплементацији фазе размрдавања, како би се на једноставнији начин претраживале што квалитетније околине.

Библиографија

[1] R. P. Araujo, A. G. dos Santos and J. E. C. Arroyo. Genetic Algorithm and Local Search for Just-in-Time Job–Shop Scheduling. In *IEEE Congress on Evolutionary Computation*, pages 18–21. IEEE, 2009.

[2] S. Talukdar, L. Baeretzen, A. Gove and P. de Souza. Asynchronous teams: Cooperation schemes for autonomous agents. *Journal of Heuristics*, 4:295–321, 1998.

[3] M. A. Boschetti, A. Letchford and V. Maniezzo. Matheuristics: survey and synthesis. *International Transactions in Operational Research*, 30(6):2840–2866, 2023.

[4] M. A. Boschetti, A. Mingozzi and S. Ricciardelli. An Exact Algorithm for the Simplified Multiple Depot Crew Scheduling Problem. *Annals of Operations Research*, 127:177–201, 2004.

[5] S. Boyd, L. Xiao, A. Mutapcic and J. Mattingley. Notes on Decomposition Methods. In *Notes for EE364B*. Stanford University, 2015.

[6] A. R. Clark and U. Aickelin. A hybrid genetic algorithm to solve a lot-sizing and scheduling problem. In *16th triannual Conference of the International Federation of Operational Research Societies*, 2002.

[7] E. Lawler, J. K. Lenstra, A. Rinnooy Kan and D. Shmoys. Sequencing and scheduling: algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[8] A. French, A. Robinson and J. Wilson. Using a hybrid genetic algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *Journal of Heuristics*, 7:551–564, 2001.

[9] M. M. Ahmadian, A. Salehipour and T. C. E. Cheng. A meta-heuristic to solve the just-in-time job-shop scheduling problem. *European Juornal of Operational Research*, 288:14–29, 2021.

[10] A. Wigderson. P, NP and mathematics - A computational complexity perspective. In *Proceedings of the International Congress of Mathematicians*, volume 1, pages 665–712, 2006.

[11] B. J. Lazović. *Primena metoda kombinatorne optimizacije za rešavanje problema formiranja grupa u nastavi.* PhD thesis, Matematički fakultet, Univerzitet u Beogradu, 2018.

[12] A. Adewumi, B. Sawyerr and M. Ali. A heuristic solution to the university timetabling problem. *Engineering Computations*, 26(8):972–984, 2009.

[13] H. Chen, C. Chu and J. M. Proth. An Improvement of the Lagrangean Relaxation Approach for Job Shop Scheduling: A Dynamic Programming Method. *IEEE Transactions on Robotics and Automation*, 14(5):786–795, 1998.

[14] C. Kaskavelis and M. Caramanis. Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems. *IIE Transactions*, 30:1085–1097, 1998.

[15] R. Congram, C. N. Potts and S. V. De Velde. An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem. *INFORMS Journal on Computing*, 14(1):52–67, 2002.

[16] D. Bertsekas. *Nonlinear programming, 2nd edition.* Athena Scientific, 2005.

[17] D. Bertsimas and J. Tsitsiklis. Simulated Annealing. *Statistical science*, 8(1):10–15, 1993.

[18] D. P. Bertsekas. *Network Optimization: Continuous and Discrete Models.* Athena Scientific, 1998.

[19] M. R. Garey, D. S. Johnson and R. Sethi. The complexity of flow shop and job shop scheduling. *Mathematical Operational Research*, 1(2):117–229, 1976.

[20] D. Woodruf. A chunking based selection strategy for integrating meta-heuristics with branch and bound. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 499–511, 1999.

[21] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorila Optimization*. Princeton University Press, 2003.

[22] M. Vanhoucke, E. Demeulemeester and W. Herroelen. An exact procedure for the resource-constrained weighted earliness-tardiness project scheduling problem. *Annals of Operations Research*, 102:179–196, 2001.

[23] E. G. Talbi. *Metaheuristics: From Design to Implementation*. Wiley, 2009.

[24] P. J. van Laarhoven, E. H. Aarts and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.

[25] E. Danna, E. Rothberg and C. Le Pape. Exploring relaxation induced neighbourhoods to improve MIP solutions. *Mathematical Programming*, 102:71–90, 2005.

[26] R. Bixby, M. Fenelon, Z. Gu, E. Rothberg and R. Wunderling. MIP: Theory and Practice - Closing the Gap. In *IFIP Conference on System Modeling and Optimization*, volume 46, pages 19–49, 2000.

[27] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision sciences*, 8(1):156–166, 1977.

