

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО»**

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

"На правах рукопису"
УДК 004.422.833

«До захисту допущено»
Завідувач кафедри

Коваль О.В.

(прізвище, ініціали)

_____ (підпис)

«_____» _____ 2018р.

Магістерська дисертація

зі спеціальності - 121 Інженерія програмного забезпечення
за спеціалізацією - Програмне забезпечення розподілених систем
на тему: «Використання генетичних алгоритмів для вирішення задачі балансування
складальної лінії»

Виконав: студент VI курсу, групи ТВ-71мп

Пругло Михайло Олексійович

(прізвище, ім'я, по батькові)

_____ (підпис)

Науковий керівник к.т.н. доцент Кублій Л.І.

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

_____ (підпис)

Рецензент _____

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

_____ (підпис)

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів без
відповідних посилань.

Студент _____

(підпис)

Київ - 2018

**Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”**

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти другий, магістерський

зі спеціальності - 121 Інженерія програмного забезпечення

за спеціалізацією - Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
Коваль О.В. _____
(прізвище, ініціали) (підпис)
« ____ » _____ 2018р.

**ЗАВДАННЯ
НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ СТУДЕНТУ**

Пруглу Михайлу Олексійовичу

(прізвище, ім'я, по батькові)

1. Тема дисертації _____ Використання генетичних алгоритмів для вирішення задачі балансування складальної лінії

Науковий керівник _____ Кублій Лариса Іванівна, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 5 листопада 2018 року №4072-с

2. Строк подання студентом дисертації 11 грудня 2018

3. Об'єкт дослідження складальні лінії та алгоритми їх балансування

4. Предмет дослідження алгоритми для оптимізації розподілу роботи складальної лінії

5. Перелік питань, які потрібно розробити проаналізувати існуюче програмне забезпечення балансування складальної лінії; проаналізувати можливі шляхи адаптації генетичних операторів до оптимізації розподілу навантаження; розробити програмний модуль балансування складальної лінії; розробити демонстраційний додаток та протестувати спроектований алгоритм

6. Перелік ілюстративного матеріалу мікросервісна система, засоби автоматичного масштабування, функції програмного забезпечення, структура програмного забезпечення, інтерфейс

7. Орієнтований перелік публікацій _____

1) Кублій Л.І., Пругло М.О. “Розв’язання задачі балансування складальної лінії з використанням генетичних алгоритмів ”

2) Кублій Л.І., Пругло М.О. “Використання генетичних алгоритмів для вирішення задачі балансування складальної лінії ”

8. Дата видачі завдання « 11 » вересня _____ 2017 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз проблеми балансування складальної лінії	11.09.2017-12.11.2017	
2	Аналіз існуючих реалізацій та шляхів вирішення проблеми	13.11.2017-15.01.2018	
3	Аналіз існуючих операторів генетичних алгоритмів	16.01.2018-05.03.2018	
4	Аналіз можливостей кодування генетичної інформації	06.03.2018-27.06.2018	
5	Модифікування евристик враховуючи граф технологічних операцій	28.06.2018-18.08.2018	
6	Моделювання схеми роботи майбутньої програми	19.08.2018-30.08.2018	
7	Розробка архітектури програмного забезпечення	01.09.2018-20.09.2018	
8	Розробка програмного модулю	21.09.2018-10.10.2018	
9	Розробка демонстраційного додатку	11.10.2018-30.11.2018	
10	Оформлення документації	01.12.2018-07.12.2018	

Студент

(підпис)

Пругло М.О.

(прізвище та ініціали)

Науковий керівник

(підпис)

Кублій Л.І.

(прізвище та ініціали)

РЕФЕРАТ

Магістерська дисертація складається зі вступу, п'яти розділів, висновку, переліку посилань з 52 найменувань, 3 додатків, і містить 17 рисунків, 22 таблиці. Повний обсяг магістерської дисертації складає 123 сторінок, з яких перелік посилань займає 5 сторінок, додатки – 18 сторінок.

Однією з тенденцій розвитку сучасної вітчизняної економіки є перехід до середньо- і дрібносерійного виробництва, тому ключовою проблемою стає підвищення гнучкості виробничої системи — зокрема, максимально можливе скорочення термінів підготовки виробництва й ефективна організація виробничого процесу. Це вимагає зміни форми подання результатів технологічного проектування, а також складних алгоритмів планування.

Дисертаційна робота магістра виконувалась у НТУУ «КПІ ім. Ігоря Сікорського» у відповідності з планом наукових досліджень кафедри АПЕПС.

Мета дослідження полягає в розробці нових програмних засобів і математичної моделі, які дозволять оптимізувати розподілення навантаження на складальній лінії.

Для досягнення поставленої задачі були сформульовані наступні завдання дослідження, що визначили логіку дослідження та його структуру:

- проаналізувати існуюче програмне забезпечення балансування складальної лінії;
- проаналізувати можливі шляхи адаптації генетичних операторів до оптимізації розподілу навантаження;
- розробити програмний модуль балансування складальної лінії;
- розробити демонстраційний додаток та протестувати спроектований алгоритм.

Об'єктом дослідження є складальні лінії та алгоритми їх балансування.

Предметом дослідження є алгоритми для оптимізації розподілу роботи складальної лінії.

Розв'язання поставлених задач виконувались засобами комп'ютерного аналізу, зокрема з використанням наступних методів:

– методи пошуку оптимального розв’язку для розв’язання задачі лінійного програмування.

Наукова новизна одержаних результатів. Найбільш суттєвими науковими результатами магістерської дисертації є:

- модифікована репрезентація в хромосомах генетичних алгоритмів;
- генетичні оператори кросовера, мутації, селекції та інверсії було модифіковано під задачі групування;
- написання специфічних евристик для збереження послідовності в графі технологічних операцій.

Практичне значення одержаних результатів роботи полягає в розробці програмного модулю, що дозволяє оптимізувати процес балансування складальних ліній, та спричинити велику економію ресурсів, тим самим збільшити конкурентоздатність Українських виробництв на ринку.

Основні результати атестаційної роботи обговорювалися на:

- V міжнародна науково-практична конференція «Сталий розвиток – XXI століття: управління, технології, моделі (наукові читання імені Ігоря Недіна)» (м. Київ, 23-24 жовтня 2018 року);
- XVI міжнародна науково-практична конференція аспірантів, магістрантів, студентів «Сучасні проблеми наукового забезпечення енергетики» (24-27 квітня 2018 року).

За матеріалами роботи опубліковано 2 наукові роботи.

Результати роботи впроваджено у ТОВ «НВП «Символ».

Ключові слова: *ОПТИМІЗАЦІЯ, ГЕНЕТИЧНІ АЛГОРИТМИ, БАЛАНСУВАННЯ СКЛАДАЛЬНОЇ ЛІНІЇ*

ЗМІСТ

Вступ.....	9
1. Задача балансування складальної лінії та системи її вирішення в контексті сучасного виробництва.....	11
1.1 Постановка задачі	11
1.2 Складальні лінії.....	13
1.3 Характеристики складальних ліній.....	15
1.4 Система OptiLine.....	19
1.5 Система Proplanner.....	20
1.6 Алгоритми поточних рішень.....	23
Висновки до першого розділу	28
2. Алгоритмічні основи використання генетичних алгоритмів для вирішення задачі балансування складальної лінії.....	29
2.1 Генетичні алгоритми.....	29
2.2 Опис реалізації програмного продукту.....	33
2.3 Альтернативні процеси.....	42
2.4 Синхронізована лінія передачі.....	45
2.5 Паралельні станції та завдання	45
2.6 Обмеження розподілення завдань за робочими станціями.....	48
Висновки до другого розділу.....	51
3. Аналіз та обґрунтування засобів реалізації	52
3.1 Мова C++	52
3.2 SFML.....	60
3.3 Visual Studio 2017.....	65
Висновки до третього розділу	72
4. Методика роботи користувача з системою.....	73
4.1 Демонстраційний додаток.....	73
4.2 Програмний модуль	75
Висновки до четвертого розділу	76
5. Розроблення стартап-проекту.....	77
5.1 Опис ідеї проекту.....	77
5.2 Технологічний аудит ідеї проекту.....	79
5.3 Аналіз ринкових можливостей запуску стартап-проекту	80
5.4 Аналіз ринкової стратегії проекту	90
5.5 Розроблення маркетингової програми стартап-проекту	93
Висновки до п'ятого розділу	96
Висновки.....	97

Список використаних джерел.....	99
Додаток А.....	104
Додаток Б.....	122

ВСТУП

У промисловому виробництві стандартизованих товарів, а також у виробництві нестандартних виробів використовують складальні лінії. Ці лінії містять робочі станції, розташовані вздовж стрічкового конвеєра або подібного механічного обладнання. Заготовки послідовно запускаються по лінії і переміщуються зі станції на станцію для обробки. Серед проблем, які виникають при керуванні складальними лініями, є задача балансування [1].

Задача балансування складальних ліній має велике промислове значення і є важливим завданням середньострокового планування виробництва. Вона полягає у визначенні робочим станціям у виробничій лінії операцій з заданого набору так, щоб:

- жодне з правил порядку збірки не порушувалося;
- жодна з робочих станцій в лінії не використовувала більше часу, ніж заздалегідь визначений час циклу виконання завдань, призначених цій станції;
- для виконання всіх завдань у наборі використовується якомога менше робочих станцій.

Враховуючи, що однією з тенденцій розвитку сучасної вітчизняної економіки є перехід до середньо- і дрібносерійного виробництва, ключовою проблемою стає підвищення гнучкості виробничої системи — зокрема, максимально можливе скорочення термінів підготовки виробництва й ефективна організація виробничого процесу. Це вимагає зміни форми подання результатів технологічного проектування, а також складних алгоритмів планування [2].

Задача балансування складальної лінії в загальному випадку потребує одне із можливих рішень, що відповідає початковим критеріям. Навіть у цьому випадку ця задача є NP-повною (клас NP-задач — це задачі, які можна розв'язати за поліноміальний час відносно обсягу вхідних даних за допомогою недетермінованого

алгоритму; NP-повна задача — задача з класу NP і всі задачі з цього класу можна звести до неї за поліноміальний час).

Пов'язана із нею задача оптимізації, що намагається знайти найбільш оптимальне рішення та використати найменшу кількість станцій, є NP-складною (тобто детермінований поліноміальний алгоритм її розв'язання можна використати для одержання детермінованого поліноміального алгоритму для кожної задачі із класу NP) [3].

Існуючі варіанти оптимізації — системи на основі мереж Петрі або прямі генетичні алгоритми — мають неадекватну швидкість та в своїй структурі несуть велику кількість обмежень [4].

Основними завданнями дослідження є: проаналізувати алгоритми вирішення задачі баласнування складальної лінії; розробити генетичний алгоритм групування для розподілу навантаження; створити програмний модуль, що реалізує цей алгоритм і може бути включений в іншу систему; розробити демонстраційний додаток для тестування та візуалізації роботи розробленого програмного забезпечення.

Практична цінність магістерської дисертації визначається тим, що вирішення основних завдань дослідження дозволить спростити процес прийняття рішень, що пов'язаний з плануванням навантаження на складальній лінії, мінімізації затрат виробництва.

Робота виконана в межах науково-дослідницького напрямку роботи кафедри автоматизації проектування енергетичних процесів і систем теплоенергетичного факультету Національного технічного університету України “Київський політехнічний інститут імені Ігоря Сікорського”. За результатами дипломної роботи була здійснена доповідь на конференції і опубліковано наукову статтю (додаток А). Результат роботи було впроваджено у ТОВ «НВП «Символ» (додаток Б).

1. ЗАДАЧА БАЛАНСУВАННЯ СКЛАДАЛЬНОЇ ЛІНІЇ ТА СИСТЕМИ ЇЇ ВИРІШЕННЯ В КОНТЕКСТІ СУЧАСНОГО ВИРОБНИЦТВА

Проаналізувавши ринок продуктів вирішення задачі балансування складальних ліній було зроблено висновок: незважаючи на велике практичне значення задачі балансування, існує невелика кількість комерційного програмного забезпечення для оптимізації балансування. Прийнято рішення створення програмного забезпечення для цієї задачі.

1.1 Постановка задачі

Задача балансування складальної лінії (БСЛ) може бути описана наступним чином: заданий набір завдань різної довжини, що підлягають обмеженням першості (тобто деякі завдання є передумовою для завершення однієї або декількох інших завдань); і деякий визначений сталий час, що називається часом циклу.

Завдання полягає в знаходженні відповіді на запитання: як мають бути завдання розподілені між робочими станціями уздовж лінії виробництва (складання), так щоб:

- жодне з правил порядку збірки не порушувалося;
- жодна з робочих станцій в лінії не використовувала більше часу, ніж заздалегідь визначений час циклу виконання завдань, призначених цій станції;
- для виконання всіх завдань у наборі використовується якомога менше робочих станцій.

У більш формальному плані ми визначаємо БСЛ як наступну задачу прийняття рішень.

Враховуючи спрямований ациклічний граф $G = (T, P)$ (вузли T , що представляють задачі, і стрілки P , що представляють обмеження пріоритету) з константою L_i (довжина завдання), призначений для кожного вузла T_i , константи C (час циклу) і константи N , чи можуть бути вузли T розподілені на N або менші підмножини S_j (завдання j -ї станції) таким чином, що

— для кожної з підмножин, сума L_i , пов'язана з вузлами підмножини, не перевищує C ;

— існує таке упорядкування підмножин, що, коли при поєднанні двох вузлів в окремих підмножинах стрілкою із G , вона йде від множини з більшим порядком (ранішої) до більш низького порядку (пізнішої) підмножини [3].

Легко показати, що БСЛ NP-повна: її можна звести до NP-повної задачі складання кошиків (ЗСК), яка є окремим випадком (а саме, набір обмежень пріоритету — граф G — порожній для ЗСК) [5].

Зрозуміло, що пов'язана з нею задача оптимізації, де запитується, яка мінімальна кількість необхідних станцій, NP-складна.

Що стосується наявного алгоритму для БСЛ, то не існує поліноміальних рішень, схожих на ті, які відомі для ЗСК. Більш того, існуючі рішення мають багато обмежень на них. Проте жорсткий зв'язок, що існує між двома проблемами, дозволить їх розглядати дуже схожим чином.

БСЛ — це класична проблема оптимізації операційних досліджень (ОД), яку намагаються вирішити протягом декількох десятиліть [6].

Для цієї проблеми запропоновано багато алгоритмів. І все ж, незважаючи на практичне значення цієї проблеми та зусилля, спрямовані на подолання цього чинного досвіду, існує невелика кількість комерційного програмного забезпечення, яке допомагає промисловості оптимізувати свої лінії [7].

Ця ситуація здається парадоксальною або, принаймні, непередбаченою: з огляду на величезний вклад в економіку, що може бути згенерований, можна було б очікувати,

що декілька пакетів програмного забезпечення збираються захопити частину цих економік.

1.2 Складальні лінії

Складальні лінії — це потокове виробництво систем, які все ще є типовими для промислового виробництва високоякісних стандартизованих товарів і навіть набувають важливості у виробництві нестандартних виробів.

Серед головних проблем, що виникають при управлінні такими системами, задачі балансування конвеєрної лінії є важливими завданнями середньострокового планування виробництва [8].

Лінія збірки складається з (робочих) станцій $k = 1, \dots, m$, розташованих уздовж стрічкового конвеєра або подібного механічного обладнання для навантаження матеріалу.

Заготовки (робочі місця) послідовно запускаються по лінії і переміщуються з станції на станцію.

На кожній станції неодноразово виконуються певні операції, враховуючи час циклу (максимальний або середній час, доступний для кожного робочого циклу).

Проблема вирішення задачі оптимального розподілу (балансування) збірочних робіт між станціями задля досягнення певної мети, називається задачею балансування конвеєрної лінії (ALBP) [9].

Виготовлення продукту на конвеєрній лінії вимагає розбиття загальної кількості роботи на набір елементарних операцій з завданнями $V = \{1, \dots, n\}$.

Виконання завдання j потребує часу t_j і вимагає певного устаткування машин та / або навичок працівників.

Через технологічні та організаційні передумови слід дотримуватися порядку слідування між операціями.

Ці елементи можна узагальнити та візуалізувати за допомогою графа першості. Він містить вузол для кожного завдання, маси вузлів для часів завдання та дуг для обмежень пріоритету.

Будь-який тип ALBP полягає у знаходженні можливих балансів лінії, тобто присвоєння кожної задачі станції таким чином, що виконуються обмеження пріоритету та інші обмеження [10].

Набір S_k -задач, призначених станції k ($= 1, \dots, m$), становить його навантаження на станцію, загальний час завдання називається часом станції.

Коли задано фіксований загальний цикл часу c (швидкість лінії), лінійний баланс можливий лише тоді, коли час станції жодної станції не перевищує c .

Балансування складальної лінії є довгостроковим рішенням і зазвичай вимагає великих капітальних вкладень.

Тому важливо, щоб така система була спроектована та збалансована, щоб вона працювала максимально ефективно.

Окрім балансування нової системи, необхідно виконати переробку періодично або після змін у виробничому процесі чи виробничій програмі [11].

Через довгостроковий ефект від балансування рішень, використовувані цілі повинні бути ретельно підібрані з урахуванням стратегічних цілей підприємства.

З економічної точки зору слід враховувати цілі, пов'язані з прибуттям.

Однак вимірювання та прогнозування витрат на експлуатацію лінії протягом декількох місяців або років та прибуток, досягнутий шляхом продажу зібраних продуктів, є досить складним і схильним до помилок [12].

Звичайна сурогатна мета полягає в тому, щоб максимізувати використання лінії, яка вимірюється за ефективністю лінії як продуктивну частку загального часу роботи лінії і безпосередньо залежить від циклу часу c і кількості станцій m .

1.3 Характеристики складальних ліній

Завдяки дуже різним умовам у промисловому виробництві, системи збірки ліній і відповідні алгоритми для роботи з ними є багатоманітними.

Нижче коротко характеризуються найбільш значущі властивості для класифікації ліній збірки.

У випадку швидкості складальної лінії, час роботи кожної станції обмежується часом циклу c як максимальним значенням для кожної деталі. Оскільки завдання є неподільними робочими елементами, c може бути не меншим, ніж найбільше завдання часу $t_{\max} = \max \{t_j \mid j = 1, \dots, n\}$.

У зв'язку з обмеженням часу циклу, встановлені лінії складання мають фіксовану швидкість виробництва (зворотну від часу циклу) [13].

За відсутності загального періоду циклу, тобто всі станції працюють з індивідуальною швидкістю, заготовкам, можливо, доведеться зачекати до того, як вони зможуть ввійти на наступну станцію, і / або станції можуть працювати в режимі очікування, коли їм доведеться чекати наступну деталь.

Ці труднощі частково подолані буферами між станціями. У цьому випадку з буферизованою (непошкодженою) конвеєрною лінією алгоритм балансування супроводжується додатковим вирішенням проблеми буферів позиціонування та розміру [14].

Якщо зібрано лише один продукт, усі заготовки ідентичні і існує одна модельна лінія.

Якщо на одній лінії виготовляється декілька продуктів (моделей), алгоритм підключається до задачі послідовності, яка повинна визначати послідовність збірки одиниць моделі [15].

Послідовність важлива для ефективності лінії, оскільки час виконання завдань може суттєво відрізнятись для різних продуктів.

Залежно від типу змішування агрегатів виникають два варіанти: лінія змішаної моделі виробляє одиниці різних моделей у довільно змішаній послідовності, тоді як мультимодельна лінія виробляє послідовність партій (кожна з яких містить одиниці тільки однієї моделі або групи аналогічних моделей) з проміжними операціями установки.

Тому в останньому випадку балансування та послідовність пов'язані з задачею визначення розміру [16].

Різні типи ліній характеризуються на рисунку 1.1, де різні моделі символізуються різними геометричними фігурами.

