



БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ
ИНСТИТУТ ПО ИНФОРМАЦИОННИ И КОМУНИКАЦИОННИ
ТЕХНОЛОГИИ

Метаевристични методи за решаване на задачи за разкрояване

Георги Евтимов Евтимов

Автореферат

НА ДИСЕРТАЦИЯ

за придобиване на образователната и научна степен “доктор”

професионално направление 4.6 “Информатика и компютърни науки”

Научен ръководител: проф. Стефка Фиданова

гр. София, 2021 г.



Съдържание

1	Увод	10
1.1	Актуалност на темата	12
1.2	Обзор на основните резултати в областта	14
1.3	Цели и задачи на дисертацията	15
1.4	Подход на изследването	15
2	Изчислителна геометрия: Основни дефиниции	18
2.1	Дефиниция на списък	18
2.2	Дефиниции на точка, сегмент и полигон	18
2.3	Ротация на точка	19
2.4	Перпендикуляр от точка към дадена права	20
2.5	Намиране на <i>box</i> на полигон.	21
2.6	Огледален образ на полигон.	21
2.7	Намиране на посока на полигон (<i>clock – wise, CW</i> или <i>anti – CW</i>)	22
2.8	Ъгъл между два вектора. Вътрешен ъгъл на полигон.	23
2.9	Пресичане на две прави	24
2.10	Точка в полигон	25
2.11	Добавяне на точки в линеен сегмент	26
2.12	Премахване на излишни точки от линеен сегмент	26
2.13	Пресичане на два полигона	26
2.14	Редуциране на върховете на полигон.	27
3	Задача за 1D разкрой.	28
3.1	Формулировка на задачата.	28
3.2	Намиране на цялостно решение за 1D разкрой.	30
3.3	Резултати при 1D разкрой. Примери.	34
4	Задача за 2D разкрой.	37
4.1	Формулировка на задачата.	37
4.2	Стратегия за избор на входящ полигон. Метаевристични методи.	40
4.3	Стратегия за избор на едно решение от валидни разположения.	44
4.4	Намиране на едно възможно разположение на планка в полигона за запълване.	45
4.5	Премахване на реалната фира при 2D разкря.	46
4.6	Резултати при 2D разкрой. Примери.	46
5	Заклучение	51
5.1	Списък на публикациите	52
5.2	Апробация на резултатите	53
5.3	Приноси	54
5.4	Декларация за оригиналност	55

5.5 Благодарности	56
Съдържание	

Списък на фигурите

1.1	Стоманена конструкция	11
1.2	Стоманена конструкция 1.	13
1.3	Стоманена конструкция 2.	13
1.4	Първоначален основен полигон (за запълване).	16
1.5	Примерни входящи полигони (размерите са съществено увеличени)	16
1.6	Основен полигон преди изрязването.	16
1.7	Основен полигон след изрязването.	16
2.1	<i>box</i> на полигон	21
2.2	Лице на трапец	22
3.1	Дефиниране на линеен разкрой.	29
3.2	Илюстрация на метода мравките.	30
3.3	Оценка.	30
3.4	Комерсиален продукт за $1D$ разкрой.	34
3.5	Решение на $1D$ разкрой с комерсиален продукт.	34
4.1	Класификация на метаевристики с популации.	42
4.2	Входящи полигони.	47
4.3	Комерсиален продукт FP Opti2D.	48
4.4	Лист от разпечатката на комерсиалния продукт.	49

Списък на таблиците

3.1	Входни данни за разкрой на профили.	30
3.2	Резултати с Мравки за $1D$ разкроя.	35
3.3	Резултати с <i>Greedy</i> + n^3 метод за $1D$ разкроя.	36
3.4	Сравнение на резултатите за $1D$ разкроя.	36
4.1	Сравнение на комерсиален продукт и представения алгоритъм	50

List of Algorithms

1	InsidePolyAngle	24
2	ACOAlgorithm	32

Списък на съкращенията

Съкращения Описание

1. CW Построение на полигон по посока на въртене на часовниковата стрелка
2. $anti - CW$ Построение на полигон по посока на обратно на въртене на часовниковата стрелка
3. A_1, A_2, \dots, A_n Име на връх от полигон A .
4. B_1, B_2, \dots, B_n Име на връх от полигон B .
5. pt_1, pt_2, \dots, pt_n Върхове от полигон. Всеки връх е вектор с координати $[X_n, Y_n]$
6. pt_m Средна точка между точки p_i и p_{i+1}
7. X_i X координата на точката pt_i
8. Y_i Y координата на точката pt_i
9. F_i Площ на i -тия полигон
10. $+F$ Положителна сума от площите на полигони
11. $-F$ Отрицателна сума от площите на полигони
12. α_i i -тия ъгъл между два вектора
13. $ptX_1, ptX_2, \dots, ptX_n$ Пресечни точки между два полигона
14. \vec{a} Вектор \vec{a} с координати $[X_i, Y_i]$
15. $list$ Списък с елементи
16. $1D$ Едномерен случай
17. $2D$ Двумерен случай
18. iff Тогава и само тогава
19. $fuzz$ Размитост на координатите. Максимално допустимо отклонение от дадена координата
20. $\&\&$ Логическо И

Дисертационният труд е обсъден и допуснат до защита на разширено заседание на секция "Научни пресмятания" на ИИКТ-БАН, състояло се на
Дисертацията съдържа 143 страници, 39 таблици, фигури 143, алгоритми 10, литературни източника 117. По дисертационния труд са публикувани 6 публикации. Защита на дисертацията ще се състои на от в зала 218 на блок 25А на ИИКТ-БАН на открито заседание на научно жури в състав:

1.;
2.;
3.;
4.

Материалите за защита са на разположение на интересуващите се в стая на ИИКТ-БАН, ул. "Акад. Г. Бончев бл. 25А

Автор Георги Евтимов Евтимов.

Заглавие : **Метаевристични методи за решаване на задачи за разкрояване**

Глава 1

Увод

Необходимостта от оптимизация на човешкия труд води до масовото разпространение на изчислителни електронни устройства, които в повечето случаи заместват човешкото присъствие. От там започва и екстремното развитие на информационните технологии (ИТ) като средство за управление на изчислителните устройства. Значителното поевтиняване на електрониката допълнително развива този процес. Все повече машини за производство се управляват от компютри, без значение дали се ползват от предприятие, което е малко, средно или голямо. Естествено продължение на този процес е развитието на мрежа за обединяване на електронните устройства наречена Интернет.

Съвременна тенденция е всички услуги да се пренасочват от реалността във виртуалната реалност. Високо технологичните индустрии в производствения сектор масово изграждат системи за планиране и управление на ресурсите на производството. Информационните системи позволяват оптимизиране на ресурсите на всички нива на организационната йерархия. Тази оптимизация в повечето случаи оказва положително влияние както върху конкурентноспособността на фирмата, така и способства за по-гъвкаво и по-бързо намиране на нови пазари. Използване на информационните системи от фирмите дава възможност на клиентите им да решават по-добре, по-бързо и по-ефективно специфични проблеми в дадена предметна област. Особено ефективен е този подход в сферата на тежката индустрия и строителството.

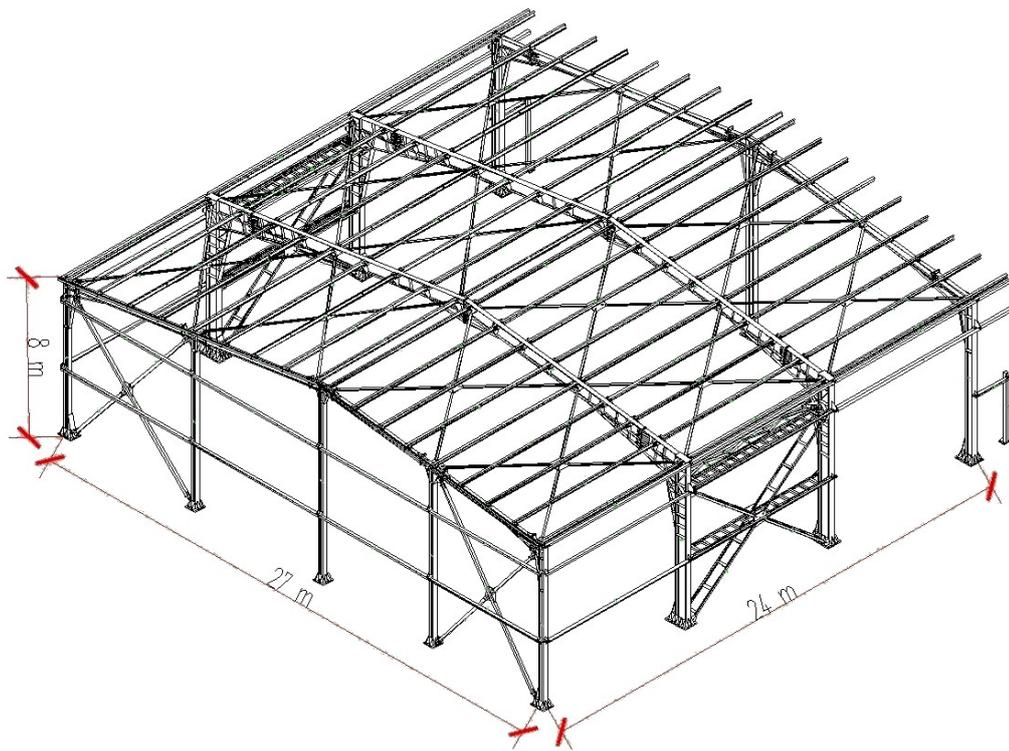
Широкото прилагане на ИТ в Европа и България създаде една динамична и високо конкурентна среда, в която фирма без внедряване на ИТ решения е често обречена на фалит. Необходимостта от съкращаване на производствени разходи е жизнено важна за оцеляването на предприятието. От друга страна изискванията на клиентите нараства, което допълнително подхранва необходимостта от бързи решения за оптимизация на материалните и човешки ресурси, което се постига ефективно с използване на софтуер. Тази ситуация на пазара открива нови и почти неограничени възможности за прилагането на приложен софтуер при решаването на най-различни задачи. Всеки софтуер, в който е приложено знание, може да се разглежда като стратегически източник на иновация. Друга гъвкавост на ИТ е възможността технологията да бъде разработена от трета страна, не принадлежаща на фирмата, но да се прилага ефективно от много други фирми.

Накратко, фокусът на научно-изследователската работа в съчетание с технологичните иновации е една от най-динамичните области на развитието на съвременната индустрия. Настоящата дисертацията представлява усилие в тази посока. Мотивацията и обектът на приложение на тази работа произлиза от строителната индустрия и по специално от производството на стоманени конструкции.

Една от най-важните и широко практикувани дейности там е следната: за нуж-

дите на строителен обект е необходимо да се разкроят определено число детайли (много често достигащ хиляди) с различни размери, форми, дебелини, а в някои случаи и от различен материал. Материалът е доставен под формата на метални листове или остатъци от листове, от които преди това са изрязвани детайли. Пример на такава стоманена конструкция е показан на Фигура 1.1. Необходимо е да се разкроят нужните детайли като се минимизира разхода на материал.

Тази постановка е частен случай на общата математическа задача за оптимален разкрой. Практическата задача за оптимален разкрой се заключава в следната лесна формулировка: зададен е определен материал (например, в текстилната индустрия това е плат, в строителните конструкции това са метални листове) и голямо количество често различни, детайли. Необходимо е да се разкроят нужните детайли като се минимизира разходът на материал. Практически, това означава да се минимизира материалът, който остава след разкроя и не може да се оползотвори освен да се предаде за вторична преработка или рециклиране. Този неизползуван материал се нарича често фира. Тази задача математически е формулирана преди повече от 80 години във връзка с индустриализацията на шивашкото производство. Подобен тип задачи възникват в много други индустриални производства и използването на оптимизация на решенията може да доведе до съществени икономии на материал.



Фигура 1.1: Стоманена конструкция

Предложените в тази работа научни методи ще бъдат полезни и най-подходящи за нуждите на стартиращи компании, включително и такива от сферата на софтуера. Методите, които са представени тук, са базирани на математика и логика и не се използват външни библиотеки, могат да се напишат, на които и да е език за програмиране и могат да помогнат развитието на която и да е компания.

1.1 Актуалност на темата

Темата за оптимален разкрой придобива още по-голяма актуалност през последните две десетилетия, намирайки разнообразни приложения в много индустриални производства. Особено важна е задачата сега, когато пазарът е отворен и фирмите трябва да се състезават с голям брой конкуренти с модерно оборудване и ниска цена на труда.

От друга страна масовото потребление на стоки води до необходимостта от оптимизиране на производството на тези стоки. Това включва както минимизиране на потреблението на енергия и суровини, така и намаляване на използване на човешки труд. Много остър е проблемът при тежките индустрии, особено когато е необходимо изработването на голям брой елементи от скъпо струващ материал. Тези две особености са налице, например, в строителната индустрия. Затова проблемът за оптималния разкрой там стои на дневен ред с особена актуалност.

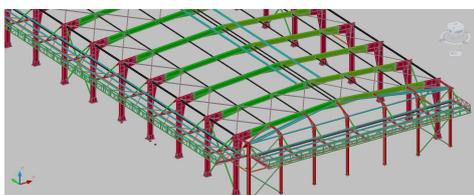
Обзор на съществуващите методи и тяхната реализация в приложен софтуер е направено по-долу в Секция 1.2. Тук ще отбележим само, че на пазара на софтуер за нуждите на оптималния разкрой има основно два вида подходи на реализация. В единия подход планките се апроксимират до правоъгълници, които се разполагат оптимално върху листа стомана, а при втория те се разполагат с точната им геометрия. При методите, които прилагат апроксимиране до правоъгълник недостатъкът е, че при фигури различни от правоъгълник отпадъкът е голям. В различните индустрии под голям може да се разбират най-различни числа. В настоящия труд 25-50% ще разбираме за голям отпадък, 10-25% за средно голям отпадък, 0-10% малък отпадък. При работа с триъгълни фигури образуван от прави по трите си страни отпадъка е 50%. Голям отпадък. Този подход има ограничено приложение, но се използва доста масово и дава добри резултати в стъklarската и хартиената промишленост. И в двата вида софтуер въвеждането на обектите за разкрой става ръчно. Полигоните се въвеждат по координати на върховете или по сегменти на страните. Това отнема много време и е възможно да се допуснат неточности и/или грешки при въвеждането на данните.

През последните три десетилетия за проектиране на строителни обекти се използват масово САД системи. В представената дисертация проблемът за оптимален разкрой на строителни елементи (или планки) е решен при предположението, че полигоните (планките) се генерират и предоставят на строителя от САД система. След това се прави предварителна обработка на данните и накрая планките се разкрояват с точната им геометрия.

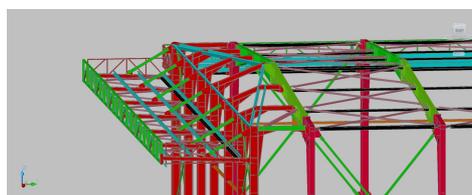
Заготовка на планки за нуждите на стоманените конструкции, представляват определен клас подзадачи за оптимален разкрой, който се характеризира с редица особености, които водят както до опростяване така и до усложняване на задачата за разкрой. Най-важните особености са:

1. често планките са със сложни форми, чиито граници са произволни несамопресичащи се полигони (в редки случаи, направата на елементи с елипсовидни контури се свежда към горния случай чрез апроксимация с полигони с достатъчна за практиката точност);
2. много често планката няма "лице" и "гръб", което позволява огледално търсене на местоположението ѝ в процеса на разкрой; тази особеност може да доведе до икономия на материал, но увеличава сложността на проблема.
3. в набора от планки, които трябва да бъдат произведени, много често има значително разнообразие на размерите, площите и формата.

В работата ударението е поставено върху методи за решаване на задачата за разкрой на планки в $2D$. Двумерната задача за разкрой е по-трудна от едномерната, особено когато изрязваните фигури не са изпъкнали и са с неправилна форма. И двете задачи са NP -пълни комбинаторни оптимизационни задачи [12], [13].



Фигура 1.2: Стоманена конструкция 1.



Фигура 1.3: Стоманена конструкция 2.

Като илюстрация ще се представят някои примери за случая на стоманени конструкции. В стоманено хале показано на фигура 1.2 може да има около 2000 - 3000 стоманени планки за разкрой. Те са с различна дебелина, в практиката често се налага да работим с 6 различни дебелини. Следователно от метален лист с дадена дебелина трябва да бъдат разкроени около 500 броя планки с доста сложна форма. В разглеждания случай ротация и огледален образ на планката са препоръчителни при оптимизацията на разкроя. Нещо повече, има планки, при които съотношението дължина към ширина (това се разбира като планката се постави в правоъгълник с минимални размери) е повече от 100.

1.2 Обзор на основните резултати в областта

Проблемът с оптималния разкрой (CSP) възниква в много индустриални области [61]. Повечето автори решават 2D разкрой, както апроксимират входящите полигони (фигурите) до правоъгълници. Тези решения също са приложими в много индустрии. Например производството на хартия и стъкло [24], зареждане на контейнери, дизайн с много мащабна интеграция (VLSI) и различни задачи за планиране [55].

