КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра системного аналізу та теорії прийняття рішень

Кваліфікаційна робота
на здобуття ступеня магістра

за спеціальністю 124 Системний аналіз

на тему:

ЗАСТОСУВАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ДЛЯ ВИРІШЕННЯ ЗАДАЧІ КОМІВОЯЖЕРА

Виконав студент 2-го курсу магістратури
Парадюк Микола Віталійович

Науковий керівник:
професор, доктор фіз.-мат. наук
Івохін Євген Вікторович

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Роботу розглянуто й допущено до захисту на засіданні кафедри системного аналізу та теорії прийняття рішень
«04» травня 2023 р.,
протокол № 11
Завідувач кафедри
О. Г. Наконечний

Київ – 2023

РЕФЕРАТ

Обсяг роботи 77 сторінок, 15 ілюстрацій, 2 таблиці, 9 джерел посилань.

ЗАДАЧА КОМІВОЯЖЕРА, ГЕНЕТИЧНИЙ АЛГОРИТМ, UML-ДІАГРАМА, ЗАДАЧА ОПТИМІЗАЦІЇ, ПОПУЛЯЦІЯ, КРОСИНГОВЕР, МУТАЦІЯ, ПОКОЛІННЯ, ХРОМОСОМА.

Метою даної магістерської роботи є дослідження та застосування генетичних алгоритмів для вирішення задачі комівояжера. Основним завданням дослідження є розробка системи, яка зможе автоматично знаходити оптимальний маршрут для заданої множини міст, який пройде через кожне місто лише один раз і повернеться до початкового міста.

Об’єкт дослідження – процес застосування генетичних алгоритмів для вирішення задачі комівояжера.

Предмет дослідження програмні засоби вивчення та розробки алгоритмів, які б дозволяли знайти оптимальний маршрут між заданими точками, який проходить через кожну точку лише один раз.

Методи дослідження. Методи дослідження, які були використані в цій роботі, включають аналіз предметної області задачі комівояжера, проектування системи з використанням генетичних алгоритмів, програмування та тестування розробленого додатку.

Практичне значення одержаних результатів полягає у можливості застосування розробленого додатку для знаходження оптимального маршруту між заданими точками, що має значення у таких сферах, як транспорт, логістика, туризм, наука та технології. Розроблення ефективних алгоритмів для розв'язання задач комівояжера є актуальним завданням в різних галузях, тому результати дослідження можуть бути використані для подальшого дослідження та розвитку в цій області.

ЗМІСТ

ВСТУП

Актуальність теми. Задача комівояжера є однією з найважливіших та найпоширеніших задач комбінаторної оптимізації. Вона має велике значення для практичних застосувань, таких як логістика, транспорт, електроніка, а також в інших галузях, де потрібно знайти найкоротший шлях, який проходить через кілька точок.

Проте, розв'язок задачі комівояжера за допомогою точних методів стає неможливим при збільшенні кількості міст, що досить часто виникає в реальних задачах. Тому дослідження методів оптимізації для задачі комівояжера є актуальним напрямком досліджень в сучасній науці.

Один з потенційних методів для розв'язання задачі комівояжера - це генетичні алгоритми. Генетичні алгоритми є еволюційним методом оптимізації, який заснований на принципах природного добору та генетики. Вони можуть бути застосовані для розв'язання задач оптимізації в різних сферах, включаючи задачу комівояжера. Однак, до цього часу було проведено недостатньо досліджень з використанням генетичних алгоритмів для розв'язання задачі комівояжера.

Тому, дана магістерська робота має значний науковий і практичний інтерес. Результати дослідження можуть стати основою для подальшого розвитку методів оптимізації, що використовують генетичні алгоритми, і бути використані для вирішення реальних проблем, пов'язаних з оптимізацією маршрутів.

Мета та завдання дослідження. Метою даної магістерської роботи є дослідження та застосування генетичних алгоритмів для вирішення задачі комівояжера. Основним завданням дослідження є розробка системи, яка зможе автоматично знаходити оптимальний маршрут для заданої множини міст, який пройде через кожне місто лише один раз і повернеться до початкового міста.