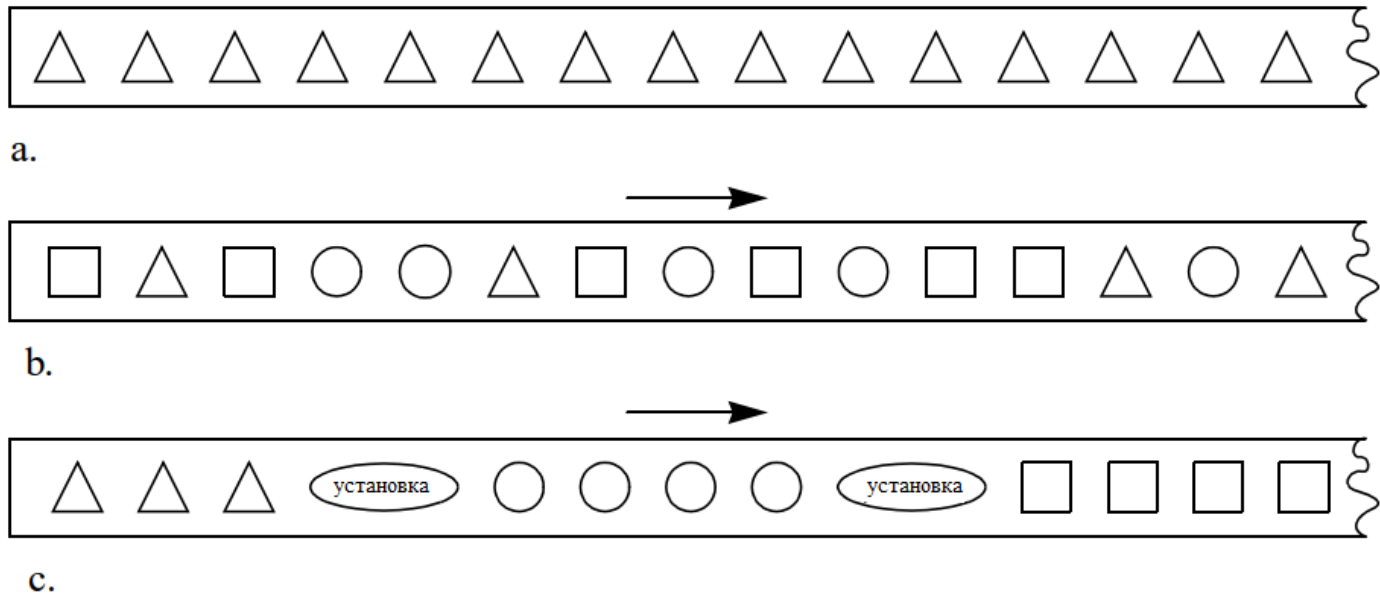


Рисунок 1.1 — Складальні лінії для одного і кількох продуктів

Залежно від цих лінійних типів, необхідно розглянути різні версії алгоритмів для їх балансування.

Ще однією важливою характеристикою, що визначає різні версії алгоритмів балансування, є мінливість часу роботи [17].

Кожного разу, коли очікувана дисперсія часу роботи досить мала, як, наприклад, у випадку простих завдань або надзвичайно надійних автоматичних станцій, час виконання завдання вважається детерміністичним.

Значні варіації, які в основному обумовлені нестабільністю людей у відношенні швидкості праці, майстерності та мотивації, а також неспроможністю чутливості складних процесів, потребують розгляду випадкових часів завдання.

Окрім випадкових коливань часу, систематичні скорочення можливі через навчальні ефекти або послідовне поліпшення виробничого процесу [18].

Через жорстку орієнтацію процесу, компонування систем виробничої лінії частково зумовлене потоком матеріалів.

Тим не менш, існують деякі можливості макетування. Традиційно конвеєрна лінія організована як серійна лінія, де по одному (прямому) конвеєрному поясу розташовані окремі станції.

Такі серійні лінії досить негнучкі та мають інші недоліки, які можуть бути подолані U-образною лінією збірки.

Обидва кінці лінії разом утворюють досить вузьке “U”. Станції можуть працювати на двох сегментах лінії одночасно один до одного (кросоверні станції) [19].

Окрім вдосконалень щодо збагачення робочих місць та стратегій розширення, U-подібна лінійна конструкція може призвести до кращого балансу навантажень станцій через більшу кількість комбінацій станцій.

Подальші удосконалення гнучкості та чутливості до збоїв системи складальних ліній можуть бути досягнуті декількома способами. Одним таким шляхом є введення деякого типу паралелізму.

У такому контексті встановлення повних паралельних ліній, кожна з яких призначена для одного продукту або сімейства суміжних продуктів, часто дозволяє краще балансувати і підвищує продуктивність [20].

Тоді алгоритми балансування супроводжуються додатковими рішеннями стосовно кількості ліній, що підлягають встановленню, та присвоєнням виробів та робочих сил до ліній.

Навіть у випадку однієї лінії переваги розпаралелювання можуть бути використані шляхом встановлення паралельних станцій, тобто оброблювані деталі розподіляються між кількома операторами, які виконують ті ж завдання [21].

Як у випадку з паралельними лініями, обладнання має бути встановлено кілька разів. Паралельні станції дозволяють зменшити (глобальний) час циклу системи, якщо певні задачі мають завдання довше, ніж час циклу [22].

Іншою можливістю скорочення часу глобального циклу є концепція паралельних завдань.

Відповідні задачі призначаються для декількох станцій серійної лінії, які циклічно виконують їх на різних заготовках. Кожного разу, коли є альтернативи процесу, тобто завдання можуть виконуватися різним обладнанням та / або за допомогою різних технологій, задача балансування пов'язана з проблемою вибору устаткування або процесу [23].

Для складання важких заготовок може знадобитися керувати двосторонньою лінією, яка складається з двох паралельних або послідовних ліній.

Замість окремих станцій пара паралельних станцій по обидва боки від лінії (з лівої та правої сторони станції) працюють паралельно, тобто вони працюють одночасно на протилежних сторонах одних і тих же заготовок [24].

Для виконання поставленої задачі станція повинна бути обладнана операторами та машинами, що мають необхідні навички та технологічні можливості.

Особливо у випадку складних виробів, як правило, неможливо мати всі станції, що однаково забезпечували б обмеження щодо присвоєння станції.

Крім того, розподіл завдань може бути обмежений пов'язаними з завданнями обмеженнями, такими як несумісність між задачами, мінімальними або максимальними

відставаннями (з точки зору часу чи простору) між станціями, що виконують пару (або підмножину) завдань [25].

Крім того, пов'язані з позицією обмеження стосуються деталей, важких, великих або закріплених на стрічці конвеєра таким чином, що вони не можуть бути повернуті в положенні, необхідному для виконання завдання на певній станції.

Інший тип обмежень присвоєння пов'язаний з оператором, оскільки оператори мають різні рівні майстерності. Тому певні комбінації завдань є можливими тільки тоді, коли оператор призначений для певної станції.

Крім того, слід враховувати аспекти задоволеності роботою працівниками [26].

1.4 Система OptiLine

Система Optiline, розроблена компанією OptiLine Enterprises — це інтегрований інструмент для оптимізації складальних ліній та зручного маніпулювання всіма даними, необхідними для виконання завдання [27].

На рисунку 1.2 представлено інтерфейс програми.

Стосовно маніпуляцій з інформацією, OptiLine пропонує зручне введення та інтуїтивно зрозуміле графічне представлення даних, що стосуються продукту та складальної лінії, існуючої чи майбутньої.

Після того, як буде вказано всі відповідні дані, OptiLine оптимізує лінію.

При цьому забезпечується детальне призначення операцій робочим станціям та операторам, з тим щоб звести до мінімуму дисбаланс робочого навантаження між робочими станціями.

Результати включають повний робочий графік всіх операторів, призначених для цієї лінії.

Результати також включають графік роботи кожного оператора на багатокористувацьких робочих станціях [28].

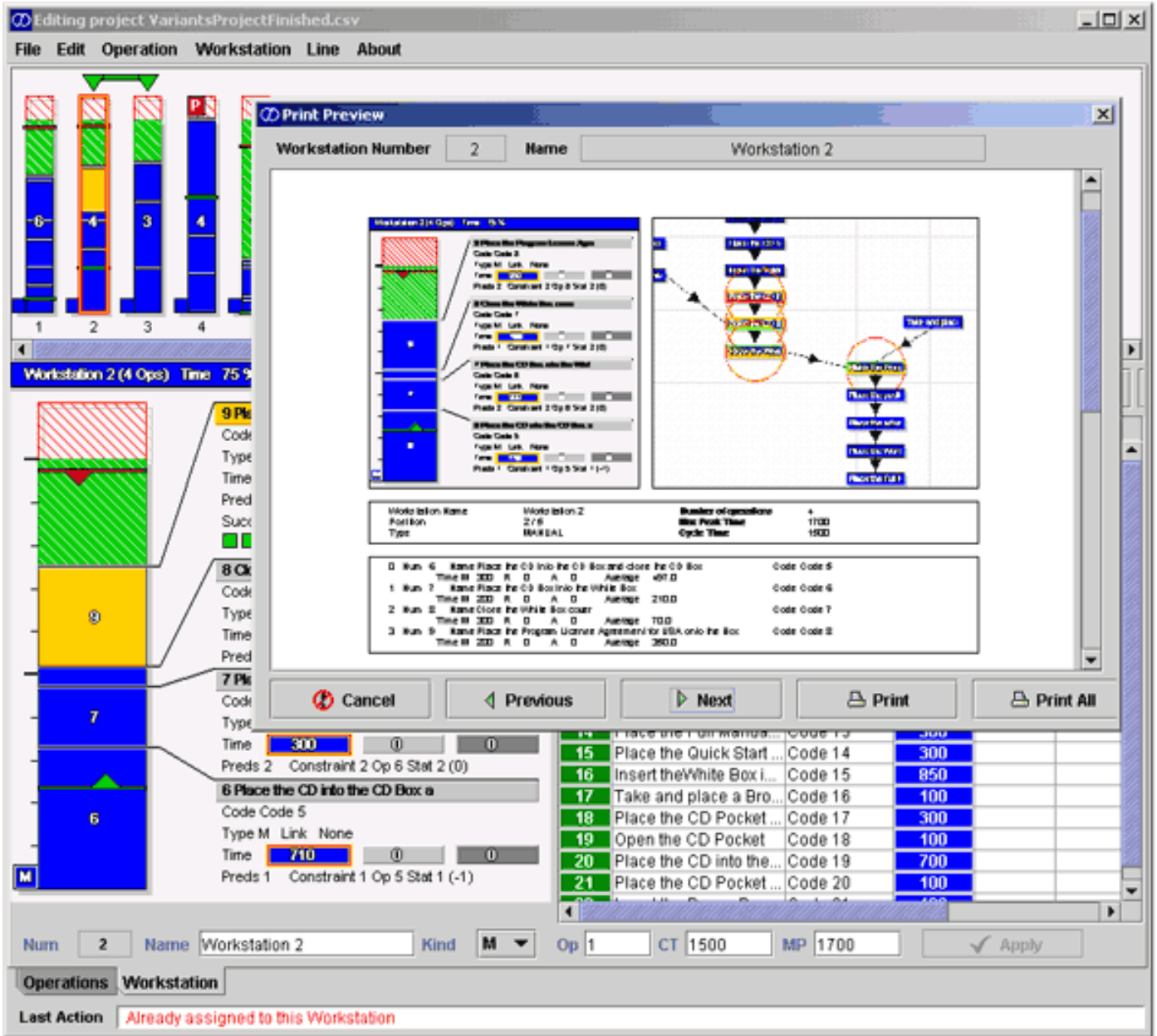


Рисунок 1.2 — Інтерфейс OptiLine

1.5 Система Proplanner

Система Proplanner — інструмент балансування складальної лінії, який пропонує простий у використанні інтерфейс електронної таблиці для роботи з завданнями, станціями, деталями, інструментами, моделями, параметрами та пріоритетом [29].

Це програмне забезпечення підтримує звичайні функції Windows, інтуїтивно зрозумілі та комплексні графіки, методи гнучкого та швидкого балансування лінії, а також звітність, яка аналізує використання ресурсів, та робочі завдання, які можна використовувати як інструкції з роботи.

Цей інструмент також інтегрований з планувальником робочого місця, який дозволяє користувачам проводити аналіз роботи в межах станції [30].

Система Proplanner пропонує індивідуальну програму оцінювання часу, яка пропонує простий у користуванні інтерфейс електронних таблиць для обробки спостережуваних завдань та / або заздалегідь визначених елементів [31].

Ця програма підтримує інтегроване відео (за наявності), звичайні функції Windows, інтуїтивно зрозумілі та складні графіки, гнучке та швидке розбирання коду, користувацькі шаблони стандартних даних на базі Excel, а також звітність, яка розраховує час та різні особисті фактори та затримки [32].

Ця програма підтримує всі промислові стандартні мови, включаючи Модапти, MTM-1, MTM-2, MTM-UAS, MTM-B, MTM-SAM, MTM-МЕК, та MOST [33].

Інтерфейс системи зображено на рисунку 1.3.

Планувальник потоку гарантує, що зміни в схемі об'єкта є фактичними поліпшеннями.

Програма працює з AutoCAD, щоб автоматично генерувати схеми потоку матеріалів та обчислювати витрати на транспортування, розміри, час і вартість макета, намальованого в AutoCAD [34].

З послідовними лініями з змінною шириною, кодованими за допомогою коду продукту, частини або способу обробки матеріалу, користувачі швидко бачать, як макет повинен бути організований, і де надмірна обробка матеріалів повинна бути виключена з виробничого процесу. Працюючи в AutoCAD, планувальник робочих місць автоматично генерує діаграми переміщення та обчислює час ходьби працівників, зменшують додану вартість, пройдену відстань та ергономічні оцінки безпеки.

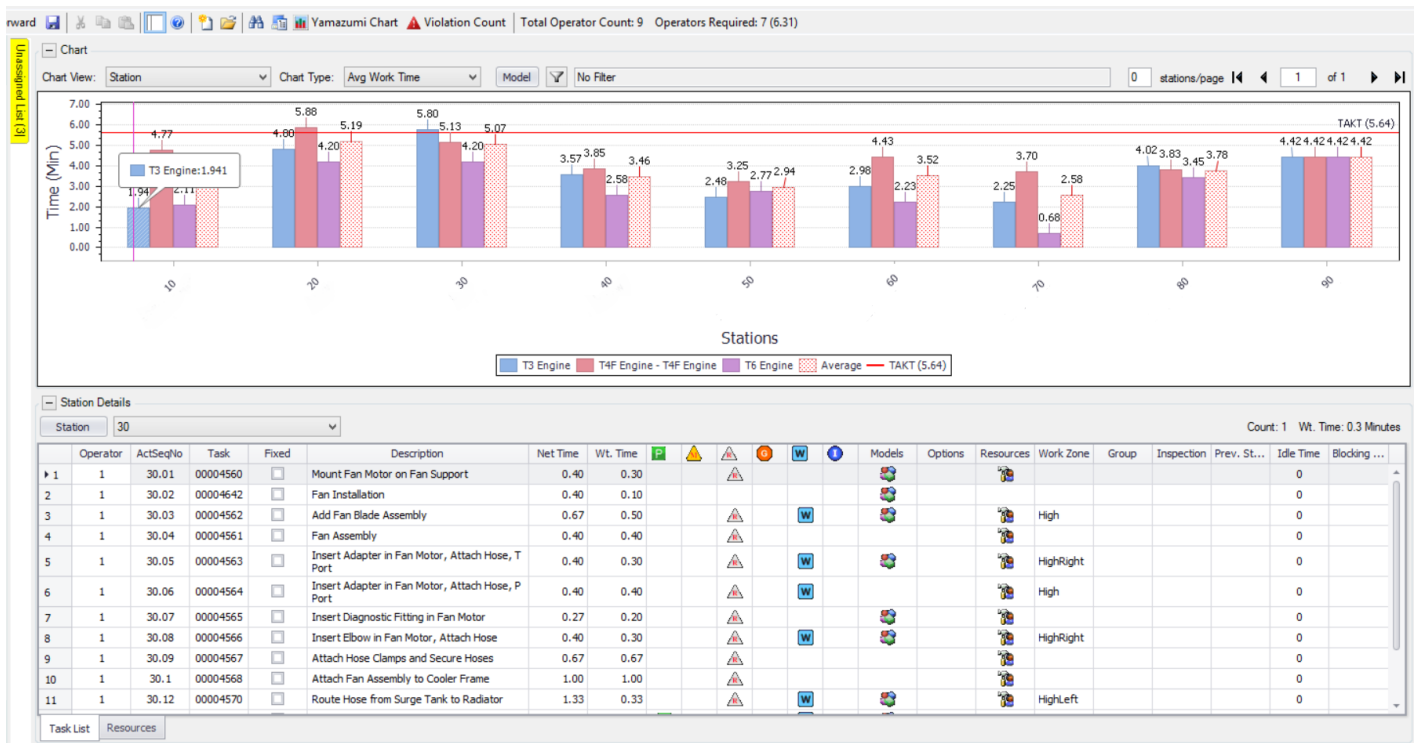


Рисунок 1.3 — Інтерфейс Proplanner.

Діаграми переміщення та анімація допомагають візуалізувати процес безпосередньо на макеті в AutoCAD [35].

Модуль Shop Floor Viewer виводить робочі інструкції на наступний рівень, щоб надати роботодавцям безпосередній доступ до перегляду електронних інструкцій з роботи за допомогою веб-браузера [36].

Цей безпаперовий метод забезпечує можливість онлайн перегляду останніх документів.

Працівники можуть сканувати номер моделі, з якою вони працюють, щоб отримати інструкцію з роботи щодо їх конкретної моделі та станції [37].

Модуль PFER Proplanner (план для кожної частини) — це база даних обробки матеріалів, головний репозиторій для внутрішньої логістики та матеріалів, що обробляють інформацію компанії [38].

База даних зберігає, управляє та аналізує всю інформацію про логістику частини від причепа в приймальній док-станції до місць збірки на підлозі заводу.

Модуль Kanban відстежує заміну повних контейнерів та наповнення пустих у зручній системі Kanban.

Модуль Kitting генерує вибірккові списки для деталей, необхідних для придбання, для задоволення конкретних потреб продукту (або робочої станції) на складальній лінії.

1.6 Алгоритми поточних рішень

Одним із варіантів ефективного вирішення задачі балансування складальної лінії є використання імітаційного моделювання з використанням мережі Петрі. Основним недоліком такого підходу є відсутність формальної аналітичної залежності при визначенні часу роботи лінії, що робить неможливим пошук оптимуму за використовуваними критеріями ефективності класичними методами оптимізації [39].

Для вирішення задачі балансування також було використано класичні генетичні алгоритми [40]. Проте вони не можуть адекватно репрезентувати задачі групування, тому отримані таким способом результати не є оптимальними.

У стандартному генетичному алгоритмі для задач групування використовуються хромосоми, що складаються з груп чисел.

Кожен предмет має один ген, який кодує номер групи, до якої предмет відноситься в відповіді, репрезентованій хромосомою.

Наприклад, хромосома ABCADD кодує рішення, де пункти 1 і 4 належать до групи А, пункт 2 до групи В, 3 — до С і 5 і 6 до D.

Багато практичних проблем групування мають жорсткі обмеження, які роблять деякі групи недійсними.

Привабливою рисою цього підходу є те, що кількість елементів є константою для одного прикладу проблеми, що призводить до стандартної хромосоми постійної довжини.

Хоча це кодування здається привабливим, коли воно використовується спільно зі стандартними операторами (не модифікованими), виникає ряд недоліків.

Найбільшою проблемою, є схрещування (кросовер).

Основною проблемою зі стандартним кросовером, що застосовується до групи хромосом, є його нечутливість до контексту.

Кросовер передає гени, що представляють собою ізольовані елементи, однак один ізольований елемент не має сенсу в групуючій проблемі.

Він має лише сенс у контексті групи інших предметів, що входять до однієї групи.

Це пояснюється тим, що за визначенням задач групування фітнес функція визначається на групах, а не на окремих об'єктах.

