БЪЛГАРСКА АКАДЕМИЯ НА НАУКИТЕ
ИНСТИТУТ ПО ИНФОРМАЦИОННИ И КОМУНИКАЦИОННИ ТЕХНОЛОГИИ

Метаевристични методи за решаване на задачи за разкрояване

Георги Евтимов Евтимов

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

Списък на фигурите

Списък на таблиците

List of Algorithms

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,.... A_n ..... Име на връх от полигон A.

4. B_1, B_2,.... B_n ..... Име на връх от полигон B.

5. pt_1, pt_2,.... 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,..... 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

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 ползуват 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.

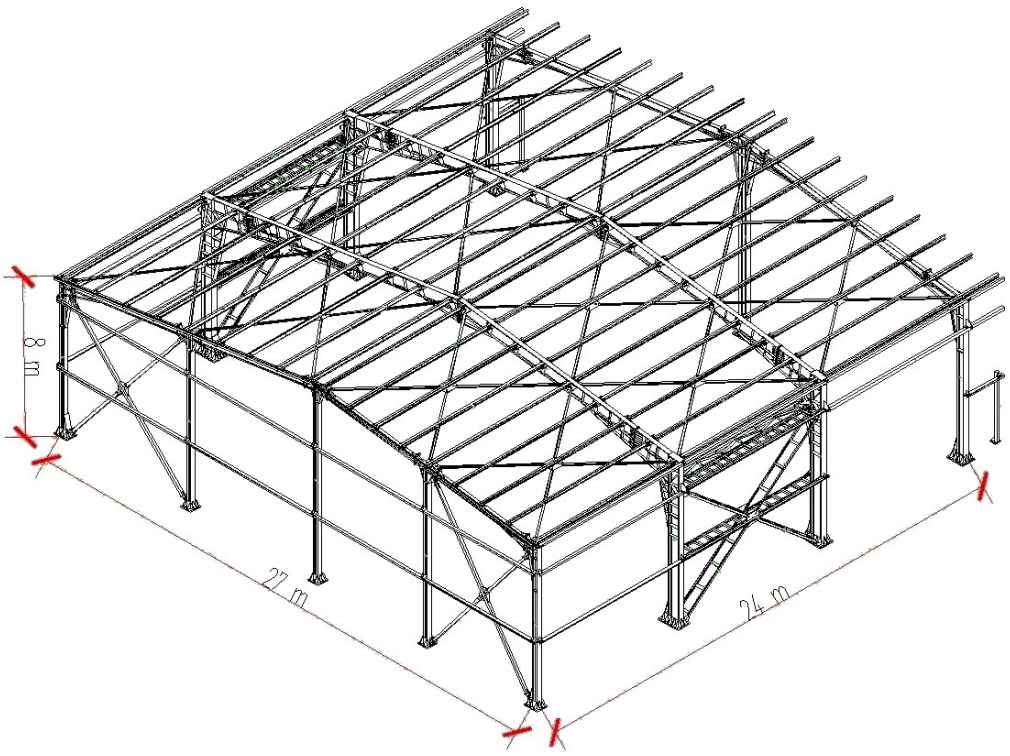
Широкото прилагане на ИТ в Европа и България създаде една динамична и високо конкурентна среда, в която фирма без внедряване на ИТ решения е често обречена на фалит. Необходимостта от съкращаване на производствени разходи е жизнено важна за оцеляването на предприятието. От друга страна изискванията на клиентите нараства, което допълнително подхранва необходимостта от бързи решения за оптимизация на материалните и човешки ресурси, което се постига ефективно с използуване 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, непринадлежаща 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 развитите 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 нуждите на строителен обект е необходимо да се разкроят определено число детайли (много често достигащ хиляди) с различни размери, форми, дебелини, а в някой случай 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 нуждите 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 използуването 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 използуване 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 образувани от прави по трите си страни отпадъка може да достигне до 50%. Този подход има ограничено приложение, но се използува доста масово и дава добри резултати в стъкларската 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 използуват масово CAD системи. В представената дисертация проблемът за оптимален разкрий на строителни елементи (или планки) е решен при предположението, че полигоните (планките) се генерират и предоставят на строителя от CAD система. След това се прави предварителна обработка на данните и накрая планките се разкрояват с точната им геометрия. Трябва да обърнем внимание, че представените тук инструменти "изчистват" полигоните от препокриване на точки или задаване на n-ъгълник с повече от n на брой точки по границата.

В този дисертационен труд е разгледан проблемът от гледна точка на стоманените конструкции в строителството. В основни линии там се поставят две основни задачи: Първата задача е разкрой на линейни профили, а втората е двумерен ($2D$) разкрой на листов материал. Това са стоманени планки, които се заварят към стоманените профили (с цел укрепването им) или се правят болтови връзки и/или други методи за съединяване. В някои отрасли формите, които трябва да изрязват са правоъгълници.

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 разкроя. Най-важните особености са:

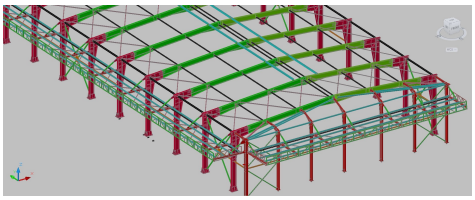
1. често планките са със сложни форми, чиито граници са произволни неса-

мопресичащи 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);

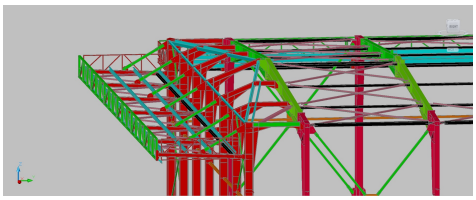
2. много често планката няма "лице" и "гръб", което позволява огледално търсене на местоположението ѝ в процеса на разкроя; тази особеност може да доведе до икономия на материал, но увеличава сложността на проблема.

3. в набора от планки, които трябва да бъдат произведени, много често има значително разнообразие на размерите, площите и формата.

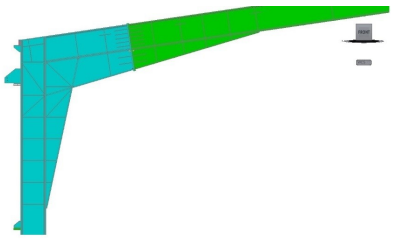
В работата ударението е поставено върху методи за решаване на задачата за разкрой на планки в $2D$. Двумерната задача за разкрой е по-трудна от едномерната, особено когато изрязваните фигури не са изпъкнали и са с неправилна форма. И двете задачи са NP-пълни комбинаторни оптимизационни задачи [12], [13].



Фигура 1.2: Стоманена конструкция 1.



Фигура 1.3: Стоманена конструкция 2.



Фигура 1.4: Корав възел при рамка.

Като илюстрация ще се представят някои примери за случая на стоманени конструкции. В стоманено хале (като такова показано на Фигури 1.2 - 1.4) 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 широчина (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 разкроя, 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 сравнвния 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.

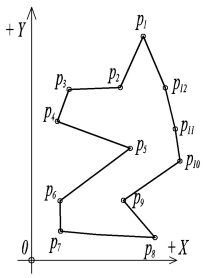
1.4 Подход на изследването

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.

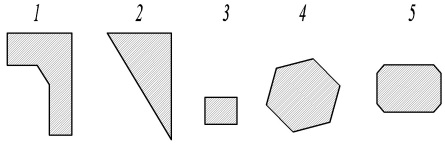
Първата задача е едномерен (линеен), $1D$ *разкрой*. За $1D$ оптимизацията не се въвеждат някакви специални дефиниции, тъй като се работи с един параметър, дължината на елемента. Задачата за минимален отпадък се свежда до намиране на минимален брой използвани профили. Въпреки, че е по-лесна от двумерната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

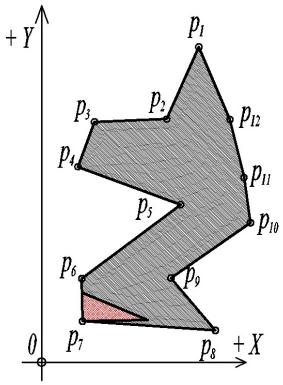
да се приложи алгоритъма за "изваждане" ("изрязване") на два полигона. Това се прави с цел за следващия входящ полигон да се търси местоположение в остатъка от основния полигон. След това тази (изрязана) планка се премахва от списъка с входящи планки. Това се повтаря докато се изчерпят всички планки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



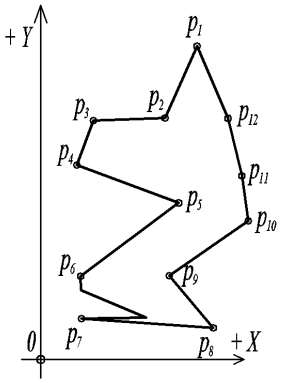
Фигура 1.5: Първоначален основен полигон (за запълване).



Фигура 1.6: Примерни входящи полигони (размерите са съществено увеличени)



Фигура 1.7: Основен полигон преди изрязването.



Фигура 1.8: 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 2.2. *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 използуване на комерсиален софтуер. Предимствата на създадената програма пред тествания комерсиален софтуер са дадени в 4.6 и в 5.

1.5 Структура на съдържанието

Дисертационният труд е разделен на увод, четири глави и заключение. Дисертацията съдържа:

1. 140 страници;
2. Фигури 142;
3. Таблици 142;
4. 117 Литературни източника.

По дисертационния труд са публикувани 6 публикации.
В глава 2 са дадени основните дефиниции в изчислителната геометрия. Избраните операции като ротация на точка, перпендикуляр към точка, пресичане на две прави, посока на построение на полигон, изпъкнала обвивка на множество точки ще бъдат необходими за намирането на едно решение на $2D$ задачата. В точка 2.5 "Пресичане на два полигона" е разработен и представен алгоритъм за намирането на сечение, обединение и изваждане на два полигона. Представения алгоритъм е един от възможните алгоритми за реализиране на логическите операции "Събиране", "Изваждане" и "Сечение". Алгоритъмът работи и за някои патологични случаи на разположение на два полигона.

В глава 3 е формулирана и решена задачата за едномерен ($1D$) разкрой. Тук задачата се свежда до оптимизиране при едно ограничение - сбора от n на брой дължини да бъде равен на дадено число. Използваме метода на мравките. В точка 3.3 е дадено сравнителното решение на три метода - Комерсиален продукт, $Greedy + n^3$ и метода на мравките ACO.

В глава 4 е формулирана задачата за $2D$ разкроя, намирането на едно възможно решение и окончателното решение с всички планки. За избора на местоположение на дадена планка се използва хибридна метаевристика която е описана в точка 4.2. В тази точка са описани основните понятия на хибридната метаевристика, дефинирани са параметрите за намиране на крайното приближено решение за $2D$ разкроя, което се заключава в избора на входящ полигон в дадения полигон за запълване. В точка 4.6 са представени резултатите от 2D разкроя. Като сравняване на настоящето изчисление и това получено с използване на комерсиален продукт. В резултатите са описани предимствата и недостатъците на използвания подход.

Глава 2

Изчислителна геометрия.

2.1 Основни дефиниции

В тази секция се използуват геометричните обекти точка, линеен сегмент и полигон в двумерната равнина. Всички точки ще бъдат представени като списък от наредени числа (координати), [9], в двумерния случаи 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.

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

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 използуващи 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.

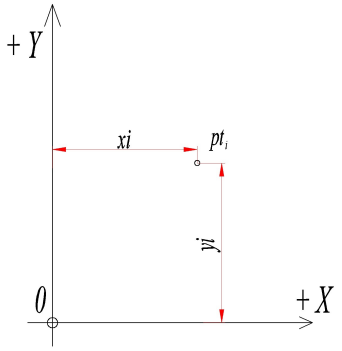
Тези правила са продиктувани от необходимостта да работим с данни произведени от CAD системи. В областта на проектиране на обекти изпълнени със стоманени конструкции, размерите на конструкциите се дават в милиметри, така че всички числени данни (координати на точки, сегменти и т.н.) са в милиметри. За 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:

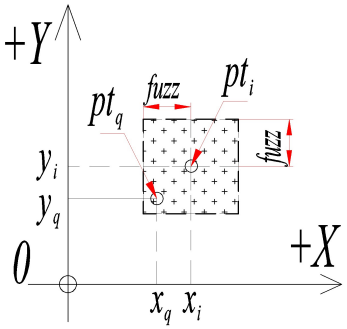
$$(x_i - fuzz) \leq x_q \leq (x_i + fuzz) \wedge (y_i - fuzz) \leq y_q \leq (y_i + fuzz) \tag{2.1}$$

или

$$max\{|x_i - x_q|, |y_i - y_q|\} \leq fuzz \tag{2.2}$$



Фигура 2.1: Точка



Фигура 2.2: Размита точка

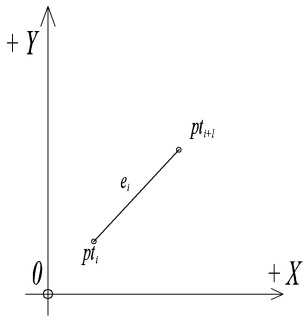
Дефиниция на линеен сегмент.

Ще използуваме два вида сегменти:

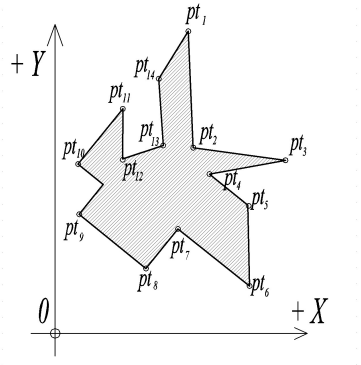
1. CAD линеен сегмент $CADe_i = list(pt_0, pt_1, \ldots, pt_i)$ се задава със списък от точки, които лежат на една права; такива сегменти се получават от работата на CAD системата, която генерира всички входни данни използувани в тази работа.

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 CAD сегмента.



Фигура 2.3: Сегмент



Фигура 2.4: Полигон

Дефиницията на полигон.

Полигон $\Pi = list\ (e_0, e_1, \ldots, e_{n-1})$ е затворена област от равнината оградена от n линейни сегменти образуващи затворена крива, [9]. Обръщаме внимание, че тук използваме линеен сегмент, а не CAD линеен сегмент. Нека $pt_0, pt_1, \ldots, pt_n$ са n точки от дадена равнина, такива, че $pt_0 = pt_n$. Точките $pt_0, pt_1, \ldots, pt_n$ 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, $\Pi = list\ (pt_0,\ pt_1,\ldots,\ pt_n)$. Казваме, че два сегмента са съседни когато имат само една обща крайна точка.

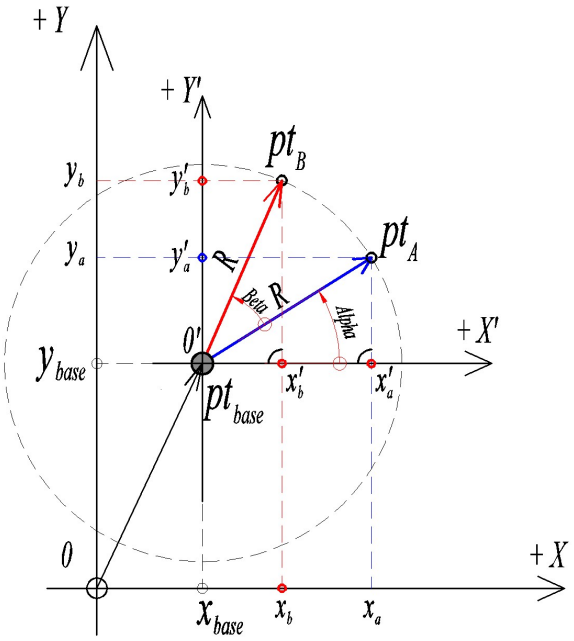
Линейни сегменти образуват полигон тогава и само тогава, когато:

1. Пресечната точка между всяка двойка съседни сегменти в цикличния списък, е : $e_i \cap e_{i+1} = pt_{i+1}$, за всички $i = 0, ..., n-1$;
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. Това е илюстрирано на Фигура 2.4, където $n = 14$.

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. Желаем 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. Това е илюстрирано на фигура 2.5.



Фигура 2.5: Ротация на точка

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:

$$R = \sqrt{{x'_a}^2 + {y'_a}^2} \tag{2.3}$$

Ъгълът на завъртане α се получава от формулата:

$$\alpha = arccos\frac{x'_a}{R} \tag{2.4}$$

Тогава координатите на точка pt_B в координатната система XOY са:

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

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

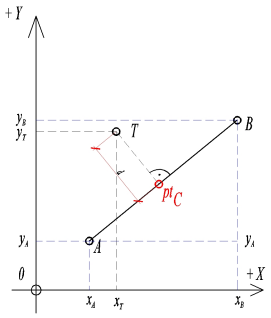
Тъй като знаменателят във формулите (2.5) и (2.6) може да бъде нула, то трябва да се направят две допълнителни уточнения, които третират и този случай:

1. Ако $|\alpha + \beta| = 0.5\pi$ или $|\alpha + \beta| = 1.5\pi$, то $x_b = x_{base}$;
2. Ако $\alpha + \beta = 0$ или $|\alpha + \beta| = \pi$, то $y_b = y_{base}$.

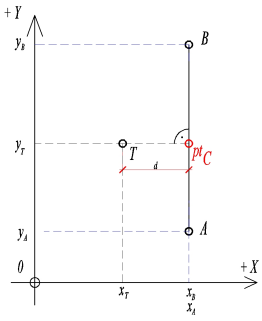
Така новата точка е: $pt_B = list(x_b, y_b)$.

Перпендикуляр от точка към дадена права.

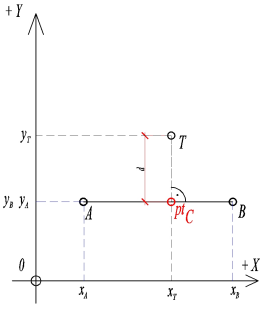
Правата е зададена с две точки $A = list(x_a, y_a)$ $B = list(x_b, y_b)$ и тестовата точка е $T = list(x_t, y_t)$. Искаме да намерим точка $C = list(x_c, y_c)$ от правата $list(A, B)$ такава, че векторът определен от точките T и C е перпендикулярен на правата, виж фигура 2.6.



Фигура 2.6: Перпендикуляр от точка към права в общо положение



Фигура 2.7: Перпендикуляр от точка към вертикална права



Фигура 2.8: Перпендикуляр от точка към хоризонтална права

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 $F = list(A, B, T)$, виж Алгоритъм

1. 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 $pt_C = list(x_A, y_T)$. Виж фигура 2.7;

2. Ако $y_A = y_B$, тогава правата е вертикална и търсената точка е $pt_C = list(x_T, y_A)$. Виж фигура 2.8;

3. Ако не са изпълнени горните условия, тогава търсим наклона m на правата $list(A, B)$:

$$m = \frac{y_B - y_A}{x_B - x_A} \tag{2.7}$$

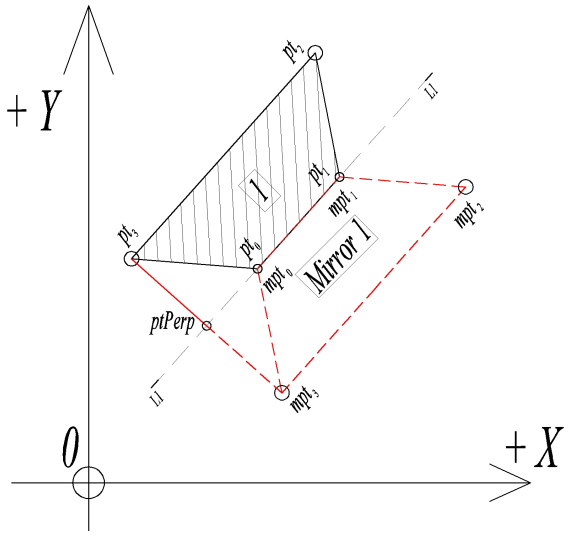
$$x_C = \frac{(\frac{x_T}{m} + y_T + m.x_A - y_A)}{m + \frac{1}{m}} \tag{2.8}$$

$$y_C = y_A + m(x_C - x_A) \tag{2.9}$$

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

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, виж фигура 2.9.



Фигура 2.9: Огледален образ на полигон.

Полигона $Mirror1 = list(mpt_0, mpt_1, mpt_2, mpt_3)$ от фигура 2.9 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.

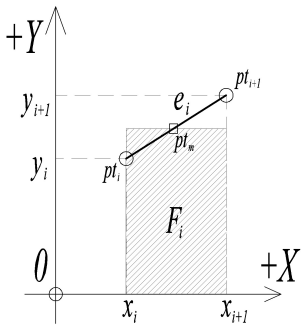
2. Образуваме правата $L1 - L1$. Правата $L1 - L1$ се образува по две точки. Първата точка е $pt_{L1} = (polar(pt_0; (angle = pt_1, pt_0); 10e10))$. Втората точка е $pt_{L2} = (polar(pt_1; (angle = pt_0, pt_1)); 10e10)$. Намираме полярните координати на точките pt_{L1} и pt_{L2} - базова точка, ъгъл 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 2.1.

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 mpt_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.

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,..., 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 2.10: Лице на трапец

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

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

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

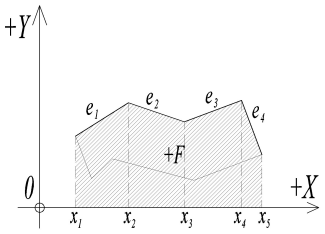
$$F_i = \frac{(x_{i+1} - x_i)(y_{i+1} + y_i)}{2} \tag{2.11}$$

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,..., pt_{n-1}) или $list$ (pt_{n-1}, pt_0,..., pt_{n-1}). Това лице ще наречем ориентирано лице. Тази процедура се прилага за всички e_i

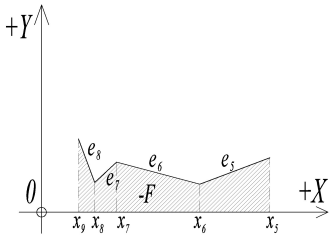
сегменти. За да се пести процесорно време няма смисъл всяко лице да се дели на 2. Затова сборът от ориентираните лица можем да запишем както следва:

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

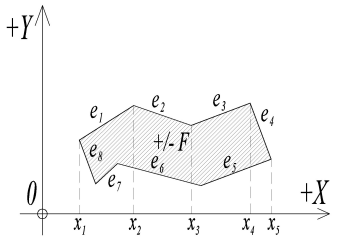
Ако координатите на точките на върхове $list(pt_0, pt_1, \ldots, pt_5)$ удовлетворяват условията $x_5 > x_4 > x_3 > x_2 > x_1 > x_0$, то съответните площи са положителни и $F > 0$. Ако коррдинатите на точките на върхове $list(pt_0, pt_1, \ldots, pt_5)$ удовлетворяват условията $x_5 > x_6 > x_7 > x_8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



Фигура 2.11: Положителна сума на площите



Фигура 2.12: Отрицателна сума на площите



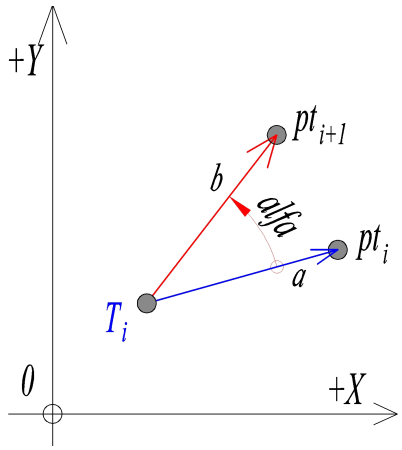
Фигура 2.13: Сума на площите

Algorithm 1 getPolyArea2

```
/*Функция за намиране на ориентираното лице на полигон*/
  procedure GETPOLYAREA2(e_1, e_2, ..., e_n)
      Sum = 0.
      for e_1, e_2, ..., e_n do
          Sum = (x_{i+1} - x_i)(y_{i+1} + y_i) + Sum
```

В примера показан на фигура 2.13 сумата на площите е положителна $\Rightarrow P > 0$, или ориентацията е по часовата стрелка (CW), иначе е обратна на часовата стрелка $(anti - CW)$.

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 XY координатна система. Виж фигура 2.14.



Фигура 2.14: Ориентиран ъгъл между два вектора

Първо определяме дължината на векторите $\vec{a}$ и $\vec{b}$, а след това и тяхното скаларно 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).

$$\|\vec{a}\| = \sqrt{x_a^2 + y_a^2} \tag{2.13}$$

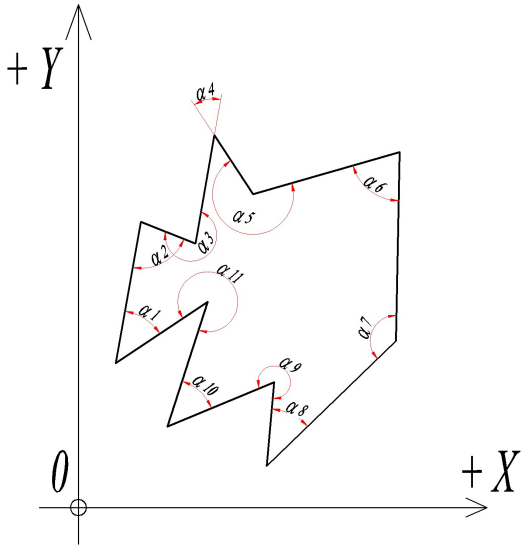
$$\|\vec{b}\| = \sqrt{x_b^2 + y_b^2} \tag{2.14}$$

скаларното произведение е:

$$\vec{a} \cdot \vec{b} = x_a x_b + y_a y_b \tag{2.15}$$

и така получаваме

$$\cos\alpha = \frac{\vec{a} \cdot \vec{b}}{\|a\|\|b\|} \tag{2.16}$$



Фигура 2.15: Вътрешни ъгли в полигон

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. Ориентацията на полигон е разгледан в точка 2.1. За да намерим вътрешните ъгли посоката на полигона трябва да бъде обратно на часовниковата стрелка $anti - CW$, виж точка 2.24. 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 2 InsidePolyAngle

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

procedure INSIDEPOLYANGLE($pt_0, pt_1, \ldots, pt_n$)

if isClockWise $pt_0, pt_1, \ldots, pt_n$ **then return** ptList = reverse $pt_0, pt_1, \ldots, pt_n$

i = 0

L = length of $pt_0, pt_1, \ldots, pt_n$

Repeat L

for pt_i, pt_{i+1}, pt_{i+2} **do** α = getInsideAngle pt_i, pt_{i+1}, pt_{i+2}

if isClockWise pt_i, pt_{i+1}, pt_{i+2} **then return** α = (2π - α)

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.

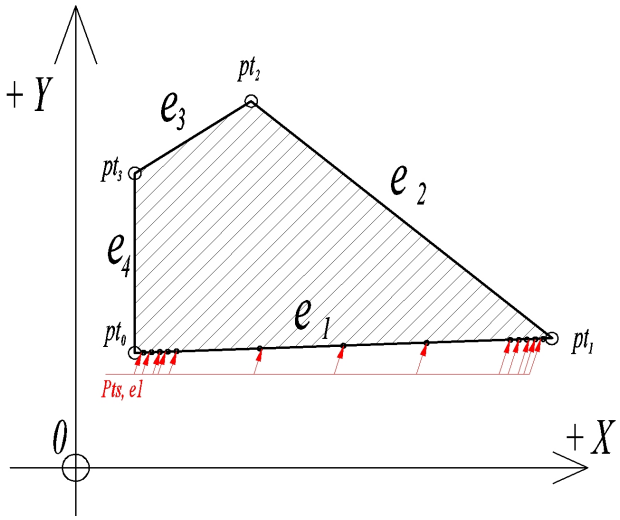
Добавяне на точки в линеен сегмент

Добавянето на точки в линеен сегмент е необходимо за намирането на повече възможни валидни разполагания на полигоните Π_i, $i = 1, \ldots,$ в полигона Δ. Виж точка 4.4. Сега да вземем един определен полигон $\Pi = list(e_1, e_2, \ldots e_n)$ от входящия списък от полигони. Всеки сегмент e_i от полигона $\Pi = list(e_1, e_2, \ldots e_n)$ се разделя на три подсегмента. За всеки подсегмент се добавят подробни точки. Принципа за поставяне на подробни точки виж фигура 2.16.

Получаването на подробни точки за сегмент $e_i = list(pt_i, pt_{i+1})$ става чрез полярни координати (базова точка, ъгъл и дължина). Дефинираме фунцията *polar* която връща точка по зададена базова точка, ъгъл и дължина. Базовата точка е pt_i. Ъгъла на сегмента e_i спрямо абцисната 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 $pt_{L1,i} = (polar(pt_0, ang_e 1, \frac{L_1}{5})$.

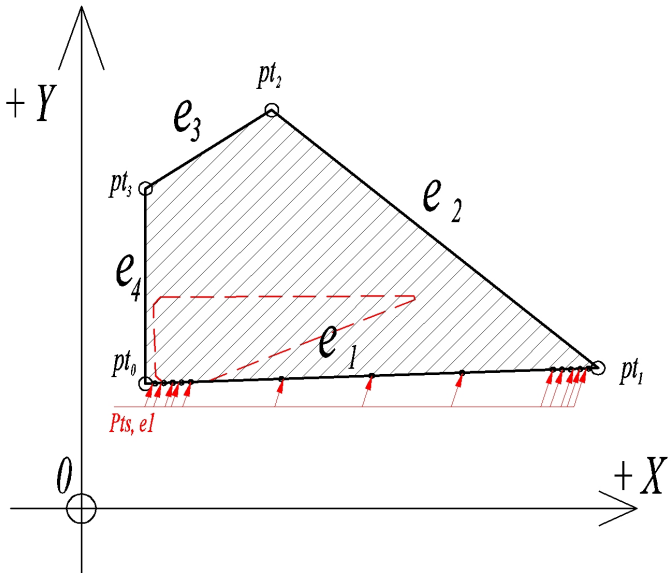
Изчисляването на другите подробни точки за сегмента e_i става в същата логика както за точка $pt_{L1,i}$. Всички подробни точки за сегмент e_i на фигура 2.16 са



Фигура 2.16: Подробни точки в сегменти на полигон

Фигура 2.17: Планка със скосени върхове

отбелязани с $Pts, e1$. Добавянето на тези подробни точки 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. За планки със скосени върхове виж фигура 2.18.

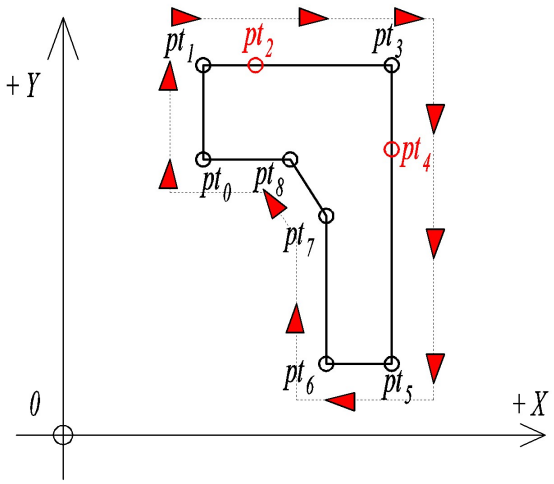


Фигура 2.18: Полигон Π_i (с пунктир в червен цвят). Разполагане на подробни точки по сегмент e_1.

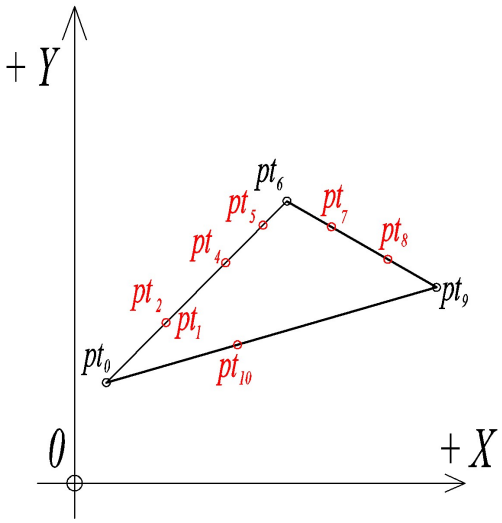
Премахване на подробни точки от линеен сегмент

Проверката се изпълнява преди да започне самото подреждане на полигоните. При представянето на полигоните като списък от върхове $list(pt_0, pt_1, \ldots, pt_n)$ 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. Проверката



Фигура 2.19: Излишни точки, които не са върхове на полигона



Фигура 2.20: Триъгълник, описан със списък съдържащ повече от три точки

за наличие на излишни точки трябва да се направи преди да започне самия разкрой. Така е необходима проверка на всички полигони за самопресичане и излишни точки. Проверката се базира на критерия, че ако площта на триъгълника $list(pt_i, pti+1, pt_{i+2})$ е близка до нула (виж формула (2.27)), то точката pt_{i+1} е излишна. Какво означава излишна в дадения случай? Това означава, че точката pt_{i+1} трябва да се премахне от списъка от върховете на полигона П с нейното име pt_{i+1} или нейния индекс. Тази проверка важи и за случая когато има съвпадане на възли, например във фигура 2.20 връх $pt_1 \equiv pt_2$. В съставения списък в таблица 2.1 точките pt_1 и pt_2 са с еднакви координати (под еднакви се разбира приблизително еднакви, съгласно формула 2.1).

Индекс	X	Y
1	2	3
0	x_0	y_0
1	x_1	y_1
2	x_2	y_2
3	x_3	y_3
...	...	...
n	x_n	y_n

Таблица 2.1: Съставен списък от точки на полигон П

След преминаване през целия списък на проверка по формула (2.27) се премахва точка с индекс 1. Ако броят на премахнатите точки е $countR$, то броят на точките в новия списък е:

$$m + 1 = n + 1 - countR \tag{2.17}$$

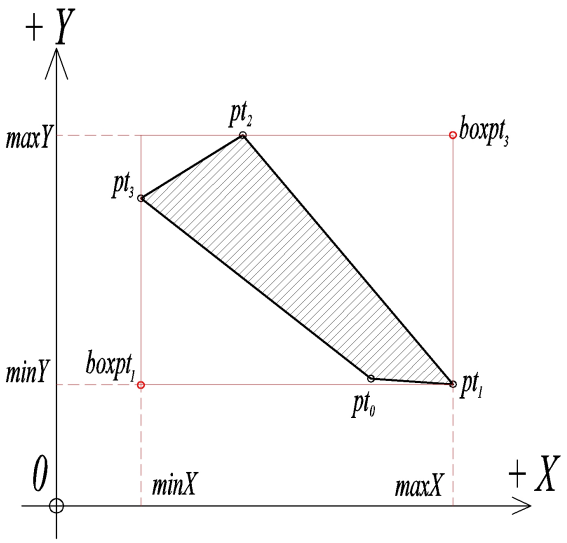
По този начин е съставен списъка на новия полигон П′.

Индекс	X	Y
1	2	3
0	x_0	y_0
nil	nil	nil
1	x_1	y_1
2	x_2	y_2
...	...	...
m	x_m	y_m

Таблица 2.2: Списък с премахнати точки

2.2 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 2.21.



Фигура 2.21: *box* на полигон

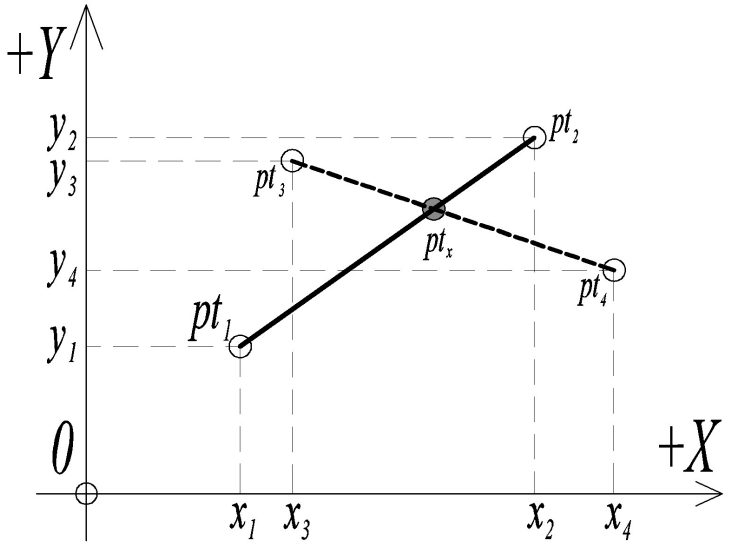
За да намерим *box* на полигона Π_i трябва да преминем през списъка от точки $\Pi_i = list(pt_0, pt_1 \dots pt_n)$ и за всяка една точка да вземем координатите й по $X.pt$ и по $Y.pt$. След това да сортираме двата нови списъка по низходящ ред, $listX = (x_0, x_1 \dots x_n)$ и $listY = (y_0, y_1 \dots y_n)$. 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.

2.3 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 $false$.

Първа функция: Намиране на координати на пресечна точка pt_x.