По-сложната версия на проблема е, когато входящите полигони (фигурите) не са апроксимирани до правоъгълници. Този проблем възниква при строителните конструкции в производството на стоманени изделия, производството на дрехи, производството на обувки и т.н. В [24] основната тема е двуизмерен ортогонален проблем с опаковането, при който фиксирана група от малки правоъгълници трябва да се монтира в голям правоъгълник и неизползваната площ от големи правоъгълници да се сведе до минимум. Алгоритъмът комбинира метод на заместване с генетичен алгоритъм. В [45] е разработена *Greedy* (алчна) процедура на произволно (рандомизирано) адаптивно търсене. В това проучване има голям първичен запас, който трябва да бъде нарязан на по-малки парчета, за да се увеличи максимално стойността на парчетата. Cintra [49] предлага точен алгоритъм, базиран на динамично програмиране, който е подходящ за малки проблеми, тъй като проблемът е NP-труден. Dusberger и Raidl [62], [63] предлагат два мета-евристични алгоритъма, базирани на търсене на променливи квартали. Гореспоменатите работи решават опростения проблем с правоъгълни елементи. В индустрията на строителните конструкции планките са полигони, които могат да имат неправилна форма и могат да бъдат изпъкнали или вдлъбнати, но не и само пресичащи се. Подобно разнообразие от форми значително увеличава трудността на проблема. Като планките или входящите полигони могат да прилагат и огледално, тъй като стоманените листа са хомогенни от двете си страни. Също сложността на задачата се увеличава и от това, че триъгълна планка може да бъде описана с повече от три точки.

1.3 Цели и задачи на дисертацията

Основните цели поставени пред докторанта са от научно-приложен и приложен характер. Те могат да бъдат обобщени по следния начин.

Цели на дисертацията:

1. Оптимален разкрой на линейни елементи при минимален отпадък;
2. Оптимален разкрой на двумерни елементи с неправилна форма при минимален отпадък.

За постигане на тези цели бяха формулирани следните задачи:

Задача 1. Разработване на алгоритъм за решаване на задачата за едномертен (линеен) разкрой;

Задача 2. Разработване на алгоритъм за решаване на задачата за разкрой на двумерни елементи;

Задача 3. Да се направи програмна реализация на разработените алгоритми и да бъдат проведени сравнения на реални строителни обекти със съществуващи в практиката методи за разкрой.

База на разработката е *CAD* среда за получаване на графична информация от даден строителен обект. След оптимизацията се генерира информация в термините на същата *CAD* среда. За целта е разработен числен алгоритъм за разкрояване (разполагане) на произволни не самопресичащи се полигони (наричани планки и генерирани от *CAD* система) от зададен от потребителя полигон (стоманен лист). Фокус на дисертацията са равнинни елементи (листов материал) *2D* фигури (наричани тук планки). Геометрически това означава определен брой фигури в равнината да се подреди в площ със зададен от потребителя затворен контур. Това е доста обща математическа задача, която може да се приложи в най-различни индустрии. Алгоритъмът позволява и допълнителни настройки и различни принципи в оптимизацията при подреждане на фигурите. Тези задачи са базирани изцяло на примери от практиката, а входните данни са от реално проектирани и изпълнени строителни обекти.

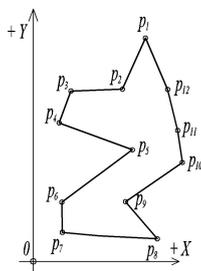
1.4 Подход на изследването

Тази дисертация се занимава с решаването на две оптимизационни задачи: (1) разкрой на линейни профили (стоманени пръти, *T*- и *П*-образни профили, и т.н.) или *1D* разкрой и (2) разкрой на двумерни (плоски) планки от стоманени листове.

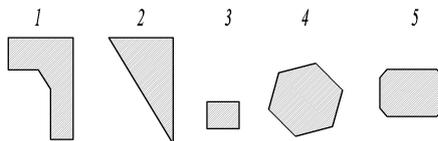
Първата задача е едномерен (линеен), *1D разкрой*. За *1D* оптимизацията не се въвеждат никакви специални дефиниции, тъй като се работи с един параметър, дължината на елемента. Задачата за минимален отпадък се свежда до намиране на минимален брой използвани профили. Въпреки, че е по-лесна от двумерната задача, тя също е *NP* сложна. Подходът използва метода на мравките.

Втората задача е *2D разкрой*. Данните включват даден входящ списък от n на брой планки (наречени входящи полигони), които трябва да се подредят възможно най-плътно в даден полигон, наречем основен. При търсенето на едно възможно разполагане на входящите полигони може да се приложи ротация и огледален образ. След като е избрано местоположение на входящия полигон е необходимо

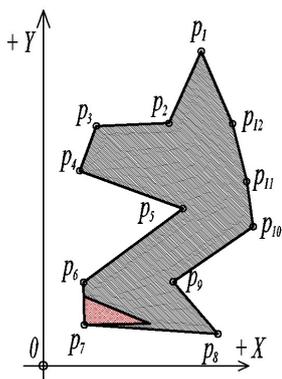
да се приложи алгоритъма за "изваждане" ("изрязване") на два полигона. Това се прави с цел за следващия входящ полигон да се търси местоположение в остатъка от основния полигон. След това тази (изрязана) планка се премахва от списъка с входящи планки. Това се повтаря докато се изчерпят всички планки от входящия списък. След това цялостното решение (съвкупността от планки) се оценява от метаевристиката. Виж точка 4.2.



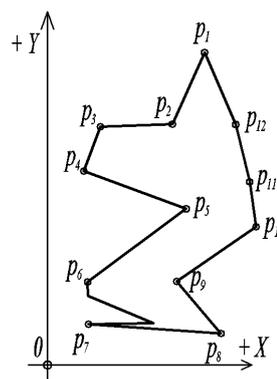
Фигура 1.4:
Първоначален основен полигон (за запълване).



Фигура 1.5: Примерни входящи полигони (размерите са съществено увеличени)



Фигура 1.6: Основен полигон преди изрязването.



Фигура 1.7: Основен полигон след изрязването.

В процеса на изучаване на проблема е използвана широк кръг от литература. Създадените нови методи и алгоритми са публикувани в статии на автора, [12, 13, 15, 14]. Създадени са три метода. Първия е за оценка на входящите полигони (множества). Спрямо ъглите им и дължините им. Този метод дава оценката от 0. до 1. за най-голямата вероятност даден полигон да се постави в полигона на запълване при даден връх. Втория метод дава оценка за най-голяма допирна дължина (площ) между два полигона. Третия метод е хибридната метаевристика. Дава оценка на всички допустими решения за даден връх. Оценката за всяко едно решение е между 0. и 1. Избира се това с най-голяма оценка. При положени, че има повече от едно решение с максимална оценка се избира произволно едно от тях. Комбинацията е от трите метода ни дава възможност да не търсим пълно изчерпване на възможните комбинации за поставяне на входящите полигони Π_i в полигона на запълване Δ . Разработените са два нови алгоритъма, които са подобрение на два съществуващи алгоритъма. Единия е *Ray* метода [40]. Добавката е, че преди да се приложи *Ray* метода се проверява дали дадената точка е в *box* на полигона, ако е така тогава се проверява за целия полигон. За дефиниция на *box*

на полигона виж точка 2.5. *box* на полигона ще се използва, ако броя на върховете на дадения полигон е по-голям от 4. Другият метод е "Bentley-Ottman" [39]. Добавката е, че не се обхождат всички сегменти, а алгоритъма спира при първото пресичане на двата сегмента.

Като краен резултат от изследването на проблема е разработен софтуер, който успешно решава двете задачи. Направено е сравнение на получените резултати с резултатите от използване на комерсиален софтуер. Предимствата на създадената програма пред тествания комерсиален софтуер са дадени в 4.6 и в 5.

Глава 2

Изчислителна геометрия: Основни дефиниции

В тази секция се използват геометричните обекти точка, линеен сегмент и полигон в двумерната равнина. Всички точки ще бъдат представени като списък от наредени числа (координати), [9], в двумерния случай това са двойки числа $P = (x, y)$. Тук също обсъждаме важните за построяването на алгоритми понятия кога една точка е вътре в даден полигон, пресичане и изваждане на два полигона, и т.н.

2.1 Дефиниция на списък

Списъка представлява строго подредени елементи. Всеки един елемент може да бъде число, стринг или друг списък. Примери:

1. $list(X, Y)$ - точка зададена с нейните декартови координати;
2. $list(pt_0, pt_1, \dots, pt_i)$ - списък от точки, където pt_i е списък, представящ точка с индекс i ;
3. $list(e_0, e_1, \dots, e_{n-1})$ - списък от сегменти, където e_i е линеен сегмент с индекс i , виж по-долу.

2.2 Дефиниции на точка, сегмент и полигон

Дефиниция на точка.

Точките в d -мерното пространство са представени като нареден списък от d числа, наречени координати, [9]. Тъй като разглеждаме задачата в равнина, то в тази работа точка pt_i е дефинирана като $pt_i = list(x_i, y_i)$, където координатите x_i и y_i са реални числа. При работа с реални числа използвайки компютърна аритметика се поставя въпросът за грешката при закръгление. Грешката от закръгление на реалните числа е важна и обширна област в математиката. Тук са приети следните правила:

1. Работим с точност четири знака след десетичната запетая .0001;
2. Приемаме отклонение $fuzz$, което е реално число, по-голямо от 0.

Тези правила са продиктувани от необходимостта да работим с данни произведени от САД системи. В областта на проектиране на обекти изпълнени със стоманени конструкции, размерите на конструкциите се дават в милиметри, така че всички числени данни (координати на точки, сегменти и т.н.) са в милиметри. За нуждите на строителната индустрия (стоманени конструкции), разликата в дължините на страните на планките от 0.5мм на дава детайла ги прави неразличими. Затова ги приемаме, че са еднакви.

Приемаме, че две точки съвпадат $P_i \equiv P_q$, ако:

$$(x_i - fuzz) \leq x_q \leq (x_i + fuzz) \wedge (y_i - fuzz) \leq y_q \leq (y_i + fuzz) \quad (2.1)$$

или

$$\max\{|x_i - x_q|, |y_i - y_q|\} \leq fuzz \quad (2.2)$$

Дефиниция на линеен сегмент.

Ще използваме два вида сегменти:

- (1) САД линеен сегмент $CADe_i = list(pt_0, pt_1, \dots, pt_i)$ се задава със списък от точки, които лежат на една права; такива сегменти се получават от работата на САД системата, която генерира всички входни данни използвани в тази работа.
- (2) Линеен сегмент $e_i = (pt_i, pt_{i+1})$ е затворено множество от точки лежащи на една права между две точки pt_i и pt_{i+1} , наречени крайни точки, [9]. Точките в списъка e_i са подредени. Като първата е начална, а втората крайна. В нашата работа линейните сегменти се получават след премахване на вътрешните точки на САД сегмента.

Дефиницията на полигон.

Полигон $\Pi = list(e_0, e_1, \dots, e_{n-1})$ е затворена област от равнината оградена от n линейни сегменти образуващи затворена крива, [9]. Обръщаме внимание, че тук използваме линеен сегмент, а не САД линеен сегмент. Нека pt_0, pt_1, \dots, pt_n са n точки от дадена равнина, такива, че $pt_0 = pt_n$. Точките pt_0, pt_1, \dots, pt_n образуват цикличен списък. Докато pt_0 е последвана от pt_1 , то pt_{n-1} е последвана от $pt_0 = pt_n$. Полигонът се описва също и от неговите върхове, крайните точки на сегментите му, така че еквивалентно, $\Pi = list(pt_0, pt_1, \dots, pt_n)$. Казваме, че два сегмента са съседни когато имат само една обща крайна точка.

Линейни сегменти образуват полигон тогава и само тогава, когато:

1. Пресечната точка между всяка двойка съседни сегменти в цикличния списък, е : $e_i \cap e_{i+1} = pt_{i+1}$, за всички $i = 0, \dots, n - 1$;
2. Несъседни сегменти не се пресичат.

Точките pt_i ще ги наричаме *върхове* на полигона, а сегмент от полигона ще наричаме линеен сегмент. Нека отбележим, че полигон с n върхове има n сегмента.

2.3 Ротация на точка

Да разгледаме две различни точки $pt_A = list(x_a, y_a)$ и $pt_{base} = list(x_{base}, y_{base})$ в координатна система XOY . Желаяем да завъртим точката pt_A около базовата точка pt_{base} на даден ъгъл β . Ако ъгълът β е положително число тогава въртенето се разбира обратно на часовниковата стрелка *anti - CW*, в противен случай въртенето е по часовниковата стрелка *CW*. След завъртането ще получим нова точка $pt_B = list(x_b, y_b)$ в координатната система XOY .

За да направим нужните пресмятания и получим изчислителните формули за координатите на точката получена след ротацията, ще въведем нова координатна система $X'O'Y'$, чието координатно начало съвпада с pt_{base} . Осите на новата координатна система $X'O'Y'$ се транслират успоредно на осите на координатната система XOY . Спрямо новата координатна система получаваме координатите на точка pt_A , $x'_a = (x_a - x_{base})$ и $y'_a = (y_a - y_{base})$ или $pt_A = list(x'_a, y'_a)$ в новата координатна система $X'O'Y'$. Радиуса на завъртане R се намира по формулата:

$$R = \sqrt{x'_a{}^2 + y'_a{}^2} \quad (2.3)$$

а ъгълът на завъртане α се получава от формулата:

$$\alpha = \arccos \frac{x'_a}{R} \quad (2.4)$$

Тогава координатите на точка pt_B в координатната система XOY са:

$$x_b = x_{base} + \frac{R}{\cos(\alpha + \beta)} \quad (2.5)$$

$$y_b = y_{base} + \frac{R}{\sin(\alpha + \beta)} \quad (2.6)$$

2.4 Перпендикуляр от точка към дадена права

Правата е зададена с две точки $A = list(x_a, y_a)$ $B = list(x_b, y_b)$ и тестовата точка е $T = list(x_t, y_t)$. Искаме да намерим точка $C = list(x_c, y_c)$ от правата $list(A, B)$ такава, че векторът определен от точките T и C е перпендикулярен на правата.

Преди да започнем търсенето на точката C , трябва да проверим дали точките $A = list(x_A, y_A)$, $B = list(x_B, y_B)$ и $T = list(x_t, y_t)$ не лежат на една права. Това става чрез намиране на лицето на триъгълника $F = list(A, B, T)$. Резултатът, които ще получим от този алгоритъм, е ориентираното лице на триъгълника $list(A, B, T)$. Нас ни интересува само дали лицето F е нула или не. Ако лицето $F = 0$, то точките лежат на една права и не е нужно да търсим перпендикулярния вектор \vec{TC} към правата $list(A, B)$. Ако лицето $F \neq 0$, то алгоритъма протича в следните стъпки, съгласно [41]:

1. Ако $x_A = x_B$, тогава правата е вертикална и търсената точка е $pt_C = list(x_A, y_T)$;
2. Ако $y_A = y_B$, тогава правата е хоризонтална и търсената точка е $pt_C = list(x_T, y_A)$;
3. Ако не са изпълнени горните условия, тогава търсим наклона m на правата $list(A, B)$:

$$m = \frac{y_B - y_A}{x_B - x_A} \quad (2.7)$$

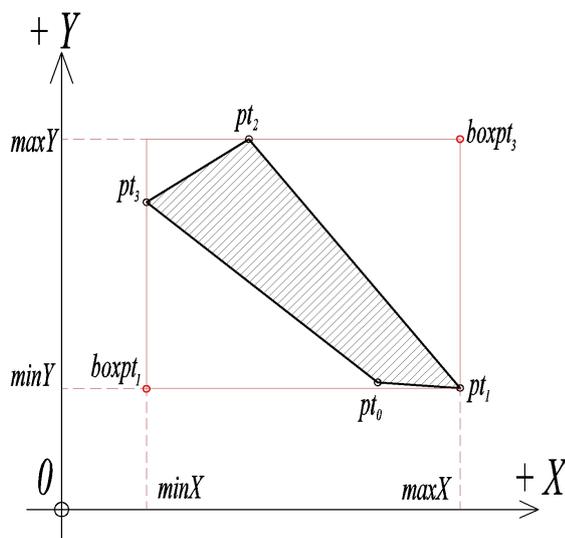
$$x_C = \frac{\left(\frac{x_T}{m} + y_T + m \cdot x_A - y_A\right)}{m + \frac{1}{m}} \quad (2.8)$$

$$y_C = y_A + m(x_C - x_A) \quad (2.9)$$

Или координатите на желаната точка $C = list(x_C, y_C)$.

2.5 Намиране на *box* на полигон.

Под *box* на полигон $\Pi_i = list(pt_0, pt_1 \dots pt_n)$ ще разбираме правоъгълната обвивка на дадения полигон Π_i . Виж фигура 2.1.



Фигура 2.1: *box* на полигон

За да намерим *box* на полигона Π_i трябва да преминаем през списъка от точки $\Pi_i = list(pt_0, pt_1 \dots pt_n)$ и за всяка една точка да вземем координатите ѝ по $X.pt$ и по $Y.pt$. След това да сортираме двата нови списъка по низходящ ред, $listX = (x_0, x_1 \dots x_n)$ и $listY = (y_0, y_1 \dots y_n)$. Първата стойност от $listX$ ще ни даде $maxX$ последната $minX$. Същото правим и за списъка $listY$. Така получаваме координатите на точките $boxpt_1 = list(minX, minY)$ и $boxpt_3 = list(maxX, maxY)$. Точка $boxpt_1$ е винаги най-долу, най-вляво. Точка $boxpt_3$ е винаги най-горе, най-вдясно.

2.6 Огледален образ на полигон.

Под огледален образ на полигон ще разбираме огледален образ по всички страни от полигона, които не са успоредни една на друга.

