НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2015 р.

Дипломна робота
на здобуття ступеня бакалавра

зі спеціальності 6.050101 Комп'ютерні науки
(код та назва спеціальності)

на тему: Аналіз алгоритмів автоматичного розміщення блоків інтегральних схем

Виконала: студентка IV курсу, групи ДА-11
(шифр групи)

Ревуцька Ірина Володимирівна
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н. Кирюша Б.А.
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант охорона праці доцент, канд. біол. наук Гусєв А. М.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст. викладач Британов О.А.
(підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2015

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 6.050101 Комп'ютерні науки
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2015 р.

ЗАВДАННЯ
на дипломну роботу студенту
Ревуцькій Ірині Володимирівні
(прізвище, ім'я, по батькові)

1. Тема роботи Аналіз алгоритмів автоматичного розміщення блоків інтегральних схем,
керівник роботи Кирюша Богдан Анатолійович, к.т.н.,доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» квітня 2015 р. №30/1-ст

2. Строк подання студентом роботи 08.06.2015

3. Вихідні дані до роботи:
- Операційна система Windows 8;
- Мова програмування – C#;

4. Зміст роботи:
1. Провести аналіз існуючих алгоритмів розміщення та обрати алгоритм для реалізації.
2. Дослідити особливості обраного алгоритму та реалізувати його на практиці.
3. Реалізувати алгоритм трасування та провести тестування реалізації алгоритму розміщання на предмет трасування.
4. Розглянути питання з охорони праці та безпеки у надзвичайних ситуаціях при використанні результатів дипломного проекту

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

1. Порівняльний аналіз алгоритмів розміщення– плакат.
2. Блок-схема алгоритму розміщення– плакат.
3. Приклад роботи алгоритму розміщення– плакат.
4. Приклад результату трасування– плакат.
5. Презентація у форматі Power Point

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	к.б.н., доц. Гусєв А.М.		

7. Дата видачі завдання 01.02.2015

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2015	
2	Збір інформації	15.02.2015	
3	Вивчення варіантів реалізації та вибір варіанту для розробки	28.02.2015	
4	Модифікація бази даних документообігу	10.03.2015	
5	Розробка серверної частини	15.03.2015	
6	Розробка модуля генерації	18.04.2015	
7	Розробка інтерфейсу користувача	27.04.2015	
8	Оформлення дипломної роботи	31.05.2015	
9	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2015	

Студент _____ І.В.Ревуцька
(підпис) (ініціали, прізвище)

Керівник роботи _____ Б.А.Кирюша
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

бакалаврської дипломної роботи Ревуцької Ірини Володимирівни
на тему: “Аналіз алгоритмів автоматичного розміщення блоків інтегральних схем ”

Метою дипломної роботи є розробка фрагменту комплексу програм для автоматизованої генерації топологій замовних інтегральних схем(ІС), який відповідає за автоматичне розміщення фрагментів ІС на кристалі та зв’язків між ними.

У роботі проведено аналіз існуючих алгоритмів розміщення за результатами останніх досліджень. На основі аналізу обрано алгоритм для реалізації.

Розроблено частину генератора топологій, що відповідає за автоматичне розміщення фрагментів ІС на кристалі та зв’язків між ними. Проведено тестування розробленого функціоналу.

Робота складається з 85 стор., з яких основна частина –65 стор., 25 рисунків, 17 таблиць, 11 посилань, 2 додатки на 1 стор. та 23 стор. відповідно.
Ключові слова: ISPD, HPWL, DPlace, хвильовий алгоритм.

ANNOTATION

to the bachelor thesis of Revutska Iryna Volodymyrivna

on “Analysis of automatically placement algorithms of integrated circuits blocks”

The aim of this thesis is to develop a part of software system designed for automated generation of topologies of customized integrated circuits (ICs). This part automatically places fragments of ICs on the chip and on connections between them.

This thesis analyzes the existing placement algorithms according to the latest researches. Based on the performed analysis an algorithm has been selected for further implementation.

A part of the topologies generator, which places fragments of ICs on the chip and on connections between them. responsible for automatic placement of IP fragments on the chip and connections between them. The developed functionality has been tested.

The work consists of 85 p., main part consists of -61 p., 25 illustrations, 17 tables, 11 references, 2 applications on 1 p. and 23 p., accordingly. Keywords: ISPD, HPWL, DPlace, wave algorithm.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ - Програмне забезпечення

САПР - Система автоматизованого проектування

КП Комутаційна плата

ІС Інтегральна схема

ISPD International Symposium on Physical Design

HPWL Half-perimeter wire length

ВСТУП

На сьогоднішній день проектування інтегральних схем невід'ємно пов'язане з технологіями Систем Автоматизованого Проектування (САПР). САПР дозволяють підвищити ефективність праці інженерів за рахунок скорочення трудомісткість проектування та планування, скорочення термінів проектування, скорочення собівартості проектування.

Сучасний ринок програмного забезпечення пропонує широкий вибір відкритих та закритих систем автоматизованого генерування топологій ІС. Проте існуючі відкриті програмні продукти погано сумісні з програмним забезпеченням, що доступне/використовується в КПІ. Щодо комерційних засобів, таких як Cadence, вони опираються на комерційні технології, котрі не призначені для відкритого використання в учбовому процесі.

Наявність власного ядра розміщення та прокладання з’єднань інтегральних схем дозволяє автоматизувати синтез і масштабування топологій у відповідності до вимог, рівня підготовки та умов роботи потенційного споживача – студента чи наукового співробітника КПІ.

Саме цим власна розробка і сьогодні, і в перспективі краще залежності від пакетів, що розробляються під чужі вимоги, стандарти, без урахування сумісності з необхідними нам платформами та програмним забезпеченням.

Метою даної роботи є розробка частини генератора топологій, що відповідає за розміщення елементів інтегральної схеми та трасування з’єднань. Для досягнення цієї мети необхідно було провести аналіз існуючих алгоритмів розміщення та трасування, обрати найбільш оптимальні алгоритми та реалізувати їх на практиці.

1 АНАЛІЗ АЛГОРИТМІВ РОЗМІЩЕННЯ

1.1 Постановка задачі

Однією з найважливіших задач, яку виконують САПР при конструюванні електронних схем, є задача оптимального розміщення елементів. Критерії оптимальності можуть бути різними: мінімум сумарної довжини з'єднань; мінімум кількості з'єднань, довжина яких перевищує задану; мінімум перетинів провідників; максимальна кількість з'єднань між сусідніми елементами. У будь-якому разі, основна ціль розміщення – створення найкращих умов для подальшого трасування з'єднань. Досягти цього можна різними методами, виходячи з метричних та топологічних параметрів схеми.

В загальному вигляді задача розміщення конструктивних елементів на комутаційній платі формулюється наступним чином. Існує множина конструктивних елементів $R=\{r_1,r_2,....,r_n\}$ і множина зв'язків між ними $V=\{v_1,v_2,...,v_p\}$, а також множина установочних позицій на комутаційній платі $T=\{t_1, t_2,...,t_k\}$. Необхідно знайти таке відображення множини R на множину T, яке забезпечить екстремум цільової функції. В більшості випадків мінімізується загальна довжина з'єднань між елементами :

$$L = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} c_{ij} \qquad (1)$$

де d_{ij} - відстань між позиціями встановлення елементів, c_{ij} - число зв'язків між елементами x_i та x_j.[1]

При розміщенні, елементи представляються точками для спрощення обчислень, а найбільш зв'язані елементи об'єднуються в групи, що в свою чергу виступають як один елемент.

Всі алгоритми,які використовує САПР, є наближеними, адже при

розрахунках враховується лише величина d_{ij} - відстань між центрами елементів, а не справжня довжина з'єднань, яка буде визначена лише на наступному етапі – етапі трасування з'єднань.

1.2 Класифікація алгоритмів

Тема вибору алгоритму є особливо актуальною в наш час, оскільки у промисловому середовищі мова йде про мільйони вузлів на ІС та ще більшу кількість входів і виходів на них, а затрати на виробництво не дозволяють зробити помилку на етапі проектування . Розміщення елементів є ключовим етапом в конструюванні інтегральних схем (ІС), який суттєво впливає на загальну продуктивність САПР. І хоча ця тема є актуальною протягом вже більш як тридцяти років, стрімкий розвиток технологій спонукає розробників до пошуку нових рішень, пов'язаних з неймовірним ростом як кількості елементів, так і їх складності.

Пошук рішень ведеться в різних напрямках, але найчастіше існуючі алгоритми розміщення поділяють на три основні групи (рисунок 1.2).

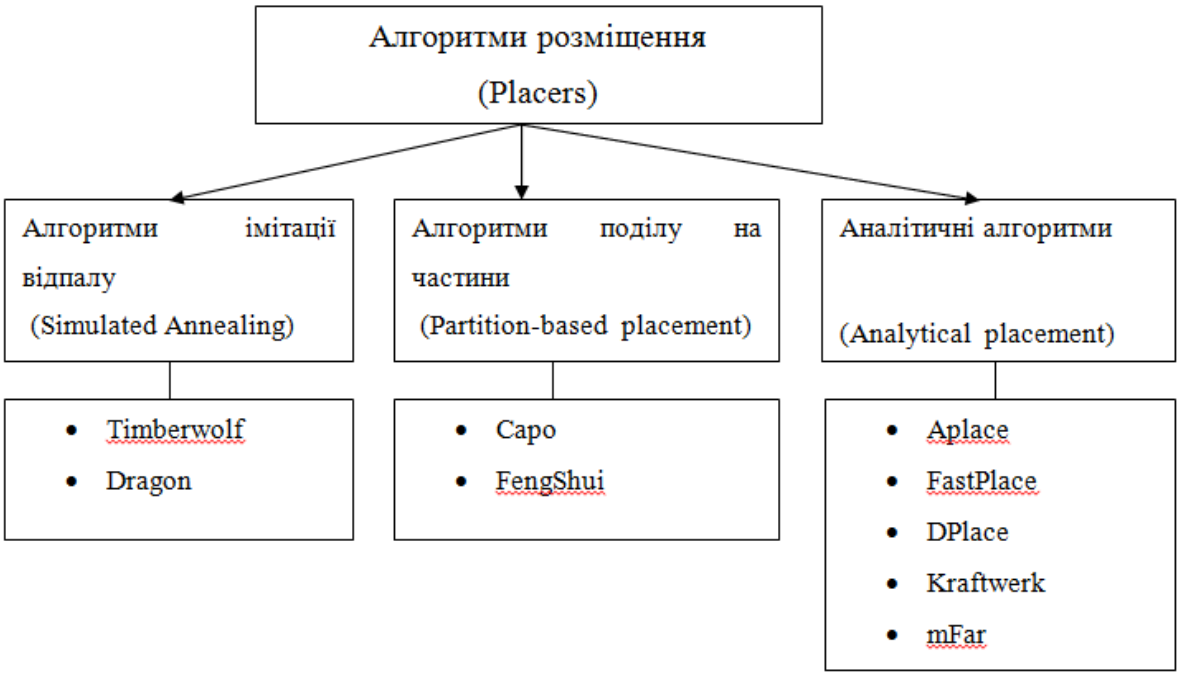


Рисунок 1.2 – Класифікація алгоритмів розміщення

Алгоритми імітації відпалу – це історично найперший метод, що був застосований для розміщення елементів ІС. Ідея надзвичайно проста: береться початкове, іноді,навіть, випадкове розміщення, яке надалі вдосконалюється, шляхом різноманітних переміщень, що приймаються або відхиляються згідно встановленого критерію. Метод достатньо ефективний, але потребує значного часу на виконання.

Алгоритми поділу на частини – головна ідея полягає в поступовому зниженні розмірності задачі. На кожній ітерації множина елементів ділиться на підмножини, а область розміщення – на ділянки. Елементи закріпляються за ділянками. Процес продовжується, поки кількість елементів в підмножині не досягне заданого числа (наприклад, для Capo - 8). Далі розміщення ведеться в середині ділянки.

Аналітичні алгоритми – задача розміщення формулюється як математична задача з квадратичною функцією. Спочатку проводиться початкове розміщення, що характеризується високим рівнем перекриття (накладання) елементів. Для подальшого розподілу елементів найчастіше застосовується поняття «сили» (force-directed placement), рідше - дифузії (наприклад, в алгоритмі DPlace).

1.3 Порівняльний аналіз алгоритмів

Аналіз існуючих алгоритмів розміщення проводився на основі опублікованих в 2007 році результатів (banchmarks) конкурсних досліджень, проведених International Symposium on Physical Design (ISPD) протягом 2005 і 2006 років. [2]

Ці дослідження відіграють важливу роль в міжнародній практиці конструювання ІС, надаючи науковому співтовариству кількісні показники для порівняння оптимальності алгоритмів та визначення best practice.

Конкурсними випробуваннями передбачено тестування за допомогою

стандартного набору ІС, що суттєво відрізняються за своїми характеристиками. В таблиці 1.1 надається опис восьми із шістнадцяти схем, що використовувались в випробуваннях 2005 року.

Таблиця 1.1 - Характеристики схем в випробуваннях 2005 року [2]

Circuit	#Obj	#Mov	#Fixed	#Net	Density (%)	Util.(%)
adaptec1	211447	210904	543	221142	76	57
adaptec2	255023	254457	566	266009	79	44
adaptec3	451650	450927	723	466758	75	34
adaptec4	496045	494716	1329	515951	63	27
bigblue1	278164	277604	560	284479	54	45
bigblue2	557866	534782	23084	577235	62	38
bigblue3	1096812	1095519	1293	1123170	86	57
bigblue4	2177353	2169183	8170	2229886	65	44

Поряд з загальною кількістю елементів (Obj) та з’єднань (Net), кількістю зафіксованих (Fixed) і рухомих елементів (Mov), введено такі важливі характеристики, як Density(%) – співвідношення сумарної площі всіх елементів до площі розміщення, Util(%) – співвідношення сумарної площі рухомих елементів до вільної для переміщення площі (різниця між загальною площею та площею фіксованих елементів(перешкоди)). Таблиця 1.2 містить результати досліджень 2005 року.

Таблиця 1.2 - Результати тестування алгоритмів розміщення 2005 р. HPWL[2]

Placer	adaptec2	adaptec4	bigblue1	bigblue2	bigblue3	bigblue4	Ratio
Aplace	87.31	187.65	94.64	143.82	357.89	833.21	1.00
mFar	91.53	190.84	97.70	168.70	379.95	876.28	1.06
Dragon	94.72	200.88	102.39	159.71	380.45	903.96	1.08
mPL	97.11	200.94	98.31	173.22	369.66	904.19	1.09
FastPlace	107.86	204.48	101.56	169.89	458.49	889.87	1.16
Capo	99.71	211.25	108.21	172.30	382.63	1098.76	1.17
NTUplace	100.31	206.45	106.54	190.66	411.81	1154.15	1.21
FengShui	122.99	337.22	114.57	285.43	471.15	1040.05	1.50
Kraftwerk	157.65	352.01	149.44	322.22	656.19	1403.79	1.84

Критерієм оптимальності в випробуваннях 2005 р. було обрано півпериметр довжини з'єднань. Показник Ratio є співвідношенням довжини загального півпериметру з'єднань схеми до мінімального значення цього показника, виявленого під час тестування. В даному випадку найкращим є Ratio для алгоритму APlace.

Окремо слід зазначити, що до розгляду допускаються лише алгоритми, які підлягають трасуванню. Оскільки цей показник важко перевірити під час розміщення, в випробуваннях 2006 року до існуючих характеристик схем додали ще одну – density target – обмежуючу величину, яка має забезпечити майбутнє трасування та наблизити процес до реальних умов виробництва. Density target лежить в межах 50% - 90%. Чим вищий цей показник, тим важче проводити розміщення.[2]

В таблиці 1.3 наведені характеристики схем, що були задіяні в випробуваннях 2006 року.

Комплексна оцінка оптимальності алгоритму в 2006 році розраховувалась за формулою (2):

$$HPWL \cdot (1+scaled\_overflow\_factor+cpu\_factor) \quad (2)$$

Таблиця 1.3 - Характеристики схем в випробуваннях 2006 року[2]

Circuit	#Obj	#Mov	#Fixed	#Net	Density (%)	Util. (%)	Density target
adaptec5	843128	842482	646	867798	79	50	0.5
newblue1	330474	330137	337	228901	86	83	0.8
newblue2	441516	330239	1277	465219	86	62	0.9
newblue3	494011	482833	11178	552199	85	26	0.8
newblue4	646139	642717	3422	637051	66	46	0.5
newblue5	1233058	1228177	4881	1284251	75	50	0.5
newblue6	1255039	1248150	6889	1288443	59	39	0.8
newblue7	2507954	2481372	26582	2636820	76	49	0.8

Для розрахунку Scaled_overflow_factor область розміщення поділяють на контейнери однакового розміру і для кожного з них знаходять bin_overflow за формулою (3):

$$bin\_overflow(b) = \sum_{movable v \in b} [area(v) - free\_space(b) \times density\_target] \quad (3)$$

Величина *free_spase(b)* дорівнює різниці між загальною площею контейнера та площею фіксованих об'єктів в ньому. Суму всіх *bin_overflow* позначають як *total_overflow*. *Scaled_overflow_factor* розраховується за формулою (4):

$$scaled\_overflow\_factor = \frac{total\_overflow \times single\_bin\_area \times density\_target}{(\sum_{movable v \in design} area(v)) \times c} \quad (4)$$

Характеристику cpu_factor було введено з метою спонукати розробників до винаходу більш швидких рішень, але з дотриманням показників якості. Тому cpu_factor обчислюється як

$$cpu\_factor = 0.04 \times \ln \frac{placer\_cputime}{median\_cputime} \quad (5)$$

Тобто, якщо алгоритм повільніший за середнє значення показника, йому начисляються штрафні бали, і, навпаки, - швидший алгоритм отримує бонус, щоправда, він обмежується 10 %.

В таблиці 1.4 наведено результати оцінювання алгоритмів під час випробувань 2006 року. Півпериметр довжини з'єднань надано з врахуванням *Scaled_overflow_factor*.

Таблиця 1.4 - Результати тестування алгоритмів розміщення 2006 р. SHPWL(scaled) [2]

Placer	ad5	nb1	nb2	nb3	nb4	nb5	nb6	nb7
Kraftwerk	457.92	78.60	208.41	280.93	315.53	569.36	545.94	1170.85
mPL6	431.14	67.02	200.93	287.05	299.66	540.67	518.70	1082.92
NTUplace3	432.58	63.49	203.68	291.15	305.79	517.63	532.79	1181.30
mFAR	476.28	77.54	212.90	303.91	324.40	601.27	535.96	1153.76
Aplace3	520.97	73.31	198.24	273.64	384.12	613.86	522.73	1098.88
Dragon	500.74	80.77	260.83	524.58	341.16	614.23	572.53	1410.54
FastPlace	805.64	84.55	212.30	362.99	429.79	962.06	574.18	1236.34
DPlace	572.98	102.75	329.92	380.14	364.45	752.08	682.87	1438.99
Capo	494.64	98.48	309.53	361.25	362.40	659.57	668.66	1518.75

Таблиця 1.4 кардинально відрізняється від Таблиці 1.2 з результатами випробувань 2005 року.

Алгоритми аналітичного розміщення (APlace, mFar, MPl, FastPlace), які демонструють вражаючі досягнення щодо довжини з'єднань, у даному випадку посіли останні місця. Навпаки, алгоритм Kraftwerk (також з групи аналітичних алгоритмів), який програє в сумарній довжині з'єднань завдяки розподілу рухомих елементів по всій площі області розміщення, з врахуванням масштабованості вийшов на перше місце.

Підсумки випробувань 2006 року наведено в таблиці 1.5.

Таблиця 1.5 Результати тестування алгоритмів розміщення 2006 р. Підсумок[2]

Placer	Avg. HPWL Ratio	Avg. OV Factor (%)	Avg. CPU Factor (%)	Score Ratio
Kraftwerk	1.09	1.68	−5.04	1.03
mPL6	1.03	1.36	1.58	1.04
NTUplace3	1.02	4.10	1.66	1.05
mFAR	1.11	2.71	−0.12	1.11
Aplace3	1.10	3.82	5.31	1.16
Dragon	1.33	0.12	−5.90	1.24
FastPlace	1.18	22.09	−5.62	1.33
DPlace	1.34	9.32	−4.54	1.36
Capo	1.38	0.32	2.69	1.39

Від’ємне значення показника Avg. CPU Factor (%) свідчить про більшу швидкість алгоритму порівняно з середнім значенням і понижує Score Ratio на свою величину. [2]

З наведених в цьому розділі даних слідує, що застосування комплексних показників оптимальності, надає об’єктивну але досить узагальнену оцінку. Таким чином, алгоритми, які мають суттєві переваги, скажімо, в досягненні мінімальної довжини з’єднань або високої щільності розміщення, але демонструють не досить вражаючу продуктивність, в загальному рейтингу можуть бути незаслужено відтиснуті назад. При виборі алгоритму слід керуватися не лише комплексною оцінкою, але й такими факторами, як масштабованість, продуктивність, простота реалізації тощо.

1.4 Обґрунтування вибору алгоритму DPlace

Наведений в попередньому розділі порівняльний аналіз існуючих алгоритмів доводить, що багато з них демонструють високі показники виконання. Але, враховуючи виклики, пов’язані з вражаючим зростанням кількості елементів схем та їх складності, що останнім часом постали перед розробниками, вирішальна роль відводиться таким характеристикам, як масштабованість та продуктивність системи. З попереднього аналізу видно, що алгоритми аналітичного розміщення є досить успішними в цьому плані. Крім того вони мають значні переваги в оптимізації рішень суто математичними методами.