Фигура 2.22: Пресичане на два сегмента

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:

$$X = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} \end{vmatrix}}, Y = \frac{\begin{vmatrix} \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & y_3 \\ x_4 & y_4 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}{\begin{vmatrix} \begin{vmatrix} x_1 & 1 \\ x_2 & 1 \end{vmatrix} & \begin{vmatrix} y_1 & 1 \\ y_2 & 1 \end{vmatrix} \\ \begin{vmatrix} x_3 & 1 \\ x_4 & 1 \end{vmatrix} & \begin{vmatrix} y_3 & 1 \\ y_4 & 1 \end{vmatrix} \end{vmatrix}}. \tag{2.18}$$

Като пресметнем детерминантите в (2.18), получаваме следните изрази за X и Y:

$$X = \frac{(x_1y_2 - y_1x_2)(x_3 - x_4) - (x_1 - x_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(x_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{2.19}$$

$$Y = \frac{(x_1y_2 - y_1x_2)(y_3 - y_4) - (y_1 - y_2)(x_3y_4 - y_3x_4)}{(x_1 - x_2)(x_3 - y_4) - (y_1 - y_2)(x_3 - x_4)}. \tag{2.20}$$

Тук трябва да обърнем внимание, че това е точката на пресичане на правите, дефинирани от крайните точки на сегментите. Формула (2.18) може да даде пресечна точка, която е извън сегментите e_1 и e_2. За да намерим дали пресечната

точка принадлежи на сегментите, ще използуваме техните уравнения:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} + t \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} = 0 \tag{2.21}$$

$$\begin{pmatrix} x_3 \\ y_3 \end{pmatrix} + u \begin{pmatrix} x_4 - x_3 \\ y_4 - y_3 \end{pmatrix} = 0. \tag{2.22}$$

За стойности на параметрите $-\infty < t, u < \infty$ получаваме двете прави а за $0 \leq t \leq 1$ и $0 \leq u \leq 1$ получаваме сегментите e_1 и e_2, съответно. Тогава пресечната точка се получава при следните стойности на

$$t_0 = \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{2.23}$$

$$u_0 = \frac{(x_1 - x_2)(y_1 - y_3) - (y_1 - y_2)(x_1 - x_3)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{2.24}$$

или:

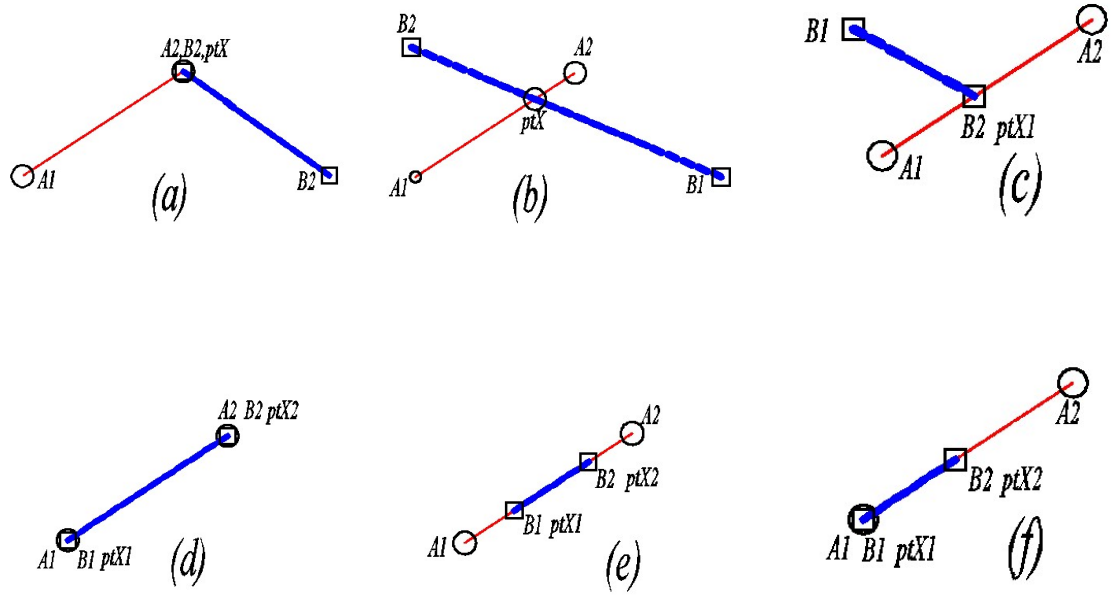
$$ptX = (x_1 + t_0(x_2 - x_1), y_1 + t_0(y_2 - y_1)) = (x_3 + u_0(x_4 - x_3), y_3 + u_0(y_4 - y_3)). \tag{2.25}$$

Очевидно, ако

$$0 \leq t_0 \leq 1 \quad \text{и} \quad 0 \leq u_0 \leq 1, \tag{2.26}$$

то пресечната точка ptX принадлежи на сегментите и е търсената точка.

Възможни варианти на пресичане на два сегмента са изобразени на фиг. 2.23. Ще приемем, че пресичане има само в случаите (a), (b) и (c). Така приемаме, че



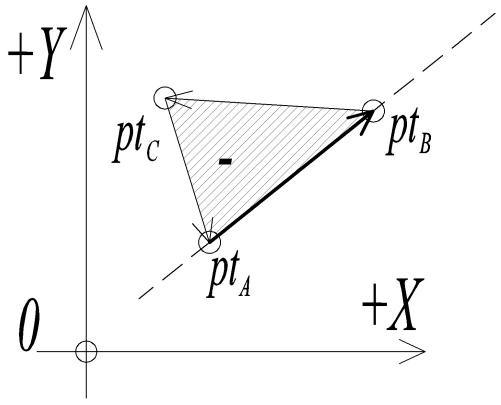
Фигура 2.23: Случаи на пресичане на два сегмента

няма пресичане на два успоредни сегмента.

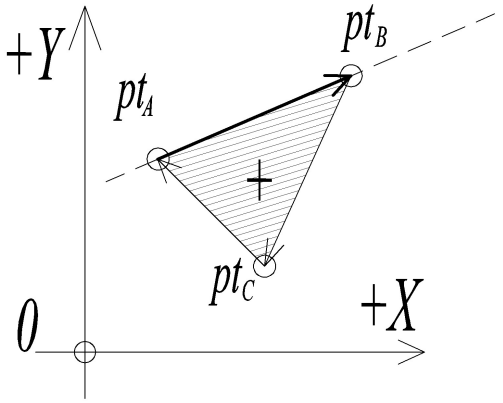
Да разгледаме случай (c). Там имаме прието пресичане в точка ptX_1 и връх B_2. В случай (f) имаме същото пресичане, но то няма да бъде взето под внимание, защото двата сегмента са успоредни. Това условие е необходимо за построяването на логическата таблица с върховете на полигоните и техните пресечни точки. Виж таблица 2.22.

Втора функция: Търсим има ли пресичане между $e_1 = list(ptA, ptB)$ и $e_2 = list(ptC, ptD)$ без да изчисляваме самите координати на пресечната точка pt_x (наречено тук логическо пресичане).

Пресмятането на стойността на тази логическа функция се прави в две стъпки. Първо пресмятаме ориентираното лице F_1 на полигона ptA, ptB, PtC по формула (2.12) и после пресмятаме ориентираното лице F_2 на полигона ptA, ptB, PtD по формула (2.12). Ако двете лица са с различни знаци, тогава имаме логическо пресичане и резултатът е $true$, иначе няма логическо пресичане и резултатът е $false$. Ще отбележим, че последователността в списъка е от основно значение за проверката на този критерий. Тук изразяваме известно несъгласие [9, стр. 29], че c е



Фигура 2.24: Ляво стояща точка



Фигура 2.25: Дясно стояща точка

лява на сегмента ab тогава и само тогава когато $\triangle abc$ има положителна площ. При дадената подредба a, b, c При фигури 2.24 и 2.25 следва логичния въпрос: Какво се случва ако ptC е колинеарна с ptA и ptB? Тогава площта на триангулацията е нула от което следва, че те лежат на една права. Тъй като работим с реални числа, може площта на триангулацията да се получи много близка до нулата. Това се получава когато ptC е много близко до правата ab, но не лежи на нея. Затова въвеждаме условието:

$$-fuzz^2 \leq F \leq fuzz^2 \Rightarrow F = 0. \tag{2.27}$$

Algorithm 3 isLeft

/*Функция за проверка дали точката pt_c от ляво на сегмента $e_i = list(pt_a, pt_b)$. */

procedure ISLEFT(pt_a pt_b pt_c)

if $(getPolyArea2(pt_a, pt_b, pt_c) < 0.$ **then return** True
elseFalse

В алгоритъм 3 се използува формула (2.12) за лице на полигон, в случая триъгълник с върхове pt_a, pt_b, pt_c. Във формула (2.12) е необходимо умножението на две числа. Тъй като умножението е "скъпа"операция за процесора и паметта, затова можем да въведем следната модификация:

Algorithm 4 isLeft+-

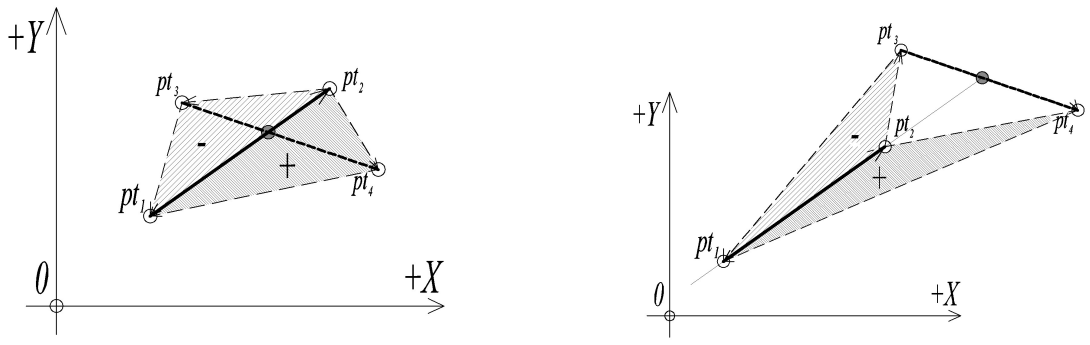
```
/*Функция за определяне на знака на ориентираното лице*/
  procedure ORIENTTRIANG+-(x_{i+1}, x_i, y_{i+1}, y_i)
      if (x_{i+1} − x_i) < 0.0) then
          a = −1.0
      else
          a = +1.0
      if (y_{i+1} + y_i) < 0.0) then
          b = −1.0
      else
          b = +1.0
      Orient = a * b
```

В алгоритъм 4 ще намерим само знака на лицето, което ще ни бъде необходимо по-нататък за намиране на логическо пресичане на два полигона.

CAD системите генерират CAD-сегменти, които съдържат вътрешни за сегмента точки, но за нашия анализ на полигоните тези вътрешни точки са излишни. Затова е необходим метод за откриване на колинеарност на три точки, което е направено в Алгоритъм 5.

Algorithm 5 Collinear

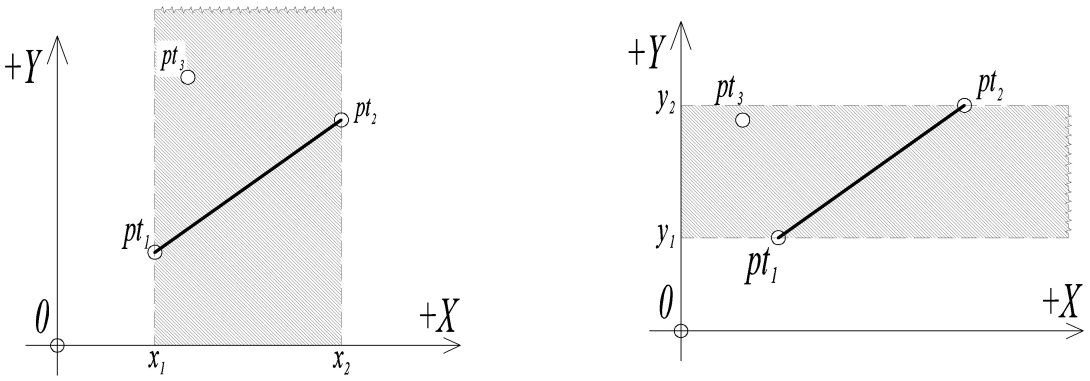
```
/*Функция колинеарност на три точки. */
1: procedure COLLINEAR(pt_1 pt_2 pt_3)
2:     Area = (getPolyArea2(pt_1, pt_2, pt_3)
3:     if (−fuzz^2 ≤ Area)&&(Area ≤ fuzz^2) then return True
4:     elseFalse
```



Фигура 2.26: Логическо пресичане на два сегмента

Възможно е разположението на двата сегмента да бъде като показаното на лявата фигура 2.26. Според критерия за ориентираните лица би трябвало сегментите да се пресичат, но виждаме, че в случая това не е така. Затова трябва да

въведем още една проверка: дали pt_3 е между точки pt_1 и pt_2.



Фигура 2.27: Точка между две точки: по x и y, съответно

Казваме, че pt_3 е между pt_1 и pt_2, когато:

$$x_1 \leq x_3 \leq x_2 \parallel y_1 \leq y_3 \leq y_2. \tag{2.28}$$

За тази проверка ще въведем псевдокод съгласно [9], стр. 32, но с малка редакция, функцията не проверя за колинеарност а връща само *true* или $flase$.

Algorithm 6 Between

/*Ако сегмента не е вертикален провери "между" по x иначе по y */

```
1: procedure BETWEEN
2:     if x1 ≠ x2 then
3:         (x1 ≤ x3)&&(x3 ≤ x2) || (x1 ≥ x3)&&(x3 ≥ x2)
4:     else
5:         (y1 ≤ y3)&&(y3 ≤ y2) || (y1 ≥ y3)&&(y3 ≥ y2)
```

След въвеждането на критерия за Between (между), проверката за логическо пресичане може да избегне думата "възможно" пресичане. След като точките $pt3$ и $pt4$ са преминали проверката за "между" и проверката за ориентираните лица по формула (2.12), които дават различен знак на площите, то сегментите имат пресечна вътрешна точка. Трябва да направим едно уточнение, посоченият метод дава както тривиалния случай на пресичане, така и не тривиалните случаи. Затова трябва да въведем допълнителни условия кое приемаме за пресичане, виж фигури 2.23 случаите (a), (b) и (c). Това означава, че трябва да допуснем една от крайните точки на единия сегмент да съвпада с една от другите две точки на първия сегмент. Или само една от крайните точки на сегмент e_j с другите две крайни точки от сегмент e_i да образуват отлична от нула площ.

$$Area(pt_1, pt_2, pt_3) \neq 0 \ \parallel \ Area(pt_1, pt_2, pt_4) \neq 0. \tag{2.29}$$

Алгоритъм за логическо пресичане.

Algorithm 7 IntersT/F

H /*Функция за логическо пресичане на два сегмента */

procedure INTERST/F(pt_1, pt_2, pt_3, pt_4)
if $((between(pt_1, pt_2, pt3)\&\&Collinear(pt_1, pt_2, pt_3)$
$\&\&(notCollinear(pt_1, pt_2, pt_4)))$ **then return** true
else if $((between(pt_1, pt_2, pt4)\&\&Collinear(pt_1, pt_2, pt_4)$
$\&\&(notCollinear(pt_1, pt_2, pt_3)))$ **then return** true
else if $((between(pt_1, pt_2, pt4)\&\&Collinear(pt_1, pt_2, pt_4)$
$\&\&(notCollinear(pt_1, pt_2, pt_3)))$ **then return** true
else if $(not(Collinear(pt_1, pt_2, pt_3))\&\&(isLeft(pt_1, pt_2, pt_3)\&\&(between(pt_1, pt_2, pt_3))$
$\&\&(notCollinear(pt_1, pt_2, pt_4))(not(isLeft(pt1_p t_2, pt_4))(between(pt_1, pt_2, pt_4)))$
then return true
else
False

На пръв поглед алгоритъмът изглежда сложен. Всъщност, иска се намирането на ориентираните лица на триъгълниците $pt_1, pt_p t2, pt_3$ и pt_1, pt_2, pt_4 и допълнителна елементарна проверка *Between*, виж алгоритъм 6. За намирането на логическото пресичане не е необходимо да се правят всички проверки в Else If. Ако една от тях е изпълнена то функцията връща *true* и спира да проверява другите условия. Затова е по-добре да се постави на първо място най-често срещаното условие за пресичане. В софтуера разработен към настоящата дисертация това е, когато единият връх лежи на другата права и върхът е между двете точки pt_1, pt_2, виж фигура 2.23, случай (c).

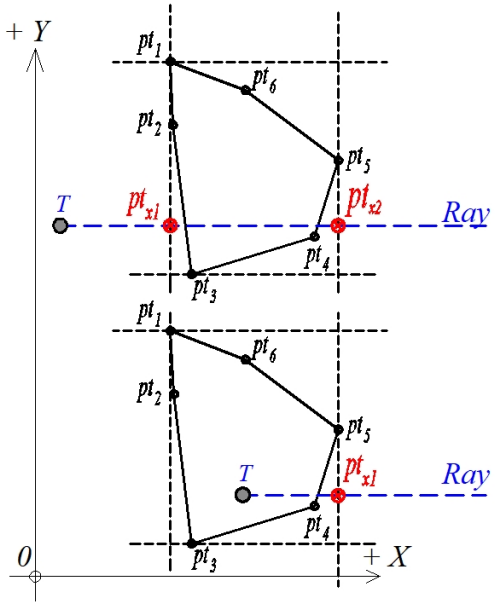
2.4 Точка в полигон

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

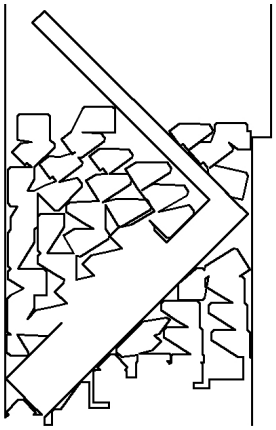
Тук ще бъдат разгледани два метода за решавана на тази задача - Ray crossing method (пресичащ лъч), [32], и Balanced sum of angles (Балансирана сума на ъглите). В разработения към дисертациония труд софтуер се използва следният подход преди да се приложи метода на пресичащия лъч за всички сегменти. На фигура 2.28 с черна пунктирна линия е показана обвивката на полигона $\Pi_i = list(pt_0, pt_1, ..., pt_n)$. Прави се проверка 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 2.2.

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 $(pt_0, pt_1, ..., pt_n)$ 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 2.5 то сложността може да се намали до $\mathcal{O}(n)$.

Ray crossing method (метод на пресичащия лъч)



Фигура 2.28: Пресичащ лъч



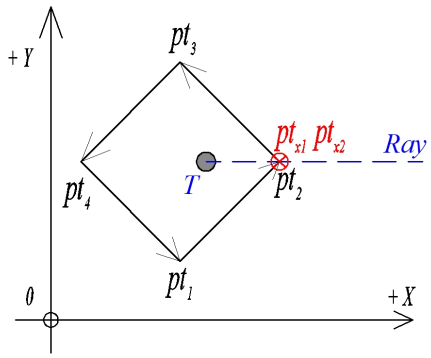
Фигура 2.29: Полигон Δ. Остатъчния (изрязания) полигон.

Проверка дали дадена точка T е в даден полигон П. 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 П. Ако боря на пресечните точки е четен, то точка T е извън полигона, иначе тя е вътре в полигона.

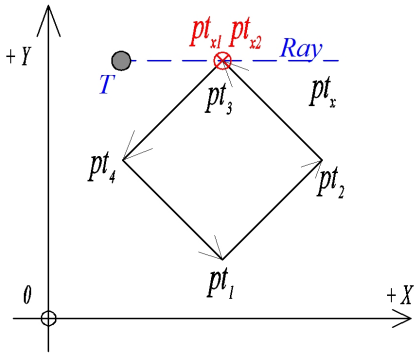
В софтуера към дисертацията труд се използва една оптимизация на този метод. Ако броя на върховете на дадените полигон е по-голям от 15, то се прави проверка за броя пресечни точки с контурната обвивка на полигона, виж фигура 2.28. В частните случаи, когато пресечните точки на лъча Ray със сегментите на полигона П съвпадат с даден връх полигона П се получават неточни резултати илюстрирани на фигури 2.30, 2.31.

Съгласно формулирания по-горе критерий за четност на пресечните точки на лъча с полигона П, тестваната точка T на фигура 2.34 би следвало да бъде вътре в полигона П, а на фигура 2.35 – вън от полигона П. Но точката е вътрешна и в двата случая, тъй като тя е от контура на полигона П.

Сега да разгледаме случая, когато пресечната точка pt_x на лъча Ray с даден сегмент от полигона П съвпада с една от крайните точки на този сегмент, виж фигури 2.30, 2.36 и 2.31.



Фигура 2.30: Случай (a) – точка T вътре в полигона П



Фигура 2.31: Случай (b) – точка T вън от полигона П

В случая показан на фигура 2.30 лъчът Ray действително пресича границата и излиза извън областта и би трябвало броя на пресичанията да бъде едно (нечетен брой). Но при проверка за броя пресичания получаваме две: първото пресичане pt_{x1} е между лъча Ray и сегмента $list(pt_1, pt_2)$, а второто pt_{x2} е между лъча Ray и сегмента $list(pt_2, pt_3)$. При което алгоритъмът ще направи извод, че точката T е извън полигона П.

В случая показан на фигура 2.31 лъчът Ray пресича П в също в две точки – с първо пресичане $pt_{x1} = Ray \cap list(pt_2, pt_3)$ и второ пресичане $pt_{x2} = Ray \cap list(pt_3, pt_4)$, т.е. четен брой пресичания. Тогава алгоритъмът ще направи извод, че точката T е извън полигона П.

Тези два примера показват, че при един и същ полигон П за различни местоположения на тестваната точка T ще получим неправилни изводи за това дали точката T е в полигона П. Това налага да се поставят допълнителни критерии за разпознаване на случай (a) и случай (b) от фигури 2.30 и 2.31. За да разграничим тези два случая ще направим допълнителна класификация на сегментите на полигона П като "Възходящ" и "Низходящ" спрямо тестваната точка T. Двата върха на даден сегмент $e_i = list(pt_i, pt_{i+1})$ и тестваната точка T образуват подреден списък от точки $L = list(T, pt_i, pt_{i+1})$. За този списък L се прилага критерия $isLeft$, който дава ориентираното лице на триъгълника L. Казваме, че сегмента е "Възходящ", ако $isLeft = True$, иначе сегмента е "Низходящ". Тези правила важат само тогава когато има не тривиално пресичане, т.е. пресечната точка съвпада с един от върховете на разглеждания сегмент, за критерия за съвпадение на две точки виж Секция **??**. Ако има съвпадение ще въведем следните правила за отчитане на броя пресичания, съгласно [33]:

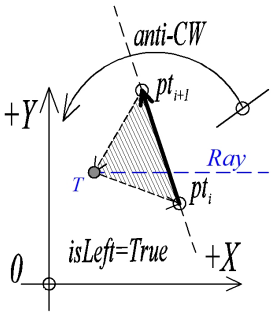
1. Когато пресечната точка съвпадне с **начална** точка на "Възходящ" сегмент отчитаме, че има пресичане;

2. Когато пресечната точка съвпадне с **крайната** точка на "Низходящ" сегмент отчитаме, че има пресичане;

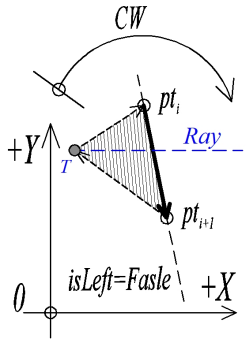
3. Хоризонталните сегменти се изключват (тъй като са колинеарни на лъча);

Обърнато е внимание, че преди да се приложат тези правила е необходимо предварително да се направят няколко проверки:

1. Ако тестваната точка T съвпада с един от върховете на сегмента, то T е



Фигура 2.32: Ляво стояща точка на e_i

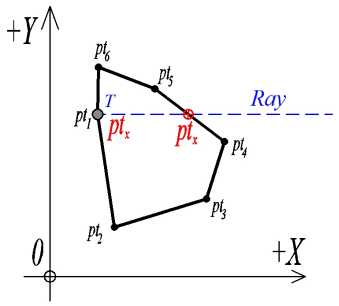


Фигура 2.33: Дясно стояща точка на e_i

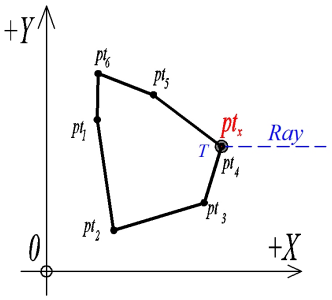
вътре в полигона П и агоритъмът спира без да прави други проверки. Виж фигури 2.34 и 2.35;

2. Ако T не съвпада с $list(pt_i, pt_{i+1})$ и има тривиално пресичане - отчиатаме броя на пресичанията;

3. Ако T не съвпада с $list(pt_i, pt_{i+1})$ и има пресичане и ($pt_x \equiv pt_i$ или $pt_x \equiv pt_{i+1}$), то за да отчетем или не отчетем пресичането прилагаме критерия за "Възходящ" и "Низходящ".



Фигура 2.34: T съвпада с един от върховете: Случай (a)

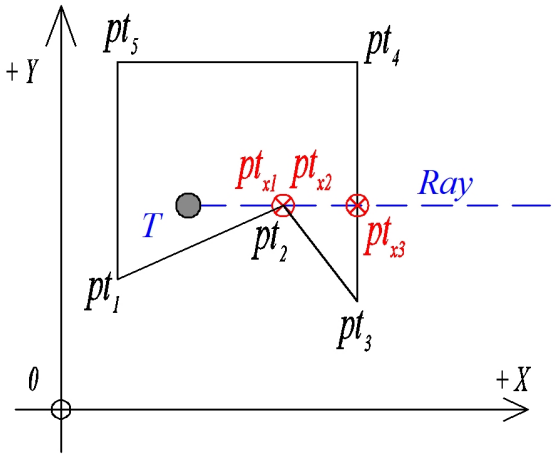


Фигура 2.35: T съвпада с един от върховете: Случай (b)

С така въведените критерии ще илюстрираме работата на алгоритъма за случаите показани на фигури 2.30 и 2.31.

Първо ще разгледаме случая показан на фигура 2.30. Първото пресичане е $pt_{x1} = Ray \cap list(pt_1, pt_2)$. Тъй като $isLeft = True$ и $pt_{x1} \equiv pt_2$, т.е. е крайна точка на сегмента, то не отчитаме, че има пресичане. Второто пресичане $pt_{x2} = Ray \cap list(pt_2, pt_3)$. Тъй като $isLeft = True$ и $pt_{x2} \equiv pt_1$, т.е. е начална точка на сегмента, то отчитаме, че има пресичане. Следователно сме отчели едно пресичане, което от приетите по-горе критерии правим извода, че тестваната точка T е в полигона П.

Сега да разгледаме случая показан на фигура 2.31. Първото пресичане е $pt_{x1} = Ray \cap list(pt_2, pt_3)$. Тъй като $isLeft = True$ и сегмента е възходящ с $pt_{x1} \equiv pt_3$ крайната точка на сегмента, то не отчитаме, че има пресичане. Второто пресичане



Фигура 2.36: Точка T вътре в П: Случай (c)

е $pt_{x2} = Ray \cap list(pt_3, pt_4)$. Тъй като $isLeft = False$, то сегмента е низходящ. Също, тъй като $pt_{x2} \equiv pt_3$ е начална точка на сегмента, то не отчитаме, че има пресичане. Следователно сме отчели нула пресичания (четен брой), което ни дава право да направим извода, че тестваната точка T е извън полигона П.

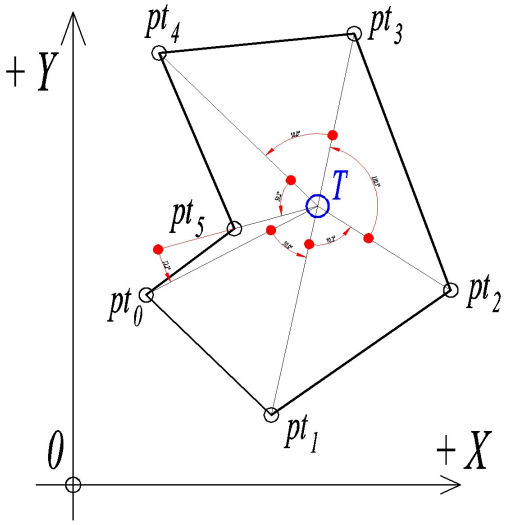
Да илюстрираме алгоритъма и за още един случай показан на фигура 2.36. Първото пресичане е $pt_{x1} = Ray \cap list(pt_1, pt_2)$. Тъй като $isLeft = True$, то сегментът е "Възходящ". Тъй като $pt_{x1} \equiv pt_2$ е крайна точка на сегмента, то не отчитаме, че има пресичане. Второто пресичане $pt_{x2} = Ray \cap list(pt_2, pt_3)$. Тъй като $isLeft = False$, сегментът е "Низходящ" и $pt_{x2} \equiv pt_2$ е начална точка на сегмента, то не отчитаме, че има пресичане. Третото пресичане $pt_{x3} = Ray \cap list(pt_3, pt_4)$. То е тривиално и не прилагаме критериите за "Възходящ" и "Низходящ" сегмент и отчитаме, че има пресичане. Следователно сме отчели едно пресичане което ни дава право да направим извода, че тестваната точка T е в полигона П. От фигура 2.36 се вижда, че ако не се приложи правилото "Възходящ"/ "Низходящ" броя на пресечните точки е три. Следователно тестваната точка T е в полигона, но това е случайност, на която не може да се разчита. Виж фигура 2.30.

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 $\vec{b} = list(T, pt_{i+1})$. Виж фигура 2.14.

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

$$Sum_\alpha = \sum_{i=1}^{n} arccos\alpha_i \tag{2.30}$$



Фигура 2.37: Балансирана сума на ъглите

Ако точката T е вътрешна за полигона П, 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 П:

1. Ако $Sum_\alpha = 2\pi$ полигона П е ориентиран CW;
2. Ако $Sum_\alpha = -2\pi$ полигона П 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 П. Затова тук използуваме метода на пресичащия лъч с предварителната проверка описана по-горе, като критерии дали дадената точка T е вътре в полигона П.

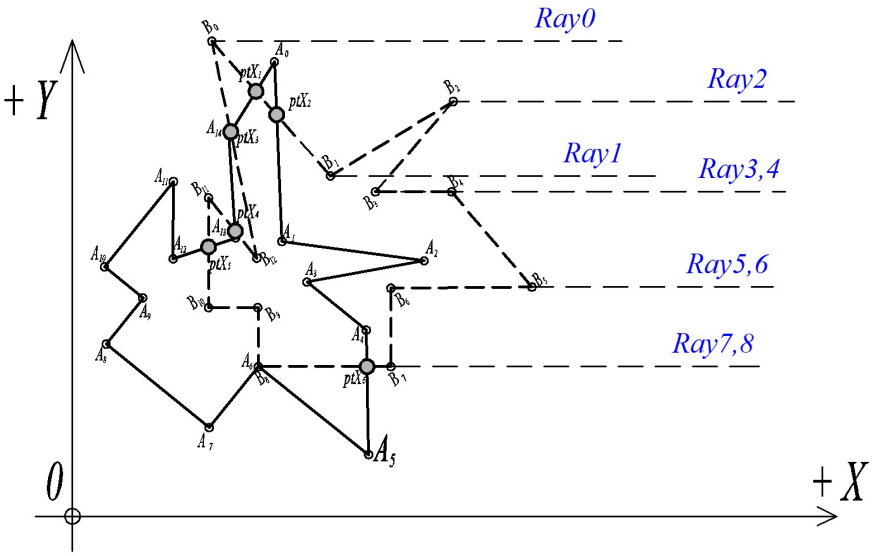
Тук ще разгледаме задачи получени от практиката и на фокус ще бъдат както изпъкнали така и не изпъкнали полигони. Представения алгоритъм ще работи еднакво добре и за двата вида на полигони.

2.5 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 използувана 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 използува в разработения от автора софтуер.

По-бързи методи за проверка дали два полигона се пресичат използуват намирането на точките на пресичане на границите им с техните координати или

проверка за логическо пресичане. В повечето случаи ще използуваме 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 използува също при търсенето на възможно положение на даден полигон Π_i спрямо основния полигон Δ. В секция 4.4 при 2D разкроя ще дадем повече информация. На фигура 2.38 е дадено едно примерно разположение на два полигона A и B (полигон A е зададен с плътна линия, а B - с пунктирана).



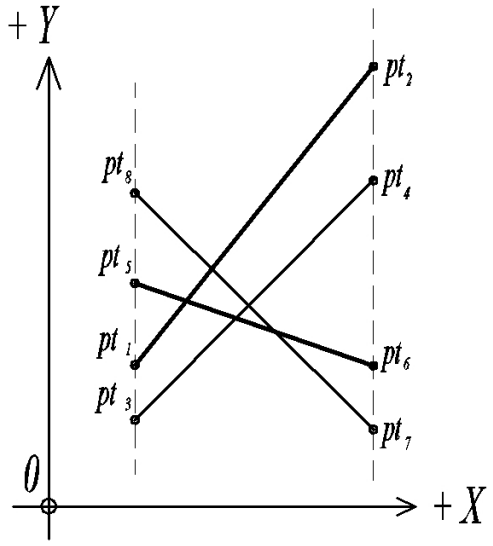
Фигура 2.38: Пресичане на два полигона

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. В примера даден на фигура 2.38 се вижда, че трябва да се направят 8 от 14 проверки или 112 цикъла. 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 ще използваме вертикална сканираща линия, метод описан по-долу. Съгласно [43] сложността на този алгоритъм за всички K пресичания между N сегмента е $\mathcal{O}((N+K)\log N)$. Както се вижда, ако всеки един сегмент се пресича с всеки друг, виж фигура 2.39, то $K = N$ и тогава този алгоритъм ще има сложност $\mathcal{O}(N^2)$.

Тъй като предположихме, че самопресичащи се полигони не се разглеждат, то в настоящия дисертационен труд е приложен този алгоритъм. И така сега ще представим този алгоритъм за дадени два полигона

$$A = list(A_1, A_2, ..., A_n) \quad \text{и} \quad B = list(B_1, B_2, ..., B_n).$$



Фигура 2.39: Пресичане на всички сегменти. Сложност на намиране пресечните точки - $\mathcal{O}(N^2)$

Всеки един от сегментите A и B се представя с два последователни върха (A_i, A_{i+1}) и съответно (B_i, B_{i+1}). Съставя се нов списък, които включва индекса на сегментите на двата полигона A и B. В структурата на списъка за даден връх е необходимо да се включи името му.

Ако координатите на точки $A_0 = list(x_0, y_0)$, и $B_0 = list(x_0, y_0)$ съвпадат по абсолютна стойност, това не ги прави еднакви, тъй като се работи само с имената (индексите) на точките. Точка с едни и същи координати може да бъде в два различни списъка.

В случая на полигон A всеки връх е получил име A плюс индекса му. Индексът се получава от подредбата в списъка. Координатите може да се добавят в списъка, но може да бъдат записани в отделен списък, асоциирани с името на всеки връх (асоциативен списък). По този начин търсенето ще става по името на върха. Този вариант не е за предпочитане при компютърна реализация тъй като изисква повече памет и допълнително търсене в друг списък.