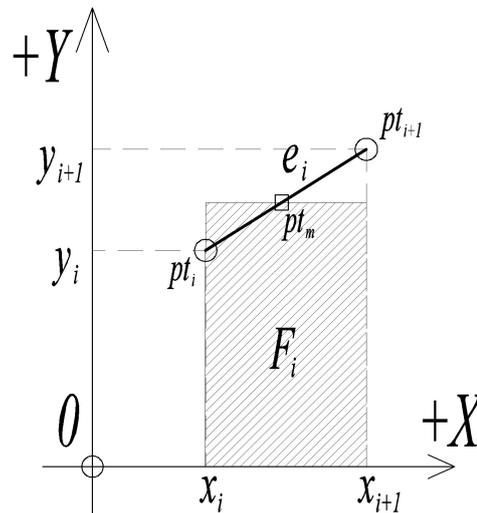
Полигона $Mirror1 = list(mpt_0, mpt_1, mpt_2, mpt_3)$ е получен от защрихования полигон $\Pi_i = list(pt_0, pt_1, pt_2, pt_3)$. За намирането на огледалния образ на полигона Π_i се използва следната последователност:

1. Взимаме първата pt_0 и втора pt_1 точка от полигона Π_i .
2. Образуваме правата $L1 - L1$. Правата $L1 - L1$ се образува по две точки. Първата точка е $pt_{L1} = (polar(pt_0; (angle = pt_1, pt_0); 10e10))$. Втората точка е $pt_{L2} = (polar(pt_1; (angle = pt_0, pt_1)); 10e10)$. Намираме полярните координати на точките pt_{L1} и pt_{L2} - базова точка, ъгъл и дължина. В случая дължината е избрана достатъчно голяма така, че да бъде допустима за *CAD* системата.
3. За всеки връх на полигона Π_i намираме петата на перпендикуляра към правата $L1 - L1$ и я означаваме с pt_{Perp_i} . Огледалната точка се получава: $mpt_i = (polar(pt_{Perp_i}; (angle = pt_i, pt_{Perp_i}); distance(pt_i, pt_{Perp_i}))$. За намиране на петата на перпендикуляра върху правата $L1 - L1$ виж Подсекция 2.4.

Правата $L1 - L1$ ще бъде колинеарна със сегмента $list(pt_0, pt_1)$. Тогава и разстоянието $distance(pt_0, pt_{Perp_i})$ ще бъде нула и точките pt_0 и pt_i ще съвпадат. За намиране на правата $L1 - L1$ може да се използва всяка една двойка последователни върхове $list = (pt_i, pt_{i+1})$ на полигона Π_i .

2.7 Намиране на посока на полигон (*clock-wise, CW* или *anti - CW*)

Тук ще обсъдим начини за определяне за посоката, (по часовата стрелка, *CW*, или обратна на часовата стрелка, *anti - CW*), на границата на полигон $\Pi = list(pt_0, pt_1, \dots, pt_n)$. От основите известни в литературата методи за ориентация на полигон, тук ще представим един от най-бързите методи за пресмятане на посоката на обхождане на върховете по границата на полигона Π [28].



Фигура 2.2: Лице на трапец

Нека $e_i \in \Pi$ е произволен сегмент и нека средната му точка P_m има координати:

$$P_m = \left(\frac{x_i + x_{i+1}}{2}, \frac{y_i + y_{i+1}}{2} \right). \quad (2.10)$$

Площта на фигурата между сегмент $e_i \in \Pi$ и координатна ос X е:

$$F_i = \frac{(x_{i+1} - x_i)(y_{i+1} + y_i)}{2} \quad (2.11)$$

Да отбележим, че лицето F_i може да бъде положително, отрицателно или нула. Знакът на лицето зависи от подредбата на точките в списъка определящи Π , тъй като подредбата може да бъде $list(pt_0, pt_1, \dots, pt_{n-1})$ или $list(pt_{n-1}, pt_0, \dots, pt_{n-1})$. Това лице ще наречем ориентирано лице. Тази процедура се прилага за всички e_i сегменти. За да се пести процесорно време няма смисъл всяко лице да се дели на

2. Затова сборът от ориентираните лица можем да запишем както следва:

$$2F = \sum_{i=0}^{n-1} (x_{i+1} - x_i)(y_{i+1} + y_i) \quad (2.12)$$

Ако координатите на точките на върхове $list(pt_0, pt_1, \dots, pt_5)$ удовлетворяват условията $x_5 > x_4 > x_3 > x_2 > x_1 > x_0$, то съответните площи са положителни и $F > 0$. Ако координатите на точките на върхове $list(pt_0, pt_1, \dots, pt_5)$ удовлетворяват условията $x_5 > x_6 > x_7 > x_8 > x_9$, тогава $F < 0$

2.8 Ъгъл между два вектора. Вътрешен ъгъл на полигон.

С цел да получим по-добра характеристика за даден полигон ще са ни нужни вътрешните му ъгли. Първо ще намерим ъгъла между два вектора $\vec{a} = list(T, pt_i)$ и $\vec{b} = list(T, pt_{i+1})$, дефинирани в XU координатна система.

Първо определяме дължината на векторите \vec{a} и \vec{b} , а след това и тяхното скалярно произведение. За да намерим дължината на вектора \vec{a} , ще транслираме точка A_i до нулата, $T_i = (list0, 0)$. Тогава векторите \vec{a} и \vec{b} ще имат координати съответно (x_a, y_a) и (x_b, y_b) .

$$\|\vec{a}\| = \sqrt{x_a^2 + y_a^2} \quad (2.13)$$

$$\|\vec{b}\| = \sqrt{x_b^2 + y_b^2} \quad (2.14)$$

скалярното произведение е:

$$\vec{a} \cdot \vec{b} = x_a x_b + y_a y_b \quad (2.15)$$

и така получаваме

$$\cos \alpha = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} \quad (2.16)$$

За намирането на вътрешните ъгли на даден полигон ще трябва да проверим дали полигонът в околност на даден връх е изпъкнал или не изпъкнал. За да може да проверим това ние трябва да намерим ориентацията на полигона. За да намерим вътрешните ъгли посоката на полигона трябва да бъде обратно на часовниковата стрелка *anti - CW*. След това започваме да проверяваме всеки три точки от полигона $list(pt_i, pt_{i+1}, pt_{i+2})$. Намираме вътрешния ъгъл α , който е при връх pt_{i+1} . Правим проверка за ориентацията на трите точки. Ако са ориентирани по часовниковата стрелка *CW*, то от 2π трябва да извадим ъгъла α . Ако ориентацията на върховете $list(pt_i, pt_{i+1}, pt_{i+2})$ е обратно на часовниковата стрелка (*anti - CW*), то записваме ъгъла α без корекция.

Algorithm 1 InsidePolyAngle

/*Функция за намиране на вътрешни ъгли на полигон*/

procedure INSIDEPOLYANGLE(pt_0, pt_1, \dots, pt_n) **if** isClockWise pt_0, pt_1, \dots, pt_n **then return** ptList = reverse pt_0, pt_1, \dots, pt_n $i = 0$ $L = \text{length of } pt_0, pt_1, \dots, pt_n$ Repeat L **for** pt_i, pt_{i+1}, pt_{i+2} **do** $\alpha = \text{getInsideAngle } pt_i, pt_{i+1}, pt_{i+2}$ **if** isClockWise pt_i, pt_{i+1}, pt_{i+2} **then return** $\alpha = (2\pi - \alpha)$ **else** α $i = i + 1$ End Repeat

По този начин ще може към всеки един полигон да се запише информация за вътрешните ъгли и дължините на страните му.

2.9 Пресичане на две прави

Намирането на пресечна точка между две прави е "скъпо" струваща операция от гледна точка на процесорно време и би следвало да се използва в "краен" случай. Затова тук ще разгледаме две функции. Първата е намиране на координатите на пресечната точка pt_x на два дадени сегмента e_1 и e_2 , а втората е проверка дали два дадени сегмента e_1 и e_2 се пресичат, без да се търси самата пресечна точка. Първата функция ще връща като стойност пресечната точка $list = (x_i, y_i)$, а втората – *true* или *false*.

Намиране на координати на пресечна точка pt_x .

Съгласно [31] и [30] пресечната точка $ptX = (X, Y)$ на два дадени сегмента $e_1 = (x_1, y_1)$ и $e_2 = (x_2, y_2)$ има координати:

$$X = \frac{\begin{vmatrix} x_1 & y_1 & x_1 & 1 \\ x_2 & y_2 & x_2 & 1 \\ x_3 & y_3 & x_3 & 1 \\ x_4 & y_4 & x_4 & 1 \end{vmatrix}}{\begin{vmatrix} x_1 & 1 & x_1 & 1 \\ x_2 & 1 & x_2 & 1 \\ x_3 & 1 & x_3 & 1 \\ x_4 & 1 & x_4 & 1 \end{vmatrix}}, Y = \frac{\begin{vmatrix} x_1 & y_1 & y_1 & 1 \\ x_2 & y_2 & y_2 & 1 \\ x_3 & y_3 & y_3 & 1 \\ x_4 & y_4 & y_4 & 1 \end{vmatrix}}{\begin{vmatrix} x_1 & 1 & y_1 & 1 \\ x_2 & 1 & y_2 & 1 \\ x_3 & 1 & y_3 & 1 \\ x_4 & 1 & y_4 & 1 \end{vmatrix}}. \quad (2.17)$$

Като пресметнем детерминантите в (2.17), получаваме следните изрази за X и Y :

$$X = \frac{(x_1 y_2 - y_1 x_2)(x_3 - x_4) - (x_1 - x_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(x_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \quad (2.18)$$

$$Y = \frac{(x_1 y_2 - y_1 x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 y_4 - y_3 x_4)}{(x_1 - x_2)(x_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}. \quad (2.19)$$

2.10 Точка в полигон

Ще разгледаме следната задача в равнината XOY . За дадена произволна точка $T = list(x, y)$ (наречена тествана точка) и полигон $\Pi = list(pt_0, pt_1, \dots, pt_n)$ да се определи дали точката е вътре в полигона или не е. Върховете на полигона са зададени с $pt_i = list(x_i, y_i)$.

Тук ще бъдат разгледани два метода за решаване на тази задача - Ray crossing method (пресичащ лъч), [32], и Balanced sum of angles (Балансирана сума на ъглите). В разработения към дисертациония труд софтуер се използва следният подход преди да се приложи метода на пресичащия лъч за всички сегменти. Първо се проверява чрез метода на пресичащия се лъч дали дадената точка T е в *box* на полигона Π_i . За повече намиране на *box* на полигон виж 2.5.

и ако е в *box* тогава се прилага Ray метода за всички сегменти на полигона Π . Този подход изисква обхождане на целия списък с точки $(pt_0, pt_1, \dots, pt_n)$ и сравнение за намиране на минимум и максимум на всяка координата. Тази проверка се прави бързо, тъй като се свежда до сравняване на две числа. Този подход е оправдан тъй като броя на сегментите в един полигон много бързо нараства. В зависимост от сложността на входните полигони и разрешените ъгли на ротация, полигона за който проверяваме Π може да достигне 300-400 сегмента. Като тази проверка се повтаря n на брой пъти. Сложността на алгоритъма е $\mathcal{O}(n^2)$, но ако се приложи методът изложен в точка 2.13 то сложността може да се намали до $\mathcal{O}(n)$.

Ray crossing method (метод на пресичащия лъч)

Проверка дали дадена точка T е в даден полигон Π . Да припомним, че областта оградена от полигона е затворена, т.е. включваща и границата. За целта построяваме хоризонтална полуправа с начало дадената точка T и крайна точка T_∞ , който ще наречем пресичащ лъч $Ray = list(T, T_\infty)$. Под точка в безкрайността ще разбираме най-голямото число което може да се генерира от дадената CAD система. В повечето случаи 10^{20} е достатъчно като приближение до безкрайност. Идеята на метода на пресичащия лъч се базира на броя пресечни точки с полигона Π . Ако броя на пресечните точки е четен, то точка T е извън полигона, иначе тя е вътре в полигона.

В частните случаи, когато пресечните точки на лъча Ray със сегментите на полигона Π съвпадат с даден връх полигона Π се получават неточни резултати. За решаване на проблема със съвпадение на пречната тока с даден връх от полигона се въвежда критерий за "Възходящ" и "Низходящ" сегмент.

Balanced sum of angles (Балансирана сума на ъглите)

Задачата се свежда до намирането на ориентирания вътрешен ъгъл между векторите $\vec{a} = list(T, pt_i)$ и $\vec{b} = list(T, pt_{i+1})$.

Сега ще дадем критерии кога дадена точка T се намира вътре или вън от даден полигон $\Pi = list(pt_0, \dots, pt_n)$. Построяваме векторите \vec{a}_i от точката T до pt_i , $i = 0, \dots, n-1$ и намираме ориентираните ъгли α_i между \vec{a}_i и \vec{a}_{i+1} , $i = 0, \dots, n-1$. След това пресмятаме

$$Sum_\alpha = \sum_{i=1}^n arccos\alpha_i \quad (2.20)$$

Ако точката T е вътрешна за полигона Π , то сборът на всички вътрешни ъгли Sum_α е равен на $\pm 2\pi$. С едно изчисление на всички ъгли (обхождане) този метод ни дава две важни характеристики за дадения полигон Π :

1. Ако $Sum_\alpha = 2\pi$ полигона Π е ориентиран CW ;
2. Ако $Sum_\alpha = -2\pi$ полигона Π е ориентиран $anti - CW$.

Това е много надежден метод, но не достатъчно бърз съгласно [14]. Също така от критично значение е и грешката, която се акумулира при събирането на ъглите. Обикновено стойностите на ъглите са малки при голям брой на сегменти на полигона и тогава може да се натрупа достатъчно голяма грешка, което от своя страна ще затрудни преценката дали точката T е в полигона Π .

2.11 Добавяне на точки в линеен сегмент

Добавянето на точки в линеен сегмент е необходимо за намирането на повече възможни валидни разполагания на полигоните Π_i , $i = 1, \dots$, в полигона Δ . Виж точка 4.4. Сега да вземем един определен полигон $\Pi = list(e_1, e_2, \dots, e_n)$ от входящия списък от полигони. Всеки сегмент e_i от полигона $\Pi = list(e_1, e_2, \dots, e_n)$ се разделя на три подсегмента. За всеки подсегмент се добавят подробни точки. Принципа за поставяне на подробни точки.

Получаването на подробни точки за сегмент $e_i = list(pt_i, pt_{i+1})$ става чрез полярни координати (базова точка, ъгъл и дължина). Дефинираме функцията *polar* която връща точка по зададена базова точка, ъгъл и дължина. Базовата точка е pt_i . Ъгъла на сегмента e_i спрямо абсцисната ос OX е $angle(pt_i, pt_{i+1})$. Намирането на дължината на всеки сегмент е както следва:

1. Разделяме сегмента e_i в съотношение $L_1 = 0.1(distance = pt_i, pt_{i+1})$.
2. L_1 се разделя на съответния брой сегменти, които искаме да постигнем. Може да се използват 3 или 5 деления.
3. Първата подробна точка ще бъде $pt_{L_1,i} = (polar(pt_0, ang_{e_1}, \frac{L_1}{5}))$.

Изчисляването на другите подробни точки за сегмента e_i става в същата логика както за точка $pt_{L_1,i}$. Добавянето на тези подробни точки по границата на полигона в някои случаи повишава качеството на валидните решения. При тестване, ако поставям полигона Π_i във връх pt_0 , тогава няма да получим валидно решение (разполагане на планката). Но, ако полигона Π_i се постави в някоя от подробните точки валидно решение ще има.

2.12 Премахване на излишни точки от линеен сегмент

Проверката се изпълнява преди да започне самото подреждане на полигоните. При представянето на полигоните като списък от върхове $list(pt_0, pt_1, \dots, pt_n)$ е възможно триъгълник да се опише с повече от три точки. Така, че броя на върховете (точките) в един списък не определя вида на фигурата.

2.13 Пресичане на два полигона

Функцията за намиране на сечението (пресичане) на два полигона A и B е една от основните операции с множества и често използвана в алгоритмите за разкрой.

За съжаление реализацията на тази операция изисква много процесорно време. Класическият подход се заключава в проверка дали всеки сегмент от границата на B пресича сегмент от границата на A . Този метод е лесен за реализация, но възможно най-бавен, има сложност $\mathcal{O}(n^2)$, [10], стр. 21. Този подход не се използва в разработения от автора софтуер.

По-бързи методи за проверка дали два полигона се пресичат използват намирането на точките на пресичане на границите им с техните координати или проверка за логическо пресичане. В повечето случаи ще използваме логическото пресичане на двата полигона, тогава не е необходимо да знаем броя или координатите на пресечните точки, достатъчно е да знаем, че един от двата полигона имат поне един връх, който е във вътрешността на другия полигон. Това се използва също при търсенето на възможно положение на даден полигон P_i спрямо основния полигон Δ . В секция 4.4 при 2D разкроя ще дадем повече информация.

Една от възможните проверки за пресичане е като започнат да се пускат лъчи от всеки връх от полигон B . И ако даден връх от полигона B е в полигона A , то двата полигона A и B се пресичат. В този подход не е задължително да се проверяват всички точки от полигон B дали са вътрешни за полигон A . Методът е надежден, но сложността му е почти $\mathcal{O}(n^2)$, тъй като всеки един лъч реално е сегмент с начален връх B_i . Този подход е сравнително по-бърз от класическия метод.

Един от бързите алгоритми е този на "Bentley–Ottmann algorithm", [39], който се заключава във въвеждане на вертикално или хоризонтално "сканираща" линия преминаваща през всички сегменти. При достигане началото на даден сегмент отчитаме "събитие" и при достигане края на сегмента имаме също "събитие". За разработването на алгоритъма се използва вертикална сканираща линия. Съгласно [43] сложността на този алгоритъм за всички K пресичания между N сегмента е $\mathcal{O}((N + K) \log N)$.