Однак, алгоритм двовимірного розміщення Kraftwerk (переможець ISPD 2006 Placement Contest), що належить до групи алгоритмів, які використовують поняття «сили» для переміщення елементів на комутаційній платі, є складнішим в реалізації і потребує великої кількості додаткових (контролюючих) макросів[2]. На відміну від нього, інший двовимірний алгоритм аналітичного розміщення DPlace має більш прозору реалізацію, хоча

й показав гірші показники. Це пояснюється тим, що DPlace порівняно новий алгоритм (в тестуванні 2005 року не брав участі). Його можливості активно вивчаються, а показники вдосконалюються.

Нижче наведені результати досліджень, проведених в Техаському університеті (Tao Luo and David Z. Pan, The University of Texas at Austin) з використанням комп'ютеру на базі ОС Linux server та CPU Xeon 64-bit, 3.4 GHz. В тестах використовувались ті ж набори IC, що і в випробуваннях ISPD 2005-2006 р. р. Розробникам вдалося значно покращити результати, наведені в розділі 1.2. Вони представили на розгляд свою матрицю Гессе (Таблиця 1.6), що має значно меншу розмірність та кількість ненульових елементів, і дає більш ніж 24-кратний виграш в часі виконання.

Таблиця 1.6 - Порівняння матриць Гессе традиційного та вдосконаленого алгоритму[2]

	Matrix A				Matrix **A′**				Solver
	Size	Non-0s	Precon(s)	Solve (s)	Size	Non-0s	Precon(s)	Solve(s)	speed-up
adaptec1	243K	196K	15.85	4.65	211K	430K	0.53	0.19	24.5×
adaptec2	355K	2,099K	25.61	7.38	254K	557K	0.90	0.30	24.6×
adaptec3	674K	3,713K	38.18	15.61	494K	1,131K	1.74	0.58	26.9×
adaptec4	508K	3,676K	38.42	15.51	451K	997K	1.97	0.49	31.7×
bigblue1	392K	2,287K	29.78	6.87	278K	603K	1.16	0.36	19.1×
bigblue2	729K	3,937K	47.79	22.78	535K	1,178K	2.29	0.82	27.8×
bigblue3	1,389K	7,290K	103.93	39.32	1,096K	2,714K	4.54	1.70	23.1×
bigblue4	2,831K	16,850K	221.47	75.70	2,169K	5,190K	10.66	3.91	19.4×
									24.6×

До показників тестування 2006 розробники додали DHPWL, що розраховується як

$$HPWL(1+Density\_Target\_penalty\_factor) \quad (6)$$

Density_Target_penalty_factor – та сама величина, що й у випробуваннях

2006 року. Результати наведені в Таблиці 1.7.

Таблиця 1.7- HPWL та час виконання для вдосконаленого алгоритму DPlace.[2]

	HPWL ($\times 10^6$)	DHPWL ($\times 10^6$)	GP (s)	DP (s)	Total (s)
adaptec5	433.06	497.56	3276	1474	4750
newblue1	89.18	89.46	1227	578	1805
newblue2	215.12	217.19	1724	768	2492
newblue3	322.39	324.55	1929	1168	3097
newblue4	266.52	324.56	2141	1361	3502
newblue5	578.52	725.12	5233	1852	7085
newblue6	579.86	599.44	4712	2863	7575
newblue7	1089.15	1215.32	13625	4475	18100

На рисунку 1 зображено порівняльну діаграму ненульових елементів в колонках матриці A та A`.

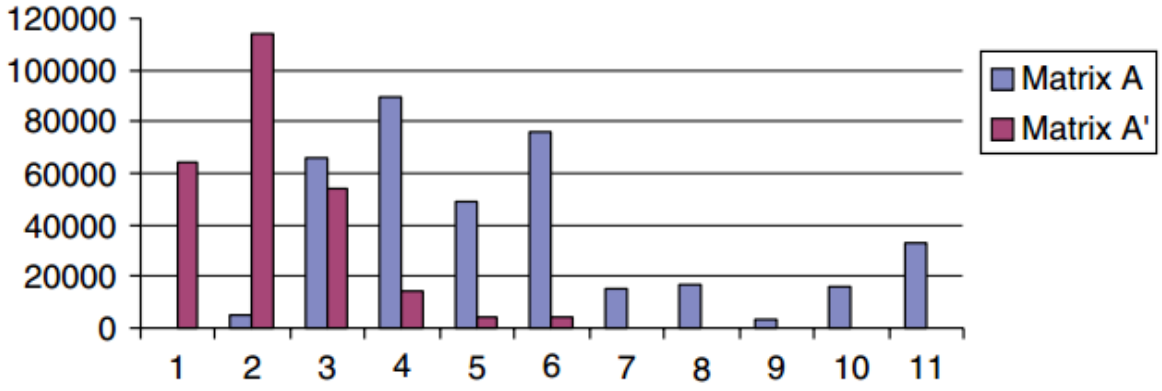


Рисунок 1.4 – Порівняльна діаграма ненульових елементів в матрицях A та A` ; по осі x відкладено номер колонки, по осі y - кількість ненульових елементів.[2]

Розмірність матриці A до перетворення складала 354 тис., тоді як розмірність матриці A` зменшилася до 254 тис.

Враховуючи всі вищезгадані особливості та переваги алгоритму, можна зробити висновок, що він якнайкраще підходить для вирішення поставленої задачі та здатен задовольнити всі поставлені вимоги.

1.5 Передумови виникнення алгоритму DPlace

Алгоритми аналітичного розміщення давно стали об’єктом пильної уваги розробників, оскільки постійно демонструють досить високі показники. Але більшість із них використовують поняття «сили», що призводить до певних ускладнень. «Сила» застосовується до кожного елементу для переміщення і зменшення перекриття, яке виникло при первинному розміщенні елементів. Величина та напрямок «сили» розраховуються на кожній ітерації. У випадку, коли початкові налаштування задані неточно (а це досить трудомісткий процес) спостерігається вихід елементу за межі області розміщення. Для усунення цієї проблеми створюють систему спеціальних контролюючих макросів, що значно ускладнює систему.

Такий підхід застосовується в алгоритмі Kraftwerk. Але, у випадку завищених коефіцієнтів віртуальні контакти та зв’язки починають домінувати над реальними, і вся система втрачає гнучкість пересування. І навпаки, при низькій вазі знову спостерігається вихід елемента за межі області розміщення.

Тому використання «сили» для ітераційного перерозподілу елементів є складним і накладає додаткові обмеження. Пошук шляхів вирішення цієї проблеми призвів до розгляду дифузії, як альтернативи силі. Цей підхід було реалізовано в алгоритмі DPlace.

1.6 Висновки

Як показали результати досліджень, алгоритми розміщення доволі стрімко розвиваються: старі алгоритми модифікуються, з’являються нові, прогресивніші, рішення. Для оцінки якості роботи алгоритму на сьогоднішній день вже не достатньо одного показника HPWL. У 2006 році ISPD ввели додаткові метрики, за допомогою яких можна оцінити не тільки довжину з’єднань, а й швидкість генерації рішення, щільність розміщення тощо. Це дозволило алгоритмам,які раніше залишалися непоміченими, вирватися вперед.

За результатами досліджень, найуспішнішою є група аналітичних

алгоритмів. Обираючи між Kraftwerk та DPlace, лідерами двовимірного розміщення серед анлітичних алгоритмів, для реалізації функції розміщення елементів інтегральної схеми на комутаційній платі було обрано алгоритм DPlace. Цей алгоритм має більш прозору реалізацію та цілком здатен задовольнити вимоги, що висуваються до даної роботи.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ РОЗМІЩЕННЯ БЛОКІВ ІНТЕГРАЛЬНИХ СХЕМ

2.1 Опис алгоритму розміщення елементів інтегральних схем

Для вирішення задачі розміщення інтегральних схем на комутаційній платі було обрано алгоритм DPlace. DPlace побудований на квадратичному розміщенні, але не обмежений ним. На відміну від традиційного квадратичного розміщення, DPlace розбиває задачу мінімізації довжин з'єднань та контролю щільності на 2 кроки. Таким чином на кожній глобальній ітерації ми маємо :

1 Крок попереднього розміщення для покращення показників щільності та скорочення кількості перекриттів елементів.

2 Крок мінімізації довжин з'єднань.

Для контролю руху елементів та запобігання повернення їх на початкові місця використовуються фіксовані віртуальні елементи ,або інакше – якірні. Створення якоря для кожного окремого елемента може призвести до виникнення надто суворих обмежень в русі реальних елементів. Саме тому DPlace використовує один якір на групу елементів, трансформуючи порцію багатоконтактних з'єднань в двоконтактні у відповідності до моделі «зірки». «Зірка» таким чином виступає в ролі якірного елемента, що дозволяє значно скоротити розмірність матриці з'єднань елементів та кількість ненульових елементів в ній.

На рисунку 2.1.1 показана узагальнена блок-схема алгоритму DPlace

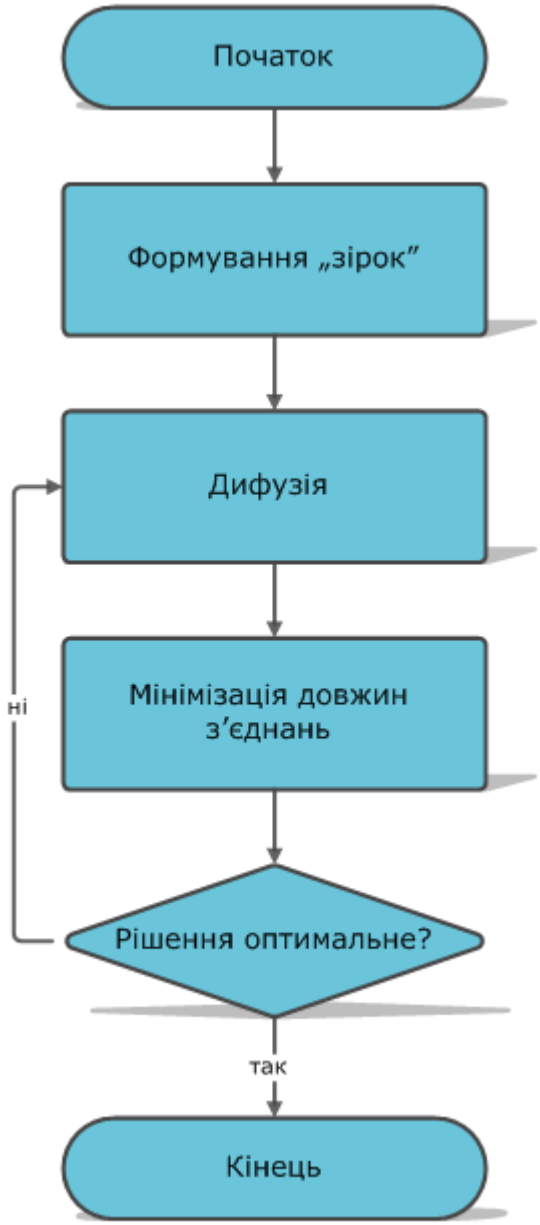


Рисунок 2.1.1 - узагальнена блок-схема алгоритму DPlace

2.2 Дифузія

На кожній ітерації глобального розміщення DPlace постає необхідність вирішувати задачу оптимізації щільності розміщення елементів для усунення перекриттів елементів. Для досягнення цієї мети DPlace використовує техніку дифузії.

Дифузія - це рух елементів із зон з високою концентрацією елементів до зон з низькою, доки концентрація усіх не стане рівною. Комутаційна плата

умовно поділяється на контейнери рівні за розміром, в які згодом дифузують елементи. Рух починається з великогабаритних елементів. Такими вважаються елементи, що за однієї з своїх сторін не поміщаються до контейнеру.

2.2.1 Визначення оптимальної кількості контейнерів та поділ області розміщення

Для досягнення результату, найбільш наближеного до рівномірного, необхідно створити умови, за яких можливе розміщення елемента в окремому контейнері(контейнерах).

Початкове ділення передбачає кількість контейнерів, що дорівнює або близька за значенням (але не менша) кількості елементів для розміщення. Ділення починається з більшої сторони і триває доти, доки кількість контейнерів не буде відповідати умовам оптимальності.

Приклад ділення для схеми з п’яти елементів зображено на Рисунку 2.2.1.1.

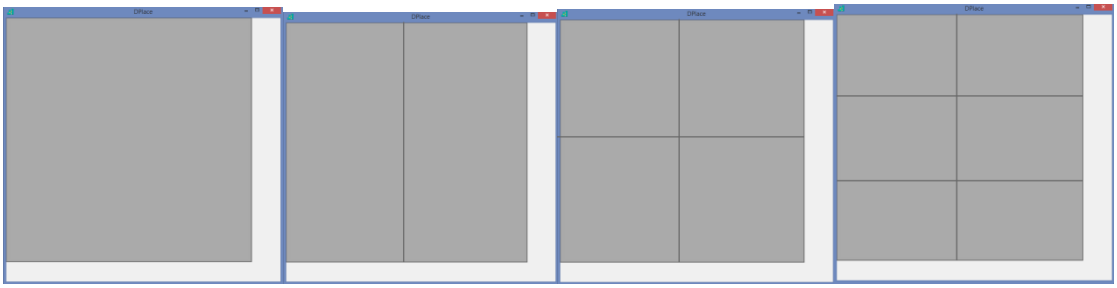


Рисунок 2.2.1.1- Процес поділу області розміщення на контейнери

2.2.2 Розміщення великогабаритних елементів

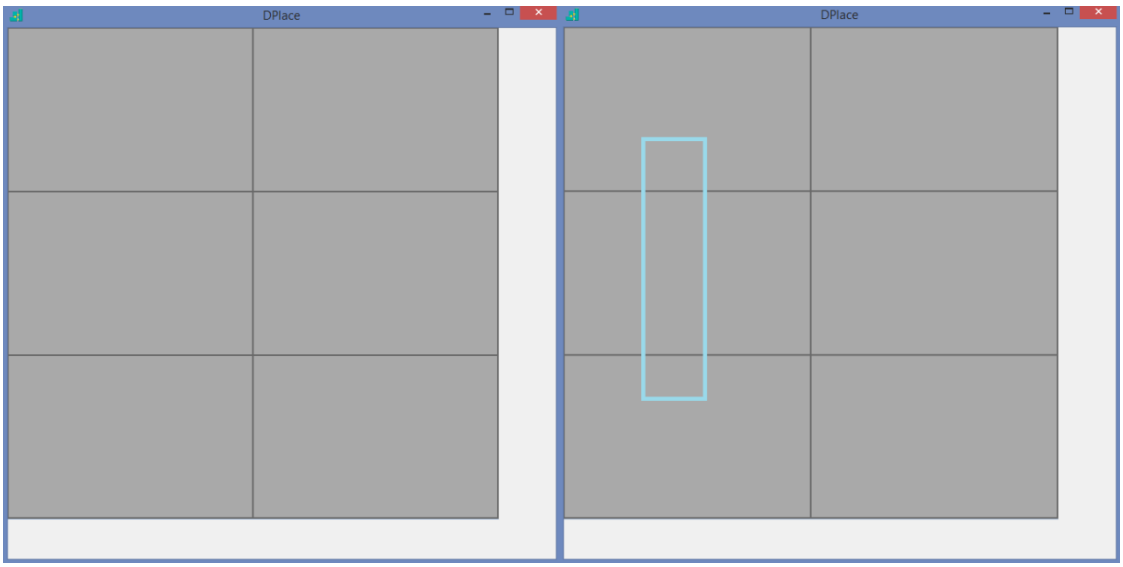
Після генерації координат контейнерів починається власне дифузія. Елементи сортуються за більшою стороною і дифузують в порядку зменшення габаритів.

У разі, якщо обидві сторони елементу менші сторін контейнера, він розміщується у центрі першого ж незайнятого контейнера. Якщо ж елемент не вписується хоча б за однією стороною в контейнер, починається процедура пошуку підходящого контейнера, що супроводжується зменшенням кількості контейнерів та перерахунком їх координат.

Розглянемо процедуру пошуку контейнера для великогабаритного елемента на прикладі схеми з шести елементів:

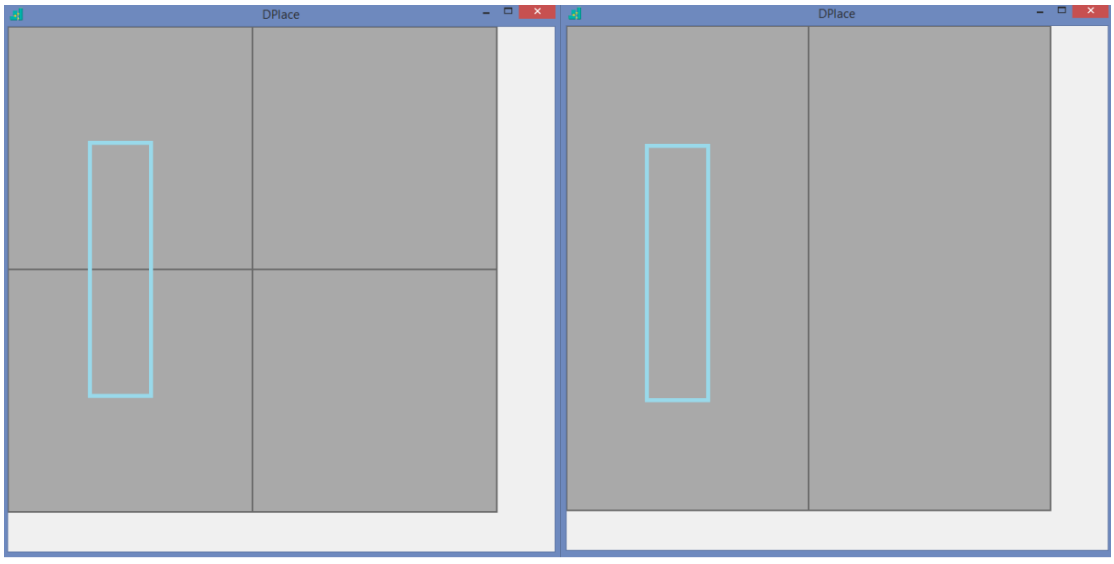
Приклад №1. Чотири елементи мають розмір 1x1 у.о. , один 4x1,а один 1x6. Розміри комутаційної плати при цьому встановимо 10x10.

На рисунку 2.2.1 показані етапи розміщення великогабаритного елемента.



а) початкова кількість контейнерів

б) перша спроба елемента розміститися в контейнері



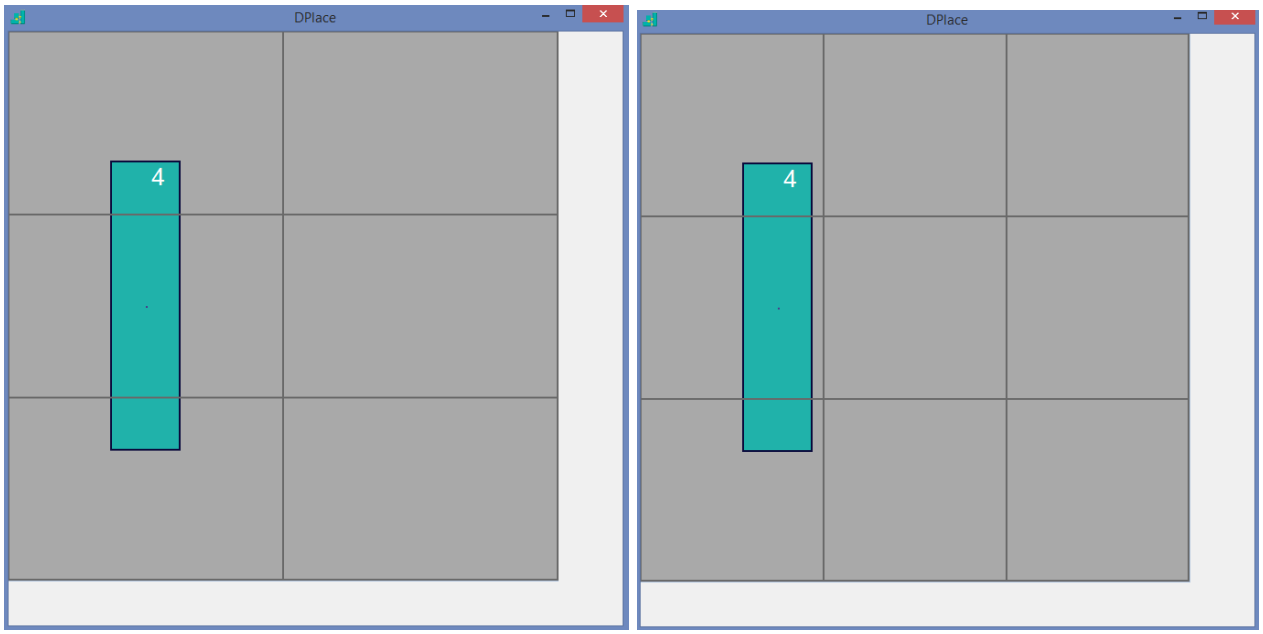
в) друга спроба елемента розміститися в контейнері

г) третя спроба елемента розміститися в контейнері

Рисунок 2.2.2.1 – Етапи розміщення великогабаритного елемента

Початкова кількість контейнерів та їх положення зображено на Рисунку 2.2.2.1 (а). Розмір контейнера при цьому становить 5x3 у.о. Очевидно, що за таких умов елемент розміром 1x6 у.о. розміститися цілком в одному контейнері не може (Рисунок 2.2.2.1 (б)). Звідси випливає необхідність зменшити кількість контейнерів за більшою стороною, як показано на Рисунку 2.2.2.1 (в). Тепер розміри контейнеру становлять 5x5 у.о. і це, як і раніше, не дозволяє елементу розміститися цілком в одному контейнері. Необхідно продовжити скорочення кількості контейнерів. Це приводить нас до ситуації, яка показана на Рисунку 2.2.2.1 (г). Розміри контейнера тепер становлять 5x10 у.о., що дозволяє елементу повністю розміститися в контейнері.