Для досягнення мети дослідження необхідно вирішити такі завдання: провести аналіз предметної області задачі комівояжера, визначити сферу

застосування даної задачі, проаналізувати програмні аналоги та обрати оптимальний алгоритм для реалізації системи. Далі потрібно розробити систему на базі генетичного алгоритму, яка зможе автоматично знаходити оптимальний маршрут для заданої множини міст, виконати тестування розробленої системи та провести аналіз результатів.

Об’єкт дослідження – процес застосування генетичних алгоритмів для вирішення задачі комівояжера.

Предмет дослідження програмні засоби вивчення та розробки алгоритмів, які б дозволяли знайти оптимальний маршрут між заданими точками, який проходить через кожну точку лише один раз.

Методи дослідження. Методи дослідження, які були використані в цій роботі, включають аналіз предметної області задачі комівояжера, проектування системи з використанням генетичних алгоритмів, програмування та тестування розробленого додатку.

Наукова новизна. Науковою новизною цієї роботи є те, що вона розглядає застосування генетичних алгоритмів для вирішення задачі комівояжера та досліджує ефективність цього підходу. Результати цієї роботи можуть бути корисними для подальшого розвитку алгоритмів розв'язання задач комівояжера, а також для практичного використання в різних галузях, які мають справу з розкладами маршрутів.

Практичне значення одержаних результатів полягає у можливості застосування розробленого додатку для знаходження оптимального маршруту між заданими точками, що має значення у таких сферах, як транспорт, логістика, туризм, наука та технології. Розроблення ефективних алгоритмів для розв'язання задач комівояжера є актуальним завданням в різних галузях, тому результати дослідження можуть бути використані для подальшого дослідження та розвитку в цій області.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ ЗАДАЧІ КОМІВОЯЖЕРА

1.1 Обгрунтування сфери застосування задачі комівояжера

Задача комівояжера (TSP) є однією з найвідоміших NP-складних задач, що означає, що немає гарантії отримання оптимального маршруту та немає точного алгоритму для її вирішення за поліноміальний час [1]. Методом, який однозначно дозволить отримати оптимальний розв'язок ТСП, є метод вичерпного перерахування та оцінки. Ця процедура починається з генерації можливості всіх турів і оцінки відповідно до тривалості туру. Тур з найменшою довжиною вибирається як найкращий і гарантовано оптимальний [2]. У цей час багато компаній і установ стикаються і мають вирішити випадки TSP. Наприклад, компанії служби доставки, які щодня стикаються з проблемами, це відбувається через те, що маршрут, за яким слід дістатися до місця призначення, не є оптимальним. Приклади випадків у службі доставки: кур’єру потрібно доставити товари деяким клієнтам з різними пунктами призначення. Тому час має велике значення при доставці товарів, він пов’язаний з репутацією компанії. Для досягнення цих цілей потрібна система, здатна забезпечити оптимальний маршрут подорожі, щоб час подорожі був мінімальним. Тому йому потрібна програма, яка може оптимізувати рішення TSP, вона використовує генетичний алгоритм для отримання оптимального рішення за короткий час.

Генетичний алгоритм (GA) — це техніка пошуку в інформатиці для пошуку наближених розв’язків задач оптимізації та пошуку, таких як TSP. GA може бути альтернативою для вирішення TSP в останні роки, оскільки було доведено, що він є більш ефективним і привабливим у пошуку оптимальних або майже оптимальних рішень [5] [6]. GA — це алгоритм на основі популяції, цей метод — сукупність рішень, що називається популяцією, і найкращі популяції підтримуватимуться для отримання оптимального рішення за допомогою генетичних операторів [3]. У застосуванні GA для розв’язання TSP є слабка сторона, тобто GA отримає оптимальні результати, коли зійдеться з глобальним

оптимумом, але коли GA зійде до локального оптимуму, він отримає рішення, яке не є оптимальним. Завдяки GA не може здійснювати пошук по всьому простору[4]. Одним із методів подолання цих недоліків є гібридизація техніки локального пошуку. Він може покращити продуктивність GA, вставивши найкращу нову особину в цикл GA. HGA виявився більш ефективним, ніж GA, який використовується в задачах оптимізації, таких як TSP [7] [9].