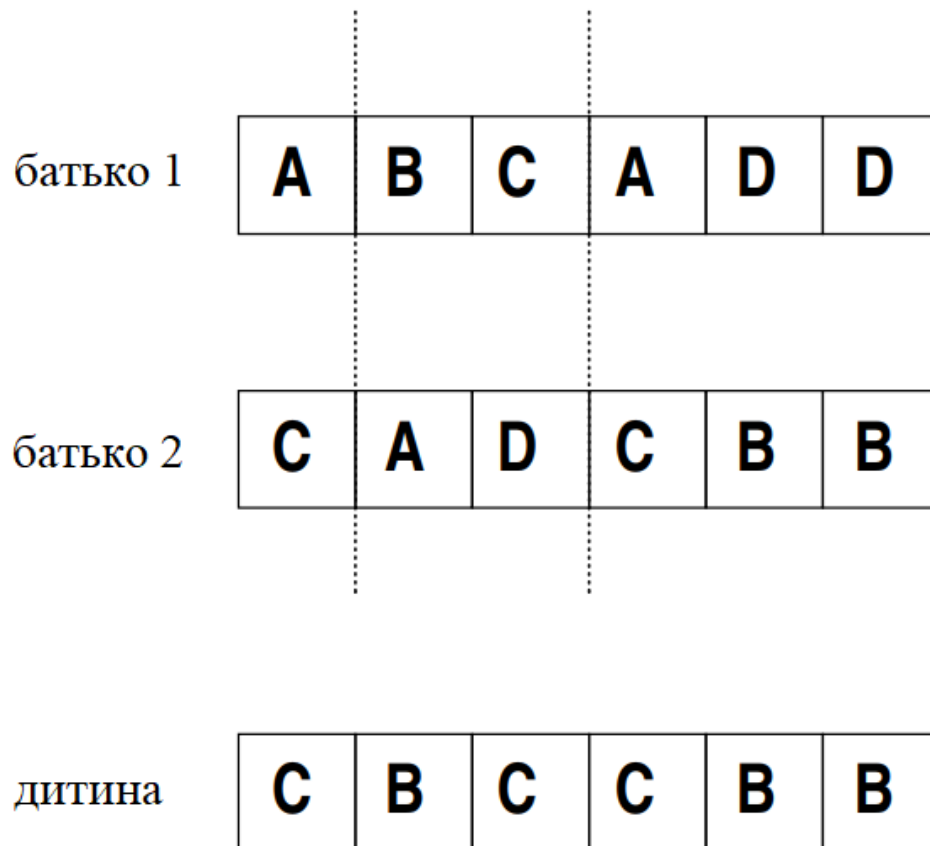


Рисунок 1.4 — Проблема стандартного кросоверу

Один аспект нечутливості до контексту наведено на рисунку 1.4, де стандартний двоточковий кросовер був використаний для побудови дитини двох хромосом.

При доброму кросовері, двоє однакових батьків повинні рекомбінувати в потомство, яке ідентичне батькам.

Два батьки в на рисунку 1.4 ідентичні, тому що, як можна перевірити, вони обидва кодують однакове групування.

Однак дитина дуже далека від ідентичності до батьків. Рішення, яке вона кодує, має лише дві групи замість чотирьох у батьків.

Можна стверджувати, що перейменування груп у прикладі полегшить проблему, але, навіть це не допомагає, якщо задача не має спеціальної структури.

Іншим аспектом нечутливості до контексту є втрата логічного змісту кросоверу. Значення генів ігнорується в кросовері.

Частини батьківських рішень, які мають сенс в контексті вирішеної проблеми, і, отже, вони повинні бути повторно перевірені (відновлені) через успадкування, втрачаються під час кросовера. Це означає, що кросовер не виконує передбачувану неявну роботу.

Інший попередньо використовуваний підхід генетичних алгоритмів до завдань групування використовує порядковий генетичний алгоритм в поєднанні з декодером.

Хромосома являє собою перестановку елементів, а декодер конструює групування певним чином, залежно від перестановки. Значення фітнес функції потім обчислюється на результуючому групуванні.

Використовувані декодери, звичайно, відрізняються для різних проблем, і мають бути вручну переписані для кожної з них.

Загальна ідея порядкових генетичних алгоритмів для проблеми складання кошиків проілюстрована на рисунку 1.5, де гени позначаються розмірами предметів.

Значення фітнес функції для закодованої упаковки, отримується шляхом наступного декодування: елементи приймаються в порядку, визначеному хромосомою

(зліва направо), вводячи поточний елемент у поточний кошик, якщо це можливо, або запитує новий контейнер, якщо ні.

Порядок має передбачати перестановку, яка дозволить декодеру побудувати хороший, в ідеалі оптимальний, спосіб вирішення проблеми групування.

Проте ситуація не буде покращена кросовером.

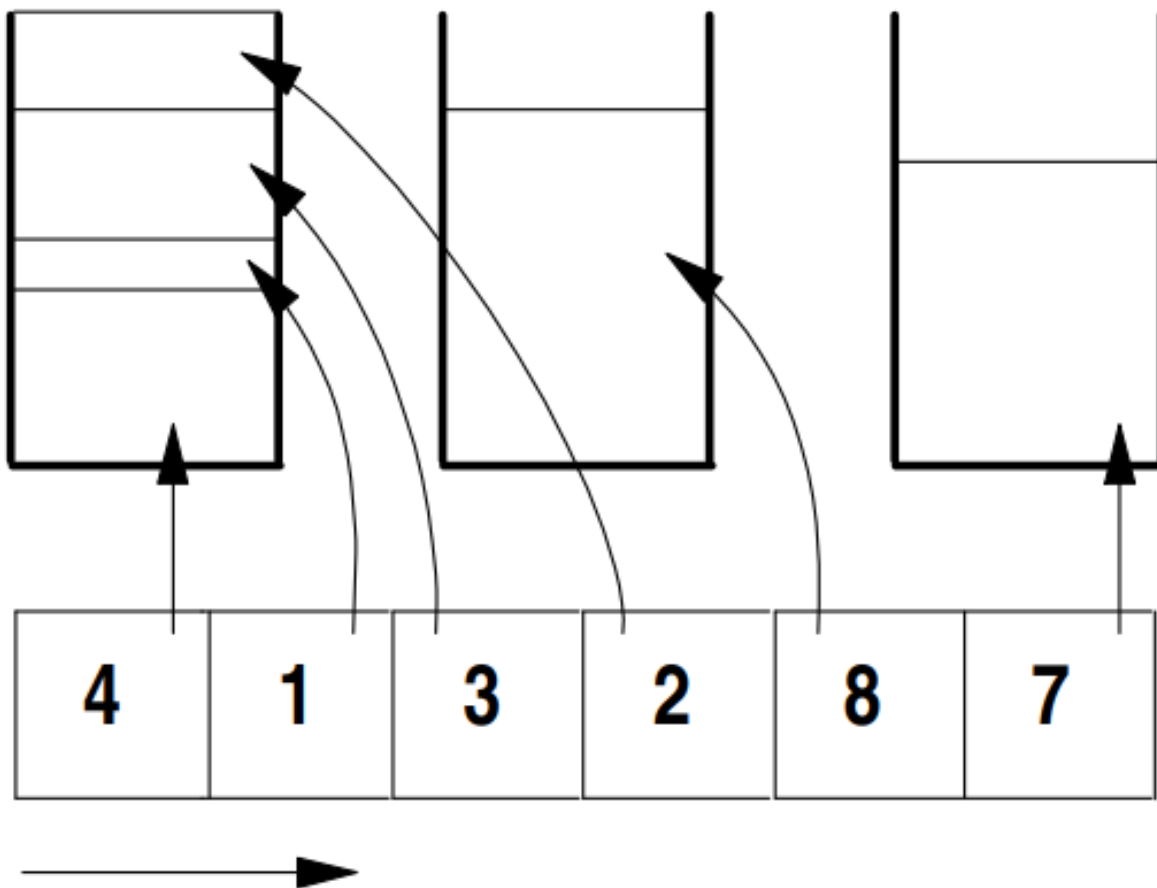


Рисунок 1.5 — Приклад хромосоми порядкового алгоритму

На сьогоднішній день використовуються декілька кросоверів для порядкових генетичних алгоритмів.

Заради прикладу використано РМХ Гольдберга, що, ймовірно, найбільш широко цитується в літературі [41]. Ситуація проілюстрована на рисунку 1.6.

Кросовер РМХ передає абсолютні положення ділянки перетину, взяті з одного батька, і відносні положення інших генів, взятих з іншого батька.

Зрозуміло, що хромосома в нижній частині грази є валідною дитиною хромосоми у верхній частині.



Рисунок 1.6 — Ефект РМХ

При доброму кросоверному операторі дитина повинна успадкувати більшість значущої інформації від одного або іншого батька. Але в РМХ це далеко не так, як видно з наведеного прикладу.

Задача складання кошика є задачею групування.

В той же час групи, отримані шляхом розшифрування дитячої хромосоми, мають дуже мало спільного з тими, які отримані з батьківської хромосоми: число і, що більш важливо, вміст груп, дуже різні (лише група, що містить елемент розміру 8, перейшла до дитини).

Знову ж таки, групи, які мають передаватися зі спадщиною через кросовер, не передаються, роблячи операцію марною.

Висновки до першого розділу

В розділі була описана формальна постановка задачі балансування складальної лінії і виявлено, що вона є NP-складною. Описані подробиці задачі. Описані різновиди складальних ліній.

Приведені приклади альтернативних рішень — їх інтерфейси та використовувані алгоритми. Показані недоліки альтернатив.

2. АЛГОРИТМІЧНІ ОСНОВИ ВИКОРИСТАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ БАЛАНСУВАННЯ СКЛАДАЛЬНОЇ ЛІНІЇ

Для вирішення задачі балансування складальної лінії було застосовано генетичний алгоритм групування.

2.1 Генетичні алгоритми

Генетичний алгоритм — це еволюційний алгоритм пошуку, що використовується для вирішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.

Особливістю генетичного алгоритму є акцент на використання оператора “схрещування”, який виконує операцію рекомбінування рішень-кандидатів, роль якої аналогічна ролі схрещення в живій природі.

Метатеорії, такі як Simulated Annealing (SA), Tabu Search (TS) та генетичні алгоритми є методами, які можна застосувати до дуже широкого кола завдань оптимізації — можливо, будь-яких [42].

Враховуючи величезну різноманітність цих завдань є дуже мало, що можна було б використати для вирішення всіх цих завдань.

Фактично, вважається, що існує лише одне цілковите загальне правило, якому слідує всі метаевристики, щоб уникнути чисто випадкового пошуку, а саме: дотримуватися того, що ви вже знаєте, що працює.

Іншими словами, в пошуках величезного простору можливих рішень, щоб отримати оптимальне рішення, слід використовувати якусь “натхнення” з найкращого рішення, яке поки що було виявлено.

Звичайно, неможливо повною мірою дотримуватися правила, тому що тоді не буде пошуку взагалі, тому необхідно прийняти певні розумні відступи. Зазвичай прийнятий ґрунтується на припущенні, що розумніше шукати в околицях добрих рішень, ніж просто шукати скрізь (як у випадку випадкового пошуку).

Таким чином, як SA, так і TS вивчають пошуковий простір, ітераційно дивлячись на невелику сусідню ділянку поточного (найкращого) рішення. Вони використовують різні механізми (вірогідність прийняття погіршень в залежності від температури в першому, перелік табу у другій), щоб переконатися, що в будь-якому разі доступ до віддаленої точки в пошуковому просторі існує, якщо поточне найкраще рішення виявиться насправді не оптимальним.

Генетичний алгоритм також відповідає загальному правилу. Оригінальність генетичного підходу полягає в тому, що, на відміну від SA і TS, визначення поняття “доброго” не є цілим рішенням, а його частинами.

Таким чином, основним припущенням в генетичному алгоритму є те, що якщо рішення є добрим (щодо виконуваного завдання оптимізації), то це, мабуть, тому, що деякі його частини є хорошими. Важливо те, що це стосується також двох рішень: якщо два рішення мають бути добрими, то це, мабуть, тому, що кожен з них містить добрі частини, але це не обов’язково (насправді, рідко) однакові в обох рішеннях.

Прийнявши це припущення, питання полягає в тому, як воно може бути використане для того, щоб слідувати метаевристичному правилу пошуку в околицях, описаному вище.

Виявляється, він є відносно простий: оскільки метою є збереження частин старих рішень, нові рішення можуть бути побудовані шляхом об’єднання в одному із них частин, що походять з двох різних рішень (неможливо зберегти всі частини лише

одного рішення і при цьому побудувати нове). Це, звичайно, ні що інше, як ідея рекомбінації, або кросовер [43].

Проблема в тому, що ми ніколи не знаємо, яка саме частина хорошого рішення робить його гарним, оскільки ми маємо лише міру вартості всього рішення (об'єктивна функція). Отже, частини повинні бути перевірені.

Але знову ж таки, неможливо протестувати частину рішення, можна оцінити лише повні рішення. Таким чином, має сенс ідея судити про якість частини за якістю різних рішень, які можуть містити її.

Таким чином, ми прийшли до ідеї відбору зразків усіх можливих рішень, що містять певну частину, для того, щоб оцінити вартість цієї частини.

Ідея, звичайно, призводить до питання про те, які частини повинні бути перевірені (тобто відібрані) з якою частотою, щоб мінімізувати очікуваний час, витрачений на тестування частин, які не призводять до хороших рішень.

Ідея поєднання частин рішень з метою створення перспективних нових рішень є надзвичайно привабливою (ми часто застосовуємо такий підхід у повсякденному житті) і відсутня в інших метаевристиках.

Цей процес виділяє генетичні алгоритми і надає їм перевагу над іншими методами оптимізації. Детально він був досліджений Джоном Голландом (англ. John Holland), книга якого “Адаптація в природних і штучних системах” є фундаментальною в цій сфері досліджень.

Задача кодується таким чином, щоб її вирішення могло бути представлено в вигляді масиву подібного до інформації складу хромосоми. Цей масив часто називають саме так “хромосома” [44].

Випадковим чином в масиві створюється деяка кількість початкових елементів “осіб”, або початкова популяція. Особи оцінюються з використанням функції пристосування, в результаті якої кожній особі присвоюється певне значення пристосованості, яке визначає можливість виживання особи

Функція пристосованості також отримала свою назву безпосередньо із генетики. Вона надає сильний вплив на функціонування генетичних алгоритмів і повинна мати точне і коректне визначення. У задачах оптимізації функція пристосованості, як правило, оптимізується (точніше кажучи, максимізується) і називається цільовою функцією.

Після цього з використанням отриманих значень пристосованості вибираються особи, допущені до схрещення (селекція).

До осіб застосовується “генетичні оператори” (в більшості випадків це оператор схрещення (crossover) і оператор мутації (mutation)), створюючи таким чином наступне покоління осіб [45].

Особи наступного покоління також оцінюються застосуванням генетичних операторів і виконується селекція і мутація. Так моделюється еволюційний процес, що продовжується декілька життєвих циклів (поколінь), поки не буде виконано критерій зупинки алгоритму.

Таким критерієм може бути:

- знаходження глобального, або надоптимального вирішення;
- вичерпання числа поколінь, що відпущені на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

Цей процес зображено на рисунку 2.1.

Генетичні алгоритми можуть бути використані для пошуку рішень в дуже великих і важких просторах пошуку [46].

В даній роботі використовується підтип генетичних алгоритмів — алгоритм групування. Він є продовженням звичайних генетичних алгоритмів, адаптованих до задач групування.

Генетичні алгоритми групування (GGA) були розроблені для вирішення проблем кластеризації. Фактично, GGA є генетичною структурою для задач групування, тобто кожна конкретна проблема потребує власної настройки.

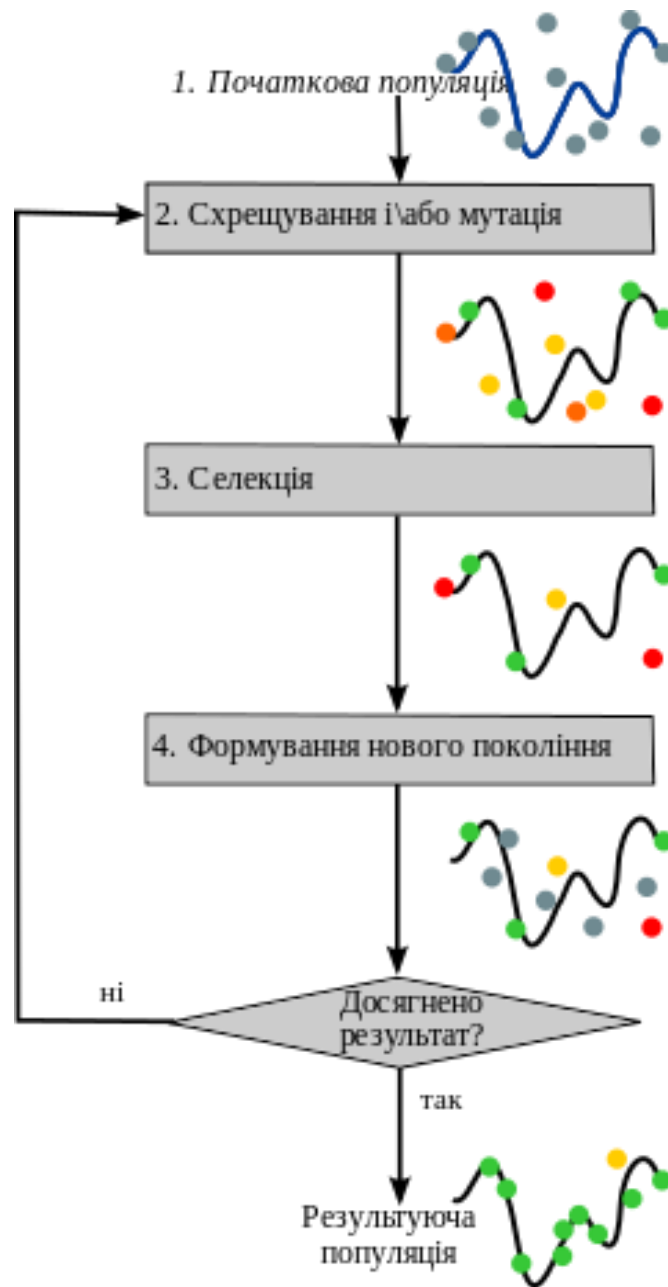


Рисунок 2.1. Схема роботи генетичного алгоритму

2.2 Опис реалізації програмного продукту

Кодування, яке використовується в генетичному алгоритмі групування, показано на рисунку 2.2.

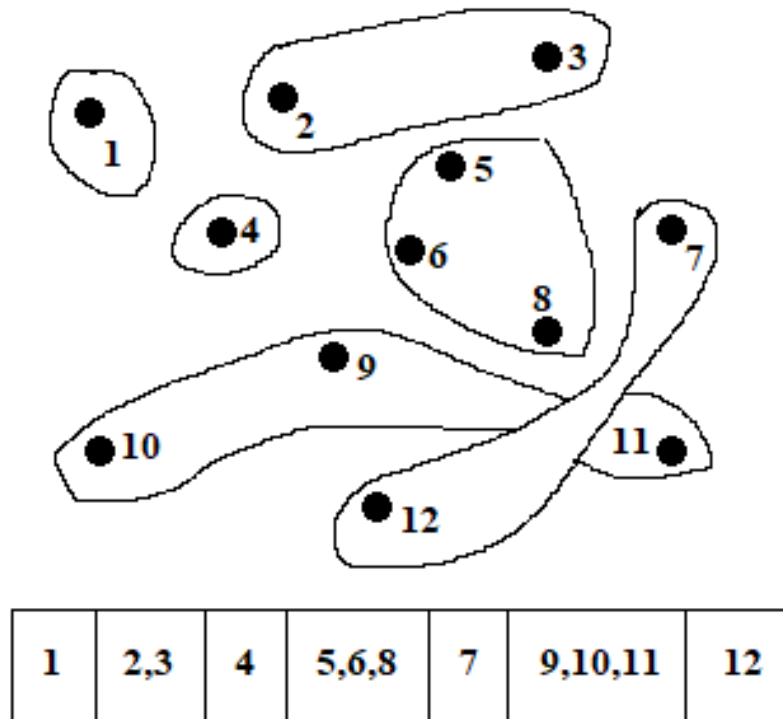


Рисунок 2.2 — Кодування груп в хромосомі.

Враховуючи вищесказане спостереження — що групи представляють значущі закономірності проблем групування — кожній групі відповідає один ген.

Елементи в наборі, знаходяться “всередині” гена.

Порядок елементів “всередині” гена не має значення — лише склад. Також, порядок генів на хромосомі є неважливим.

Кодування має декілька особливостей. Найбільш помітним є те, що, якщо задача не накладає обмеження на кількість груп, то довжина хромосоми може змінюватись.