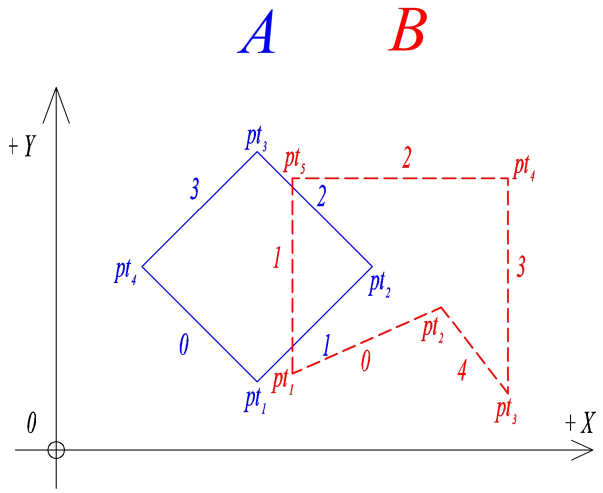
Индекс	Име	X	Y
1	2	3	4
0	A_0	x_0	y_0
1	A_1	x_1	y_1
2	A_2	x_2	y_2
...	...	...	...
n	A_n	x_n	y_n

Таблица 2.3: Структура на полигон A

Индекс	Име	X	Y
1	2	3	4
0	B_0	x_0	y_0
1	B_1	x_1	y_1
2	B_2	x_2	y_2
...	...	...	...
n	B_n	x_n	y_n

Таблица 2.4: Структура на полигон B

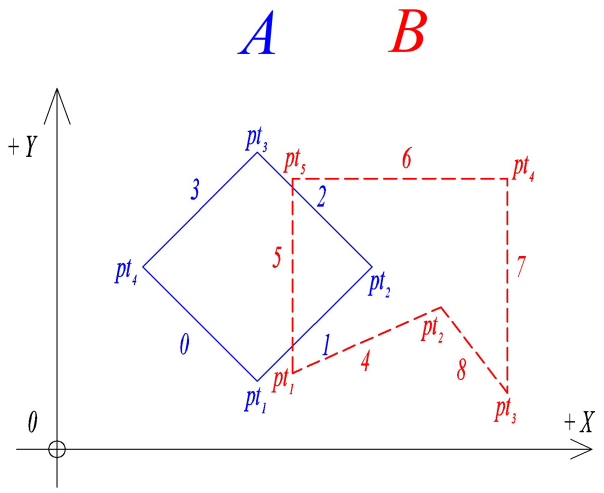
В общия случай трябва да се реши задачата за пресичането на n на брой сегменти. Затова може да обединим двата списъка от сегменти на полигоните A и B. Тук трябва да обърнем внимание, автоматичното събиране на двата списъка ще доведе до неправилно изпълнение на алгоритъма показан по долу. Както се вижда в таблици 2.3 и 2.4 имаме повтарящи се индекси. Това не се допуска за правилната работа на алгоритъма. Функцията, която преобразува списъка от таблиците, трябва да има атрибут (параметър) за начален индекс на сегментите. Ако започваме с първия полигон A, то индексът ще бъде 0. При втория полигон B индексът на сегментите трябва да започне от $(length.pointList.A)$. Тук начален индекс за B не добавяме $+1$ в индекса за B, защото броенето на върховете на полигон A е започнал от 0, а дължината на списъка започва от 1. При даден полигон с един сегмент дължината на списъка е 1, а индексът на списъка започва от 0. В таблица 2.5 i_A е индексът на съответния сегмент. Следователно $i_A = (length.pointList.A - 1)$. На фигура 2.40 е дадено примерно състояние на индексите на два полигона A и B. На фигура 2.41 е дадено крайното състояние на индексите на полигони A и B. Обединяването на двата полигона са дадени в таблица 2.5.



Фигура 2.40: Начално състояние на индексите на полигони A и B.

Да съставим списък на сегментите показан на фигура 2.42. Това ще бъде входящият списък за проверка дали сегментите се пресичат.

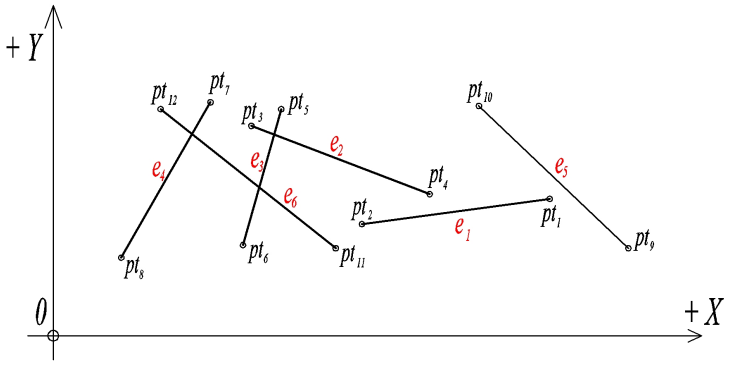
Във входящия списък за всеки един сегмент трябва да направим проверка дали координата x на първия връх има по-малка стойност от на координата x от втория връх. Например, за сегмент e_1, ако $x_1 < x_2$, то подредбата си остава v_1, v_2 иначе



Фигура 2.41: Начално състояние на индексите на полигони A и B.

Сегмент	Име	X	Y	Име	X	Y
1	2	3	4	5	6	7
0	A_0	x_0	y_0	A_1	x_1	y_1
1	A_1	x_1	y_1	A_2	x_2	y_2
2	A_2	x_2	y_2	A_3	x_3	y_3
...						
i_A	A_{i-1}	x_{i-1}	y_{i-1}	A_i	x_i	y_i
i_A+1	B_0	x_0	y_0	B_1	x_1	y_1
i_A+2	B_1	x_1	y_2	B_2	x_2	y_2
i_A+3	B_2	x_2	y_2	B_3	x_3	y_3
...	...	...	...	...	...	...
i_A+i_B	B_{i-1}	x_0	y_0	B_i	x_1	y_1

Таблица 2.5: Структура на пълния списък за Полигони A и B



Фигура 2.42: Пресичане на сегменти от полигони A и B.

v_2, v_1. Така новият списък добива следния вид: считаме, че има събитие, когато вертикалната сканираща линия премине през връх на сегмент. Следователно, ако имаме n сегмента ще получим $2n$ събития.

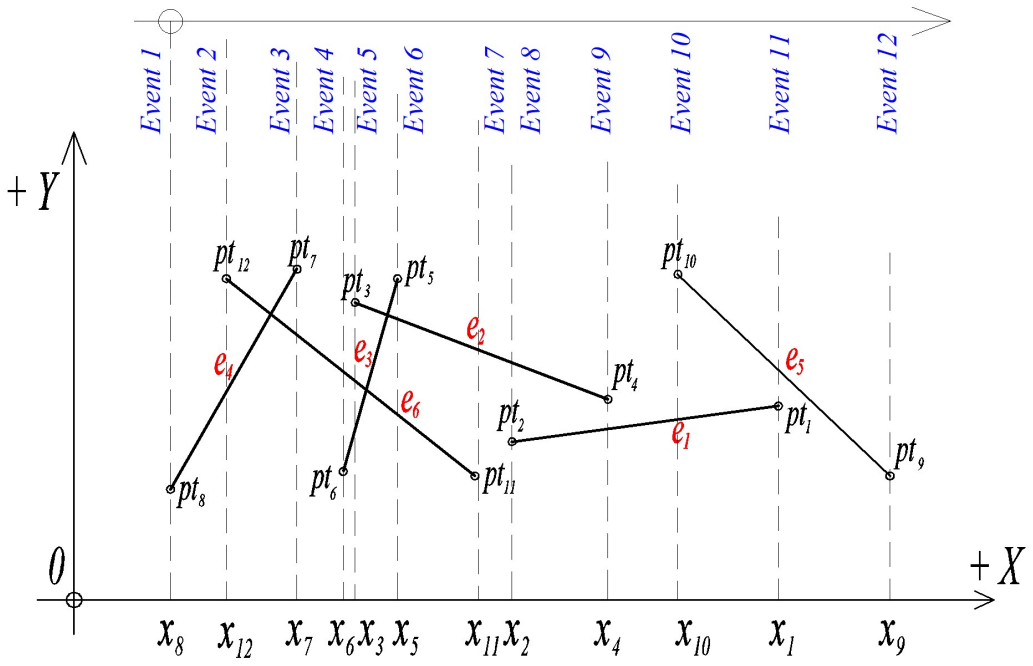
За да запишем всички събития е необходимо да преминем през целия списък

Сегмент	$Vertex_{begin}$	X_{begin}	Y_{begin}	$Vertex_{end}$	X_{end}	Y_{end}
1	2	3	4	5	6	7
e_1	v_1	x_1	y_1	v_2	x_2	y_2
e_2	v_3	x_3	y_3	v_4	x_4	y_4
e_3	v_5	x_5	y_5	v_6	x_6	y_6
e_4	v_7	x_7	y_7	v_8	x_8	y_8
e_5	v_9	x_9	y_9	v_{10}	x_{10}	y_{10}
e_6	v_{11}	x_{11}	y_{11}	v_{12}	x_{12}	y_{12}

Таблица 2.6: Входящ списък от сегменти

Сегмент	$Vertex_{begin}$	X_{begin}	Y_{begin}	$Vertex_{end}$	X_{end}	Y_{end}
1	2	3	4	5	6	7
e_1	v_2	x_2	y_2	v_1	x_1	y_1
e_2	v_3	x_3	y_3	v_4	x_4	y_4
e_3	v_6	x_6	y_6	v_5	x_5	y_5
e_4	v_8	x_8	y_8	v_7	x_7	y_7
e_5	v_{10}	x_{10}	y_{10}	v_9	x_9	y_9
e_6	v_{12}	x_{12}	y_{12}	v_{11}	x_{11}	y_{11}

Таблица 2.7: Ориентиран списък от сегменти по $x_i < x_{i+1}$



Фигура 2.43: Верикална сканираща линия

и като резултат ще получим структурата показана в таблица 2.8. Тази таблица има три колонки. В първата е записана координата по x а във втората колонка е записана информация дали координата x е начало или край на сегмент. Можем да въведем следните правила:

1. За начало на сегмент даваме стойност 0;

2. За край на сегмент даваме стойност 1;

3. Когато сегментът е вертикален, то x-координатите на началото и краят съвпадат и тогава записваме 2.

Информацията за попълването на тази таблица се получава от таблица 2.7. Трябва да отбележим, че за всеки елемент трябва да проверим дали елемента е вертикален. Затова се сравняват двете x-координати на върховете на сегмента като се използува размитост от вертикалата по формула (2.31):

$$|(x_1 - x_2)| \leq fuzz. \tag{2.31}$$

Тук x_1 е x координата на първия връх на сегмента, x_2 е x-координата на втория връх на сегмента. Третата колонка показва на кой сегмент принадлежи координата x.

Събитие	Begin/End	От сегмент
1	2	3
x_2	0	e_1
x_1	1	e_1
x_3	0	e_2
x_4	1	e_2
x_6	0	e_3
x_5	1	e_3
x_8	0	e_4
x_7	1	e_4
x_{10}	0	e_5
x_9	1	e_5
x_{12}	0	e_6
x_{11}	1	e_6

Таблица 2.8: Списък от събития

След съставянето на списъка от таблица 2.8 прилагаме сортировка по критерия $x_i < x_{i+1}$.

Събитие	Begin/End	От сегмент
1	2	3
x_8	0	e_4
x_{12}	0	e_6
x_7	1	e_4
x_6	0	e_3
x_3	0	e_2
x_5	1	e_3
x_{11}	1	e_6
x_2	0	e_1
x_4	1	e_2
x_{10}	0	e_5
x_1	1	e_1
x_9	1	e_5

Таблица 2.9: Сортиран списък от събития по критерии $x_i < x_{i+1}$

Преминаването на вертикалната сканираща линия означава обхождане на списъка на таблица 2.9 в дадената последователност. Отваряме нов празен списък за събитията, който е наречен динамичен списък. В този списък се записват или отписват съответните сегменти от таблица 2.7 в зависимост от следните критерии: когато събитието има стойност 0, тогава записваме, когато е 1, отписваме (втората колонка на таблица 2.9).

Връзката между сортирания списък от събития от таблица 2.9 и ориентирания списък от сегменти от таблица 2.7 е името на сегмента. При всяко записване на нов сегмент в динамичния списък правим проверка дали новия сегмент не се пресича със сегментите записани в него. Тук по наш избор можем да направим една от следните две проверки: 1. Намиране на реалните точки на пресичане; 2. Логическо пресичане. Тъй като намирането на точка на пресичане е "скъпа" процедура от гледна точка на процесорно време, ще се търси само логическото пресичане с резултат $True$ или $False$. Ще добавим и още едно подобрение: алгоритъмът да спре при намиране на първата потенциална пресечна точка. Ако има поне една пресечна точка, то двата полигона се пресичат. Под пресечна точка разбираме тривиалния случай на пресичане, виж фигура 2.23, случай (b).

Алгоритъмът работи по следния начин. Записваме първия елемент в новия списък. Тъй като $Begin/End$ е със стойност 0. Правим проверка за пресичане на новия елемент e_4 със списъка от сегменти. В случая списъкът е празен.

Събитие	Begin/End	От сегмент
x_8	0	e_4

Таблица 2.10: Динамичен списък: Първо събитие

След това вземаме втория елемент и го добавяме в новия списък. Тъй като $Begin/End$ е със стойност 0. Правим проверка за пресичане на новия елемент e_6 със списъка от сегменти.

Събитие	Begin/End	От сегмент
x_{12}	0	e_6
x_8	0	e_4

Таблица 2.11: Динамичен списък: Второ събитие

По-нататък, вземаме третия елемент и вадим сегмент e_4. Тъй като $Begin/End$ е със стойност 1, не правим проверка за пресичане със списъка от сегменти.

Събитие	Begin/End	От сегмент
x_{12}	0	e_6

Таблица 2.12: Динамичен списък: Трето събитие

Продължавайки, вземаме четвъртия елемент и го добавяме в новия списък. Тъй като $Begin/End$ е със стойност 0, то правим проверка за пресичане на новия елемент e_3 със списъка от сегменти.

Вземаме петия елемент и го добавяме в новия списък. Тъй като $Begin/End$ е със стойност 0, то правим проверка за пресичане на новия елемент e_2 със списъка от сегменти.

Вземаме шестия елемент и го изваждаме от новия списък. Тъй като $Begin/End$ е със стойност 1, не правим проверка за пресичане на новия елемент e_3 със списъка от сегменти.

Събитие	Begin/End	От сегмент
x_6	0	e_3
x_{12}	0	e_6

Таблица 2.13: Динамичен списък: Четвърто събитие

Събитие	Begin/End	От сегмент
x_3	0	e_2
x_6	0	e_3
x_{12}	0	e_6

Таблица 2.14: Динамичен списък: Пето събитие

Събитие	Begin/End	От сегмент
x_3	0	e_2
x_{12}	0	e_6

Таблица 2.15: Динамичен списък: Шесто събитие

Вземаме седмия елемент и го изваждаме от новия списък. Тъй като $Begin/End$ е със стойност 1, то не правим проверка за пресичане на новия елемент e_6 със списъка от сегменти.

Събитие	Begin/End	От сегмент
x_{11}	0	e_2

Таблица 2.16: Динамичен списък: Седмо събитие

Вземаме осмия елемент и го добавяме в новия списък. Тъй като $Begin/End$ е със стойност 0, то правим проверка за пресичане на новия елемент e_1 със списъка от сегменти.

Събитие	Begin/End	От сегмент
x_2	0	e_1
x_3	0	e_2

Таблица 2.17: Динамичен списък: Осмо събитие

Вземаме деветия елемент и го изваждаме от новия списък. Тъй като $Begin/End$ е със стойност 1, не правим проверка за пресичане на новия елемент e_2 със списъка от сегменти.

Събитие	Begin/End	От сегмент
x_2	0	e_1

Таблица 2.18: Динамичен списък: Девето събитие

Вземаме десетия елемент и го добавяме в новия списък. Тъй като $Begin/End$ е със стойност 0, то правим проверка за пресичане на новия елемент e_5 със списъка от сегменти.

Вземаме единадесетия елемент и го вадим от новия списък. Тъй като $Begin/End$ е със стойност 1, не правим проверка за пресичане на новия елемент e_1 със списъка от сегменти.

Събитие	Begin/End	От сегмент
x_{10}	0	e_5
x_2	0	e_1

Таблица 2.19: Динамичен списък: Десето събитие

Събитие	Begin/End	От сегмент
x_{10}	0	e_5

Таблица 2.20: Динамичен списък: Единадесето събитие

Накрая, вземаме дванадесетия елемент и го изваждаме от новия списък. Тъй като *Begin/End* е със стойност 1, не правим проверка за пресичане на новия елемент e_5 със списъка от сегменти.

Събитие	Begin/End	От сегмент

Таблица 2.21: Динамичен списък: Дванадесето събитие

Както се вижда списъкът с елементи, които трябва да се проверят, има максимум два елемента. Асимптотичната сложност на алгоритъма е $\mathcal{O}(n \log n)$. Даже, ако сегментите нямат никакво пресичане сложността ще бъде $\mathcal{O}(n)$. Ако сегментите имат координати както е показано на фигура 2.39, то сложността ще бъде $\mathcal{O}(n^2)$.

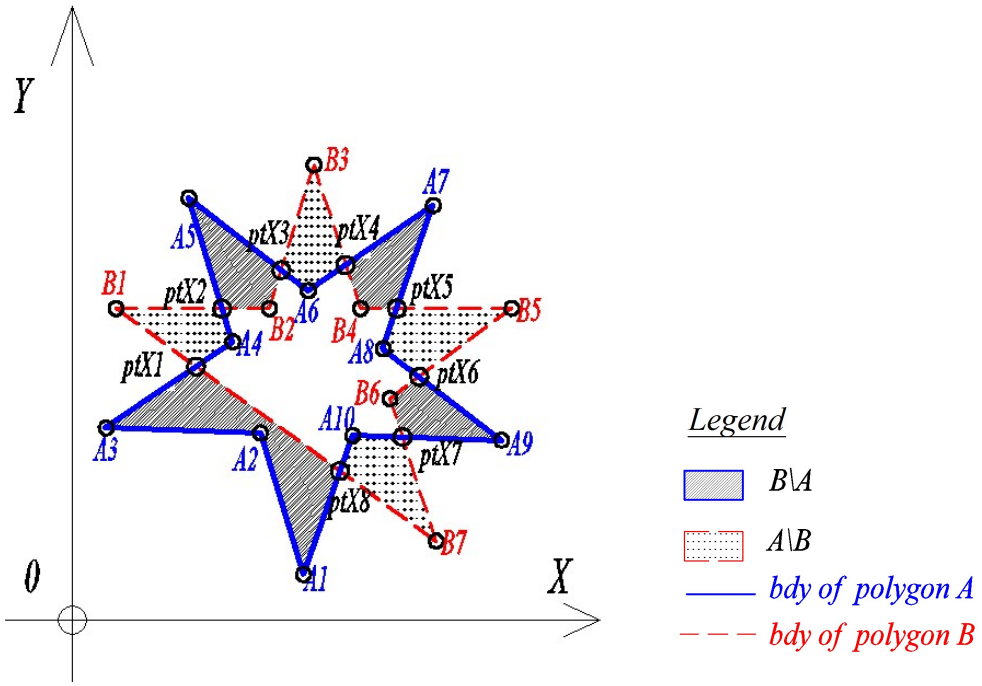
2.6 Операции с полигони като множества

Съгласно дадената дефиниция, полигонът е затворено ограничено множество в равнината. Тогава за дадени две множества може да се възползваме от дефиниции за пресичане, събиране и изваждане на множествата. $A \cap B$, $A \cup B$, $A \setminus B$, $B \setminus A$. Въпреки, че разглеждаме само едносвързани полигони A и B резултатът от пресичането може да бъде доста сложна многосвързана област. Алгоритъмът, които предлагаме, намира всички подобласти. Това е илюстрирано на фигура 2.6, където $A \setminus B$ има 4 подобласти и $B \setminus A$ има четири подобласти.

Ще решим задачата за изваждане на два полигона A и B в най-общо положение, [12], като пример, виж фигура 2.6. Ще представим едно решение на задачата като съставим логическа таблица за двата полигона.

Изваждане на два полигона (математически $A \setminus B$ или практически изрязване на един полигон B от A) е сложна и трудоемка задача. В научната литература има методи за пресичане на правоъгълни полигони [64]. Съществуват решения на задачата чрез приближение на полигоните със сума от правоъгълници [66]. Но в повечето случаи това решение не е ефективно и при прилагането му в разкроя води до голяма фира (загуба на материал). Тъй като това е базова операция при прилагане на предложения алгоритъм в производството на стоманени планки, ще опишем подробно метода за изваждане на два полигона или практически изрязване на даден полигон от друг полигон.

Двата полигона A и B, съгласно Секция **??**, са зададени със циклични списъци от подредени върхове $A = list(A_0, A_1, \ldots, A_n)$ и $B = list(B_0, B_1, \ldots, B_n)$ в ортогонална координатната система XOY. Предложеният тук алгоритъм за намиране на множеството $A \setminus B$ използва следните понятия/операции:



Фигура 2.44: Изрязване на два полигона A и B: $A \setminus B$ или $B \setminus A$

1. Ориентация на полигона A по часовата стрелка CW и ориентация на полигона B обратно на часовата стрелка $anti - CW$;

2. Намиране на пресечни точки на границите на двата полигона;

3. Проверка дали дадена точка е в даден полигон.

Намирането на $B \setminus A$ се решава като се разменят местата на полигоните в таблица 2.22.

Проверка на ориентацията на полигоните A и B.

Първо е необходимо да се проверят посоките на ориентация на полигоните A и B. Ако полигон A е ориентиран по посока на часовниковата стрелка (CW), то полигон B трябва да бъде ориентиран обратно на часовниковата стрелка ($anti-CW$). Начинът за проверка на посока на полигон е даден в подсекция 2.1. Смяната на посоката на даден полигон се изразява с начина на подреждане на върховете в списъка определящ полигона. Ако списъкът от върхове $list(pt_0, pt_1, \ldots, pt_n)$ е ориентиран по CW, то смяната на посоката се осъществява като списъкът се запише в обратен ред $list(pt_n, pt_{n-1}, \ldots, pt_0)$. Тези списъци са основна част от логическата таблица.

Намиране на пресечните точки на границите на два полигона A и B.

Ще илюстрираме изложения подход за случая показан на фигура 2.6, изобразяваща пресичане на два полигона $A = list(A_1, A_2, \ldots, A_{10})$ и $B = list(B_1, B_2, \ldots, B_7)$. За целта ще обходим всички сегменти от границата на полигон A и проверим дали се пресичат със сегментите от полигон B. За нагледност тук ще представим алгоритъма с метода на изчерпване на всички комбинации. Това ще рече да намерим

пресечните точки на границите на два полигона чрез изчерпане на всички комбинации за пресичане на сегментите им. Ще напомним, че сегментите са зададени като списък от два последователни върха – $list(A_i, A_{i+1})$ и $list(B_i, B_{i+1})$.

Пресечните точки на двата полигона от фигура 2.6 намираме по следната схема. Започваме с първи сегмент на полигон $list(A_1, A_2)$ и търсим последователно пресечните точки със сегментите на полигон B започвайки с първия сегмент $list(B_1, B_2)$. Същата процедура правим и за останалите сегменти на полигон A. Резултатите са показани по-долу.

Списъкът с пречените точки на границите на полигоните A и B, показани на фигура 2.6, е:

1. $list(A_1, A_2) \cap list(B_1, B_2) \implies \emptyset$;
2. $list(A_1, A_2) \cap list(B_2, B_3) \implies \emptyset$;
3. $list(A_1, A_2) \cap list(B_3, B_4) \implies \emptyset$;
4. $list(A_1, A_2) \cap list(B_4, B_5) \implies \emptyset$;
5. $list(A_1, A_2) \cap list(B_5, B_6) \implies \emptyset$;
6. $list(A_1, A_2) \cap list(B_6, B_7) \implies \emptyset$;
7. $list(A_1, A_2) \cap list(B_7, B_1) \implies \emptyset$;
8. $list(A_2, A_3) \cap list(B_1, B_2) \implies \emptyset$;
9. $list(A_2, A_3) \cap list(B_2, B_3) \implies \emptyset$;
10. $list(A_2, A_3) \cap list(B_3, B_4) \implies \emptyset$;
11. $list(A_2, A_3) \cap list(B_4, B_5) \implies \emptyset$;
12. $list(A_2, A_3) \cap list(B_5, B_6) \implies \emptyset$;
13. $list(A_2, A_3) \cap list(B_6, B_7) \implies \emptyset$;
14. $list(A_2, A_3) \cap list(B_7, B_1) \implies \emptyset$;
15. $list(A_3, A_4) \cap list(B_1, B_2) \implies \emptyset$;
16. $list(A_3, A_4) \cap list(B_2, B_3) \implies \emptyset$;
17. $list(A_3, A_4) \cap list(B_3, B_4) \implies \emptyset$;
18. $list(A_3, A_4) \cap list(B_4, B_5) \implies \emptyset$;
19. $list(A_3, A_4) \cap list(B_5, B_6) \implies \emptyset$;
20. $list(A_3, A_4) \cap list(B_6, B_7) \implies \emptyset$;
21. $list(A_3, A_4) \cap list(B_7, B_1) \implies pt_{X1}$;
22. $list(A_4, A_5) \cap list(B_1, B_2) \implies pt_{X2}$;
23. $list(A_4, A_5) \cap list(B_2, B_3) \implies \emptyset$;
24. $list(A_4, A_5) \cap list(B_3, B_4) \implies \emptyset$;

25. $list(A_4, A_5) \cap list(B_4, B_5) \implies \emptyset$;

26. $list(A_4, A_5) \cap list(B_5, B_6) \implies \emptyset$;

27. $list(A_4, A_5) \cap list(B_6, B_7) \implies \emptyset$;

28. $list(A_4, A_5) \cap list(B_7, B_1) \implies \emptyset$;

29. $list(A_5, A_6) \cap list(B_1, B_2) \implies \emptyset$;

30. $list(A_5, A_6) \cap list(B_2, B_3) \implies pt_{X3}$;

31. $list(A_5, A_6) \cap list(B_3, B_4) \implies \emptyset$;

32. $list(A_5, A_6) \cap list(B_4, B_5) \implies \emptyset$;

33. $list(A_5, A_6) \cap list(B_5, B_6) \implies \emptyset$;

34. $list(A_5, A_6) \cap list(B_6, B_7) \implies \emptyset$;

35. $list(A_5, A_6) \cap list(B_7, B_1) \implies \emptyset$;

36. $list(A_6, A_7 \cap list(B_1, B_2) \implies \emptyset$;

37. $list(A_6, A_7) \cap list(B_2, B_3) \implies \emptyset$;

38. $list(A_6, A_7) \cap list(B_3, B_4) \implies pt_{X4}$;

39. $list(A_6, A_7) \cap list(B_4, B_5) \implies \emptyset$;

40. $list(A_6, A_7) \cap list(B_5, B_6) \implies \emptyset$;

41. $list(A_6, A_7) \cap list(B_6, B_7) \implies \emptyset$;

42. $list(A_6, A_7) \cap list(B_7, B_1) \implies \emptyset$;

43. $list(A_7, A_8) \cap list(B_1, B_2) \implies \emptyset$;

44. $list(A_7, A_8) \cap list(B_2, B_3) \implies \emptyset$;

45. $list(A_7, A_8) \cap list(B_3, B_4) \implies \emptyset$;

46. $list(A_7, A_8) \cap list(B_4, B_5) \implies pt_{X5}$;

47. $list(A_7, A_8) \cap list(B_5, B_6) \implies \emptyset$;

48. $list(A_7, A_8) \cap list(B_6, B_7) \implies \emptyset$;

49. $list(A_7, A_8) \cap list(B_7, B_1) \implies \emptyset$;

50. $list(A_8, A_9) \cap list(B_1, B_2) \implies \emptyset$;

51. $list(A_8, A_9) \cap list(B_2, B_3) \implies \emptyset$;

52. $list(A_8, A_9) \cap list(B_3, B_4) \implies \emptyset$;

53. $list(A_8, A_9) \cap list(B_4, B_5) \implies \emptyset$;

54. $list(A_8, A_9) \cap list(B_5, B_6) \implies pt_{X6}$;

55. $list(A_8, A_9) \cap list(B_6, B_7) \implies \emptyset$;

56. $list(A_8, A_9) \cap list(B_7, B_1) \implies \emptyset$;

57. $list(A_9, A_{10}) \cap list(B_1, B_2) \implies \emptyset$;

58. $list(A_9, A_{10}) \cap list(B_2, B_3) \implies \emptyset$;

59. $list(A_9, A_{10}) \cap list(B_3, B_4) \implies \emptyset$;

60. $list(A_9, A_{10}) \cap list(B_4, B_5) \implies \emptyset$;

61. $list(A_9, A_{10}) \cap list(B_5, B_6) \implies \emptyset$;

62. $list(A_9, A_{10}) \cap list(B_6, B_7) \implies pt_{X7}$;

63. $list(A_9, A_{10}) \cap list(B_7, B_1) \implies \emptyset$;

64. $list(A_{10}, A_1) \cap list(B_1, B_2) \implies \emptyset$;

65. $list(A_{10}, A_1) \cap list(B_2, B_3) \implies \emptyset$;

66. $list(A_{10}, A_1) \cap list(B_3, B_4) \implies \emptyset$;

67. $list(A_{10}, A_1) \cap list(B_4, B_5) \implies \emptyset$;

68. $list(A_{10}, A_1) \cap list(B_5, B_6) \implies \emptyset$;

69. $list(A_{10}, A_1) \cap list(B_6, B_7) \implies \emptyset$;

70. $list(A_{10}, A_1) \cap list(B_7, B_1) \implies pt_{X8}$;

Пресечните точки са 8 на брой. Трябва да отбележим, че съществено значение има номерирането на пресечните точки. Не е допустимо намирането на пресечните точки и след това да им бъде даден индекс на тяхната последователност, защото истинската им последователност може да бъде нарушена. Виж фигура 2.6 и таблица 2.22.

Съставяне на логическата таблица за два полигона A и B.

Сега ще се състави една таблица, в която има полезна информация за двата полигона. Ще се нарече логическа таблица. Полигон A ще бъде основен (базов, стоманен лист), т.е. от него ще се изрязва. Полигон B ще бъде планка (нож), която ще се изрязва от полигон A. Съставя се таблица от гледна точка на полигон A към полигон B. Ако искаме да се изреже от полигон B полигон A, то тогава трябва да си сменят местата. B да бъде основен, Aбъде планка (нож), която ще се изрязва от полигон B. Има значение кой полигон е основен и кой се изрязва. Таблицата съдържа следните колони:

1. Индекс: Индекс на точките $0, 1, 2, \ldots, n$. Започване с индекс 0 заради удобство с програмиране със списъци. Индексът се получава директно от подредбата в списъка на съответния полигон;

2. Име: Етикет на всяка точка $A_1, A_2, pt_{X1}, \ldots$ и т.н. Чрез поставянето на имена на точките ние даваме информация дали точката от списъка е връх или пресечна точка на границите на полигоните. Пресечните точки са ключови при формирането на таблиците, тъй като те превключват в търсенето в таблиците;

3. X, Y: Координати на всяка точка;

4. Вън от A (B): Атрибут за позицията на точката спрямо другия полигон – "True" за вътре и "False" за вън;

5. Проверка: Атрибут за това дали точката е преминала през оценка на алгоритъма – в началото всички точки имат атрибут "NoPass", когато алгоритъмът премине през точката тогава променя атрибута й на "Pass";

6. Подобласт (на $A \setminus B$) : Индекс за номера на подобластите на $A \setminus B$.

Съставяне на логическата таблица 2.22 за полигон A. Отваряме празна логическа таблица, където ще бъде записана новата информация. Колонка 5 в начално състояние винаги е $NoPass$. Колонка 6 с подобластите се запълва в процеса на работа на алгоритъма. Започваме да запълваме по редове. С индекс 0 поставяме точка A_1 в колона 2, в колона 3 поставяме координатите й. В колона 4 проверяваме дали точка A_1 е вътре в полигона B. Ако е вътре то даваме стойност $True$, иначе $False$. Попълваме следващия ред . Проверяваме дали в сегмента A_1, A_2 има регистрирана пресечна точка с границата на полигона B. Това са редове от 1 до 7 на списъка на пресечните точки. В случая няма да можем да запишем точка A_2 в колонка 2, а в колонка 3 слагаме нейните координати. В колона 4 поставяме $True$ тъй като A_2 не е полигон B. В случая не слагаме $True$. Същото се получава и за точка A_3. С индекс 3 проверяваме дали сегмента A_3, A_4 Има пресечната точка с границата на полигона B. От редове 15 до 21 от списъка на пресечните точки виждаме, че такава пресечна точка има – pt_{X1}. В логическата таблица под индекс 3 слагаме в колона 2 точката pt_{X1}, в колона 3 слагаме нейните координати. В колонка 4 поставяме $Flase$ тъй като тази точка е в полигон B. Продължаваме с поставянето на точка A_4. Процедурата се повтаря до изчерпването на списъка от полигон A. Попълването на таблицата за полигон се изчерпва с направените проверка за сегмент $list(A_{10}, A_1)$ от редове 64 до 70 на списъка на пресечните точки. Виждаме, че има пресечна точка и тя става последна в списъка за полигона A.

Аналогично е попълването на графите за полигон B в таблица 2.22.

Таблица 2.22: Логическа таблица за полигони A и B показани на фигура 2.6

Полигон A						Полигон B					
Индекс	Име	X, Y	Вън от B	Проверка	Домейн	Индекс	Име	X, Y	Вън от A	Проверка	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
0	A_1	X, Y	True	NoPass		0	B_1	X, Y	True	NoPass	
1	A_2	X, Y	True	NoPass		1	pt_{X1}	X, Y	False	NoPass	
2	A_3	X, Y	True	NoPass		2	pt_{X8}	X, Y	False	NoPass	
3	pt_{X1}	X, Y	False	NoPass		3	B_7	X, Y	True	NoPass	
4	A_4	X, Y	False	NoPass		4	pt_{X7}	X, Y	False	NoPass	
5	pt_{X2}	X, Y	False	NoPass		5	B_6	X, Y	False	NoPass	
6	A_5	X, Y	True	NoPass		6	pt_{X6}	X, Y	False	NoPass	
7	pt_{X3}	X, Y	False	NoPass		7	B_5	X, Y	True	NoPass	
8	A_6	X, Y	False	NoPass		8	pt_{X5}	X, Y	False	NoPass	
9	pt_{X4}	X, Y	False	NoPass		9	B_4	X, Y	False	NoPass	
10	A_7	X, Y	True	NoPass		10	pt_{X4}	X, Y	False	NoPass	
11	pt_{X5}	X, Y	False	NoPass		11	B_3	X, Y	True	NoPass	
12	A_8	X, Y	False	NoPass		12	pt_{X3}	X, Y	False	NoPass	
13	pt_{X6}	X, Y	Fasle	NoPass		13	B_2	X, Y	False	NoPass	
14	A_9	X, Y	True	NoPass		14	pt_{X2}	X, Y	False	NoPass	
15	pt_{X7}	X, Y	False	NoPass							
16	A_{10}	X, Y	False	NoPass							
17	pt_{X8}	X, Y	False	NoPass							

Пресечните точки принадлежат и на двата полигона A и B. Това означава две възможни логически стойности – "True" или "False", но по дефиниция полигона е затворено множество от точки. Затова на пресечната точка се дава статус "False", т.е. вътре в полигона.

Взимаме първия възможен връх от полигон A, които е извън полигона B или има статус на четвърта колонка "Вън от $B = True$". След това разглеждаме следващата точка. Ако тя е пресечна (има "X" в името си), то трябва да превключим към таблица "Полигон B" и да продължим да записваме точките докато не стигнем пресечна точка и отново се върнем в таблица "Полигон A".

Сега да анализираме таблица 2.22 както следва:

1. Започваме циклична ротация на таблицата (спрямо полигон A или гледаме само точки от полигон A) докато намерим точка която е едновременно със статус "noPass" и стаус "Вън от B" = "True". В случая в таблицата това е втората колона;

2. Взимаме първата точка A_1 - и я записваме, защото не е пресечна точка и има статус "Вън от B" = True;

3. Вземаме следващата точка A_2 - и я записваме, защото не е пресечна точка и има статус Вън от B = True;

4. Вземаме следващата точка A_3 - и я записваме, защото не е пресечна точка и има статус Вън от B = True;

5. Не взимаме следващата точка pt_{X1} - превключваме на към дясната таблица на полигон B;

6. Намираме pt_{X1} от таблицата за полигон B (колонка 8). Записваме я. От тук нататък процеса продължава в таблица за полигон B ;

7. Не взимаме следващата точка pt_{X8} - превключваме към таблицата за полигон A и намираме точка pt_{X8} от таблица A;

8. Взимаме pt_{X8} от таблица A и я записваме. Активната таблица е на полигон A;

9. Взимаме следващата точка A_1. Тя съвпада с началната точка, с което цикъла завършва тук. Ще обърнем внимание, че работим с имена на точките а не с техните координати. Затова дали една точка съвпада с друга не зависи от нейните координати.

Този цикъл съдържа първата подобласт от сечението $list(A_1, A_3, pt_{X1}, pt_{X8}, A_1)$. В края на този цикъл ние може да дадем име на област "Domain 1". Методът трябва да се повтори докато няма точки от полигон A (втората колона), които са със статус "Вън от B" = "True" и статус "NoPass". Първата точка от полигон A е отбелязана с получерен шрифт.

Таблица 2.23: Резултат от процедурата след първото преминаване на алгоритъма

спрямо полигон A						спрямо полигон B					
Индекс	Име	X, Y	Вън от B	Проверка	Домейн	Индекс	Име	X, Y	Вън от A	Проверка	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
0	A_1	X, Y	True	Pass	**1**	0	B_1	X, Y	True	NoPass	
1	A_2	X, Y	True	Pass	1	1	pt_{X1}	X, Y	False	Pass	1
2	A_3	X, Y	True	Pass	1	2	pt_{X8}	X, Y	False	Pass	1
3	pt_{X1}	X, Y	False	Pass	1	3	B_7	X, Y	True	NoPass	
4	A_4	X, Y	False	NoPass		4	pt_{X7}	X, Y	False	NoPass	
5	pt_{X2}	X, Y	False	NoPass		5	B_6	X, Y	False	NoPass	
6	A_5	X, Y	True	NoPass		6	pt_{X6}	X, Y	False	NoPass	
7	pt_{X3}	X, Y	False	NoPass		7	B_5	X, Y	True	NoPass	
8	A_6	X, Y	False	NoPass		8	pt_{X5}	X, Y	False	NoPass	
9	pt_{X4}	X, Y	False	NoPass		9	$B4$	X, Y	False	NoPass	
10	A_7	X, Y	True	NoPass		10	pt_{X4}	X, Y	False	NoPass	
11	pt_{X5}	X, Y	False	NoPass		11	B_3	X, Y	True	NoPass	
12	A_8	X, Y	False	NoPass		12	pt_{X3}	X, Y	False	NoPass	
13	pt_{X6}	X, Y	Fasle	NoPass		13	B_2	X, Y	False	NoPass	
14	A_9	X, Y	True	NoPass		14	pt_{X2}	X, Y	False	NoPass	
15	pt_{X7}	X, Y	False	NoPass							
16	A_{10}	X, Y	False	NoPass							
17	pt_{X8}	X, Y	False	Pass	1						

На втората стъпка на алгоритъма игнорираме точките, които имат статус "Pass".