Наведемо декілька прикладів застосування задачі комівояжера на практиці.

Пряме застосування TSP знаходиться в задачі свердління друкованих плат (PCB) (Grötschel та ін., 1991). Щоб з'єднати провідник на одному шарі з провідником на іншому шарі або розташувати контакти інтегральних схем, необхідно просвердлити отвори в платі. Отвори можуть бути різного розміру. Щоб просвердлити два отвори різного діаметру поспіль, головку верстата необхідно перемістити в інструментальний ящик і змінити свердлильне обладнання. Це займає досить багато часу. Таким чином, зрозуміло, що потрібно вибрати певний діаметр, просвердлити всі отвори однакового діаметру, змінити свердло, просвердлити отвори наступного діаметру тощо. Таким чином, цю задачу свердління можна розглядати як серію TSP, один для діаметр кожного отвору, де «міста» є початковою позицією та набором усіх отворів, які можна просвердлити одним і тим же свердлом. «Відстань» між двома містами визначається часом, необхідним для переміщення бурової головки з одного положення в інше. Мета полягає в тому, щоб мінімізувати час руху головки машини.

(Plante та ін., 1987) повідомили про це застосування, і це відбувається, коли газотурбінні двигуни літаків повинні бути капітально відремонтовані. Щоб гарантувати рівномірний потік газу через турбіни, на кожному ступені турбіни розташовані вузли сопло-направляюча лопатка. Такий вузол в основному складається з ряду направляючих лопаток сопла, прикріплених по його колу. Всі ці лопатки мають індивідуальні характеристики, і правильне розташування лопаток може призвести до істотних переваг (зменшення вібрації, підвищення рівномірності потоку, зниження витрати палива). Проблема розміщення

лопаток найкращим чином може бути змодельована як TSP зі спеціальною цільовою функцією.

Аналіз структури кристалів (Bland & Shallcross, 1989; Dreissig & Uebach, 1990) є важливим застосуванням TSP. Тут рентгенівський дифрактометр використовується для отримання інформації про структуру кристалічного матеріалу. Для цього детектор вимірює інтенсивність відбиття рентгенівського випромінювання кристала з різних позицій. У той час як саме вимірювання може бути виконано досить швидко, є значні накладні витрати на час позиціонування, оскільки для деяких експериментів потрібно реалізувати до сотень тисяч позицій. У двох прикладах, які ми згадуємо, позиціонування передбачає переміщення чотирьох двигунів. Час, необхідний для переходу з однієї позиції в іншу, можна розрахувати дуже точно. Результат експерименту не залежить від послідовності, в якій проводяться вимірювання в різних положеннях. Однак загальний час, необхідний для експерименту, залежить від послідовності. Тому проблема полягає в пошуку послідовності, яка мінімізує загальний час позиціонування. Це призводить до проблеми комівояжера.

(Lenstra & Rinnooy Kan, 1974) повідомили про окремий випадок з’єднання компонентів на платі комп’ютера. Модулі розташовані на платі комп'ютера, і необхідно підключити певний набір контактів. На відміну від звичайного випадку, коли потрібне з’єднання дерева Штейнера, тут вимога полягає в тому, щоб до кожного контакту було приєднано не більше двох проводів. Отже, ми маємо проблему пошуку найкоротшого гамільтонового шляху з невизначеними початковою та кінцевою точками. Подібна ситуація має місце для так званої проводки тестової шини. Щоб протестувати виготовлену плату, потрібно реалізувати з’єднання, яке входить до плати в певній точці, проходить через усі модулі та завершується в певній точці. Для кожного модуля ми також маємо визначену точку входу та виходу для цієї тестової проводки. Ця проблема також зводиться до розв’язання проблеми Гамільтонова шляху з тією різницею, що відстані не є симетричними, а початкова та кінцева точки вказані.