Більш цікавою властивістю кодування є те, що немає чіткої різниці між локусом і алелем: значення гена виключно визначається складом групи, яку він кодує.

Якщо взяти цю композицію за локус, то немає реального поняття алеля.

Нарешті, якщо ми наполягаємо на тому, що склад групи фактично є алелем (тобто це означає, що тоді немає реального поняття локусу), тоді алфавіт кодування не

тільки експоненційно великий, але також залежить від екземпляра проблеми, що вирішується.

2.2.1 Фітнес функція

Мета полягає в тому, щоб знайти мінімальну кількість необхідних контейнерів.

Перша вартісна функція, яку можна використати — це просто кількість контейнерів, які використовуються для “упаковки” всіх об’єктів.

Це правильно з суто математичної точки зору, але на практиці непридатне для використання. Дійсно, така функція вартості призводить до надзвичайно “недружнього” ландшафту пошукового простору: дуже мале число оптимальних точок у просторі втрачається в морі точок, де ця передбачувана функція вартості становить лише одну одиницю над оптимальним.

Більше того, всі ці трохи недооптимальні моменти дають однаковий результат.

Проблема в тому, що така функція вартості не має можливості керувати напрямком пошуку.

Розглянемо екстремальний випадок, коли тільки одне розташування об’єктів дає оптимум, скажімо, N контейнерів.

Кількість можливих рішень збірки $N + 1$ контейнерів зростає експоненційно з N і, таким чином, дуже велике навіть для невеликих за розміром проблем.

Тим не менше, всі ці точки в пошуковому просторі дають однакову вартість $N + 1$ і, таким чином, виявляються абсолютно рівними за пріоритетом для пошуку їх оточення.

Іншими словами, алгоритм повинен опинитися в оптимальному рішенні лише випадково. Це було б непрактично. Таким чином, доведеться знайти функцію вартості, яка призначає подібні (але не рівні) значення аналогічним рішенням, при цьому вони мають такі ж оптимуми, що й функція, описана вище.

Іншими словами, ми повинні визначити найменший натуральний шматочок рішення, який є достатньо значущим, щоб передати інформацію про очікувану якість рішення, яке є його частиною.

Це можливо зробити у випадку зі складанням кошиків: кожен контейнер являє собою природний “інформаційний квант”. Ми просто помічаємо, що чим краще кожен з контейнерів використовується, тим менше контейнерів потрібно для упаковки всіх об’єктів.

І навпаки — погане використання ємності контейнерів призводить до необхідності додаткових контейнерів, щоб упакувати об’єкти, що не помістились в змарнований пустий простір.

Для того, щоб виділити контейнер, а не загальну продуктивність всіх контейнерів, ми також повинні визнати наступне: якщо ми візьмемо два контейнери та перемішаємо їх вміст серед них, то ситуація, коли один з контейнерів майже повний (залишаючи інший практично порожнім) краще, ніж коли два контейнера приблизно однаково наповнені.

Це пояснюється тим, що майже порожній контейнер спрощує розміщення додаткових об’єктів, які в іншому випадку можуть бути занадто великими, щоб вписатись у будь-який з напівзаповнених контейнерів.

Таким чином ми визначаємо наступну фітнес-функцію:

$$f = \frac{\sum_{i=1}^N \left(\frac{fill[i]}{C}\right)^k}{N},$$

де:

N — кількість контейнерів,

$fill[i]$ — наповненість i -го контейнера,

C — ємність контейнера,

k — константа, $k > 1$.

Іншими словами, функція для максимізації є середнім k -ї потужності “ефективності контейнера” (його заповненості) між усіма контейнерами.

Константа k виражає фокус на добре заповнених “елітних” контейнерах порівняно з менш заповненими.

Якщо $k = 1$, важливою буде лише загальна кількість використаних контейнерів, проти чого було застережено вище.

Чим більше k , тим більше віддаємо перевагу “екстремістам”, на відміну від набору з приблизно однаково заповнених контейнерів.

Проведені експерименти з кількома значеннями k виявили, що $k = 2$ дає хороші результати. Більш великі значення k призводять до передчасної конвергенції алгоритму.

2.2.2 Кросовер

Після проведення операції селекції генів відповідно до величини їх фітнес-функцій, відбувається процес схрещування.

Схрещування є найважливішою генетичною операцією, під час якої передається частина генетичної інформації від батьків.

Використовується стандартний двоточковий оператор схрещування, адаптований для роботи на хромосомах генетичного алгоритму групування.

З огляду на специфіку хромосом, з якими вона працює, кросовер генетичного алгоритму групування має бути модифікованим.

Він не може піти шляхом простого копіювання та вставки генів, як у стандартних кросоверів.

Кросовер слідує схемі, зображеній на рисунках 2.3 — 2.6.

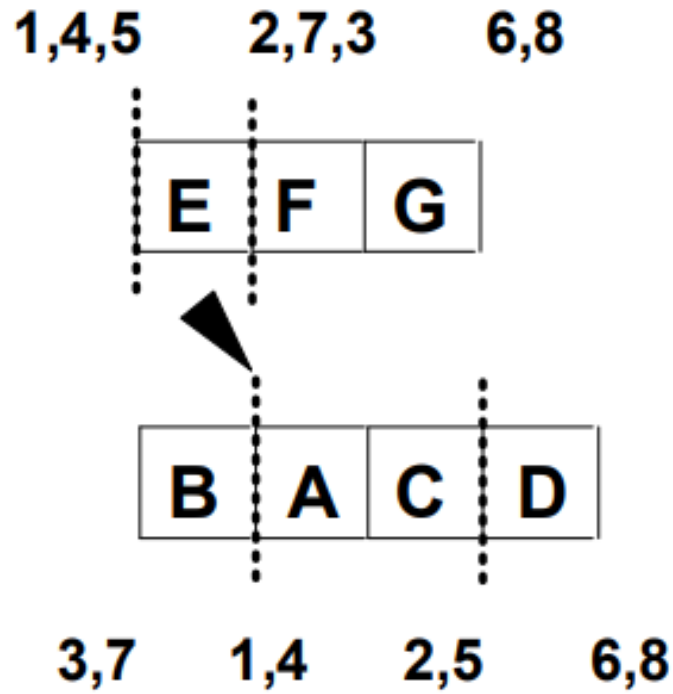


Рисунок 2.3 — Вибір точок кросоверу.

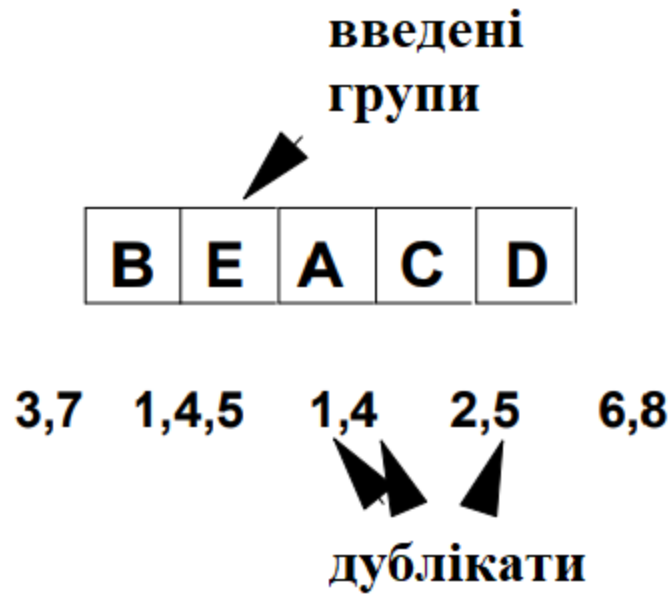


Рисунок 2.4 — Введення груп.

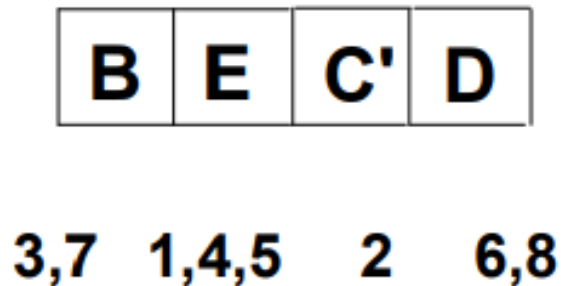


Рисунок 2.5 — Видалення дублікатів.

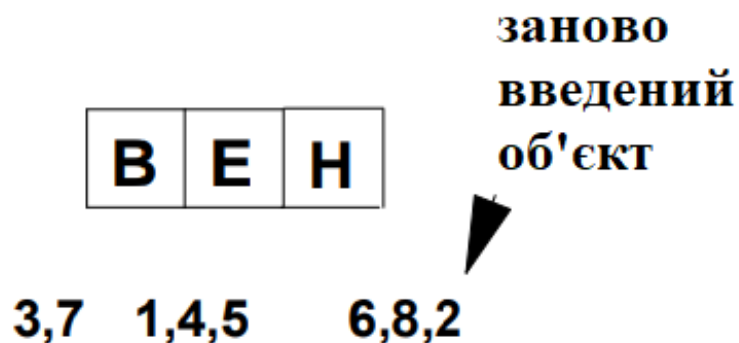


Рисунок 2.6 — Адаптація модифікованих груп

Він функціонує таким чином. По-перше, точки кросоверу вибираються випадковим способом у першому батькові. У випадку, зображеному на рисунку 2.3, це ген позначений E.

Як і в стандартному кросовері, цей розділ перетину потім “експортується” до іншого батька. При стандартному кодуванні гени в ділянці перехрестя просто замінять гени у другого батька, що мають однакові позиції, але оскільки в нашому кодуванні не існує реального поняття позиції, необхідно застосувати більш складний механізм.

Тому наш кросовер вводить розділ перетину до іншого батька. Однак, оскільки кожен з генів являє собою групу предметів, це означає, що в рішенні, представленому дитиною хромосоми на цьому етапі (рисунок 2.4) містяться деякі дублікати.

Оскільки повторення не допускаються в задачі пакування, дублікати усуваються шляхом ліквідації “старих” елементів, як на рисунку 2.5. Таким чином, членство в групі деяких елементів в другому батькові змінилось на те, що було в першому батькові.

На цьому етапі дитина-хромосома є правильним рішенням проблеми групування і може бути використана як вона є. Проте в процесі усунення подвійних випадків предметів деякі групи, що надходять від другого батька, були модифіковані. Наприклад, ген С' на рисунку 2.5.

Такі групи, ймовірно, мають низьку якість (ці групи не входили до батьківського набору). Тому ми виправляємо це на додатковому етапі адаптації: модифіковані групи видаляються, а елементи, які вони містять, знову вставляються в рішення. Ми робимо це за допомогою евристики FF.

Фундаментальна властивість кросовера виявляється на рисунку 2.6: дитина успадкувала групу, позначену буквою В з другого батька, і групу з номером Е від першого батька. Іншими словами, дитина успадкувала інформацію, важливу для вирішення проблеми, з обох батьків. Таким чином, кросовер виконує свою призначену місію.

2.2.3 Мутація

Оператор мутації для задачі складання кошиків простий: для хромосоми, ми випадковим чином вибираємо кілька генів і видаляємо їх.

Таким чином, об'єкти, що склали ці контейнери, відсутні в результаті, і ми використовуємо FF, щоб вставити їх назад у випадковому порядку.

Для того, щоб поліпшити шанси на мутацію для покращення поточного рішення, ми дотримуємося двох правил: найбільш порожній контейнер завжди знаходиться серед вилучених, а також ми завжди вилучаємо та згодом повторно вставляємо

щонайменше три гени (кількість використаних контейнерів не може бути покращено з меншою кількістю).

Слід зауважити, що, як і у випадку з кросовером, навіть деякі контейнери, які не були виділені для усунення, можуть бути заповнені об'єктами з вилучених контейнерів.

2.2.4 Інверсія

Роль оператора інверсії — зміна позицій деяких генів на хромосомі, щоб допомогти кросоверу передавати хороші контейнери потомству.

На відміну від двох інших операторів, цей оператор впливає лише на подання генетичного вмісту хромосоми, а не на саму інформацію. Зміна позиції генів на хромосомі не впливає на склад об'єктів у контейнерах, об'єктна частина хромосоми залишається незмінною.

Наприклад, хромосома BECDA може бути перетворена в SEBDA. Приклад ілюструє корисність цього оператора: якщо B та D будуть перспективними генами (тобто добре заповненими контейнерами), імовірність передачі їх обох під час наступного кросовера поліпшується після інверсії, оскільки вони тепер ближче до одного. Це у свою чергу полегшує розповсюдження хороших генів.

2.2.5 Модифікація до балансування складальної лінії

Враховуючи схожість між упаковкою контейнерів та балансуванням ліній, зазначених у розділі 1, ми можемо легко модифікувати оператори кросовера та мутації для останньої проблеми.

Інверсія не змінює інформацію в хромосомі, тому описаний вище оператор може бути використано для обох задач.

Єдиною відмінністю між двома задачами є додаткове обмеження графу першості для балансування ліній. Тому все що потрібно зробити для адаптації операторів ЗСК для БСЛ — забезпечити, щоб жоден з операторів не порушував це обмеження.

Це можна зробити, якщо відзначити наступне:

- усунення груп (тобто робочих станцій) від дієвого рішення не порушує обмежень, тобто нове (часткове) рішення знову є валідним;
- кожного разу, коли ми доповнюємо рішення, розмістивши об'єкти в контейнерах, ми використовуємо евристики FF або FFD.

Зрозуміло, що переконавшись, що евристики ніколи не порушують обмеженість пріоритету, ми забезпечимо перехід до проблеми балансування ліній.

Повертаючись до визначення БСЛ вище, ми відзначаємо, що обмеження пріоритету порушуються, коли в графі, отриманому з вихідного графа G об'єднанням вузлів однієї групи (робочої станції) в один вузол, з'являється цикл.

Дійсно, з циклом у вищезгаданому графі неможливо сказати, яка з груп передує або йде за іншою в циклі, тобто неможливо знайти необхідний порядок підмножин.

Щоб запобігти циклам у графі, ми змінюємо евристики наступним чином: кожен раз, коли ми збираємось вставити об'єкт у групу (завдання на складальну лінію, або в ЗСК — об'єкт у контейнер), ми обчислюємо довжину (з точки зору кількості стрілок) найдовшого шляху переходу, що веде від поточного об'єкта до всіх вже існуючих груп. Вставка дозволяється тільки у випадку, якщо об'єкт взагалі не пов'язаний з групою або пов'язаний рівно однією стрілкою.

2.3 Альтернативні процеси

Розглянемо модель, яка поєднує в собі задачу балансування з задачею про альтернативні процеси.

Модель розглядає базовий процес, який може бути доповнений одним або кількома необов'язковими альтернативними процесами, кожен з яких зменшує певний час роботи або навіть повністю виключає певні завдання, так що кожна комбінація альтернатив визначає граф з пріоритетністю.

Ці альтернативи викликають фіксовані витрати за одиницю часу. В якості додаткової категорії витрат розглядаються ставки заробітної плати за одиницю часу.

Загальна заробітна плата залежить від кількості станцій, кожна з яких обслуговується одним оператором, і час циклу, який може відрізнятись в певному діапазоні.

Через бажану швидкодію виробництва час циклу встановлюється мінімально можливий. Приймаючи більший цикл часу, виникає певний відсоток переробок, який повинен виплачуватись за рахунок підвищення рівня заробітної плати.

Було запропоновано розгалуження та зв'язану процедуру, яка відділяє, вибираючи або відкидаючи одиночні альтернативи.

Після цього, обчислюючи нижню і навіть верхню межі, вирішує відповідні приклади задачі балансування сладальної лінії.

Також було розглянуто моделі для одного або кількох продуктів, де станції, які будуть встановлені на конвеєрі, вибираються з безлічі нетипових станцій з різним обладнанням.

Проте, задача балансування спрощується шляхом прийняття фіксованої послідовності завдань (графік серійного пріоритету).

Іншою альтернативою розглядають змінне використання обладнання та мінімізування загальної вартості обладнання за певний час циклу.

Кожна станція забезпечена одним обладнанням, вибраним із доступного набору типів.

Кожен тип має індивідуальні витрати та індивідуальний вплив на час роботи. Отже, виникають дві проблеми:

- необхідно встановити і забезпечити обладнання різним числом станцій;
- завдання повинні бути розподілено специфічно до станцій, беручи до уваги обмеження. Деякі завдання можуть виконуватися лише з конкретною підмножиною типів обладнання.

Можливо представити точний евристичний алгоритм вирішення проблеми. Перший варіант — це філійна та пов'язана процедура, яка базується на схемі, орієнтованій на завдання та використовує стратегію MLB.

Супроводжуючи призначення першого завдання на нову станцію, вибирається обладнання для цієї станції.

Нижні межі обчислюються шляхом відкидання обмежень пріоритету та цілісності графу технологічних операцій. А також розгляду сукупних обмежень часу циклу.

Відгалужена процедура здатна вирішувати проблеми помірному розміру, до 30 завдань та не більше 10 типів обладнання.

Тому запропоновано евристичну версію процедури, яка пропускає вузли дерева розмежування та пов'язаного з ним дерева, керованого заданим користувачем параметром.

Було пропоновано процедури відгалуження та скорочення для базової та узагальненої системи робочих станцій. А також алгоритми, де режим роботи, що визначає час виконання завдання та вартість обладнання, повинен бути обраний для кожного завдання.

Проблема паралельності станцій є особливим випадком вищезгаданої проблеми вибору обладнання.

Тому проблема паралельності станцій може бути вирішена методами, викладеними вище, і може бути об'єднана з проблемою вибору обладнання без зміни моделі.

2.4 Синхронізована лінія передачі

Синхронізована лінія передачі складається з послідовності автоматизованих станцій, які виконують блоки завдань послідовно, та завдання для кожного паралельного блоку з використанням спеціалізованого обладнання (голівки шпинделя).

Об'єднання завдань блокам підпадає під обмеження графу технологічних операцій. Час блокування визначається максимальним завданням часу в блоці та додатковим часом для операцій перенесення.

Цільовою функцією є мінімізація вартості лінії життєвого циклу. Це відбувається за допомогою позбавлення від постійних витрат на станцію та додаткових витрат на блок, з урахуванням часу циклу.

Проблема вирішується за допомогою методом розкладання на основі змішаної цільної програми та стохастичною евристичною процедурою.

Було розглянуто варіацію проблеми з блоками станції, що виконуються одночасно, і адаптують підхід знайдення найкоротшого шляху.

Проблема вибору обладнання еквівалентна проблемі вибору працівників, де працівники різної кваліфікації, з точки зору швидкості виробництва, або якості, доступні і оплачуються відповідно до їх кваліфікації [47].

2.5 Паралельні станції та завдання

Встановлення паралельних станцій спричиняє додаткові фіксовані витрати. Для оцінки нового дизайну ліній необхідно введення додаткових метрик.

Було представлено підхід, де допускається дублювання існуючих станцій. Мета полягає в тому, щоб звести до мінімуму витрати на оплату праці, які складаються з постійних витрат на дублювання станцій, регулярних витрат на оплату праці та витрат

на понаднормові години, які виникають, якщо реалізований цикл перевищує бажаний час циклу.

Запропонована галузева процедура присвоєння запитів до станцій і вирішення питання про стан дублікацій станцій.

Було розглянуто поняття паралельних завдань. Довгострокові завдання вважаються розбитими на більш короткі завдання, які мають такі ж пріоритетні відносини, як і оригінальні.

Тепер ці паралельні завдання призначаються для різних станцій для того, щоб отримати бажаний баланс для бажаного часу циклу взагалі та покращити ефективність лінії.

Проте через неподільність завдань оригінальне завдання може виконуватися лише однією станцією за цикл.

Це відбувається по черзі. Завдання виконується на кожній з відповідних станцій, тим самим приймаючи місцеві цикли, що відрізняються від оригінального.

Мета полягає в тому, щоб звести до мінімуму сукупні витрати, які складаються з витрат на об'єкти та витрати на оплату праці, визначених у паралельній станції.

Витрати на установку виникають, навіть коли в паралельній справі не потрібно додаткових станцій, оскільки необхідно встановити транспортну систему, яка приводить до тимчасових порушень часу циклу.

Пропонується дати математичну модель проблеми і вирішувати її за допомогою відгалуженої процедури. У цьому алгоритмі паралельні завдання не повинні бути віднесені до більш ніж двох станцій, а час завдання підрозділяється порівну між двома частинами завдання.