Таблица 2.24: Циклично ротиране на таблицата. Пас втори.

спрямо полигон A						спрямо полигон B					
Индекс	Име	X, Y	Вън от B	Проверка	Домейн	Индекс	Име	X, Y	Вън от A	Проверка	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
						0	B_1	X, Y	True	NoPass	
						1	pt_{X1}	X, Y	False	Pass	1
						2	pt_{X8}	X, Y	False	Pass	1
3	pt_{X1}	X, Y	False	Pass	1	3	B_7	X, Y	True	NoPass	
4	A_4	X, Y	False	NoPass		4	pt_{X7}	X, Y	False	NoPass	
5	pt_{X2}	X, Y	False	Pass	2	5	B_6	X, Y	False	NoPass	
6	A_5	X, Y	True	Pass	**2**	6	pt_{X6}	X, Y	False	NoPass	
7	pt_{X3}	X, Y	False	Pass	2	7	B_5	X, Y	True	NoPass	
8	A_6	X, Y	False	NoPass		8	pt_{X5}	X, Y	False	NoPass	
9	pt_{X4}	X, Y	False	NoPass		9	B_4	X, Y	False	NoPass	
10	A_7	X, Y	True	NoPass		10	pt_{X4}	X, Y	False	NoPass	
11	pt_{X5}	X, Y	False	NoPass		11	B_3	X, Y	True	NoPass	
12	A_8	X, Y	False	NoPass		12	pt_{X3}	X, Y	False	Pass	2
13	pt_{X6}	X, Y	Fasle	NoPass		13	B_2	X, Y	False	Pass	2
14	A_9	X, Y	True	NoPass		14	pt_{X2}	X, Y	False	Pass	2
15	pt_{X7}	X, Y	False	NoPass							
16	A_10	X, Y	False	NoPass							
17	pt_{X8}	X, Y	False	Pass	1						

Този цикъл съдържа втората подобласт, която се определя от списъка от точки $list(A_5, pt_{X3}, B_2, pt_{X2}, A_5)$. Алгоритъмът спира когато всички точки със атрибут "Вън от B" = "True" са с атрибут "Проверка = Pass".

Таблица 2.25: Таблица с крайните резултати при спиране на алгоритъма.

спрямо полигон A						спрямо полигон B					
Индекс	Име	X,Y	Вън от B	Проверка	Домейн	Индекс	Име	X,Y	Вън от A	Проверкаk	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
0	A_1	X,Y	True	Pass	**1**	0	B_1	X,Y	True	NoPass	
1	A_2	X,Y	True	Pass	1	1	pt_{X1}	X,Y	False	Pass	1
2	A_3	X,Y	True	Pass	1	2	pt_{X8}	X,Y	False	Pass	1
3	pt_{X1}	X,Y	False	Pass	1	3	B_7	X,Y	True	NoPass	
4	A_4	X,Y	False	NoPass		4	pt_{X7}	X,Y	False	Pass	4
5	pt_{X2}	X,Y	False	Pass	2	5	B_6	X,Y	False	Pass	4
6	A_5	X,Y	True	Pass	**2**	6	pt_{X6}	X,Y	False	Pass	4
7	pt_{X3}	X,Y	False	Pass	2	7	B_5	X,Y	True	NoPass	
8	A_6	X,Y	False	NoPass		8	pt_{X5}	X,Y	False	Pass	3
9	pt_{X4}	X,Y	False	Pass	3	9	B_4	X,Y	False	Pass	3
10	A_7	X,Y	True	Pass	**3**	10	pt_{X4}	X,Y	False	Pass	3
11	pt_{X5}	X,Y	False	Pass	3	11	$B3$	X,Y	True	NoPass	
12	A_8	X,Y	False	NoPass		12	pt_{X3}	X,Y	False	NoPass	2
13	pt_{X6}	X,Y	Fasle	Pass	4	13	B_2	X,Y	False	Pass	2
14	A_9	X,Y	True	Pass	**4**	14	ptX2	X,Y	False	Pass	2
15	pt_{X7}	X,Y	False	Pass	4						
16	A_{10}	X,Y	False	NoPass							
17	pt_{X8}	X,Y	False	Pass	1						

В таблица 2.25 може да получим информация за всички подобласти (изрязани области). Тук е представено прочитане на резултатите:

1. под-домейн 1 е описан от точки: $list(A_1, A_2, A_3, pt_{X1}, pt_{X8}, A_1)$;
2. под-домейн 2 е описан от точки: $list(A_5, pt_{X3}, B_2, pt_{X2}, A_5)$;
3. под-домейн 3 е описан от точки: $list(A_7, pt_{X5}, B_4, pt_{X4}, A_7)$;
4. под-домейн 4 е описан от точки: $list(A_9, pt_{X7}, B_6, pt_{X6}, A_9)$.

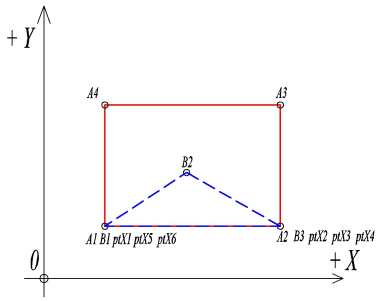
Тъй като подобластите са затворени множества, то те започват с даден връх и завършват със същия връх. Това може да служи като критерии за правилната работа на алгоритъма.

До тук показахме как алгоритъмът работи при изрязването на два полигона и получаване на повече от една подобласт (изрязъци). В практиката много често се срещат варианти за изваждане на два полигона A и B със съвпадащи се върхове. Възможните схеми на съвпадение на върхове на полигони са показани на фигури от 2.45 до 2.47.

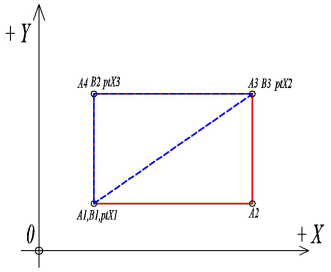
Да се спрем по-детайлно на фигура 2.47 при която има съвпадение на един от върховете на полигон A с някой от върховете на полигон B. Нека полигон $A = list(A_1, A_2, A_3, A_4)$ е зададен с 4 точки а полигон $B = list(B_1, B_2, B_3)$ – с 3 точки. Полигоните са дефинирани с техните координати в правоъгълна координатна система XOY. Виж фигура 2.47.

В случая полигон $A = list(A_1, A_2, A_3, A_4)$ ще бъде основен (от които ще вадим или на практика планката, от която ще изрязваме), а полигон $B = list(B_1, B_2, B_3)$ ще бъде изрязващ полигон (планката, която ще бъде изрязана). Целта е да извадим полигон B от полигон A при съвпадение (подравняване) на върхове $A_1 \equiv B_1$, без да прилагаме ротация или огледално трансформиране на полигон B. Получава се схемата показана на фигура 2.47.

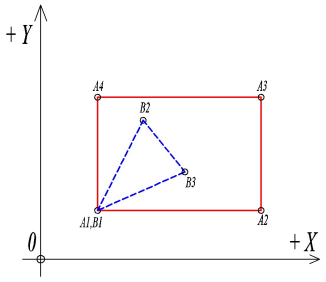
Тук трябва да обърнем внимание, че преди започване на построяването на логическата таблица посоката на ориентация на полигон A трябва да бъде различна от посоката на ориентация на полигон B. Това е така, защото ще прилагаме



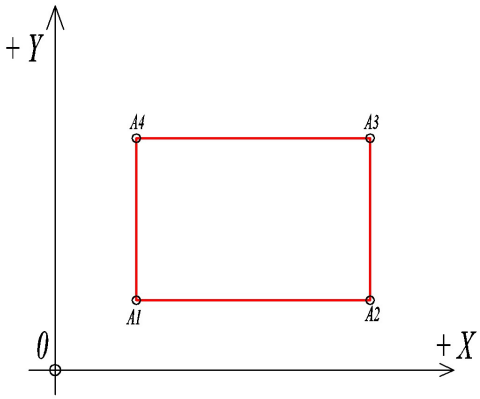
Фигура 2.45: Съвпадащи възли, Вариант 1.



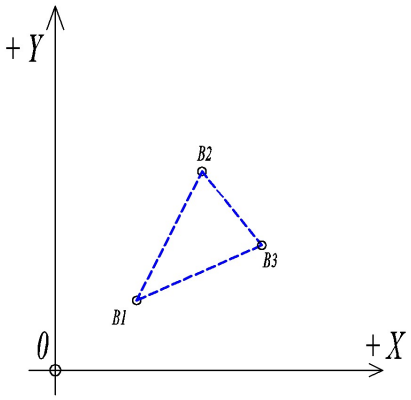
Фигура 2.46: Съвпадащи възли, Вариант 2.



Фигура 2.47: Съвпадащи възли, Вариант 3.

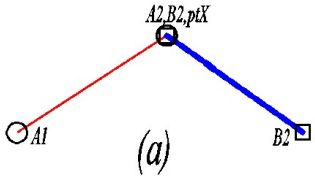


Фигура 2.48: Полигон A

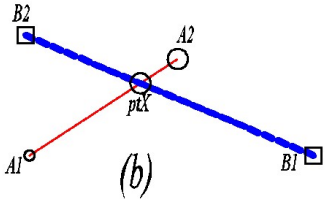


Фигура 2.49: Полигон B.

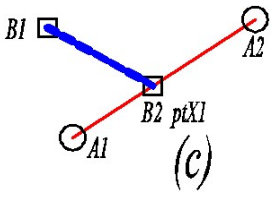
функцията изваждане. При събиране полигоните трябва да бъдат ориентирани в еднаква посока. При намирането на пресечна точка на два сегмента е нужно да проверим дали пресечната точка е вътре в сегментите, както и следните случаи на пресичане:



Фигура 2.50: Валидна пресечна точка, (a).



Фигура 2.51: Валидна пресечна точка, (b).



Фигура 2.52: Валидна пресечна точка, (c).

Нека да намерим пресечните точки за двата полигона.

1. $list(A_1, A_2) \cap list(B_1, B_2) \implies pt_{X1}$;
2. $list(A_1, A_2) \cap list(B_2, B_3) \implies \emptyset$;
3. $list(A_1, A_2) \cap list(B_3, B_1) \implies pt_{X2}$;
4. $list(A_2, A_3) \cap list(B_1, B_2) \implies \emptyset$;
5. $list(A_2, A_3) \cap list(B_2, B_3) \implies \emptyset$;
6. $list(A_2, A_3) \cap list(B_3, B_1) \implies \emptyset$;
7. $list(A_3, A_4) \cap list(B_1, B_2) \implies \emptyset$;
8. $list(A_3, A_4) \cap list(B_2, B_3) \implies \emptyset$;
9. $list(A_3, A_4) \cap list(B_3, B_1) \implies \emptyset$;
10. $list(A_4, A_1) \cap list(B_1, B_2) \implies pt_{X3}$;
11. $list(A_4, A_1) \cap list(B_2, B_3) \implies \emptyset$;
12. $list(A_4, A_1) \cap list(B_3, B_1) \implies pt_{X4}$;

Пресечните точки са 4 на брой. Съставяне на логическата таблицата.

Таблица 2.26: Логическа таблица за полигони A и B показани на фигура 2.47

Полигон A						Полигон B					
Индекс	Име	X, Y	Вън от B	Проверка	Домейн	Индекс	Име	X, Y	Вън от A	Проверка	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
1	A_1	X, Y	False	NoPass		1	B_1	X, Y	False	NoPass	
2	pt_{X1}	X, Y	False	NoPass		2	pt_{X1}	X, Y	False	NoPass	
3	pt_{X2}	X, Y	False	NoPass		3	pt_{X3}	X, Y	False	NoPass	
4	A_2	X, Y	True	NoPass		4	B_2	X, Y	False	NoPass	
5	A_3	X, Y	True	NoPass		5	B_3	X, Y	False	NoPass	
6	A_4	X, Y	True	NoPAss		6	pt_{X2}	X, Y	False	NoPass	
7	pt_{X3}	X, Y	False	NoPass		7	pt_{X4}	X, Y	False	NoPass	
8	pt_{X4}	X, Y	False	NoPass							

Ще отбележим, че и двата списъка от точки за полигони A и B от таблица 2.26 са циклични, това са втора и осма колони, което означава, че след достигне до последния индекс за съответния полигон алгоритъмът тук продължава от индекс 1. Сега трябва да намерим точка от таблицата за полигон A, за която едновременно са изпълнени условията на два параметъра Вън от $B = True$ и Проверка $= NoPass$. Ако няма такава точка алгоритъмът приключва тук. В случая имаме точка, която отговаря и на двата параметъра. Това е точка A_2. Това е началото на първата подобласт. При взимането на всяка точка сменяме статуса й "Проверка" на $Pass$. Сега последователно проверяваме и точки A_3, A_4. При точка pt_{X3} превключваме таблиците. Пресечната точка е индикаторът, че трябва да сменим таблиците. Сега сме в таблицата с полигон B. Взимаме точка pt_{X3} и точка B_2, B_3 стигаме до точка pt_{X2}. Превключваме в таблицата на полигон A. Взимаме точка pt_{X2}, след това точка A_2. Последната точка A_2 съвпада с първата точка A_2, с което получихме първата подобласт. Алгоритъмът не продължава, защото няма повече точки от полигон A, които да отговарят на двете условия за Вън от $B = True$ и Проверка $= NoPass$. В таблица 2.27 са дадени резултатите от работата на алгоритъма. В колоната Домейн е записано число разделено с точка. Първата

цифра е номера на подобласта. В случая едно, защото имаме една подобласт. Второто число е последователността на взимане на точките. Разгледаният алгоритъм е публикуван в [12].

Таблица 2.27: Логическа таблица за полигони A и B с един съвпадащ връх. Фигура 2.47. Крайни резултати.

Полигон A						Полигон B					
Индекс	Име	X, Y	Вън от B	Проверка	Домейн	Индекс	Име	X, Y	Вън от A	Проверка	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
1	A_1	X, Y	False	NoPass		1	B_1	X, Y	False	NoPass	
2	pt_{X1}	X, Y	False	NoPass		2	pt_{X1}	X, Y	False	NoPass	
3	pt_{X2}	X, Y	False	Pass	1.7	3	pt_{X3}	X, Y	False	Pass	1.4
4	A_2	X, Y	True	Pass	1.1	4	B_2	X, Y	False	Pass	1.5
5	A_3	X, Y	True	Pass	1.2	5	B_3	X, Y	False	Pass	1.6
6	A_4	X, Y	True	Pass	1.3	6	pt_{X2}	X, Y	False	NoPass	
7	pt_{X3}	X, Y	False	NoPass		7	pt_{X4}	X, Y	False	NoPass	
8	pt_{X4}	X, Y	False	NoPass							

Списъкът от точки дава крайния резултат

$$A \setminus B = list(A_2, A_3, A_4, pt_{X3}, B_2, B_3, pt_{X2}, A_2).$$

Както се вижда при съвпадение на върховете на полигоните A и B се получава списък от върхове с еднакви координати. За да не съставят "тежки" полигони, например триъгълник описан с повече от три точки, то трябва на получения списък да се приложи функцията "Премахване на излишни точки", виж Секция 2.1. В конкретния случаи показан на фигура 2.47 редуцирането на списъка от точки ще бъде

$$A \setminus B = list(A_2, A_3, A_4, pt_{X3}, B_2, B_3, A_2).$$

Впоследствие, ще работим именно с този списък на получения полигон.

Сега да разгледаме друг много често срещан случай, съвпадение на два върха на полигоните A и B, виж фигура 2.45

Както винаги е намирането на пресечните точки на двата полигона.

1. $list(A_1, A_2) \cap list(B_1, B_2) \implies pt_{X1}$;
2. $list(A_1, A_2) \cap list(B_2, B_3) \implies pt_{X2}$;
3. $list(A_1, A_2) \cap list(B_3, B_1) \implies \emptyset$;
4. $list(A_2, A_3) \cap list(B_1, B_2) \implies \emptyset$;
5. $list(A_2, A_3) \cap list(B_2, B_3) \implies pt_{X3}$;
6. $list(A_2, A_3) \cap list(B_3, B_1) \implies pt_{X4}$;
7. $list(A_3, A_4) \cap list(B_1, B_2) \implies \emptyset$;
8. $list(A_3, A_4) \cap list(B_2, B_3) \implies \emptyset$;
9. $list(A_3, A_4) \cap list(B_3, B_1) \implies \emptyset$;
10. $list(A_4, A_1) \cap list(B_1, B_2) \implies pt_{X5}$;
11. $list(A_4, A_1) \cap list(B_2, B_3) \implies \emptyset$;
12. $list(A_4, A_1) \cap list(B_3, B_1) \implies pt_{X6}$;

Пресечните точки са 6 на брой. Сега съставяме логическата таблицата.

Таблица 2.28: Логическа таблица за полигони A и B показани на фигура 2.45. Първоначално състояние.

Полигон A						Полигон B					
Индекс	Име	X,Y	Вън от B	Проверка	Домейн	Индекс	Име	X,Y	Вън от A	Проверка	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
1	A_1	X,Y	False	NoPass		1	B_1	X,Y	False	NoPass	
2	pt_{X1}	X,Y	False	NoPass		2	pt_{X1}	X,Y	False	NoPass	
3	pt_{X2}	X,Y	False	NoPass		3	pt_{X5}	X,Y	False	NoPass	
4	A_2	X,Y	False	NoPass		4	B_2	X,Y	False	NoPass	
5	pt_{X3}	X,Y	False	NoPass		5	pt_{X2}	X,Y	False	NoPass	
6	pt_{X4}	X,Y	False	NoPass		6	pt_{X3}	X,Y	False	NoPass	
7	A_3	X,Y	True	NoPass		7	B_3	X,Y	False	NoPass	
8	A_4	X,Y	True	NoPAss		8	pt_{X4}	X,Y	False	NoPass	
9	pt_{X5}	X,Y	False	NoPass		9	pt_{X6}	X,Y	False	NoPass	
10	pt_{X6}	X,Y	False	NoPass							

Двата списъка от точки за полигони A и B от таблица 2.28 са циклични (при сегмент или отворен полигон списъкът не е цикличен). Това са втора и осма колонки. Намираме точка от таблицата на полигон A, за която едновременно са изпълнени условията на два параметъра "Вън от B" $= True$ и "Проверка" $= NoPass$. Ако няма такава точка алгоритъмът спира тук. В случая това е точка A_3. Това е началото на първата подобласт. При взимането на всяка точка сменяме статута й "Проверка" на $Pass$. Сега последователно взимаме и точки A_3, A_4. При точка pt_{X5} превключваме в таблицата на B. Сега сме в таблицата с полигон B. Взимаме точки pt_{X5} и B_2, стигаме до точка pt_{X2}. Превключваме в таблицата на полигона A. Взимаме точка pt_{X2}, след това точка A_2, стигаме до точка pt_{X3} и превключваме на таблицата от полигон B. Намираме точка pt_{X3}, взимаме точка pt_{X3}, след това B_3, след това превключваме на таблицата на полигонаA , взимаме точка pt_{X4} и точка A_3. Тук алгоритъмът спира, защото последната точка съвпада с първата. Ще обърнем внимание, че сравнението на точките става по име, а не по координати. Сравняването по име дава прецизност и скорост на алгоритъма за сравнение на точките. Алгоритъмът не продължава по-нататък, защото няма повече точки от полигон A, които да отговарят на двете условия за Вън от $B = True$ и Проерка $= NoPass$. В таблица 2.29 са дадени резултатите от работата на алгоритъма.

Таблица 2.29: Логическа таблица за полигони A и B с два съвпадащи върха. Фигура 2.45. Краен резултат.

Полигон A						Полигон B					
Индекс	Име	X,Y	Вън от B	Проверка	Домейн	Индекс	Име	X,Y	Вън от A	Проверка	Домейн
1	2	3	4	5	6	7	8	9	10	11	12
1	A_1	X,Y	False	NoPass		1	B_1	X,Y	False	NoPass	
2	pt_{X1}	X,Y	False	NoPass		2	pt_{X1}	X,Y	False	NoPass	
3	pt_{X2}	X,Y	False	NoPass	1.5	3	pt_{X5}	X,Y	False	NoPass	1.3
4	A_2	X,Y	False	NoPass	1.6	4	B_2	X,Y	False	NoPass	1.4
5	pt_{X3}	X,Y	False	NoPass		5	pt_{X2}	X,Y	False	NoPass	
6	pt_{X4}	X,Y	False	NoPass	1.9	6	pt_{X3}	X,Y	False	NoPass	1.7
7	A_3	X,Y	True	NoPass	1.1	7	B_3	X,Y	False	NoPass	1.8
8	A_4	X,Y	True	NoPAss	1.2	8	pt_{X4}	X,Y	False	NoPass	
9	pt_{X5}	X,Y	False	NoPass		9	pt_{X6}	X,Y	False	NoPass	
10	pt_{X6}	X,Y	False	NoPass							

Така списъкът от точки на крайния резултат е:

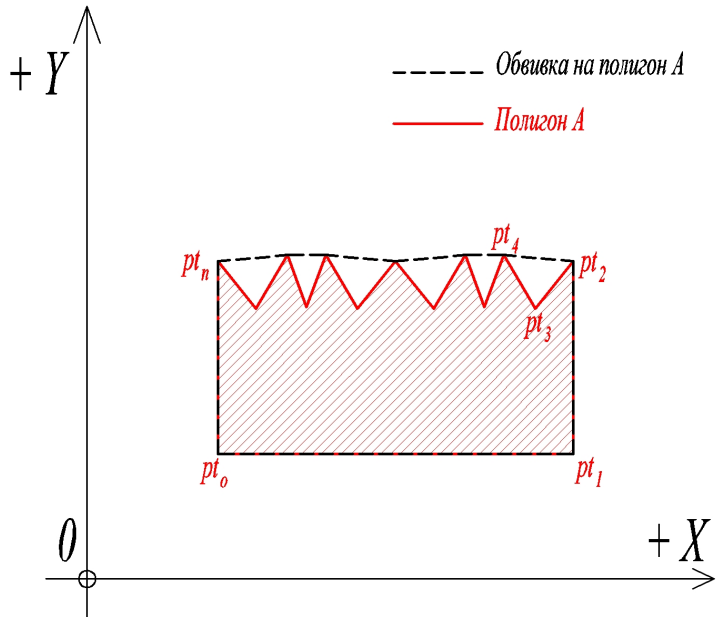
$$A \setminus B = list(A_3, A_4, pt_{X5}, B_2, A_2, pt_{X3}, B_3, pt_{X4}, A_3)$$

Както се вижда при съвпадение на върховете на полигоните A и B се получава списък от върхове с еднакви координати. За тяхното премахване се прилага функция "Премахване на излишни точки", виж Секция 2.1. В резултат на премахването на излишните точки получаваме следната характеризация на областта.

$$A \setminus B = list(A_3, A_4, pt_{X5}, B_2, A_2, A_3).$$

2.7 Обвивка на множество от точки

В някои от случаите на разкрой на планки е предпочитателно да работим с изпъкнала обвивка на дадения полигон, виж фигура 2.53.



Фигура 2.53: Обвивка на полигон.

Нека е даден полигона $A = list(pt_0, pt_1, \ldots, pt_n)$. Под изпъкнала обвивка на множество от точки ще разбираме такъв полигон $C = list(pt_{c0}, pt_{c1}, \ldots, pt_{cn})$, които съдържа всички точки от множеството A и границата му има най-малка дължина. Под понятието съдържа ще означава, че за всички точки от списъка A е изпълнено условието "Точка в полигон"спрямо новия полигон C. Виж точка 2.4. Сега ще представим алгоритъма Graham's scan за намиране на обвивка на краен брой точки в равнината XOY, който има сложността $\mathcal{O}(nlogn)$ [11]. Методът е публикуван през е 1972 год. от Ronald Graham. Алгоритъмът намира всички върхове на изпъкналата обвивка в подреден списък. За целта трябва да се изпълнят следните стъпки:

1. Намиране на точката с най-малка Y ордината. Ако имаме повече от една точка то взимаме тази с най-малка X стойност. В случая $pt_0 \equiv pt_{base}$;

2. Изчисляваме ъглите на всички точки спрямо точка pt_{base} и сортираме списъка от точки спрямо ъглите като първа е точката с най-малък ъгъл, а последна е точката с най-голям ъгъл;

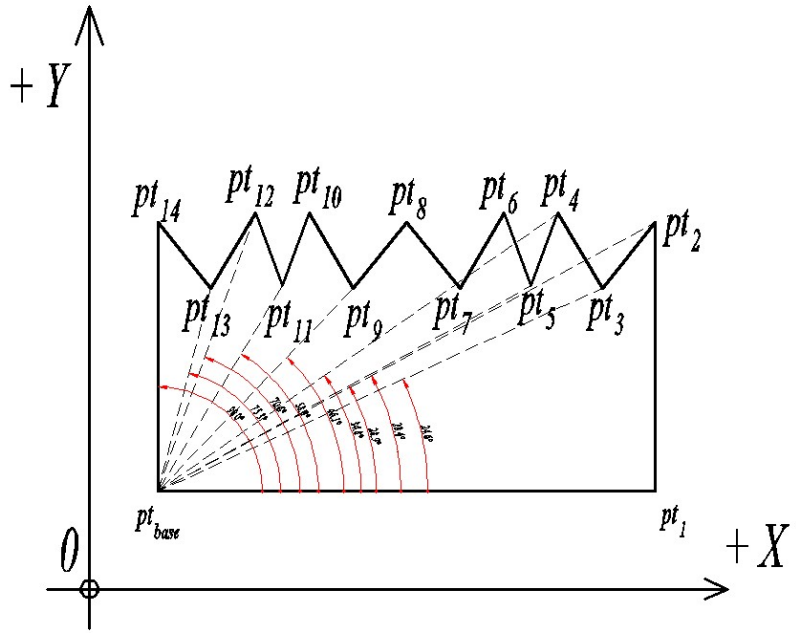
3. Точка pt_{base} винаги е от обвивката (съгласно точка 1). Започваме да проверяваме всеки три точки за правилото $isLeft$. Ако $isLeft = False$, то взимаме средната точка pt_2, за критерия $isLeft$. Виж алгоритъм 3, както и фигури 2.24 и 2.25.

Сега да приложим алгоритъма за фигура 2.53. След филтриране на всички точки по минимум Y намираме, че остават две точки pt_0 и pt_1. От тези две точки избираме тази с най-малка X стойност.

Algorithm 8 GetMinPtConvexHull

/*Функция за намиране на една точка от изпъкналата обвивка на множество точки*/

procedure GetMinPtConvexHull($list(pt_0, pt_1, \ldots, pt_n)$)
(set $pt_0 = pt_{base}$)
(foreach pt_i <- ptL
(cond
if $(y.pt_i \equiv y.pt_{base})$ **then if** $(x.pt_i < x.pt_{base})$ **then** set $pt_i = pt_{base}$;end of cond 1
if $(y.pt_i < y.pt_{base})$ **then** set $pt_i = pt_{base}$;end of cond 2
);cond
);end of foreach



Фигура 2.54: Сортиране по ъгли към обвивка на полигон.

След като приложим алгоритъм 8 за списъка от точки $A = list(pt_0, pt_1, \ldots, pt_n)$, точка pt_{base} ще има възможно най-ниска ордината Y и най-малката абсциса X. На фигура 2.53 това е точка pt_0. Затова съставяме от списъка A премахваме точка pt_{base} и получаваме новия списък $A' = list(pt_1, pt_2, \ldots, pt_n)$. Не е задължително първата точка от списъка A да бъде pt_{base}. Затова критерия за премахване на точка pt_{base} от списъка A трябва да стане по индекс, а не по съвпадащи се координати

на точките. Сега изчисляваме ъглите от pt_{base} до всяка една точка от списъка A'. Виж фигура 2.54. Сортираме списъка A' от най-малкия ъгъл до най-големия. Получаваме следния сортиран списък

$$A'_{sort} = list(pt_1, pt_3, pt_2, pt_5, pt_7, pt_4, pt_6, pt_9, pt_8, pt_{11}, pt_{10}, pt_{12}, pt_{13}, pt_{14}).$$

Сега създаваме нов празен списък B. В този списък ще записваме само точките от изпъкналата обвивка на списъка $A = list(pt_0, pt_1, \ldots, pt_n)$. Първата точка която е от обвивката е pt_{base}. Затова добавяме първата точка в списъка $B = list(pt_{base})$ и започваме да работим успоредно с двата списъка. Критерия за записване на точка в списъкаB е проверката $isLeft$. Проверката $isLeft$ иска три точки. Първата точка трябва да бъде "базата"от която ще се мери Това е последно въведената точка от списъка B. Другите две точки pt_i , pt_{i+1} се вземат от списъка A'. Те са определяне на посоката на ротация. Алгоритъмът за фигура 2.54 работи както следва:

1. $isLeft- > list(pt_{base}, pt_1, pt_3) = True$. Условието е вярно, следователно добавяме точка pt_1 в B или $B = list(pt_1, pt_{base})$;

2. $isLeft- > list(pt_1, pt_3, pt_2) = False$. Условието не е вярно, следователно не добавяме точка pt_3 в B;

3. $isLeft- > list(pt_1, pt_2, pt_5) = True$. Условието е вярно, следователно добавяме точка pt_2 в B или $B = list(pt_2, pt_1, pt_{base})$;

4. $isLeft- > list(pt_2, pt_5, pt_7) = False$. Условието не е вярно, следователно не добавяме точка pt_5 в B;

5. $isLeft- > list(pt_2, pt_7, pt_4) = Flase$. Условието не е вярно, следователно не добавяме точка pt_7 в B;

6. $isLeft- > list(pt_2, pt_4, pt_6) = True$. Условието е вярно, следователно добавяме точка pt_4 в B или $B = list(pt_4, pt_2, pt_1, pt_{base})$;

7. $isLeft- > list(pt_4, pt_6, pt_9) = True$. Условието е вярно, следователно добавяме точка pt_6 в B или $B = list(pt_6, pt_4, pt_2, pt_1, pt_{base})$;

8. $isLeft- > list(pt_6, pt_9, pt_8) = False$. Условието не е вярно, следователно не добавяме точка pt_9 в B;

9. $isLeft- > list(pt_6, pt_8, pt_{11}) = True$. Условието е вярно, следователно добавяме точка pt_8 в B или $B = list(pt_8, pt_6, pt_4, pt_2, pt_1, pt_{base})$;

10. $isLeft- > list(pt_8, pt_{11}, pt_{10}) = False$. Условието не е вярно, следователно не добавяме точка pt_{11} в B;

11. $isLeft- > list(pt_8, pt_{10}, pt_{12}) = True$. Условието е вярно, следователно добавяме точка pt_{10} в B или $B = list(pt_{10}, pt_8, pt_6, pt_4, pt_2, pt_1, pt_{base})$;

12. $isLeft- > list(pt_{10}, pt_{12}, pt_{13}) = True$. Условието е вярно, следователно добавяме точка pt_{12} в B или $B = list(pt_{12}, pt_{10}, pt_8, pt_6, pt_4, pt_2, pt_1, pt_{base})$;

13. $isLeft- > list(pt_{12}, pt_{13}, pt_{14}) = Flase$. Условието не е вярно, следователно не добавяме точка pt_{13} в B;

14. $isLeft- > list(pt_{12}, pt_{14}, pt_0) = True$. Условието е вярно, следователно добавяме точка pt_{14} в B или $B = list(pt_{14}, pt_{12}, pt_{10}, pt_8, pt_6, pt_4, pt_2, pt_1, pt_{base})$;

Така получихме списъка:

$$B = list(pt_{14}, pt_{12}, pt_{10}, pt_8, pt_6, pt_4, pt_2, pt_1, pt_{base}),$$

който е изпъкналата обвивка на множеството от точки:

$$A = list(pt_0, pt_1, pt_2, pt_3, pt_4, pt_5, pt_6, pt_7, pt_8, pt_9, pt_{10}, pt_{11}, pt_{12}, pt_{13}, pt_{14})$$

показани на фигура 2.54.

Съществуват различни критерии за прилагане на обвивка на даден полигон. Някои от тях са :

1. Броя на върховете на полигона е по-голям от даден параметър. Този параметър трябва да бъде съобразен с хардуера където ще се реализира алгоритъмът;
2. Ако полигонът не е изпъкнал;
3. Ако съотношението на площта на дадения полигон към площта на обвивката е по-голямо от даден параметър;

$$ratio_{convex} = \frac{A_{poly}}{A_{convex}} \leq 1.0 \tag{2.32}$$

В алгоритъма можем да заложим минимална стойност, например,

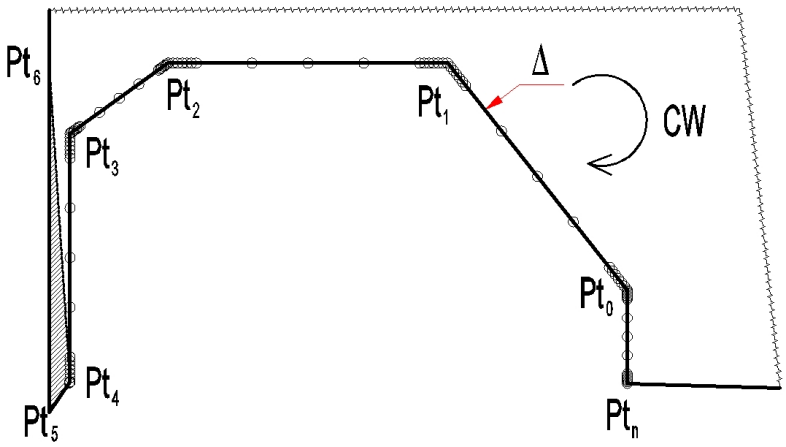
$$0.95 \leq ratio_{convex} \leq 1.$$

Другият параметър може де е борят на върховете на дадения полигон - $Count_{vertex}$. Тогава, ако е $ratio_{convex} \geq 0.95$. и $Count_{vertex} > n$, ще използваме процедурата за намиране на обвивка на входящия полигон. Търсенето на местоположението на полигона може да става с неговата обвивка, а реалното му изчертаване (изрязване) ще бъде с действителния му контур. Ефект от тази процедура ще има за полигони с много върхове. Например, при детайлите зададени с много дъги или с окръжности, които са апроксимирани със полигон.

2.8 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. Виж фигура 2.55. Около върховете на полигона подробните точки са сгъстени. 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.

В щрихованата област $list(pt_4, pt_5, pt_6)$ от фигура 2.55 няма как да се уплътни с други фигури. Преценката за това е на база площ на полигоните. Ако щрихованата област $list(pt_4, pt_5, pt_6)$ е по-малка от най-малката площ от входящите полигони $minArea$, то тя може да бъде премахната. Ако не се приложи редуциране на полигона, то поради сегменти $e_i = list(pt_4, pt_5)$ и $e_j = list(pt_5, pt_6)$ ще се добавят 30



Фигура 2.55: Подробни точки вмъкнати по сегментите на полигон.

подробни точки. Те от своя страна ще предизвикат транслиране на всички входящи полигони заедно със съответните ротации около всяка една подробна точка. Следователно, 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, виж точка 2.1. 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 $list(pt_{i-1}, pt_i, pt_{i+1})$, $i = 1 \ldots n$. 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.

Algorithm 9 ReducePolyVertex

/*Функция за премахване на върхове на полигона*/

procedure REDUCEPOLYVERTEX($pt_0, pt_1, \ldots, pt_n$)
$minArea$ = apply 'min ($getPolyArea\Pi_i$)
$isCW$ = (isClockWise Δ)
$ptList$ = nil

for pt_{i-1}, pt_i, pt_{i+1} **do**
if (isClockWise pt_{i-1}, pt_i, pt_{i+1}) = $isCW$ **then**
return
if (getPolyArea pt_{i-1}, pt_i, pt_{i+1}) >= $minArea$ **then return** $Add.pt_i- > ptList$
else$Add.pt_i- > ptList$

Нека да приложим алгоритъма до pt_6.
$isCW = true$
$minArea$ - положителна константа.