Розміщення великогабаритного елементу на комутаційний платі призводить до порушення умови оптимальності розбиття КП на контейнери, оскільки для оптимального розміщення нам необхідно мати мінімум 4 вільних контейнера. Щоб виправити цю ситуацію необхідно здійснити перерахунок оптимальної кількості контейнерів. Для цього відновимо початкову кількість та положення контейнерів, як показано на Рисунку 2.2.2.2 (а) та порахуємо кількість контейнерів, занятих великогабаритним елементом – їх 3. Отже, маючи 3 занятих контейнери та 5 нерозміщені елементи, отримуємо нову мінімальну кількість контейнерів - 8. На Рисунку 2.2.2.2 (б) зображений поділ області розміщення елементів на контейнери у відповідності до нової мінімальної кількості контейнерів.



а)Положення розміщеного великогабаритного елемента та початкове розбиття на контейнери

б)Розбиття на контейнери після перерахунку оптимальної мінімальної кількості контейнерів

Рисунок 2.2.2.2 – Перерахунок оптимальної мінімальної кількості контейнерів

2.2.3 Результати розміщення після дифузії

Дифузія є ітераційним процесом, який триває доти, доки хоча б один з контейнерів містить в собі більше ніж один елемент. Результатом дифузії є рівномірне розміщення елементів на області розміщення.

Розміщення після кроку дифузії для попереднього прикладу зображено на Рисунку 2.2.3.1.

Результати розміщення після кроку дифузії можуть сильно різнитися в залежності від вхідних розмірів комутаційної плати. Для прикладу №1, при заданій КП 30x10 у.о., результат розміщення буде мати вигляд, як показано на Рисунку 2.2.3.2.

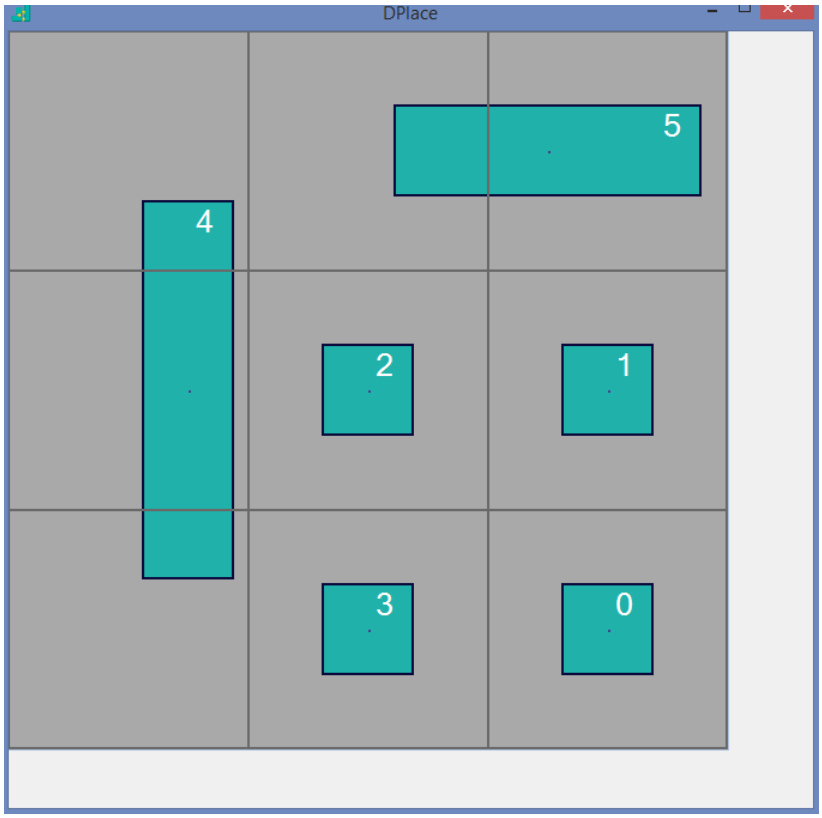


Рисунок 2.2.3.1 – Розміщення після дифузії для прикладу 1

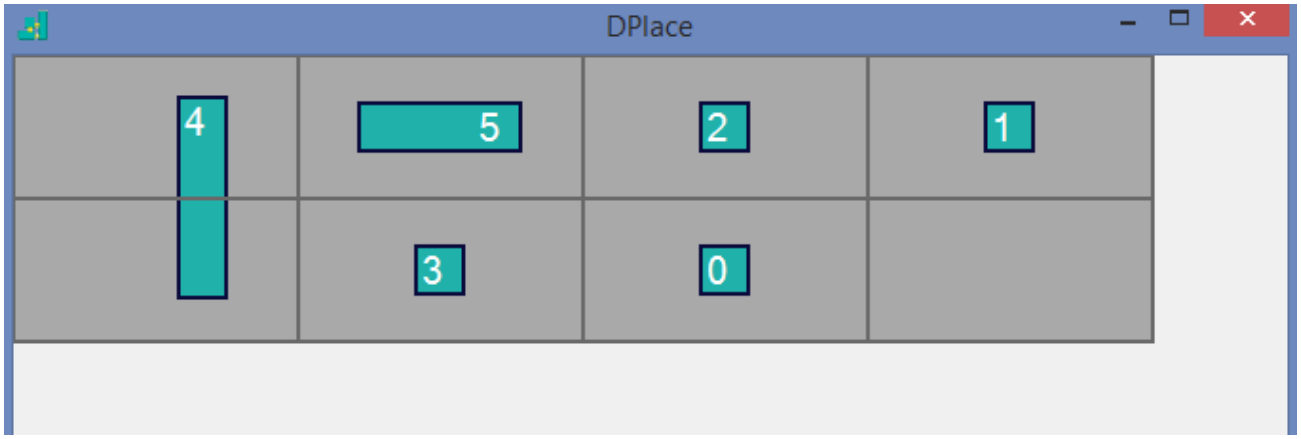


Рисунок 2.2.3.2 – Результат розміщення після кроку дифузії для прикладу 1 зі зміненими розмірами КП

2.3 Якірні елементи

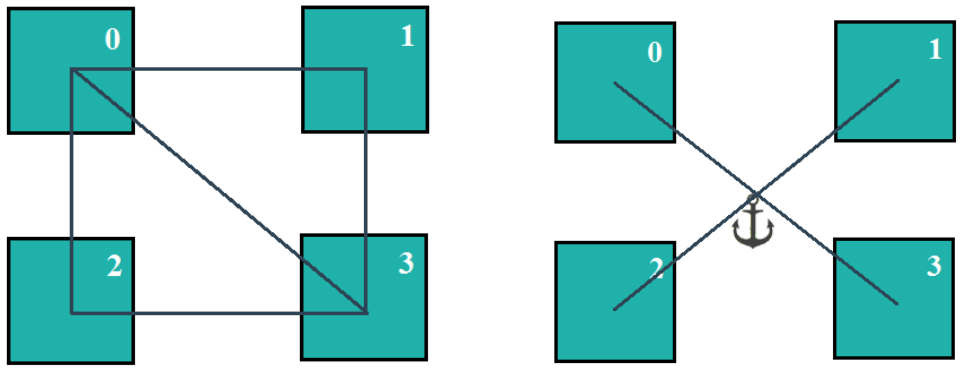
Якірний елемент – це деякий реальний або віртуальний фіксований елемент, який в процесі мінімізації довжин з'єднань не змінює свого положення. Таким чином, для нефіксованих елементів він виступає у ролі якоря до якого елементи змушені притягуватися для досягнення оптимальної

довжини з’єднань.

Створення якоря для кожного окремого елемента може призвести до виникнення надто суворих обмежень в русі реальних елементів. Замість цього доцільно використовувати один якірний елемент для групи реальних елементів.

Для цього DPlace використовує модель «зірки», що дозволяє перетворити багатоконтактні з’єднання в двоконтактні. «Зірка» таким чином виступає у ролі «якоря».

На рисунку 2.3.1 приведений приклад перетворення групи елементів в «зірку».



а) схема логічних з’єднань елементів до перетворення в «зірку»

б) схема логічних з’єднань елементів після перетворення на «зірку»

Рисунок 2.3.1 – Схема логічних з’єднань до і після перетворення на «зірку»

Для формування «зірки» обирається елемент з найбільшою кількістю зв’язків та найбільшою кількістю елементів, що не входять до жодної «зірки» . Його, та всі елементи, з якими він зв’язаний долучають до «зірки». Процес формування «зірок» триває доти, доки кожен з елементів не буде входити хоча б до однієї «зірки».

Створення «зірок» позбавляє необхідності проводити дифузію та мінімізацію довжин з’єднань одночасно для всіх елементів. Натомість, ці кроки

виконуються для кожної «зірки» окремо, що значно прискорює роботу програми, оскільки за таких умов кількість контейнерів значно зменшується і, відповідно, для перевірки занятих елементами контейнерів необхідно менше ітерацій.

Після мінімізації довжин з'єднань в зірках, відносне положення елементів щодо центру «зірки» запам'ятовується, а «зірка» розглядається як елемент, тобто дифузує за тими ж правилами, що і реальні елементи. Таким чином проводиться мінімізація довжин з'єднань між зірками, що дозволяє досягти більш оптимального результату.

Для покращення розв'язку можна, вважаючи «зірки» елементами, погрупувати їх в нові «зірки» за тими ж принципами, що і елементи, так би мовити створити «сузір'я» і провести для них процедуру мінімізації довжин з'єднань та дифузію. Таким чином найбільш зв'язані «зірки» розташуються поруч.

Керуючись цими принципами можна масштабувати топологію у відповідності до потреб користувача, для досягнення найкращого результату.

2.4 Мінімізація довжин з'єднань

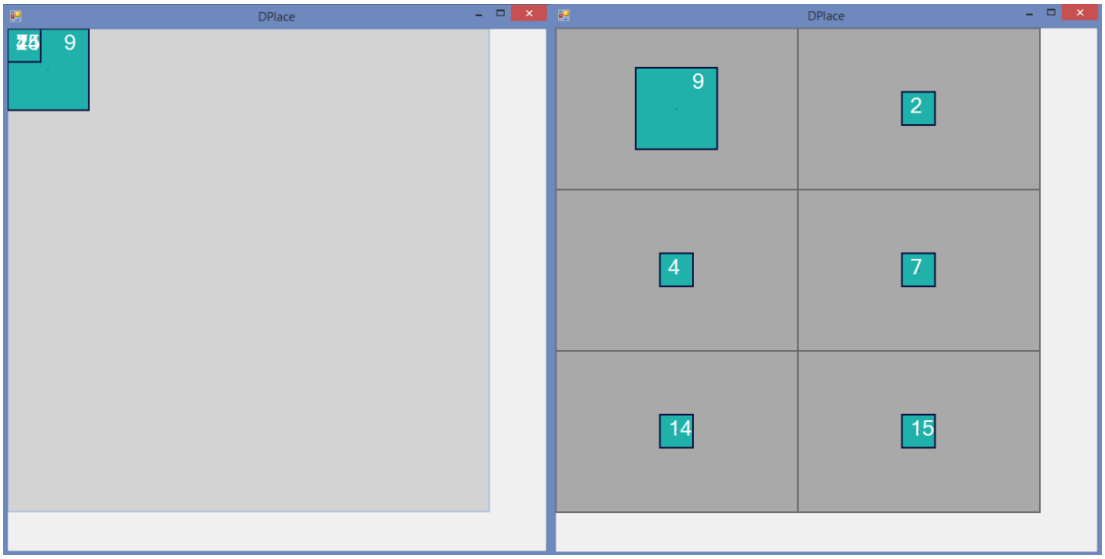
Процес мінімізації довжин з'єднань тісно пов'язаний з дифузією та слідує за нею на кожній ітерації глобального розміщення.

Мінімізація довжин з'єднань побудована на ідеї поступового зменшення області розміщення та рівномірного розташування елементів на ній. Процес зменшення області розміщення триває доти, доки є можливість рівномірного розташування елементів на ній.

Спершу область розміщення мінімізується по осі x та y одночасно. Після отримання результату, для його покращення, відбувається мінімізація по кожному напрямку окремо.

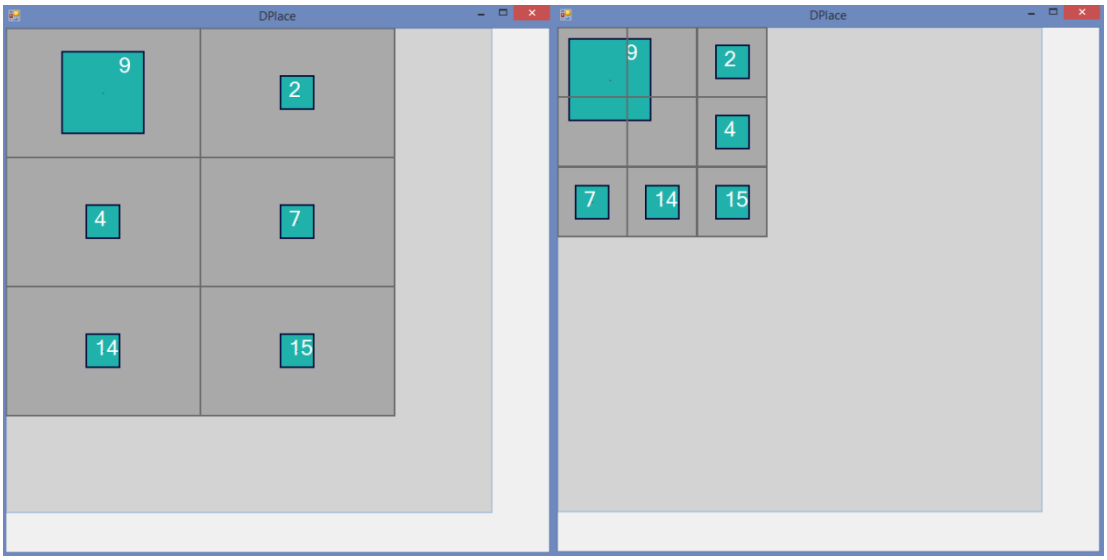
Цей алгоритм мінімізації довжин з'єднань дозволяє максимально зменшити довжину з'єднань, не створюючи при цьому перекриттів елементів.

На Рисунку 2.4.1 показані кроки мінімізації довжин з’єднань для однієї зірки із шести елементів.



а) Початкове розміщення елементів зірки

б)Результат розміщення після кроку дифузії



в) Розміщення елементів зірки після k кроків мінімізації довжини з’єднань та дифузії

г) Розміщення елементів зірки після k+n кроків мінімізації довжини з’єднань та дифузії

Рисунок 2.4.1 – Кроки мінімізації довжин з’єднань для однієї «зірки»

На рисунку 2.4.2 показаний остаточний результат розміщення для елементів в зірці.

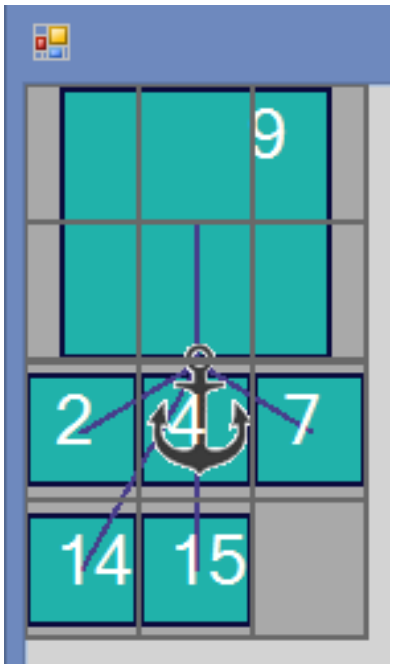
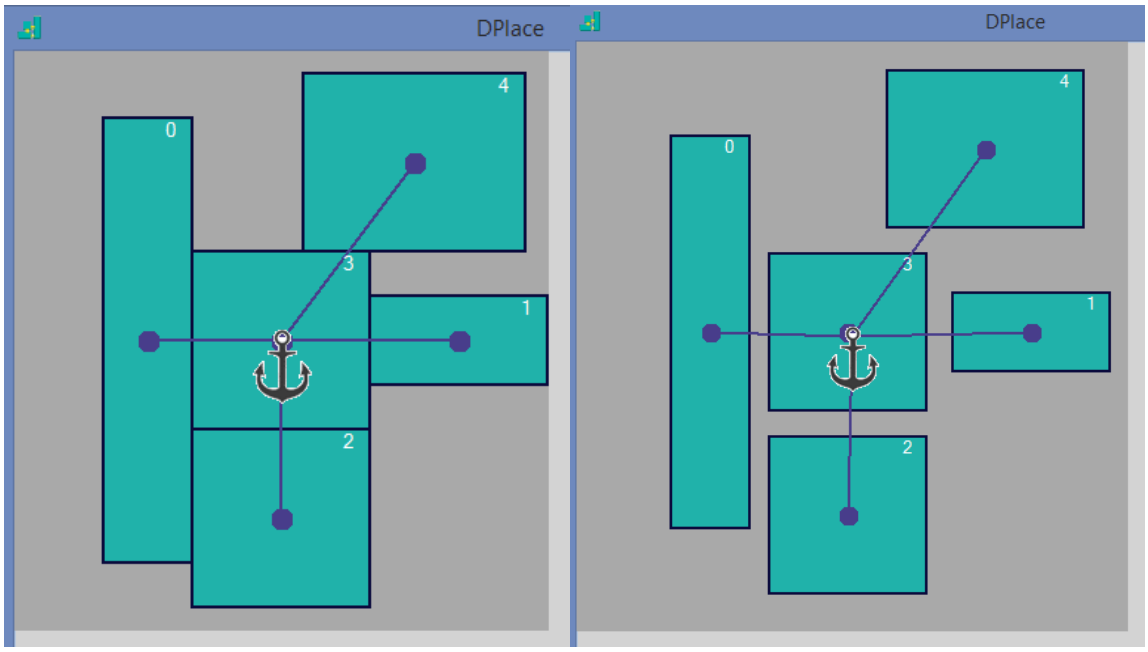


Рисунок 2.4.2- Остаточне розміщення елементів в «зірці»

За цими ж принципами відбувається мінімізація довжин з’єднань між «зірками», «сузір’ями» і т.д.

Результатами мінімізації розмірів області розміщення може бути дуже щільне розташування елементів, що дозволяє нам значно скоротити розміри комутаційної плати, проте перешкоджає прокладанню з’єднань (Рисунок 2.4.3 (а)). Щоб запобігти виникненню такої ситуації, необхідно встановити цільову щільність - мінімальний показник щільності(density target). Для цього до реальних розмірів елементів буде доданий відступ, а після отримання рішення, елементи повернуться до своїх нормальних розмірів. Таким чином ми отримаємо розміщення як показано на Рисунку 2.4.3 (б).

Врахування мінімальної дистанції між елементами дозволяє нам максимально зменшити площу, що займають елементи та мінімізувати довжину з’єднань, залишаючи при цьому місце для прокладання з’єднань.



а) розташування елементів без урахування цільової щільності

б) розташування елементів з урахуванням цільової щільності

Рисунок 2.4.3 – Розташування елементів після мінімізації без встановлення цільової щільності та з встановленням.

2.5 Узагальнення алгоритму

Вхідними даними для програми є список параметрів елементів(ширина та висота), зв’язки між ними, мінімальна дистанція між елементами та ширина траси та розміри комутаційної плати.

Розглянемо процес глобального розміщення на Прикладі № 2:

Нехай набір елементів складається з 49-ти елементів, параметри який зображено на Рисунку 2.5.1. Матриця з’єднань приведена в Додатку А.