Інший варіант розглядає паралельні завдання і станції, а також мертві часи, тобто час, необхідний для транспортування заготовок з однієї станції до наступного, під час якого завдання не можуть бути виконані.

Для серійної лінії мертвий час зменшує робочий час циклу. Якщо, наприклад, час циклу становить 10, а мертвий час становить 1, працівники повинні платити за 10 одиниць часу, а час завдання не повинен перевищувати 9 для кожної серійної станції.

З паралельними станціями непродуктивна частина часу циклу може бути зменшена.

Якщо розглянути наведений вище приклад, то станції, що дублюються, мають місцевий час циклу 20 і продуктивний час становить 19. Тобто у прикладі паралельні станції використовують 95%, а серійні станції — лише 90% часу циклу продуктивно.

Було запропоновано процедуру, яка ґрунтується на підходах ДП для вирішення викладеної проблеми.

Розглядається модель з паралельними станціями, яка передає постійні витрати на додаткове обладнання і спрямована на мінімізацію загального простою. Пропонується загальна процедура, заснована на пріоритеті, для вирішення цієї проблеми.

Інший підхід до дублювання станцій полягає в тому, щоб поставити їх поруч у серійній лінії. Це може бути корисним через обмеження простору або коли потрібна менш складна транспортна система. У цьому випадку буфери потрібні перед та за дубльованими станціями.

Розглянемо приклад з 4 серійними станціями 1, 2a, 2b та 3, причому 2a та 2b є дублікатами один одного.

Коли станція 1 обробляє свою роботу на заготовці, тоді як станції 2a і 2b все ще зайняті, заготовка надходить у буфер. У наступному циклі 2b подають із заготовки з буферу, а до 2a безпосередньо доставляється зі станції 1.

Вихід станції 2b переноситься на таку станцію 3, а 2a переміщується через 2b до зовнішнього буфера.

Було представлено стохастичний підхід до цієї проблеми, де випадкові несправності, час відновлення та час обробки варіюються.

У випадковому випадку дублюючі станції часто не починають і не закінчують свої процеси одночасно. Поки перша із станцій зайнята, друга станція не може бути завантажена з наступною заготовкою.

Чотири принципи подання вхідних заготовок на дублюючі станції порівнюються моделюванням.

Інший тип паралелі — призначити більш ніж одному оператору станцію (декілька комплектацій).

Ефект, подібний до паралельних станцій, складається з об'єднання станцій у великі одиниці (агрегатні станції), які експлуатуються командами операторів.

Агрегатні станції мають доступний вплив вихідного часу циклу і оператори можуть змінюватись, що збільшує рівень задоволеності роботою.

Іноді команди відповідають за повний продукт, такий, що “лінія” перетворюється на лише одну агрегатну станцію.

Тобто, збірні системи, орієнтовані на команду, можуть усунути основні характеристики традиційних ліній збірки, таких як суворий розподіл праці та темп роботи [48].

2.6 Обмеження розподілення завдань за робочими станціями

Кілька типів обмежень можуть скоротити присвоєння завдань станціям. Позитивні обмеження викликають потребу в лінійній станції з різних боків.

Особливо великі заготівки, такі як автобуси або вантажні автомобілі, опрацьовуються станціями, які виконують лише завдання на одній зі сторін, тому що рух навколо заготовки коштуватиме багато часу.

Це призводить до обмежень, пов'язаних із завданням, оскільки завдання з лівого боку не повинні поєднуватися з правими сторонами.

Загалом, завдання, які не можуть бути призначені для однієї станції, називаються несумісними.

Було представлено двосторонні збірні лінії, де пара робочих станцій розташовані навпроти один одного ліворуч і праворуч від лінії. Кожна пара станцій одночасно працює над одним елементом.

Завдання згруповані по стороні транспортного засобу, над якою вони можуть виконувати роботу.

Отже, існують завдання з правої сторони, наприклад монтаж правого колеса, завдання з лівого боку, завдання, які можна призначити на будь-якій стороні лінії, наприклад, монтаж радіо, а також завдання, які одночасно повинні виконувати обидві парні партії, наприклад, встановлення заднього сидіння.

Звичайно, двостороння лінія може мати безпілотні станції. У цьому випадку деякі станції не мають протилежного супутника.

Було показано, що для певного часу циклу в деяких випадках, залежно від обмежень пріоритету, двостороння лінія вимагає менше станцій, ніж традиційна однобічна лінія, але ніколи не вимагає більше станцій.

Важливе зауваження полягає в тому, що для пари протилежних станцій необхідно поважати обмеження пріоритетів призначених завдань.

Якщо ми розглянемо завдання 3 з попередниками 1 і 2, при чому час виконання завдання $t_1 < t_2$, завдання 1 та 3 призначаються для однієї станції, а завдання 2 призначається для протилежного.

Тоді завдання 3 не повинна починатися, перш ніж остання станція завершить завдання 2.

Таким чином, простой може відбутися навіть на початку або в середині циклу, коли одна станція повинна чекати, не починаючи завдання, поки протилежна станція не закінчить попередників [49].

Можлива реалізація модифікованої версії евристичного правила пріоритету у програмі, яка дозволяє користувачам перенаправляти деякі завдання на певні станції.

Це потрібно, наприклад, коли для деяких завдань потрібне спеціальне обладнання, доступне лише на певній станції. Такі пов'язані зі станціями обмеження можуть бути легко включені в моделі та процедури для задачі баласнування складальної лінії, оскільки вони просто перешкоджають присвоєнню певного завдання для підмножини станцій.

Процедура ДП для вирішення проблеми несумісності між задачами та деякими комбінаціями фіксованих станцій задач була вирішена. Мета полягала у згладженні робочого навантаження серед певної кількості станцій (вертикальне балансування).

Представлено генетичний алгоритм, який намагається мінімізувати кількість станцій у двосторонньому алгоритмі. Було описано дві альтернативні цілі для цієї проблеми: перша спроба об'єкту присвоювати завдання, які безпосередньо пов'язані з графіком пріоритету, на ті ж станції.

Останній намагається уникати призначення відповідних завдань на протилежні станції або, якщо це неможливо, максимально збільшити проміжок часу між закінченням часу одного завдання та початком його наступника на протилежній станції.

Процедура основана на принципі пріоритетів, який базується на задачах групування. На реальній однобічній лінії з заданою кількістю станцій є різні типи обмежень присвоєння:

- завдання пов'язані: деякі завдання повинні бути призначені для однієї і тієї ж станції, і, таким чином, можуть бути об'єднані в одне завдання;
- позиція пов'язана: деякі завдання можуть виконуватися лише з лівої сторони або правої сторони великої, незмінної заготовки.

Крім того, деякі завдання повинні бути виконані у верхній або нижній частині заготовки. Завдання лівої та правої сторони, а також верхнє і нижнє завдання не повинні бути віднесені до тих самих станцій, відповідно.

Це має бути зроблено без використання двосторонньої лінії, тобто деякі станції повинні бути розташовані з лівого боку конвеєрної стрічки, інші — з правої сторони, крім того, деякі станції вимагають, щоб робітник працював на висоті, інші повинні дозволити працювати внизу.

Щоб вирішити цю велику проблему реального світу, було представлено змішану цільову лінійну програму та двофазну евристичну процедуру (фаза 1: урізаний підхід DP, фаза 2: вдосконалення за допомогою локального пошуку) [50].

Мета полягає в згладжуванні навантажень станцій таким чином, щоб всі часові станції мали значення з заданого інтервалу цілей.

Висновки до другого розділу

В розділі були описані парадигма генетичних алгоритмів, основні поняття та оператори.

Описано загальний принцип дії генетичних алгоритмів, послідовність симулювання в класичних алгоритмах.

Приведена модифікація генетичних операторів з урахуванням зазначених в попередньому розділі недоліків. В подробицях описані модифікації операторів схрещування, мутації, інверсії, селекції.

Описано застосування генетичного алгоритму групування до задачі балансування складальної лінії.

Показані модифікації алгоритму для врахування графу технологічних операцій.

Описані нюанси складальних ліній, що впливають на роботу алгоритму — такі як обмеження розподілення завдань по робочим станціям, а також паралельні завдання.

3. АНАЛІЗ ТА ОБҐРУНТУВАННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ

Програмний модуль для вирішення задачі балансування складальної лінії було написано на мові C++. Розроблений модуль може бути включено до складу більшої системи.

Для демонстрації роботи та тестування модуля було створено демонстраційний додаток.

Під час написання були використані Microsoft Visual Studio 2017 а також бібліотека SFML.

3.1 Мова C++

Для розробки ефективного та простого у підтримці програмного модулю було використано мову C++.

Мова програмування C ++ забезпечує модель пам'яті та обчислень, яка тісно відповідає більшості комп'ютерів. Крім того, вона забезпечує потужні та гнучкі механізми абстракції; тобто конструкції мови, які дозволяють програмісту вводити та використовувати нові типи об'єктів, які відповідають поняттям програми.

Таким чином, C ++ підтримує стилі програмування, які покладаються на досить пряму маніпуляцію апаратними ресурсами, щоб забезпечити високий ступінь ефективності плюс стилі високого рівня програмування, які спираються на визначені користувачем типи, щоб забезпечити модель даних та обчислення, що ближче до погляду людини на завдання, виконувані комп'ютером.

Ці стилі вищого рівня програмування часто називають абстракцією даних, об'єктно-орієнтованим програмуванням і загальним програмуванням.

Мова програмування C++ має історію, починаючи з 1979 р., Коли Бьєрне Страуструп отримав звання доктора філософії.

Одна з мов, яку Страуструп взяв за основу, і з якою мав можливість працювати, була мова, що називається Simula. Вона, як випливає з назви, призначена головним чином для моделювання.

Мова Simula 67 — це варіант, з яким працював Страуструп — розглядається як перша мова, що підтримує парадигму об'єктно-орієнтованого програмування.

Страуструп встановив, що ця парадигма була дуже корисною для розробки програмного забезпечення, однак мова Simula була надто повільна для практичного використання.

Незабаром після цього він почав працювати над “С з класами”, який, як випливає з назви, мав на увазі суперсет мови С.

Його метою було додати об'єктно-орієнтоване програмування на мову С, яка була і залишається мовою, яка добре поважається через її переносимість без шкоди для швидкості або низького рівня функціональності.

У його мові входили класи, основне успадкування, вбудовані дані, аргументи функції за умовчанням, а також перевірка сильного типу на додаток до всіх функцій мови С.

Першим компілятором С з класами називався Sfront, який був отриманий з компілятора С, який називається CPre.

Це була програма, призначена для перекладу С з класом на звичайний С.

Дуже цікавий момент, який варто відзначити, полягає в тому, що Sfront написано переважно в С з класами, що робить його компілятором, який самостійно розміщує себе (компілятор, який може компілювати себе).

Компілятор Sfront пізніше буде відмовлено в 1993 році, після того, як складно інтегрувати в неї нові функції, а саме С ++ виключення. Тим не менше, Sfront справила величезний вплив на реалізацію майбутніх компіляторів та операційну систему Unix.

У 1983 році назва мови була змінена з C з класами на C++.

Оператор ++ в мові C є оператором для збільшення змінної, що дає деяке уявлення про те, як Струstrup розглядав мову.

У цей час було створено багато нових функцій, найпомітнішими з яких є віртуальні функції, перевантаження функції, посилання на символ, амперсанд, ключове слово const і однорядкові коментарі за допомогою двох прямих косу рисунків (це функція, взята з мови BCPL).

У 1985 році було опубліковано посилання Струstrup на мову “Мова програмування C++”. У тому ж році C++ було реалізовано як комерційний продукт. Мова ще не була офіційно стандартизована, що робить книгу дуже важливою. У 1989 році мова була оновлена, щоб включити захищені та статичні члени, а також спадщину з кількох класів.

У 1990 році було випущено “Довідник з анотованого C++”. У тому ж році компілятор Borland’s Turbo C++ буде випущений як комерційний продукт.

Компілятор Turbo C++ додав безліч додаткових бібліотек, які мали б значний вплив на розвиток C++. Хоча останній стабільний реліз Turbo C++ був у 2006 році, компілятор все ще широко використовується.

У 1998 році Комітет з стандартизації C++ опублікував перший міжнародний стандарт для C++ ISO / IEC 14882: 1998, який буде неформально відомий як C++ 98. Сказано, що анотований довідник C++ має великий вплив на розробку стандарту.

Також була включена стандартна бібліотека шаблонів, яка розпочала концептуальний розвиток у 1979 році. У 2003 році Комітет відреагував на численні проблеми, про які повідомлялося зі своїм стандартом 1998 року, і відповідно переглянули його. Змінена мова була названа C++ 03.

У 2005 році Комітет з стандартизації C++ випустив технічний звіт (названий TR1) з детальними характеристиками, які вони планували додати до останнього стандарту C++.

Новий стандарт був неофіційним названим C ++ 0x, оскільки його очікували випустити колись до кінця першого десятиліття. За іронією долі, новий стандарт не буде випущений до середини 2011 року.

До цього часу було випущено кілька технічних звітів, і деякі компілятори почали додавати експериментальну підтримку нових функцій.

У середині 2011 року був завершений новий стандарт C ++ (dubbed C ++ 11). Проект бібліотеки Boost зробив значний вплив на новий стандарт, а деякі нові модулі були отримані безпосередньо з відповідних бібліотек Boost.

Деякі нові функції включали підтримку регулярного вираження (докладніше про регулярні вирази можна знайти тут), всебічну бібліотеку рандомізації, нову бібліотеку часу C ++, підтримку атомів, стандартну бібліотеку рішень (яка до 2011 року не мала C і C ++), новий для синтезу циклу, що забезпечує функціональність, подібну до форм кожного циклу в деяких інших мовах, автоматичне ключове слово, нові класи контейнерів, краща підтримка профспілок та списки ініціалізації масиву та варіаційні шаблони.

Мова C ++ була розроблена, щоб забезпечити гнучкість та ефективність програм C для системного програмування разом із можливостями Simula для організації програми (звичайно називається об'єктно-орієнтованим програмуванням).

Було зроблено велике увагу, що технології програмування вищого рівня від Simula можуть бути застосовані до домену системного програмування.

Тобто механізми абстракції, надані C ++, були спеціально розроблені для застосування програмних завдань, що вимагали найвищої ефективності та гнучкості. Підтримка загальних програм виникла пізно як явна мета.

Мета C ++ полягала у поліпшенні якості програм, отриманих шляхом спрощення використання дизайну та програмного забезпечення та прийнятності. Більшість з цих методів мають свій корінь в Simula, як правило, обговорюються під ярликами об'єктно-орієнтованого програмування та об'єктно-орієнтованого дизайну.

Проте метою було завжди підтримувати різні стилі дизайну та програмування. Це суперечить думці щодо дизайну мовлення, яка намагається спрямувати всю систему на єдиний сильно підтриманий і примусовий стиль (парадигма).

Ці правила повинні розглядатися в контексті, створеному з більш загальних цілей. Зокрема, прагнення до високої сумісності C, безкомпромісної ефективності та негайного реагування в реальному світі вимагає повної безпеки, повної загальної та абстрактної краси.

З Simula C ++ запозичив поняття визначених користувачем типів та ієрархії класів.

Проте в Simula та багатьох подібних мовах існують принципові відмінності в підтримці, наданій для визначених користувачем типів і для вбудованих типів. Наприклад, Simula не дозволяє об'єктам визначених користувачем типів виділяти в стек і безпосередньо звертатися до нього.

Замість цього всі об'єкти класу повинні бути виділені в динамічну пам'ять і доступні через покажчики.

І навпаки, вбудовані типи можуть бути справді локальними (виділено стек-кадр), не можуть бути виділені в динамічну пам'ять і не можуть бути посилання на покажчики. Ця різниця у вживанні вбудованих типів і типів, визначених користувачем, мала серйозні наслідки для ефективності.

Наприклад, коли представлена як посилання на об'єкт, що виділяється в динамічній пам'яті, визначений користувачем тип — наприклад, комплекс — нараховує витрати часу та простору, які вважаються неприйнятними для тих програм, для яких C++ був призначений.

Також різниця в стилі використання не дозволить одноманітно трактувати семантично подібні типи в загальному програмуванні.

Підтримуючи велику програму, програміст повинен постійно вносити зміни на основі неповних знань і дивлячись лише на невелику частину коду [51].

Отже, C++ забезпечує класи, простору імен та контроль доступу, щоб допомогти локалізувати проектні рішення. Деякі залежності залежностей неминучі на мові, призначеній для однозарядної компіляції.

Наприклад, в C++ змінна або функція не може бути використана, перш ніж вона буде оголошена.

Проте правила для імен класів та правил розподілу переважань були зроблені незалежно від порядку декларації, щоб мінімізувати плутанину та помилку.

Фундаментальна властивість комп'ютерів у широкому використанні залишалася надзвичайно постійною: пам'ять — це послідовність слів або байтів, індексована цілими числами, що називаються адресами.

Сучасні машини, скажімо, розроблені протягом останніх 20 років, мають тенденцію підтримувати безпосередньо поняття стеків викликів функції.

Крім того, всі популярні машини мають деякі важливі об'єкти, такі як введення-виведення, які добре не входять у звичайну байтову або словоорієнтовану модель пам'яті або обчислення.

Ці об'єкти можуть вимагати спеціальних інструкцій з машин або доступ до “пам'яті” місць з властивою семантикою.

У будь-якому випадку, з точки зору мови вищого рівня, використання цих об'єктів є брудним і специфічним для машинобудування.

Мова C++ на сьогоднішній день є найбільш успішною, яка надає програмісту модель програмування, яка точно відповідає моделі машини.

Також C++ надає мовні та незалежні від машиноорганізма поняття, які безпосередньо відносяться до ключових апаратних уявлень: символи для використання байтів, цілі числа для використання слів, покажчики для використання адресних механізмів, функції абстракції програми та відсутність обмежувальних мовних функцій так що програміст може маніпулювати неминучими незрозумілими конкретними деталями обладнання.

Чистий ефект полягає в тому, що C++ відносно легко вивчати і використовувати в тих областях, де певні знання реальної машини є вигодою. Крім того, C++ досить легко застосувати, що він став майже доступним у всьому світі.

Мову C++ використовують сотні тисяч програмістів практично у кожному домені додатків. Це використання підтримується приблизно десятком незалежних реалізацій, сотнями бібліотек, сотнями підручників, декількома технічними журналами, численними конференціями та незліченними консультантами.

Навчання та навчання на різних рівнях є широко доступними. Ранні додатки, як правило, володіли сильним смаком системного програмування.

Наприклад, кілька основних операційних систем були написані на C++ і багато інших мають основні частини, виконані в C++.

Мова C++ була розроблений таким чином, що кожна функція мови використовується в кодї при важких обмеженнях часу та простору. Це дозволяє використовувати C++ для драйверів пристроїв та іншого програмного забезпечення, які покладаються на пряме маніпулювання апаратними засобами в режимі реального часу.

У такому кодексі передбачуваність продуктивності, принаймні така ж важлива, як сила швидкості. Часто така компактність отриманої системи.

Більшість програм мають розділи коду, які є критичними для прийнятної продуктивності. Однак найбільша кількість коду не в таких розділах.

Для більшості коду, ремонтпридатність, простота розширення та легкість тестування є ключовим.

Підтримка C++ з цих проблем призвела до її широкого використання, де необхідна надійність, і в тих областях, де вимоги значно змінюються з часом.

Наприклад, банківська справа, торгівля, страхування, телекомунікації та військові додатки. Протягом багатьох років центральний контроль міжміського

телефонного зв'язку США покладався на C++, і кожні 800 дзвінків (тобто дзвінок, сплачений дзвінковою стороною) був направлений програмою C++.

Багато таких програм є великими і довговічними. У результаті стабільність, сумісність та масштабованість були постійними проблемами при розробці C++. Millionline C++ програми не рідкісні.

Як і в C, C++ спеціально не розроблено з урахуванням чисельного обчислення. Проте численні, наукові та інженерні обчислення виконуються на C++.

Основна причина цього полягає в тому, що традиційна чисельна робота часто повинна поєднуватися з графікою та з розрахунками, що спираються на структури даних, які не входять у традиційну Фортранську форму.