1. $(isClockWise(pt_n, pt_0, pt_1)/ = isCW) \implies Add.pt_0$;
2. $(isClockWise(pt_0, pt_1, pt_2)/ = isCW) \implies Add.pt_1$;

3. $(isClockWise(pt_1, pt_2, pt_3) / = isCW) \implies Add.pt_2$;

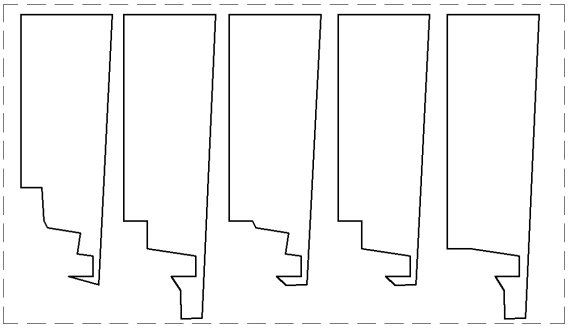
4. $(isClockWise(pt_2, pt_3, pt_4) / = isCW) \implies Add.pt_3$;

5. $(isClockWise(pt_3, pt_4, pt_5) = isCW) \implies$
   $(getPolyAreapt_3, pt_4, pt_5) < minArea \implies \emptyset$;

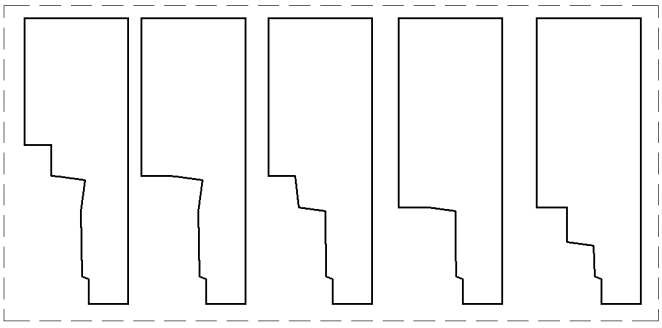
6. $(isClockWise(pt_4, pt_5, pt_6) = isCW) \implies$
   $(getPolyAreapt_4, pt_5, pt_6) < minArea \implies \emptyset$;

Така получихме следния списък от точки $list(pt_0, pt_1, pt_2, pt_3)$. Да допуснем, че и pt_6 е във списъка. Тогава списъкът добива вида $list(pt_0, pt_1, pt_2, pt_3, pt_6)$. Вижда се, че площта между $list(pt_2, pt_1, pt_3, pt_6)$ от новия полигон може също да отговаря на условието $(getPolyAreapt_2, pt_3, pt_6) < minArea$. Затова алгоритъм 9 трябва да се изпълнява докато $(getPolyAreapt_{i-1}, pt_i, pt_{i+1}) >= minArea$.

Тук ще покажем полигони без използуване на алгоритъм 9.



Фигура 2.56: Полигони без редукция.



Фигура 2.57: Полигони с редукция.

Методът е доста бавен ако не използваме редукцията на върховете на изрязания полигон.

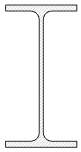
Глава 3

Задача за $1D$ разкрой.

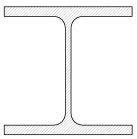
3.1 Формулировка на задачата.

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 правопропорционална 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.

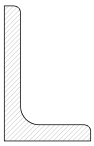
В тази конструкция за прътовите елементи се използват сечения от най-различен вид. Двойно Т" профили, "L" профили, "C" профили. Част от най-използваните сечения са показани на фигури 3.1, 3.2, 3.3, 3.4 и 3.5.



Фигура 3.1: Профил 1

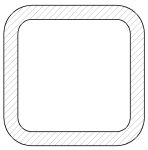


Фигура 3.2: Профил 2



Фигура 3.3: Профил 3

Фигура 3.4: Профил 4

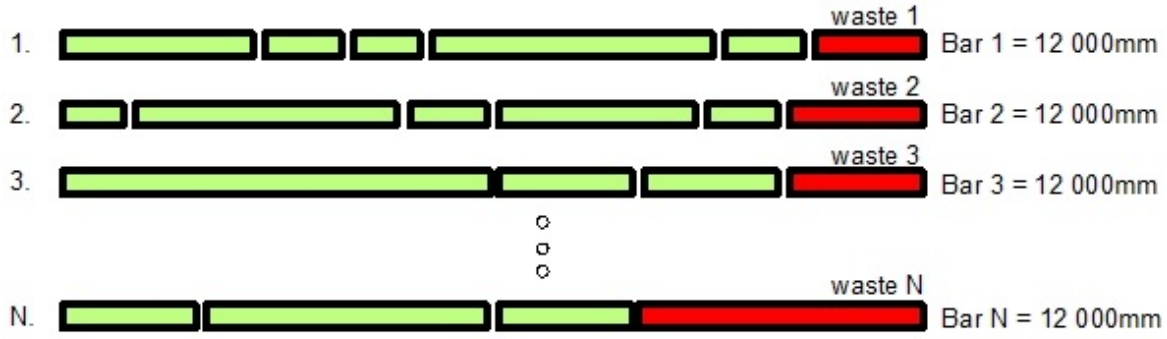


Фигура 3.5: Профил 5

Сеченията показани на фигури 3.1, 3.2 и 3.4 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 Bar_i. Или това е линейното разполагане на част от профилите l_i в дадена дължина Bar_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 Bar_i, виж фигура 3.6.



Фигура 3.6: Дефиниране на линеен разкрой.

Където Bar_i, $i = 1 \ldots n$, виж фигура 3.6 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).Виж схема 4.6.

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 $waste1$ им между всичките намерени решения. Трябва да обърнем внимание, че $waste1$ 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 3.6) ги изключваме от входящия списък. Задачата се повтаря до като входящия списък стане празен. След това записваме в една променлива $sumWaste = (waste_1 + waste_2 + \cdots + wasten)$ 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.

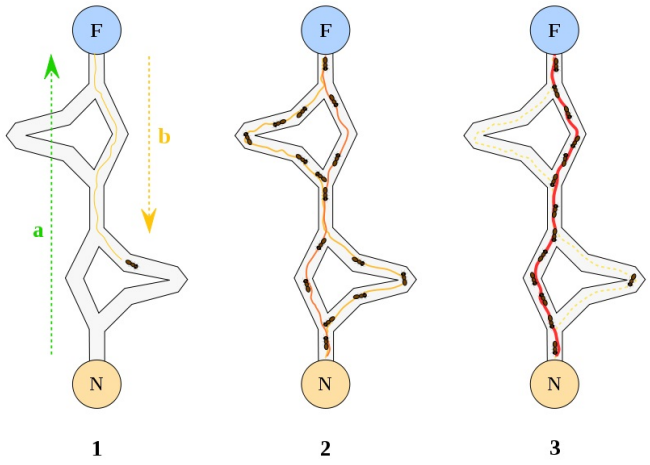
Нека е даден следния входящ списък с профили за разкрояване. Входящият списък се получава директно от CAD системата. Входни данни за $1D$ разкрой е показан в таблица 3.1. :

Таблица 3.1: Списък за разкрой на профили.

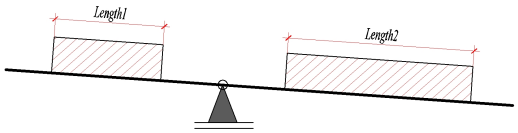
n	Сечение	Броя	Дължина
1	2	3	4
1	Profile 1	36	320
2	Profile 12	54	330
3	Profile 8	4	330
4	Profile 15	18	334
5	Profile 31	54	340
6	Profile 25	54	350
7	Profile 19	360	365

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

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 Bar_i.



Фигура 3.7: Илюстрация на метода мравките.



Фигура 3.8: Оценка.

Метода на мравките (Ant Colony Optimization) е част от методите с популации. Методите с популации са част от Метаевристиката. Виж фигури 4.6 и 4.7.

Най-общо това е вероятностен подход за решаване на най-различни изчислителни задачи. ACO 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 помалко мравки се използват за намирането на глобалния оптимум токова 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, които 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.

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

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.

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

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

Algorithm 10 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 Bar_i. Дължината на профила е неговата тежест, защото сечението е едно и също. То е константа. Виж фигура 3.8. 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 Bar_i е по-голям при по-дългите профили. В този случай метода на мравките проявява *Greedy* характер.

Вторият метод е динамично оптимиране. При този подход не се прави комбинации между профилите, а записва сбора на дължите 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.

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

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

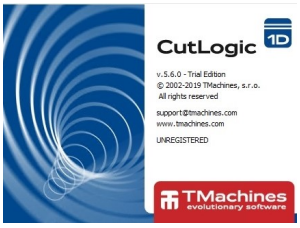
$$\frac{10!}{3!(10-3)!} = \frac{3628800}{6.5040} = 120 \tag{3.4}$$

Дължините на профилите с еднакво сечение не варират в големи граници. При направените сравнения между трите метода, за целите на стоманените конструкции хибридния 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.

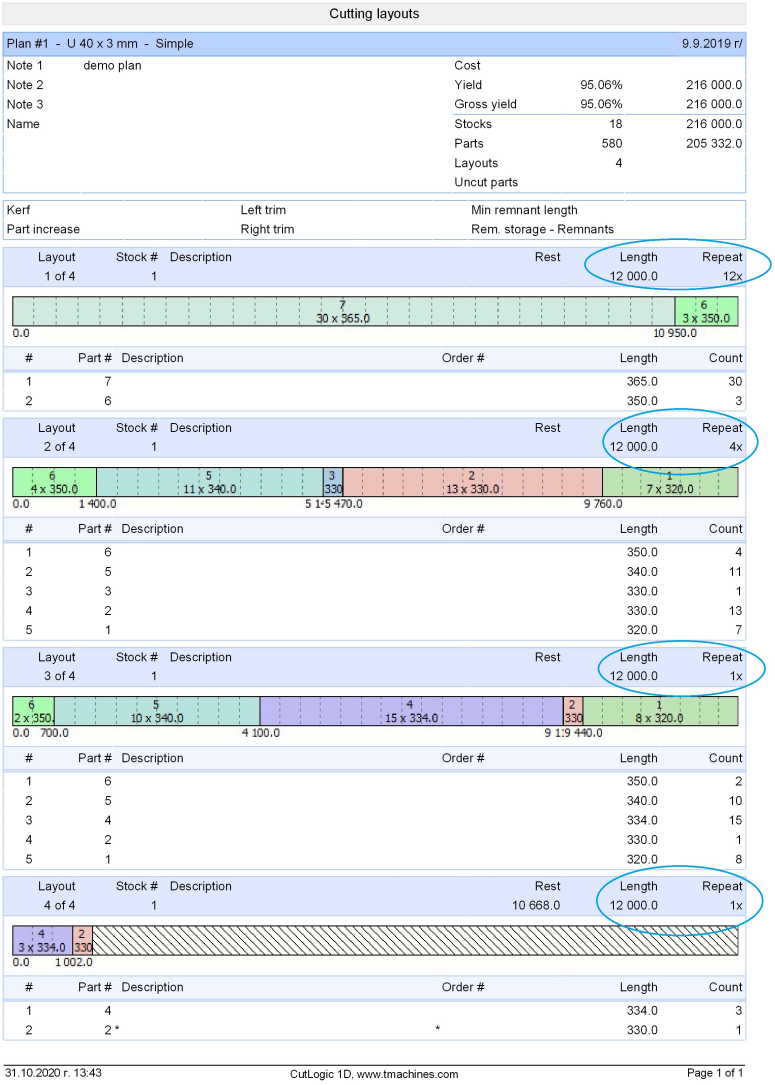
3.3 Резултати при $1D$ разкрой. Примери.

Ще използваме следния комерсиалния продукт:



Фигура 3.9: 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

7 30 x 365.0 | 6 3 x 350.0
0.0 10 950.0

#	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

6 4 x 350.0 | 5 11 x 340.0 | 3 330 | 2 13 x 330.0 | 1 7 x 320.0
0.0 1 400.0 5 1•5 470.0 9 760.0

#	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

6 2 x 350 | 5 10 x 340.0 | 4 15 x 334.0 | 2 330 | 1 8 x 320.0
0.0 700.0 4 100.0 9 1:9 440.0

#	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

4 3 x 334.0 | 2 330
0.0 1 002.0

#	Part #	Description	Order #	Length	Count
1	4			334.0	3
2	2 *		*	330.0	1

31.10.2020 г. 13:43 CutLogic 1D, www.tmachines.com Page 1 of 1

Фигура 3.10: Решение на $1D$ разкрой с комерсиален продукт.

В сините елипси на фигура 3.10 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
$Greedy + n^3$	17x12000 + 1080 мм	< 1
Мравки ACO	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.

Тук ще се дадем решение на една структурна оптимизация на задача която идва от строителната индустрия. 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 4.1.

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 използуване 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 CAD 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 CAD системата ще приложим функцията за изчистване на излишните точки. След премахване на излишните точки получаваме входящ допустим полигон описан чрез списък от точки

$$\Pi = list(pt_0, pt_1, ...pt_n), \tag{4.1}$$

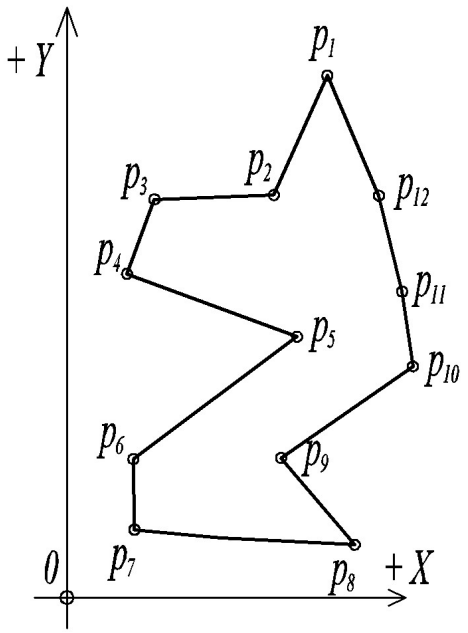
където pt_i са върхове на полигона. Този списък има следните свойства: 1. Списъкът е подреден; 2. Списъкът е цикличен $pt_n = pt_0$; 3. Няма самопресичане.

В процеса на работа на алгоритъма ще ни бъде необходимо понятието сегмент, което е отсечка между два последователни върха. По този начин генерираме сегментите $e_1 = list(pt_0, pt_1), \ldots, e_n = list(pt_{n-1}, pt_n)$ и получаваме друга характеризация на полигона $\Pi = list(e_1, e_2, \ldots, e_n)$. Планки с криволинейни граници се апроксимират с полигони с достатъчен брой върхове. Примери за входящи допустими полигони са показани на фигура 4.8.

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

$$\Delta = list(p_0, p_1, \ldots, p_m) \tag{4.2}$$

Контурът за запълване не трябва да бъде самопресичащ се.

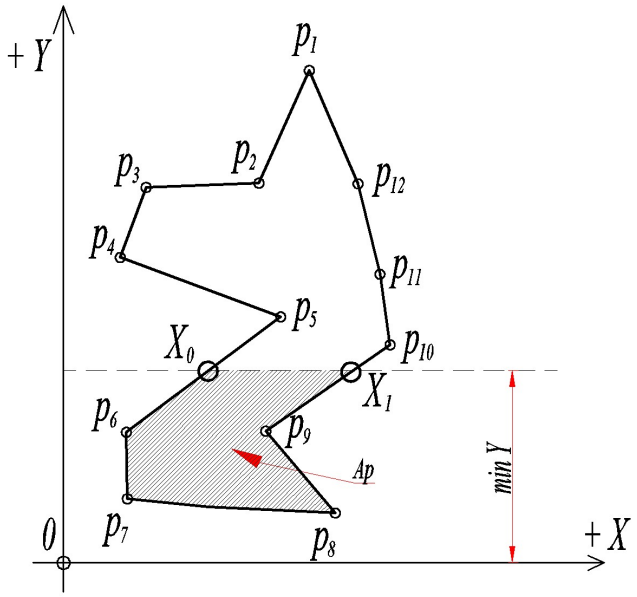


Фигура 4.1: Полигон за запълване.

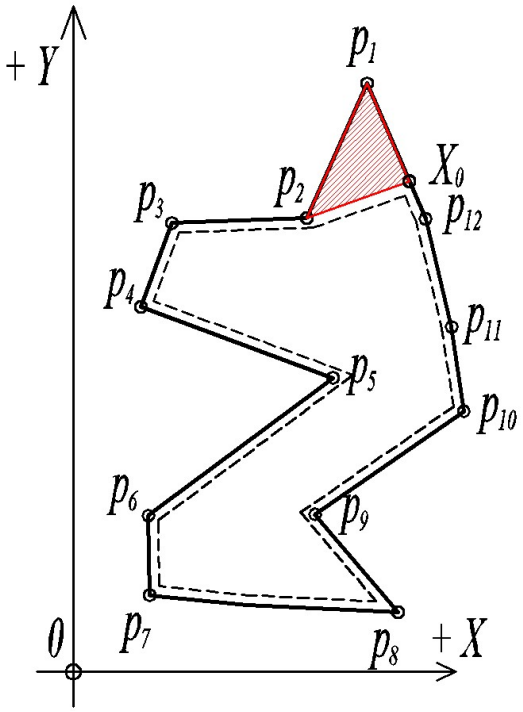
Има два критерия за оптимизация на задачата:

1. Оптимизация на минимална височина на запълване - $minY$. Виж фигура 4.2;

2. Оптимизация на броя на върховете на остатъчния полигон след изрязването на входящия полигон от полигона на запълване - $minVertex$. Виж фигура 4.3;



Фигура 4.2: Полигон за запълване-min Y.



Фигура 4.3: Полигон за запълване-min Vertex

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.

Да дефинираме референтно лице на фигурата между точките :

$$A_P = list(p_7, p_8, p_9, X_1, X_0, p_6) \tag{4.3}$$

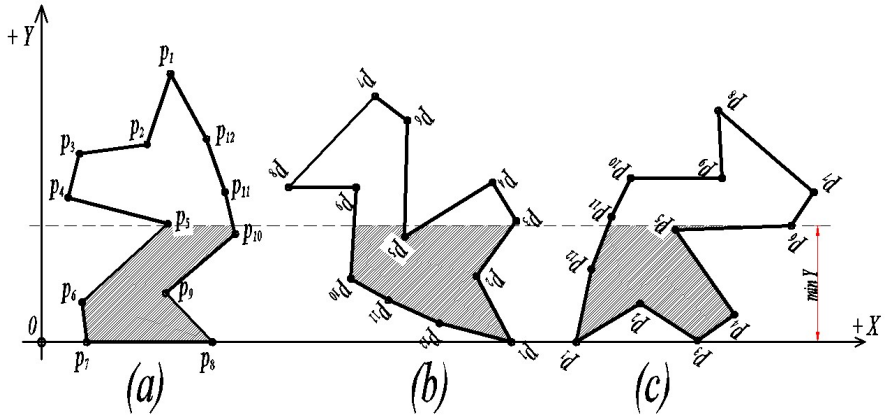
За намиране на лице на полигон виж формула 2.12. Формула 2.12 дава удвоеното лице на полигона.

Тогава определяме коефициента на запълване $ratio$ като:

$$ratio = \frac{\sum_{i=1}^{n} F_i}{A_P} \tag{4.4}$$

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 зависимостта 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. На фигура 4.4 са показани първите три случая на ротация на полигона P. Броя на ротациите е равен на броя на сегментите.

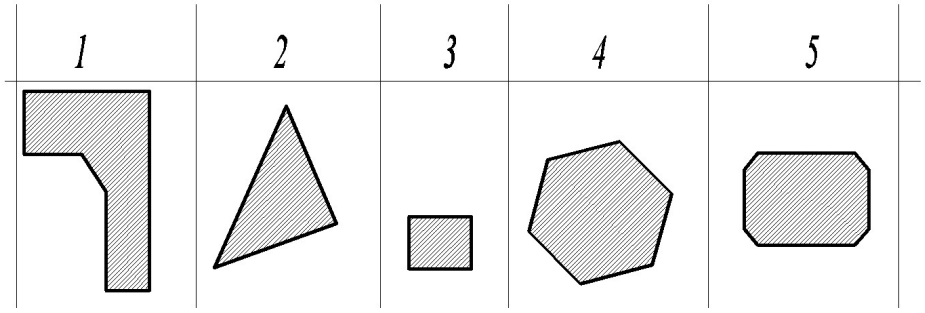


Фигура 4.4: Ротация на полигона за запълване

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.



Фигура 4.5: Входящи множества (полигони/планки).

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. Виж фигура 4.3. 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. полиноминално) 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. Метаевристката 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.

Класификация на метаевристиките съгласно [27]
.

Фигура 4.6: Класификация на метаевристиките.

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 4.7;
2. Вероятностно предвиждане на избора на даден елемент;
3. Йерархично оценяване на решенията.

Имаме даден списък от полигони

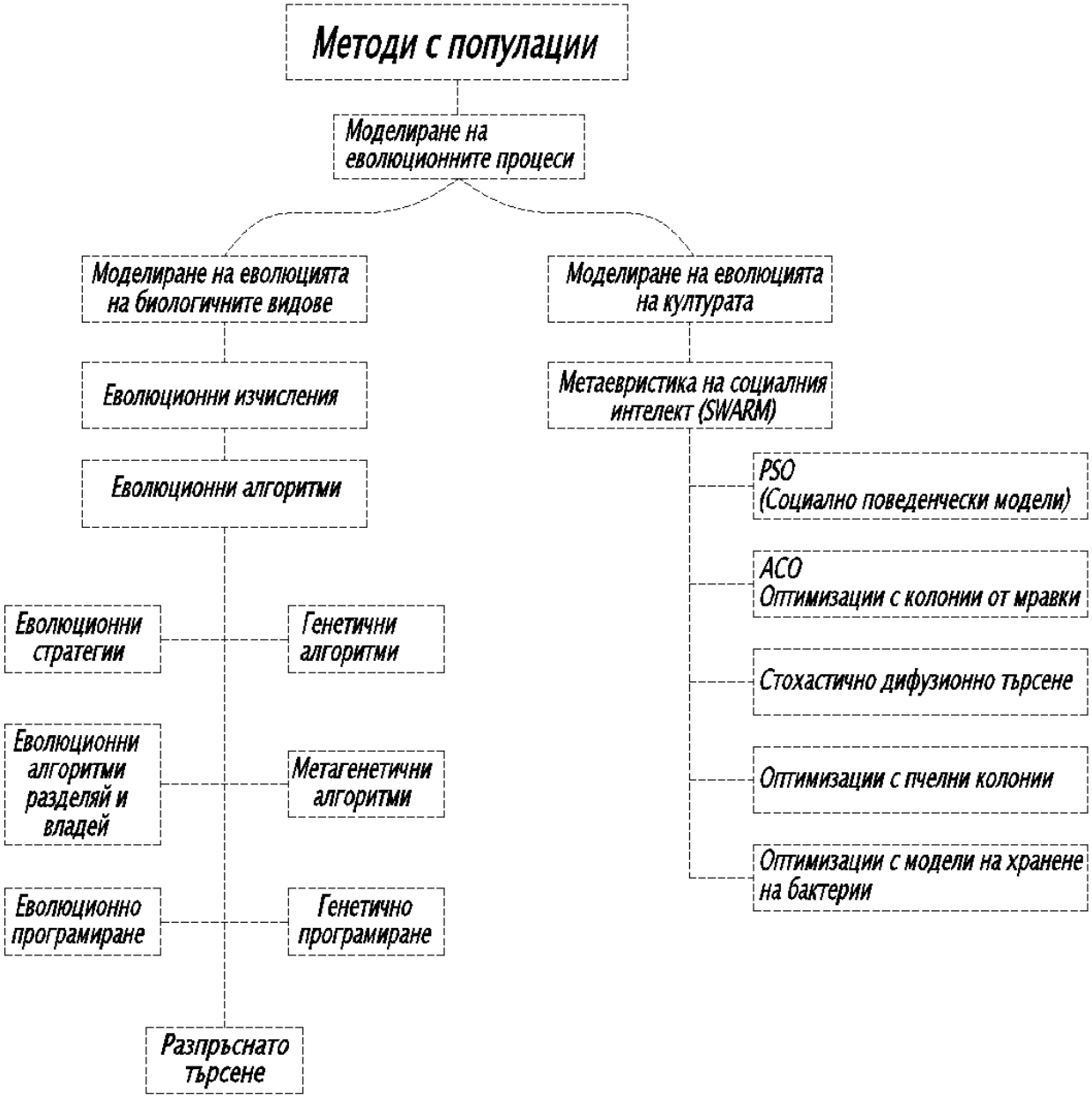
$$\Pi_L = (list(list_1(pt_0, pt_1, ...pt_n), (list_2(pt_0, pt_1, ...pt_n), \dots (list_n(pt_0, pt_1, ...pt_n)), \quad (4.5)$$

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

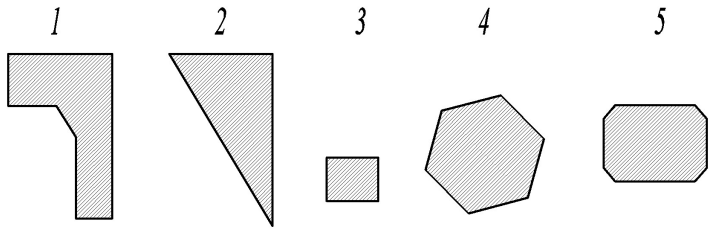
Полигона Δ, които трябва да се запълни се описва със списък от точки:

$$\Delta = list(p_0, p_1, \dots, p_m) \quad (4.6)$$

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.



Фигура 4.7: Класификация на метаевристики с популации.



Фигура 4.8: Примери на входящи множества Π_L (полигони/планки)

.

Преди да се избере входящо множество е необходимо да се направи оценка за

съвпадението на ъглите и страните от текущия входящ полигон

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

към то ъглите и страните на полигона за запълване $\Delta = (list(pt_0, pt_1, ...pt_n)$. За целта се съставят два нови производни списъка на Π_i и Δ. Те съответно съдържат последователно подредени списъци $list(previosLength, Angle, NextLength)$. 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:

$\Pi_i = list((list(cons"previosLength"(distance_{pt_n,pt_0}))$
$(cons"angle"pt_n, pt_0, pt_1)$
$(cons"NextLength"(distance_{pt_0,pt_1})))$
...
$(list(cons"previosLength"(distance_p t_n pt_{n-1}))$
$(cons"angle"pt_0, pt_n, pt_{n-1})$
$(cons"NextLength"(distance_{pt_n,pt_0}))))$

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, които има по-голяма дължина (по-голям брой елементи). На фигура 4.9 това е списъка $listA$. Списъкът $listB$ е този с по-малката дължина. 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 $listA$. Виж 4.9 случай (1). 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. Виж 4.9 случай (2). Това се повтаря докато преминем през всички индекси на $listA$ или цикълът се повтаря толкова пъти колкото е дължината на списъка $listA$. Виж 4.9. За всяка проверка - случай от (1) до (10) на фиг. 4.9 записваме броя на последователните съвпадения. Оценката на списъка $listB$ се дава със следната формула:

$$k = \sum Ratio_{angle_i} + \sum Ratio_{Length_i}, \tag{4.7}$$

където

$$Ratio_{angle_i} = (if|(angle_i - angle_{Bi})| < fuzz \rightarrow return1., else0.).$$

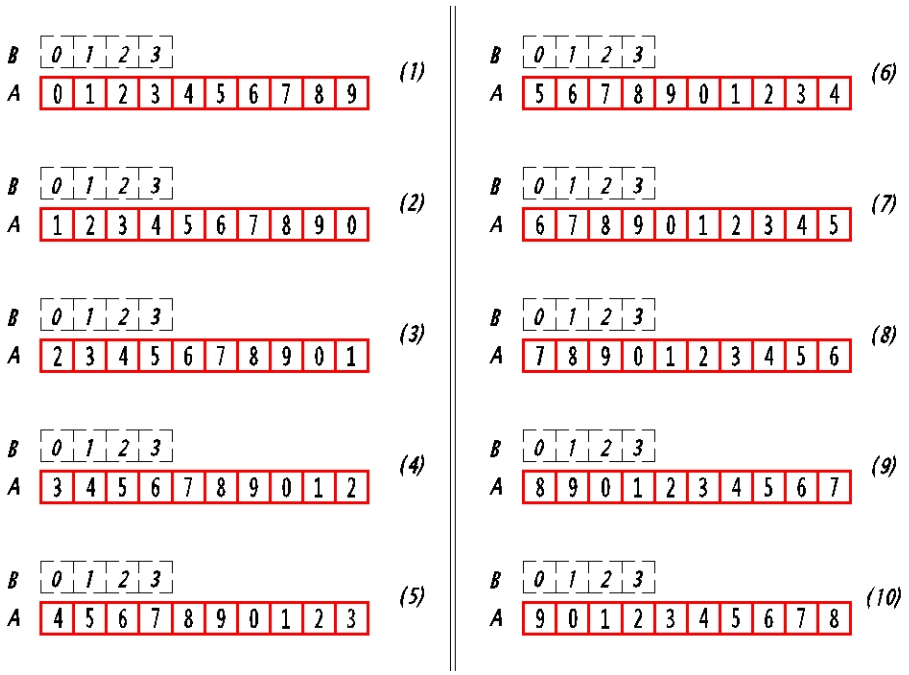
При ъглите търсим пълно съвпадение. Като Ai е ъгъл от списъка $listA$, а Bi е съответния ъгъл от списъка $listB$.

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

$$Ratio_{Length_i} = (if(length_{Ai} > length_{Bi}) \rightarrow return(\frac{length_{Bi}}{length_{Ai}}), else(\frac{length_{Ai}}{length_{Bi}}) \quad (4.8)$$

Условие 4.8 дава коефициенти близки или равни на 1., без значение коя дължина е по-голяма $lengthA_i$ или $lengthB_i$. Това е така, защото търсим полигони, на които страните им почти съвпадат.

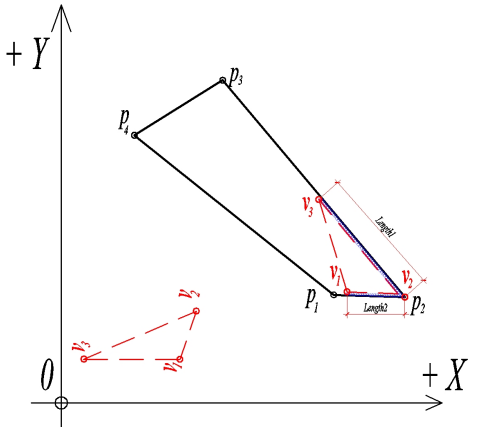
Както се вижда от формула (4.7) полигоните с по-голям брой върхове ще имат по-голяма вероятност за по-високи оценки на съвпадение. Това е добре, защото след изваждането на двата полигона $A \backslash B$ ще получим полигон с по-малко върхове. Коефициентът от формула (4.7) може да се използва като оценка за подобие или еднаквост на фигури.



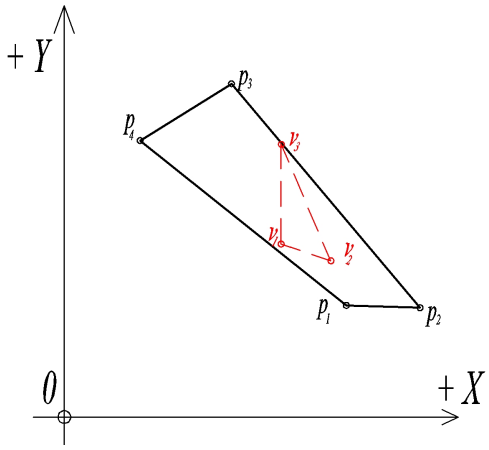
Фигура 4.9: Сравняване на два списъка.

Съставяме списъка от оценени полигони $\Pi_{L,evaluate}$. Сортираме списъка по най-голям брой съвпадения. Ако имаме повече от един полигон с максимално съвпадение избираме произволен полигон от тези с максимално съвпадение. Този подход за оценка на входящите полигони има едно предимство и един недостатък. Предимството е, че входящият полигон се оценява в координатите, в които е дефиниран. Не е необходимо той да се транслира до всеки връх на полигона за запълване. Там да се ротира и след това да се търси съвпадение на ъгли и страни. Недостатъкът е, че не винаги дадена страна от входящия полигон ще съвпада със дадена страна от полигона за запълване. Оттам ще се получи неточно оценяване на дадения входящ полигон. При по-сложни полигони този недостатък може да се окаже съществен. Виж фигура 4.10. Полигона $\Pi_i = list(v_1, v_2, v_3)$ при представения метод ще получи оценка 1, защото имаме едно съвпадение на двата списъка от ъгли и

дължини. Но ако се приложи методът за оценка на максимално опиране тогава полигона $\Pi_i = list(v_1, v_2, v_3)$ ще получи по-висока оценка тъй като има известна обща дължина между страните на двата полигона. В резултат на което ще има по-голяма вероятност да се вземе като добро решение. Разбира се при един и същи полигон Δ само, че с друга ротация и транслация на полигона $\Pi_i = list(v_1, v_2, v_3)$ методът за максимално опиране ще даде дължина на опиране нула, виж фигура 4.11. Затова в разработения софтуер метода за максимално опиране се прилага след като се намерят валидните решения (изчислени са точните координати на входящите полигони). Основен недостатък на този метод е скоростта за оценка на даден полигон. За да използваме метода за максимално опиране е необходимо да се разполага със значителни изчислителни ресурси. При паралелна обработка на данните и наличие на хардуер с графични процесори методът на максималното опиране е за предпочитане а реализиране, спрямо сравняването на полигоните по ъгли и дължини.



Фигура 4.10: Дължина на съвпадение (a).



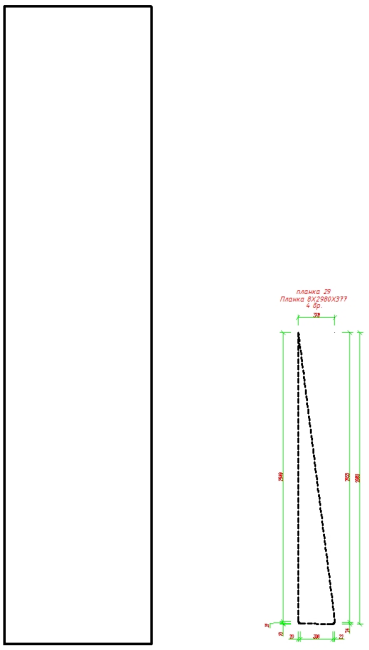
Фигура 4.11: Дължина на съвпадение (b).

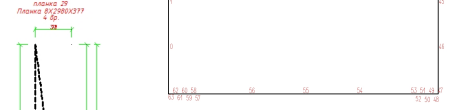
4.3 Стратегия за избор на едно решение от валидни разположения.

Нека разгледаме два полигона. Полигона за запълване $\Delta = list(pt_1, pt_2, pt_3, pt_4)$ и входящ полигон $\Pi_i = list(pt_1, pt_2, pt_3, pt_4, pt_5, pt_6)$. Виж фигура 4.12. Десният Π_i полигон е входящия. Левият полигон Δ е полигона за запълване. Полигона Π_i не е триъгълник! На фигура 4.13 е показано 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. На фигура 4.14 са показани част от възможните решения. Разрешена е ротация на полигона. Огледален образ не е приложен. В случая имаме 112 броя валидни решения. Виж фигура 4.14.

От тези 112 валидни решения оценяваме тези, които имат най-голяма допирна дължина с полигона Δ и разстоянието $LengthA = distance(pt_0, pt_M)$. На фигура 4.15 са показани две равностойни решения като допирна площ, но качествено различни като местоположение. Затова между дължината на опиране и дължината $LengthA$ има зависимост. Тази зависимост е приета от автора. Раз-

бира 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 $\Delta \setminus \Pi_i$. Виж фигура 4.16.

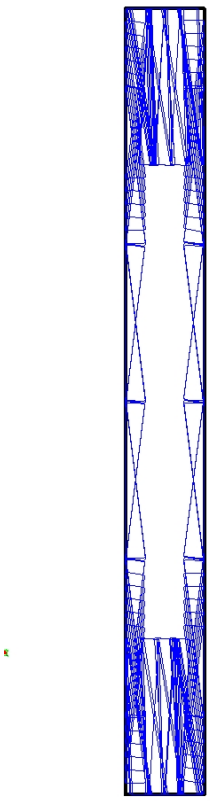




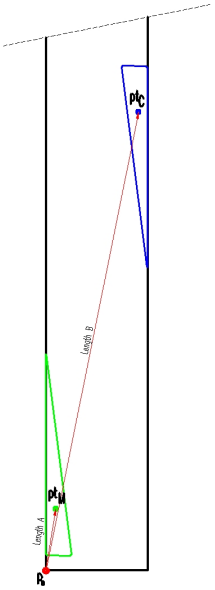
Фигура 4.13: Добавяне на точки в Δ.

Фигура 4.12: Полигони Δ и Π_i.

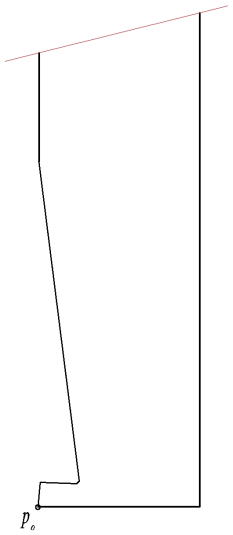
.