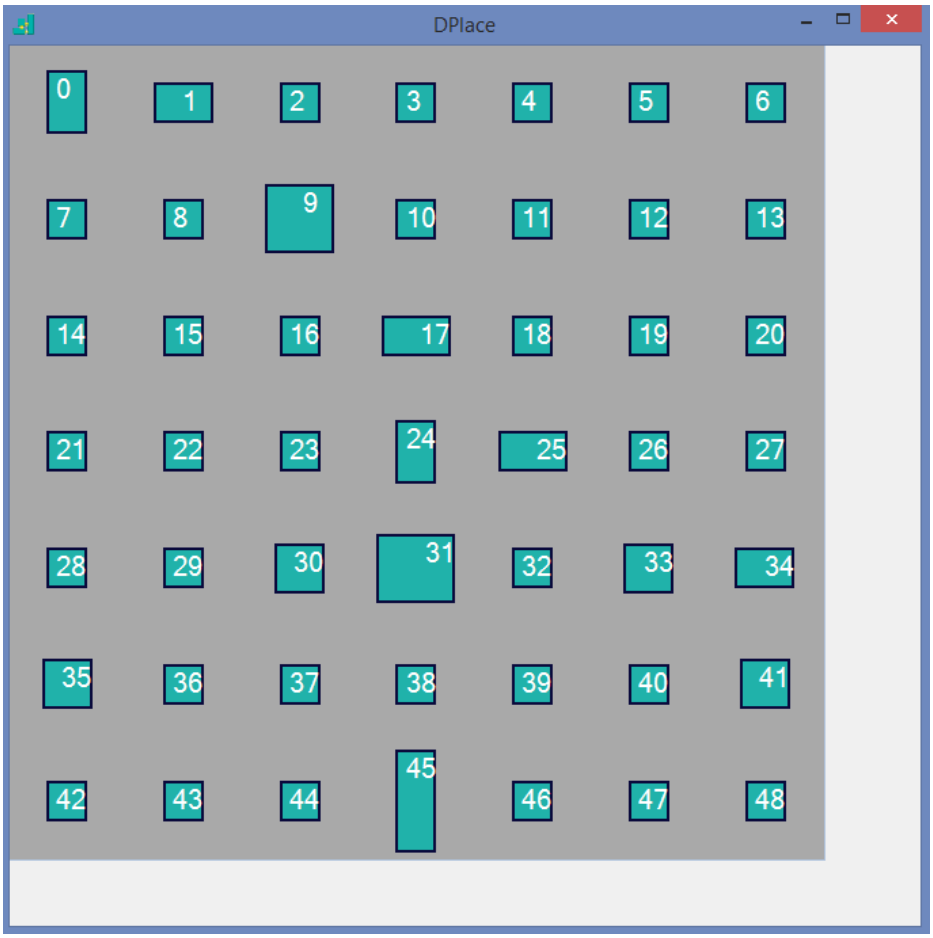


Рисунок 2.5.1 – Візуалізація вхідних даних(параметрів елементів) для прикладу № 2

На першому кроці роботи програми елементи групуються в «зірки». Для нашого прикладу – це 13 зірок такого складу:

1 2, 9, 4, 7,14,15

2 45, 39, 48, 47,46

3 1, 13, 8, 11, 19

4 34, 22, 27, 37, 36

5 17, 38, 21, 26, 28

6 25, 30, 33, 41

7 31

8 25, 18

9 0,5,12

10 10, 3, 6

11 24 ,32, 29

12 43, 40, 42, 44

13 20, 16, 23

Наступним кроком є мінімізація довжин з'єднань (що супроводжується дифузією) в межах кожної зірки окремо. На рисунку 2.5.2 показане положення елементів відносно центру «зірки» після мінімізації довжин з’єднань кожної «зірки».

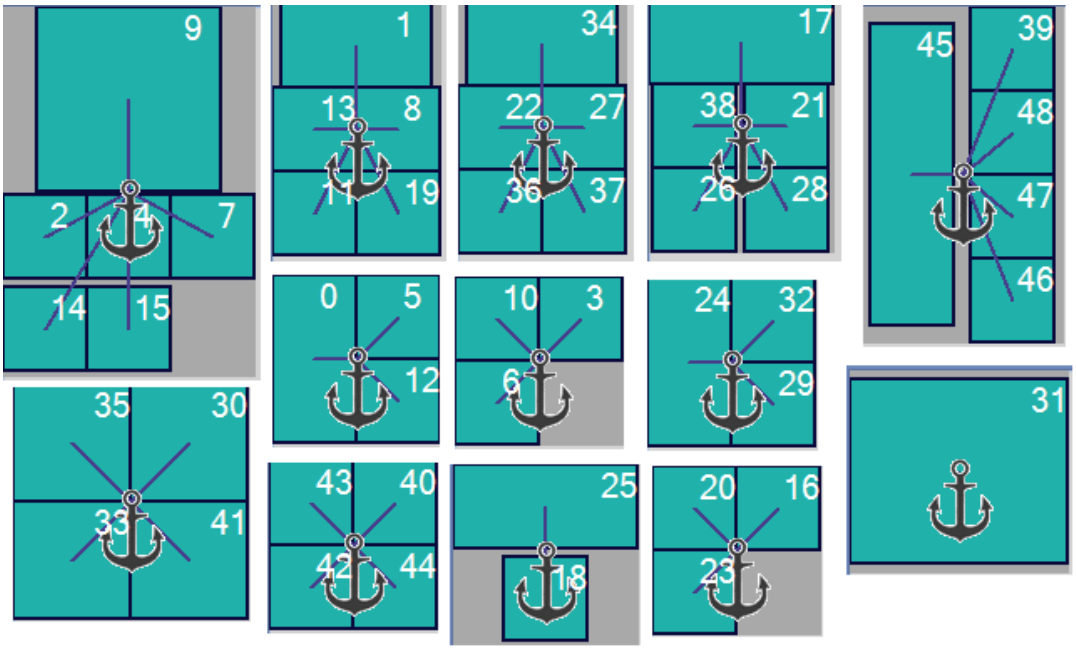
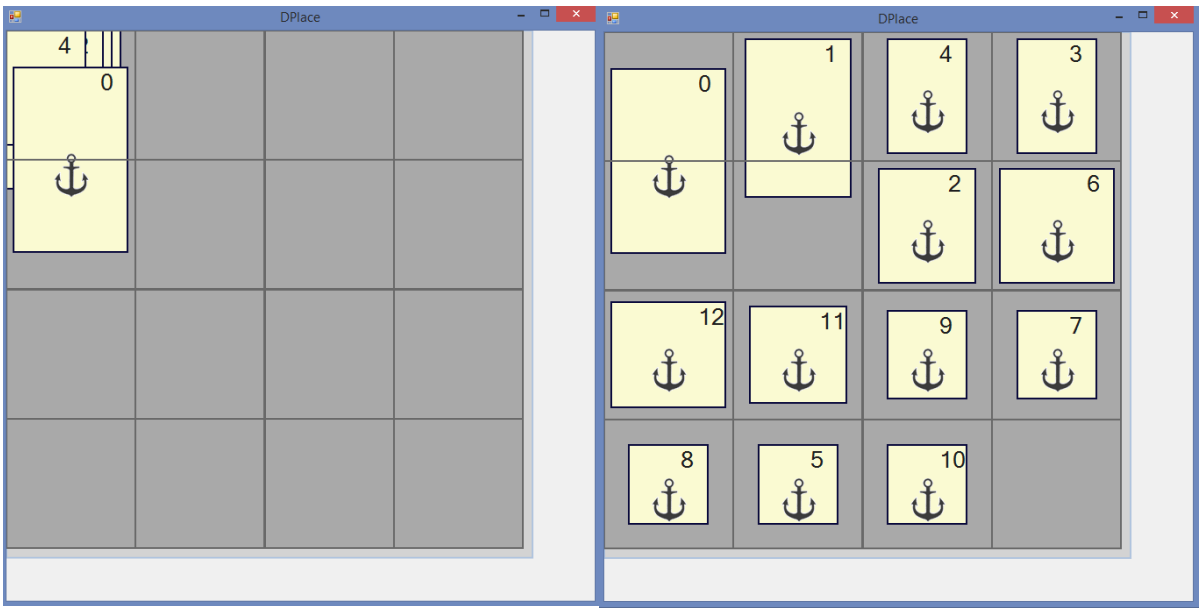


Рисунок 2.5.2 – Розташування елементів «зірки» відносно її центру після мінімізації довжин з’єднань кожної «зірки» для прикладу №2

Варто зауважити, що на цьому кроці елементи насправді ще не розташовані на комутаційній платі.

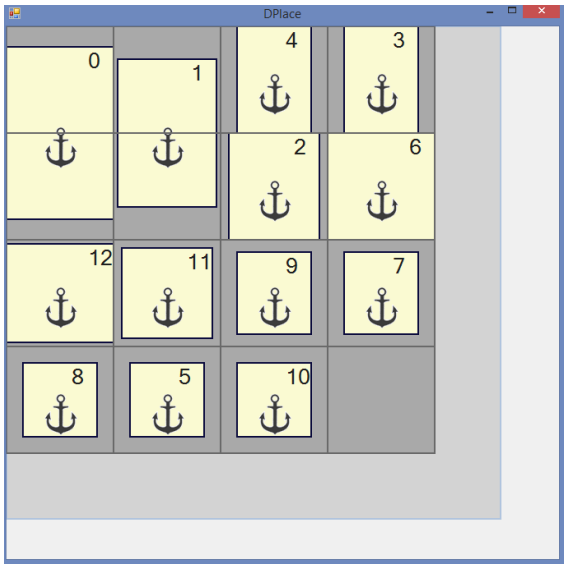
Наступним кроком є дифузія та мінімізація з’єднань між «зірками». «Зірки» у цьому разі виступають у ролі елементів. Шириною зірки вважається ширина обмежувальної рамки «зірки», а довжиною – довжина обмежувальної рамки «зірки». На рисунку 2.5.3 показані етапи розміщення «зірок» на комутаційній платі. Спершу всі «зірки» розташовані у верхньому лівому куті(рисунок 2.5.3 (а)), таким чином перший контейнер має найбільшу

щільність. Далі, в процесі дифузії, елементи поступово розміщуються в контейнерах з низькою щільністю, аж доки не буде досягнуто рівномірності розміщення (рисунок 2.5.3(б)). Далі, як і у випадку мінімізації довжин з'єднань між елементами, ми мінімізуємо область розміщення доти, доки можливе рівномірне розташування елементів на області розміщення. Рішення, отримане після цього кроку, показане на рисунку 2.5.3(в).



а) Початкове розміщення «зірок»

б) Розміщення «зірок» після кроку дифузії



в) Розміщення зірок після k кроків мінімізації довжин з'єднань

Рисунок 2.5.3 – Кроки мінімізації довжини з'єднань між «зірками» для прикладу № 2

Лише після цього кроку відбувається реальне розміщення елементів : їх координати вираховуються виходячи з нового положення центру зірки та відносного розташування елемента в «зірці».

Таким чином, для прикладу №2 отримуємо остаточне розміщення, що показане на рисунку 2.5.4.

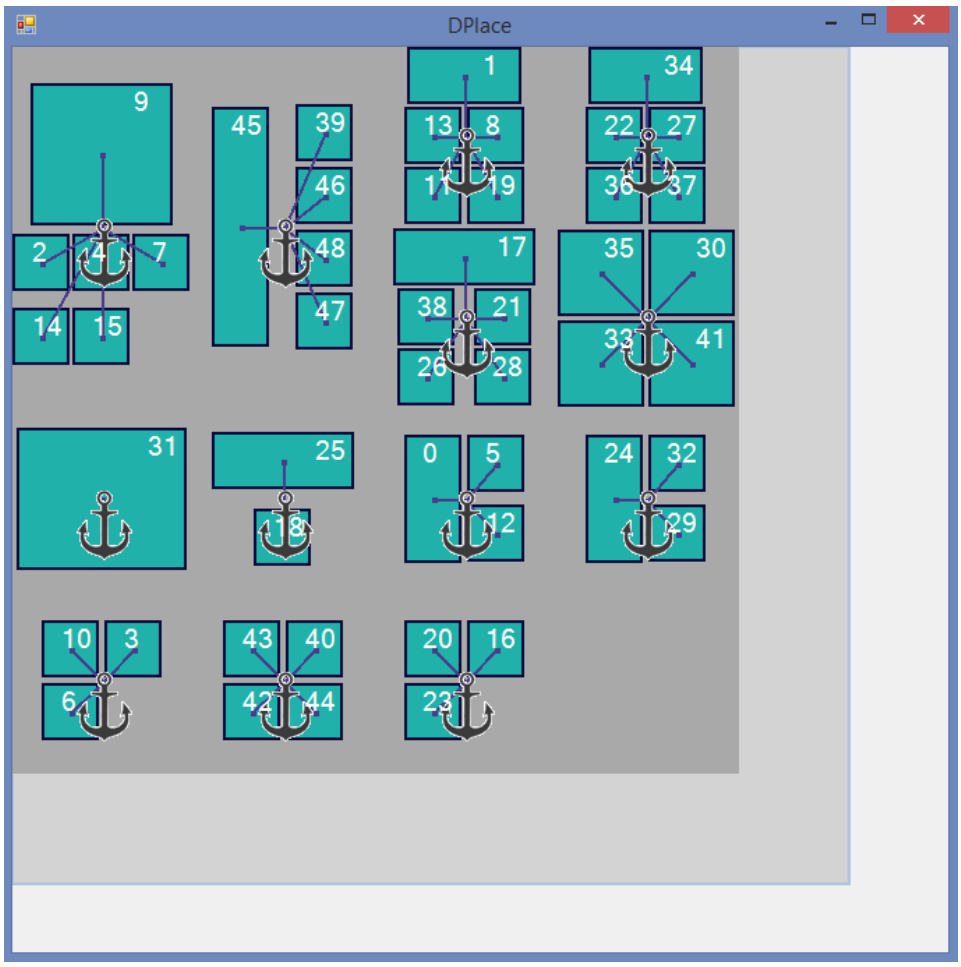


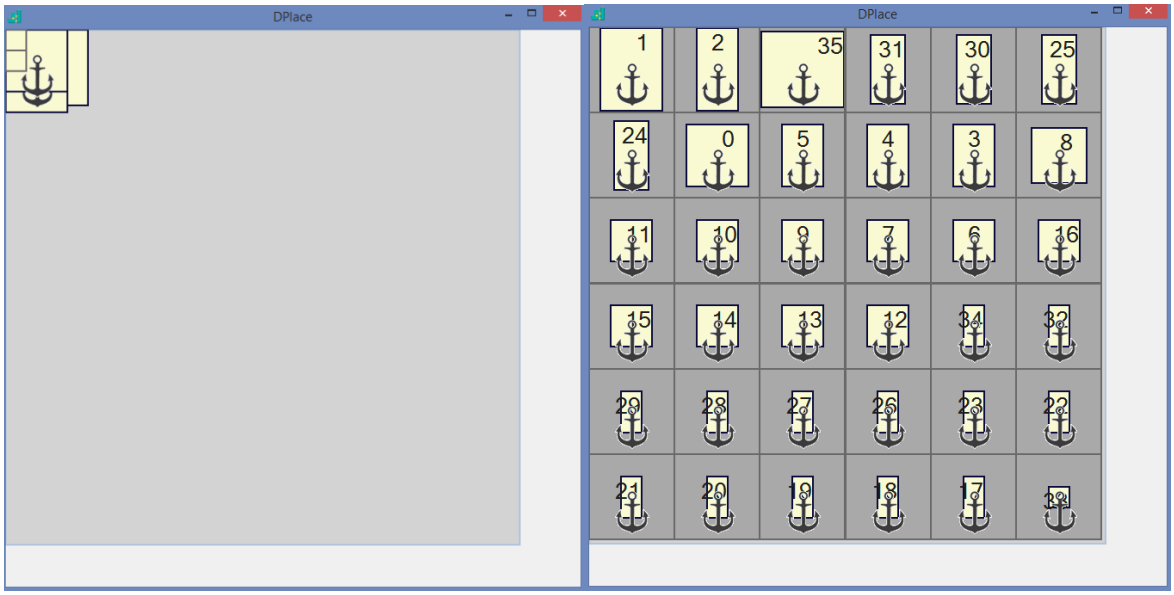
Рисунок 2.5.4 – Остаточне розміщення елементів для прикладу №2

2.6 Оцінка довжини з’єднань та приклади

Найпоширенішою метрикою для оцінки довжини з’єднань є півпериметр рамки,що обмежує з’єднання (half-perimeter bounding box wire length (HPWL)).[2]

HPWL добре підходить для оцінки двовимірного розміщення та легкий у обчисленні.

На рисунку 2.6.1(а) показано початкове положення «зірок» для прикладу №3,що складається зі ста елементів. HPWL при цьому становить 42 у.о. На рисунку 2.6.1 (б) показане положення «зірок» після кроку дифузії, HPWL при цьому становить 350 у.о.

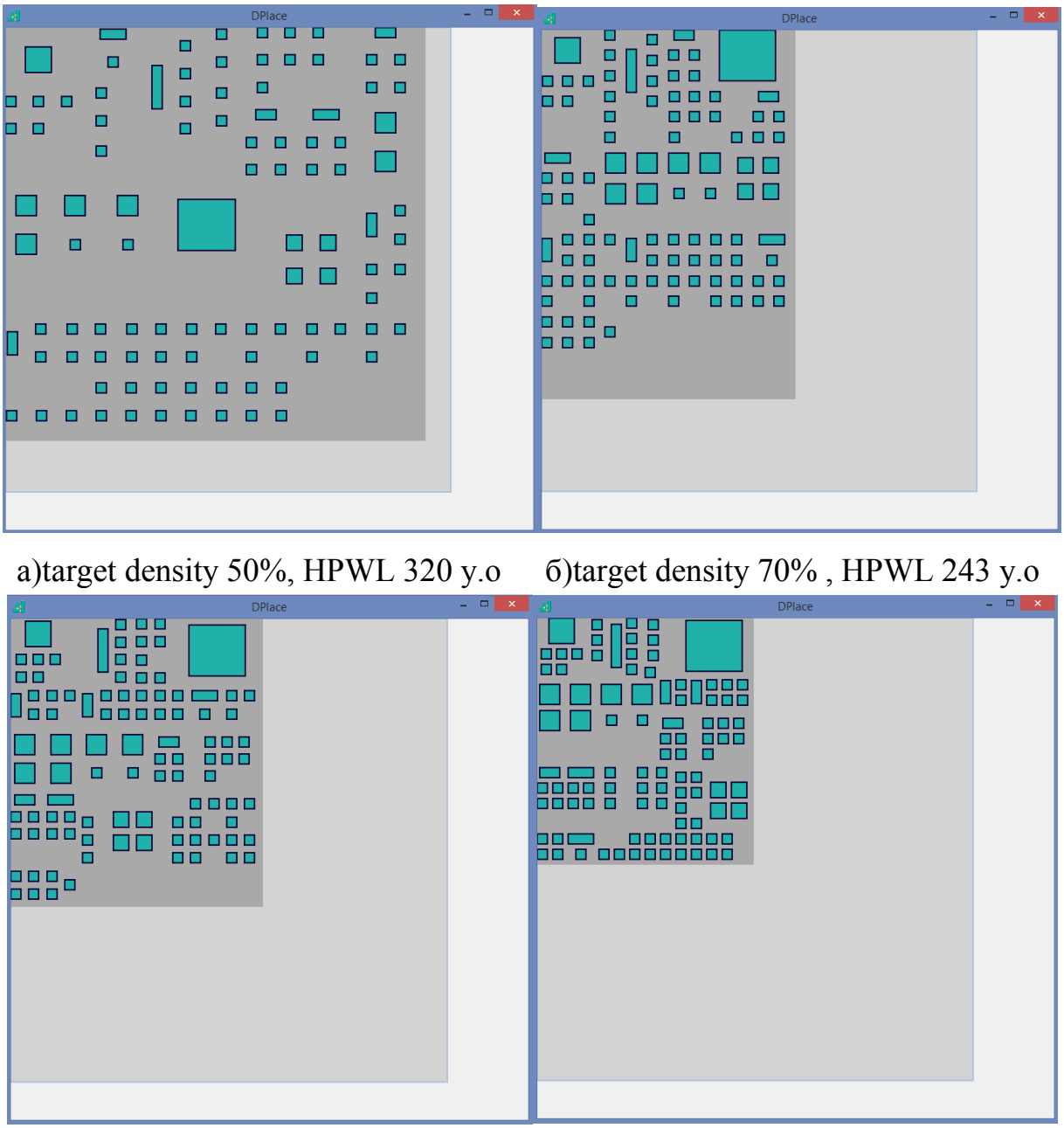


а) початкове розміщення

б) розміщення після кроку дифузії « зірок»

Рисунок 2.6.1 – Ітерації розміщення для прикладу №3

На рисунку 2.6.2(а) показане кінцеве положення елементів для density target 50%, HPWL скоротилося до 320 у.о. На рисунку 2.6.2(б) показане розміщення елементів при density target 70%, HPWL становить 243 у.о. На рисунку 2.6.2(в) показане розміщення елементів при target density 80%, HPWL при цьому становить 210у.о. Максимально допустимим density target для цієї схеми є 85%. При такому стисненні HPWL становить 180 у.о. На рисунку 2.6.2.(г) показане кінцеве розміщення елементів при даній цільовій щільності.



а)target density 50%, HPWL 320 у.о б)target density 70% , HPWL 243 у.о

в)target density 80%, HPWL 210у.о г)target density 85%, HPWL 180 у.о

Рисунок 2.6.2- Показники HPWL для різної цільової щільності

2.7 Висновки

В якості частини функціоналу генератора тополгій було реалізовано алгоритм розміщення елементів DPlace. Даний алгоритм , на відміну від більшості аналітичних алгоритмів, не використовує «силу», замість цього задача розміщення розбивається на два кроки: дифузію та мінімізацію довжин з’єднань. DPlace використовує модель «зірки», що дозволяє зменшити розмірність матриці з’єднань та покращити довжину з’єднань.

В подальших версіях даного програмного продукту пропонується вдосконалити технологію формування «зірок» таким чином, щоб після отримання рішення про розміщення елементів всередині «зірки», «зірка», виступаючи в ролі елемента, об’єднувалася в з іншими «зірками» в нові угрупування по принципу побудови «зірок», формуючи ,так звані, « сузір’я». А далі «сузір’я» , після отримання результату про розміщення «зірок» в них, створювали нові угрупування за тим самим принципом і так далі, доки це необхідно.