2.14 Редуциране на върховете на полигон.

В процеса на търсене на валидно решение по контура на полигона се "вмъкват" подробни точки, които повишават съществено вероятността да се намери валидно решение. В алгоритъма разработен от автора [13] на всеки сегмент се поставят по 15 подробни точки. Три интервала по пет точки. Това е областта на търсене на решения - тези подробни точки. За всяка една от тях входящия полигон се транслира и завърта до намиране на валидно решение. Валидно решение е това което входящия полигона е в полигона на запълване. Без пресичане на двата полигона.

Редукцията на полигона ще повиши значително скоростта на алгоритъма и съответно ще спести изчислително време. За редуцирането на полигона трябва да определим неговата посока на построение. Да приемем, че посоката на построение на полигона Δ е CW . Отваряме нов празен списък и започваме обхождане на полигона Δ със всеки три точки $list(pt_{i-1}, pt_i, pt_{i+1})$, $i = 1 \dots n$. Изчисляваме посоката на въртене на точките $list(pt_{i-1}, pt_i, pt_{i+1})$. Ако посоката им съвпада с посоката на въртене на полигона Δ , правим проверка дали $(getPolyArea(pt_{i-1}, pt_i, pt_{i+1}) < minArea$ и ако това е така то не записваме точка pt_i в празния списък, иначе записваме pt_i в празния списък. Ако не съвпада посоката на въртене на $list(pt_{i-1}, pt_i, pt_{i+1})$ с Δ то записваме pt_i в списъка.

Глава 3

Задача за $1D$ разкрой.

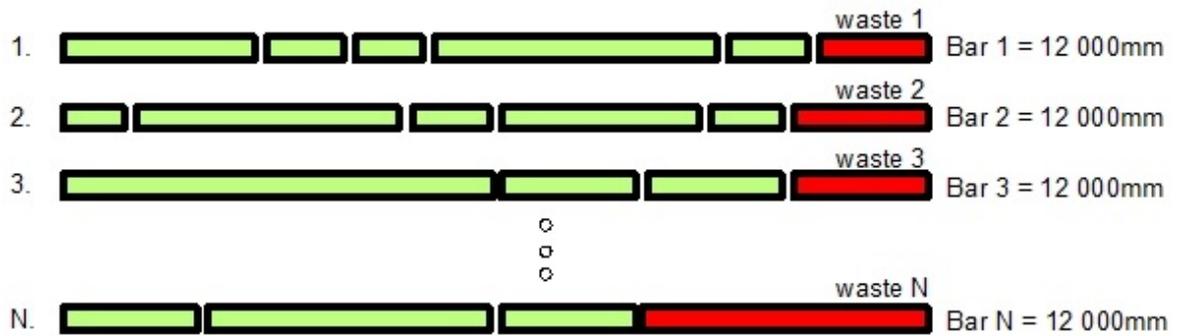
3.1 Формулировка на задачата.

Проблемът за оптимално разкрояване на елементи от зададена заготовка датира от началото на индустриалната революция, втората половина на 18-ти век и началото на 19-ти век. Характерно за този период от време е експоненциалното развитие на производствените сили. Индустриалната революция е свързана не само с началото на масовото използване на машини, но и с рязкото повишаване на производителността на труда. Високата производителност на труда е в пряка и правопрпорционална зависимост от разхода на суровини. От тук се ражда необходимостта от оптимално използване на ресурсите в производството. Задачата за оптимален линеен разкрой засяга предимно промишлеността. Индустрията е сектор, който включва добива на полезни изкопаеми и преработката на суровини в междинни или крайни продукти. Условно може да разделим индустрията на два сектора добивна и преработваща. Във вторичния сектор се включва и строителството. Нашата задача произлиза от вторичния сектор - строителството. В строителството се използват основно няколко вида материали: Стоманобетон, стомана, дърво и други. В случая ще се фокусираме върху стоманените и дървените конструкции. Тези конструкции позволяват да се произведат в цех и да се монтират на строежа. Производството и на двата вида материали (стомана и дърво) позволяват разкрояване. Да вземем за пример стоманената конструкция показана на фигура 1.1.

В тази конструкция за прътовите елементи се използват сечения от най-различен вид. Двойно Т” профили, "L" профили, "С" профили.

Сеченията стоманените профили често достигат 100 кг/м. При цена на един килограм стомана от порядъка на 3,5 лв./кг. (към 2021 година) прави 350 лв. на линеен метър. И когато можем да направим икономия, която се повтори за всеки профил, то ползата от оптимален разкрой е очевидна.

Даден е списък от дължини на входящи профили $L = l_i$. Решението на задачата за $1D$ ще бъде сведено до намирането на решение за един Var_i . Или това е линейното разполагане на част от профилите l_i в дадена дължина Var_i . Следващите етапи са до изчерпването на всички профили от списъка L . Оптимизацията се заключава в такова разполагане на профилите, че да се получи възможно най-малък остатък за всяка една дадена дължина Var_i , виж фигура 3.1.



Фигура 3.1: Дефиниране на линеен разкрой.

Където Bar_i , $i = 1 \dots n$, виж фигура 3.1 е изобразена целевата дължина на запълване за намиране на едно разполагане на профилите. Проблема ще бъде решен с метода на мравките *ACO*. Метода на мравките е метаевристичен метод за решаване на изчислителни задачи [16]. Този алгоритъм е част от алгоритмите на Социалния интелект (SWARM модели на еволюцията на културата).

В дадения случай тя е 12 000 единици. Тя не може да бъде по-малка от най-малката дължина на входящите профили. За намирането на едно валидно решение взимаме това с най-малка стойност на остатъка $waste_1$ им между всичките намерени решения. Трябва да обърнем внимание, че $waste_1$ е съставен от два отпадъка. Единия отпадък е реалния - този, които остава като материал, записан в променлива $wasteReal$. Другия е технологичен, записан в променлива $wasteCut$. Това е ширината на режещия инструмент. Следователно общия отпадък е $waste_i = wasteReal + wasteCut$. След това профилите включени в избраното решение (зеления цвят на фигура 3.1) ги изключваме от входящия списък. Задачата се повтаря до като входящия списък стане празен. След това записваме в една променлива $sumWaste = (waste_1 + waste_2 + \dots + waste_n)$ фирата от всички профили и запазваме решението. След като получим следващото решение ги сравняваме по $sumWaste$. Избираме това с по-малкия $sumWaste$. Решенията се повтарят или докато получим фира под 5% или до дадено време за изчисление.

3.2 Намиране на цялостно решение за 1D разкрой.

Нека е даден следния входящ списък с профили за разкрояване. Входящият списък се получава директно от *CAD* системата. Един такъв списък е показан в таблица 3.1.

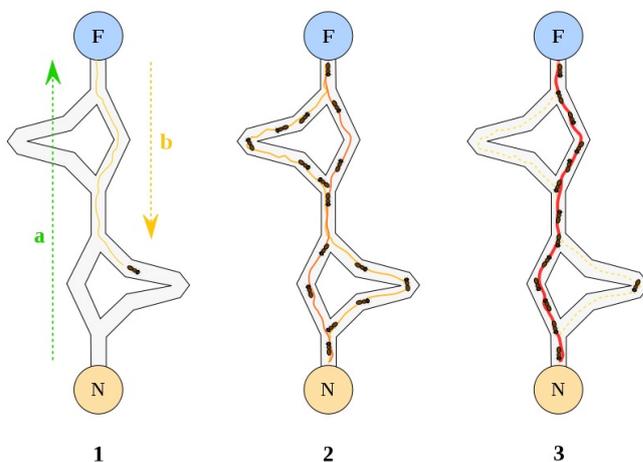
Входни данни за 1D разкрой:

Таблица 3.1: Входни данни за разкрой на профили.

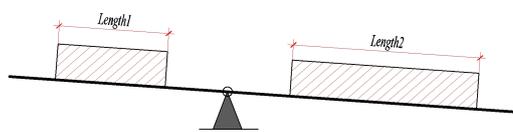
n	Сечение	Броя	Дължина
1	2	3	4
1	Profile 1	36	320
2	Profile 12	54	330
3	Profile 8	4	330
4	Profile 15	18	334
5	Profile 31	54	340
6	Profile 25	54	350
7	Profile 19	360	365

С n ще отбележим "свития" списък с 18 броя елемента. С N ще отбележим развития списък с 580 броя елемента.

Изпробвани са три метода на подреждане. Първият е комбинаторна оптимизация по метода на мравките *ACO*, [53] и [54]. Използвана е една мравка за намиране на решение за един Var_i .



Фигура 3.2: Илюстрация на метода мравките.



Фигура 3.3: Оценка.

Метода на мравките (Ant Colony Optimization) е част от методите с популации. Методите с популации са част от Метаевристиката. Виж фигура 4.1. Най-общо това е вероятностен подход за решаване на най-различни изчислителни задачи. АСО метода решава задачи по зададени параметри. За първи път този метод е предложен от проф. Марко Дориго през 1992 в неговата дисертация.

От тогава до днес методът предложен от проф. Дориго е разширяване и модифицирана за да може да решава по-широк кръг от задачи. Идеята е взета от природата, затова тук ще използваме термини като "мравка" и "феромон". Феромона е химическо вещество което се отделя от мравките в процеса на изминаване на даден път. Този феромон служи за комуникация с други мравки. В началото мравките се движат на случаен принцип. При откриване на храна се завръщат в гнездото си. През целия път на отиване и връщане те отделят феромон. Този феромон се отлага по пътя им. Феромона привлича мравките. Колкото повече феромон има на даден път толкова повече вероятността мравките да се движат по него е по-голяма. По този начин количеството феромон се увеличава и пътя до източника на храна става по привлекателен за другите мравки. За решаването на всяка една задача могат да се използват различен брой мравки. Колкото по-малко мравки се използват за намирането на глобалния оптимум толкова по-малко изчислителни ресурси (процесорно време) ще са необходими. В настоящия случай е използвана една мравка. Мравката избира един произволен валиден елемент и го поставя във валидните решения. За следващо решение използва функция наречена вероятност на прехода. Тази функция е произведение на количеството феромон и евристична информация. Количеството феромон представлява опита на предходните итерации на мравките. Евристичната функция е информация представляваща предварително познание за задачата. Мравката избира този преход, който има на голямо произведение на феромона и евристичната информация. Това е най-голямата вероятност на вярно решение. Ако има повече от едно решение с еднаква вероятност, то се избира едно от тях на произволен принцип. След като всички мравки намерят решенията си, следва да се обнови феромона. Първо феромона се намалява за да се намали влиянието от предишни решения. След това се добавя нов феромон пропорционален на стойността на целевата функция. Като логиката е, че решенията с повече феромон са по-добри от тези с по-малко феромон и така те ще станат по-желани на следващата итерация. В конкретната задача феромонът се поставя на преходите.

Вероятност на прехода.

$$p_{i,j} = \frac{(\tau_{i,j}^\alpha)(\eta_{i,j}^\beta)}{\sum (\tau_{i,j}^\alpha)(\eta_{i,j}^\beta)}, \quad (3.1)$$

където:

- $\tau_{i,j}$ е количеството феромон, съответстващо на прехода от връх i във връх j ;
- α е параметър за контрол на влиянието на $\tau_{i,j}$;
- $\eta_{i,j}$ е евристична информация. Комбинация от параметрите на целевата функция и ограниченията;
- β е параметър за контрол на влиянието на $\eta_{i,j}$.

Обновяване на феромона.

$$X = \begin{cases} L_k, & \text{ако мравка } K \text{ премине през реброто } i, j \\ 0, & \text{иначе} \end{cases} \quad (3.2)$$

Алгоритъм на метода на мравките *ACO*, съгласно [26].

Algorithm 2 ACOAlgorithm

/*Алгоритъм на мравките*/

procedure ACOALGORITHM

Begin

 Поставяне на начален феромон

While докато критерия е истина **do**

 Поставяне на всяка мравка в начален връх

Repeat

For each , приложи за всяка мравка

 Избор на следващ връх

End Foreach , край на приложи за всяка мравка

Until , всяка мравка е построила решение

 Обновяване на феромона

End While , край на while

End

От създаването на алгоритъма през 1992 год. от проф. Дориго до днес са разработени различни модификации на *ACO* алгоритъма. Едни от най-популярните варианти на алгоритъма за оптимизация по метода на мравките са изложени в [17], [18], [19], [20], [21], [22], [23], [53], [54],

В настоящата задача за $1D$ разкороя най-голям феромон получават тези профили, които дават най-малък остатък от Var_i . Дължината на профила е неговата тежест, защото сечението е едно и също. То е константа. Виж фигура 3.3. Профил $Length2$ ще получи по-голям феромон (оценка) спрямо профил $Length1$. Или това са по-дългите профили ще имат по-висока оценка. Сбора на феромона за Var_i е по-голям при по-дългите профили. В този случай метода на мравките проявява *Greedy* характер.

Вторият метод е динамично оптимизиране. При този подход не се прави комбинации между профилите, а записва сбора на дължините на профилите до даден индекс. При следващо търсене на сбора на дължините не се преминава през целия списък до дадената позиция, а се използва сбора им от предишни изчисления. Нека е даден списък с профили с техните дължини $L_p = list(p_0, p_1 \dots p_n)$. Сбора от дължините на първите 5 профила ще бъде:

1. $sum_1 = p_0 + p_1;$
2. $sum_2 = sum_1 + p_2;$
3. $sum_3 = sum_2 + p_3;$
4. $sum_4 = sum_3 + p_4;$
5. ...
6. $sum_i = sum_{i-1} + p_i;$

Третият метод е комбинация *Greedy* и комбинаторен метод с пълно изчерпване на n -елемента, k -ти клас. В *Greedy*. метода профилите се сортират по дължина. От най-голям към най-малък. Подреждането на профилите започва като първи се поставят най-големите дължини. Следващото поставяне е комбинаторния метод. Метода най-голямата дължина от сбора на дължините от всеки до три елемента в списъка L_p . Класа на комбинацията е ограничен до три елемента. Всяка комбинация се различава една от друга с поне 1 елемент.

$$C_n^k = \frac{V_n^k}{P_n^k} = \frac{n!}{k!(n-k)!} \quad (3.3)$$

Или при десет профила имаме следния брой комбинации с три елемента:

$$\frac{10!}{3!(10-3)!} = \frac{3628800}{6.5040} = 120 \quad (3.4)$$

Дължините на профилите с еднакво сечение не варират в големи граници. При направените сравнения между трите метода, за целите на стоманените конструкции хибридният подход *Greedy* + n^3 алгоритъма дава най-добри резултати по отношение на плътност и време. Комбинаторният метод работи със "свития" списък ще бележим с n елемента = 18 броя. Максималния брой итерации е $7^3 = 343$. Не е задължително след всяко поставяне на профил броя на елементите n да намалее с едно. При развития списък ще бележим с $N = 580$ броя. Максималният брой итерации би следвало да бъдат е $580^3 = 195112000$.

Дължини за разкрояване 12000 мм , ширина режещия нож 0. мм. Броят на дължините (профилите) за разкрояване е неограничен. Брой профили за разкрой - 580. Обща дължина на профилите за разкрояване е 205 332 мм. Следователно глобалният минимум ще бъде около $\frac{205332}{12000} = 17.11$, или до 18 броя профили по 12 000м.

3.3 Резултати при 1D разкрой. Примери.

Ще използваме следния комерсиалния продукт:



Фигура 3.4: Комерсиален продукт за 1D разкрой.

Резултати от изчисленията:

Cutting layouts						
Plan #1 - U 40 x 3 mm - Simple						9.9.2019 r/
Note 1	demo plan	Cost				
Note 2		Yield	95.06%	216 000.0		
Note 3		Gross yield	95.06%	216 000.0		
Name		Stocks	18	216 000.0		
		Parts	580	205 332.0		
		Layouts	4			
		Uncut parts				
Kerf	Left trim	Min remnant length				
Part increase	Right trim	Rem. storage - Remnants				
Layout	Stock #	Description	Rest	Length	Repeat	
1 of 4	1			12 000.0	12x	
<p>Diagram showing a bar of length 10 950.0. Part 7 (30 x 365.0) is cut from the start. Part 6 (3 x 350.0) is cut from the end.</p>						
#	Part #	Description	Order #	Length	Count	
1	7			365.0	30	
2	6			350.0	3	
Layout	Stock #	Description	Rest	Length	Repeat	
2 of 4	1			12 000.0	4x	
<p>Diagram showing a bar of length 9 760.0. Parts 6 (4 x 350.0), 5 (11 x 340.0), 3 (330), 2 (13 x 330.0), and 1 (7 x 320.0) are cut from the start.</p>						
#	Part #	Description	Order #	Length	Count	
1	6			350.0	4	
2	5			340.0	11	
3	3			330.0	1	
4	2			330.0	13	
5	1			320.0	7	
Layout	Stock #	Description	Rest	Length	Repeat	
3 of 4	1			12 000.0	1x	
<p>Diagram showing a bar of length 9 19 440.0. Parts 6 (2 x 350), 5 (10 x 340.0), 4 (15 x 334.0), 2 (330), and 1 (8 x 320.0) are cut from the start.</p>						
#	Part #	Description	Order #	Length	Count	
1	6			350.0	2	
2	5			340.0	10	
3	4			334.0	15	
4	2			330.0	1	
5	1			320.0	8	
Layout	Stock #	Description	Rest	Length	Repeat	
4 of 4	1		10 668.0	12 000.0	1x	
<p>Diagram showing a bar of length 1 002.0. Parts 4 (3 x 334.0) and 2 (330) are cut from the start.</p>						
#	Part #	Description	Order #	Length	Count	
1	4			334.0	3	
2	2*		*	330.0	1	

Фигура 3.5: Решение на 1D разкрой с комерсиален продукт.