Графічні та користувацькі інтерфейси — це області, в яких сильно використовується C++. Все це вказує на те, що може бути сильнішою силою C++: його здатність ефективно використовуватися для додатків, які вимагають роботи в різних областях застосування.

Цілком звичайним є пошук додатків, що включають в себе локальні та широкомасштабні мережі, цифри, графіку, взаємодію з користувачами та доступ до бази даних.

Традиційно такі області застосування були розглянуті окремо, і їх найчастіше обслуговують різні технічні спільноти, що використовують різні мови програмування. Проте C++ широко використовується у всіх цих областях.

Крім того, він здатний співіснувати з фрагментами коду та програмами, написаними іншими мовами. C++ широко використовується для навчання та досліджень.

Це здивувало деяких, котрі — правильно — вказують на те, що C++ не є найменшою або найчистішою мовою, яка коли-небудь розроблена [52].

Однак C++ — достатньо чистий для успішного навчання основних понять, — реалістичний, ефективний і достатньо гнучкий для вимогливих проєктів, — достатній

для організацій та співробітництва, що спираються на різноманітні середовища розробки та виконання, — достатньо комплексний, щоб бути інструментом навчання сучасні концепції та техніки, а також — достатньо комерційні, щоб бути засобом покладання того, що вивчається, для неакадемічного використання.

Завдяки стандартам ISO, C++ також добре вказується, стабільний і підтримується стандартною бібліотекою.

3.2 SFML

Бібліотека SFML — це простий у користуванні та портативний API, написаний на C++. Він може бути описаний як об'єктно-орієнтований SDL.

Бібліотека SFML складається з модулів, щоб бути максимально корисним для всіх. Можна використовувати SFML як мінімалістську систему вікон, щоб використовувати OpenGL, або як повну мультимедійну бібліотеку, повну функції для створення відеоігор або мультимедійних програм.

Остання версія SFML наразі доступна і повністю функціональна в Windows (10, 8, 7, Vista, XP), Linux та macOS. SFML працює як на 32, так і на 64-бітних системах.

Якщо старіші версії Windows потребують підтримки, слід використовувати можливість використання SFML 2.0 (зобов'язання щодо видалення Windows 9x та подібних).

Оскільки SFML 2.2 також існує експериментальна підтримка iOS та Android, які протягом багатьох років мають чудову форму і повинні працювати досить стабільно.

Бібліотеку SFML реалізовано на C++. Тим не менш, було створено декілька прив'язок для інших мов, які дозволяють використовувати SFML з C, C#, C++ / CLI, D, Ruby, OCaml, Java, Python і VB.NET.

Бібліотека SFML залежить від декількох інших бібліотек, тому перед початком компіляції необхідно встановити свої файли розробки.

У Windows та macOS всі необхідні залежності надаються безпосередньо з SFML, тому не потрібно нічого завантажувати / встановлювати. Компіляція буде працювати з коробки.

Проте в Linux немає нічого, і SFML залежить від власної установки бібліотек, від яких залежить. Список того, що потрібно встановити перед складанням SFML:

- pthread
- opengl
- xlib
- xrandr
- udev
- freetype
- openal
- flac
- vorbis

Точне ім'я пакетів залежить від кожного розповсюдження. Важливо встановлювати необхідну версію розробки цих пакетів.

SFML також має внутрішні залежності: аудіо та вікно залежать від системи, а графіка залежить від системи та вікна.

Для того, щоб використовувати графічний модуль, треба зв'язуватися з графікою, вікном та системою (порядок зв'язків з GCC).

Для того, щоб статично зв'язати SFML, потрібно налаштувати середовище збирання для зв'язку зі статичними бібліотеками SFML. Статичні бібліотеки — це суфікс -s, наприклад sfml-graphics-s.

Далі слід додати параметр SFML_STATIC до опції препроцесора, і слід обов'язково зв'язати бібліотеки налагодження (-d суфікс) у режимі налагодження та бібліотеки випуску (no -d suffix) у випуску режим

У минулому SFML включив у Windows усі свої залежності в бібліотеки SFML. Однак це було змінено, щоб усунути численні проблеми та отримати загальноприйнятту поведінку (повне обговорення).

Тепер SFML поводитьься так само як в Linux, так і в ОС Windows, що, однак, означає, що потрібно пов'язати SFML з власними зв'язками при статичному зв'язуванні.

Оскільки залежності не є очевидними, ось список:

- sfml-window
- sfml-system
- opengl32
- winmm
- gdi32
- sfml-graphics
- sfml-system
- sfml-window
- opengl32
- freetype
- sfml-audio
- sfml-system
- openal32
- flac
- vorbisenc
- vorbisfile
- vorbis
- ogg
- sfml-network
- sfml-system

- ws2_32
- sfml-system
- winmm

Аудіо-модуль здатний відтворювати файли wav, ogg / vorbis та flac.

Формат MP3 наразі не підтримується, оскільки патенти та ліцензії на MP3 тільки нещодавно вичерпані.

Бібліотека SFML обробляє створення та введення в Windows, а також створює та керує контекстами OpenGL.

Він також забезпечує графічний модуль для просте апаратне прискорення 2D комп'ютерної графіки, що включає в себе перетворення тексту за допомогою FreeType, аудіо модуля, який використовує OpenAL, і мережевий модуль для передачі базового протоколу керування передачею (TCP) та користувацького протоколу передачі даних (UDP).

Бібліотека SFML — це безкоштовне програмне забезпечення з відкритим кодом, надане згідно з умовами ліцензії zlib / png.

Він доступний у Windows, Linux, macOS та FreeBSD. Перша версія v1.0 була випущена 9 серпня 2007 року, остання версія v2.5.1 була випущена 15 жовтня 2018 року.

Бібліотека SFML в першу чергу використовується розробниками ігор-любителів, невеликими незалежними розробниками відеоігор та компаніями-початками, що складаються з декількох розробників.

Оскільки SFML не вимагає писання великої кількості коду, він був прийнятий багатьма учасниками Ludum Dare.

У порівнянні зі старими бібліотеками, такими як Simple DirectMedia Layer (SDL) та Allegro, користувацька база SFML є відносно невеликою, але зростає. Станом на 10 травня 2018 р. Його сховище програмного забезпечення GitHub було зіграно на 4255 користувачів.

Бібліотека SFML використовується для навчання в університетах та наукових проектах.

Приклади створених відеоігор:

- «Atom Zombie Smasher», стратегія в реальному часі;
- «Chesster», гра-головоломка;
- «Cosmoscroll», безкоштовна космічна космічна космічна гра з відкритими вихідними кодами;
- «Crea», модульна 2D гра пісочниці;
- «HolySpirit», 3D ізометрична гра хак і коса праворуч;
- «Kroniax», мінімалістський бічний скроллер, і перша гра SFML для Android;
- «M.A.R.S.», багатокористувацька стрілялка;
- «Moonman», гра піксельного мистецтва розвідки пісочниці;
- проект «Black Sun», ретро 2D відеопрोगравач, що прокручує бічну смугу;
- «Vagante», RPG гра;
- «Open Hexagon», безкоштовний кеш Super Hexagon із відкритим кодом;
- «Limit Theory», нескінченна, процедурна простір гри;
- «Postmortem»: треба померти, розповідна пригодницька гра;
- «Pioneers», покрокова пошукова гра з деякими елементами RPG;
- «Zloxx», 2D платформер дій;
- «KeeperRL», симулятор підземелля з елементами шаленої та RPG;
- «Норе», наведіть вказівник миші на пригодницьку гру (наприклад, Myst);
- «Tas Wars», головоломка RPG, яка втілює гномів проти гоблінів;
- «Extreme Tux Racer», безкоштовна арктична гоночна версія з відкритим кодом, що включає Tux (використовуючи SFML з версії 0.7).

3.3 Visual Studio 2017

Вбудоване середовище розробки Visual Studio — це креативна панель запуску, яку можна використовувати для редагування, налагодження та створення коду, а потім публікувати додаток. Інтегроване середовище розробки (IDE) — це багатofункціональна програма, яка може використовуватися для багатьох аспектів розробки програмного забезпечення. Понад стандартний редактор і налагоджувач, який забезпечують більшість IDE, Visual Studio включає в себе компілятори, інструменти для завершення коду, графічні розробники та багато інших функцій для полегшення процесу розробки програмного забезпечення.

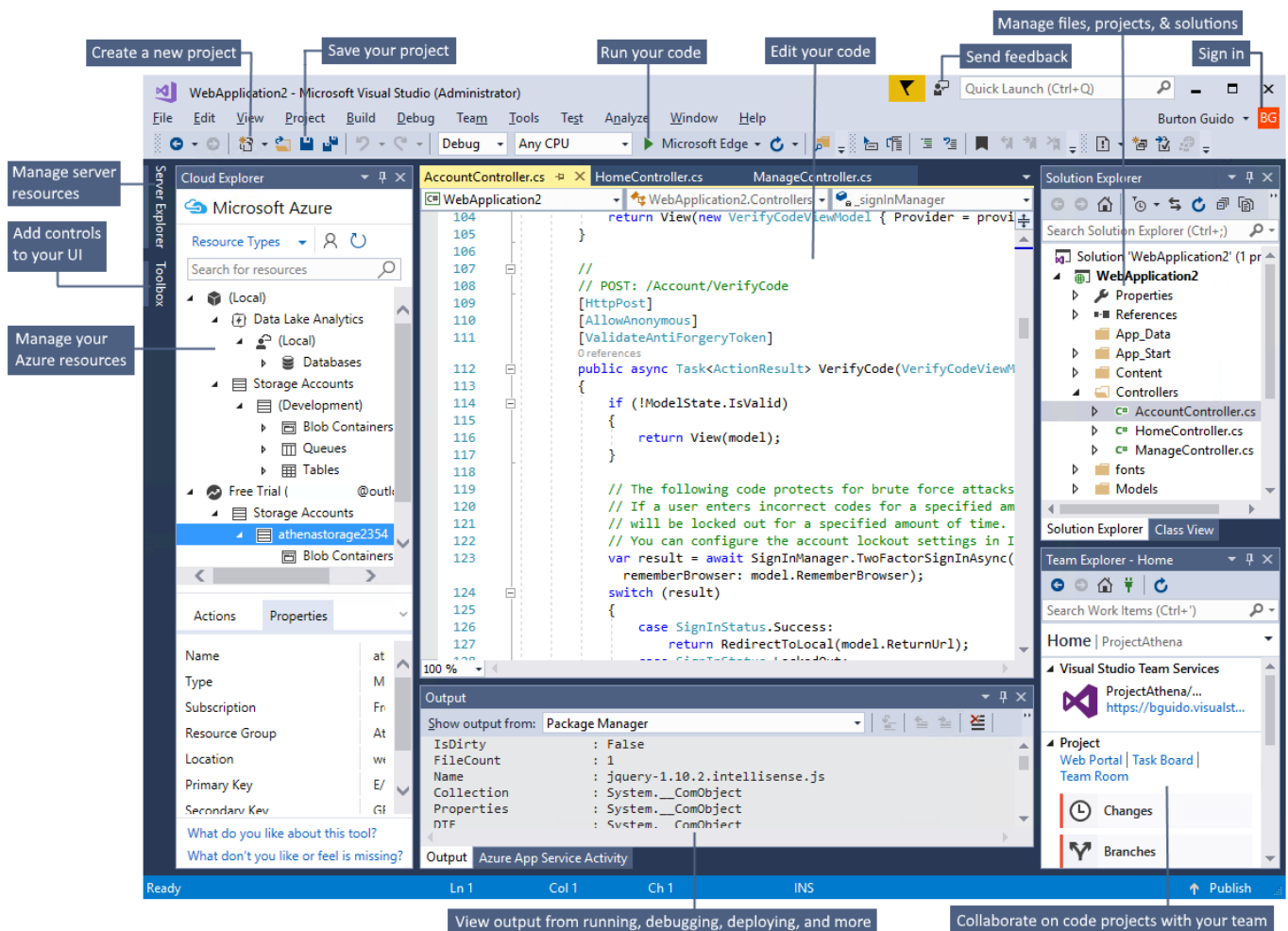


Рисунок 3.1 — Інтерфейс Visual Studio

На рисунку 3.1 представлено інтерфейс Visual Studio з відкритим проектом та кількома основними віконцями, які найчастіше використовуються:

- вікно Solution Explorer (вгорі праворуч) дозволяє переглядати, переміщатись та керувати кодами файлів. Solution Explorer може допомогти організувати ваш код, об'єднавши файли в рішення та проекти;

- вікно редактора (в центрі) відображає вміст файлу. Тут можливо редагувати код або спроектувати користувальницький інтерфейс, наприклад вікно з кнопками та текстовими полями;

- вікно Output (знизу в центрі) — це місце, де Visual Studio надсилає сповіщення, такі як налагодження та повідомлення про помилки, попередження компілятора, публікація повідомлень про статус тощо. Кожне джерело повідомлень має свою вкладку;

- вікно Team Explorer (знизу справа) дозволяє відстежувати робочі елементи та спільно використовувати код з іншими користувачами за допомогою технологій керування версіями, таких як Git та Team Control Version Control (TFVC).

Visual Studio доступна для Windows і Mac. Visual Studio for Mac має багато схожих функцій, таких як Visual Studio 2017, і оптимізовано для розробки крос-платформних та мобільних додатків.

Існує три випуски Visual Studio 2017: Community, Professional і Enterprise. Див. Порівняйте середовища розробки Visual Studio 2017, щоб дізнатись, які функції підтримуються у кожному випуску.

Рефакторинг включає в себе такі операції, як інтелектуальне перейменування змінних, витягування одного чи декількох рядків коду в новий метод, зміна порядку параметрів методу тощо.

Технологія IntelliSense — це термін для набору функцій, який відображає інформацію про ваш код безпосередньо в редакторі та, у деяких випадках, для вас пише невеликі коди.

Це схоже на наявність базової документації в редакторі, яка заощаджує вас від необхідності шукати інформацію про тип в іншому місці. Функції IntelliSense залежать від мови.

Розумна попередня вибірка визначатиме “тип цілі”, необхідний для позиції в коді, і буде попередньо вибрати елементи у списку доповнень IntelliSense, що відповідають цьому типу.

Це прискорює ваш типографічний потік і усуває тягар необхідності з’ясувати очікуваний тип у певному місці.

Фільтрування IntelliSense дозволяє фільтрувати список завершенень за категоріями; Наприклад, можливо відфільтрувати методи розширення або переглядати лише події.

Ця функція підвищує продуктивність, при роботі на великій кодовій базі, де є багато елементів у списку завершенень або при роботі з незнайомим кодом.

Нарешті, цей випуск забезпечує абсолютно новий досвід роботи з XAML IntelliSense, який допомагає розробникам швидко і правильно зв’язуватися і переглядати лише релевантну інформацію.

Цей розумний досвід завершення включає завершення прив’язування подій, шляхів і функцій з x: Bind; підтримка відповідності camelCase (наприклад, “RTB” буде завершена як “RichTextBox”); і автоматичне завершення префікса простору імен.

Аналіз коду Visual Studio 2015 представила функцію аналізу живого коду, яка дає змогу створювати зворотній зв’язок у своєму коді.

Це дає змогу вивчати проблеми рано, перш ніж вони створюватимуться, а не накопичувати набір проблем, які ніколи не можна вирішити.

Для вирішення помилок, виявлених під час аналізу живого коду, ви використовуєте меню лампочки або ярлик “Ctrl +” для доступу до виправлення та редагування коду.

Середа розробки Visual Studio 2017 RC використовує аналіз життєвого аналізу та виправлення коду на наступний крок шляхом посилення наявності рефакторингів та виправлень коду та введення аналізаторів стилів коду, які визначають проблеми стилю в коді, як тільки вони набираються.

Середа розробки Visual Studio 2015 включає в себе деякі основні рефакціонування: метод вилучення або інтерфейс, підпис у режимі зміни, вбудована тимчасова змінна, введення локальної змінної та видалення непотрібного використання та імпорту.

Середа розробки Visual Studio 2017 RC розширює набір рефакторингів і виправлень, щоб допомогти підтримувати читабельну кодову базу та каталізувати робочі процеси розробки.

Наприклад, значна кількість розробників спочатку записує всі свої класи, інтерфейси та інші типи в один файл, а потім витягує кожний тип у файл з відповідним ім'ям пізніше.

Середа розробки Visual Studio 2017 RC прискорює цей процес за допомогою опції рефакторингу “Перемістити тип у відповідний файл”.

Інші рефакціонування, включають:

- тепер програма установки пропонує можливість завантажувати всі файли перед початком інсталяції;
- підвищення продуктивності під час розвантаження або перезавантаження проекту та перемикання гілок;
- час завантаження рішення може бути покращено шляхом відключення автоматичного відновлення документів;

- значні поліпшення роботи тесту під час роботи великих рішень з декількома тестовими проектами;
- тепер Visual Basic надає значне покращення продуктивності;
- професійна профілювання тепер пропонує можливість призупинити або відновити збір даних і додано новий інструмент відстеження об'єктів .NET;
- удосконалення інструмента “Використання процесора” для профілізації ефективності;
- додавання нових функцій продуктивності, таких як очищення коду, інвертування рефакторингу, перехід до блоку блокування, підтримку Multi-Caret та нові профілі клавіатури;
- можливість обрати цільовий екземпляр під час налагодження розширень;
- запуск налагоджувач знімків безпосередньо на сторінці підсумки опублікування;
- представили F # 4.5, нову мовну версію, яка повністю підтримує Span <'T> і містить істотні покращення в стеку слідів для коду асинхронного коду;
- підвищення продуктивність і додавали нові функції в інструменти F #, такі як Ctrl + клацніть, щоб перейти до визначення;
- додатки до набору інструментів C ++ включають удосконалення оптимізатора SSA та компонувальника;
- додатки для розробки крос-платформної версії C ++ включають оновлення ClangFormat та шаблони налаштування у CMake і відкритій папці для MinGW, Linux та Windows;
- зміни до продуктивності C ++ включають в себе шаблони IntelliSense, підказки для швидкої інформації на макросах, перетворення в лайтбокси constexpr, кодування в кодї аналізу в редакторі тощо;
- покращення налагодження C ++ включає оновлення Just My Code та нові точки зупинки даних;

- існує безліч розробок та функцій JavaScript та TypeScript;
- тепер можливо керувати бібліотеками клієнта у своїх веб-проектах;
- додавання новий досвід окремих проектів Docker контейнера для веб-проектів ASP.NET Core;
 - тепер можете налаштувати тег для зображення Docker за допомогою оновлень, опублікованих у цьому випуску;
 - покращення Xamarin включає підтримку Xcode 9.4 та розумніших послідовних збігів Android;
 - можливість використовувати емулятор Google для Android разом з Hyper-V у Windows 10 квітня 2018 року оновлення;
 - додавання розділювальний вигляд редактора для дизайнера Xamarin.Android;
 - попередній перегляд Xamarin.Forms тепер має підтримку інструментів;
 - можливість використовувати крос-мову налагодження за допомогою Python 3.7.0rc1;
 - функція міграції локальних Azure функцій Azure для Azure тепер відображає нові значення;
 - з додатковою підтримкою функцій Azure тепер є новий цільовий хост в діалоговому вікні “Налаштувати безперервну доставку”;
 - поліпшення панелі підсумкових панелей Test Explorer тепер забезпечує більш інформативний статус тесту;
 - розширення Extensions для тесту .NET Test Adapter: виправлення змін та знецінення;
 - додавання нативної підтримки протоколу сервера мовлення;
 - підтримка секретів для проектів ASP.NET. NET Framework;
 - середовище Visual Studio тепер пропонує інструменти розробки .NET Framework 4.7.2 для підтримуваних платформ із включеним runtime 4.7.2;

- пакети автозавантаження Async затримуються до завершення завантаження та завантаження рішення;
- засоби розробки Visual Studio 2017 тепер підтримують робочий процес і включають в себе VSSDK;
- стан Git і TFS тепер правильно оновлюється для зовнішніх змін файлів у проектах .NET Core;
- включено .NET Core SDK 2.1.400;

Крім того, цей випуск вводить базовий аналіз коду та виправлення для XAML. Використовуючи той самий механізм лампочки в C # та Visual Basic, можливо сортувати та видаляти непотрібні назви та додавати відсутні прогами імен у файли XAML.