Реалізація цього підходу дозволила б значно скороти довжину найдовших зв’язків та покращити розв’язок загалом.

Для покращення результату на кроці мінімізації довжин з’єднань можна ввести змінний крок зменшення параметрів області розміщення. Це дозволило б за необхідності більше стиснути елементи схеми.

Результати тестування функції розміщення показали, що генерація рішення можлива для набору елементів, що містить великогабаритні елементи, на щільність розміщення це впливає несуттєво.

Тестові приклади показують, що алгоритм розміщення дозволяє досягти до 100% щільності, проте її легко регулювати для створення простору для прокладання з’єднань, задаючи цільову щільність. Таким чином алгоритм мінімізує периметр з’єднань, проте залишає можливим трасування.

3 РЕАЛІЗАЦІЯ ТРАСУВАННЯ З’ЄДНАНЬ

3.1 Алгоритми трасування монтажних з’єднань

Трасування інтегральних схем - один з етапів проектування радіоелектронної апаратури, що полягає в покроковому проектуванні структури провідників вручну або з використанням однієї з САПР. Вхідними даними для такої задачі є координати точок, між якими потрібно провести провідник. Ці координати обчислюються на етапі розміщення елементів інтегральної схеми, виходячи з остаточного положення елемента на комутаційній платі.

Алгоритм трасування істотно залежить від конструктивно - технологічних обмежень: допускаються перетини чи ні, чи можливий перехід з шару на шар, кількість шарів, ортогональне трасування чи ні.

В залежності від технології можна виділити три основних типи прокладання з’єднань[1]:

- схема реалізується на одному шарі;
- схема реалізується на декількох шарах, проте міжшаровий перехід здійснюється лише через контактні площадки з металізацією наскрізних отворів;
- схема реалізується на декількох шарах, а переходи здійснюються через спеціальні перехідні отвори, число яких може бути значним, а положення довільним.

У даній роботі реалізовано третій тип - для прокладання з’єднань доступно три шари металу, а переходи можуть розташовуватися у будь-якому місці.

Основними метриками якості трасування монтажних з’єднань є[1]:

- сумарна довжина з’єднань;
- довжина найдовшого з’єднання;
- кількість шарів;
- кількість перехідних отворів.

Для пошуку шляху між двома комірками найчастіше використовую такі два алгоритми:

1. Променевий алгоритм
2. Хвильовий алгоритм

У обох випадках комутаційну плату умовно поділяють на комірки, розмір яких становить ширину траси. Шлях прокладається по вільним коміркам. Для вибору комірки у променевому алгоритмі використовується ортогональна метрика[3] :

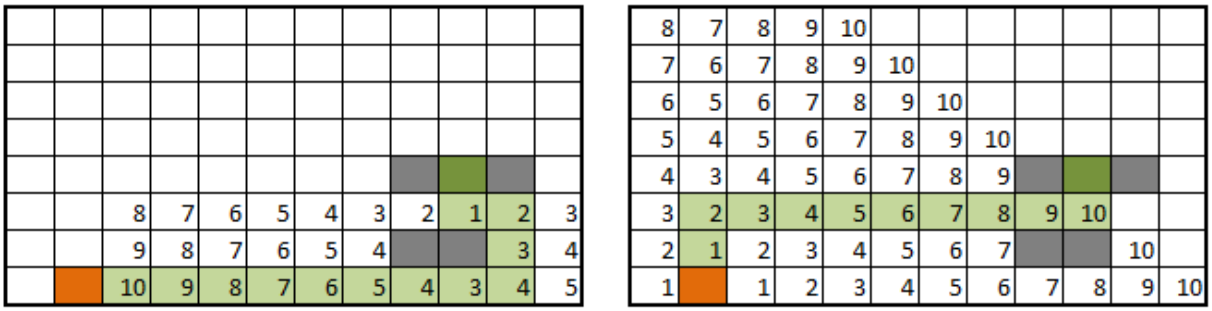
$$\boldsymbol{d_{ij} = \Delta x_{ij} + \Delta y_{ij},} \qquad (6)$$

де Δx_{ij}, Δy_{ij} – відстані між комірками i та j відповідно по осям X та У. Одиниця виміру –комірка.

У променевому алгоритмі, як і у хвильовому, обирається сусідня не зайнята комірка з найменшим номером. Спочатку перевіряється сусідня права комірка, за нею – верхня, потім - ліва і останньою - нижня.

Променевий алгоритм виконує операцію пошуку шляху від однієї точки до іншої значно швидше за хвильовий, проте отримане рішення не завжди є найоптимальнішим.

На рисунку 3.1.1 зображено результат прокладання шляху за обома алгоритмами для одного прикладу. Помаранчевим виділена початкова комірка, а темно-зеленим - кінцева. Сірі комірки - це комірки, що містять перешкоди, світло-зеленим кольором позначені комірки, через які проходить шлях. Як бачимо, для променевого алгоритму довжина шляху складає 12 комірок, в той час як для хвильового 10.



а) променевий алгоритм б) хвильовий

Рисунок 3.1.1 – Результат прокладання шляху променевим та хвильовим алгоритмом.

3.2 Алгоритм хвильового трасування (алгоритм Лі)

Для реалізації функції трасування було обрано хвильовий алгоритм, оскільки він дає найоптимальніший результат.

Алгоритм хвильового трасування - алгоритм пошуку найкоротшого шляху на пленарному графі. Був запропонований у 1962 році при формалізації алгоритмів трасування друкованих плат[4].

Алгоритм працює на дискретному полі, розбитому на комірки шириною яких є ширина траси. Множина всіх комірок поділяється на : «прохідні», «непрохідні», тобто ті, що містять перешкоди, «прохідні» лише для горизонтальних провідників, «прохідні» лише для вертикальних провідників, початкова комірка та кінцева.

Робота алгоритму включає в себе три етапи: ініціалізація, поширення хвилі, відновлення шляху.

Підчас ініціалізації комутаційна плата розбивається на комірки, коміркам присвоюється статус – «прохідна» чи «непрохідна», запам’ятовуються початкова та кінцева комірки.

Далі, від стартової комірки здійснюється крок у сусідню комірку, при цьому перевіряється чи є вона «прохідною». В залежності від технології,

сусідні комірки визначаються по різному. Якщо можливе прокладання з’єднань по діагоналі, то сусідніми комірками вважаються всі 8 комірок, що межують з даною. У нашому випадку можливе прокладання з’єднань лише по вертикалі та по горизонталі. Для такої задачі сусідніми вважаються лише 4 комірки, що прилягають своїми сторонами до даної комірки, проте не кутами.

При виконанні умови «прохідності» та якщо комірці досі не був присвоєний номер фронту хвилі, їй присвоюється атрибут,рівний кількості кроків від початкової комірки. На першому кроці це буде одиниця. Кожна наступна комірка породжує наступні кроки в сусідню комірку. Процес поширення хвилі триває доти, доки кінцева комірка не буде помічена або ж не буде жодної комірки для подальшого поширення хвилі[4].

На рисунку 3.2.1 показано як поширення хвиля з контакту на елементі 0 в контакт на елементі 1.

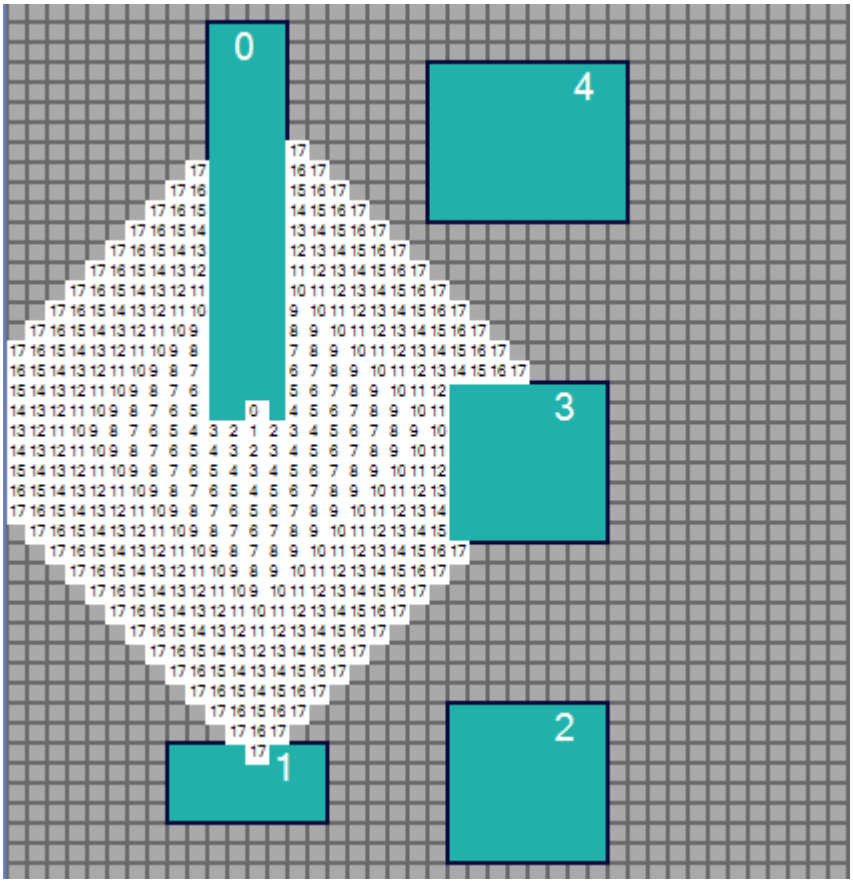


Рисунок3.2.1–Поширення хвилі

Відновлення шляху відбувається в зворотному напрямку : при виборі комірки від кінцевої до початкової на кожному кроці обирається комірка, що має значення фронту хвилі на одиницю менше ніж поточна комірка. Очевидно, що таким способом отримується найкоротший шлях[4]. На рисунку 3.2.2 показане відновлення шляху для попереднього прикладу.

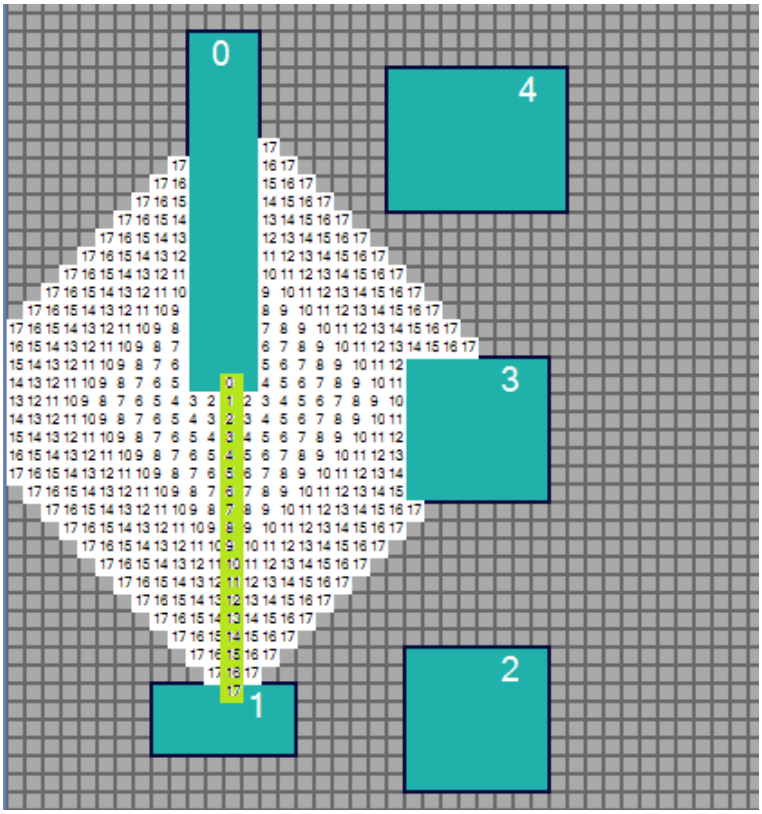


Рисунок 3.2.2 – Відновлення шляху

3.3 Модифікації алгоритму

Як слідує із наведеного опису алгоритму, час, затрачений на вирішення задачі, та об’єми пам’яті при використанні алгоритму Лі пропорційні кількості вузлів в сітці. Для сучасних топологій ця величина буде дуже великою. Для того щоб зменшити час обробки даних та покращити розв’язок застосуємо такі модифікації:

1 Вибір початкової точки. При пошуку маршруту необхідно обирати той контакт, що лежить ближче до кінцевої точки, якщо він доступний[1]. Відстань можна оцінити за ортогональною метрикою(формула 1). Це

дозволить скоротити час обробки даних, оскільки менша кількість комірок буде задіяна у процесі.

2 Введення міток «зайнято по х», «зайнято по у». Часто найоптимальніший шлях прокладання з’єднань для декількох випадків однаковий на деякому відрізку. Конструктивне рішення обмежує нас у кількості шарів і унеможливлює прокладання з’єднань для такого випадку. Щоб зменшити кількість накладань провідників введемо мітки для комірок: « зайнято по х» та « зайнято по у». Таким чином, при розповсюдженні хвилі, комірка буде вважатися недоступною, якщо вона помічена як «зайнята по х», а хвиля розповсюджується по горизонталі. Аналогічним чином по вертикалі. На рисунку 3.3.1 показана ситуація, в якій провідник мав би накластися на вже проведене з’єднання. Помаранчевим виділені комірки, що позначені як « зайняті по х», блакитним - «зайняті по у», фіолетовим – по х і по у.Оскільки спочатку перевіряється ліва комірка, а лише потім нижня, рух мав би продовжуватися по горизонталі, проте введена модифікація цього не дозволяє, оскільки ця комірка помічена як «зайнята по х та по у». Дана особливість дозволяє зменшити кількість перекриттів з’єднань, що є важливою метрикою для оцінки результатів трасування.

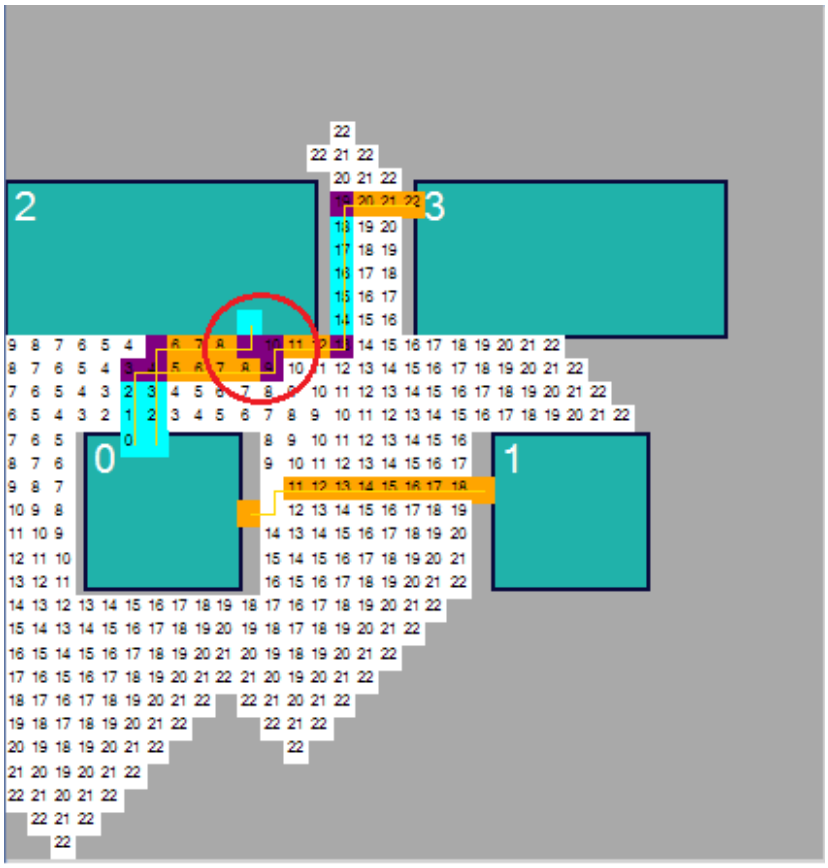


Рисунок 3.3.1 Усунення надмірних перетинів

3.4 Приклади трасування

На рисунку 3.4.1 показано результат трасування для невеликого прикладу з п’яти елементів.

На рисунку 3.4.2 показаний результат трасування для прикладу №2.

Приклад № 3 також піддається трасуванню для будь-якої з наведених цільових щільностей, проте результати трасування важко оцінити візуально. Довжина півпериметра для цього прикладу наведена у попередньому розділі.

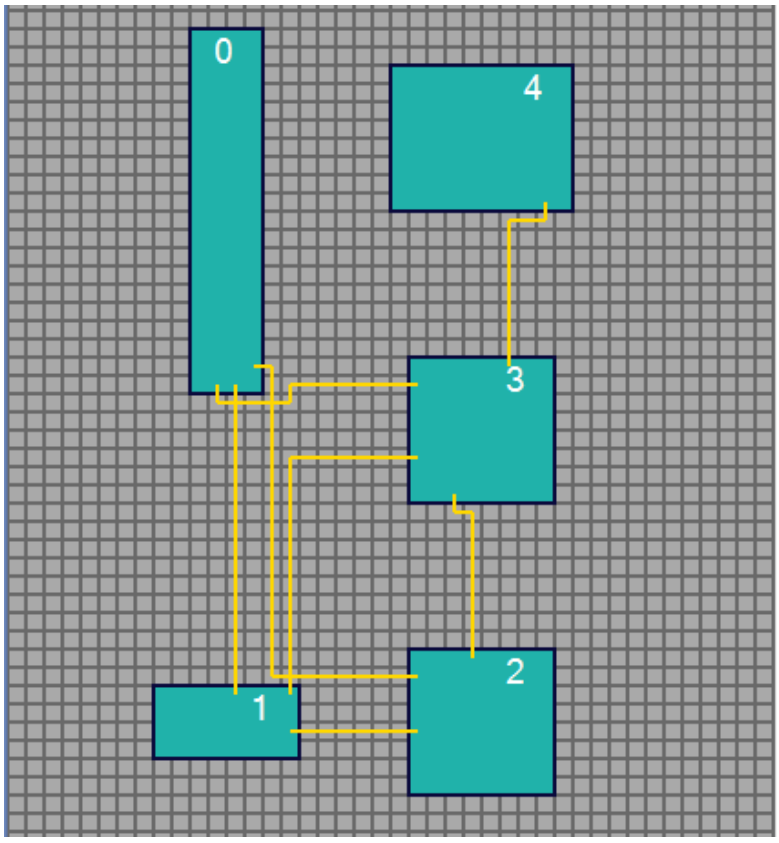


Рисунок 3.4.1 - Приклад результату трасування

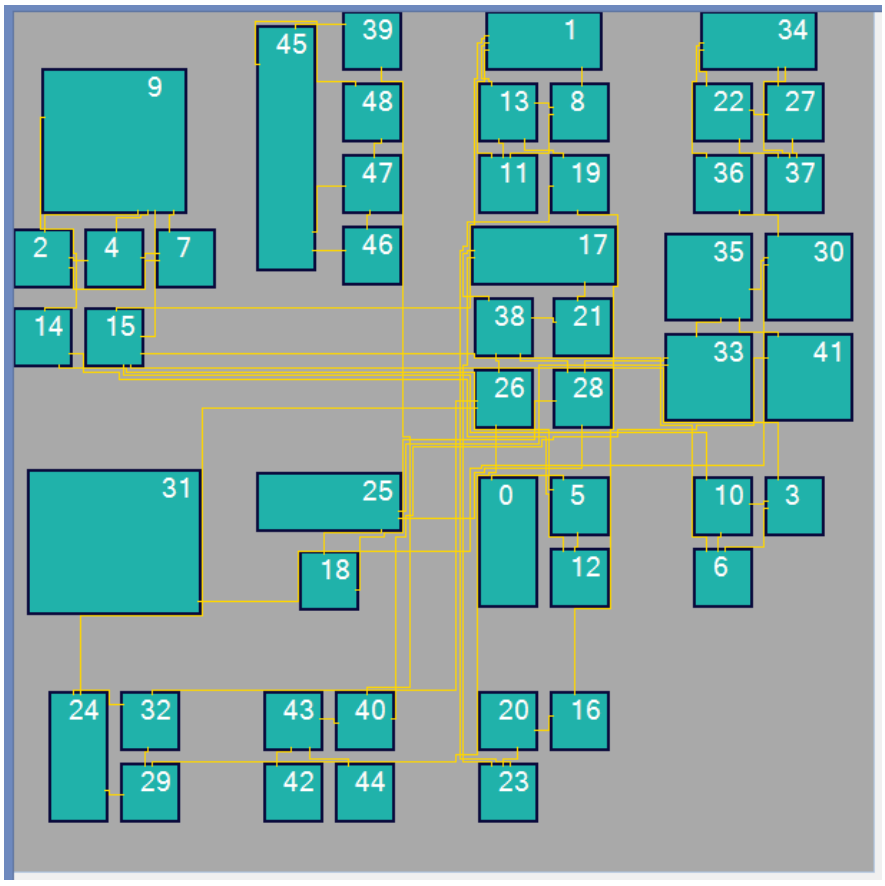


Рисунок 3.4.2 – Результат трасування для прикладу №2

3.5 Висновки

Отже, за результатами аналізу існуючих алгоритмів трасування було обрано хвильовий алгоритм як алгоритм, що дає найоптимальніший результат. З метою удосконалення рішення було проведено модифікації алгоритму, а саме – пошук найоптимальніших для з'єднання контактів елемента та усунення надлишкових перекриттів.