Фигура 4.14: Валидни решения.

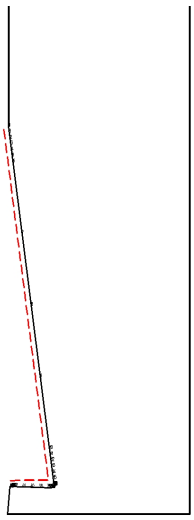


Фигура 4.15: Полигони Δ и Π_i.

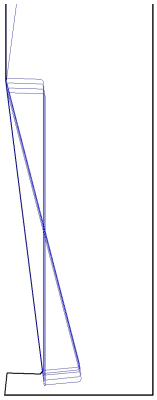


Фигура 4.16: $\Delta \setminus \Pi_i$

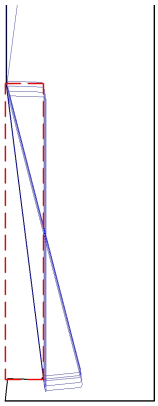
След като сме избрали първо решение пристъпваме към избор на второ. Избора на входящ полигон става по процедурата описна в точка 4.2. Тъй като целта ни е максимално уплътняване на решенията то избора на второто решение трябва да се търси по контура на вече намерените решения. В случая имаме едно намерено решение. Виж червената пунктирна линия на фигура 4.17. Валидните решения са 11 броя. Дадени са на фигура 4.18. 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: $eval = LengthOfTouch + (2.0 * LengthA)$. Виж фигура 4.19. Избраното решение е показано на фигура 4.20. 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.



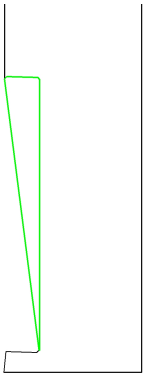
Фигура 4.17: Област на търсене.



Фигура 4.18: Валидни решения.



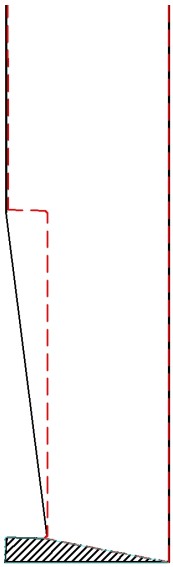
Фигура 4.19: *box* за търсене.



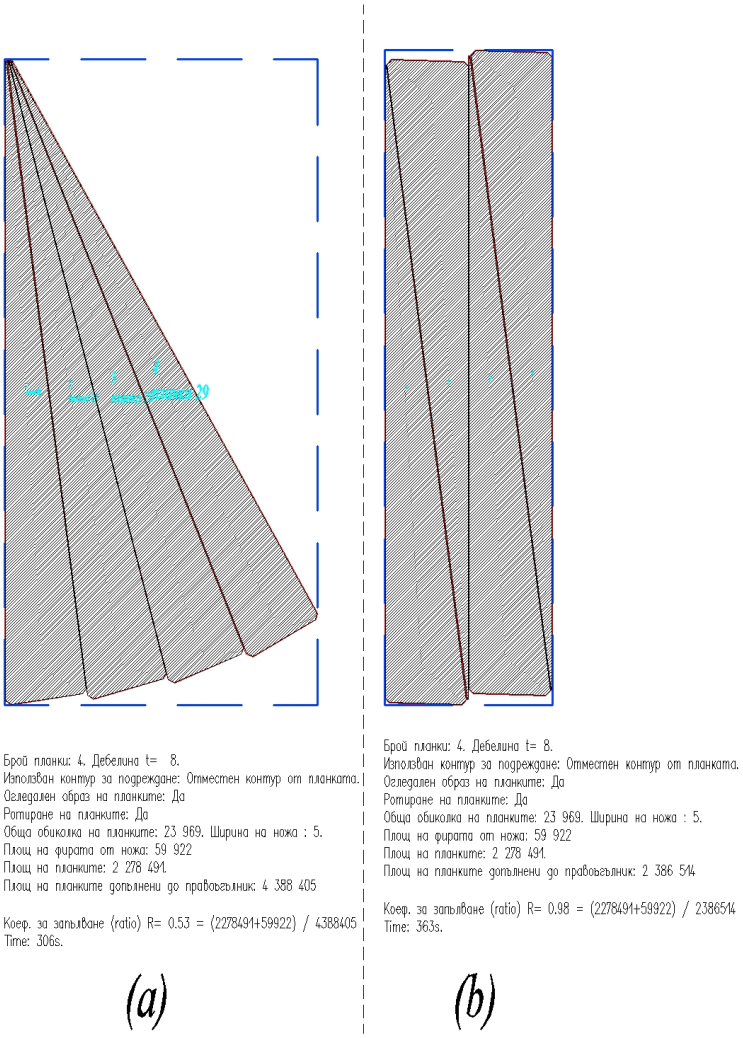
Фигура 4.20: Избрано решение.

След изрязването на двата полигона се получава новия контур. Виж фигура 4.21 червения пунктир. Този контур ще послужи като контур на запълване за следващите полигони. Както се забелязва нови контур не следва напълно стария. Виж защрихованата област на фигура 4.21. Новия контур целенасочено е редуциран. За повече информация за начина на редуциране на контура виж точка 2.8.

$$ratio_{Global} = \frac{A_{box}}{A_{\Pi_L}} \leq 1.0 \tag{4.9}$$



Фигура 4.21: Редуциран полигон за запълване.



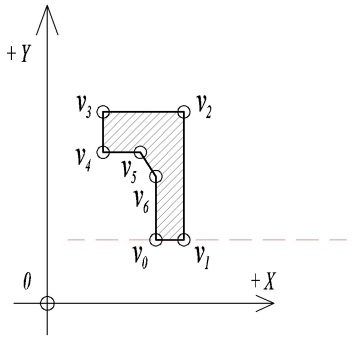
Фигура 4.22: *box* на подредените планки.

На фигура 4.22 със син пунктир е изчертан *box* цлед рфазполагането на четири еднакви планки. Фигура 4.22 (a) изобразява подреждане на планките без йерархично избиране от валидните решения. Вижда се, че йерархичното избиране дава значително по-добър резултат от колкото произволно търсене от валидните решения. Даже и да е приложен критерия за $minY$ подредбата на планките ще бъде както на фигура 4.22 (a).

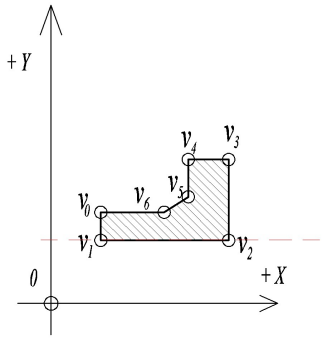
Стратегията се повтаря докато се изчерпят (изберат) всички входящи планки. След това на всички подредени планки се изчислява *box*. След това се изчислява $ratio_{Global}$. Броя на итерациите се повтаря докато се достигне определено $ratio_{Global}$, но не повече от дадено време. Тези параметри зависят от потребителя. Зависи дали материалът, от които ще се реже е скъп и до колко потребителят разполага с хардуерен ресурс за изчисление на задачата.

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

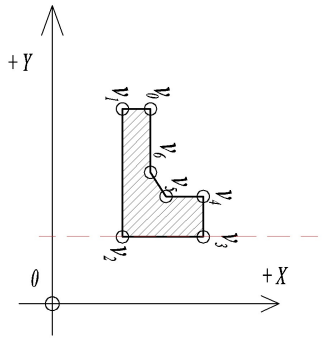
Намирането на едно възможно разполагане на планките (представяни като полигони) става чрез поставяне на входящ полигон в полигона за запълване и прилагане на функцията изваждане на два полигона. Функцията е описана по-долу. В зависимост от изискването на конкретния случай може да построим производни полигони от входящия полигон често с разрешение за прилагане на ротация и огледална симетрия. Виж фигури от 4.23 до 4.29.



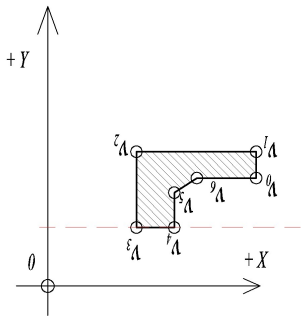
Фигура 4.23: Ротация 1.



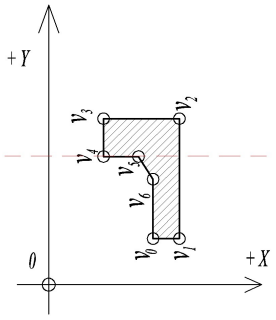
Фигура 4.24: Ротация 2.



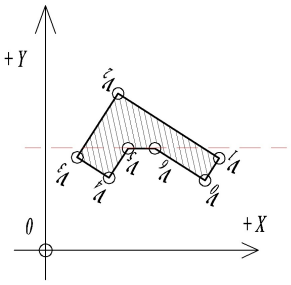
Фигура 4.25: Ротация 3.



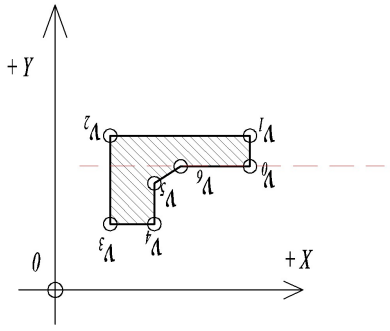
Фигура 4.26: Ротация 4



Фигура 4.27: Ротация 5



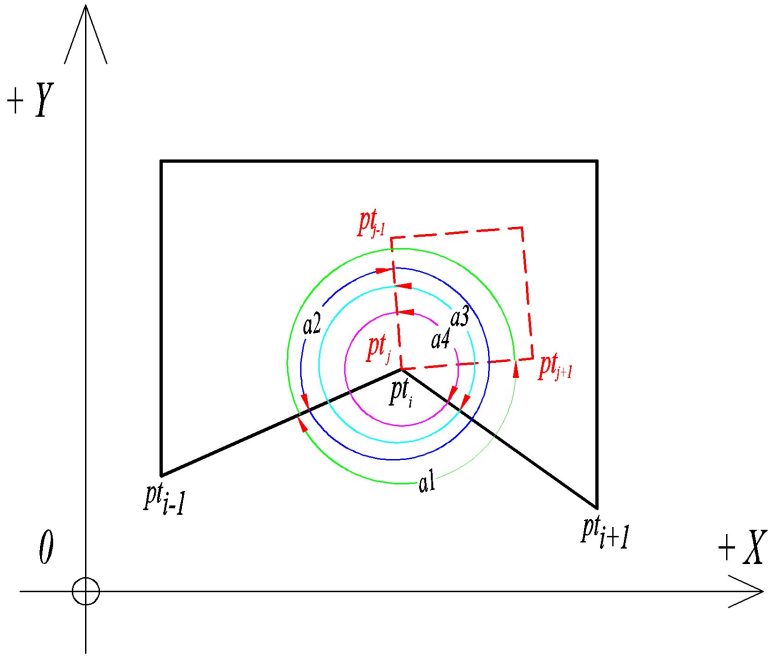
Фигура 4.28: Ротация 6



Фигура 4.29: Ротация 7

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 входяшия полигон с абсцисната ос X. След това могат да се добавят точните ъгли $(0; 0.5\pi; \pi; 1.5\pi)$. Другите допълнителни ъгли, които се добавят към списъка с ъглите са дадени на фигура 4.30.



Фигура 4.30: Подравняване на ъгли

Транслираме текущия връх pt_j от дадения полигон до текущия връх pt_i от полигона запълване. Взимаме предишния и следващия връх от съответния полигон. В случая имаме два списъка: $list = (pt_{i-1}, pt_i, pt_{i+1})$ и $list = (pt_{j-1}, pt_j, pt_{j+1})$. Изчисляваме ъглите между следните върхове:

1. $\alpha 1 = (3PointAlglept_{i-1}, pt_i, pt_{j+1})$;
2. $\alpha 2 = (3PointAlglept_{i-1}, pt_i, pt_{j-1})$;
3. $\alpha 3 = (3PointAlglept_{i+1}, pt_i, pt_{j-1})$;
4. $\alpha 4 = (3PointAlglept_{i+1}, pt_i, pt_{j+1})$;

Както и техните допълнения до 2.0π:

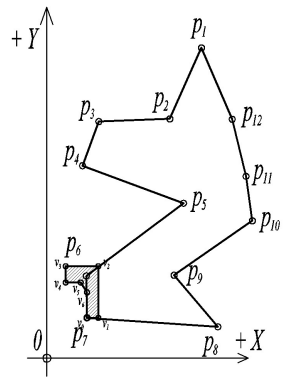
1. $\alpha 5 = (2.0\pi - \alpha 1)$;
2. $\alpha 6 = (2.0\pi - \alpha 2)$;
3. $\alpha 7 = (2.0\pi - \alpha 3)$;
4. $\alpha 8 = (2.0\pi - \alpha 4)$;

Както и противоположните им посоки:

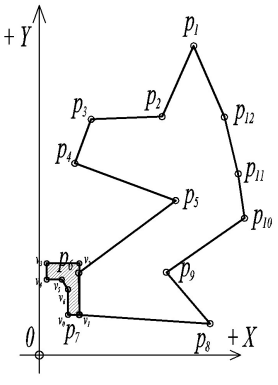
1. $\alpha 9 = -\alpha 1$;
2. $\alpha 10 = -\alpha 2$;
3. $\alpha 11 = -\alpha 3$;
4. $\alpha 12 = -\alpha 4$;
5. $\alpha 13 = -\alpha 5$;
6. $\alpha 14 = -\alpha 6$;
7. $\alpha 15 = -\alpha 7$;
8. $\alpha 16 = -\alpha 8$;

Така към списъка от ъгли добавяме и тези 16 броя. Тук може задачата да се разшири като се добавят ъгли близки до тези 16 ъгъла. Примерно ъгли с разлика от дадения ъгъл $\alpha i = \alpha i \pm 0.01$ rad. Това се прави с цел да се избегне проблема с точността на изчисление на координатите на точките. Преди да започнем ротирането на дадения полигон е необходимо координатите на двата полигона - даденият и полигона за запълване, да се превърнат в стрингове след това да се "изрежат"до четвъртия знак след запетаята и след това да с е превърнат във формат $float$.

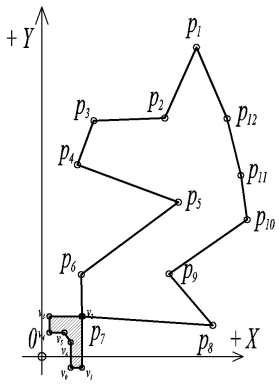
Ако е необходимо и огледано прилагане на входящия полигон, то броят на допълнителните полигони ще се увеличи. Някои състояния на ротация ще бъдат напълно идентични като геометрия, но сравняването им с цел да се филтрират еднаквите състояния е скъпа операция. След като е съставен списъкът от възможните ъгли на полигона започваме да нанасяме всеки един полигон в полигона за запълване. Прилагането трябва да се извърши като всеки един връх от входящия полигон се приравни към текущия връх от полигона за запълване. Виж фигури от 4.31 до 4.37



Фигура 4.31: $v_0 \equiv p_7$



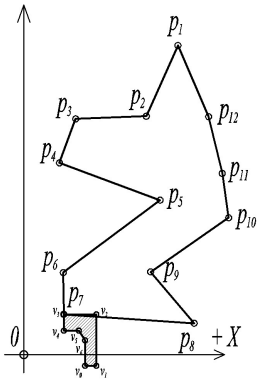
Фигура 4.32: $v_1 \equiv p_7$



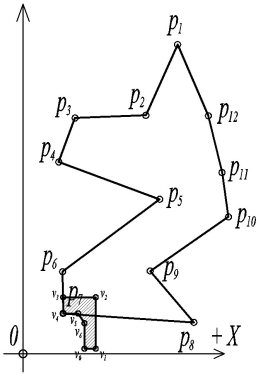
Фигура 4.33: $v_2 \equiv p_7$

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

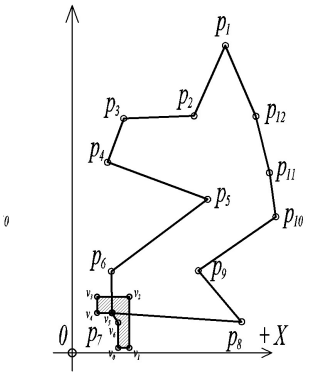
1. Проверка за всеки един възел дали е в полигона за запълване. Тази проверка е добре да стане с цикъл while, ако текущата тествана точка е извън контура решението отпада, без да продължава нататък;
2. Проверка дали има тривиално пресичане на двата полигона. Ако има пресичане, то решението отпада, иначе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



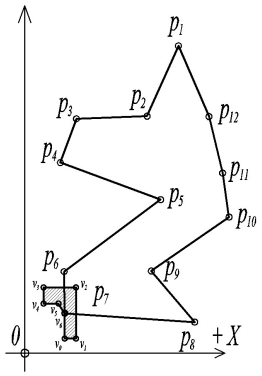
Фигура 4.34: $v_3 \equiv p_7$



Фигура 4.35: $v_4 \equiv p_7$



Фигура 4.36: $v_5 \equiv p_7$

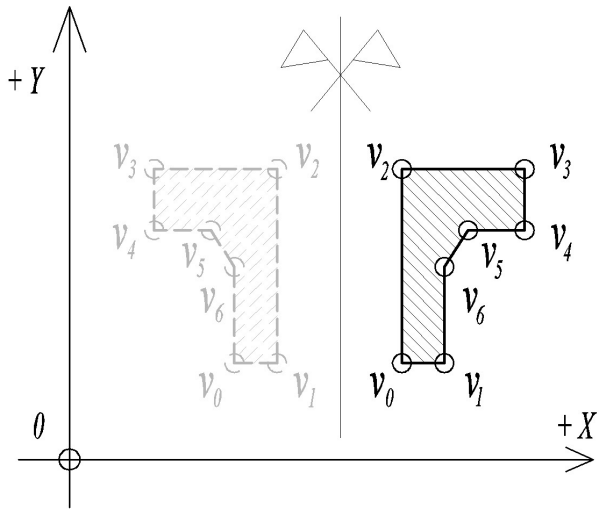


Фигура 4.37: $v_6 \equiv p_7$

Според автора при различните тестове за множество полигони първата проверка е по-бърза. Ако са разрешени ротации и огледално симетрия и входящия полигон има n на брой върхове на полигона за запълване има n на брой върхове, то сложността в общия случай за един полигон ще бъде $\mathcal{O}(2n^2)$. При някои критерии минималната сложност ще бъде $\mathcal{O}(n \log n)$.

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 брои 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 решенията от фигура 4.31 до фигура 4.37 могат да бъдат реализирани като паралелни задачи. Паралелните изчисления могат да се направят и за останалите възможни състояния на входящия полигон. Проверките свързани с полигоните показани на фигури 4.31 - 4.37 могат да се направят независимо една от друга. Или решението на всяка една задача от фигури 4.31 до 4.37 не зависят една от друга и могат да се изчислят паралелно.

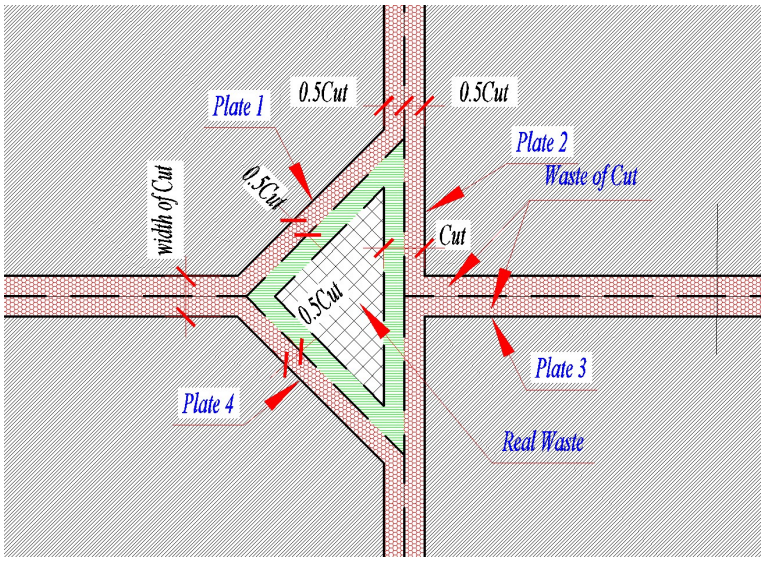


Фигура 4.38: Огледален образ на полигон

На фигура 4.38 десния полигон е огледален (генериран), а левия входящ. Алгоритъмът се повтаря за генерирания огледален образ на входящия полигон. За огледалния полигон са валидни всички генерирани ъгли на ротация. Виж фигури от 4.23 до 4.29 и фигура 4.30.

4.5 Премахване на реалната фира при 2D разкроя.

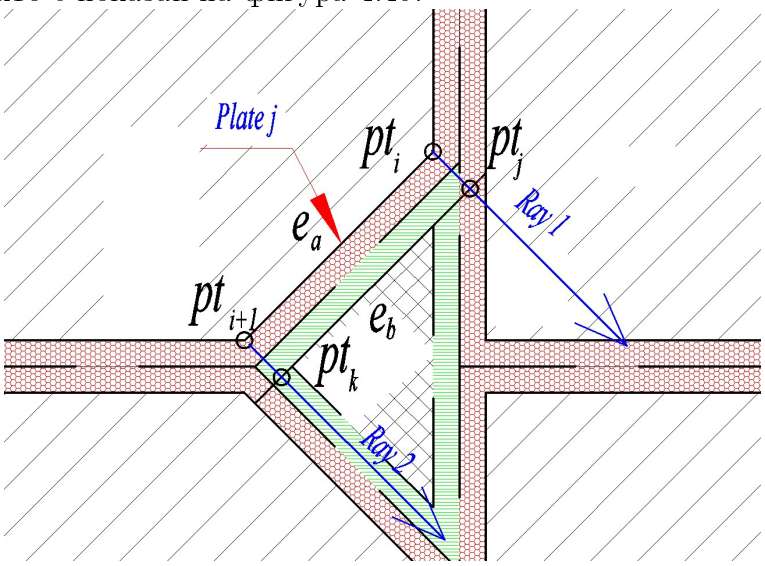
За изрязването на фигурите се използва режещ инструмент с определена харакеристика на рязането. По тази прочина трябва да се осигури разстояние между полигоните. Въвеждаме параметъра $cutW$ за ширина на фугата (ножа) между полигоните. Виж фигура 4.39.



Фигура 4.39: Ширина на фугата 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 $0.5cutW$. Прави се една "обвивка" на входящия полигон. Тази обвивка е представена с пунктирна линия на фигура 4.39. Този параметър е един и същ за всеки полигон и за всяко едно разположение на всеки един полигон. Затова формула 4.4 ще даде относително точен критерий за избор на решение. Относителен е затова защото ние работим

с $0,5cutW$, а там където има сегмент, които не граничи с друг сегмент трябва да се използва ширина на фугата $1.0cutW$. Виж хоризонталната щриховка на фиг. 4.39. Този подход ще подреди полигоните без колизии (презастъпване). При изчисляване на реалния отпадък методът ще даде по-големи остатъци, а те реално ще бъдат по-малки. Този компромис е допустим, защото ширината на ножа (фугата) е много малка в сравнение с очакваните входящи полигони. Загубата на точност ще бъде малка. Печалбата е бързата направа на обвивката около полигоните и бързите проверки за застъпване на полигоните. В случай на малко габаритни елементи и широк нож на рязане в алгоритъма може да се въведе проверка дали даден сегмент не гранични с друг сегмент от вече подредените полигони. Проблемът може да се реши както е показан на фигура 4.40.



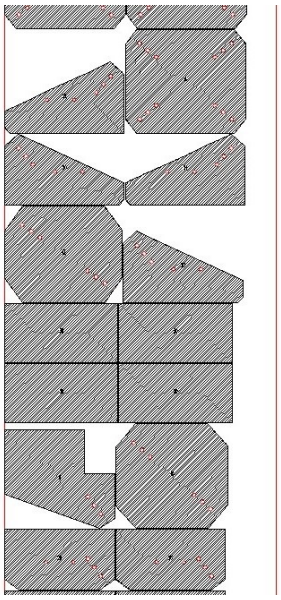
Фигура 4.40: Ширина на фуга при малки полигони

Ако искаме да преместим сегмент e_a от фигура 4.40 на разстояние $cutW$ то трябва намерим перпендикулярна точка на сегмента e_a в случая pt_j, същото да се повтори и за точка pt_{i+1}. След като сме намерили точките pt_k и pt_j проверяваме бъдещия сегмент e_b пресичали се с вече подредените полигони, ако не то можем локално да изместим обвивката на входящия полигон. Това изместване важи само за тази комбинация от подреждане на полигоните. Това изместване на обвивката на полигона няма да доведе до по-добра оптимизацията на полигоните. Ще изчисли точно колко е реалната фира. Общата фира (реална + изрязана) може да се изчисли като разлика между сумата на реалните площи на входящите полигони и площта на запълнения контур.

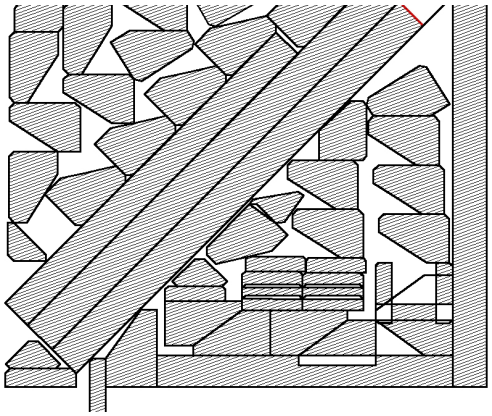
$$waste = A_P - \sum_{i=1}^{n} F_i' \tag{4.10}$$

4.6 Резултати при $2D$ разкрой. Примери.

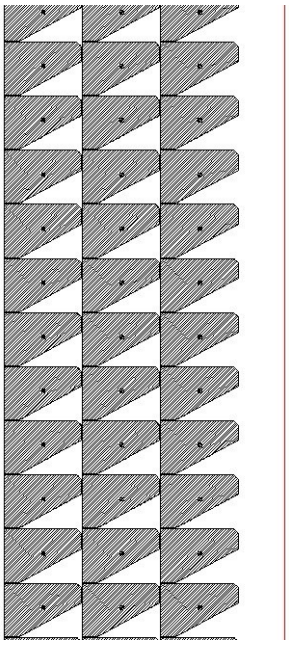
Примери за разкроени планки от реален строителен обект.



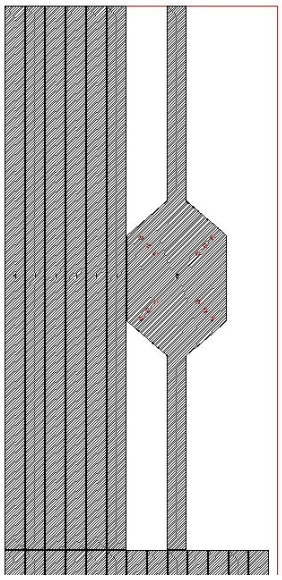
Фигура 4.41: 2D разкрой 1



Фигура 4.42: 2D разкрой 2



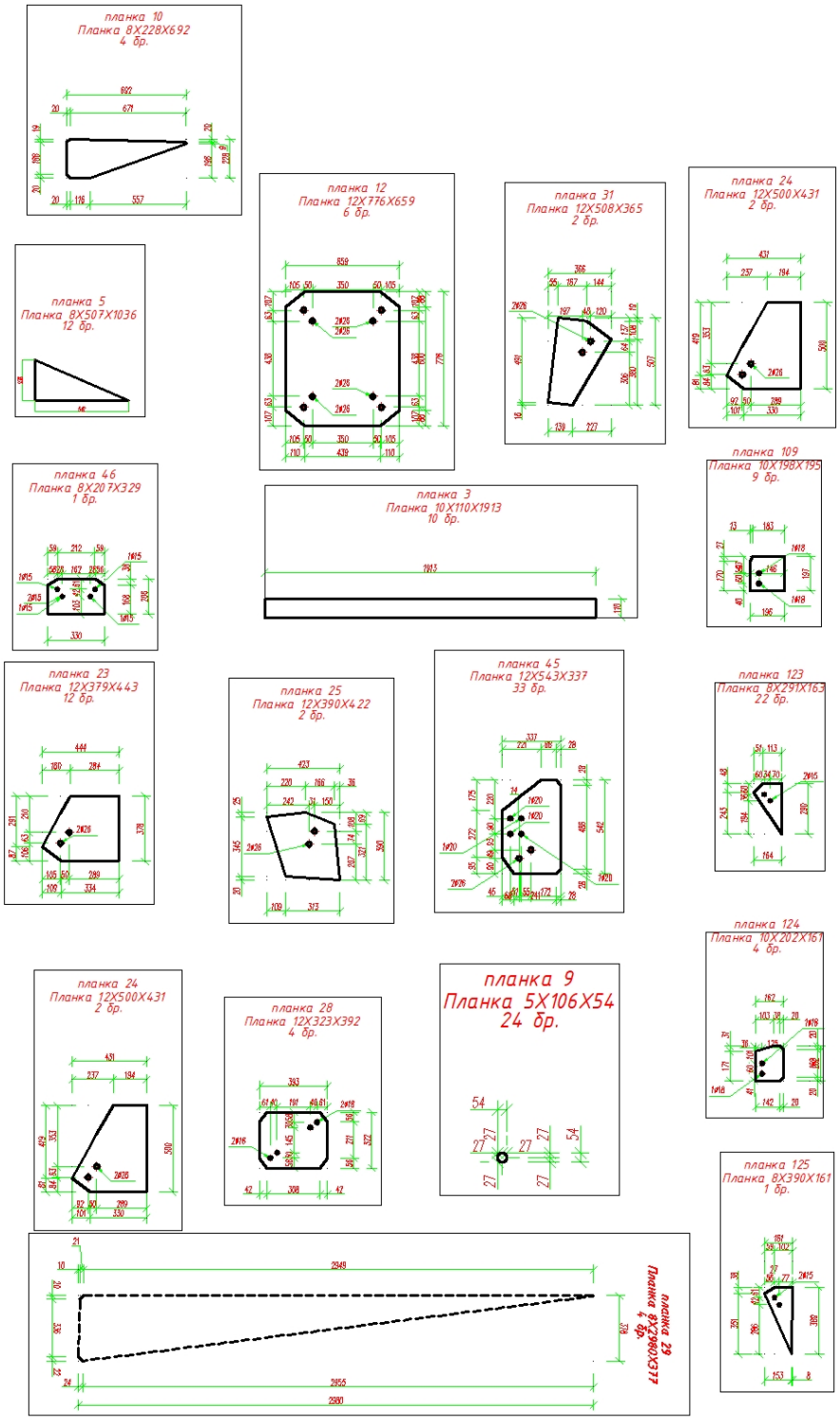
Фигура 4.43: 2D разкрой 3.



Фигура 4.44: 2D разкрой 4.

Фигури 4.41, 4.42, 4.43 и 4.44 илюстрират някои примери на получения разкрой, който е резултат от работата на разработената в дисертацията софтуерна изчислителна среда.

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



Фигура 4.45: Входящи полигони.

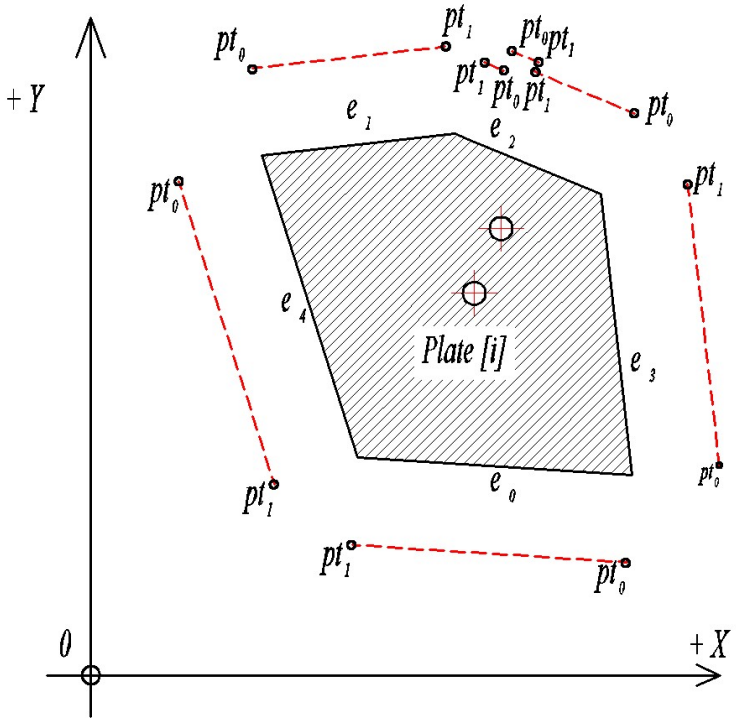
Полигонът за запълване ще бъде стандартен правоъгълен лист със широчина 1500мм. За входящите полигони ще покажем резултати с контурната линия на полигона отместена със широчината на ножа. Ширина на ножа 5мм. Размерите на планките са в милиметри. Преди да започнем изчисление на входящите полигони те минават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

1. Сортиране на планките по дебелина;

2. Прочитане на бройката им;

3. Намиране на контура на планките.

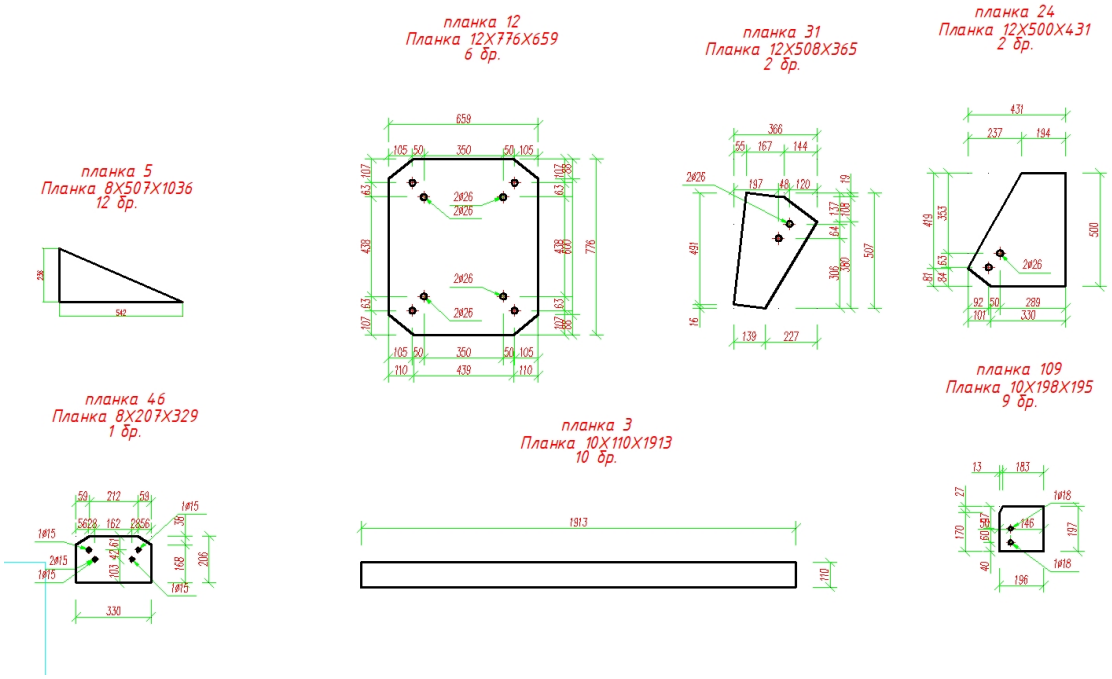
Функцията намиране на контура на планките е необходима тъй като от CAD системата контура им се изчертават като отделни сегменти $list_i = (pt_0, pt_1)$, тъй като те са проекции от 3D модел на планката. Тази обработка на контурите спестява много човешка работа за изчертаване то им като $PolyLine$. Полилинията ($PolyLine$) в CAD системата е цикличен списък от точки, строго подреден. Докато изчертаната планка чрез списък от сегменти не е строго подреден и края на единия сегмент може да не отговаря за началото на следващия сегмент. Виж фигура 4.46.



Фигура 4.46: Входящ полигон: съставни сегменти

.