Підтримка послідовної, зрозумілої бази коду є складним завданням. Існують вагомні причини прагнути до читабельної кодової бази: якщо весь код виглядає послідовним, його простіше вбудувати в нові розробники; і якщо весь код виглядає послідовно, огляди коду можуть зосереджуватися на логіці, а не на форматі та стилі мінутей.

Середовище розробки Visual Studio 2017 RC встановлює спосіб налаштування стилю коду з вбудованими правилами стилю та індивідуальними умовними позначеннями.

Підтримка C ++ у VS2017 продовжує розвиватися, і це включає подальше дотримання стандартів, а також виправлення помилок. Розробники Active C ++ повинні обов'язково перевіряти зауваження щодо відповідності корпорації Майкрософт, оскільки нові зміни поведінки вплинуть на те, як раніше дозволений код розглядається в 15.9.

В цій версії вперше з'явилась нова функція “Імпорт / Експорт Конфігурації”, яка надає ряд переваг незалежно від того, чи є ви самотнім розробником або працює в налаштуваннях корпоративної команди.

З огляду на конфігурування VS2017 з його модульним інсталятором, з'явилась можливість зберігати безліч додатків для заощадження часу. Розробники команд можуть переконатися, що всі вони мають однакові робочі процеси. І всі розробники можуть мати можливість ввести їх конфігурацію у своє сховище джерела проекту, якщо вони цього бажають.

Користувачам NuGet-пакетів у VS2017 надаються важливі дані про безпеку: NuGet Client Policies.

Це дає змогу запобігти встановленню непідписаних пакетів NuGet; також можна включити білий список надійних авторів пакунків.

Важлива зміна для розробників, що використовують .NET Core з VS2017, була розроблена для того, як IDE має справу з різними SDK, які можуть бути встановлені в системі. З 15,9.

Середовище VS2017 буде використовувати останню стабільну версію .NET Core SDK. Мотивація цього полягає в тому, щоб уникнути сценарію встановлення нового SDK, але не підтримуваного в VS2017.

Середовище Visual Studio 2017 RC розширює набір рефакторингів і виправлень, щоб допомогти підтримувати читабельну кодову базу та каталізувати робочі процеси розробки.

Висновки до третього розділу

В розділі були описано мову розробки C++. Вибір на її користь було зроблено через її швидкодію, багатофункціональність, читабельність та простоту підтримки а також акцент на вбудовуванні системи.

Описано бібліотеку Simple And Fast Multimedia Library, згадано інтегроване середовище розробки Visual Studio, що було обране через зручність кодування та встановлення бібліотек.

4. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ

В результаті роботи розроблено програмний модуль мовою C++, який може бути включений в більший проект. Для тестування та демонстрації роботи алгоритму було створено демонстраційний додаток з перевіркою коректності обробки реальних даних.

4.1 Демонстраційний додаток

Нижче на рисунках 4.1 та 4.2 представлені зразки інтерфейсу демонстраційного додатку.

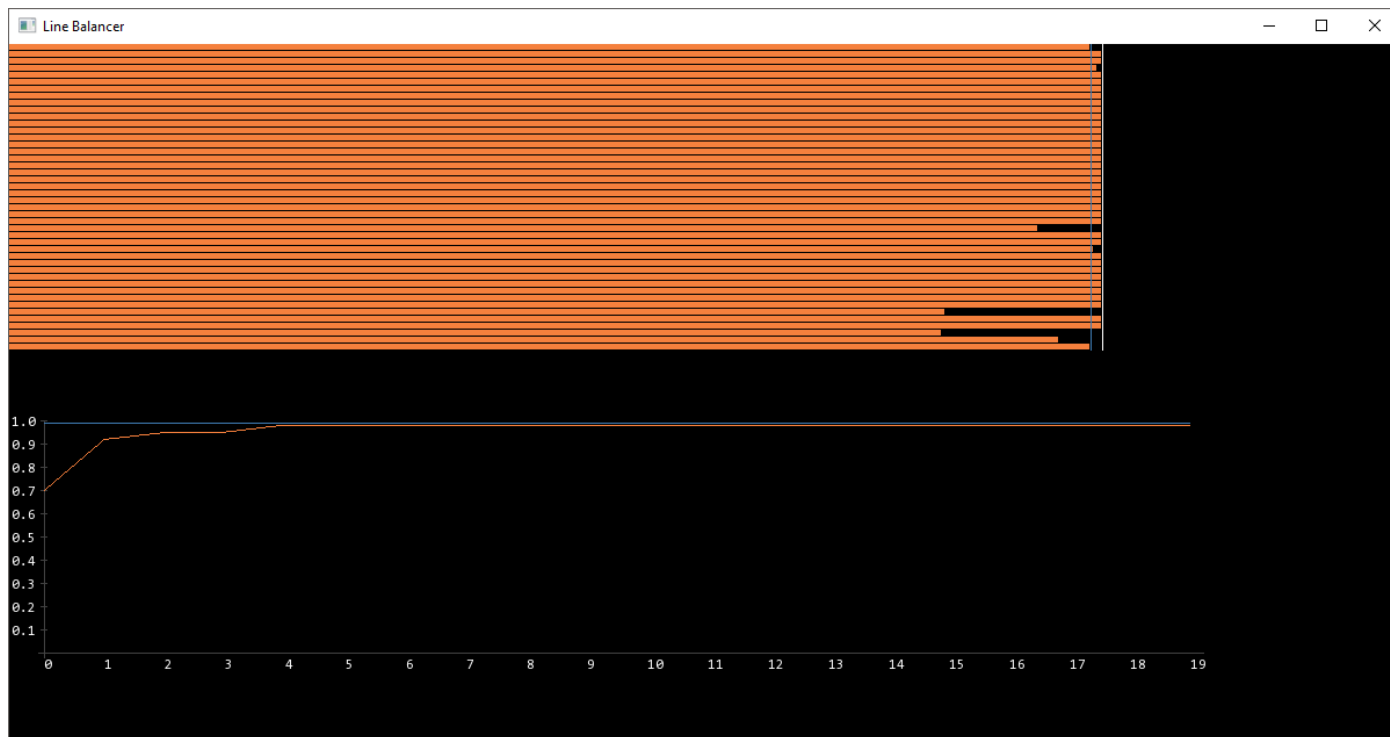


Рисунок 4.1 — Графічна репрезентація рішення


```

C:\Users\mickl\documents\visual studio 2017\Projects\LineBalancer\Debug\LineBalancer.exe
Bin capacity: 942
Leeway: 1
Capacity: 932
Bins Amount: 44
Building precedence matrix...done.
{3,1,8,11,65,444,2,106,121,29,6,652,11,38,15,17,3,2,2,8,271,256,750,40,48,1,7,158,49,147,238,441,4,68,52,303,326,69,26,2
2,895,809,5,61,195,2,353,16,11,1,172,643,818,529,251,3,6,30,1,8,1,166,19,102,3,1,202,85,4,7,15,815,3,7,169,284,54,188,1,
5,529,9,19,67,1,13,60,2,5,1,125,1,243,10,247,5,4,9,161,21,462,1,1,9,430,202,1,28,7,1,118,17,296,4,11,2,2,11,77,168,51,35
,350,198,519,57,1,65,78,121,726,344,148,89,2,215,38,2,444,30,534,1,23,712,3,23,386,1,31,229,258,557,315,108,21,199,1,37,
68,118,9,1,402,105,27,62,1,2,376,119,351,76,33,70,2,4,9,25,73,1,31,1,2,5,37,803,663,1,2,29,17,1,280,2,235,1,1,145,2,191,
5,903,137,54,38,2,453,27,133,1,16,198,3,153,842,1,26,80,2,233,1,270,10,358,3,3,1,27,1,111,19,2,1,531,105,129,249,44,8,15
,202,60,282,223,69,29,331,56,39,221,5,17,1,4,1,14,34,3,523,9,10,38,4,2,160,200,2,98,831,21,45,60,10,39,13,30,3,28,289,1,
89,290,1,10,285,1,142,1,179,106,5,12,165,27,4,52,10,686,9,236,14,549,442,39,13,1,312,3,359,359,265,130,218,25,6,2,1,184,
99,} [319 items]

workstation 0: 0 5 25 42 45 72 78 87 93 97 99 101 103 106 113 114 115 116 121 134 142 145 154 156 164 175 176 177 187 19
1 193 198 207 209 212 215 224 227 228 230 245 252 254 262 266 272 276 293 295 304
workstation 1: 1 146 251 255 258
workstation 2: 2 38 77 117 132 162 174 213 225
workstation 3: 3 26 39 49 56 66 68 81 98 126 196 202 203 210 231 239 257 282 285 289 291 292
workstation 4: 4 9 24 33 34 37 48 95 120 148 171 173 178 180 184 244 248 256 261 270
workstation 5: 6 28 53 205 308
workstation 6: 7 23 29 153 197 226 234 235 264
workstation 7: 8 59 65 206 223
workstation 8: 10 140 159 166 242
workstation 9: 11 278
workstation 10: 12 183 194 219 249 299
workstation 11: 13 46 181 301
workstation 12: 14 15 57 58 62 63 74 79 129 139 141 149 208 275 303
workstation 13: 16 61 135 151
workstation 14: 17 20 88 186
workstation 15: 18 36 69 124 125 144 147 179 182 260 274
workstation 16: 19 110 155 211 265 312
workstation 17: 21 32 51 136
workstation 18: 22 217 229
workstation 19: 27 80 84 165 199
workstation 20: 30 92 105 236 259
workstation 21: 31 167 247 302
workstation 22: 35 47 91 246 281
workstation 23: 40 73 89 108 200 220 222 232 250 298
workstation 24: 41 137 311
workstation 25: 43 60 107 127 157 158 172 189 204 216 241 267 271 273 280 313 317
workstation 26: 44 50 119 170
workstation 27: 52 169 253
workstation 28: 54 96 297
workstation 29: 55 100 102 133 195 218 263 277 283 286 287 288 296 305 307
workstation 30: 64 111 122 190 233 269 315
workstation 31: 67 128 131 168 237 300
workstation 32: 70 75 118 192 284
workstation 33: 71 90 109
workstation 34: 76 94 163 221 310
workstation 35: 82 86 160 161 188 214 238
workstation 36: 83 104 138
workstation 37: 85 130 240
workstation 38: 112 123 306
workstation 39: 143 243 314
workstation 40: 150 152 290 294 309
workstation 41: 185
workstation 42: 201 316
workstation 43: 268 279 318
Result: 0.979864
Best possible: 0.989384
Algorithm running time: 2290ms

```

Рисунок 4.2 — Зразок інтерфейсу демонстраційного додатку

На рисунку 4.2 зображена текстова репрезентація результатів роботи алгоритму. Зверху програмного вікна відображена частина вхідних даних: ємність робочих

станцій, процент незаповненості, тестова ємність, результуюча кількість станцій для перевірки результатів.

Нижче подано повідомлення про наявність графу технологічних операцій (занадто громіздкий для репрезентації), та безпосередньо інформація про процеси.

Після порожнього рядочка відображено результати розподілення завдань по робочим станціям (станція та ід процесів призначених на неї), фітнес отриманого результату та максимально можливий фітнес (враховуючи процент незаповненості, він буде менше 1.0). В самому низу представлено час роботи алгоритму в мілісекундах.

На рисунку 4.1 зображено графічне представлення цих результатів: часове навантаження складальних ліній, а також граф прогресу фітнес функції через покоління генетичного алгоритму.

Горизонтальна вісь — номер покоління, вертикальна — значення фітнес функції. Синя лінія — максимально можливе значення, помаранчева — реальне значення.

Демонстраційний додаток може завантажувати реальні тестові дані, або генерувати їх згідно до бажання користувача. Для переходу на новий тестовий випадок необхідно натиснути клавішу “R”.

4.2 Програмний модуль

Розроблений програмний модуль має головний клас GeneticBalancer з одним публічним методом

```
std::vector<std::vector<int>> balance(std::vector<int> items, int binCapacity, const PrecedenceGraph& pg);
```

Рисунок 4.3 — Оголошення методу balance

На вхід він приймає список інформації про процеси (визначені їх довжиною), час циклу робочої станції, а також граф першості, інтерфейс якому надано в цьому ж класі.

```

public:
class PrecedenceGraph
{
    std::vector<std::vector<int>> longestPath;
public:
    PrecedenceGraph() {};
    PrecedenceGraph(const std::vector<std::pair<int, int>>& edges);
    int getMaxDistance(int a, int b) { return longestPath[std::min(a, b)][std::max(a, b)]; };
};

```

Рисунок 4.4 — Інтерфейс класу PrecedenceGraph

Граф першості можна створити надавши список ребер у вигляді вектора пар ід першого і другого процесів. Використавши другий конструктор, можна передати створений об'єкт до методу `balance` і отримати результат.

Результат приходить у вигляді впорядкованого вектору результатів розподілення завдань по станціям. Кожен результат, в свою чергу, є впорядкованим вектором ід процесів.

Висновки до четвертого розділу

В розділі розглянуто інтерфейс створеного демонстраційного додатку, описана ключова інформація для розуміння і роботи з ним, описана робота з декількома тестовими випадками.

Описано публічний інтерфейс класів `GeneticBalancer` та `PrecedenceGraph`, та описані необхідні дії для інтеграції програмного модулю в систему.

5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

Стартап запускається окремими засновниками або підприємцями для пошуку повторюваної та масштабованої бізнес-моделі. Більш конкретно, стартап — це підприємство, що нещодавно з'явилося, метою якого є розробка життєздатної бізнес-моделі для задоволення потреб або проблем на ринку.

Засновники розробляють стартапи для ефективної розробки та перевірки масштабування бізнес-моделі.

Стартап капітал, або стартовий капітал — це гроші, необхідні для запуску нового бізнесу. Він може надходити з різних джерел і може використовуватися для будь-яких цілей, що допомагає перетворити початкову ідею до справжнього бізнесу.

Стартапи стикаються з високою невизначеністю і мають високі показники невдач. Проте меншість, які стають успішними компаніями, мають потенціал для того, щоб стати великими та впливовими.

5.1 Опис ідеї проекту

В межах даного підрозділу були проаналізовані і подані у вигляді таблиць наступні питання:

- зміст ідеї (що пропонується);
- напрямки застосування;
- основні вигоди, що може отримати користувач товару чи послуги;
- відмінності від існуючих товарів чи послуг.

Перші три питання розглянуті у вигляді таблиці (таблиця 5.1) і дають цілісне уявлення про зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 5.1. Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Надання механізмів для вирішення задачі балансування складальної лінії	1. Виробничі лінії	1. Швидкодія завдяки генетичному
	2. Виробництва з декількома складальними станціями	алгоритму групування 2. Можливість точного
	3. Виробництва, що проводять збірку та обробку матеріалів	моніторингу та динамічної модифікації навантаження

Аналіз потенційних техніко-економічних переваг ідеї порівняно із пропозиціями конкурентів передбачає:

- визначення переліку техніко-економічних властивостей та характеристик ідеї;
- визначення попереднього кола конкурентів (проектів-конкурентів) або товарів-замінників чи товарів-аналогів, що вже існують на ринку, та проводиться збір інформації щодо значень техніко-економічних показників для ідеї власного проекту та проектів-конкурентів відповідно до визначеного вище переліку;
- проводиться порівняльний аналіз показників: для власної ідеї визначаються показники, що мають а) гірші значення (слабкі сторони); б) аналогічні (нейтральні сторони) значення; в) кращі значення (сильні сторони) (таблиця 5.2).

Таблиця 5.2. Визначення сильних, слабких та нейтральних характеристик ідеї проекту

No		(потенційні) товари/концепції конкурентів		
		Мій проект	Система балансування 1	Система балансування 2
1	Слабка сторона	Не врахування різних типів станцій	Підтримка декількох типів станцій	Можливість задавати тип станцій
2	Сильна сторона	Швидкодія та можливість зміни навантаження	Неможливість динамічного навантаження	Низька швидкодія
3	Нейтральна сторона	Близька до ідеальної оптимізація	Результати в районі 90% від ідеалу	Майже ідеальні результати оптимізації

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

5.2 Технологічний аудит ідеї проекту

В межах даного підрозділу було проведено аудит технології, за допомогою якої можна реалізувати ідею проекту (технології створення товару). Визначення

технологічної здійсненності ідеї проекту передбачає аналіз таких складових (таблиця 5.3):

- технологія виготовлення товару згідно з ідеєю проекту;
- наявність потреби допрацювання технології;
- доступність технологій.

Таблиця 5.3. Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
1	Модуль балансування	Мова програмування C++	Наявна	Умовно безкоштовна
2	Інтерфейс користувача	Бібліотека SFML	Наявна	Умовно безкоштовна
3	Алгоритми оптимізації	Мова програмування C++	Наявна	Умовно безкоштовна
Висновок: проект можливо реалізувати за вибраними технологіями.				

За результатами аналізу таблиці зроблено висновок щодо можливості технологічної реалізації проекту. Технологічним шляхом реалізації проекту було обрано такі технології, як C++ та SFML через їх доступність та безкоштовність.

5.3 Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану

ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів.

Місткість ринку товарів, коло споживачів яких доволі широке, може бути визначена за допомогою статистичних методів, що враховують як тенденції минулих років у збуті товарів, так і перспективні (фактори науково-технічного прогресу, їх динаміку).

Спочатку було проведено аналіз попиту: наявність попиту, обсяг, динаміка розвитку ринку (таблиця 5.4).

Таблиця 5.4. Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	300 грн
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	50 %

Середню норму рентабельності в галузі було порівняно із банківським відсотком на вкладення. Останній є меншим, тому є сенс вкладати гроші саме у цей проект.

За результатами аналізу (таблиця 5.4) було зроблено висновок, що ринок є привабливим для входження.

Надалі були визначені потенційні групи клієнтів, їх характеристики та сформовано орієнтовний перелік вимог до товару для кожної групи (таблиця 5.5).

Потенційний клієнт — термін, що означає приватну особу або організацію, які не є в даний момент клієнтом компанії, але що входять в цільову ринкову групу цієї компанії. Теоретично, при якісній роботі, всі потенційні клієнти можуть стати реальними, якщо будуть виконані певні умови (більш інтенсивна реклама, підвищення якості, зниження ціни, вмiла робота менеджерів з продажу).

Таблиця 5.5. Характеристика потенційних клієнтів стартап-проекту

No	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Оптимізація розподілення навантаження робочих станцій	Виробництва зі складальними лініями	Компанії заключають довготривалі договори, а стартапери віддають перевагу пробному терміну	Швидкодія; стабільність роботи; невисока ціна; наявність документації; підтримка необхідних платформ;

Після визначення потенційних груп клієнтів було проведено аналіз ринкового середовища: складено таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають (таблиця 4.6, 4.7).

Таблиця 5.6. Фактори загроз

No	Фактор	Зміст загрози	Можлива реакція компанії
1	Не підходить для нових проєктів	Потребує зміни представлення даних	Корекція інтерфейсу модуля
2	Власний формат зберігання	При необхідності потрібно розробка сервісу преведення до визначеного формату	Додавання можливості автоматизованого експорту в різні типи сховищ, розробка додаткового ПЗ
3	Обмеженість функцій	Інструмент обмежений наявними функціями і не має деяких функцій, які мають конкуренти	Додавання нових функцій за потреби

Таблиця 5.7 — Фактори можливостей

No	Фактор	Зміст можливості	Можлива реакція компанії
1	Алгоритми групування	Швидкодія та підвищена точність	Вихід на ширший ринок виробництв

2	Недоліки в існуючих альтернативах	Існуючі альтернативи або працюють повільно, або не є орієнтованими на конкретну предметну область	Модифікація існуючих платформ
---	-----------------------------------	---	-------------------------------

Надалі було проведено аналіз пропозиції: визначили загальні риси конкуренції на ринку (таблиця 5.8).

Пропозиція — це обсяг товарів та послуг, який виробники хочуть і можуть поставити на ринок за різною ціною за певний проміжок часу. Сталий причинно-наслідковий зв'язок між ціною та обсягом товарів (послуг), який товаровиробник здатний поставити на ринок, виражається законом пропозиції. Сталий причинно-наслідковий зв'язок між ціною та обсягом товарів (послуг), який товаровиробник здатний поставити на ринок, виражається законом пропозиції.