[28] F. Glover. Tabu Search: A Tutorial. *Interfaces*, 20(4):74–94, 1990.

[29] F. Glover and G. A. Kochenberher, editors. *Handbook of metaheuristics*. Kluwer Academic Publishers, 2003.

[30] F. Kappel and A. V. Kuntsevich. An implementation of Schor's r-algorithm. *Computational Optimization and Applications*, 15(2):193–205, 2000.

[31] F. Sourd and S. Kedad-Sidhoum. The One-Machine Problem with Earliness and Tardiness Penalties. *Journal of Scheduling*, 6(6):533–549, 2003.

[32] R. Möhring, A. Schulz, F. Stork and M. Uetz. Solving Project Scheduling Problems by Minimum Cut Computations. *Management Science*, 49(3):330–350, 2003.

[33] G. B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, 1960.

[34] G. Di Caro. *Ant Colony Optimization and its Application to Adaptive Routing in Telecommunication Networks*. PhD thesis, Faculté des Sciences Appliquées, Université Libre de Bruxelles, 2004.

[35] G. Raidl. Decomposition based hybrid metaheuristics. *European Journal of Operational Research*, 244(1):66–76, 2015.

[36] G. Klau, I. Ljubić, A. Moser, P. Mutzel, P. Neuner, U. Pferschy, G. Raidl and R. Weiskircher. Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. In *Genetic and Evolutionary Computation*, volume 3102, pages 1304–1315, 2004.

[37] H. Chen and P. Luh. An alternative framework to Lagrangian relaxation approach for job shop scheduling. *European Journal of Operational Research*, 149(3):499–512, 2003.

[38] I. Dumitrescu and T. Stützle. Usage of Exact Algorithms to Enhance Stochastic Local Search Algorithms. In *Matheuristics*, pages 103–134, 1970.

[39] I. Dumitrescu and T. Stützle. Combinations of local search and exact algorithms. In *EvoWorkshops*, volume 2611, 2003.

[40] International Business Machines Corporation (IBM). IBM ILOG CPLEX Optimizer Deployment Entry Edition 22.1.0. Инсталација је доступна на линку: https://www.ibm.com/support/pages/downloading-ibm-ilog-cplex-optimizer-deployment-entry-edition-2210. (Преузето 24. 9. 2022. године).

[41] Z. Z. Lin, J. Bean and C. White. A Hybrid Genetic/Optimization Algorithm for Finite-Horizon, Partially Observed Markov Decision Processes. *INFORMS*, 16(1):27–38, 2004.

[42] J. C. Beck and P. Refalo. A Hybrid Approach to Scheduling with Earliness and Tardiness Costs. *Annals of Operations Research*, 118(1-4):49–71, 2003.

[43] J. Denzinger and T. Offermann. On cooperation between evolutionary algorithms and other search paradigms. In *Congress on Evolutionary Computation*, 1999.

[44] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[45] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* University of Michigan, 1975.

[46] M. Bazaraa, J. Jarvis and H. Sherali. *Linear programming and network flows, 2nd edition.* Wiley, 1990.

[47] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks.* IEEE, 1995.

[48] B. Guenin, J. Kónemann and L. Tunçel. *A Gentle Introduction to Optimization.* Cambridge University Press, 2014.

[49] R. Ahuja, Ö. Ergun, J. Orlin and A. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.

[50] J. Puchinger and G. Raidl. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 41–53, 2005.

[51] J. Puchinger and G. Raidl. Models and algorithms for three-stage two-dimensional bin packing. *European Journal of Operations Research*, 183(3):1304–1327, 2007.

[52] N. Kangarloo, J. Rezaeian and X. Khosrawi. JIT scheduling problem on flexible flow shop with machine break down, machine eligibility and setup times. *Journal of Mathematics and Computer Science*, 16(1):50–68, 2016.

[53] J. Tind and L. Wolsey. An elementary survey of general duality theory in Mathematical Programming. *Mathematical Programming*, 21:241–261, 1981.

[54] H. A. Yang, J. Y. Li and L. L. Qi. An Improved Genetic Algorithm For Just-In-Time Job-Shop Scheduling Problem. *Advanced materials research*, 472-475:2462–2467, 2012.

[55] K. Holmberg and J. Ling. A Lagrangean heuristic for the facility location problem with staircase costs. *European Journal of Operational Research*, 97(1):63–74, 1997.

[56] A. J. Kootanaee, K. N. Babu and H. F. Talari. Just-In-Time Manufacturing System: From Introduction to Implement. *International Journal of Economics, Business and Finance*, 1(2):7–25, 2013.