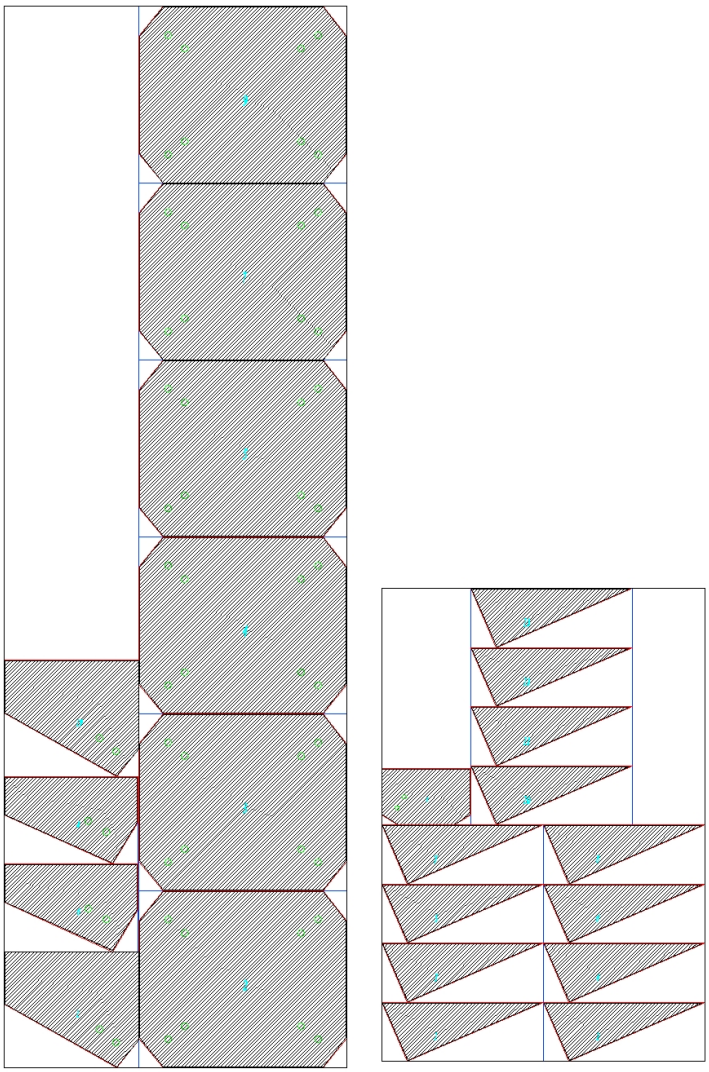
Да вземем сегмент $e_0 = list(pt_1, pt_0)$ и сегмент $e_1 = list(pt_0, pt_1)$. Това, че имат последователни индекси не ги прави последователни в чертежа. Следователно трябва да се намери съседния сегмент на сегмент e_0. Те могат да бъдат два на брой. В случая това са сегменти $e_3 = list(pt_1, pt_0)$ и $e_4 = list(pt_0, pt_1)$. Строго подреденият списък при линейните сегменти не е задължителен. Или $list_1(pt_0, pt_1) = list_1(pt_1, pt_0)$. Следователно ако вземем pt_1 от сегмент e_0, алгоритъма избира кой сегмент е следващия и от списъка $list(pt_1, pt_0)$ избира правилната точка. Така се получава затворен контур и вече може да работим със списък от последователно наредени точки, които не се самопресичат! Тази процедура става автоматично с разработения софтуер от автора. За тестване на алгоритъма ще тестваме с два полигона за всяка входяща планка. Единият е отместен с дебелината на ножа от основния контур, а другия ще бъде box на полигона. Изчисленията ще се изпълняват 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.



Фигура 4.47: Входящи планки

Пример 1: Параметри на настройката за рязане от лист с дебелина 8 на планка #46, и 12 броя планка #5. От лист с дебелина 12 на 6 броя входяща планка #12 и 2 броя планки #31. Виж фигура 4.47:

Контур	Огледален образ	Ротация	Подробни точки
box	Не	Не	Не



Подредени планки: 10.Дебелина t= 12.
Използван контур за подреждане: Правоъгълник
Огледален образ на планките: Не
Ротиране на планките: Не
Редуцирани върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 19 577. Ширина на ножа : 5.
Площ на фирата от ножа: 48 942
Площ на планките: 3 507 151.
Площ на планките допълнени до правоъгълник: 5 124 963

Коеф. за запълване (ratio) R= 0.69 = (3507151+48942) / 5124963
Time: 3s.

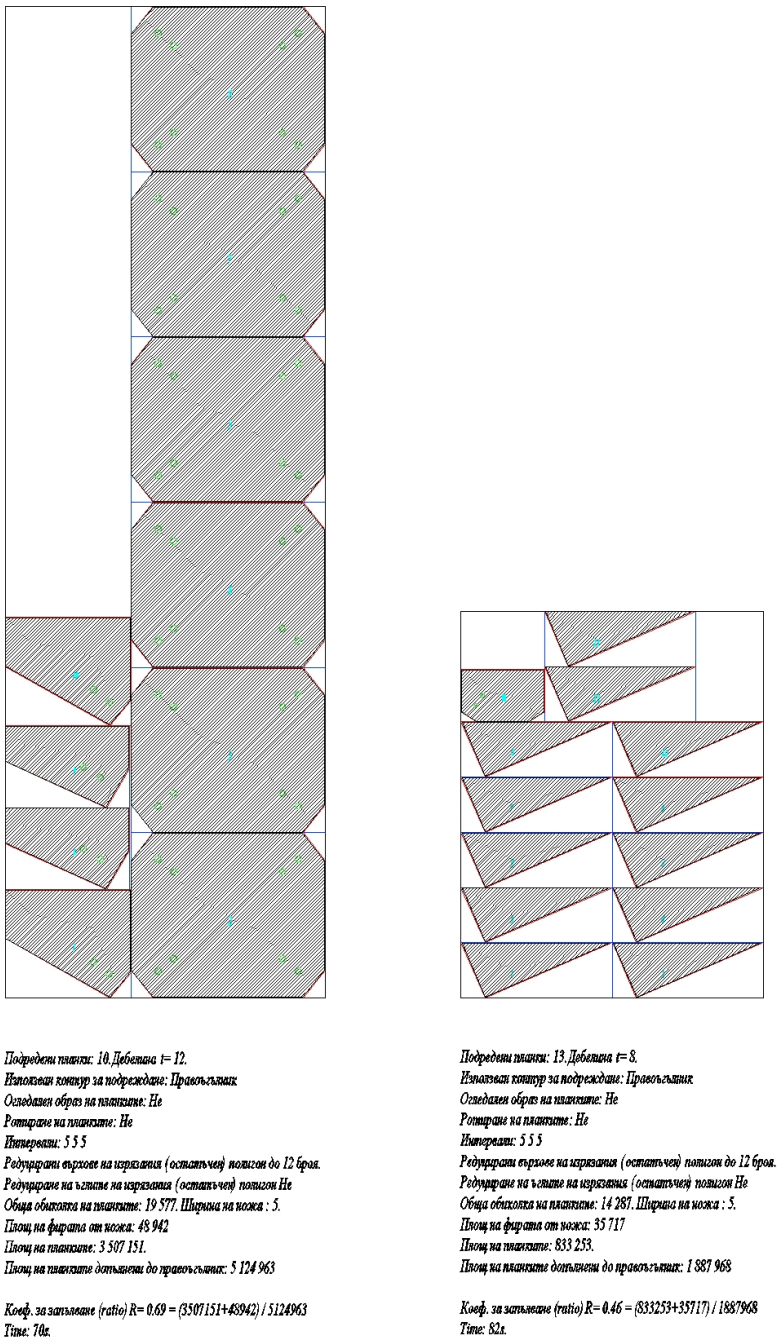
Подредени планки: 13.Дебелина t= 8.
Използван контур за подреждане: Правоъгълник
Огледален образ на планките: Не
Ротиране на планките: Не
Редуцирани върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 14 287. Ширина на ножа : 5.
Площ на фирата от ножа: 35 717
Площ на планките: 833 253.
Площ на планките допълнени до правоъгълник: 2 157 677

Коеф. за запълване (ratio) R= 0.4 = (833253+35717) / 2157677
Time: 3s.

Фигура 4.48: Резултати 1

Пример 2: Същите планки от фигура 4.48. При следните параметри:

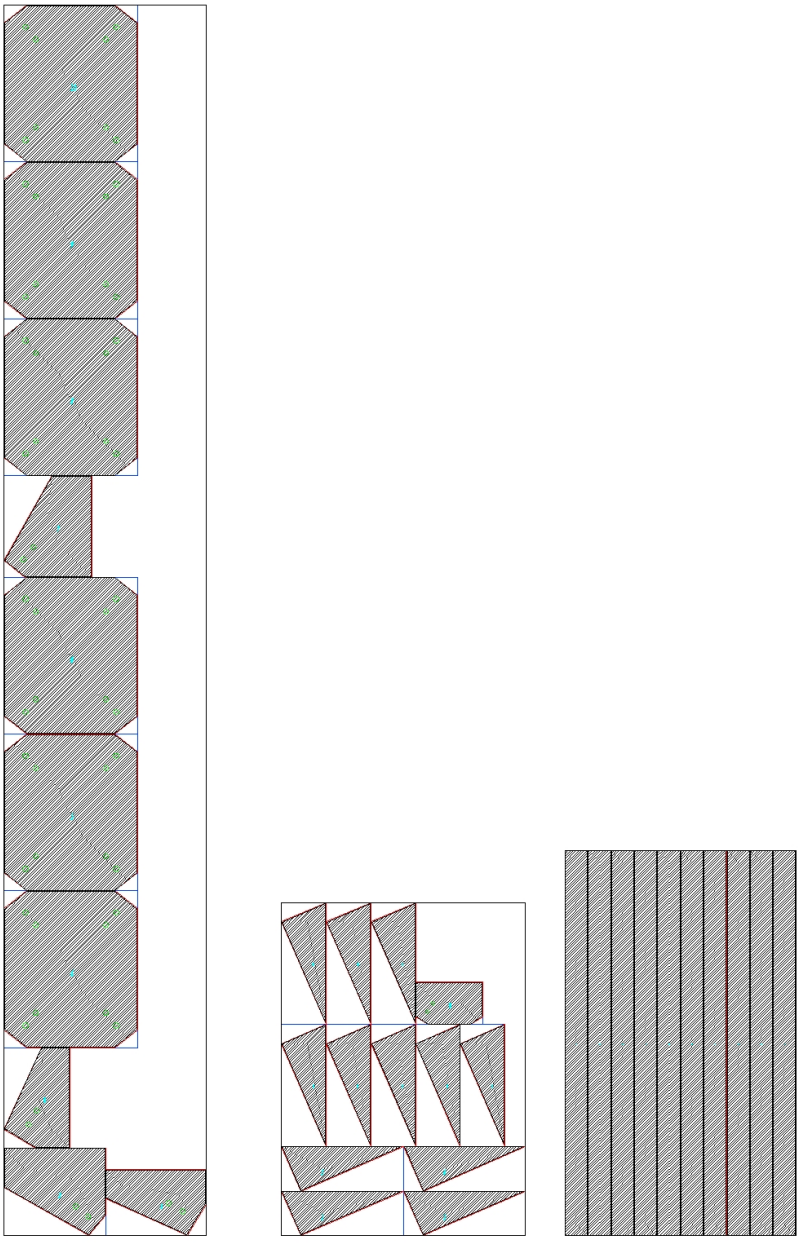
Контур	Огледален образ	Ротация	Подробни точки
box	Не	Не	Да: интервалите се делят $list(555)$



Фигура 4.49: Резултати 2

Пример 3: Същите планки от фигура 4.47, при следните параметри:

Контур	Огледален образ	Ротация	Подробни точки
box	Не	Да	Да: интервалите се делят *list*(535)



Фигура 4.50: Резултати 3

Огледален образ на планки, който са с контур *box* няма смисъл да се прави. Обобщени данни.

Таблица 4.1: Обобщени резултати с контур *box*.

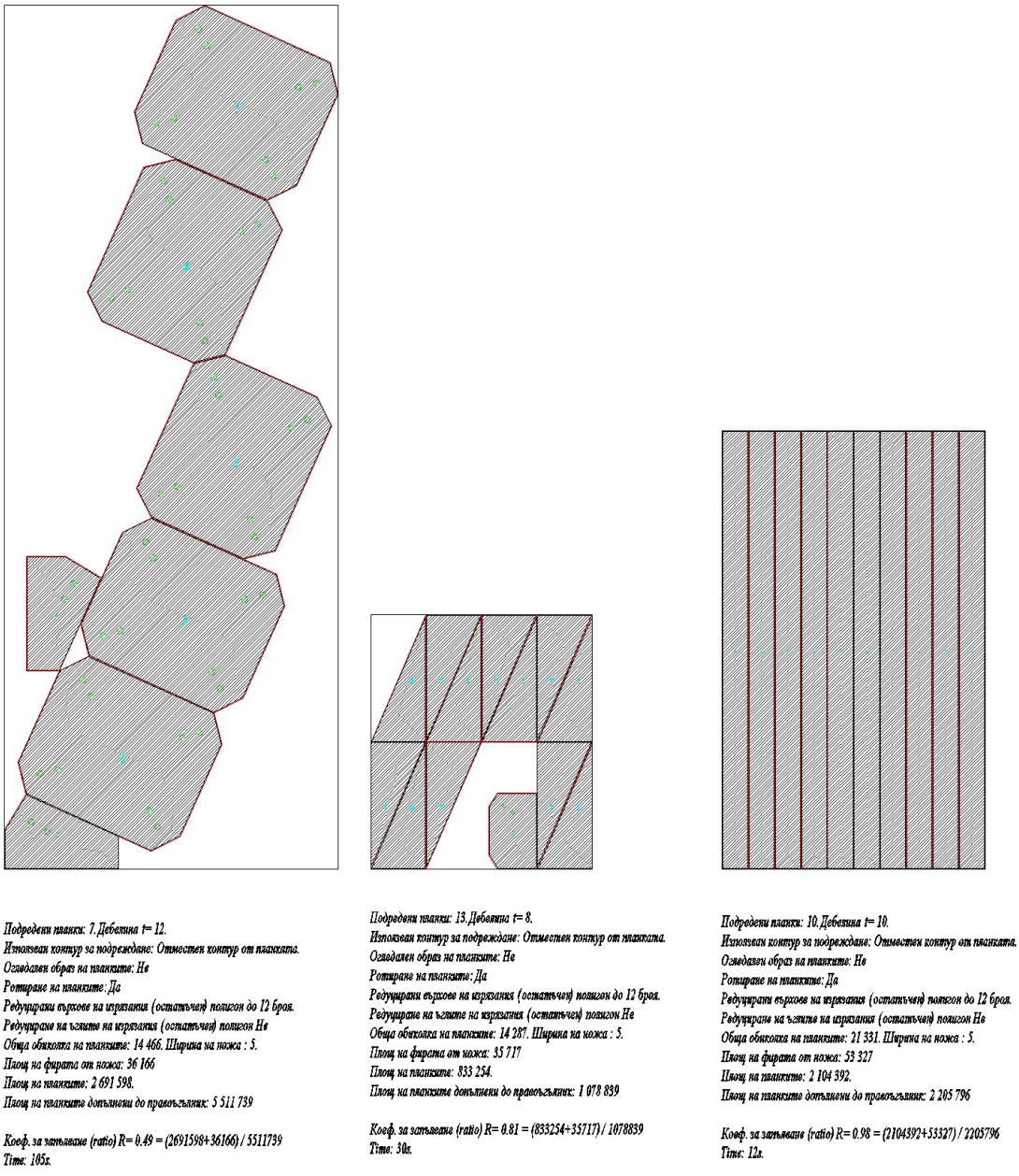
Включени параметри	Дебелини	Брой планки	Ratio	Време [s]
1	2	3	4	5
Mirror:No, Rortate:No, Intervals:No	$8mm$	13	0.40	3
Mirror:No, Rortate:No, Intervals:No	$10mm$	N/A	N/A	N/A
Mirror:No, Rortate:No, Intervals:No	$12mm$	10	0.69	3
Mirror:No, Rortate:No, Intervals:*list*(535)	$8mm$	13	0,46	82
Mirror:No, Rortate:No, Intervals:*list*(535)	$10mm$	N/A	N/A	N/A
Mirror:No, Rortate:No, Intervals:*list*(535)	$12mm$	10	0,69	70
Mirror:No, Rortate:Yes, Intervals:*list*(535)	$8mm$	13	0,43	166
Mirror:No, Rortate:Yes, Intervals:*list*(535)	$10mm$	10	0,98	64
Mirror:No, Rortate:Yes, Intervals:*list*(535)	$12mm$	10	0,58	114

N/A - означава, че не е намерено решение за дадения лист за запълване.

Вижда се, че при увеличаване на параметрите някой решения се подобряват, а при изключени параметри някои планки нямат решение в този лист за запълване. Вижда се, че *Ratio* за всички планки (без значение от дебелината) е относително еднакво докато времето нараства с геометрична прогресия. Това означава, че в зависимост от цената на материала, потребителят ще трябва да прави баланс между *Ratio* и време.

Пример 4: Същите планки от фигура 4.47. При следните параметри:

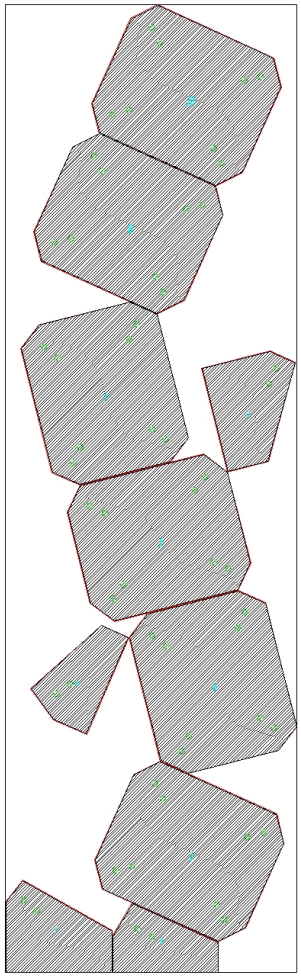
Контур	Огледален образ	Ротация	Подробни точки
$Offset$	Не	Да	Не

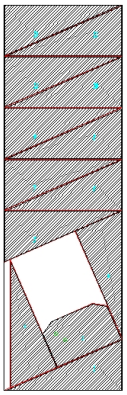


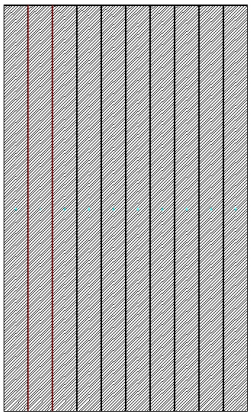
Фигура 4.51: Резултати 4

Пример 5: Същите планки от фигура 4.47. При следните параметри:

Контур	Огледален образ	Ротация	Подробни точки
$Offset$	Да	Да	Не







Подредени планки: 10. Дебелина t= 12.
Използван контур за подреждане: Отместен контур от планката.
Огледален образ на планките: Да
Ротиране на планките: Да
Редуциране върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 19 577. Ширина на ножа : 5.
Площ на фигурата от ножа: 48 942
Площ на планките: 3 507 151.
Площ на планките допълнени до правоъгълник: 6 276 986

Коеф. за запълване (ratio) R= 0.57 = (3507151+48942) / 6276986
Time: 1056s.

Подредени планки: 13. Дебелина t= 8.
Използван контур за подреждане: Отместен контур от планката.
Огледален образ на планките: Да
Ротиране на планките: Да
Редуциране върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 14 287. Ширина на ножа : 5.
Площ на фигурата от ножа: 35 717
Площ на планките: 833 253.
Площ на планките допълнени до правоъгълник: 1 012 061

Коеф. за запълване (ratio) R= 0.86 = (833253+35717) / 1012061
Time: 58s.

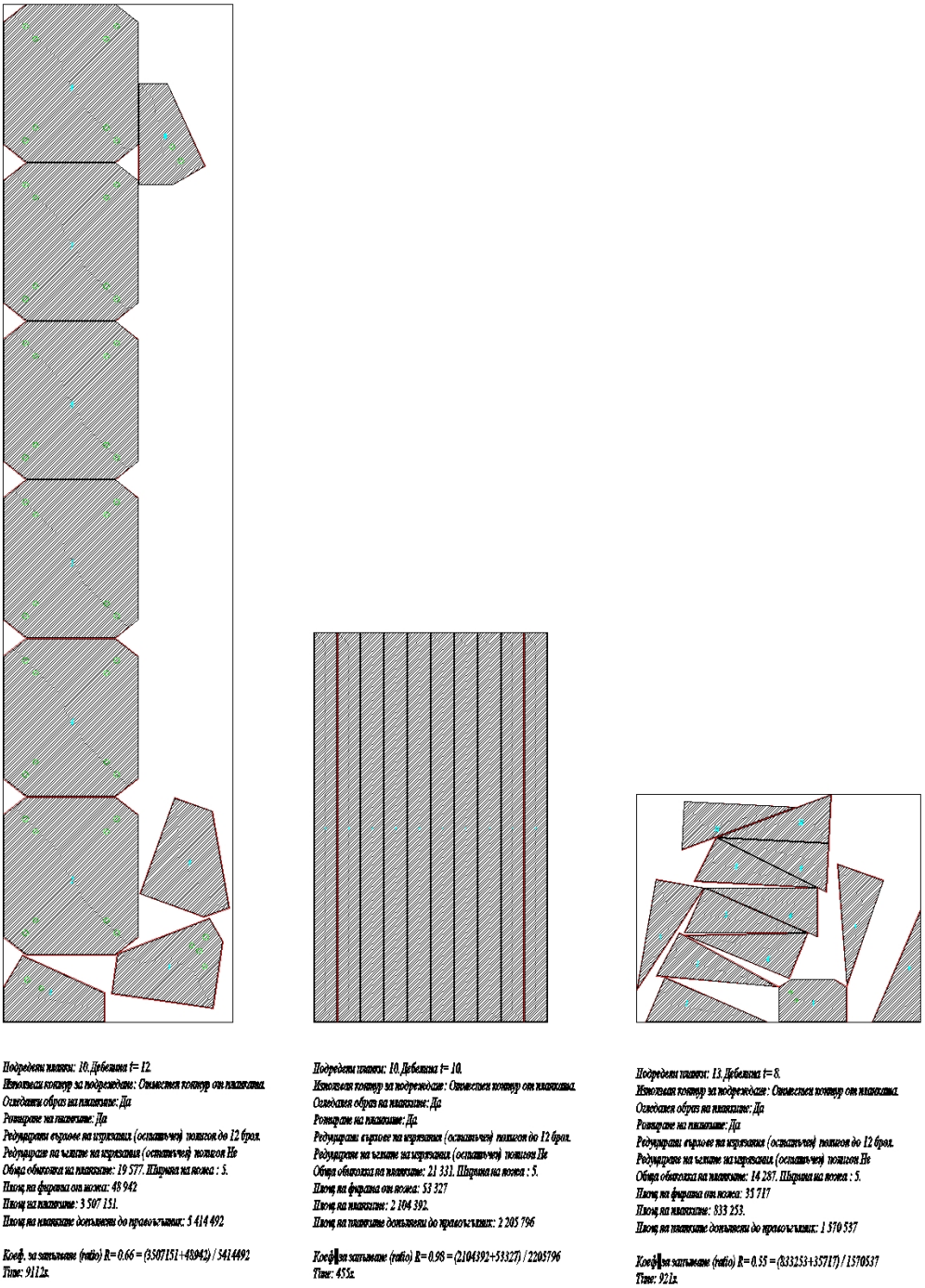
Подредени планки: 10. Дебелина t= 10.
Използван контур за подреждане: Отместен контур от планката.
Огледален образ на планките: Да
Ротиране на планките: Да
Редуциране върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 21 331. Ширина на ножа : 5.
Площ на фигурата от ножа: 53 327
Площ на планките: 2 104 392.
Площ на планките допълнени до правоъгълник: 2 205 796

Коеф. за запълване (ratio) R= 0.98 = (2104392+53327) / 2205796
Time: 37s.

Фигура 4.52: Резултати 5

Пример 6: Същите планки от фигура 4.47. При следните параметри:

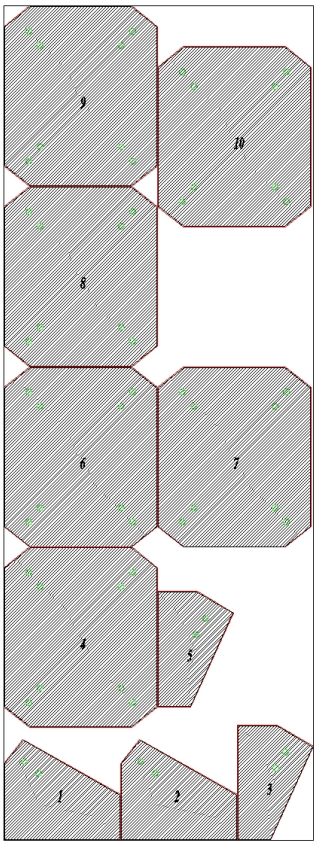
Контур	Огледален образ	Ротация	Подробни точки
$Offset$	Не	Да	Да: интервалите се делят $list(535)$



Фигура 4.53: Резултати 6

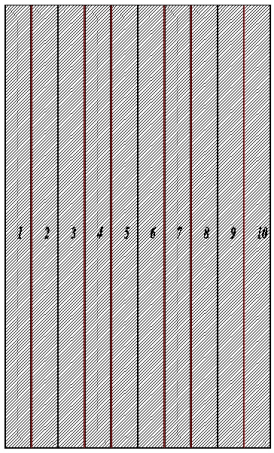
Пример 7: Същите планки от фигура 4.47. При следните параметри:

Контур	Огледален образ	Ротация	Подробни точки
$Offset$	Да	Да	Да: интервалите се делят $list(511)$



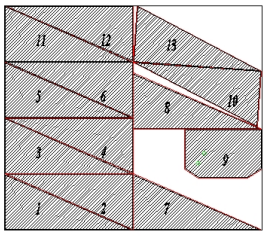
Подредени планки: 10. Дебелина t= 12.
Използван контур за подреждане: Отместен контур от планката.
Огледален образ на планките: Не
Ротиране на планките: Да
Разделяне сегментите на интервали: 5 1 1
Тест на всички уникални планки: Да
Редуцирани върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 19 577. Ширина на ножа : 5.
Площ на фирата от ножа: 48 942
Площ на планките: 3 507 151.
Площ на планките допълнени до правоъгълник: 4 832 604

Коеф. за запълване (ratio) R= 0.74 = (3507151+48942) / 4832604
Time: 1974s.



Подредени планки: 10. Дебелина t= 10.
Използван контур за подреждане: Отместен контур от планката.
Огледален образ на планките: Не
Ротиране на планките: Да
Разделяне сегментите на интервали: 5 1 1
Тест на всички уникални планки: Да
Редуцирани върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 21 331. Ширина на ножа : 5.
Площ на фирата от ножа: 53 327
Площ на планките: 2 104 392.
Площ на планките допълнени до правоъгълник: 2 205 796

Коеф. за запълване (ratio) R= 0.98 = (2104392+53327) / 2205796
Time: 351s.



Подредени планки: 13. Дебелина t= 8.
Използван контур за подреждане: Отместен контур от планката.
Огледален образ на планките: Не
Ротиране на планките: Да
Разделяне сегментите на интервали: 5 1 1
Тест на всички уникални планки: Да
Редуцирани върхове на изрязания (остатъчен) полигон до 12 броя.
Редуциране на ъглите на изрязания (остатъчен) полигон Не
Обща обиколка на планките: 14 287. Ширина на ножа : 5.
Площ на фирата от ножа: 35 717
Площ на планките: 833 253.
Площ на планките допълнени до правоъгълник: 1 078 838

Коеф. за запълване (ratio) R= 0.81 = (833253+35717) / 1078838
Time: 294s.

Фигура 4.54: Резултати 7

Таблица 4.2: Обобщени резултати с контур $Offset$.

Включени параметри	Дебелини	Брой планки	Ratio	Време [s]
1	2	3	4	5
Mirror:No, Rortate:Yes, Intervals:No	$8mm$	13	0.81	30
Mirror:No, Rortate:Yes, Intervals:No	$10mm$	10	0.98	12
Mirror:No, Rortate:Yes, Intervals:No	$12mm$	10	0.49	105
Mirror:Yes, Rortate:Yes, Intervals:$list$(535)	$8mm$	13	0,55	921
Mirror:Yes, Rortate:Yes, Intervals:$list$(535)	$10mm$	10	0.98	455
Mirror:Yes, Rortate:Yes, Intervals:$list$(535)	$12mm$	10	0.66	9112
Mirror:No, Rortate:Yes, Intervals:$list$(511)	$8mm$	13	0,81	294
Mirror:No, Rortate:Yes, Intervals:$list$(511)	$10mm$	10	0.98	351
Mirror:No, Rortate:Yes, Intervals:$list$(511)	$12mm$	10	0.74	1974

Резултати сортирани по дебелини на планките:

Таблица 4.3: Обобщени резултати за дебелина 8mm

Включени параметри	Контур	Брой планки	Ratio	Време [s]
1	2	3	4	5
Mirror:No, Rortate:No, Intervals:No	box	13	0.40	3
Mirror:No, Rortate:No, Intervals:$list$(555)	box	10	0.46	82
Mirror:No, Rortate:Yes, Intervals:$list$(535)	box	10	0.43	166
Mirror:No, Rortate:Yes, Intervals:No	$Offset$	13	0.81	30
Mirror:Yes, Rortate:Yes, Intervals:$list$(535)	$Offset$	10	0.55	921
Mirror:No, Rortate:Yes, Intervals:$list$(511)	$Offset$	10	0.81	294

Таблица 4.4: Обобщени резултати за дебелина 10mm

Включени параметри	Контур	Брой планки	Ratio	Време [s]
1	2	3	4	5
Mirror:No, Rortate:No, Intervals:No	box	N/A	N/A	N/A
Mirror:No, Rortate:No, Intervals:$list$(555)	box	N/A	N/A	N/A
Mirror:No, Rortate:Yes, Intervals:$list$(535)	box	10	0.98	64
Mirror:No, Rortate:Yes, Intervals:No	$Offset$	13	0.98	12
Mirror:Yes, Rortate:Yes, Intervals:$list$(535)	$Offset$	10	0.98	455
Mirror:No, Rortate:Yes, Intervals:$list$(511)	$Offset$	10	0.98	351

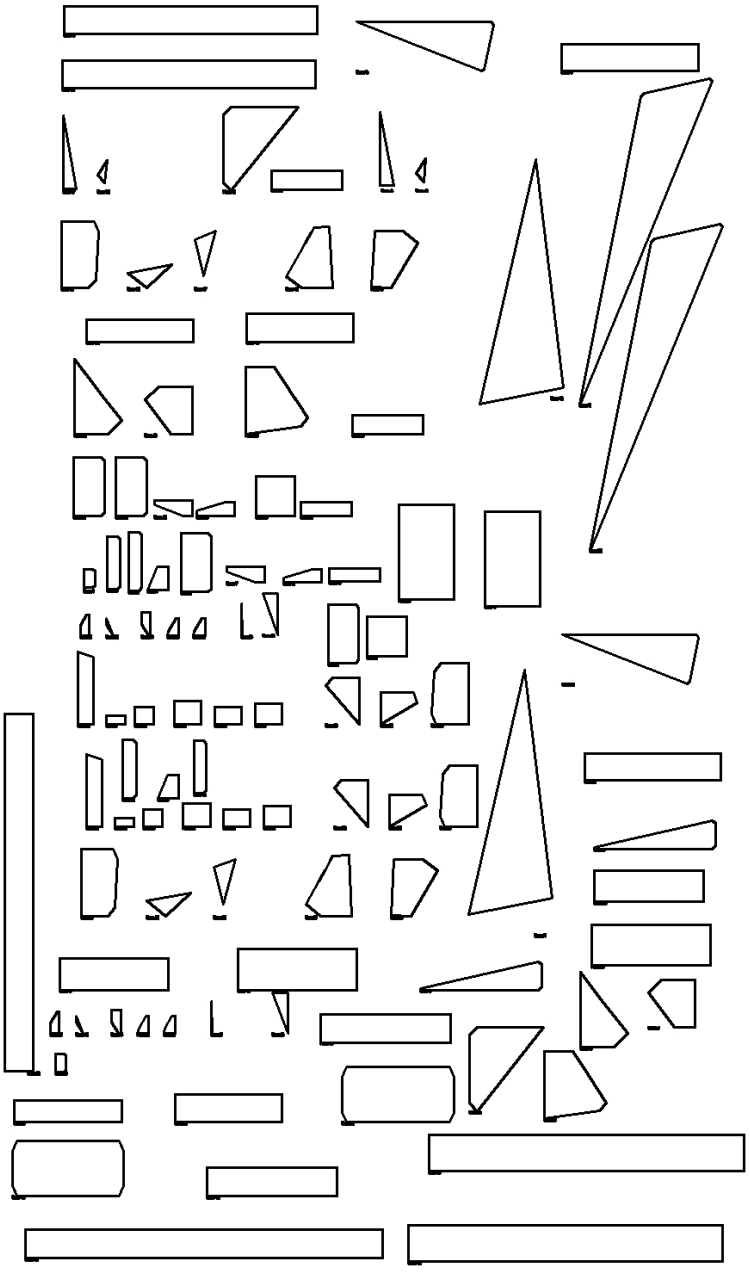
Таблица 4.5: Обобщени резултати за дебелина 12mm

Включени параметри	Контур	Брой планки	Ratio	Време [s]
1	2	3	4	5
Mirror:No, Rortate:No, Intervals:No	box	10	0.69	3
Mirror:No, Rortate:No, Intervals:$list$(555)	box	10	0.69	70
Mirror:No, Rortate:Yes, Intervals:$list$(535)	box	10	0.58	114
Mirror:No, Rortate:Yes, Intervals:No	$Offset$	13	0.49	105
Mirror:Yes, Rortate:Yes, Intervals:$list$(535)	$Offset$	10	0.66	9112
Mirror:No, Rortate:Yes, Intervals:$list$(511)	$Offset$	10	0.74	1974

Сравняване на настоящия алгоритъм с комерсиален продукт.

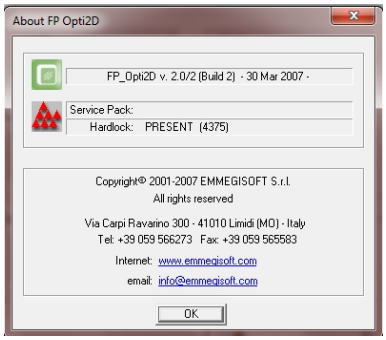
За изработването на една стоманена конструкция от фигура 1.1 отнема около три месеца. Брои планки в една такава конструкция е около 1000. Брои уникални планки около 100. Броя итерации е относителен. Той зависи от това дали материалът е скъп или не е. Могат да се пуснат няколко итерации на различни компютри 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 4.64. Общия брои 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.



Фигура 4.55: Входящи планки за разкрой.

Използван комерсиален продукт.



Фигура 4.56: 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.

Optimization Printout v. Date/Time: 28.11.2018 г. 09:50:52 - Page 1

Opti2D (s.n. 4375)

WORKORDER

Work Order	Planki St 28.11.2018.R01		
Status	Optimized		
Date	28.11.2018 г.		
U.M.	mm	Kg	mq
Description	Planki St 28.11.2018		

MATERIAL

Material	PLANKI_STOMANA
Full Description	
Thickness	0,0 mm
Weight	0.00 Kg

OPTIMIZATION RESULTS

Number of Pieces	106		
Number of Positions	106		
Total Area		14,72 mq	
Number of Used Pieces	106		
Used Piece Area	100,00 %	14,72 mq	0.00 Kg
Cutting Plans	2		
Used Sheet Area		18,00 mq	
Sold Pieces		16,16 mq	
Real Waste	18,15 %	3,27 mq	0.00 Kg
Trim Waste	0,00 %	0,00 mq	0.00 Kg
Total Waste	18,22 %	3,28 mq	0.00 Kg
Tool Waste	0,00 %	0,00 mq	0.00 Kg
Reusable Waste	0,00 %	0,00 mq	0.00 Kg
Final Waste	18,22 %	3,28 mq	0.00 Kg
Linear Cutting Length	102,51 m		
Fast Movement	172,15 m		

PARAMETERS

X1 / Y1 Trims	0,0 mm	0,0 mm
X2 / Y2 Trims	0,0 mm	0,0 mm
Minimum Distance	0,0 mm	
X Crosscut	9999,0 mm	
Y Crosscut	9999,0 mm	
X/Y Waste	99999999,9	99999999,9
X1/X2 Bars	0,0 mm	0,0 mm
Y1/Y2 Bars	0,0 mm	0,0 mm
X3 Bar	0,0 mm	
X/Y Tool Exit	0,0 mm	0,0 mm
Tool	0,0 mm	
Crossing cuts	1,0 mm	

AVAILABLE SHEETS LIST

Pos.	X Dim.	Y Dim.	X/Y	Waste	Rack	Available	Used	Remaining	Used	Tot. Wst.
1	6000,0 mm	1500,0 mm	Y	No		999	2	997	18,00 mq	18,22 %

Фигура 4.57: Лист 1 от разпечатката на комерсиалния продукт.