Ця проблема пов'язана з обробкою матеріалів на складі (Ratliff & Rosenthal, 1983). Припустимо, що на склад надходить замовлення на певну підмножину товарів, що зберігаються на складі. Деякі транспортні засоби мають зібрати всі предмети цього замовлення, щоб відправити їх клієнту. Відношення до TSP видно відразу. Місця зберігання елементів відповідають вузлам графа. Відстань між двома вузлами визначається часом, необхідним для переміщення транспортного засобу з одного місця в інше. Проблема пошуку найкоротшого маршруту для транспортного засобу з мінімальним часом посадки тепер може бути вирішена як TSP. В особливих випадках цю проблему можна легко вирішити, див. (van Dal, 1992) для широкого обговорення та для посилань.

Припустимо, що в місті n поштових скриньок потрібно спорожняти щодня протягом певного періоду часу, скажімо, 1 години. Проблема полягає в тому, щоб знайти мінімальну кількість вантажівок, щоб зробити це, і найкоротший час для збирання, використовуючи цю кількість вантажівок. Як інший приклад, припустимо, що n клієнтів потребують певної кількості деяких товарів, і постачальник повинен задовольнити всі вимоги за допомогою парку вантажівок. Проблема полягає в тому, щоб знайти розподіл клієнтів по вантажівкам і графік доставки для кожної вантажівки, щоб не перевищувати пропускну здатність кожної вантажівки і мінімізувати загальну відстань подорожі. Кілька варіантів цих двох проблем, де поєднуються обмеження часу та потужності, поширені в багатьох прикладних програмах реального світу. Цю задачу можна розв’язати як TSP, якщо немає обмежень щодо часу та потужності та якщо кількість вантажівок фіксована (скажімо, m). У цьому випадку ми отримуємо задачу m - продавців. Тим не менш, можна застосувати методи для TSP, щоб знайти хороші можливі рішення цієї проблеми (див. Lenstra & Rinnooy Kan, 1974).

Для виготовлення кожного шару друкованої плати, а також для шарів інтегральних напівпровідникових приладів необхідно виготовити фотомаску. У нашому випадку для друкованих плат це робиться механічним графічним пристроєм. Плоттер переміщує лінзу над скляною пластиною зі

світлочутливим покриттям. Затвор можна відкрити або закрити, щоб відкрити певні частини пластини. Існують різні отвори, щоб мати можливість генерувати різні структури на дошці. Необхідно розглянути два типи структур. Лінія експонується на пластині шляхом переміщення закритого затвора до однієї кінцевої точки лінії, потім відкриття затвора та переміщення його до іншого кінця лінії. Потім затвор закривається. Структура точкового типу створюється шляхом переміщення (з відповідною діафрагмою) до положення цієї точки, потім відкриття затвора, щоб зробити короткий спалах, а потім його знову закриття. Точне моделювання проблеми керування плоттером призводить до проблеми, складнішої, ніж TSP, а також складнішої, ніж задача сільського листоноші. Реальне застосування у реальному виробничому середовищі описано в (Grötschel та ін., 1991).

Основне застосування mTSP виникає в реальному сценарії, оскільки він здатний працювати з кількома продавцями. Такі ситуації виникають переважно в різних проблемах маршрутизації та планування. Нижче наведено деякі випадки застосування в літературі.

і. Проблема планування друкарської машини: одне з основних і основних застосувань mTSP виникає при плануванні друкарського верстата для періодичного видання з кількома тиражами. Тут існує п'ять пар циліндрів, між якими одночасно друкуються рулони паперу та обидві сторони сторінки. Існує три типи форм, а саме 4-, 6- та 8-сторінкові форми, які використовуються для друку видань. Проблема планування полягає в тому, щоб вирішити, яка форма буде на якому циклі, і тривалість кожного циклу. У словнику mTSP витрати на зміну номерних знаків є міжміськими витратами. Для отримання більш детальної інформації можна послатися на статті Gorenstein (1970) і Carter & Ragsdale (2002).