В подальших версіях даного програмного продукту пропонується вдосконалити алгоритм трасування таким чином: для кожної «зірки» виконувати трасування окремо, після чого виконувати трасування між «зірками». Такий підхід дозволить мінімізувати площу поширення хвилі, що допоможе скоротити час генерації рішення.

Іншою можливою модифікацією є розповсюдження хвилі одночасно з кінцевої комірки та з початкової. Таким чином поле пошуку зменшується приблизно в два рази.

Окрім атрибуту, що позначає номер фронту хвилі можна ввести додатковий атрибут, який допоможе визначитись з напрямок у разі, коли існує декілька можливих шляхів. Таким атрибутом може бути ортогональна відстань від поточної комірки до кінцевої.

За допомогою цих модифікацій можна буде ще більше покращити час обчислення рішення.

Результати проектування розміщення провідників з використанням хвильового алгоритму показали, що рішення, отримані за допомогою алгоритму розміщення DPlace піддаються трасуванню.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Вступ

Питання охорони праці є ключовим при створенні безпечного та комфортного робочого місця. Умови праці безпосередньо впливають на фізичний та психологічний стан людини, а вони , в свою чергу, визначають продуктивність праці. Саме тому, будь–яка людина, працівник чи працедавець, зобов'язана уміти аналізувати і оцінювати весь комплекс шкідливих чинників.

В умовах роботи в сучасному офісі фахівець постійно потерпає від стресів. Однією з причин ,що призводять до цього, є порушення норм організації робочого місця – підвищений рівень шуму, замала площа робочого місця, погана вентиляція тощо.

Організація повинна забезпечити виконання вимог безпеки праці таким чином, щоб вони відповідали фізіологічним, ергономічним і технічним вимогам відповідно до чинного законодавства України [12]. Дана дипломна робота виконана з урахуванням вимог охорони праці, пожежної та екологічної безпеки виробництва.

4.2 Аналіз умов праці у приміщенні

4.2.1 План виробничого приміщення

Приміщення розташоване в 16-ти поверховому офісному приміщенні на третьому поверсі. Розрахунки умови праці розглянуті на прикладі офісу у дослідницькому центрі. Офіс розрахований на 7 робочих місць. Основні показники даного приміщення:

* ширина = 6 м, довжина =9 м, висота = 3 м;

* площа приміщення = 54 м^2, об'єм = 162 м^3;

* площа обладнання = 7,75 м^2, об'єм обладнання = 5,996 м^3.

План робочої зони даного приміщення зображений на рис. 4.1.

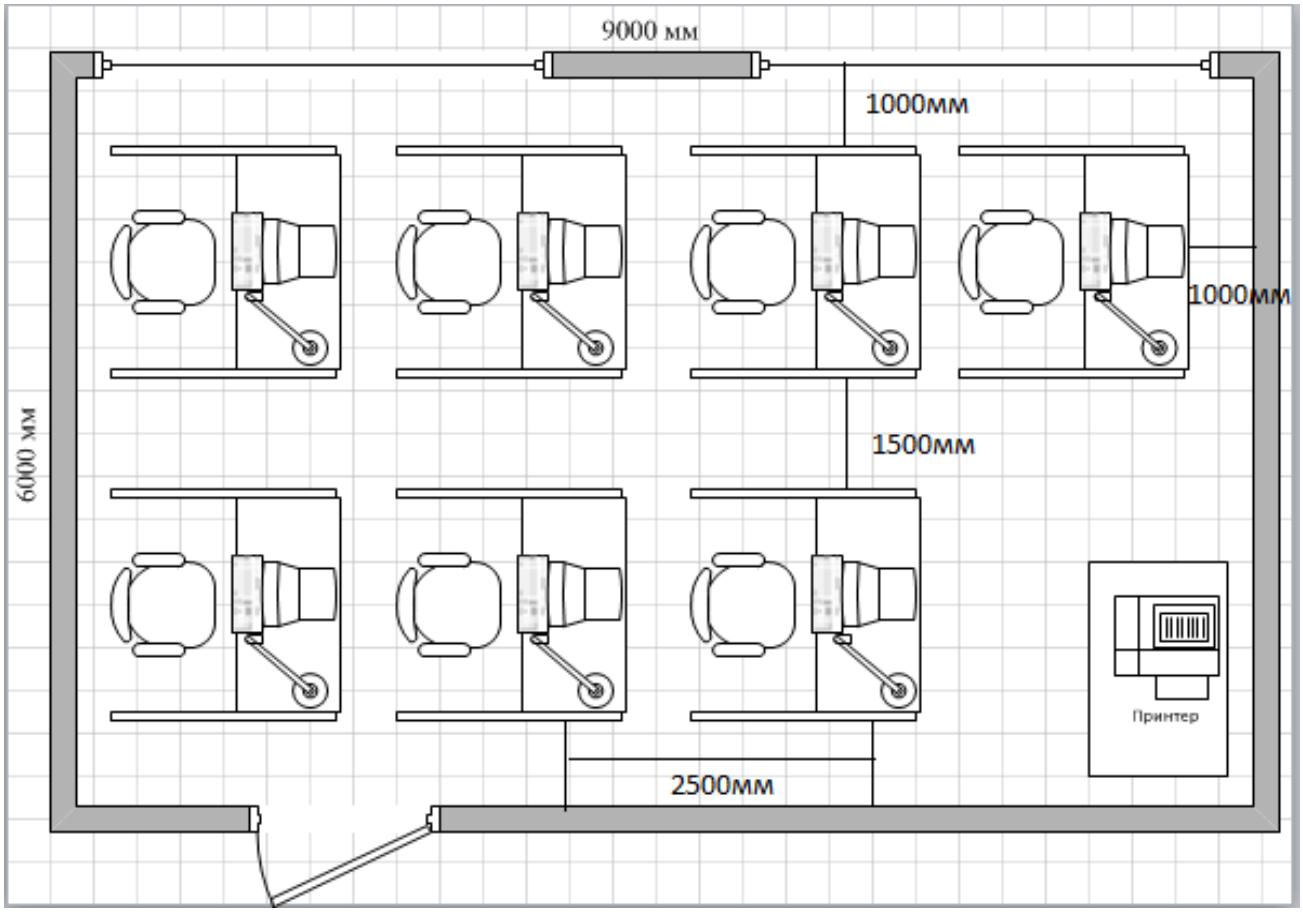


Рисунок 4.1 – Приміщення роботи користувача ПЕОМ

Розраховуємо об’єм та площу приміщення на одного працівника відділу:

$V1пр = Vпр – Vобл/n = (162 - 5,996)/7 = 22,3\ м^3$

$S1пр = (54 – 7,75)/2 = 6,6\ м^2$

Відповідність вимогам нормативних документів щодо площі та об’єму на одного працівника, а також правильність розташування обладнання, наводиться у табл. 4.1.

Таблиця 4.1 – Характеристики робочого місця користувача

Найменування параметра	Значення	
	Нормативне	Фактичне
Площа	не менше 6 м^2	6,6 м^2
Висота	не менше 3 м	3 м
Обсяг на одну людину	20 м^3	22,3 м^3
Висота робочої поверхні на рівнем підлоги	700 мм	700 мм
Оптимальні розміри робочої поверхні	1900 х 900 мм^2	1900 х 900 мм^2
Дальність клавіатури від краю стола	не більше 300 мм	300 мм
Відстань між очима користувача і екраном монітора	400 – 800 мм	600 мм
Регулювання висоти сидіння робочого стільця	400 – 500 мм	450 мм
Глибина сидіння	380 мм	380 мм
Ширина сидіння	400 мм	400 мм
Висота опорної поверхні спинки	300 мм	380 мм
Ширина опорної поверхні спинки	380 мм	380 мм

Отже, фактичні показники вважаються цілком задовільними.

4.3 Шкідливі та небезпечні фактори

Основні джерела небезпек та шкідливих факторів наведені в таблиці 4.2.

Таблиця 4.2 – Основні джерела небезпеки

Основними джерелами небезпеки шкідливих факторів для працівників в даному відділі є	Електронебезпека
	Низька (висока) температура повітря
	Висока відносна вологість повітря
	Нервово-емоційні перевантаження
	Перенапруження зорового аналізатора
	Пожежа

4.3.1 Несприятливі мікрокліматичні умови

Мікроклімат приміщення, визначається наступними параметрами:

- температура повітря, t (^{0}C);

- відносна вологість повітря, φ (%);

- швидкість руху повітря, v (м/с);

- інтенсивність теплового випромінювання, j (Вт/м^2);

- температура поверхонь будівельних конструкцій, $t_п$ (^{0}C).

Перші три параметри встановлюються відповідно до пори року і категорії роботи за енерговитратами. Робота оператора ЕОМ, яка розглядається, виконується сидячи і не потребує фізичного напруження; витрати енергії становлять до 120 ккал/год. Відповідно така робота відноситься до категорії Іа, і нормовані параметри мікроклімату визначені у [13] і наведені у табл. 4.3:

Таблиця 4.3 - Оптимальні параметри мікроклімату

Пора року	Категорія робіт	Температура повітря, град. С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		Оптимальна	оптимальна	оптимальна
Холодна	легка-1 а	22 – 24	40 - 60	0,1
Тепла	легка-1 а	23 – 25	40 - 60	0,1

4.3.2 Освітленість приміщення

Дані про зорову роботу та шкідливі фактори освітленості наведені відповідно в таблицях 4.4 та 4.5.

Таблиця 4.4 – Характеристика зорової роботи

Параметр	Значення
Розподільна здатність екрану	1024x768
Величина 1 пікселя	0.3 мм
Розмір об’єктів на моніторі (в ср.)	7 – 10 пк (2 – 3мм)
Розмір мінімальних об’єктів	3 пк (0.9 мм)
Контраст	великий
Фон	світлий
Точність зорової роботи	середня точність

Таблиця 4.5 – Шкідливі фактори – освітлення

Шкідливі фактори освітленості	Вплив на організм людини	Заходи щодо усунення або зниження їх впливу на працюючих
Недостатня освітленість робочого місця	Передчасне зорове стомлення, зменшення рівня працездатності, збільшення ймовірності механічних помилок	Збільшення кількості світильників, розповсюджених рівномірно по робочій кімнаті.
Підвищена яскравість світла	Знижує світлочутливість очей, зменшення рівня працездатності, збільшення ймовірності механічних помилок	Реорганізація світильників, встановлення жалюзі на вікнах

4.3.3 Оцінка шуму та вібрації

В таблиці 4.6 наведені шкідливі фактори шуму.

Таблиця 4.6 – Шкідливі фактори - шум

Шкідливі фактори	Вплив на організм людини	Заходи щодо усунення або зниження їх впливу на працюючих
Підвищений рівень шуму на робочому місці	Є джерелом погіршення слуху та зниження продуктивності праці людини, що може призвести до збільшення ймовірності виникнення механічних помилок під час роботи з даними	Зниження шуму в джерелі створення за рахунок якісного монтажу окремих вузлів комп’ютера, зменшення шуму на шляху поширення (за рахунок місцевої та загальної звукоізоляції, раціонального розміщення робочих місць та технічного устаткування, встановлення шумовлюючих екранів, поглинаючих фільтрів); раціоналізація режимів праці та відпочинку; використання глушителів шуму, засобів віброізоляції, звукоізоляції та звукопоглинання

4.3.4 Випромінювання при роботі з обчислювальною технікою

При роботі з комп'ютером людина піддається впливу ряду небезпечних і шкідливих виробничих факторів, приклади яких наведені в таблиці 4.8.

Таблиця 4.7 – Шкідливі та небезпечні фактори при роботі з електронно-обчислювальною технікою

Небезпечні і шкідливі виробничі фактори	Влив на організм людини	Заходи щодо усунення або зниження їх впливу на працюючих
Електростатичне поле, магнітне поле, високоінтенсивне іонізуюче випромінювання, ультрафіолетове і м’яке рентгенівське випромінювання, змінне елктромагнітне поле	Значна напруга зорового апарату з проявом скарг на незадоволеність роботою,зменшення працездатності,головні болі , дратівливість, порушення сну, втома	додержуватись відстані від очей до екрану в межах 60 - 80 см; використання додаткових захисних екранів; дотримуватись вимог санітарних норм режиму праці та відпочинку;

4.3.5 Небезпека ураження людини електричним струмом

Для захисту електрообладнання від короткого замикання застосовуються запобіжники. На шафах управління, розподільних коробках нанесені знаки електричної напруги. Джерела ураження людини електричним струмом наведені у таблиці 4.8.

Таблиця 4.8. – Джерела враження людини електричним струмом

Джерела ураження струмом	Конструкторські заходи зменшення небезпеки	Організаційно-технічні заходи зменшення небезпеки
Пошкодження електрично-струменевого шнуру	Захисний кожух	Проведення інструктажу про безпечні методи роботи з електроустаткуванням
Електропроводка в приміщенні	Прихована (АПВ, 5,25 мм)	Проведення інструктажу
Підлога	Ізолююча (лінолеум)	Проведення інструктажу

Загальними рекомендаціями є обслуговування діючих електроустановок, проведення в них оперативних перемикань, ремонтних, монтажних, налагоджувальних робіт виключно електротехнічним персоналом.

4.3.6 Оцінка пожежної безпеки

Будинок виконано із залізобетонних плит, тобто він відноситься до II ступеня вогнестійкості[14,15]. Межа вогнестійкості конструкції 0,5-2,5 год. У приміщенні знаходяться важко горючі тверді й волокнисті речовини й матеріали. За рівнем вибухонебезпечної і пожежної небезпеки приміщення відноситься до категорії В: стелажі з дерева, підлога, касовий апарат.

Джерела виникнення пожежі наведені в таблиці 4.9.

Таблиця 4.9. – Джерела виникнення пожежі

Джерела виникнення пожежі	Конструкторські заходи зменшення небезпеки	Організаційно-технічні заходи зменшення небезпеки
Перевантаження проводів	2 вогнегасники: вогнегасник CO_2 (вуглекислотний) переносний ВВ-3 (ВВК 2) місткістю балона 3 літри (2 кілограми), призначений для гасіння загорянь різних речовин, горіння яких не може відбуватися без доступу повітря, на один вогнегасник = 107 м^2; датчики пожежної безпеки.	Дотримання технологічного режиму, контроль параметрів температури за допомогою термометра LaCrosse WS8005; використання кондиціонеру LG G24LHT (для кондиціонування і провітрювання) розвантаження електровузлів після виконання роботи, реконструкція електропроводки; ознайомлення з інструкціями по використанню електроприладів; узгоджений план евакуації
Коротке замикання		
Поява великого перехідного опору		

4.4 Висновки

В даному розділі були розглянуті основні вимоги до охорони праці у приміщеннях, в яких, за звичай, працюють оператори ЕОМ. Зазначено вимоги до освітлення, мікроклімату, приміщення, електробезпеки. Приділено увагу пожежній безпеці – визначені категорії приміщень за вибухо- та пожежною небезпекою, клас можливих пожеж. Встановлені необхідні типи пожежної сигналізації та вогнегасників. Наведено інструкцію з техніки безпеки при роботі з ПК.

ВИСНОВКИ

Для вибору алгоритму розміщення було проведено аналіз популярних існуючих алгоритмів. Матеріалами для досіджень слугвали результати конкурсних досліджень , опубліковані в 2007 році, проведених International Symposium on Physical Design (ISPD) протягом 2005 і 2006 років.

За результатами даного дослідження алгоритмом для реалізації було обрано аналітичной алгоритм DPlace, оскільки він був одним із кращих алгоритмів двомірного розміщання та, навідміну від свого конкурента Kraftwerk, він не використовує поняття «сили», що значно спрощує його реалізацію.

За результатами тестування функції розміщення було визначено, що алгоритм здатен досягати заданої цільової щільності.

Використання моделі «зірки» дозволяє досягти непоганих показників сумарної довжини з'єднань, проте для мінімізації найдовших зв'язків необхідно удосконалити реалізований алгоритм. Процедуру мінімізації довжин з'єднань необхідно проводити групуючи «зірки» в так звані «сузір'я» , вважаючи їх при цьому елементами. Далі так само групувати «сузір'я» і масштабувати таким чином топологію доти, доки це необхідно.

Оскільки лише алгоритми, результат роботи яких підлягає трасуванню, мають право розглядатися, була реалізована функція трасування, що дало можливість оцінити роботу DPlace на практиці. Для реалізації трасування був обраний хвильовий алгоритм, оскільки він найкраще здійснює пошук найкоротшого маршруту.

Результати тестування показали, що рішення, згенеровані за допомогою алгоритму DPlace, підлягають трасуванню.

Для покращення результатів проектування розміщення провідників алгоритм трасування був модифікований таким чином, щоб зменшити кількість перекриттів провідників.

Для пришвидшення генерації рішення на кроці трасування з'єднань

пропонується обмежити поле пошуку. Якщо проводити спершу трасування з’єднань в кожній «зірці» окремо, хвиля не буде поширюватися на ділянки плати, що скоріше за все задіяні не будуть. Це дозволяє економити час та ресурси пам’яті.

Іншою можливою модифікацією є встановлення додаткового атрибута для комірки. Для ситуації, коли існує декілька доступних кроків, необхідно обирати ту комірку, у якої найменша ортогональна відстань від поточної комірки до кінцевої.

Оскільки дана розробка є лише частиною генератора топологій, існує багато шляхів для удосконалення програми. Такими можуть бути: реалізація інтерфейсу користувача, реалізація взаємодії з бібліотеками, синтезатор логіки тощо.

ПЕРЕЛІК ПОСИЛЯНЬ

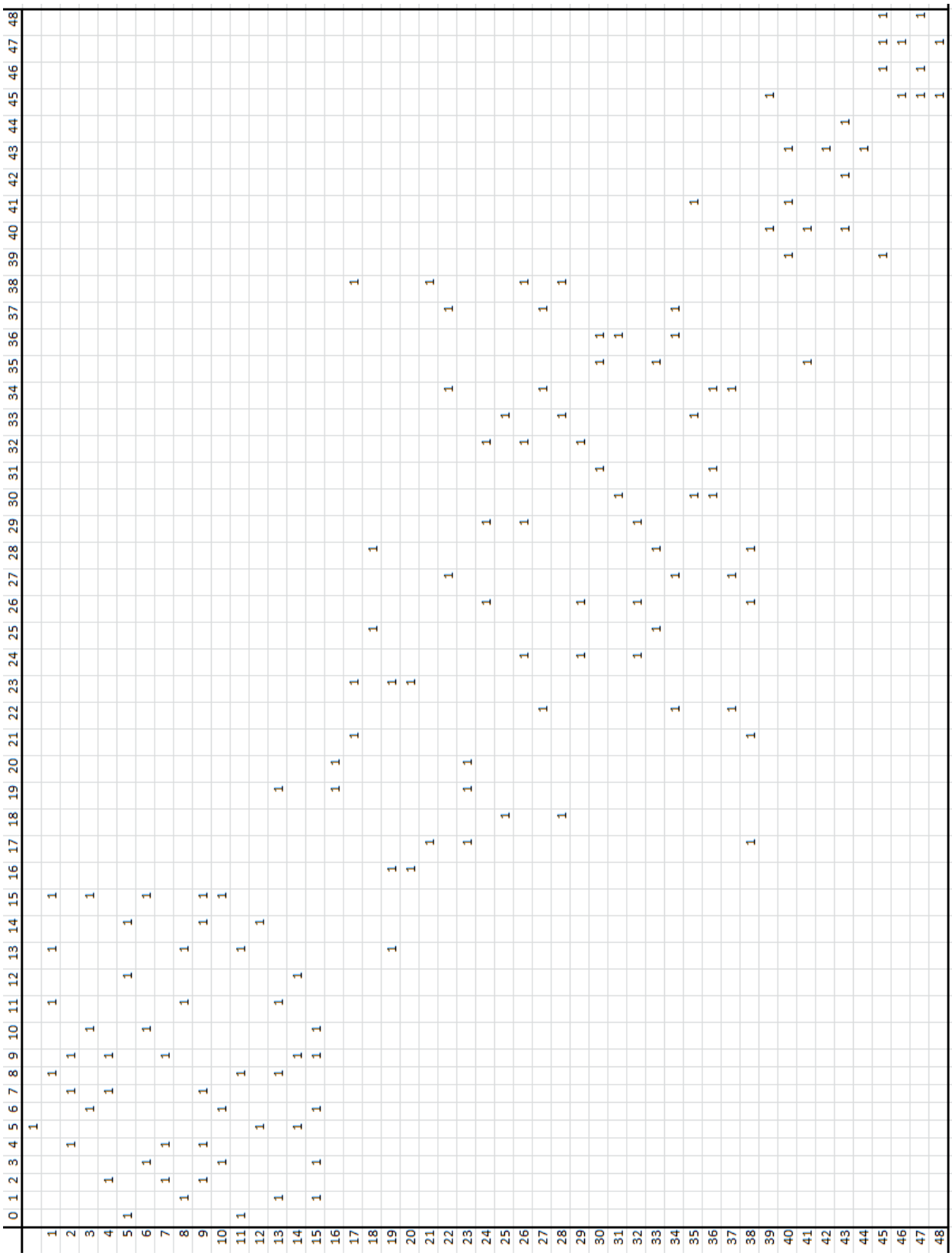
1. Проектирование топологии печатных плат и интегральных схем- Режим доступа: http://elib.bsu.by/bitstream/123456789/8753/4/%D0%A2%D0%B5%D0%BC%D0%B0_5_%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%82%D0%BE%D0%BF%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B8.pdf
2. Modern Circuit Placement Best Practices and Results/ Gi-Joon Nam Jason Cong - Springer Science+Business Media, LLC, 2007 -319 с
3. Лучевой алгоритм –Режим доступа : http://100byte.ru/100btwrks/wv/r.html
4. Алгоритм Ли – Режим доступа: https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B8
5. Общая постановка задачи – Режим доступа: http://www.intuit.ru/studies/courses/650/506/lecture/11525?page=1
6. Класификация алгоритмов размещения – Режим доступа: http://bigor.bmstu.ru/?cnt/?doc=010_EDA/eda081.mod/?cou=Default/020_ECAD.cou
7. Модель квадратичного назанчения – Режим доступа: http://studopedia.ru/4_5327_model-kvadratichnogo-naznacheniya.html
8. Трассировка печатных плат – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D1%81%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D0%BD%D1%8B%D1%85_%D0%BF%D0%BB%D0%B0%D1%82
9. Техника разводки печатных плат – Режим доступа: http://www.pcbtech.ru/pages/view_page/141
10. Технологический процесс в электронной промышленности – Режим

доступа: https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_%D0%B2_%D1%8D%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D0%BF%D1%80%D0%BE%D0%BC%D1%8B%D1%88%D0%BB%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8

11. Офіційний сайт Департаменту навчальної роботи НТУУ «КПІ». [Електронний ресурс]. – Режим доступу : http://osvita.kpi.ua. – Дата доступу : 25.04.2012.
12. Державні санітарні правила і норми. Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин. ДСанПіН 3.3.2.007-98 (затверджено Постановою Головного державного санітарного лікаря України від 10.12.1998 р. № 7).
13. Державні санітарні норми та правила "Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу" ЗАТВЕРДЖЕНО Наказ Міністерства охорони здоров'я України 08 квітня 2014 року N 248
14. Типові норми належності вогнегасників (затверджено наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 2 квітня 2004 р. N 151).
15. Норми визначення категорій приміщень, будинків та зовнішніх установок за вибухопожежною та пожежною безпекою – Режим доступу: http://3umf.com/doc/5516/ .