В сините елипси на фигура 3.5 са посочени броя на необходимите профили. Те са 17 броя цели дължини по 12 000мм + 1002 мм от 18-та дължина.

Резултати в настоящия метод:

Таблица 3.2: Резултати с Мравки за 1D разкроя.

N	Дължини	Остатък	Профили
1	2	3	4
1	12000	120	profile 19 (360x33),
2	12000	120	profile 19 (360x33),
3	12000	120	profile 19 (360x33),
4	12000	120	profile 19 (360x33),
5	12000	120	profile 19 (360x33),
6	12000	120	profile 19 (360x33),
7	12000	120	profile 19 (360x33),
8	12000	120	profile 19 (360x33),
9	12000	120	profile 19 (360x33),
10	12000	120	profile 19 (360x33),
11	12000	120	profile 12 (330x36),
12	12000	120	profile 31 (330x36),
13	12000	100	profile 25 (350x34),
14	12000	120	profile 1 (320x36), profile 19 (360x1),
15	12000	160	profile 19 (360x29), profile 25 (350x4),
16	12000	120	profile 31 (330x18), profile 12 (330x18),
17	12000	130	profile 15 (330x18), profile 25 (350x16), profile 8 (330x1),
18	12000	11010	profile 8 (330x3),

Таблица 3.3: Резултати с *Greedy* + n^3 метод за 1D разкроя.

N	Дължини	Остатък	Профили
1	2	3	4
1	12000	150	profile 1 (320x36), profile 12 (330x1),
2	12000	120	profile 12 (330x36),
3	12000	120	profile 12 (330x17), profile 8 (330x4), profile 15 (330x15),
4	12000	120	profile 15 (330x3), profile 31 (330x33),
5	12000	170	profile 31 (330x21), profile 25 (350x14),
6	12000	100	profile 25 (350x34),
7	12000	180	profile 25 (350x6), profile 19 (360x27),
8	12000	120	profile 19 (360x33),
9	12000	120	profile 19 (360x33),
10	12000	120	profile 19 (360x33),
11	12000	120	profile 19 (360x33),
12	12000	120	profile 19 (360x33),
13	12000	120	profile 19 (360x33),
14	12000	120	profile 19 (360x33),
15	12000	120	profile 19 (360x33),
16	12000	120	profile 19 (360x33),
17	12000	120	profile 19 (360x33),
18	12000	10920	profile 19 (360x3),

Таблица 3.4: Сравнение на резултатите за 1D разкроя.

Софтуер	Броя използвани профили	Време[s]
1	2	3
Комерсиален продукт	17x12000 + 1002 мм	7
<i>Greedy</i> + n^3	17x12000 + 1080 мм	< 1
Мравки <i>ACO</i>	17x12000 + 990мм	1

Явно комерсиалния продукт ползва много сложна евристика или други методи за оптимизиране. Комерсиалният продукт е най-бавен спрямо другите два метода *Greedy* + n^3 и *ACO*. Както се вижда от сравнителната таблица 3.4, методът на мравките *ACO* дава най-добър резултат от гледна точка на най-малко отпадък, което е главната ни цел. Като време за пресмятане *Greedy* + n^3 е малко по-бърз от *ACO* за сметка на по-лошото решение. Комерсиалният продукт е бавен и дава по-лошо решение от метода на мравките *ACO*. Затова може да заключим, че предложеният от автора на дисертацията *ACO* алгоритъм превъзхожда другите два.

Глава 4

Задача за $2D$ разкрой.

4.1 Формулировка на задачата.

Индустрията поставя за решаване най-различни оптимизационни задачи. Тези задачи могат да се класифицират по много признаци зависещи от:

1. характера на решавания проблем;
2. структура на задачата;
3. броя на управляващите параметри;
4. характера на зависимостта на критерия и ограниченията на параметрите;
5. наличието на различни ограничения;
6. характера на търсения минимум;
7. броя на критериите;
8. и други.

При производството на стоманени конструкции се налага да се изрязват планки от даден стоманен лист. Планките идват като полигони от дадена CAD система. Това налага подреждане на входящите планки върху листа така, че да се получи минимална фира. Това е изрязването на определен брой фигури от даден материал, които в общия случай ще бъде полигон. Този полигон ще наричаме полигон за запълване. Виж фигура 1.4.

Тази задача е известна е още като Cutting Stock Problem или (CSP), [69]. Този проблем е NP- сложна комбинаторна задача [88]. В литературата се дават точни решения на задачата за фигури (планки), които са правоъгълници. По долу ще бъде даден алгоритъм за намиране на едно решение на задачата за разполагане на даден брой произволни геометрични фигури (планки описани чрез полигони) вписани в произволен контур (полигон за запълване). Методът позволява използване на въртене и огледален образ на фигурата. Процесорното изчислителното време се увеличава съществено с увеличаване на броя на фигурите и на тяхната сложност като геометрия. Намирането на решение чрез изчерпване на всички възможни комбинации е неприемливо като твърде голям обем изчисления. При съвременното развитие на изчислителната техника е възможно да се реши сложна задача на супер компютър, който да намери всички възможни решения, но в повечето случаи това не е оправдано. Разбира се, разход на значителен изчислителен

ресурс зависи от важността на задачата. Представеният по-долу алгоритъм има възможност за паралелизация на изчислителните процеси. Но масовите задачи ще се реализират на настолен или персонален компютър. Целта е да се създаде алгоритъм, който дава за кратко време приемливо решение на сложни комбинаторни задачи използвайки мобилни изчислителни устройства. Ще въведем всички математически понятия, които се използват за описание на математическия модел и алгоритъм за решаването на оптимизационната задача.

От САД системата се получават полигони, които са списъци от точки в координата на система XOY . В общия случай тези полигони съдържат и точки, които не са върхове на полигона, т.е. точки лежащи на една права. Тези точки ще наричаме излишни. Затова на всички полигони генерирани от САД системата ще приложим функцията за изчистване на излишните точки. След премахване на излишните точки получаваме входящ допустим полигон описан чрез списък от точки

$$\Pi = list(pt_0, pt_1, \dots, pt_n), \quad (4.1)$$

където pt_i са върхове на полигона. Този списък има следните свойства: 1. Списъкът е подреден; 2. Списъкът е цикличен $pt_n = pt_0$; 3. Няма самопресичане.

В процеса на работа на алгоритъма ще ни бъде необходимо понятието сегмент, което е отсечка между два последователни върха. По този начин генерираме сегментите $e_1 = list(pt_0, pt_1), \dots, e_n = list(pt_{n-1}, pt_n)$ и получаваме друга характеристика на полигона $\Pi = list(e_1, e_2, \dots, e_n)$. Планки с криволинейни граници се апроксимират с полигони с достатъчен брой върхове. Примери за входящи допустими полигони са показани на фигура 1.5.

Полигон за запълване Δ , които трябва да се запълни също ще се описва със списък от точки:

$$\Delta = list(p_0, p_1, \dots, p_m) \quad (4.2)$$

Контурът за запълване не трябва да бъде самопресичащ се.

Има два критерия за оптимизация на задачата:

1. Оптимизация на минимална височина на запълване - $minY$;
2. Оптимизация на броя на върховете на остатъчния полигон след изрязването на входящия полигон от полигона на запълване - $minVertex$.

Да разгледаме оптимизация на минималната височина.

Из между всички разполагания за най-добро ще смятаме онова което с най-малка ордината в координата система XOY . Ако се получат повече от едно решение с еднакви ординати $minY$ то взимаме това с най-добър коефициент на запълване. Ако има повече от едно решение с максимален коефициент на запълване избираме тези решения, които изрязват полигона за запълване с най-малък брой върхове. Това означава, че изрязването постига "правилни" фигури. Ако има повече от едно решение с най-малък брой върхове, тогава избираме първото в списъка.

За намиране на лице на полигон виж формула 2.12. Тази формула дава удвоеното лице на полигона.

Тогава определяме коефициента на запълване $ratio$ като:

$$ratio = \frac{\sum_{i=1}^n F_i}{A_P} \quad (4.3)$$

Следователно минималната стойност $ratio = 0$ на коефициента е при не запълване на полигона A_P . Максималната стойност $ratio = 1.0$ е при максимално запълване на A_P .

Разбира се, дефиницията за $minY$ води до положението как е въведен полигона за запълване P . Ако искаме да избегнем този проблем то решението на задачата ще трябва да се проведе за няколко различни ъгли на полигона за запълване P . Ъглите на ротация ще бъдат ъглите, които всеки сегмент сключва с абцисната ос. Броя на ротациите е равен на броя на сегментите на съответния полигон.

Да разгледаме оптимизация на броя на върховете на остатъчния полигон.

Из между всички разполагания за най-добро ще се смята онова което при изрязването на входящия полигон в полигона на запълване има най-малко върхове на остатъчния полигон. В настоящия дисертационен труд това е критерият един полигон да бъде "по-гладък". Намирането на подходящ полигон от входящите полигони ще става чрез сравнение на страните на двата полигона - входящият и полигона за запълване. Ако имаме пълно съвпадение на дължините на страните на двата полигона, то ще изберем този входящ полигон. Ако няма никакво съвпадение на страните на входящия полигон към то полигона на запълване, то ще използваме съвпадение на ъглите на двата полигона.

Ако се получат повече от едно решение с еднакъв брой върхове на остатъчните полигони $minVertex$, то взимаме първото решение от списъка с валидни решения. Разликата между $minY$ и $minVertex$ е в това, че при $minVertex$ се дава възможност на големи (дълги) полигони да "влезнат" първи в полигона на запълване, защото вероятността да произведат гладък остатък е по-голяма. Минималната гладкост на остатъчния полигон която може да се получи е полигон с три върха $list = (pt_0, pt_1, pt_2)$ с площ различна от нула.

4.2 Стратегия за избор на входящ полигон. Метаевристични методи.

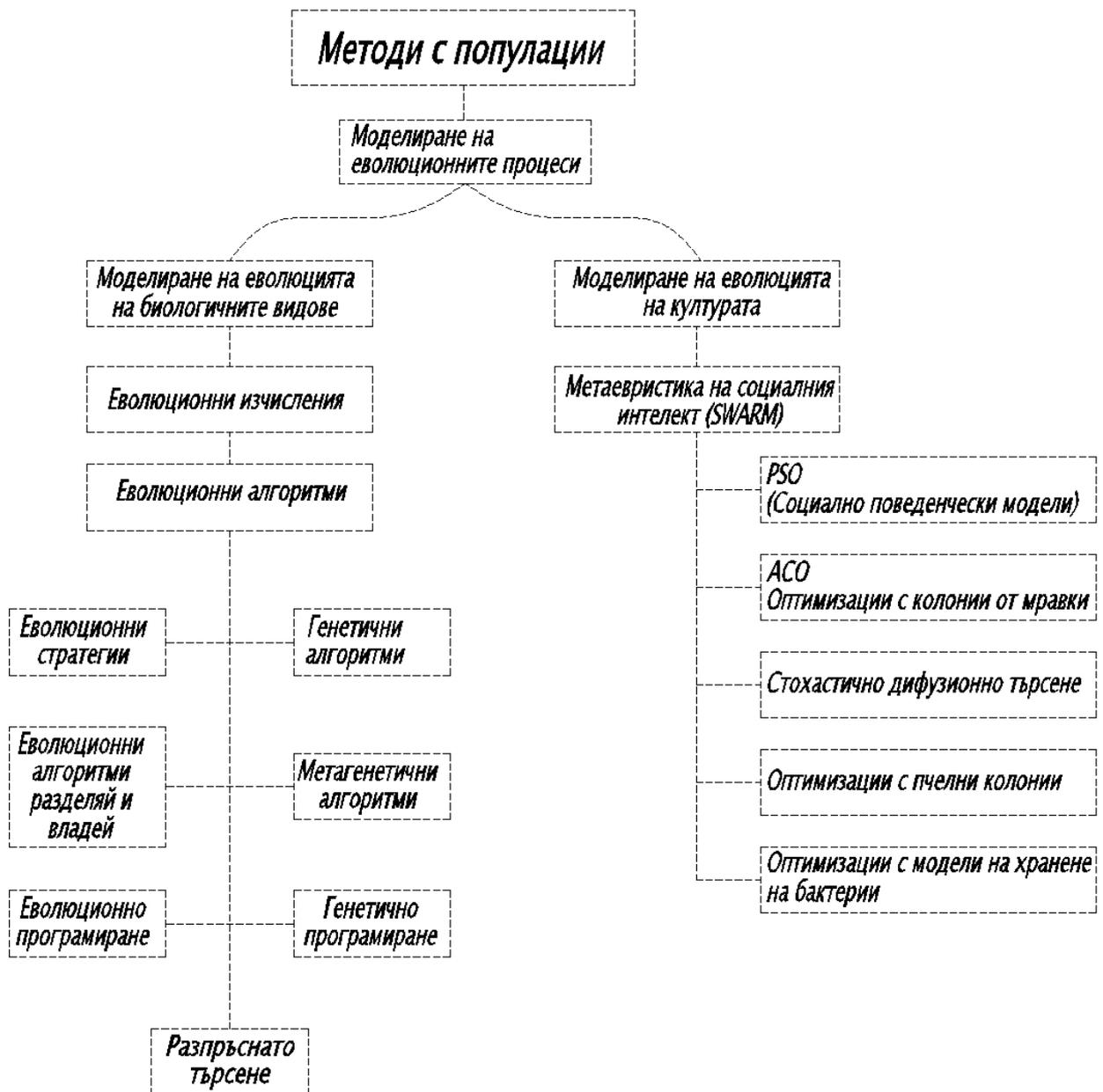
Метаевристиката е мощен инструмент за намирането на оптимално или субоптимално решение на сложни комбинаторни задачи. Основна роля за развитието на метаевристичните методи е необходимостта да се намери приемливо решение за приемливо време при ограничени хардуерни ресурси. Съгласно [34] метаевристичните алгоритми (на английски: metaheuristic algorithms, накратко: метаевристики, metaheuristics) в компютърните науки са алгоритми за математическа оптимизация, с които се решават комбинаторни оптимизационни задачи. Тези задачи в общия случай са сложни, представяни чрез дискретизация на входящите данни. Такива задачи обикновено се характеризират със силна нелинейност, множество параметри, разнообразни сложни ограничения за удовлетворяване и множество – често противоречащи си – оптимизационни критерии. Дори и при един оптимизационен критерий може да не съществува нито едно допустимо решение. Само тогава не съществува оптимално решение. Ако има дори само едно допустимо решение, то тогава задължително съществува и оптимално решение.

Като цяло откриването на оптимално или дори близко до оптималното решение е трудно постижимо.

Терминът "метаевристика" е въведен от Фред Глоувър в основополагащата му статия от 1986 г. като надграждане на термина "евристичен" алгоритъм, с който в най-общ смисъл се разбира алгоритъм за търсене на решение, базиран на пробата и грешката. Частицата "мета" означава "отвъд", "свръх", "на по-високо ниво" и с метаевристичния алгоритъм се означава "по-висша" стратегия, която направлява и модифицира други евристични алгоритми, за да постигне решения по-добри от тези, които нормално биха се получили при търсене на локален оптимум [35], [36]. В допълнение всички метаевристични алгоритми балансират между глобално и локално търсене. Качествените решения на трудни оптимизационни задачи могат да се постигнат в разумно (т.е. полиномиално) време, но без гаранция, че ще бъдат постигнати (глобалните) оптималните решения. Двата основни компонента на всеки метаевристичен алгоритъм са: интензификация и диверсификация (intensification and diversification), или още изследване и експлоатация (exploration and exploitation). Диверсификацията означава да се генерират разнообразни решения, така че пространството на търсене да може да бъде проучвано в широк диапазон, докато интензификацията означава да се фокусира търсенето в локален регион, знаейки, че текущото най-добро решение се намира в този регион. При подбора на най-добрите решения трябва да се открие добър баланс между интензификацията и диверсификацията с цел да се подобри скоростта на сходимост на алгоритъм. Изборът на най-доброто текущо решение осигурява, че решенията ще схождат към оптимум, докато диверсификацията посредством избор на случайни стойности на променливи позволяващи да се избегне попадането в локален екстремум и в същото време да се повиши разнообразието на решението. Добрата комбинация от тези два основни компонента обичайно води до намиране на глобален оптимум. Съгласно [27] основните свойства на метаевристиката могат да бъдат обобщени както следва:

1. Метаевристиката осигурява стратегии за направляване на процеса на търсене;
2. Целта ни е ефективно да се изследва пространството на търсене, за да се открият оптимални или субоптимални решения;

-
3. Метаевристичните техники обхващат широк спектър процедури - от процедури за локално търсене до сложни процедури с машинно обучение;
 4. Метаевристичните алгоритми са приближени и обикновено недетермистични;
 5. Метаевристичните алгоритми в общия случай осигуряват механизми за избягване на съсредоточаване на търсенето само в ограничени области на пространството;
 6. Основните концепции на метаевристиката могат да бъдат описани на абстрактно ниво;
 7. Метаевристичните алгоритми са универсални;
 8. Метаевристиката може да използва знание, специфично за областта под формата на евристика, която се управлява от стратегия на високо ниво;
 9. За направляване на търсенето съвременната метаевристика използва натрупания опит при търсенето;
 10. Метаевристиката представлява стратегии от високо ниво на изследване на пространството на търсене чрез използване на различни методи;
 11. Изисква динамичен баланс между използването на две фундаментални концепции: диверсификация и интензификация.