Таблиця 5.8. Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції - монополія	На ринку присутні декілька компаній-конкурентів, але їх товар дещо відрізняється між собою.	Підтримка якості продукту та постійні нововведення, вдосконалення.
2. За рівнем	Компанії-конкуренти з	Створити основу ПП

конкурентної боротьби - міжнародний	інших країн	таким чином, щоб можна було легко переробити даний ПП для використання у галузях інших країн.
3. За галузевою ознакою - міжгалузева	Продукт може використовуватись для різних галузей	Постійне вдосконалення продукту, що не має прив'язки до сфери
4. Конкуренція за видами товарів: - товарно-видова	Конкуренція між видами ПП, їх особливостями.	Створити ПП, враховуючи недоліки конкурентів
5. За характером конкурентних переваг - нецінова	Вдосконалення технології створення ПП, щоб собівартість була нижчою	Удосконалення моделі. Використання більш дешевих технологій для розробки, ніж використовують конкуренти, але тільки якщо ці технології відповідають необхідним вимогам якості.
6. За інтенсивністю - не марочна	Бренд присутній, але його роль незначна	Реклама, участь у конференціях, семінарах.

Було проведено аналіз конкуренції у галузі за моделлю М. Портера (таблиця 4.9).

Таблиця 5.9. Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Висновки:	Визначити інтенсивність конкурентної боротьби з боку прямих конкурентів	Є можливості виходу на ринок, оскільки існуючі рішення не надають потрібних переваг	Постачальники підлаштовуються під ринок	Клієнти диктують вимоги згідно з умовами експлуатації	Обмеження для роботи на ринку через товари-замінники

За результатами аналізу таблиця 5.9 було зроблено висновок про можливість роботи на ринку з огляду на конкурентну ситуацію.

Цей висновок був врахований при формулюванні переліку факторів конкурентоспроможності у наступному пункті. На основі аналізу конкуренції, проведеного в таблиці 5.9, а також із урахуванням характеристик ідеї проекту (таблиця 5.2), вимог споживачів до товару (таблиця 5.5) та факторів маркетингового середовища (таблиці 5.6, 5.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлено у таблиці 5.10. Конкурентоспроможність — здатність підприємства створювати, виробляти і продавати товари та послуги, цінові й нецінові якості яких привабливіші, ніж в аналогічній продукції конкурентів.

Таблиця 5.10. Обґрунтування факторів конкурентоспроможності

No	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Орієнтація на швидкий перехід до середньо- та дрібносерійного виробництва	Існуючі конкуренти або не враховують особливості формування сценаріїв, або виконують процес побудови не оптимально

За визначеними факторами конкурентоспроможності (таблиця 5.10) проведено аналіз сильних та слабких сторін стартап-проекту (таблиця 5.11).

Таблиця 5.11. Порівняльний аналіз сильних та слабких сторін проекту

No п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з даним продуктом						
			-3	-2	-1	0	1	2	3
1	Орієнтація на швидкий перехід до середньо- та дрібносерійного виробництва	20	+						

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (таблиця 5.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін.

Перелік ринкових загроз та ринкових можливостей було складено на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу

факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення.

Наприклад: зниження доходів потенційних споживачів — фактор загрози, на основі якого можна зробити прогноз щодо посилення значущості цінового фактору при виборі товару та відповідно, — цінової конкуренції (а це вже — ринкова загроза). SWOT-аналіз широко застосовується у процесі стратегічного планування, що полягає в розділенні чинників і явищ на чотири категорії.

Таблиця 5.12. SWOT-аналіз стартап-проекту

<p>Сильні сторони:</p> <p>Швидкодія балансуючого алгоритму</p> <p>Висока наближеність до ідеального оптимуму</p> <p>Актуальність користування системою, яка викликана переходом до середньо- та дрібносерійного виробництва</p> <p>Невелика ціна користування за місяць</p>	<p>Слабкі сторони:</p> <p>Потребує масштабної рекламної компанії</p> <p>Орієнтація на виробництва, які можуть не мати коштів на покращення</p>
<p>Можливості:</p> <p>Можливе продовження розробки проекту за кордоном, тому що проблема переходу виробництва актуальна не лише в Україні</p>	<p>Загрози:</p> <p>Відсутність користувачів через погану рекламну компанію</p>

На основі SWOT-аналізу було розроблено альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний

час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок.

Визначені альтернативи були проаналізовані з точки зору строків та ймовірності отримання ресурсів (таблиця 5.13).

Таблиця 5.13. Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Орієнтація поточної моделі на ринок стартаперів	25 %	8 год
2	Орієнтація поточної моделі на ринок державних установ	20 %	72 год
3	Орієнтація поточної моделі на ринок ентерпрайз	35 %	168 год
4	Переорієнтація на розробку серверної частини	75 %	120 год
5	Переорієнтація на веб-розробку	45 %	96 год

Після аналізу було обрано альтернативу №2.

5.4 Аналіз ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: було проведено опис цільових груп потенційних споживачів (таблиця 5.14).

Таблиця 5.14. Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Стартапери	Готові	Високий	Висока	Просто
2	Державні установи	Потребують недовгих переговорів	Середній	Середня	Складно
3	Ентерпрайз	Потребують довгих переговорів	Низький	Низька	Дуже складно
Які цільові групи обрано: стартапери					

За результатами аналізу потенційних груп споживачів було обрано цільові групи, для яких буде запропоновано даний товар, та визначено стратегію охоплення ринку — стратегію диференційованого маркетингу (компанія працює з декількома сегментами).

Для роботи в обраних сегментах ринку сформовано базову стратегію розвитку (таблиця 5.15).

Таблиця 5.15. Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Орієнтація поточної моделі на ринок стартаперів	Стратегія концентрованого маркетингу	Стартапери потребують швидкості розробки, яку надає підтримка декількох платформ даним продуктом	Стратегія спеціалізації (спирається на диференціацію)

Наступним кроком обрано стратегію конкурентної поведінки (таблиця 5.16).

Таблиця 5.16. Визначення базової стратегії конкурентної поведінки

Чи є проект “першопрохідцем” на ринку?	Чи буде компанія шукати нових споживачів	Чи буде компанія копіювати основні характеристики конкурента	Стратегія конкурентної поведінки
Ні	Шукати нових споживачів, забирати існуючих у конкурентів	Ні	Стратегія заняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап-компанії) та до продукту, а також в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки розроблено стратегію позиціонування (таблиця 4.17), що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Позиціонування — це маркетингове забезпечення товарів бажаного місця на ринку і у свідомості потенційних покупців (образ). Позиція компанії чи продукту показує чим він унікальний УТП (унікальну торговельну пропозицію), чим відрізняється від конкурентів (відстройка від конкурентів), чим корисний споживачу.

Таблиця 5.17. Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувавши комплексну позицію власного проекту (три ключових)
1	Легкість розуміння, зручний інтерфейс, надійний, швидкий, точний та достовірний ПП для побудови моделей і прогнозів.	Стратегія диференціації	Позиція на основі порівняння фірми з товарами конкурентів; Відмінні особливості споживача	Економія часу; Зручність застосування; Практичність та точність результату

Результатом виконання підрозділу стала узгоджена система рішень щодо ринкової поведінки компанії, яка визначає напрями роботи стартап-компанії на ринку.

5.5 Розроблення маркетингової програми стартап-проекту

Сформовано маркетингову концепцію товару, який отримає споживач. Для цього у таблиці 5.18 підсумовано результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18. Визначення ключових переваг концепції потенційного товару

Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
Пришвидшення оптимальності роботи алгоритму	Алгоритм групування, що надає високу точність та швидкодію	Конкуренти формують сценарії не оптимальним шляхом

Розроблено трирівневу маркетингову модель товару: уточнюється ідея продукту та/або послуги, його фізичні складові, особливості процесу його надання (таблиця 5.18).

1-й рівень При формуванні задуму товару вирішується питання щодо того, засобом вирішення якої потреби і / або проблеми буде даний товар, яка його основна вигода. Дане питання безпосередньо пов'язаний з формуванням технічного завдання в процесі розробки конструкторської документації на виріб.

2-й рівень Цей рівень являє рішення того, як буде реалізований товар в реальному/ включає в себе якість, властивості, дизайн, упаковку, ціну.

3-й рівень Товар з підкріпленням (супроводом) — додаткові послуги та переваги для споживача, що створюються на основі товару за задумом і товару в реальному виконанні (гарантії якості, доставка, умови оплати та ін).

Таблиця 5.19. Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові		
I. Товар за задумом	Зручність та швидкість отримання практичного результату щодо побудови моделі та прогнозування процесів.		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. функція побудови моделі процесу		
	2. функція побудови прогнозу		
	Якість: достовірність побудови математичної моделі, достовірність побудови прогнозу		
	Пакування: відсутнє		
	Марка: OptiLine		
III. Товар із підкріпленням	До продажу: відсутнє		
	Після продажу: персональна підтримка в обслуговуванні за додаткову платню.		
Вихідний код та математична модель будуть закриті. На ідею зареєстровано патент.			

Після формування маркетингової моделі товару слід відмітити, що проект буде захищено від копіювання за допомогою ноу-хау. Наступним кроком є визначення цінних меж, якими необхідно керуватись при встановленні ціни на потенційний товар

(остаточне визначення ціни відбувається під час фінансово-економічного аналізу проекту), яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (таблиця 5.20). Аналіз проведено експертним методом.

Таблиця 5.20 — Визначення меж встановлення ціни

N	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	27...250 грн	105...300 грн	25000...50000 грн	27...105 грн

Наступним кроком є визначення оптимальної системи збуту, в межах якого було прийняте рішення (таблиця 5.21).

Таблиця 5.21. Формування системи збуту

No	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Клієнт повинен надаватися в режимах “тріал” та “повний” сплатити після закінчення випробувального строку	Легість в встановленні, легкість в сплаті послуг	Веб-сайт	Проводити збут силами посередника формування сценаріїв

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (таблиця 5.22).

Таблиця 5.22. Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
Купляють програми через авторизовану мережу	Веб-сайти	Формування сценарію розвитку	Довести, що програмний продукт оптимально формує сценарій	Формування сценарію розвитку

Висновки до п'ятого розділу

В розділі проведено аналіз програмного продукту у якості стартап-проекту. Визначено потенціал проекту до комерціалізації. Встановлена наявність конкурентів на ринку, розроблені стратегії для подолання конкуренції.

Подальший розвиток проекту признано доцільним, оскільки він знайде свою цільову аудиторію.

ВИСНОВКИ

В ході магістерської дисертації було досліджено існуючі системи оптимізації виробництва, що спеціалізуються на балансуванні складальних ліній, виявлено їхні недоліки. Серед основних можна виділити:

- відсутність формальної аналітичної залежності при визначенні часу роботи лінії, що робить неможливим пошук оптимуму за використовуваними критеріями ефективності класичними методами оптимізації;

- нечутливість до контексту;

- низька швидкодія.

У результаті дослідження було виявлено, що існуючі програмні системи спеціалізуються не на оптимізації, а на роботі з даними, тому необхідно розробляти і застосовувати генетичні алгоритми для підвищення швидкодії.

На основі аналізу предметної області і визначення мети роботи як використання генетичних алгоритмів для покращення оптимізації планування виробництва, сформульовані такі завдання досліджень:

- проаналізувати генетичні оператори та модифікувати їх до задач групування;

- розробити алгоритм для розв'язання задачі балансування складальної лінії, використовуючи модифіковані оператори;

- розробити демонстраційний додаток для тестування й відображення результатів роботи алгоритму.

Для усунення існуючих недоліків і побудови системи, що дає можливість підвищити швидкодію, оптимальність результату, та чутливість до контексту алгоритму, було розроблено нові підходи, які становлять наукову новизну:

- змінено кодування хромосоми в генетичних операторах;

- змінено підрахунок фітнес-функції;
- модифіковано генетичні оператори: схрещування, мутації, інверсії;
- змінено стандартні евристичні алгоритми для врахування порядку виконання операцій.

На основі запропонованих алгоритмів розв'язання задачі балансування складальної лінії, був розроблений програмний модуль, який за рахунок нового підходу до кодування і використання генетичних операторів по швидкодії та оптимальності результатів перевищує існуючі аналоги. Для тестування й відображення результатів роботи було створено демонстраційний додаток. Програмний модуль написано мовою C++.

Створена комп'ютерно-інформаційна система спрощує процес оптимізації виробництва шляхом ефективного розподілення навантаження робочих станцій. Розроблений програмний продукт може бути використаний на виробництвах, які містять складальні лінії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Беляева О.П. Организационные методы повышения гибкости производственных систем: дисс. канд. техн. наук. — Кострома: КГУ им. Н.А. Некрасова, 2006. — 126 с.
2. Becker C., Scholl A. A survey on problems and methods in generalized assembly line balancing. — № 21, 2003.
3. Рейнгольд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы. Теория и практика. — М.: Мир, 1980. — 476 с. — С. 443-445
4. Баранов А.А., Денисов А.Р., Левин М.Г. Подсистема имитационного моделирования работы производственных линий. // Управление большими системами. Вып. 21. — М.: ИПУ РАН, 2008 . — С. 173-185.
5. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы. — 2-е изд., испр. и доп. — М.: Физматлит, 2006. — 320 с.
6. Емельянов В. В., Курейчик В. В., Курейчик В. М. Теория и практика эволюционного моделирования. — М: Физматлит, 2003. — 432 с.
7. Sacerdoti Earl D. A structure for plans and behavior, Stanford Research Institute, Elsevier NorthHolland Inc.
8. Davis Lawrence (Ed) Genetic Algorithms and Simulated Annealing, Pitman Publishing, London.
9. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы: Учебное пособие. — 2-е изд. — М: Физматлит, 2006. — 320 с.
10. Генетичний алгоритм [Електронний ресурс]. — Режим доступу https://howlingpixel.com/i-uk/Генетичний_алгоритм
11. Гладков Л. А., Курейчик В. В, Курейчик В. М. и др. Биоинспирированные методы в оптимизации: монография. — М: Физматлит, 2009. — 384 с.

12. Davis Lawrence (Ed) Genetic Algorithms and Simulated Annealing, Pitman Publishing, London.
13. Peng Yanming The Algorithms for the Assembly Line Balancing Problem, CRIF Internal Report.
14. Ting, Chuan-Kang (2005). “On the Mean Convergence Time of Multi-parent Genetic Algorithms Without Selection”. *Advances in Artificial Life*: 403-412.
15. Akbari, Ziarati (2010). “A multilevel evolutionary algorithm for optimizing numerical functions” *IJIEC* 2 (2011). — P. 419-430.
16. Bäck, Thomas; Kok, Joost N. *Handbook of Natural Computing*. Springer Berlin Heidelberg. — P. 1035-1069.
17. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы = Sieci neuronowe, algorytmy genetyczne i systemy rozmyte. — 2-е изд. — М: Горячая линия-Телеком, 2008. — 452 с.
18. Скобцов Ю. А. Основы эволюционных вычислений. — Донецк: ДонНТУ, 2008. — 326 с.
19. Holland J.H. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.
20. Schwefel, Hans-Paul (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie : mit einer vergleichenden Einführung in die Hill-Climbing- und Zufallsstrategie*. Basel; Stuttgart: Birkhäuser.
21. Слюсар В.И. Синтез антенн на основе генетических алгоритмов. //Первая миля. Last mile (Приложение к журналу “Электроника: наука, технология, бизнес”). — 2008. — № 6. — С. 16-23.
22. Слюсар В.И. Синтез антенн на основе генетических алгоритмов. Часть 2. //Первая миля. Last mile (Приложение к журналу “Электроника: наука, технология, бизнес”). — 2009. — № 1. -. — С. 22-25.

23. Barricelli, Nils Aall (1957). Symbiogenetic evolution processes realized by artificial methods. *Methodos*: 143-182.
24. Pelikan, Martin; Goldberg, David E.; Cantú-Paz, Erick (1 January 1999). “BOA: The Bayesian Optimization Algorithm”. *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation — Volume 1*. Morgan Kaufmann Publishers Inc. — P. 525-532.
25. Sadowski Krzysztof L., Bosman Peter A.N., Thierens Dirk (1 January 2013). “On the Usefulness of Linkage Processing for Solving MAX-SAT”. *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. ACM. — P. 853-860.
26. Barricelli, Nils Aall (1954). “Esempi numerici di processi di evoluzione”. *Methodos*. — P. 45-68.
27. Barricelli, Nils Aall (1957). “Symbiogenetic evolution processes realized by artificial methods”. *Methodos*. — P. 143-182.
28. Fraser, Alex; Burnell, Donald (1970). *Computer Models in Genetics*. New York: McGraw-Hill.
29. Crosby, Jack L. (1973). *Computer Simulation in Genetics*. London: John Wiley & Sons.
30. Pelikan, Martin; Goldberg, David E.; Cantú-Paz, Erick (1 January 1999). “BOA: The Bayesian Optimization Algorithm”. *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation — Volume 1*. Morgan Kaufmann Publishers Inc. — P. 525–532.
31. Pelikan, Martin (2005). *Hierarchical Bayesian optimization algorithm : toward a new generation of evolutionary algorithms (1st ed.)*. Berlin [u.a.]: Springer.
32. Ferreira, C. “Gene Expression Programming: A New Adaptive Algorithm for Solving Problems” (PDF). *Complex Systems*, Vol. 13, issue 2. — P. 87-129.

33. Zlochin, Mark; Birattari, Mauro; Meuleau, Nicolas; Dorigo, Marco (1 October 2004). "Model-Based Search for Combinatorial Optimization: A Critical Survey". *Annals of Operations Research*. 131 (1–4). — P. 373-395.
34. "Optimizing a hybrid vendor-managed inventory and transportation problem with fuzzy demand: An improved particle swarm optimization algorithm". *Information Sciences*. 272. — P. 126-144.
35. Barricelli, Nils Aall (1957). Symbiogenetic evolution processes realized by artificial methods. *Methodos*. — P. 143-182.
36. Sacerdoti Earl D. A structure for plans and behavior, Stanford Research Institute, Elsevier NorthHolland Inc.
37. Davis Lawrence (Ed) *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, London.
38. Bäck, Thomas; Kok, Joost N. *Handbook of Natural Computing*. Springer Berlin Heidelberg. — P. 1035-1069.
39. Гладков Л. А., Курейчик В. В, Курейчик В. М. и др. Биоинспирированные методы в оптимизации: монография. — М: Физматлит, 2009. — 384 с.
40. Davis Lawrence (Ed) *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, London.
41. Coffin, David; Smith, Robert E. (1 January 2008). "Linkage Learning in Estimation of Distribution Algorithms". *Linkage in Evolutionary Computation*. Springer Berlin Heidelberg. — P. 141-156.
42. Zhang, J.; Chung, H.; Lo, W. L. (2007). "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms". *IEEE Transactions on Evolutionary Computation*. 11 (3): 326–335.
43. Patrascu, M.; Stancu, A.F.; Pop, F. (2014). "HELGA: a heterogeneous encoding lifelike genetic algorithm for population evolution modeling and simulation". *Soft Computing*. 18. — P. 2565-2576.

44. Hornby, G. S.; Linden, D. S.; Lohn, J. D., Automated Antenna Design with Evolutionary Algorithms.
45. Rania Hassan, Babak Cohanim, Olivier de Weck, Gerhard Venter (2005) A comparison of particle swarm optimization and the genetic algorithm.
46. Fogel, David B. (editor) (1998). *Evolutionary Computation: The Fossil Record*. New York: IEEE Press.
47. Janikow, C. Z.; Michalewicz, Z. (1991). "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms" (PDF). *Proceedings of the Fourth International Conference on Genetic Algorithms*. — P. 31-36.
48. Eiben, A. E. et al (1994). "Genetic algorithms with multi-parent recombination". *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*. — P. 78-87.
49. Zhang, J.; Chung, H.; Lo, W. L. (2007). "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms". *IEEE Transactions on Evolutionary Computation*. 11 (3). — P. 326-335.
50. Echegoyen, Carlos; Mendiburu, Alexander; Santana, Roberto; Lozano, Jose A. (8 November 2012). "On the Taxonomy of Optimization Problems Under Estimation of Distribution Algorithms". *Evolutionary Computation*. 21 (3). — P. 471-495.
51. Stroustrup, Bjarne (2000). *The C++ Programming Language (Special ed.)*. Addison-Wesley. — P. 46.
52. Stroustrup, Bjarne (2013). *The C++ Programming Language*. Addison Wesley. — P. 349.