ДОДАТОК А

Матриця з’єднань для Прикладу №2



ДОДАТОК Б

Лістинг коду

Diffusion.cs-Функція дифузії

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    class Diffusion
    {
        static public double bigestElement = 0;
        static public bool solution = true;

        public static void run(List<diffusionObjects> objectsList,Point kp)
        {

            objectsList = objectsList.OrderBy(diffusionObjects =>
diffusionObjects.bigestSide).ToList();


               if (bigestElement < objectsList.Last().bigestSide) bigestElement =
objectsList.Last().bigestSide;
            for (int i = 0; i < objectsList.Count; i++)
            {
                Console.WriteLine(objectsList.ElementAt(i).name +" "+
objectsList.ElementAt(i).bigestSide);
            }

                for (int i = objectsList.Count - 1; i >= 0; i--)
                {

                    while (moving(objectsList, i, kp) != 0)
                    {
                        if (solution == false) break;
                    }
                    if (solution == false) break;
                }

        }
        public static int moving(List<diffusionObjects> objectsList,int i, Point kp)
        {

            if (objectsList.ElementAt(i).sizes.x > BinsList.list.Last().sizes.x ||
objectsList.ElementAt(i).sizes.y > BinsList.list.Last().sizes.y)
            {
                if (objectsList.ElementAt(i).sizes.x > BinsList.list.Last().sizes.x)
                {

                    Optimization.xSplit--;
```

```
                                }
                if (objectsList.ElementAt(i).sizes.y > BinsList.list.Last().sizes.y)
                {

                    Optimization.ySplit--;

                }
                Optimization.split(kp);
                return 1;

            }
            placing(objectsList, i, 0);
            int l = 0;
            Diffusion.densityMeasuring(objectsList);
            if (objectsList.First().bigestSide > BinsList.list.First().sizes.x ||
objectsList.First().bigestSide > BinsList.list.First().sizes.y)
            {
                Diffusion.solution = false;

            }
            if ((objectsList.Last().bigestSide < kp.x || objectsList.Last().bigestSide <
kp.y)&& Diffusion.solution!= false)
            {
                while (BinsList.list.Count - usedBins() < objectsList.Count -
fixedElments(objectsList))
                {
                    l=l+1;

                    Optimization.run(objectsList.Count + l, kp);
                    Diffusion.densityMeasuring(objectsList);

if(objectsList.First().bigestSide>BinsList.list.First().sizes.x||objectsList.First().bigestS
ide>BinsList.list.First().sizes.y) {
                        Diffusion.solution = false;
                        break;
                    }

                }
            }            else Diffusion.solution = false;


            return 0;

        }
        public static void placing(List<diffusionObjects> objectsList,int i,int j)
        {

            densityMeasuring(objectsList);

            if (BinsList.list.ElementAt(j).objectsList.Exists(diffusionObjects =>
diffusionObjects.fix == true))
            {
                try
                {
                    if (j + 1 < BinsList.list.Count)
                    {

                        placing(objectsList, i, j+1);
                    }
                    else
                    {
                        Console.WriteLine("Can't find solution");
                       // Console.Read();
                        solution = false;
```

```
                    //  Environment.Exit(0);
                }

            }
            catch (OverflowException e)
            {
                Console.WriteLine("stack");
            }

        }
        else
        {
            objectsList.ElementAt(i).center = BinsList.list.ElementAt(j).center;
            objectsList.ElementAt(i).refreshCoordinates(1);
            objectsList.ElementAt(i).fix = true;

            Console.WriteLine("Element " + objectsList.ElementAt(i).name + " become
fixed");
            densityMeasuring(objectsList);

        }
    }
    private static int usedBins()
    {
        int usedB = 0;
        for (int i = 0; i < BinsList.list.Count; i++)
        {
            for (int j = 0; j < BinsList.list.ElementAt(i).objectsList.Count; j++)
            {
                if
(BinsList.list.ElementAt(i).objectsList.ElementAt(j).fix.Equals(true))
                {
                    usedB++;
                }
            }
        }
        return usedB;
    }

    private static int fixedElments(List<diffusionObjects> eList)
    {
        int fixE = 0;
        for (int i = 0; i < eList.Count; i++)
        {
            if (eList.ElementAt(i).fix == true) fixE++;

        }
        return fixE;
    }
    public static void densityMeasuring(List<diffusionObjects> objectsList)
    {
        for (int i = 0; i < BinsList.list.Count; i++)
        {
            BinsList.list.ElementAt(i).cleaning();
        }


        for (int i = 0; i < BinsList.list.Count; i++)
        {
            for (int j = 0; j < objectsList.Count; j++)
            {
                if (((objectsList.ElementAt(j).top.x >=
BinsList.list.ElementAt(i).top.x) && (objectsList.ElementAt(j).top.x <=
```

```
BinsList.list.ElementAt(i).bottom.x)) && ((objectsList.ElementAt(j).top.y >=
BinsList.list.ElementAt(i).top.y) && (objectsList.ElementAt(j).top.y <=
BinsList.list.ElementAt(i).bottom.y)) && (objectsList.ElementAt(j).top.x !=
BinsList.list.ElementAt(i).bottom.x) && (objectsList.ElementAt(j).top.y !=
BinsList.list.ElementAt(i).bottom.y))
                    {

                        length(j, i, objectsList);

                        width(j, i, objectsList);

                    }
                }

            }
        }

        static private void length(int j, int i, List<diffusionObjects> objectsList)
        {
            BinsList.list.ElementAt(i).objectsList.Add(objectsList.ElementAt(j));

            if ((objectsList.ElementAt(j).bottom.x) > (BinsList.list.ElementAt(i).bottom.x))
            {

                if ((BinsList.list.Count) > i + 1)
                {
                    if (!BinsList.list.ElementAt(i +
1).objectsList.Contains(objectsList.ElementAt(j)))
                    {

                        length(j, i + 1, objectsList);

                        width(j, i + 1, objectsList);
                    }
                }
            }
        }


        static private void width(int j, int i, List<diffusionObjects> objectsList)
        {



            if ((objectsList.ElementAt(j).bottom.y) > (BinsList.list.ElementAt(i).bottom.y))
            {

                if ((BinsList.list.Count) > i + Optimization.xSplit)
                {
                    if (!BinsList.list.ElementAt(i +
Optimization.xSplit).objectsList.Contains(objectsList.ElementAt(j)))
                    {

                        length(j, i + Optimization.xSplit, objectsList);
                        width(j, i + Optimization.xSplit, objectsList);
                    }
                }
            }

        }
    }
}
```

Optimization.cs-Функція мінімізації довжин з’єднань

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    static public class Optimization
    {
       static  public int xSplit=1, ySplit = 1;
       public static List<Star> starList = new List<Star>();
       public static List<Star> finalStarList = new List<Star>();
       private static List<Star> connectionsList = new List<Star>();

         public static void run(int count,Point _kp)
        {

            xSplit = 1;
            ySplit = 1;
            double x,y;
            x = _kp.x;
            y = _kp.y;
            while ((xSplit* ySplit) < count )
            {
                if (x > y)
                {
                    xSplit++;
                    x = x / xSplit;

                }
                else
                {
                     ySplit++;
                    y = y / ySplit;
                }

            }

                split(_kp);


        }


         public static void starsGenaration()
         {

             int tmp = starList.Count;
             for (int i = 0; i < tmp; i++)
             {
                 starList.Remove(starList.ElementAt(0));

             }
             int name = 0;
             for (int i = 0; i < ElementsList.list.Count; i++)
             {
                 connectionsList.Add(new Star(1));
                 connectionsList.Last().eList.Add(
ElementsList.list.ElementAt(ElementsList.list.ElementAt(i).name));
                 for(int j=0; j< ElementsList.list.ElementAt(i).conectionList.Count();j++)
                 {
                     connectionsList.Last().eList.Add(
ElementsList.list.ElementAt(ElementsList.list.ElementAt(i).conectionList[j]));
```

```
                }

            }

            connectionsList = connectionsList.OrderBy(Star => Star.eList.Count).ToList();
            starList.Add(connectionsList.Last());
            starList.Last().name = starList.Count - 1;

            for (int i = 0; i < connectionsList.Count; i++)
            {
                Console.WriteLine();
                for (int j = 0; j < connectionsList.ElementAt(i).eList.Count; j++)
                {
                    Console.Write(connectionsList.ElementAt(i).eList.ElementAt(j).name+"
");
                }
            }
            Console.WriteLine();


            for( int k = connectionsList.Count - 2;k>=0;k--)
            {
                starsCompetition(k);


            }

            for (int i = 0; i < starList.Count; i++)
            {
                finalStarList.Add(new Star(starList.ElementAt(i).name));
                finalStarList.Last().eList = new List<Element>();
                for (int j = 0; j < starList.ElementAt(i).eList.Count(); j++)
                {

finalStarList.Last().eList.Add(starList.ElementAt(i).eList.ElementAt(j));
                }
            }

        }
        private static  void starsCompetition(int k)
        {
            Star tmp = new Star(1);
            for (int i = 0; i < starList.Count; i++)
            {
                for (int j = 0; j < starList.ElementAt(i).eList.Count; j++)
                {
                    tmp.eList.Add(starList.ElementAt(i).eList.ElementAt(j));
                }

            }


            for (int i = 0; i < tmp.eList.Count; i++)
            {
                Console.Write(tmp.eList.ElementAt(i).name);
            }

            List<Star> tmpStarList = new List<Star>();
            for (int i = k; i >= 0; i--)
            {
                if (connectionsList.ElementAt(i).eList.Count ==
connectionsList.ElementAt(k).eList.Count) tmpStarList.Add(connectionsList.ElementAt(i));
                connectionsList.ElementAt(i).newNets = 0;
            }
```

```
            int maxNets=0;

              for (int i = 0; i < tmpStarList.Count; i++)
              {

                  for (int j = 0; j < tmpStarList.ElementAt(i).eList.Count; j++)
                  {
                      if (!tmp.eList.Exists(Element => Element.name ==
tmpStarList.ElementAt(i).eList.ElementAt(j).name)) tmpStarList.ElementAt(i).newNets++;
                  }
                  if(tmpStarList.ElementAt(i).newNets>maxNets) maxNets=
tmpStarList.ElementAt(i).newNets;
              }

              //tmpStarList = tmpStarList.OrderBy(Star => Star.newNets).ToList();
              if (tmpStarList.First().newNets != 0 && tmpStarList.First().newNets==maxNets)
starList.Add(tmpStarList.First());
              starList.Last().name = starList.Count-1;

              Console.WriteLine();
              for (int i = 0; i < tmpStarList.Count; i++)
              {
                  Console.WriteLine("new Nets "+ tmpStarList.ElementAt(i).newNets );
                  for (int j = 0; j < tmpStarList.ElementAt(i).eList.Count; j++)
                  {
                      Console.Write(tmpStarList.ElementAt(i).eList.ElementAt(j).name+ " ");
                  }
                  Console.WriteLine();
              }

          }

          public static int cheaking(List<diffusionObjects> objectsList, Point kp)
         {
             for (int i = 0; i < objectsList.Count; i++)
             {
                 if(objectsList.ElementAt(i).sizes.x>kp.x||
objectsList.ElementAt(i).sizes.y>kp.y)
                 {
                     Console.WriteLine("Parameters of KP should be increased ");
                     return 1;
                 }
             }
                 return 0;
         }

          public static void split(Point _kp)
          {


              int tmp = BinsList.list.Count;
              for (int i = 0; i < tmp; i++)
              {
                  BinsList.list.Remove(BinsList.list.ElementAt(0));

              }
              for (int i = 0; i < (xSplit * ySplit); i++)
              {
                  BinsList.list.Add(new Bin(new Point(0, 0), new Point(0, 0)));
              }
              int j = 0;
              for (int i = 0; i < BinsList.list.Count; i++)
              {
```

```
                BinsList.list.ElementAt(i).top.x = (_kp.x / (xSplit)) * j;
                BinsList.list.ElementAt(i).bottom.x = (_kp.x / (xSplit) * j) + _kp.x /
(xSplit);
                j++;
                if (j == (xSplit)) j = 0;

            }
            j = 0;
            for (int i = 0; i < BinsList.list.Count; i++)
            {
                if ((i % (xSplit) == 0) && (i != 0)) j++;
                BinsList.list.ElementAt(i).top.y = (_kp.y / (ySplit)) * j;
                BinsList.list.ElementAt(i).bottom.y = (_kp.y / (ySplit) * j) + _kp.y /
(ySplit);
                BinsList.list.ElementAt(i).refreshCoordinates();
            }

           // Program.form();

        }

    }
}
```

Tracing.cs -Функція трасування

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    class Tracing
    {
        public static List<List<Point>> netsList = new List<List<Point>>();


        public static Point newTrace = new Point(0,0);
        public static void bins()
        {

            Console.WriteLine("Calculating...");
            Optimization.xSplit = (int)(Program.finalKP.x / Program.traceWidth);
            Optimization.ySplit = (int)(Program.finalKP.y / Program.traceWidth);
            Optimization.split(Program.finalKP);
            newTrace.x = Program.finalKP.x / Optimization.xSplit;
            newTrace.y = Program.finalKP.y / Optimization.ySplit;
            List<diffusionObjects> tmpList = new List<diffusionObjects>();
            for (int i = 0; i < ElementsList.list.Count;i++ )
            {
                tmpList.Add(ElementsList.list.ElementAt(i));
            }
            Diffusion.densityMeasuring(tmpList);




        }
        private static void statusCleaning()
        {
            for (int i = 0; i < BinsList.list.Count; i++)
            {
```

```
                BinsList.list.ElementAt(i).status = -1;
            }
        }

        public static void shortcut(int i,int j)
        {
           // Program.form();
            Console.WriteLine("tracing... ");

ElementsList.list.ElementAt(j).conectionList.Remove(ElementsList.list.ElementAt(j).conection
List.Find(Element=> Element==ElementsList.list.ElementAt(i).name));



            statusCleaning();
            double[,] distancesMatrix = new
double[ElementsList.list.ElementAt(i).contactsCoordinates.Count(),ElementsList.list.ElementA
t(j).contactsCoordinates.Count()];
            double shortest;
            int sl = 0, sk = 0;
            shortest =
Math.Abs(ElementsList.list.ElementAt(i).contactsCoordinates.ElementAt(0).x -
ElementsList.list.ElementAt(j).contactsCoordinates.ElementAt(0).x) +
Math.Abs(ElementsList.list.ElementAt(i).contactsCoordinates.ElementAt(0).y -
ElementsList.list.ElementAt(j).contactsCoordinates.ElementAt(0).y);
            for (int k = 0; k < ElementsList.list.ElementAt(i).contactsCoordinates.Count();
k++)
            {
                for (int l=0; l <
ElementsList.list.ElementAt(j).contactsCoordinates.Count(); l++)
                {
                    distancesMatrix[k,l] =
Math.Abs(ElementsList.list.ElementAt(i).contactsCoordinates.ElementAt(k).x -
ElementsList.list.ElementAt(j).contactsCoordinates.ElementAt(l).x) +
Math.Abs(ElementsList.list.ElementAt(i).contactsCoordinates.ElementAt(k).y -
ElementsList.list.ElementAt(j).contactsCoordinates.ElementAt(l).y);
                    if((distancesMatrix[k,l])< shortest)
                    {


                            shortest = distancesMatrix[k, l];
                            sl = l;

                            sk = k;




                    }
                }
            }

            netsList.Add(new List<Point>());


           //
Console.WriteLine((int)(ElementsList.list.ElementAt(i).contactsCoordinates.ElementAt(sk).y /
newTrace.y) *(Optimization.xSplit ) +
(int)(ElementsList.list.ElementAt(i).contactsCoordinates.ElementAt(sk).x / newTrace.x) + 1);

BinsList.list.ElementAt(getBinsNumber(ElementsList.list.ElementAt(i).contactsCoordinates.Ele
mentAt(sk))).status=0;
```

```
          //
BinsList.list.ElementAt(getBinsNumber(ElementsList.list.ElementAt(j).contactsCoordinates.Ele
mentAt(sl))).status = -2;

wave(BinsList.list.ElementAt(getBinsNumber(ElementsList.list.ElementAt(i).contactsCoordinate
s.ElementAt(sk))),
getBinsNumber(ElementsList.list.ElementAt(j).contactsCoordinates.ElementAt(sl)));

track(getBinsNumber(ElementsList.list.ElementAt(j).contactsCoordinates.ElementAt(sl)),getBin
sNumber(ElementsList.list.ElementAt(i).contactsCoordinates.ElementAt(sk)));

ElementsList.list.ElementAt(i).contactsCoordinates.Remove(ElementsList.list.ElementAt(i).con
tactsCoordinates.ElementAt(sk));

ElementsList.list.ElementAt(j).contactsCoordinates.Remove(ElementsList.list.ElementAt(j).con
tactsCoordinates.ElementAt(sl));




        }
        public static void track(int firstPoint, int destination)
        {

             int lastPoint=firstPoint;

             netsList.Last().Add(BinsList.list.ElementAt(firstPoint).center);
            //lastPoint!=destination
            // Program.form();

             while (lastPoint != destination)

            {



                if (BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x + newTrace.x,
BinsList.list.ElementAt(lastPoint).center.y))).status ==
BinsList.list.ElementAt(lastPoint).status - 1 && BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x + newTrace.x,
BinsList.list.ElementAt(lastPoint).center.y))).x!=true)
                {
                    BinsList.list.ElementAt(lastPoint).x = true;

                    lastPoint = getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x + newTrace.x,
BinsList.list.ElementAt(lastPoint).center.y));
                    netsList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x ,
BinsList.list.ElementAt(lastPoint).center.y))).center);
                    BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y))).x = true;


                    // Program.form();
                    continue;


                }

                else
```

```
                        if (BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y - newTrace.y))).status ==
BinsList.list.ElementAt(lastPoint).status - 1 && BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y - newTrace.y))).y!=true)
                {
                    BinsList.list.ElementAt(lastPoint).y = true;

                    lastPoint = getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x ,
BinsList.list.ElementAt(lastPoint).center.y-newTrace.y));
                    netsList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y))).center);
                  BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y))).y = true;


                    //   Program.form();
                continue;

                }
                        else
                            if (BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x - newTrace.x,
BinsList.list.ElementAt(lastPoint).center.y))).status ==
BinsList.list.ElementAt(lastPoint).status - 1 && BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x - newTrace.x,
BinsList.list.ElementAt(lastPoint).center.y))).x != true)
                            {
                                BinsList.list.ElementAt(lastPoint).x = true;

                                lastPoint = getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x - newTrace.x,
BinsList.list.ElementAt(lastPoint).center.y));

netsList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y))).center);
                                BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y))).x = true;


                                //   Program.form();
                                continue;

                            }
                else
                            if (BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y + newTrace.y))).status ==
BinsList.list.ElementAt(lastPoint).status - 1 && BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y + newTrace.y))).y!=true)
                {
                    BinsList.list.ElementAt(lastPoint).y = true;

                    lastPoint = getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y + newTrace.y));
```

```
                    netsList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y))).center);
                  BinsList.list.ElementAt(getBinsNumber(new
Point(BinsList.list.ElementAt(lastPoint).center.x,
BinsList.list.ElementAt(lastPoint).center.y))).y = true;