Фигура 4.1: Класификация на метаевристики с популации.

Хибридната метаевристика осигурява възможности за повишаване на ефективността на търсене като комбинира различните метаевристични алгоритми. В настоящата дисертация е използвана хибридна метаевристика. Използвана е следната стратегия :

1. "Разпръснато търсене" от Еволюционните алгоритми от Метода с популации. Виж фигура 4.1;
2. Вероятностно предвиждане на избора на даден елемент;
3. Йерархично оценяване на решенията.

Имаме даден списък от полигони

$$P_L = (list(list_1(pt_0, pt_1, \dots, pt_n), list_2(pt_0, pt_1, \dots, pt_n), \dots, list_n(pt_0, pt_1, \dots, pt_n))), \quad (4.4)$$

където $list_i$ са полигони описание чрез върховете им. Върховете са описани чрез списък от две реални числа $list(X, Y)$, виж 2.1. Списъкът Π_L ще наричаме списък от входящи полигони.

Полигона Δ , които трябва да се запълни се описва със списък от точки:

$$\Delta = list(p_0, p_1, \dots, p_m) \quad (4.5)$$

Контурът за запълване не трябва да бъде самопресичащ се. В дадената задача полигона за запълване Δ е един на брой. Ако имаме повече от един полигон то решението на задачата се повтаря за всеки един от тях.

Преди да се избере входящо множество е необходимо да се направи оценка за съвпадението на ъглите и страните от текущия входящ полигон

$$\Pi_i = (list(pt_0, pt_1, \dots, pt_n), i = 1, \dots, n)$$

към то ъглите и страните на полигона за запълване $\Delta = (list(pt_0, pt_1, \dots, pt_n))$. За целта се съставят два нови производни списъка на Π_i и Δ . Те съответно съдържат последователно подредени списъци $list(previousLength, Angle, NextLength)$. Добре е елементите на двата списъка да се съставят от конструктивни двойки. Конструктивната двойка е съставена от два елемента - първият с име, вторият с променлива. Може да се запише така $cons("name".AnyValue)$. В "name" записваме "angle" за ъгъл или "length" за дължина. В AnyValue - произволна стойност която може да бъде Integer, Real, String или List.

Или новите производни списъци са:

$$\begin{aligned} \Pi_i = & list((list(cons("previousLength"(distance_{pt_n,pt_0})) \\ & (cons("angle" pt_n, pt_0, pt_1) \\ & (cons("NextLength"(distance_{pt_0,pt_1}))) \\ & \dots \\ & (list(cons("previousLength"(distance_{pt_n}pt_{n-1})) \\ & (cons("angle" pt_0, pt_n, pt_{n-1}) \\ & (cons("NextLength"(distance_{pt_n,pt_0})))))) \end{aligned}$$

Където n броя на върховете на полигона Π_i . Ако $(i + 1) > n$ тогава $i = 0$. Списъка Π_i е цикличен.

Съставя се и подобен списък и на полигона за запълване Δ . Списъка Δ е цикличен.

Всеки елемент от Списъка Π_i се сравнява със съответния елемент от списъка Δ . Най-големият брой на последователно съвпадащи се елементи от двата списъка ще ни даде най-голямата вероятност полигона Π_i да съвпадне с полигона Δ . За да направим коректно сравнение на двата списъка ще трябва да изберем кой списък ще бъде основен. Под основен списък ще разбираме този, който има по-голяма дължина (по-голям брой елементи). В списъка $list(A)$ може да бъде или Π_i или Δ . Не е задължително броя на върховете на Δ да бъде по-голям от броя на върховете на Π_i . Обхождането на двата списъка ще става на базата индекси. Първата итерация е когато индекс 0 от списъка $listB$ съвпада с индекс 0 на списъка $listA$. Сега първия индекс от $listA$ се увеличава с 1. Следва втора итерация когато индекс 0 от списъка B съвпада с индекс 1 на списъка A . Като индекс 0 от $listA$ отива като последен в $listA$. Или списъка $listA$ има циклично поведение. Това се повтаря докато преминем през всички индекси на $listA$ или цикълът се повтаря толкова пъти колкото е дължината на списъка $listA$. За всяка проверка

записваме броя на последователните съвпадения. Оценката на списъка $listB$ се дава със следната формула:

$$k = \sum Ratio_{angle_i} + \sum Ratio_{Length_i}, \quad (4.6)$$

където

$$Ratio_{angle_i} = (if|(angle_i - angle_{Bi})| < fuzz \rightarrow return 1., else 0.).$$

При ъглите търсим пълно съвпадение. Като A_i е ъгъл от списъка $listA$, а B_i е съответния ъгъл от списъка $listB$.

След като имаме пълно съвпадение на съответните ъгли $Ratio_{angle_i} = 1.$, тогава пристъпваме към оценка на съответните им дължини.

$$Ratio_{Length_i} = (if(length_{A_i} > length_{B_i}) \rightarrow return(\frac{length_{B_i}}{length_{A_i}}), else(\frac{length_{A_i}}{length_{B_i}})) \quad (4.7)$$

Условие 4.7 дава коефициенти близки или равни на 1., без значение коя дължина е по-голяма $length_{A_i}$ или $length_{B_i}$. Това е така, защото търсим полигони, на които страните им почти съвпадат.

Както се вижда от формула (4.6) полигоните с по-голям брой върхове ще имат по-голяма вероятност за по-високи оценки на съвпадение. Това е добре, защото след изваждането на двата полигона $A \setminus B$ ще получим полигон с по-малко върхове. Коефициентът от формула (4.6) може да се използва като оценка за подобие или еднаквост на фигури.

4.3 Стратегия за избор на едно решение от валидни разположения.

Нека разгледаме два полигона. Полигона за запълване $\Delta = list(pt_1, pt_2, pt_3, pt_4)$ и входящ полигон $\Pi_i = list(pt_1, pt_2, pt_3, pt_4, pt_5, pt_6)$. Десният Π_i полигон е входящия. Левият полигон Δ е полигона за запълване. Полигона Π_i не е триъгълник! Необходимо е добавянето на подробни точки по границата на полигона за запълване Δ . Тези точки ще бъдат области на поставяне на полигона Π_i и проверката дали полигона Π_i е в полигона Δ . Ако проверката е удовлетворена, то записваме това решение като възможно. Разрешена е ротация на полигона. Огледален образ не е приложен. От тези валидни решения оценяваме тези, които имат най-голяма допирна дължина с полигона Δ и разстоянието $LengthA = distance(pt_0, pt_M)$. Между дължината на опирание и дължината $LengthA$ има зависимост. Тази зависимост е приета от автора. Разбира се могат да се напишат и други зависимости. Оценката на всяко решение е $eval = LengthOfTouch + (2.0 * LengthA)$. Тя дава се тежест на разстоянието от центъра на тежестта на полигона до избрана точка от полигона на запълване. В случая това е точката най-долу, най-вляво. Може да се избере друга точка без значение дали е от множеството точки от полигона за запълване. След като сме оценили решенията, избираме решението със зеления контур на полигона. Изваждаме $\Delta \setminus \Pi_i$.

След като сме избрали първо решение пристъпваме към избор на второ. Избора на входящ полигон става по процедурата описна в точка 4.2. Тъй като целта ни е максимално уплътняване на решенията то избора на второто решение трябва да се търси по контура на вече намерените решения. На този етап се прилага

йерархично оценяване на решенията. Около избраните полигони построяваме правоъгълен контур. Ще го наречем *box*. Първо ще търсим решения, които влизат в *box* от валидните решения, ако не намерим такива използваме за оценка всички валидни решения. Оценяването на решенията вътре в *box* също става по формулата: $eval = LengthOfTouch + (2.0 * LengthA)$. Тук трябва да се отбележи, че тази процедура за избор на решения след първото трябва да обходи всички входящи полигони и тогава да се вземе това с най-висока оценка. В софтуера разработен от автора това обхождане не се прави поради липсата на изчислителен ресурс, но предложения метод не е ограничен в тази посока. При наличие на достатъчно мощен хардуер плюс графични процесори алгоритъма ще даде много близки резултати до глобалния оптимум.

След изрязването на двата полигона се получава новия контур. Този контур ще послужи като контур на запълване за следващите полигони. Както се забелязва нови контур не следва напълно стария. Новия контур целенасочено е редуциран. За повече информация за начина на редуциране на контура виж точка 2.14.

$$ratio_{Global} = \frac{A_{box}}{A_{\Pi_L}} \leq 1.0 \quad (4.8)$$

4.4 Намиране на едно възможно разположение на планка в полигона за запълване.

Намирането на едно възможно разполагане на планките (представяни като полигони) става чрез поставяне на входящ полигон в полигона за запълване и прилагане на функцията изваждане на два полигона. Функцията е описана по-долу. В зависимост от изискването на конкретния случай може да построим производни полигони от входящия полигон често с разрешение за прилагане на ротация и огледална симетрия.

Броят на ротациите, на които можем да ротираме дадения полигон е произволен. Колкото повече ъгли имаме в списъка за ротиране толкова е по-голяма вероятността да получим възможно решение. С цел да ограничим произволното ротиране на полигона без да има качествено решение (полигона да бъде в полигона на запълване) е необходимо да изберем подходящи ъгли на ротиране. Като начало могат да се вземат ъглите на сегментите на входящия полигон с абсцисната ос X . След това могат да се добавят точните ъгли $(0; 0.5\pi; \pi; 1.5\pi)$. Ако е необходимо и огледано прилагане на входящия полигон, то броят на допълнителните полигони ще се увеличи. Някои състояния на ротация ще бъдат напълно идентични като геометрия.

Проверката дали даден полигон се съдържа в друг може да стане по два начина:

1. Проверка за всеки един възел дали е в полигона за запълване. Тази проверка е добре да стане с цикъл `while`, ако текущата тествана точка е извън контура решението отпада, без да продължава нататък;
2. Проверка дали има тривиално пресичане на двата полигона. Ако има пресичане, то решението отпада, иначе – се приема.

Ако първият критерий за оптимално решение е $minY$, то може да се провери само върха на полигона P с най-малка ордината (в случая p_7). Но най-добре е да се направи за всички върхове на полигона за запълване P . Друг критерий за избор

на решение е когато при изрязването на двата полигона се получи гладка фигура. Критерият за гладка фигура ще бъде минимален брой на върховете след изрязване на дадения полигон от полигона за запълване. Възможна е и комбинация за критериите. На избраните валидни решения за $minY$, ще се приложи критерия за минимален брой на върховете след изрязването им с полигона за запълване.

При наличие на многоядрен процесор и езика на които се пише приложението позволяват паралелни изчисления. Паралелните изчисления могат да се направят и за останалите възможни състояния на входящия полигон.

Алгоритъмът се повтаря за генерирания огледален образ на входящия полигон. За огледалния полигон са валидни всички генерирани ъгли на ротация.

4.5 Премахване на реалната фира при 2D разкроя.

За изрязването на фигурите се използва режещ инструмент с определена характеристика на рязането. По тази причина трябва да се осигури разстояние между полигоните. Въвеждаме параметъра $cutW$ за ширина на фугата (ножа) между полигоните.

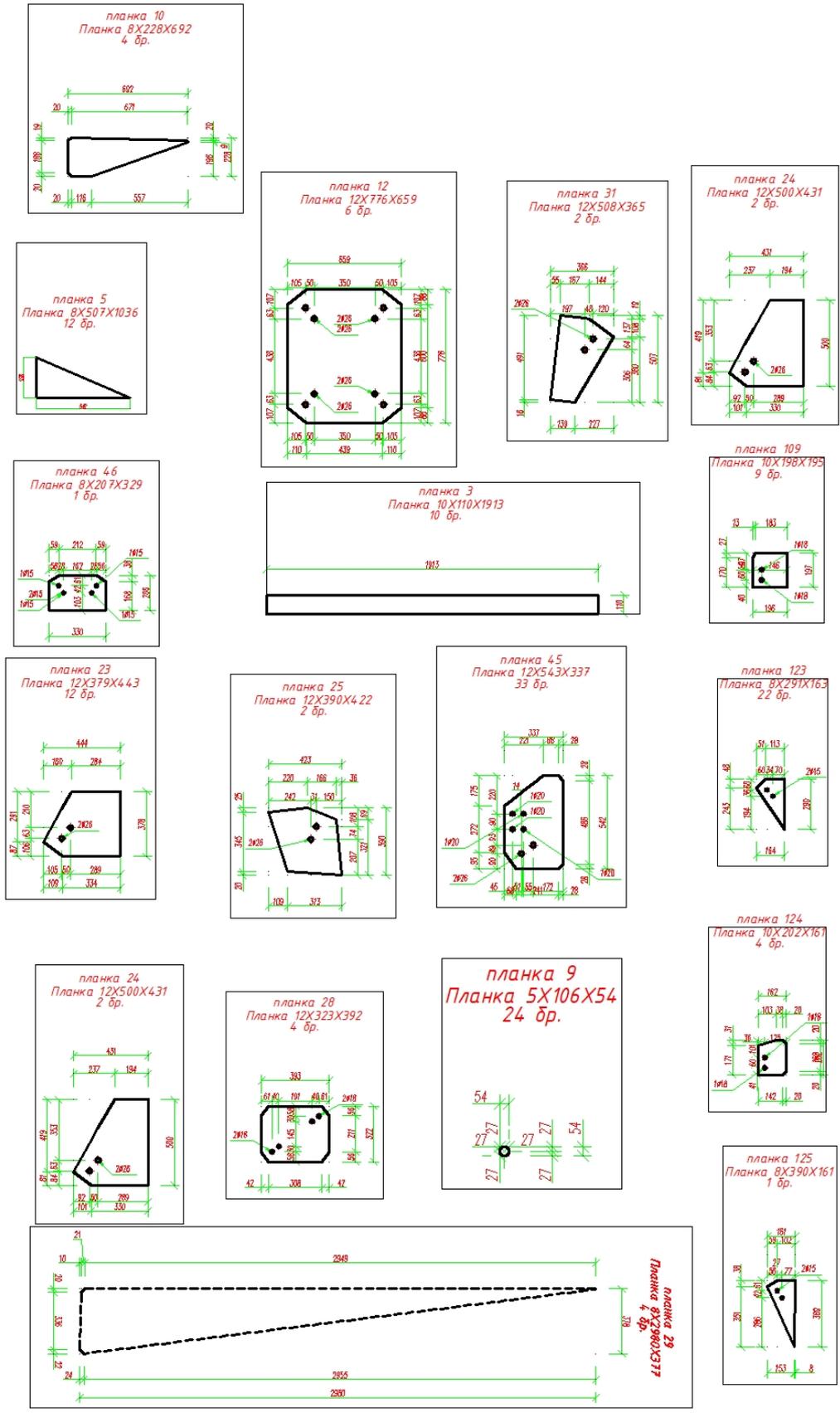
Този проблем е решен като входящите полигони са разширени с ивица с ширина $0.5cutW$ и всеки един сегмент е отместен извън полигона с $0.5cutW$.

4.6 Резултати при 2D разкрой. Примери.

В следващите страници ще илюстрираме разкрояване на следните входящи полигони (планки). Примерите са взети от реален строителен обект.

Полигонът за запълване ще бъде стандартен правоъгълен лист със широчина 1500мм. За входящите полигони ще покажем резултати с контурната линия на полигона отместена със широчината на ножа. Ширина на ножа 5мм. Размерите на планките са в милиметри. Преди да започнем изчисление на входящите полигони те минават през препроцесорна обработка, която включва:

1. Сортиране на планките по дебелина;
2. Прочитане на бройката им;
3. Намиране на контура на планките.



Фигура 4.2: Входящи полигони.

Сравняване на настоящия алгоритъм с комерсиален продукт.

За изработването на една стоманена конструкция от фигура 1.1 отнема около три месеца. Брой планки в една такава конструкция е около 1000. Брой уникални планки около 100. Броя итерации е относителен. Той зависи от това дали материалът е скъп или не е. Могат да се пуснат няколко итерации на различни компютри и да се вземе най-доброто от тях. Това е въпрос на решение на потребителя. При наличието на хардуер могат да се направят няколко итерации докато се получи приемлив отпадък.

В настоящето сравнение ще използваме планките дадени на фигура 4.4. Общия брой е 106. С тези планки е направено сравнение между комерсиалния продукт и разработения метод в настоящата дисертация. Намирането на крайното решение с представения метод е за една итерация. При по еднородни планки може да се направят повече итерации.

Използван комерсиален продукт.



Фигура 4.3: Комерсиален продукт FP Opti2D.

Съгласно сайта на производителя, продукта предлага следните функционалности:

1. User friendly graphical interface.
2. Handles panels, metal sheets and glazing.
3. Specific functions for aluminum composite panels.
4. Defines the parts to be optimized.
5. Directly uses the panels and glazing lists generated by FP PRO.
6. Imports work lists from Excel and from external calculation programs.
7. Manages the stock of full sheets and short bars.
8. Graphic display and printing of the optimized sheets, with clear indication of the cuts to be made and layout of the workpieces.
9. Provides statistical information on use of the sheets.

данни на оператора оптимизацията е продължила около 20 минути или 1200 сек. Размери на листа за запълване 1500мм / 6000мм. Време за работа не е посочено в разпечатката. Отпадък 18.2%. Всички планки са с отместен контур на 5mm. С този контур ще работим в изчисленията. Реалният отпадък може да се изчисли както следва:

1. Обща използвана площ - 2 листа x 1500mm x 6000mm = 18 000 000 ²;
2. Реална площ на всички 106 планки - 9 859 421 ²;

Реалната фира е 8 140 579 ². Или 45% фира. Реално запълване 54%.