Както се вижда от разпечатката комерсиалния продукт версия V.2.0/2 (Build 2) работи само с правоъгълни планки. Разрешено е завъртане на планките. По

Optimization Printout v. Date/Time: 28.11.2018 г. 09:50:53 - Page 2

Opti2D (s.n. 4375)

WORKORDER				PIECES	
Work Order	Planki St 28.11.2018.R01			*Number of Pieces*	106
Status	Optimized			*Number of Positions*	106
Date	28.11.2018 г.			*Total Area*	14,72 mq
U.M.	mm	Kg	mq	*Sold Area*	16,16 mq
Description	Planki St 28.11.2018			*Exceeding pieces*	0

MATERIAL	
Material	PLANKI_STOMANA
Full Description	
Thickness	0,0 mm
Weight	0.00 Kg

LIST OF PIECES TO CUT

Pos.	Id.	Ins.	Us.	±	Rot.	X Dim.	Y Dim.	Shape	Customer	Order	Rack
1	Планка 106	1	1	0	Yes	180,0 mm	2273,0 mm				
2	Планка 105	1	1	0	Yes	230,0 mm	2000,0 mm				
3	Планка 104	1	1	0	Yes	180,0 mm	1613,0 mm				
4	Планка 103	1	1	0	Yes	180,0 mm	826,0 mm				
5	Планка 102	1	1	0	Yes	351,0 mm	709,0 mm				
6	Планка 101	1	1	0	Yes	180,0 mm	680,0 mm				
7	Планка 100	1	1	0	Yes	140,0 mm	680,0 mm				
8	Планка 99	1	1	0	Yes	471,0 mm	535,0 mm				
9	Планка 98	1	1	0	Yes	61,0 mm	146,0 mm				
10	Планка 97	1	1	0	Yes	85,0 mm	469,0 mm				
11	Планка 96	1	1	0	Yes	120,0 mm	450,0 mm				
12	Планка 95	1	1	0	Yes	392,0 mm	428,0 mm				
13	Планка 94	1	1	0	Yes	299,0 mm	300,0 mm				
14	Планка 93	1	1	0	Yes	299,0 mm	475,0 mm				
15	Планка 92	1	1	0	Yes	295,0 mm	362,0 mm				
16	Планка 91	1	1	0	Yes	293,0 mm	388,0 mm				
17	Планка 90	1	1	0	Yes	134,0 mm	285,0 mm				
18	Планка 89	1	1	0	Yes	148,0 mm	280,0 mm				
19	Планка 88	1	1	0	Yes	234,0 mm	423,0 mm				
20	Планка 87	1	1	0	Yes	234,0 mm	388,0 mm				
21	Планка 86	1	1	0	Yes	203,0 mm	234,0 mm				
22	Планка 85	1	1	0	Yes	217,0 mm	299,0 mm				
23	Планка 84	1	1	0	Yes	134,0 mm	170,0 mm				
24	Планка 83	1	1	0	Yes	110,0 mm	170,0 mm				
25	Планка 82	1	1	0	Yes	150,0 mm	170,0 mm				
26	Планка 81	1	1	0	Yes	112,0 mm	120,0 mm				
27	Планка 80	1	1	0	Yes	55,0 mm	120,0 mm				
28	Планка 79	1	1	0	Yes	96,0 mm	461,0 mm				
29	Планка 78	1	1	0	Yes	96,0 mm	259,0 mm				
30	Планка 77	1	1	0	Yes	7,0 mm	202,0 mm				
31	Планка 76	1	1	0	Yes	73,0 mm	110,0 mm				
32	Планка 75	1	1	0	Yes	68,0 mm	110,0 mm				
33	Планка 74	1	1	0	Yes	56,0 mm	150,0 mm				
34	Планка 73	1	1	0	Yes	42,0 mm	110,0 mm				
35	Планка 72	1	1	0	Yes	56,0 mm	134,0 mm				
36	Планка 71	1	1	0	Yes	528,0 mm	1556,0 mm				
37	Планка 70	1	1	0	Yes	170,0 mm	869,0 mm				

Фигура 4.58: Лист 2 от разпечатката на комерсиалния продукт.

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

Optimization Printout v. Date/Time: 28.11.2018 г. 09:50:53 - Page 3

Opti2D (s.n. 4375)

LIST OF PIECES TO CUT

Pos.	Id.	Ins.	Us.	±	Rot.	X Dim.	Y Dim.	Shape	Customer	Order	Rack
38	Планка 69	1	1	0	Yes	309,0 mm	869,0 mm				
39	Планка 68	1	1	0	Yes	845,0 mm	2070,0 mm				
40	Планка 67	1	1	0	Yes	178,0 mm	772,0 mm				
41	Планка 66	1	1	0	Yes	260,0 mm	755,0 mm				
42	Планка 65	1	1	0	Yes	200,0 mm	690,0 mm				
43	Планка 64	1	1	0	Yes	350,0 mm	600,0 mm				
44	Планка 63	1	1	0	Yes	85,0 mm	322,0 mm				
45	Планка 62	1	1	0	Yes	250,0 mm	250,0 mm				
46	Планка 61	1	1	0	Yes	85,0 mm	240,0 mm				
47	Планка 60	1	1	0	Yes	100,0 mm	240,0 mm				
48	Планка 59	1	1	0	Yes	196,0 mm	373,0 mm				
49	Планка 58	1	1	0	Yes	196,0 mm	373,0 mm				
50	Планка 57	1	1	0	Yes	130,0 mm	150,0 mm				
51	Планка 56	1	1	0	Yes	86,0 mm	373,0 mm				
52	Планка 55	1	1	0	Yes	81,0 mm	335,0 mm				
53	Планка 54	1	1	0	Yes	67,0 mm	116,0 mm				
54	Планка 53	1	1	0	Yes	180,0 mm	2273,0 mm				
55	Планка 52	1	1	0	Yes	230,0 mm	2000,0 mm				
56	Планка 51	1	1	0	Yes	180,0 mm	1613,0 mm				
57	Планка 50	1	1	0	Yes	180,0 mm	826,0 mm				
58	Планка 49	1	1	0	Yes	351,0 mm	709,0 mm				
59	Планка 48	1	1	0	Yes	180,0 mm	680,0 mm				
60	Планка 47	1	1	0	Yes	140,0 mm	680,0 mm				
61	Планка 46	1	1	0	Yes	471,0 mm	535,0 mm				
62	Планка 45	1	1	0	Yes	61,0 mm	146,0 mm				
63	Планка 44	1	1	0	Yes	85,0 mm	469,0 mm				
64	Планка 43	1	1	0	Yes	120,0 mm	450,0 mm				
65	Планка 42	1	1	0	Yes	392,0 mm	428,0 mm				
66	Планка 41	1	1	0	Yes	299,0 mm	300,0 mm				
67	Планка 40	1	1	0	Yes	299,0 mm	475,0 mm				
68	Планка 39	1	1	0	Yes	295,0 mm	362,0 mm				
69	Планка 38	1	1	0	Yes	293,0 mm	388,0 mm				
70	Планка 37	1	1	0	Yes	134,0 mm	285,0 mm				
71	Планка 36	1	1	0	Yes	148,0 mm	280,0 mm				
72	Планка 35	1	1	0	Yes	234,0 mm	423,0 mm				
73	Планка 34	1	1	0	Yes	234,0 mm	388,0 mm				
74	Планка 33	1	1	0	Yes	203,0 mm	234,0 mm				
75	Планка 32	1	1	0	Yes	217,0 mm	299,0 mm				
76	Планка 31	1	1	0	Yes	134,0 mm	170,0 mm				
77	Планка 30	1	1	0	Yes	110,0 mm	170,0 mm				
78	Планка 29	1	1	0	Yes	150,0 mm	170,0 mm				
79	Планка 28	1	1	0	Yes	112,0 mm	120,0 mm				
80	Планка 27	1	1	0	Yes	55,0 mm	120,0 mm				
81	Планка 26	1	1	0	Yes	96,0 mm	461,0 mm				
82	Планка 25	1	1	0	Yes	96,0 mm	259,0 mm				
83	Планка 24	1	1	0	Yes	7,0 mm	202,0 mm				
84	Планка 23	1	1	0	Yes	73,0 mm	110,0 mm				

Фигура 4.59: Лист 3 от разпечатката на комерсиалния продукт.

разпечатката. Отпадък 18.2%. Всички планки са с отместен контур на 5mm. Виж фигура 4.39. С този контур ще работим в изчисленията. Реалният отпадък може

Optimization Printout v. Date/Time: 28.11.2018 г. 09:50:53 - Page 4

Opti2D (s.n. 4375)

LIST OF PIECES TO CUT

Pos.	Id.	Ins.	Us.	±	Rot.	X Dim.	Y Dim.	Shape	Customer	Order	Rack
85	Планка 22	1	1	0	Yes	68,0 mm	110,0 mm				
86	Планка 21	1	1	0	Yes	56,0 mm	150,0 mm				
87	Планка 20	1	1	0	Yes	42,0 mm	110,0 mm				
88	Планка 19	1	1	0	Yes	56,0 mm	134,0 mm				
89	Планка 18	1	1	0	Yes	528,0 mm	1556,0 mm				
90	Планка 17	1	1	0	Yes	170,0 mm	869,0 mm				
91	Планка 16	1	1	0	Yes	309,0 mm	869,0 mm				
92	Планка 15	1	1	0	Yes	845,0 mm	2070,0 mm				
93	Планка 14	1	1	0	Yes	178,0 mm	772,0 mm				
94	Планка 13	1	1	0	Yes	260,0 mm	755,0 mm				
95	Планка 12	1	1	0	Yes	200,0 mm	690,0 mm				
96	Планка 11	1	1	0	Yes	350,0 mm	600,0 mm				
97	Планка 10	1	1	0	Yes	85,0 mm	322,0 mm				
98	Планка 9	1	1	0	Yes	250,0 mm	250,0 mm				
99	Планка 8	1	1	0	Yes	85,0 mm	240,0 mm				
100	Планка 7	1	1	0	Yes	100,0 mm	240,0 mm				
101	Планка 6	1	1	0	Yes	196,0 mm	373,0 mm				
102	Планка 5	1	1	0	Yes	196,0 mm	373,0 mm				
103	Планка 4	1	1	0	Yes	130,0 mm	150,0 mm				
104	Планка 3	1	1	0	Yes	86,0 mm	373,0 mm				
105	Планка 2	1	1	0	Yes	81,0 mm	335,0 mm				
106	Планка 1	1	1	0	Yes	67,0 mm	116,0 mm				

Фигура 4.60: Лист 4 от разпечатката на комерсиалния продукт.

да се изчисли както следва:

1. Обща използвана площ - 2 листа x 1500mm x 6000mm = 18 000 000 ^{2};

Optimization Printout v. Date/Time: 28.11.2018 г. 09:50:53 - Page 5

Opti2D (s.n. 4375)

WORKORDER			
Work Order	Planki St 28.11.2018.R01		
Description	Planki St 28.11.2018		
Date	28.11.2018 г.		
U.M.	mm	Kg	mq
Status	Optimized		

MATERIAL	
Material	PLANKI_STOMANA
Full Description	
Thickness	0,0 mm
Weight	0.00 Kg

MATERIAL PARAMETERS	
X1 Trim	0,0 mm
Y1 Trim	0,0 mm
X2 Trim	0,0 mm
Y2 Trim	0,0 mm
Minimum Distance	0,0 mm
X Crc.	9999,0 mm
Y Crc.	9999,0 mm
Min. X Waste	99999999,9
Min. Y Waste	99999999,9

CUTTING PLANS LIST

Pos.	X Dim.	Y Dim.	Rep.	X/Y	Waste	Rack	Prior.	Tot. Wst.	Area	Block
1	6000,0 mm	1500,0 mm	1	Y	No		0	4,26%	0,38 mq	1
2	6000,0 mm	1500,0 mm	1	Y	No		0	32,18%	2,90 mq	2

LIST OF PIECES TO CUT

Pos.	Work	Rem.	±	Rot.	X Dim.	Y Dim.	Shape	Customer	Order	Rack	Id.
2	1	0	0	Yes	230,0 mm	2000,0 mm					Планка 105
4	1	0	0	Yes	180,0 mm	826,0 mm					Планка 103
5	1	0	0	Yes	351,0 mm	709,0 mm					Планка 102
12	1	0	0	Yes	392,0 mm	428,0 mm					Планка 95
14	1	0	0	Yes	299,0 mm	475,0 mm					Планка 93
15	1	0	0	Yes	295,0 mm	362,0 mm					Планка 92
16	1	0	0	Yes	293,0 mm	388,0 mm					Планка 91
20	1	0	0	Yes	234,0 mm	388,0 mm					Планка 87
22	1	0	0	Yes	217,0 mm	299,0 mm					Планка 85
29	1	0	0	Yes	96,0 mm	259,0 mm					Планка 78
30	1	0	0	Yes	7,0 mm	202,0 mm					Планка 77
34	1	0	0	Yes	42,0 mm	110,0 mm					Планка 73
37	1	0	0	Yes	170,0 mm	869,0 mm					Планка 70
38	1	0	0	Yes	309,0 mm	869,0 mm					Планка 69
39	1	0	0	Yes	845,0 mm	2070,0 mm					Планка 68
41	1	0	0	Yes	260,0 mm	755,0 mm					Планка 66
42	1	0	0	Yes	200,0 mm	690,0 mm					Планка 65
43	1	0	0	Yes	350,0 mm	600,0 mm					Планка 64
48	1	0	0	Yes	196,0 mm	373,0 mm					Планка 59
55	1	0	0	Yes	230,0 mm	2000,0 mm					Планка 52
57	1	0	0	Yes	180,0 mm	826,0 mm					Планка 50
58	1	0	0	Yes	351,0 mm	709,0 mm					Планка 49
65	1	0	0	Yes	392,0 mm	428,0 mm					Планка 42
68	1	0	0	Yes	295,0 mm	362,0 mm					Планка 39
69	1	0	0	Yes	293,0 mm	388,0 mm					Планка 38
73	1	0	0	Yes	234,0 mm	388,0 mm					Планка 34
75	1	0	0	Yes	217,0 mm	299,0 mm					Планка 32
81	1	0	0	Yes	96,0 mm	461,0 mm					Планка 26
82	1	0	0	Yes	96,0 mm	259,0 mm					Планка 25
83	1	0	0	Yes	7,0 mm	202,0 mm					Планка 24
87	1	0	0	Yes	42,0 mm	110,0 mm					Планка 20

Фигура 4.61: Лист 5 от разпечатката на комерсиалния продукт.

2. Реална площ на всички 106 планки - 9 859 421 ^{2};

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

Optimization Printout v. Date/Time: 28.11.2018 г. 09:50:53 - Page 6

Opti2D (s.n. 4375)

LIST OF PIECES TO CUT

Pos.	Work	Rem.	±	Rot.	X Dim.	Y Dim.	Shape	Customer	Order	Rack	Id.
90	1	0	0	Yes	170,0 mm	869,0 mm					Планка 17
91	1	0	0	Yes	309,0 mm	869,0 mm					Планка 16
92	1	0	0	Yes	845,0 mm	2070,0 mm					Планка 15
94	1	0	0	Yes	260,0 mm	755,0 mm					Планка 13
95	1	0	0	Yes	200,0 mm	690,0 mm					Планка 12
96	1	0	0	Yes	350,0 mm	600,0 mm					Планка 11
102	1	0	0	Yes	196,0 mm	373,0 mm					Планка 5

LIST OF PIECES TO CUT

Pos.	Work	Rem.	±	Rot.	X Dim.	Y Dim.	Shape	Customer	Order	Rack	Id.
1	1	0	0	Yes	180,0 mm	2273,0 mm					Планка 106
3	1	0	0	Yes	180,0 mm	1613,0 mm					Планка 104
6	1	0	0	Yes	180,0 mm	680,0 mm					Планка 101
7	1	0	0	Yes	140,0 mm	680,0 mm					Планка 100
8	1	0	0	Yes	471,0 mm	535,0 mm					Планка 99
9	1	0	0	Yes	61,0 mm	146,0 mm					Планка 98
10	1	0	0	Yes	85,0 mm	469,0 mm					Планка 97
11	1	0	0	Yes	120,0 mm	450,0 mm					Планка 96
13	1	0	0	Yes	299,0 mm	300,0 mm					Планка 94
17	1	0	0	Yes	134,0 mm	285,0 mm					Планка 90
18	1	0	0	Yes	148,0 mm	280,0 mm					Планка 89
19	1	0	0	Yes	234,0 mm	423,0 mm					Планка 88
21	1	0	0	Yes	203,0 mm	234,0 mm					Планка 86
23	1	0	0	Yes	134,0 mm	170,0 mm					Планка 84
24	1	0	0	Yes	110,0 mm	170,0 mm					Планка 83
25	1	0	0	Yes	150,0 mm	170,0 mm					Планка 82
26	1	0	0	Yes	112,0 mm	120,0 mm					Планка 81
27	1	0	0	Yes	55,0 mm	120,0 mm					Планка 80
28	1	0	0	Yes	96,0 mm	461,0 mm					Планка 79
31	1	0	0	Yes	73,0 mm	110,0 mm					Планка 76
32	1	0	0	Yes	68,0 mm	110,0 mm					Планка 75
33	1	0	0	Yes	56,0 mm	150,0 mm					Планка 74
35	1	0	0	Yes	56,0 mm	134,0 mm					Планка 72
36	1	0	0	Yes	528,0 mm	1556,0 mm					Планка 71
40	1	0	0	Yes	178,0 mm	772,0 mm					Планка 67
44	1	0	0	Yes	85,0 mm	322,0 mm					Планка 63
45	1	0	0	Yes	250,0 mm	250,0 mm					Планка 62
46	1	0	0	Yes	85,0 mm	240,0 mm					Планка 61
47	1	0	0	Yes	100,0 mm	240,0 mm					Планка 60
49	1	0	0	Yes	196,0 mm	373,0 mm					Планка 58
50	1	0	0	Yes	130,0 mm	150,0 mm					Планка 57
51	1	0	0	Yes	86,0 mm	373,0 mm					Планка 56
52	1	0	0	Yes	81,0 mm	335,0 mm					Планка 55
53	1	0	0	Yes	67,0 mm	116,0 mm					Планка 54
54	1	0	0	Yes	180,0 mm	2273,0 mm					Планка 53
56	1	0	0	Yes	180,0 mm	1613,0 mm					Планка 51
59	1	0	0	Yes	180,0 mm	680,0 mm					Планка 48

Фигура 4.62: Лист 6 от разпечатката на комерсиалния продукт.

Резултати от представения метод в настоящия труд. Брой планки 106. Всички планки са с отместен контур на 5мм от действителния им контур. Виж фигура

Optimization Printout v. Date/Time: 28.11.2018 г. 09:50:54 - Page 7

Opti2D (s.n. 4375)

LIST OF PIECES TO CUT

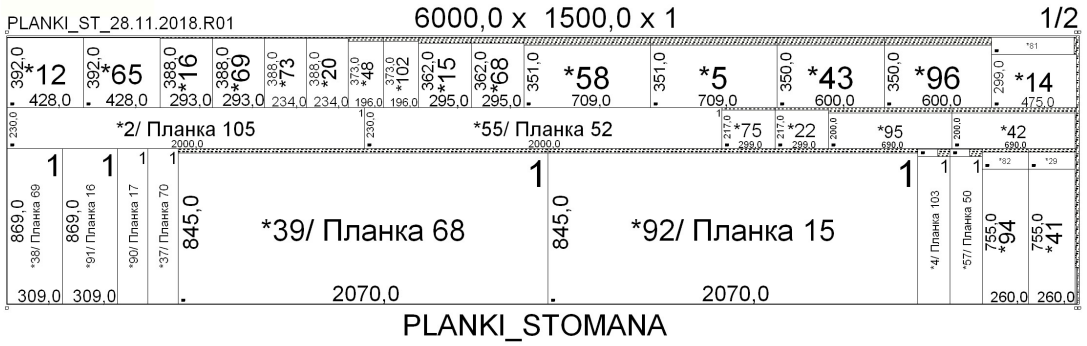
Pos.	Work	Rem.	±	Rot.	X Dim.	Y Dim.	Shape	Customer	Order	Rack	Id.
60	1	0	0	Yes	140,0 mm	680,0 mm					Планка 47
61	1	0	0	Yes	471,0 mm	535,0 mm					Планка 46
62	1	0	0	Yes	61,0 mm	146,0 mm					Планка 45
63	1	0	0	Yes	85,0 mm	469,0 mm					Планка 44
64	1	0	0	Yes	120,0 mm	450,0 mm					Планка 43
66	1	0	0	Yes	299,0 mm	300,0 mm					Планка 41
67	1	0	0	Yes	299,0 mm	475,0 mm					Планка 40
70	1	0	0	Yes	134,0 mm	285,0 mm					Планка 37
71	1	0	0	Yes	148,0 mm	280,0 mm					Планка 36
72	1	0	0	Yes	234,0 mm	423,0 mm					Планка 35
74	1	0	0	Yes	203,0 mm	234,0 mm					Планка 33
76	1	0	0	Yes	134,0 mm	170,0 mm					Планка 31
77	1	0	0	Yes	110,0 mm	170,0 mm					Планка 30
78	1	0	0	Yes	150,0 mm	170,0 mm					Планка 29
79	1	0	0	Yes	112,0 mm	120,0 mm					Планка 28
80	1	0	0	Yes	55,0 mm	120,0 mm					Планка 27
84	1	0	0	Yes	73,0 mm	110,0 mm					Планка 23
85	1	0	0	Yes	68,0 mm	110,0 mm					Планка 22
86	1	0	0	Yes	56,0 mm	150,0 mm					Планка 21
88	1	0	0	Yes	56,0 mm	134,0 mm					Планка 19
89	1	0	0	Yes	528,0 mm	1556,0 mm					Планка 18
93	1	0	0	Yes	178,0 mm	772,0 mm					Планка 14
97	1	0	0	Yes	85,0 mm	322,0 mm					Планка 10
98	1	0	0	Yes	250,0 mm	250,0 mm					Планка 9
99	1	0	0	Yes	85,0 mm	240,0 mm					Планка 8
100	1	0	0	Yes	100,0 mm	240,0 mm					Планка 7
101	1	0	0	Yes	196,0 mm	373,0 mm					Планка 6
103	1	0	0	Yes	130,0 mm	150,0 mm					Планка 4
104	1	0	0	Yes	86,0 mm	373,0 mm					Планка 3
105	1	0	0	Yes	81,0 mm	335,0 mm					Планка 2
106	1	0	0	Yes	67,0 mm	116,0 mm					Планка 1

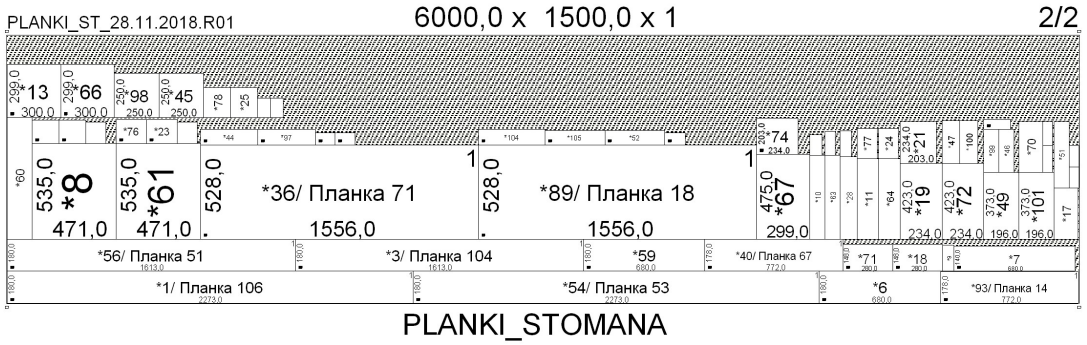
Фигура 4.63: Лист 7 от разпечатката на комерсиалния продукт.

4.39. Разрешено завъртане на планките : Да. Контур : $offsetPtL$.

Opti2D (s.n. 4375)

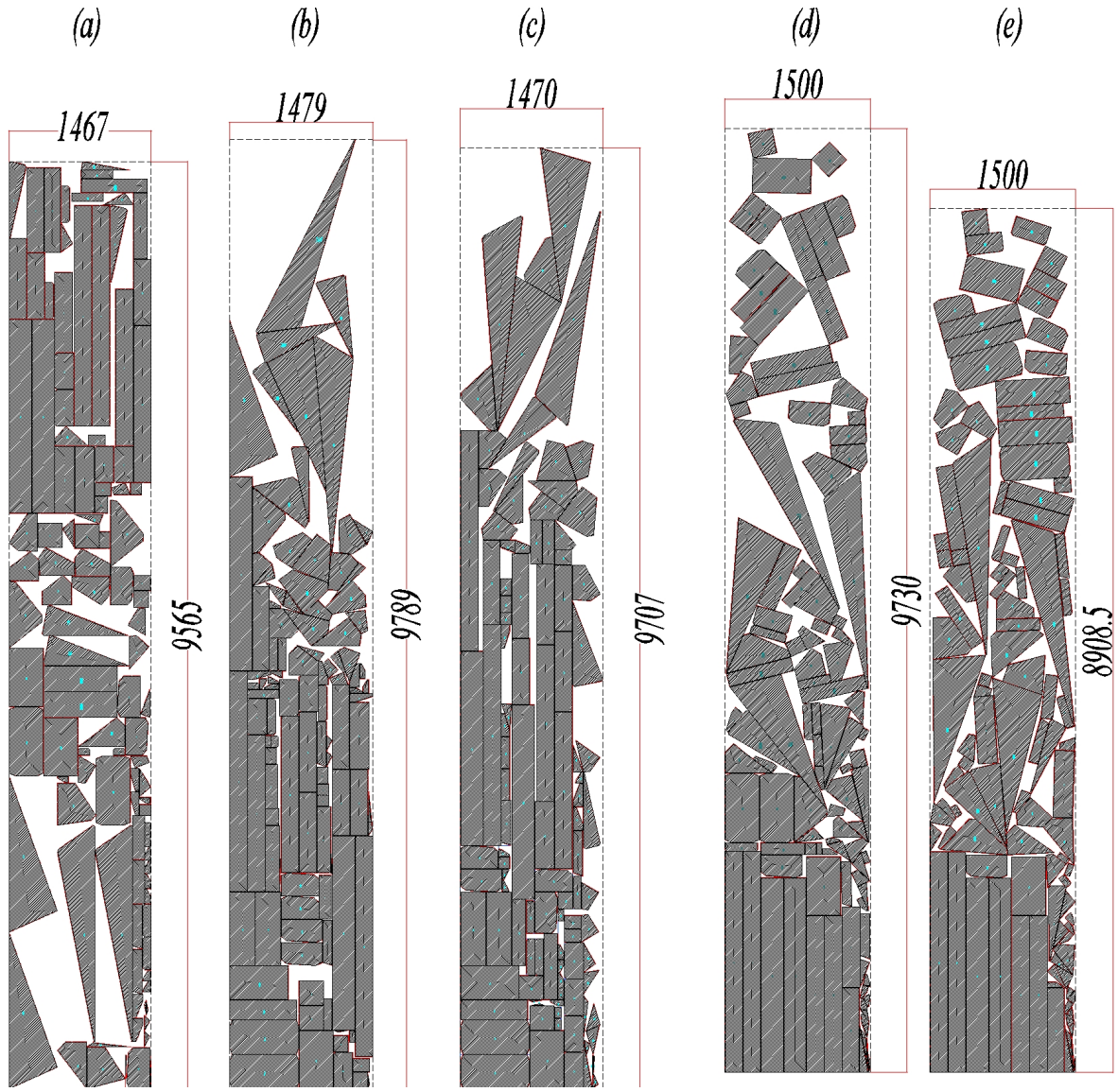
WORKORDER				MATERIAL	
Work Order	Planki St 28.11.2018.R01			Material	PLANKI_STOMANA
Description	Planki St 28.11.2018			Full Description	
Date	28.11.2018 г.			Thickness	0,0 mm
U.M.	mm	Kg	mq	Weight	0.00 Kg
Status	Optimized				





Фигура 4.64: Лист 8 от разпечатката на комерсиалния продукт.

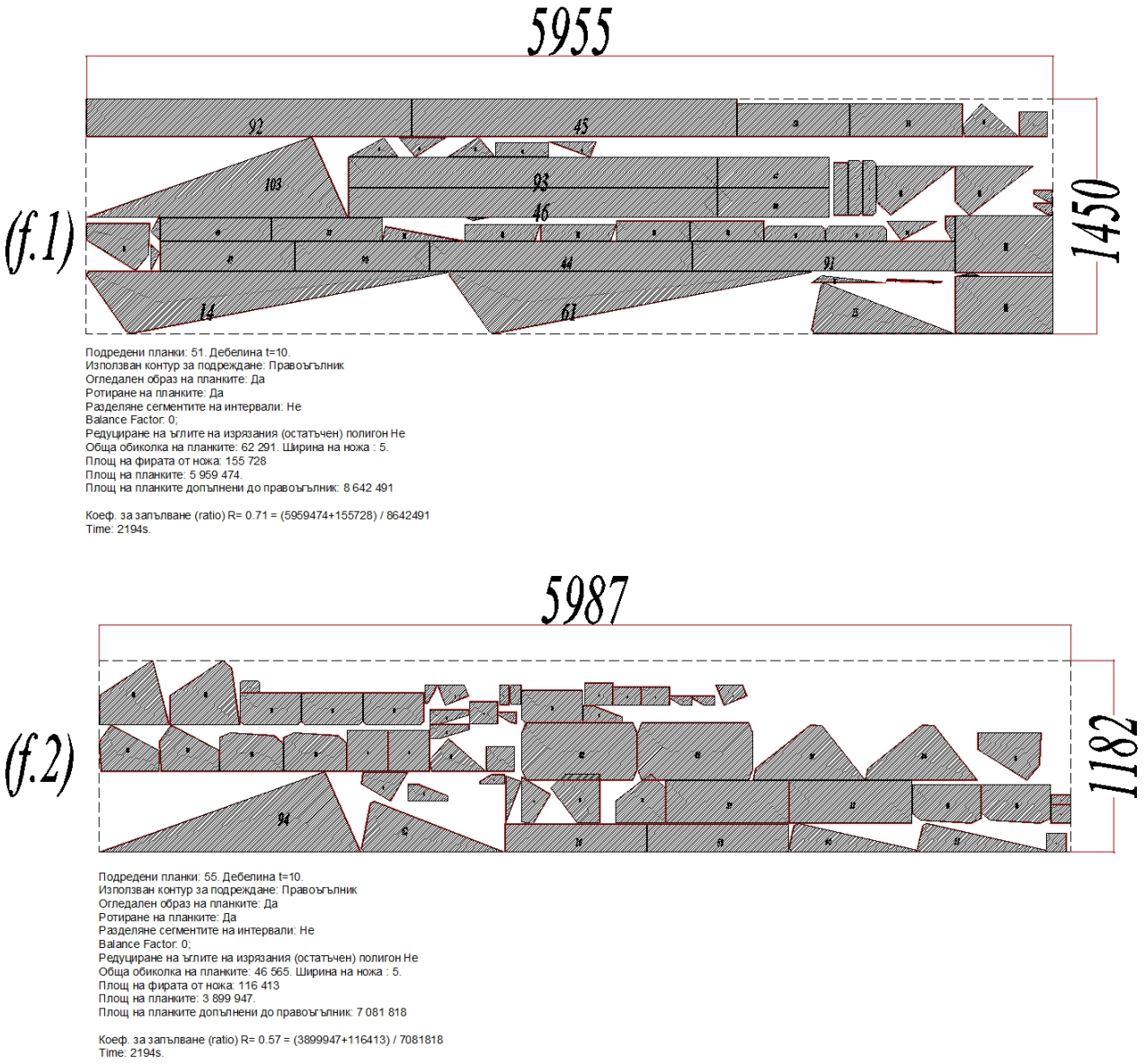
Информация към разкрой 1.



Фигура 4.65: Разкрой 1 с представения алгоритъм.

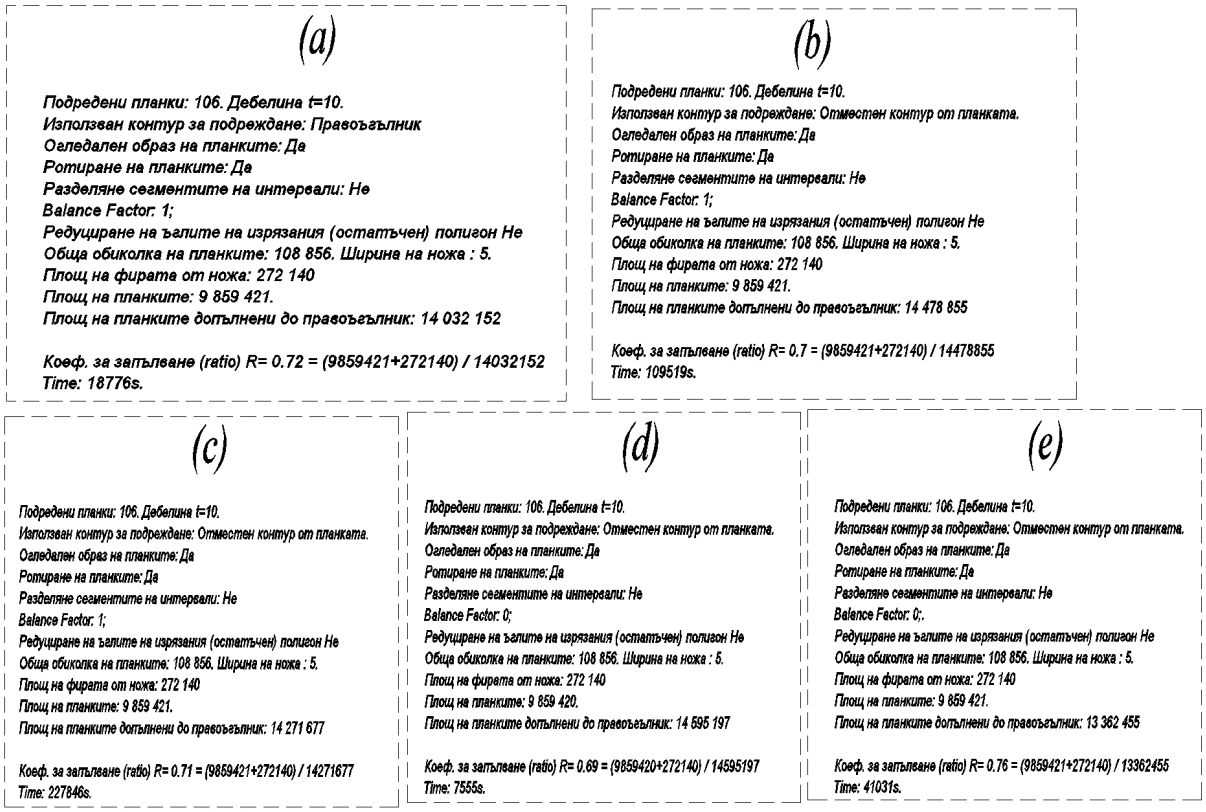
Таблица 4.6: Сравнение на комерсиален продукт и представения алгоритъм

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



Фигура 4.66: Разкрой 2 с хоризонтално поставени полигони(листа) за запълване.

За случай $f.1$ и $f.2$ в скоби са дадени запълванията на планките спрямо основния лист 1500мм х 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. Ъглите на завъртане са сведени до два: 0° и 90 °. По отношение на коефициента на запълване *Ratio*, представения алгоритъм е много по-добър. виж таблица 4.6. 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 уплътне-



Фигура 4.67: Информация към разкрой 1 с представения алгоритъм.

ние на фигурите. Друго негово предимство е много добрата му пригодност към паралелизация на изчисленията.

Глава 5

Заключение

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 проявя 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. Тестовете 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. **"Ant Colony Optimization algorithm for 1D Cutting Stock Problem"**, pp. 25-31, *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_3

2. Evtimov G. Fidanova S. **"2D Optimal Cutting Problem"**, pp. 33-39, *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

3. 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

4. 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

5. 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

6. 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

7. 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

8. 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 Апробация 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 могатда 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 използуват 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 използуват 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 кликвания 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 използува. Алгоритъмът, разработен в представеният дисертационен труд, позволява използуване на огледален образ на полигоните, ротация и други операции, които могат да доведат до съществено подобрение на полученото приближено решение. Разбира се, за целите на широкомащабни 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

Изказвам 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 разкрой.

Изказвам 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 Groupc 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 theone-dimensional cutting stock problem"**, European Journal of Operational Research93:522–535,(1996).

[3] Foerster H., Wscher G. **"Pattern reduction in one-dimensional cutting stock problems"**, In: Proceedings of the 15th Trienal Conference of the International Federationof 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 procedureto a real-life 1.5-dimensional cutting stock problem"**. European Journal of OperationalResearch 175:1870–1889, (2006).

[6] Suliman S.M.A. **"Pattern generating procedure for the cutting stock problem"**, IntJ 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 theone-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

[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

[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

[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 Eddditional 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\cyrm\T2A\cyri

[35] Glover F., **"Future paths for integer programming and links to artficial 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, (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, (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

[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, (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, (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, (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"**, InX. 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), Spinger, 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.

[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

[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

[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., (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. Compuy. 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 Freemann, 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 andways 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 toone-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 OperationalResearch. 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 Journalof 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).