іі. Проблема маршрутизації шкільного автобуса: (Angel та ін., 1972) досліджували проблему планування автобусів як варіант mTSP з деякими побічними обмеженнями. Метою планування є отримання такої моделі завантаження автобуса, щоб мінімізувати кількість маршрутів, підтримувати

мінімальну загальну відстань, пройдену всіма автобусами, жоден автобус не був перевантажений, а час, необхідний для проходження будь-якого маршруту, не перевищував максимум дозволена політика.

iii. Проблема планування екіпажу: заявка на перенесення депозиту між різними філії банків повідомляє (Svestka & Huckfeldt, 1973). Тут депозити потрібно забрати у відділеннях банків і повернути в центральний офіс бригадою посильних. Проблема полягає в тому, щоб визначити маршрути, які мають загальну мінімальну вартість. Дві подібні програми описані (Lenstra & Rinnooy Kan, 1975 та Zhang et al., 1999). Документи можуть бути направлені на відкладений аналіз.

iv. Проблема планування інтерв'ю: (Gilbert & Hofstra, 1992) знайшли застосування mTSP, що має багатоперіодні варіації, для планування інтерв'ю між туристичними брокерами та продавцями індустрії туризму. Кожному брокеру відповідає продавець, який повинен відвідати певний набір стендів продавців, які представлені набором T міст.

v. Проблема планування гарячої прокатки: у металургійній промисловості замовлення плануються на стані гарячої прокатки таким чином, щоб загальні витрати на налаштування під час виробництва можна було мінімізувати. Подробиці нещодавнього застосування моделювання такої проблеми можна прочитати в (Tang et al., 2000). Тут замовлення розглядаються як міста, а відстань між двома містами береться як вартість штрафу за перехід виробництва між двома замовленнями. Рішення моделі дасть повний графік для стану гарячої прокатки.

vi. Проблема планування місії. Проблема планування місії полягає у визначенні оптимального шляху для кожного армійця (або планувальника) для досягнення цілей місії за мінімально можливий час. Планувальник місії використовує варіацію mTSP, де є n планувальників, m цілей, які повинні відвідати деякі планувальники, і базове місто, до якого всі планувальники повинні зрештою повернутися. Про застосування mTSP у плануванні місії повідомляють (Brummit & Stentz, 1996; Brummit & Stentz, 1998; і Yu et al., 2002). Подібним

чином проблеми маршрутизації, що виникають під час планування застосування безпілотних літальних апаратів, досліджені (Ryan et al., 1998), також можуть бути змодельовані як mTSP.

vii. Розробка геодезичних мереж глобальної навігаційної супутникової системи

Цікаве застосування mTSP, як досліджено (Saleh & Chelouah, 2004), виникає при розробці геодезичних мереж глобальної навігаційної супутникової системи (GNSS). GNSS — це космічна супутникова система, яка забезпечує охоплення всіх місць у всьому світі та має вирішальне значення в реальних програмах, таких як раннє попередження та управління наслідками катастроф, моніторинг навколишнього середовища та сільського господарства тощо. Метою зйомки є визначення географічного розташування невідомих точок на землі та над нею за допомогою супутникового обладнання. Ці точки, на яких розміщені приймачі, координуються серією сеансів спостереження. Коли є кілька приймачів або кілька робочих періодів, проблема пошуку найкращого порядку сеансів для приймачів може бути сформульована як mTSP.

1.2 Аналіз програм-аналогів задачі комівояжера

Для її розв'язання можна використовувати генетичні алгоритми. Існує багато програмних додатків, які реалізують задачу комівояжера з використанням генетичних алгоритмів.