Резултати от представения метод в настоящия труд. Брой планки 106. Всички планки са с отместен контур на 5мм от действителния им контур. Разрешено завъртане на планките : Да. Контур : *offsetPtL*.

Таблица 4.1: Сравнение на комерсиален продукт и представения алгоритъм

Включени параметри	Контур	Брой планки	Ratio	Време [s]
1	2	3	4	5
(a) Mirror:Yes, Rortate:Yes, Intervals:No	<i>box</i>	106	0.72	18 776
(b) Mirror:Yes, Rortate:Yes, Intervals:No	<i>Offset</i>	106	0.70	109 519
(c) Mirror:Yes, Rortate:Yes, Intervals:No	<i>Offset</i>	106	0.71	227 846
(d) Mirror:Yes, Rortate:Yes, Intervals:No	<i>Offset</i>	106	0.69	7 555
(e) Mirror:Yes, Rortate:Yes, Intervals:No	<i>Offset</i>	106	0.76	41 031
(f.1) Mirror:Yes, Rortate:Yes, Intervals:No	<i>box</i>	51	0.71(0.67)	2 194
(f.2) Mirror:Yes, Rortate:Yes, Intervals:No	<i>box</i>	55	0.57 (0.44)	2 454
(Комерсиален продукт)				
Mirror:N/A, Rortate:Yes, Intervals:N/A	<i>box</i>	106	0.54	1200

За случай *f.1* и *f.2* в скоби са дадени запълванията на планките спрямо основния лист 1500мм x 6000мм. Използвани са същите параметри както при комерсиалния продукт. Комерсиалният продукт има редица ограничения. Някои от тях са, че фигурите се апроксимират до правоъгълник, огледален образ на фигурите не се използва. Тъглите на завъртане са сведени до два: 0° и 90°. По отношение на коефициента на запълване *Ratio*, представения алгоритъм е много по-добър. виж таблица 4.1. Коефициентите са 0.71 за настоящия алгоритъм съпоставен с 0,67 за комерсиалния продукт. Второто сравнение е 0,57 за настоящия алгоритъм съпоставен с 0,44 за комерсиалния продукт. При големи обеми от работа или скъп материал, от които ще се изрязва разликата нараства още повече в полза на представения алгоритъм. Представеният алгоритъм в настоящата дисертация е по-добър от комерсиалния продукт, защото дава по-голям процент на уплътнение на фигурите. Друго негово предимство е много добрата му пригодност към паралелизация на изчисленията.

Глава 5

Заклучение

1D разкрой.

Както се вижда от сравнителната таблица 3.4 метода на мравките (*ACO*) дава най-добър резултат за много кратко време. В случая *ACO* метода проявява характер на *Greedy* алгоритъм. *ACO* алгоритъма е по-добър от комерсиалния продукт и по време и по оптимизиране. За големи обеми на разкрояване и компютри с по-слаби процесори *ACO* метода е много подходящ.

2D разкрой.

След проведените тестове на различни видове планки заключението е, че за приемливо решение на дадена задача са необходими по-голям на брой итерации. Резултатите в настоящата дисертацията са при 3 броя итерации. Приети са три броя итерации, защото намирането на едно решение отнема значително време. За някои видове планки този брой се оказва недостатъчен. Тестовите бяха направени на настолен компютър с операционна система Windows ®10 Pro, x64. Процесор Intel ®Core (TM) i5-9500@3.0 GHz. Използвани процесори един. Тип на процесора CPU. Въпреки, че процесорът е едно от последните поколения към дата на писане на настоящия труд се оказва слаб за по-висока степен на уплътняване на планките. Но ако се търси сравнително бързо подреждане и неголям брой планки настолният компютър може да справи. Представеният подход за решаване на проблема може да се приложи в 90% от случаите в практиката. Трябва да се отбележи, че за малко по-голяма плътност на решението е необходимо значително по-голямо време за изчисление. Дали ще се жертва време за сметка на материал зависи от това доколко е скъп даденият материал, от които ще се изрязват фигурите. Като следващо развитие на проблема ще бъде разработването му за хардуер с достатъчни изчислителни ресурси базиран на GPU процесори. Резултатите от настоящата дисертация са докладвани на различни национални и международни конференции.

5.1 Списък на публикациите

1. Evtimov G. Fidanova S. "**Subtraction of Two 2D Polygons with Some Matching Vertices**", pp. 80-87, *Numerical Methods and Applications, 9th International Conference NM&A'18, Borovets, Bulgaria, 2018*, Geno Nikolov, Natalia Kolkovska, Krassimir Georgiev (eds.), ISBN 978-3-030-10691-1, doi:[10.1007/978-3-030-10692-8_9](https://doi.org/10.1007/978-3-030-10692-8_9)
2. Evtimov G. Fidanova S. "**Heuristic Algorithm for 2D Cutting Stock Problem**", pp.350-357 *Large-Scale Scientific Computing, 11th International Conference, LSSC 2017, Sozopol, Bulgaria*, Ivan Lirkov, Svetozar Margenov (eds.), Springer, Vol. 793, ISBN 978-3-319-73440-8, doi:[10.1007/978-3-319-73441-5_37](https://doi.org/10.1007/978-3-319-73441-5_37)
3. Evtimov G. Fidanova S. "**Analyses and Boolean Operation of 2D Polygons**", pp. 107-118, *Advanced Computing in Industrial Mathematics, BGSIAM 2017*, K. Georgiev, I. Georgiev (eds.), Springer, Vol. 793, ISBN 978-3-319-97276-3, doi:[10.1007/978-3-319-97277-0_9](https://doi.org/10.1007/978-3-319-97277-0_9)
4. Evtimov G. Fidanova S. "**2D Optimal Cutting Problem**", pp. 33-40, *Advanced Computing in Industrial Mathematic, BGSIAM 2016*, K. Georgiev, I. Georgiev (eds.), Springer, Vol. 728, ISBN 978-3-319-65529-1, doi:[10.1007/978-3-319-65530-7_4](https://doi.org/10.1007/978-3-319-65530-7_4)
5. Ana Avdzhieva, Todor Balabanov, Georgi Evtimov, Ivan Jordanov, Nikolai Kitanov, Nadia Zlateva, **Two Dimensional Optimal Cutting Problem, 120th European Study Group with Industry (ESGI'120)**, Problems & Final Reports
6. V. Bodurov, D. Dimov, G. Evtimov, I. Georgiev, S. Harizanov, G. Nikolov, V. Pirinski, **The 2D/3D Best-Fit Problem, 113th European Study Group with Industry (ESGI'113)**, Problems & Final Reports, pp. 62-73, 2015. ISBN:978-619-7223-12-5
7. A. Avdzhieva, T. Balabanov, G. Evtimov, D. Kirova, H. Kostadinov, T. Tsachev, S. Zhelezova, N. Zlateva, **Optimal Cutting Problem, 113th European Study Group with Industry (ESGI'113)**, Problems & Final Reports, pp. 49-61, 2015. ISBN:978-619-7223-12-5

5.2 Аprobация на резултатите

Резултатите в настоящия дисертационен труд са докладвани на различни мероприятия на секция "Паралелни алгоритми" към ИИКТ-БАН като:

1. 113th European Study Group with Industry (BGSIAM - 2015);
2. 11th Annual Meeting of the Bulgarian Section of SIAM (BGSIAM - 2016);
3. 120th European Study Group with Industry (ESGI'120 - 2016);
4. 12th Annual Meeting of the Bulgarian Section of SIAM (BGSIAM - 2017) ;
5. 13th Annual Meeting of the Bulgarian Section of SIAM (BGSIAM - 2018);
6. Conference on Large-Scale Scientific Computations LSSC'17, Sozopol, 2017;
7. Ninth International Conference on Numerical Methods and Applications NM&A'18, Borovets.

5.3 Приноси

Приносите в тази дисертация могат да бъдат разделени на научни и научно-приложни, като научните приноси касят разработването на методи и алгоритми за 1D и 2D разкрой, а научно-приложните се отнасят към тяхната програмна реализация.

Научните приноси са:

- Разработен е алгоритъм за оптимален разкрой в едномерното пространство;
- Разработен е алгоритъм за оптимален разкрой в двумерното пространство;
- Разработен е метод за двумерен разкрой на базата на хибридна оптимизация;

Научно-приложните приноси са:

- Направена е програмна реализация на алгоритъма за едномерен разкрой;
- Направена е програмна реализация на алгоритъма за двумерен разкрой;

Резултатите на настоящия дисертационен труд могат да се използват в най-различни области от науката и инженерната практика:

- Проектирането на сгради и съоръжения;
- Проектирането на износването на детайли при машините както и проектирането на механизми;
- Земна механика - консолидация на почвите;
- Авиационната техника - намиране на оптимален път в среда с препятствия;
- И в много други области, където се използват CAD-системи.

Приложните приноси могат да се развият и във фирми които произвеждат стоманени конструкции. Приложния софтуер може да се внедри и в други отрасли които не са свързани със строителството на сгради и съоръжения. Друго много голямо приложно предимство е, че входните данни се вземат директно от базата данни на CAD системата, с която е проектирано съоръжението. Това многократно повишава скоростта на получаване и точността на данните, с които работи програмата. С няколко кликания могат да се изберат хиляди полигони и да стартира програмата за разкрой. Софтуерът може сам да отстрани или коригира "неправилните" полигони за да се избегнат неточности в изходните резултати. Решението протича в рамките на няколко минути в зависимост от производителността на компютърната система, на която софтуерът се използва. Алгоритъмът, разработен в представения дисертационен труд, позволява използване на огледален образ на полигоните, ротация и други операции, които могат да доведат до съществено подобрене на полученото приближено решение. Разбира се, за целите на широко-машабни научни изследвания алгоритъмът може да се внедри на супер-компютър тъй като позволява значителна паралелизация на изчисленията.

5.4 Декларация за оригиналност

Декларация за оригиналност на резултатите

Декларирам, че настоящата дисертация съдържа оригинални резултати, получени при проведени от мен научни изследвания с подкрепата и съдействието на научния ми ръководител. Резултатите, които са получени, описани и/или публикувани от други учени, са надлежно и подробно цитирани в библиографията.

Настоящата дисертация не е прилагана за придобиване на научна степен в друго висше училище, университет или научен институт.

Подпис:

5.5 Благодарности

Издавам голяма благодарност на моя научен ръководител проф. Стефка Фиданова за помощта и ценните съвети, за интерпретацията на резултатите от решенията и за цялостното ръководство по време на работата над дисертацията и особено за въвеждането на метаевристичните методи в тематиката на автоматичния разкрий.

Издавам специална благодарност на проф. Райчо Лазаров за помощта при работата ми върху статията [12]. В тази работа беше направена липсващата брънка в технологията за приближено решаване на тази сложна оптимизационна задача. Без тази статия решаването на задачата щеше да бъде много трудно.

Благодаря на доц. Иван Георгиев и доц. Станислав Харизанов за помощта при интерпретирането на някои математически модели. Благодаря на д-р Тодор Балабанов за съветите по време на писането на настоящата дисертация.

Благодаря на проф. Стефка Димова, която ме въведе в тази тематика чрез организацията на 113-та и 120-та European Study Group with Industry в град София; Благодаря на чл. кор. Светослав Маргенов и на целия екип от Института по Информационни и Комуникационни Технологии при Българска Академия на Науките за доверието и подкрепата по време на работата ми върху дисертацията.

Библиография

- [1] Valerio de Carvalho J.M. "**Lp models for bin packing and cutting stock problems**", European Journal of Operational Research 141:253–273,(2002).
- [2] Chen C.L.S., Hart S.M., Tham W.M. "**A simulated annealing heuristic for the one-dimensional cutting stock problem**", European Journal of Operational Research 93:522–535,(1996).
- [3] Foerster H., Wscher G. "**Pattern reduction in one-dimensional cutting stock problems**", In: Proceedings of the 15th Triennial Conference of the International Federation of Operational Research Societies, (1999).
- [4] Falkenauer E., "**A hybrid grouping genetic algorithm for bin packing**", Journal of Heuristics Vol. 2, 1996
- [5] Song X., Chu C.B., Nie Y.Y., Bennell J.A. "**An iterative sequential heuristic procedure to a real-life 1.5-dimensional cutting stock problem**". European Journal of Operational Research 175:1870–1889, (2006).
- [6] Suliman S.M.A. "**Pattern generating procedure for the cutting stock problem**", Int J Production Economics 74:293–301, (2001).
- [7] Vahrenkamp R. "**Random search in the one-dimensional cutting stock problem**", European Journal of Operational Research 95:191–200, (1996).
- [8] Vanderbeck F. "**Exact algorithm for minimizing the number of setups in the one-dimensional cutting stock problem**", Operations Research 48:915–926, (2000).
- [9] O'Rourke J., "**Computational Geometry in C second edition**", ISBN 0 521 64976 5, <http://www.cambridge.org>
- [10] de Berg M., van Kreveld M., Overmars M., Schwarzkopf O.C. "**Computational Geometry: Algorithms and Applications, Second Edition**", ISBN 3-540-65620-0
- [11] Graham, R.L. (1972). "**An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set**", Information Processing Letters. 1 (4): 132–133. doi:10.1016/0020-0190(72)90045-2 , https://en.wikipedia.org/wiki/Graham_scan
- [12] Evtimov G., Fidanova S. "**Subtraction of Two 2D Polygons with Some Matching Vertices**", pp. 80-87, Numerical Methods and Applications, 9th International Conference NM&A'18, Borovets, Bulgaria, 2018, ISBN 978-3-030-10691-1, doi:10.1007/978-3-030-10692-8_9

-
- [13] Evtimov G., Fidanova S. "**Heuristic Algorithm for 2D Cutting Stock Problem**", pp.350-357, Large-Scale Scientific Computing, 11th International Conference, LSSC 2017, Sozopol, Bulgaria, Springer, Vol. 793, ISBN 978-3-319-73440-8, doi:[10.1007/978-3-319-73441-5_37](https://doi.org/10.1007/978-3-319-73441-5_37)
- [14] Evtimov G., Fidanova S. "**Analyses and Boolean Operation of 2D Polygons**", pp. 107-118, Advanced Computing in Industrial Mathematics, BGSIAM 2017, Springer, Vol. 793, ISBN 978-3-319-97276-3, doi:[10.1007/978-3-319-97277-0_9](https://doi.org/10.1007/978-3-319-97277-0_9)
- [15] Evtimov G. Fidanova S. "**2D Optimal Cutting Problem**", pp. 33-40, *Advanced Computing in Industrial Mathematic, BGSIAM 2016*, Springer, Vol. 728, ISBN 978-3-319-65529-1, doi:[10.1007/978-3-319-65530-7_4](https://doi.org/10.1007/978-3-319-65530-7_4)
- [16] Evtimov G., Fidanova S. "**Ant Colony optimization algorithm for 1D Cutting Stock Problem**", pp. 24, *Advanced Computing in Industrial Mathematic, BGSIAM 2016*
- [17] Fidanova S. "**ACO Algorithm with Edditional Reinforcement, From Ant Colonies to Artificial Ants**", *Lecture Notes in Computer Science, N2463, Spinger, Germany, 2002,292-293*
- [18] Fidanova S., Marinov P., Alba E. "**ACO for Optimal Sensor Layout, In Proc. of Int. Conf. on Evolutionary Computing, Valencia, Spain, Joaquim Filipe and Janus Kacprzyk eds.**", *SciTePress-science and Tchnology Publications Portugal, ISBN 978-989-8425-31-7, 2010, 5-9*
- [19] Fidanova S., Marinov P., Alba E. "**Wireless Sensor Network layout, In Monte Carlo Methods and aplications**", *K. Sabelfeld, I. Dimov eds., Chapter 9, De Gruyter Pub, Berlin, germany, 2012, 39-46*
- [20] Fidanova S., Shindarov M., Marinov P. "**Mono-objective algorithm for Wireless Sensor Network layout, In Proc of OMKO-NET Int.**", *Conference, Southampton, UK, 2012a, 57-63*
- [21] Fidanova S., Shindarov M., Marinov P. "**Optimal Sensor Layou Using Multi-objective Metaheuristic, In Proc of OMKO-NET Int.**", *Of Int. Conference of information systems and Grid Technologies, Sofia, Bulgaria, 2011, 114-122*
- [22] Fidanova S., Shindarov M., Marinov P. "**Multi-Objective ant algorithm for Wireless Sensor Network Positioning.**", *Proceedings of Bulgarian academy of Science, Vol 66(3), 2013, 353-360.*
- [23] Fidanova S., Shindarov M., Marinov P. "**Wireless Sensor Positioning ACO Algorithm.**", *Studies of Computational intelligence, J.Kacprzyk and K. atanassov eds., Spinger, Germany*
- [24] Gonalves J.F., "**A hybrid genetic algorithm-heuristic for a two-dimensional or-thogonal packing problem**" *Eur J Oper Res*, Vol 183, No 3, 2007, 1212-1229.
- [25] V. Bodurov, D. Dimov, G. Evtimov, I. Georgiev, S. Harizanov, G. Nikolov, V. Pirinski, **The 2D/3D Best-Fit Problem, 113th European Study Group with Industry (ESGI'113)**, Problems & Final Reports, pp. 62-73, 2015. ISBN:978-619-7223-12-5