                //  Program.form();
                  continue;

              }
              else
              {
                  Console.WriteLine("Tracing: Can't find solution");
                  Console.Read();
                  Environment.Exit(0);
              }

          }

      }
       public static  void  wave(Bin nullBin,int destination)
       {


           int d = 1;
           List<List<Bin>> wavesList = new List<List<Bin>>();
           wavesList.Add(new List<Bin>());
           wavesList.Last().Add(nullBin);
           int count = 0;
           //
           while (BinsList.list.ElementAt(destination).status < 0)
           {
               wavesList.Add(new List<Bin>());

               for (int i = 0; i < wavesList.ElementAt(wavesList.Count - 2).Count; i++)
               {

                   if (wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x +
newTrace.x < Program.finalKP.x
                       && (BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x + newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).objectsList.Count == 0
                       ||getBinsNumber(new Point(wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.x + newTrace.x, wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.y))==destination)
                       && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x + newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).status == -1
                       && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x + newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).x==false)
                   {

                       BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x + newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).status = d;
                       wavesList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x + newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))));
                   }
                   if (wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x -
newTrace.x > 0
```

```
                        &&(BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x - newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).objectsList.Count == 0
                        ||getBinsNumber(new Point(wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.x - newTrace.x, wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.y))==destination)
                        && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x - newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).status==-1
                        && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x - newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).x==false)
                    {

                        BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x - newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))).status = d;
                        wavesList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x - newTrace.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y))));
                    }
                    if (wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.y+newTrace.y < Program.finalKP.y
                        &&( BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y +
newTrace.y))).objectsList.Count == 0
                        ||getBinsNumber(new Point(wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.x, wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y +
newTrace.y))==destination)
                        && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y + newTrace.y))).status == -1
                         && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y + newTrace.y))).y==false)
                    {

                        BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y + newTrace.y))).status = d;
                        wavesList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y + newTrace.y))));
                    }
                    if (wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y-
newTrace.y > 0
                        &&  (wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.y-newTrace.y <Program.finalKP.y)
                        && (BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y -
newTrace.y))).objectsList.Count == 0
                        || getBinsNumber(new Point(wavesList.ElementAt(wavesList.Count -
2).ElementAt(i).center.x, wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y -
newTrace.y)) ==destination)
                        && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y - newTrace.y))).status == -1
                        && BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y - newTrace.y))).y==false)
                    {
```

```
                        BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y - newTrace.y))).status = d;
                        wavesList.Last().Add(BinsList.list.ElementAt(getBinsNumber(new
Point(wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.x,
wavesList.ElementAt(wavesList.Count - 2).ElementAt(i).center.y - newTrace.y))));
                     }
                    // Program.form();

                }
                d++;

                //Program.form();
                Console.WriteLine("+");

            }

         }
        private static int  getBinsNumber(Point point)
        {
            return (int)(point.y / newTrace.y) * (Optimization.xSplit) + (int)(point.x /
newTrace.x);
        }

    }
}
```

Клас BinsList.cp – клас,що описує набір контейнерів

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    public class BinsList
    {
        public static List<Bin> list = new List<Bin>();
    }
}
```

Клас ElementList.cs – описує список елементів

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    public class ElementsList
    {
       public static  List<Element> list = new List<Element>();
       public static double elemrntsSquare = 0;
       public static double indent = 1;
        public static double targetDensity=0.2;
      private static  void squareCount()
       {
           elemrntsSquare = 0;
           for (int i = 0; i < ElementsList.list.Count; i++)
           {
```

```
                elemrntsSquare+= ElementsList.list.ElementAt(i).sizes.x *
ElementsList.list.ElementAt(i).sizes.y;
            }
        }
        public static void bubble(bool on)
        {


           if(on ==true)
           {
               squareCount();
              indent = ((elemrntsSquare * targetDensity) / (ElementsList.list.Count))/ 4;

                for (int i = 0; i < ElementsList.list.Count; i++)
            {
                ElementsList.list.ElementAt(i).center = new
Point(ElementsList.list.ElementAt(i).center.x + indent,
ElementsList.list.ElementAt(i).center.y + indent);
                ElementsList.list.ElementAt(i).bottom = new
Point(ElementsList.list.ElementAt(i).bottom.x + indent * 2,
ElementsList.list.ElementAt(i).bottom.y + indent * 2);
                ElementsList.list.ElementAt(i).sizes = new
Point(ElementsList.list.ElementAt(i).sizes.x + indent * 2,
ElementsList.list.ElementAt(i).sizes.y + indent * 2);
            }
           }
           else
           {
               for (int i = 0; i < ElementsList.list.Count; i++)
               {
                   ElementsList.list.ElementAt(i).center = new
Point(ElementsList.list.ElementAt(i).center.x - indent,
ElementsList.list.ElementAt(i).center.y - indent);
                   ElementsList.list.ElementAt(i).bottom = new
Point(ElementsList.list.ElementAt(i).bottom.x - indent * 2,
ElementsList.list.ElementAt(i).bottom.y - indent * 2);
                   ElementsList.list.ElementAt(i).sizes = new
Point(ElementsList.list.ElementAt(i).sizes.x - indent * 2,
ElementsList.list.ElementAt(i).sizes.y - indent * 2);
               }
           }


        }
        public static void cleaning(List<diffusionObjects> objectsList)
        {
            for (int i = 0; i < objectsList.Count; i++)
            {
                objectsList.ElementAt(i).top = new Point(0, 0);
                objectsList.ElementAt(i).bottom = new Point(objectsList.ElementAt(i).sizes.x,
objectsList.ElementAt(i).sizes.y);
                objectsList.ElementAt(i).center = new Point(objectsList.ElementAt(i).sizes.x
/ 2, objectsList.ElementAt(i).sizes.y / 2);
                objectsList.ElementAt(i).fix = false;
            }
        }

    }
}
```

Клас Star.cs- описує «зірку»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Placer2._0
{
    public class Star: diffusionObjects
    {
        public List<Element> eList;
        public int newNets=0;

        public Star(int name): base (name)
        {

             eList = new List<Element>();
        }
        public void HPWL(Point kp)
        {
            this.center = new Point(kp.x / 2, kp.y / 2);
            this.sizes = new Point(kp.x, kp.y);
            if (this.sizes.x > this.sizes.y) this.bigestSide = this.sizes.x; else
this.bigestSide = this.sizes.y;

        }


    }
}
```

Клас Bin.cs – описує одиничний контейнер

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    public class Bin: diffusionObjects
    {
        public int status=-1;
        public bool x = false;
        public bool y = false;
        public List<diffusionObjects> objectsList = new List<diffusionObjects>();
        public Bin(Point top, Point bottom): base ( top, bottom)
        {

        }

        public void refreshCoordinates()
        {
            sizes.y = bottom.y - top.y;
            sizes.x = bottom.x - top.x;
            center = new Point(top.x + sizes.x / 2, top.y + sizes.y / 2);
        }

        public void cleaning()
        {

            int tmp = objectsList.Count;
```

```
            for (int i = 0; i < tmp; i++)
            {
                objectsList.Remove(objectsList.ElementAt(0));
            }

        }
    }
}
```

Клас Element.cs –описує елемент

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    public class Element: diffusionObjects
    {
        public List<int> conectionList;
        public  double  delta = 0.3;
        public double step;
        public Point relativeCoordinates = new Point(0, 0);
        public int starName;
        public List<Point> contactsCoordinates = new List<Point>();

       public  Element(Point sizes, int[] conectionList, int name): base (sizes, name)
        {
            this.conectionList = conectionList.ToList();
            this.step = delta;
        }
       public void setRelativeCoordinates(Point kp, int starName)
       {
           relativeCoordinates = new Point(center.x-kp.x/2,center.y-kp.y/2);
           this.starName = starName;
       }
       public void setContCoordinates()
       {

           for(int i=0;i<conectionList.Count(); i++)
           {
               delta = delta + step * 2;
               contactsCoordinates.Add(new Point(top.x+delta,top.y));

           }
           delta = step;
           for (int i = 0; i < conectionList.Count(); i++)
           {
               delta = delta + step * 2;
               contactsCoordinates.Add(new Point(bottom.x - delta, bottom.y));

           }
           delta = step;
           for (int i = 0; i < conectionList.Count(); i++)
           {
               delta = delta + step * 2;
               contactsCoordinates.Add(new Point(bottom.x, bottom.y - delta));
           }
           delta = step;
           for (int i = 0; i < conectionList.Count(); i++)
           {
               delta = delta + step * 2;
               contactsCoordinates.Add(new Point(top.x, top.y + delta));
```

```
            }

        }


    }
}
```

Клас DiffusionObjects.cs – абстрактний клас,що описуэ об’єкти,здатні до дифузії(зірки та елементи)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Placer2._0
{
    public class diffusionObjects
    {
        public Point top=new Point(0,0);
        public Point bottom = new Point(0, 0);
        public Point center = new Point(0, 0);
        public Point sizes = new Point(0, 0);
        public int name;
        public double bigestSide;
        public bool fix = false;
        //Element
        public diffusionObjects(Point sizes, int name)
        {
            this.sizes = sizes;
            this.name = name;
            this.top = new Point(0, 0);
            this.bottom = new Point(sizes.x, sizes.y);
            this.center = new Point(sizes.x / 2, sizes.y / 2);
            if (sizes.x > sizes.y) bigestSide = sizes.x; else bigestSide = sizes.y;
        }
        public diffusionObjects(Point top, Point bottom)
        {
            this.top = top;
            this.bottom= bottom;
        }
        public diffusionObjects(int name)
        {
            this.name=name;
        }
        public void refreshCoordinates(int i)
        {
            top = new Point(center.x - sizes.x / 2, center.y - sizes.y / 2);
            bottom = new Point(center.x + sizes.x / 2, center.y + sizes.y / 2);

        }

    }
}
```

Клас form1.cs- windows-form для візуалізації отриманих даних або покрокового перегляду роботи програми

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Placer2._0
{
    public partial class Form1 : Form
    {
        Color color;
         double scale = 0;
        public Form1()
        {
            InitializeComponent();
            if (Program.kp.x > Program.kp.y) scale = 600 / Program.kp.x;
            else scale = 600 / Program.kp.y;
            Icon icon = Placer2._0.Properties.Resources.ico;
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void draw(Point _top, Point _sizes,Color color, PaintEventArgs e, bool fill)
        {
            Point top = new Point(((int)(_top.x*scale)),((int)(_top.y*scale)));
            Point sizes = new Point(((int)(_sizes.x * scale)), ((int)(_sizes.y * scale)));
            Rectangle rect = new Rectangle((int)top.x,(int)top.y,(int)sizes.x,(int)sizes.y);
            e.Graphics.DrawRectangle(new Pen(color,2),rect);
            if (fill == true) e.Graphics.FillRectangle(new System.Drawing.SolidBrush(color),
rect);
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {

            Image image = Placer2._0.Properties.Resources.anchor;

            draw(new Point(0, 0), Program.kp, Color.LightGray, e, true);
            draw(new Point(0, 0), Program.kp, Color.LightSteelBlue, e, false);
            draw(new Point(0, 0), Program.finalKP, Color.DarkGray, e, true);

            if (Program.currentStar == -3)
            {
                for (int i = Optimization.starList.Count - 1; i >= 0; i--)
                {

                    for (int eNum = 0; eNum <
Optimization.starList.ElementAt(i).eList.Count; eNum++)
                    {
                        draw(Optimization.starList.ElementAt(i).eList.ElementAt(eNum).top,
Optimization.starList.ElementAt(i).eList.ElementAt(eNum).sizes, Color.LightSeaGreen, e,
true);
                        draw(Optimization.starList.ElementAt(i).eList.ElementAt(eNum).top,
Optimization.starList.ElementAt(i).eList.ElementAt(eNum).sizes, Color.FromArgb(10, 10, 60),
e, false);

                    }


                }
```

```
                for (int i = Optimization.starList.Count - 1; i >= 0; i--)
                {

                    for (int eNum = 0; eNum <
Optimization.starList.ElementAt(i).eList.Count; eNum++)
                    {

                        //

                Color color= Color.Violet;

                for (int i = 0; i < Tracing.netsList.Count(); i++)
                {
                    for (int j = 0; j < Tracing.netsList.ElementAt(i).Count-1 ; j++)
                    {

                        e.Graphics.DrawLine(new Pen(new SolidBrush(Color.Gold), 1), new
PointF((int)(Tracing.netsList.ElementAt(i).ElementAt(j).x * scale),
(int)(Tracing.netsList.ElementAt(i).ElementAt(j).y * scale)), new
PointF((int)(Tracing.netsList.ElementAt(i).ElementAt(j + 1).x * scale),
(int)(Tracing.netsList.ElementAt(i).ElementAt(j + 1).y * scale)));


            if (Program.currentStar == -1)
            {
                for (int i = Optimization.starList.Count - 1; i >= 0; i--)
                {

                    for (int eNum = 0; eNum <
Optimization.starList.ElementAt(i).eList.Count; eNum++)
                    {
                        draw(Optimization.starList.ElementAt(i).eList.ElementAt(eNum).top,
Optimization.starList.ElementAt(i).eList.ElementAt(eNum).sizes, Color.LightSeaGreen, e,
true);
                        draw(Optimization.starList.ElementAt(i).eList.ElementAt(eNum).top,
Optimization.starList.ElementAt(i).eList.ElementAt(eNum).sizes, Color.FromArgb(10, 10, 60),
e, false);
                        e.Graphics.FillEllipse(new SolidBrush(Color.DarkSlateBlue),
(int)((Optimization.starList.ElementAt(i).eList.ElementAt(eNum).center.x) * scale) -
(int)(scale / 4), (int)((Optimization.starList.ElementAt(i).eList.ElementAt(eNum).center.y)
* scale) - (int)(scale / 4), (int)scale / 2, (int)scale / 2);

                    }


                }

                for (int i = 0; i < Optimization.starList.Count; i++)
                {
                    for (int j = 0; j < Optimization.starList.ElementAt(i).eList.Count; j++)
                    {
                        e.Graphics.DrawLine(new
Pen(Optimization.starList.ElementAt(i).color, Convert.ToInt16(2)),
(int)(Optimization.starList.ElementAt(i).center.x * scale),
(int)(Optimization.starList.ElementAt(i).center.y * scale),
(int)(Optimization.starList.ElementAt(i).eList.ElementAt(j).center.x * scale),
(int)(Optimization.starList.ElementAt(i).eList.ElementAt(j).center.y * scale));

                    }
                    e.Graphics.FillEllipse(new SolidBrush(Color.White),
(int)((Optimization.starList.ElementAt(i).center.x) * scale) - (int)scale / 10,
```

```
(int)((Optimization.starList.ElementAt(i).center.y) * scale) - (int)scale / 10, (int)scale /
5, (int)scale / 5);
                    e.Graphics.DrawImage(image, new
PointF(((int)(Optimization.starList.ElementAt(i).center.x * scale) - 19),
((int)(Optimization.starList.ElementAt(i).center.y * scale) - 7)));
                }

                for (int i = Optimization.starList.Count - 1; i >= 0; i--)
                {

                    for (int eNum = 0; eNum <
Optimization.starList.ElementAt(i).eList.Count; eNum++)
                    {


e.Graphics.DrawString(Convert.ToString(Optimization.starList.ElementAt(i).eList.ElementAt(eN
um).name), new Font("s", (int)(10)), new SolidBrush(Color.White), new
PointF(((int)((Optimization.starList.ElementAt(i).eList.ElementAt(eNum).bottom.x * scale) -
20)), (int)((Optimization.starList.ElementAt(i).eList.ElementAt(eNum).top.y * scale))));

                    }

                }

            }
            else if (Program.currentStar != -2 && Program.currentStar != -3)
            {

                    for (int eNum = 0; eNum <
Optimization.starList.ElementAt(Program.currentStar).eList.Count; eNum++)
                    {
                        if
(Optimization.starList.ElementAt(Program.currentStar).eList.ElementAt(eNum).fix!=false)
draw(Optimization.starList.ElementAt(Program.currentStar).eList.ElementAt(eNum).top,
Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(eNum).sizes,
Color.LightSeaGreen, e, true);
                        if
(Optimization.starList.ElementAt(Program.currentStar).eList.ElementAt(eNum).fix != false)
draw(Optimization.starList.ElementAt(Program.currentStar).eList.ElementAt(eNum).top,
Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(eNum).sizes,
Color.FromArgb(10, 10, 60), e, false);
                        if
(Optimization.starList.ElementAt(Program.currentStar).eList.ElementAt(eNum).fix != false)
e.Graphics.FillEllipse(new SolidBrush(Color.DarkSlateBlue),
(int)((Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(eNum).center.x
) * scale) - (int)(scale / 8),
(int)((Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(eNum).center.y
) * scale) - (int)(scale / 8), (int)scale / 4, (int)scale / 4);
                        //
                    }


                    if (Optimization.starList.ElementAt((Program.currentStar)).center.x != 0
&& Optimization.starList.ElementAt((Program.currentStar)).center.y != 0)
                    {
                        for (int j = 0; j <
Optimization.starList.ElementAt((Program.currentStar)).eList.Count; j++)
                    {
                        e.Graphics.DrawLine(new
Pen(Optimization.starList.ElementAt((Program.currentStar)).color, Convert.ToInt16(2)),
(int)(Optimization.starList.ElementAt((Program.currentStar)).center.x * scale),
(int)(Optimization.starList.ElementAt((Program.currentStar)).center.y * scale),
(int)(Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(j).center.x *
scale),
```

```
(int)(Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(j).center.y *
scale));

                    }
                    }


if(Optimization.starList.ElementAt((Program.currentStar)).center.x!=0&&Optimization.starList
.ElementAt((Program.currentStar)).center.y!=0)
                        e.Graphics.DrawImage(image, new
PointF(((int)(Optimization.starList.ElementAt((Program.currentStar)).center.x * scale) -
19), ((int)(Optimization.starList.ElementAt((Program.currentStar)).center.y * scale) - 7)));




                    for (int eNum = 0; eNum <
Optimization.starList.ElementAt((Program.currentStar)).eList.Count; eNum++)
                    {
                        if
(Optimization.starList.ElementAt(Program.currentStar).eList.ElementAt(eNum).fix != false)
e.Graphics.DrawString(Convert.ToString(Optimization.starList.ElementAt((Program.currentStar)
).eList.ElementAt(eNum).name), new Font("s", (int)(15)), new SolidBrush(Color.White), new
PointF(((int)((Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(eNum).
bottom.x * scale) -25)),
(int)((Optimization.starList.ElementAt((Program.currentStar)).eList.ElementAt(eNum).top.y *
scale))));

                    }

                for (int i = 0; i < BinsList.list.Count; i++)
                {
                    draw(BinsList.list.ElementAt(i).top, BinsList.list.ElementAt(i).sizes,
Color.DimGray, e, false);
                }
            }
            else if( Program.currentStar!=-3)
            {
                for (int eNum = 0; eNum < Optimization.starList.Count; eNum++)
                {
                    draw(Optimization.starList.ElementAt(eNum).top,
Optimization.starList.ElementAt(eNum).sizes, Color.LightGoldenrodYellow, e, true);
                    draw(Optimization.starList.ElementAt(eNum).top,
Optimization.starList.ElementAt(eNum).sizes, Color.FromArgb(10, 10, 60), e, false);

e.Graphics.DrawString(Convert.ToString(Optimization.starList.ElementAt(eNum).name), new
Font("s", (int)(20)), new SolidBrush(Color.FromArgb(30,30,30)), new
PointF((int)(Optimization.starList.ElementAt(eNum).bottom.x * scale-35),
(int)(Optimization.starList.ElementAt(eNum).top.y * scale)));
                    e.Graphics.FillEllipse(new SolidBrush(Color.White),
(int)((Optimization.starList.ElementAt(eNum).center.x) * scale) - (int)scale / 10,
(int)((Optimization.starList.ElementAt(eNum).center.y) * scale) - (int)scale / 10,
(int)scale / 5, (int)scale / 5);
                    e.Graphics.DrawImage(image, new
PointF(((int)(Optimization.starList.ElementAt(eNum).center.x * scale) - 19),
((int)(Optimization.starList.ElementAt(eNum).center.y * scale) - 7)));

                }
                for (int i = 0; i < BinsList.list.Count; i++)
                {
                    draw(BinsList.list.ElementAt(i).top, BinsList.list.ElementAt(i).sizes,
Color.DimGray, e, false);
                }
            }
```

```
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Enter) this.Close();
        }
    }
}
```

Form1.Designer.cs

```
namespace Placer2._0
{
    partial class Form1
    {
        private System.ComponentModel.IContainer components = null;
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code


        private void InitializeComponent()
        {
            System.ComponentModel.ComponentResourceManager resources = new
System.ComponentModel.ComponentResourceManager(typeof(Form1));
            this.SuspendLayout();
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(671, 649);
            this.Icon = ((System.Drawing.Icon)(resources.GetObject("$this.Icon")));
            this.Location = new System.Drawing.Point(300, 300);
            this.Name = "Form1";
            this.Text = "DPlace";
            this.Load += new System.EventHandler(this.Form1_Load);
            this.Paint += new System.Windows.Forms.PaintEventHandler(this.Form1_Paint);
            this.KeyDown += new System.Windows.Forms.KeyEventHandler(this.Form1_KeyDown);
            this.ResumeLayout(false);

        }

        #endregion
    }
                                                }
```