[57] M. A. Boschetti and V. Maniezzo. Benders decomposition, Lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15(3):283–312, 2009.

[58] M. A. Boschetti and V. Maniezzo. A set covering based matheuristic for a real-world city logistics problem. *International Transactions in Operational Research*, 22(1):169–195, 2015.

[59] M. A. Boschetti and V. Maniezzo. Matheuristics: using mathematics for heuristic design. *4OR*, 20:173–208, 2022.

[60] V. Maniezzo, M. A. Boschetti and T. Stützle. *Matheuristics: Algorithms and Implementations*. 2021.

[61] V. Vujčić, M. Ašić and N. Miličić. *Matematičko programiranje*. Matematički institut, Beograd, 1980.

[62] L. Jourdan, M. Basseur and E. G. Talbi. Hybridizing Exact methods and Metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3):620–629, 2009.

[63] R. de Freitas Rodrigues, M. C. Dourado and J. L. Szwarcfiter. Scheduling problem with multi-purpose parallel machines. *Discrete Applied Mathematics*, 164(1):313–319, 2014.

[64] A. Colorni, M. Dorigo and V. Maniezzo. Distributed Optimization by Ant Colonies. In *European Conference on Artificial Life*, 1991.

[65] M. E. Dyer and L. A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer problem. *Discrete Applied Mathematics*, 26(2-3):255–270, 1990.

[66] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98(1):23–47, 2003.

[67] M. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 50(12):1861–1871, 2004.

[68] P. Baptiste, M. Flamini and F. Sourd. Langrangian Bounds for Just-In-Time Job-Shop Scheduling. *Computers & Operations Research*, 35(3):906–915, 2008. Инстанце за JIT-JSSP доступне су на линку: https://drive.google.com/drive/folders/1fhpqCAT-WABCldP31DdG3ag75GdXJZCy?usp=sharing.

[69] M. G. C. Resende and C. C. Ribeiro. Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations and Applications. In *Handbook of Metaheuristics*, pages 283–319, 2010.

[70] F. Della Croce, M. Ghirardi and R. Tadei. Recovering Beam Search: Enhancing the Beam Search Approach for Combinatorial Optimization Problems. *Journal of Heuristics*, 10(1):89–104, 2004.

[71] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems, 3rd Edition.* Springer, 2008.

[72] D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz and M. Savelsbergh. Heuristic optimization: A hybrid AI/OR approach. *Computer Science*, 1997.

[73] M. R. Garey and D. S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[74] V. Maniezzo, M. A. Boschetti, M. Roffilli and A. B. Röhler. Matheuristics: Optimization, Simulation and Control. In *Hybrid Metaheuristics*, pages 171–177, 2009.

[75] A. Colorni, M. Dorigo, M. Tubian and V. Maniezzo. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

[76] N. Fateha, M. Y. Nafrizuan and Y. A. Razlan. Review on Elements of JIT Implementation. In *International Conference on Automotive, Mechanical and Materials Engineering*, 2012.

[77] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[78] P. Hansen, N. Mladenović and J. A. Moreno-Pérez. Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[79] O. Flippo and A. R. Kan. Decomposition in general mathematical programming. *Mathematical Programming*, 60:361–382, 1993.

[80] O. Goldreich. *P, NP, and NP-Completeness: The Basics of Complexity Theory*. Cambridge University Press, 2010.

[81] P. Brucker. *Scheduling Algorithms, 5th edition*. Springer, 2016.

[82] P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[83] P. Lučić and D. Teodorović. Bee system: Modeling Combinatorial optimization transportation engineering problems by swarm intelligence. *Preprints of the TRISTAN IV triennial symposium on transportation analysis*, pages 441–445, 2001.

[84] P. Lučić and D. Teodorović. Computing with Bees: Attacking Complex Transportation Engineering Problems. *International Journal of Artificial Intelligence Tools*, 12:375–394, 2003.

[85] P. Ow and T. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26(1):35–62, 1988.

[86] P. Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *International Conference on Global Trends in Signal Processing, Information Computing and Communication*. IEEE, 2016.

[87] R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell system technical journal*, 45(9):1563–1581, 1966.

[88] A. G. dos Santos, R. P. Araujo and J. E. C. Arroyo. A Combination of Evolutionary Algorithm, Mathematical Programming, and a New Local Search Procedure for the Just-In-Time Job-Shop Scheduling Problem. In

International Conference on Learning and Intelligent Optimization, pages 10–24, 2010.