-
- [26] A. Avdzhieva, T. Balabanov, G. Evtimov, D. Kirova, H. Kostadinov, T. Tsachev, S. Zhelezova, N. Zlateva, "**Optimal Cutting Problem, 113th European Study Group with Industry (ESGI'113)**", Problems & Final Reports, pp. 49-61, 2015. ISBN:978-619-7223-12-5
- [27] Боровска П., "**Синтез и анализ на паралелни алгоритми**", ISBN:978-954-438-764-4
- [28] https://www.sit.ac.jp/user/konishi/JPN/Tech_inform/Pdf/GeometricalMoment.pdf
- [29] Р. Даскалов, Е. Даскалова, "**Висша математика, част I, Аналитична геометрия**", Габрово, 2012
- [30] Antonio, Franklin "**Chapter IV.6: Faster Line Segment Intersection**", In Kirk, David (ed.). Graphics Gems III. Academic Press, Inc. pp. 199–202. ISBN 0-12-059756-X, (1992).
- [31] "Weisstein, Eric W. "**Line-Line Intersection. From MathWorld**", A Wolfram Web Resource. Retrieved 2008-01-10.
- [32] Shimrat, M., "**Algorithm 112: Position of point relative to polygon**", 1962, Communications of the ACM Volume 5 Issue 8, Aug. 1962
- [33] Eric Haines, "**Point in Polygon Strategies**" in Graphics Gems IV", (1994) http://geomalgorithms.com/a03-_inclusion.html
- [34] https://bg.wikipedia.org/wiki/\T2A\CYRM\T2A\cyre\T2A\cyrt\T2A\cyra\T2A\cyre\T2A\cyrv\T2A\cyrr\T2A\cyri\T2A\cyrs\T2A\cyrt\T2A\cyri\T2A\cyrch\T2A\cyrn\T2A\cyri_\T2A\cyra\T2A\cyrl\T2A\cyrg\T2A\cyro\T2A\cyrr\T2A\cyri\T2A\cyrt\T2A\cym\T2A\cyri
- [35] Glover F., "**Future paths for integer programming and links to artificial intelligence**", Computers and Operations Research,13,533-549 (1986).
- [36] Glover F. and Laguna M., "**Tabu Search**", Kluwer, Boston, (1997).
- [37] Balaban, I. J. "**An optimal algorithm for finding segments intersections**", Proc. 11th ACM Symp. Computational Geometry, pp. 211–219, doi:10.1145/220279.220302, (1995).
- [38] Bartuschka, U.; Mehlhorn, K.; Näher, S. "**A robust and efficient implementation of a sweep line algorithm for the straight line segment intersection problem**", in Italiano, G. F.; Orlando, S. (eds.), Proc. Worksh. Algorithm Engineering, (1997).
- [39] Bentley, J. L.; Ottmann, T. A. "**Algorithms for reporting and counting geometric intersections**", IEEE Transactions on Computers, C-28 (9): 643–647, doi:10.1109/TC.1979.1675432, (1979).
- [40] "**Point in Polygon, One More Time...**", Archived 2018-05-24 at the Wayback Machine, Ray Tracing News, vol. 3 no. 4, October 1, 1990.
- [41] <http://www.theswamp.org/index.php?topic=1891.0>

-
- [42] Boissonat, J.-D.; Preparata, F. P. "**Robust plane sweep for intersecting segments**", (PDF), SIAM Journal on Computing, 29 (5): 1401–1421, doi:[10.1137/S0097539797329373](https://doi.org/10.1137/S0097539797329373), (2000).
- [43] Brown, K.Q. "**Comments on "Algorithms for Reporting and Counting Geometric Intersections"**", IEEE Transactions on Computers, C-30 (2): 147, doi:[10.1109/tc.1981.6312179](https://doi.org/10.1109/tc.1981.6312179), (1981).
- [44] Chazelle, Bernard; Edelsbrunner, Herbert (1992), "**An optimal algorithm for intersecting line segments in the plane**", Journal of the ACM, 39 (1): 1–54, doi:[10.1145/147508.147511](https://doi.org/10.1145/147508.147511)
- [45] Alvarez-Valdes R, Parajon A, Tamarit J. M, "**A computational study of heuristicalgorithms for two-dimensional cutting stock problems**", 4th metaheuristics inter-national conference (MIC2001), 2001, 16-20.
- [46] Chen, E.Y.; Chan, T.M. "**A space-efficient algorithm for segment intersection**", Proc. 15th Canadian Conference on Computational Geometry (PDF), (2003).
- [47] Clarkson, K.L. "**Applications of random sampling in computational geometry**", II Proc. 4th ACM Symp. Computational Geometry, pp. 1–11, doi:[10.1145/73393.73394](https://doi.org/10.1145/73393.73394), (1988).
- [48] Eppstein, D.; Goodrich, M.; Strash, D. "**Linear-time algorithms for geometric graphs with sublinearly many crossings**", Proc. 20th ACM-SIAM Symp. Discrete Algorithms (SODA 2009), pp. 150–159, (2009).
- [49] Cintra G., Miyazawa F., Wakabayashi Y., Xavier E., "**Algorithms for two-dimensional cutting stock and strip packing problems using dynamic programming and column generation**", European Journal of Operational Research, European Journal of Operational Research, Vol. 191, 2008,61-85.
- [50] Mulmuley, K. "**A fast planar partition algorithm**", I Proc. 29th IEEE Symp. Foundations of Computer Science (FOCS 1988), pp. 580–589, doi:[10.1109/SFCS.1988.2197](https://doi.org/10.1109/SFCS.1988.2197), (1988).
- [51] Preparata, F. P.; Shamos, M. I. "**Section 7.2.3: Intersection of line segments**", Computational Geometry: An Introduction, Springer-Verlag, pp. 278–287, (1985).
- [52] Shamos, M. I.; Hoey, Dan "**Geometric intersection problems**", 17th IEEE Conf. Foundations of Computer Science (FOCS 1976), pp. 208–215, doi:[10.1109/SFCS.1976.16](https://doi.org/10.1109/SFCS.1976.16), (1976)
- [53] Dorigo M., Manieezz M., mColorni A., "**The ant syste: Optimization by a colony of cooperating agents**", IEEE Transactions on Systems, Man and Cybernetics B, Vol 26(1), 1996, pp 29-41
- [54] Dorigo M., "**Optimization, Learning and Natural Algorithms**", PhD thesis, Politecnico di Milano, Italie, 1992.
- [55] Lodi, A., Martello S., Vigo D., "**Recent advances on two-dimensional bin packingproblems**", Discrete Applied Mathematics, Vol. 123, 2002, 379-396.

-
- [56] Falkenauer E., "**A hybrid grouping genetic algorithm for bin packing**", *Journal of Heuristics*, Vol. 2, 1996, pp. 5-30
- [57] Hinterding R., Khan L., "**Genetic algorithms for cutting stock problems: with and without countiguity**", In: X. Yao, editor, *Progress in Evolutionary Computation*, Berlin, Germany, Springer, 1995, pp. 166-186
- [58] Jaromi M.H., Tavakkoli-Moghaddam, R., Makui, A. Shamsi A., "**Solving an one-dimensional cutting stock problem by simulated and tabu search**", *J. of Industrial Engineering International*, Vol. 8 (1), Springer, 2012, pp. 24
- [59] Reeves C., "**Hybrid genetic algorithms for bin-packing and related problems**", *Annals of Opera Research* 63, 1996, pp.371-396
- [60] Vink M., **Solving combinatorial problems using evolutionary algorithms**, 1997 <http://citeseer.nj.nec.com/vink97solving.html>
- [61] Parmar K., Prajapati H., Dabhi V., "**Cutting stock problem: A survey of evolutionary computing based solution**", In *Proc. of Green Computing Communication and Electrical Engineering*, 2014, doi:[10.1109/ICGCCEE.2014.6921411](https://doi.org/10.1109/ICGCCEE.2014.6921411).
- [62] Dusberger, F., Raidl, G.R., "**A variable neighborhood search using very large neighborhood structures for the 3-staged 2-dimensional cutting stock problem**", In: Blesa, M.J., Blum, C., Voß, S. (eds.) *HM 2014. LNCS*, vol. 8457, pp. 85–99. Springer, Cham (2014). doi:[10.1007/978-3-319-07644-7_7](https://doi.org/10.1007/978-3-319-07644-7_7)
- [63] Dusberger, F., Raidl, G.R., "**Solving the 3-staged 2-dimensional cutting stock problem by dynamic programming and variable neighborhood search.**", *Electron. Notes Discret. Math.* 47, 133–140 (2015) *MathSciNetCrossRefzbMATH Google Scholar*
- [64] Lin Z., Li Y., "**An Efficient Algorithm for Intersection, Union and Difference between Two Polygons**", *In proc. of Computer Network and Multimedia Technology, Wuhan, China, 2009*, 1-4
- [65] Gonçalves J.F., "**A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem**", *European Journal of Operational Research*, Vol 183, No 3, 2007, 1212-1229.
- [66] Alvarez-Valdes R., Parreno F., Tamarit J.M., "**A Tabu Search algorithm for two dimensional non-guillotine cutting problems**", *European Journal of Operational Research* , Vol.183(3), 2007, 1167-1182.
- [67] Alvarez-Valdes R., Parajon, A., Tamarit, J.M. "**A computational study of heuristic algorithms for two-dimensional cutting stock problems**", In: 4th *Metaheuristics International Conference (MIC 2001)*, pp. 16–20 (2001)
- [68] Lodi A., Martello S., Vigo D., "**Recent advances on two-dimensional bin packing problems**", *Discrete Applied Mathematics*, Vol. 123, 2002, 379-396.
- [69] Delorme M., M. Iori, S. Martello, "**Bin packing and Cutting Stock Problems: Mathematical models and exact algorithms**", *European Journal of Operational Research* 2016, 255, 1–20, doi:[10.1016/j.ejor.2016.04.030](https://doi.org/10.1016/j.ejor.2016.04.030)

-
- [70] Wäscher, G.; Haußner, H.; Schumann, H. "**An Improved Typology of Cutting and Packing Problems**", European Journal of Operational Research Volume 183, Issue 3, 1109-1130
- [71] M.P. Johnson, C. Rennick and E. Zak "**Skiving addition to the cutting stock problem in the paper industry**", SIAM Review, 472-483, (1997).
- [72] Raffensperger, J. F. "**The generalized assortment and best cutting stock length problems**", International Transactions in Operational Research. 17: 35–49, doi:[10.1111/j.1475-3995.2009.00724.x](https://doi.org/10.1111/j.1475-3995.2009.00724.x), (2010).
- [73] Kantorovich, V.L., "**Mathematical methods of organizing and planning production**", Leningrad State University. 1939
- [74] Kantorovich L.V. and Zalgaller V.A. "**Calculation of Rational Cutting of Stock**", Lenizdat, Leningrad, (1951)
- [75] Gilmore P.C., R.E. Gomory "**A linear programming approach to the cutting-stock problem**", Operations Research 9: 849-859, (1961).
- [76] Gilmore P.C., R.E. Gomory "**A linear programming approach to the cutting-stock problem - Part II**", Operations Research 11: 863-888, (1963).
- [77] Gradiar M., Kljaji M., Resinovi, G., Jesenko J. "**A sequential heuristic procedure for one-dimensional cutting**", European Journal of Operational Research. 114. 3, 557-568, (1999).
- [78] Gradisar M., Resinovic G., Kljajic, M. , "**A hybrid approach for optimization of one-dimensional cutting**", European Journal of Operational Research. 119. 3, 719-728, (1999).
- [79] Gradisar M., Resinovic G., Kljajic M. (2002) "**Evaluation of algorithms for one-dimensional cutting**", Computers and Operations Research 29:1207–1220
- [80] Dyckhoff H., "**A typology of cutting and packing problems**", European Journal of Operational Research. 44, 145-159,(1990).
- [81] Vance P., Barnhart, C., Johnson, E.L., Nemhauser, G.L. "**Solving binary cutting stock problems by column generation and branch-and-bound**", Computational Optimization and Applications. 3. 111-130,(1994).
- [82] Vance P., "**Branch-and-price algorithms for the one-dimensional cutting stock problem**", Computational Optimization and Applications. 9, 211-228,(1998).
- [83] Goulimis C. "**Optimal solutions for the cutting stock problem**", European Journal of Operational Research 44: 197-208, (1990)
- [84] de Carvalho V. "**Exact solution of cutting stock problems using column generation and branch-and-bound**", International Transactions in Operational Research 5: 35–44, (1998)
- [85] Berberler M.E., Nuriyev U.G., "**A New Heuristic Algorithm for the One-Dimensional Cutting Stock Problem**". Appl. Compu. Math. 9.1, 19-30,(2010).

-
- [86] Diegel A., E. Montocchio, E. Walters, S. van Schalkwyk and S. Naidoo (1996), "**Setup minimizing conditions in the trim loss problem**", European Journal of Operational Research 95:631-640
- [87] McDiarmid C., "**Pattern Minimisation in Cutting Stock Problems**", Discrete Applied Mathematics, 121-130, (1999)
- [88] Garey M.R., Johnson, D.S., "**Computers and Intractability: A Guide to the Theory of NP-Completeness**", WH Freeman, New York (1979).
- [89] Kantorovich L.V., "**Mathematical methods of organizing and planning production**", Management Science 6.4, 366-422,(1960).
- [90] Jahromi, M.H., Tavakkoli-Moghaddam, R., Makui, A. Shamsi A. "**Solving an one-dimensional cutting stock problem by simulated annealing and tabu search**", Journal of Industrial Engineering International, Vol. 8(1), Springer, 2012, paper 24.
- [91] Jahromi, M.H., Tavakkoli-Moghaddam, R., Makui, A., Shamsi, A., "**Solving an one dimensionalcutting stock problem by simulated annealing and tabu search**", Journal of Industrial Engineering International. (2012).
- [92] "**Constantine Goulimis. Counterexamples in the CSP**", arXiv:2004.01937
- [93] Maria Garcia de la Banda, P. J. Stuckey., "**Dynamic Programming to Minimize the Maximum Number of Open Stacks**", INFORMS Journal on Computing, Vol. 19, No. 4, Fall 2007, 607-617.
- [94] Chvátal, V. "**Linear Programming**", W.H. Freeman. ISBN 978-0-7167-1587-0, (1983).
- [95] Cherri, A.C., Arenales, M.N., Yanasse, H.H., "**The one-dimensional cutting stock problems with usable leftover: a heuristic approach**", European Journal of Operational Research. 196.3, 897-908,(2009).
- [96] Cherri, A.C., Arenales, M.N., Yanasse, H.H., "**The usable leftover one-dimensional cutting stock problem-a priority-in-use heuristic**", Intl. Trans. in Op. Res. 20, 189-199,(2013).
- [97] Valerio de Carvalho, J.M., "**Exact solution of bin-packing problems using column generation and branch-and-bound**", Annals of Operation Research. 86, 629-659, (1999).
- [98] Hatem Ben Amor, J.M. Valério de Carvalho, "**Cutting Stock Problems in Column Generation**", edited by Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, Springer, 2005, XVI, ISBN 0-387-25485-4.
- [99] Waescher, G., Gau, T., "**Heuristics for the Integer one-dimensional Cutting Stock Problem**", A Computational Study. OR Spektrum18. 3, 131-144,(1996).
- [100] Vanderbeck, F., "**Computational study of a column generation algorithm for binpacking and cutting stock problems**", Mathematical Programming A. 86, 565-594, (1999).

-
- [101] Vanderbeck, F., **"On DantzigWolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm"**, Operations Research. 48, 111-128,(2000).
- [102] Mobasher, A., Ekici A., **"Solution approaches for the cutting stock problem with setup cost"**, Computers and Operations Research. 40, 225-235,(2013).
- [103] Johnston, R.E., Sadinlija, E., **"A new model for complete solutions to one-dimensional stock problems"**, European Journal of Operational Research. 153, 176-183,(2004).
- [104] Johnston, R.E. **"Rounding algorithm for cutting stock problems"**, Journal of Asian-Pacific Operations Research societies 3:166–171, (1986).
- [105] Umetani, S., Yagiura, M., Ibaraki, T., **"One-dimensional cutting stock problem to minimize the number of different patterns"**, European Journal of Operational Research. 146, 388-402,(2003).
- [106] S. Umetani, M. Yagiura, and T. Ibaraki **"One dimensional cutting stock problem to minimize the number of different patterns"**, European Journal of Operational Research 146, 388–402, (2003)
- [107] Scheithauer, G., Terno, J., Muller, A., Belov, G., **"Solving one-dimensional cutting stock problems exactly with a cutting plane algorithm"**, Journal of the Operational Research Society. 52, 1390-1401,(2001).
- [108] Belov, G., Scheithauer, G., **"A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting"**, European Journal of Operational Research. 171, 85-106,(2006).
- [109] Belov, G., Scheithauer, G., **"A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths"**, European Journal of Operational Research. 141, 274-294,(2002).
- [110] Mukhacheva, E.A., Belov, G., Kartak, V., Mukhacheva, A. S., **"One-dimensional cutting stock problem: Numerical experiments with the sequential value correction method and a modified branch-and-bound method"**, Pesquisa Operacional. 2000.2, 153-168,(2001).
- [111] Belov G., Scheithauer G. **"Setup and open stacks minimization in one-dimensional stock cutting"**, INFORMS Journal of Computing 19(1):27–35, (2007).
- [112] Belov G., Scheithauer G. **"The number of setups (different patterns) in one-dimensional stock cutting"**, Technical Report, Desden University, (2003).
- [113] Belov G., Scheithauer G. **"A branch-and-cut-and-price algorithm for one-dimensional stock cutting and two-dimensional two-stage cutting"**, European Journal of Operational Research 171:85–106, (2006).
- [114] Dikili A.C., Sarz E., PekN. A., **"A successive elimination method for one-dimensional stock cutting problems in ship production"**, Ocean engineering. 34.13, 1841-1849, (2007).