[89] P. Hansen, N. Mladenović, R. Todosijević and S. Hanafi. Variable neighbourhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[90] S. Chetty and A. O. Adewumi. A Study on the Enhanced Best Performance Algorithm for the Just-in-Time Scheduling Problem. *Discrete Dynamics in Nature and Society*, 264:1–12, 2015.

[91] S. Gupta and J. Kyparisis. Single machine scheduling research. *Omega*, 15(3):207–227, 1987.

[92] S. Q. Liu and E. Kozan. Scheduling a flow shop with combined buffer conditions. *International Journal of Production Economics*, 117(2):371–380, 2009.

[93] S. S. Rao. *Engineering Optimization: Theory and Practice, 5th Edition*. Wiley, 2019.

[94] D. Cvetković, M. Čangalović, Đ. Dugošija, V. Kovačević-Vujčić, S. Simić and J. Vuleta. *Kombinatorna optimizacija*. Društvo operacionih istraživača Jugoslavije - DOPIS, 1996.

[95] S. Singh and D. Garg. JIT System: Concepts, Benefits and Motivation in Indian Industries. *International Journal of Management & Business Studies*, 1(1):26–30, 2011.

[96] D. P. Williamson, L. A. Hall, J. A. Hoogeveen, C. A. J. Hurkens, J. K. Lenstra, S. V. Sevastyanov and D. B. Shmoys. Short Shop Schedules. *Operations Research*, 45(2):288–294, 1997.

[97] M. Caserta, S. Voß and M. Siendovich. Applying the corridor method to a blocks relocation problem. *OR Spectrum*, 33:915–929, 2011.

[98] S. Wang and Y. Li. Variable Neighbourhood Search and Mathematical Programming for Just-in-Time Job-Shop Scheduling Problem. *Mathematical Problems in Engineering*, 2014.

[99] T. A. Feo and M. G. C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[100] T. Gonzalez and S. Sahni. Open Shop Scheduling to Minimize Finish Time. *Journal of the ACM*, 23(4):665–679, 1976.

[101] D. Applegate, R. Bixby, V. Chvátal and W. Cook. On the Solution of Traveling Salesman Problems. *Documenta Mathematica*, pages 645–656, 1998.

[102] V. Dugošija. *Linearno programiranje*. Zavod za udžbenike - Beograd, 2011.

[103] V. Jeyakumar and A. M. Rubinov, editors. *Continuous Optimization: Current Trends and Modern Applications*. Springer New York, 2005.

[104] M. A. Boschetti, V. Maniezzo and M. Roffilli. Decomposition techniques as metaheuristic frameworks. *Matheuristics, Annals of Information Systems*, 10:135–158, 2009.

[105] W. Xing and J. Zhang. Parallel machine scheduling with splitting jobs. *Discrete Applied Mathematics*, 103(1-3):259–269, 2000.

[106] X. S. Yang. *Optimization Techniques and Applications with Examples*. Wiley, 2018.

[107] J. N. Monette, Y. Deville and P. V. Hentenryck. Just-In-Time Scheduling with Constraint Programming. In *19th International Conference on Automated Planning and Scheduling*, volume 19, pages 19–23, 2009.

[108] M. Gen, Y. Tsujimura and E. Kubota. Solving job-shop scheduling problems by genetic algorithm. In *IEEE International Conference on Systems, Man and Cybernetics*. IEEE, 1994.

[109] Z. Dražić. *Modifikacija metode promenljivih okolina i njihove primene za rešavanje problema raspoređivanja prenosa datoteka*. PhD thesis, Matematički fakultet, Univerzitet u Beogradu, 2014.

[110] Z. Stanimirović. *Nelinearno programiranje*. Matematički fakultet, Univerzitet u Beogradu, 2014.

[111] Đ. Dugošija and A. Savić. *Operaciona istraživanja*. Matematiči fakultet, Univerzitet u Beogradu, 2018.

Биографија аутора

Анђела Р. Вучинић рођена је 01. јуна 1995. године у Косовској Митровици. Основну школу „Стана Бачанин" у Лешку завршила је као Вуковац и ђак генерације. Школовање је наставила у Средњој школи „Никола Тесла" у Лепосавићу на природно-математичком смеру гимназије, коју такође завршава као Вуковац и ђак генерације. Математички факултет у Београду уписала је 2014. године и дипломирала на студијском програму Математика, модул Примењена математика. Након завршетка основних студија, уписала је мастер студије на истом смеру на Математичком факултету. Од фебруара 2022. године, до марта 2023. године била је запослена у Републичком заводу за статистику. Тренутно ради у ИТ индустрији као софтверски инжењер.