1. Concorde TSP Solver розв'язує задачу комівояжера для графів з до 85 000 вершинами. Програма має відкритий вихідний код і є безкоштовною.
2. LKH LKH (Lin-Kernighan-Helsgaun) є програмою для розв'язання задачі комівояжера, яка базується на алгоритмі Ліна-Кернігана. Програма є дуже ефективною та може розв'язувати задачі з великою кількістю вершин. LKH також має відкритий вихідний код та є безкоштовною.
3. Ant Colony Optimization Ant Colony Optimization (ACO) - це програма для розв'язання задачі комівояжера, яка базується на імітації поведінки мурах. ACO може розв'язувати задачі з великою кількістю вершин та є

ефективною для вирішення задачі комівояжера з додатковими обмеженнями. Програма має відкритий вихідний код та є безкоштовною.

Таблиця 1.1 – Порівняльний аналіз програм-аналогів задачі комівояжера

Додаток	Особливості	Переваги	Недоліки
TSPLIB95	Безкоштовний, може працювати з файлами у форматі TSPLIB	Є багато різних графів, можливість порівняти різні алгоритми та реалізації	Не має графічного інтерфейсу, тільки командний рядок
Concorde TSP Solver	Може розв'язувати найбільші задачі комівояжера, має реалізації для багатьох мов програмування	Дуже швидкий та точний, використовуєтьс я в наукових дослідженнях	Платний, складно налаштовуват и
Ant Colony Optimizatio n	Використовує мурахиний алгоритм, є безкоштовним та має графічний інтерфейс	Ефективний та точний, може працювати з великими графами	Може бути повільним для складних задач, потребує налаштування параметрів

1.3 Обгрунтування сфери застосування генетичних алгоритмів

Генетичний алгоритм (GA) був запропонований американським вченим Холландом у 1975 році. Це свого роду алгоритм оптимізації та ефективного пошуку, який імітує теорію природного відбору та генетику, засновану на виживанні найпристосованіших. Завдяки сильній здатності вирішувати проблеми та широкій адаптивності, останніми роками він проник у різні галузі досліджень та інженерії та досяг хороших результатів. Генетичний алгоритм — це різновид алгоритму випадкового глобального пошуку, який шукає цільовий простір випадковим чином. Він розглядає можливі рішення в проблемній області як особину або хромосому популяції та кодує кожну особину в двійковій системі чи нотації з плаваючою комою, щоб досягти параметризації моделі, одну за одною відображаючи в хромосомному просторі хромосоми, повторювані генетичні групи на основі операція, згідно з попередньо визначеною цільовою функцією кожної особини, яку необхідно оцінити через базовий процес генетичної операції, і повторювана ітераційна оптимізація розведення для отримання нового покоління, продовжує отримання кращої групи, тоді як глобальний паралельний пошук для пошуку найкращого осіб у групі оптимізації, щоб отримати оптимальне рішення для задоволення вимог.

Хромосома: при використанні GA проблему потрібно розв’язати у вигляді фіксованого рядка символів, кожен із яких представляє ген. Хромосома представляє рішення проблеми. Кожну хромосому називають особиною. Популяція (Population): загальна кількість хромосом, утворених за покоління. Група містить набір деяких рішень проблеми в цьому поколінні. Фітнес: кожній людині відповідає рішення конкретної проблеми. Значення функції, що відповідає кожному розв’язку, є функцією придатності. Це індикатор для вимірювання відповідності хромосоми навколишньому середовищу та об’єктивна функція, яка відображає фактичну проблему.

Процес розв’язування за допомогою GA базується на наборі параметрів проблеми, яку потрібно розв’язати, популяція генерується випадковим чином,

обчислюються функція відповідності та швидкість відбору, а також виконуються генетичні операції, такі як відбір, схрещування та мутація. Якщо умова ітераційної конвергенції задовольняється, популяція є найкращими індивідами, інакше, генерація нового покоління груп для повторної генетичної операції, зворотно-поступального циклу, доки не будуть виконані умови.

Основними генетичними операціями є: Виберіть, відповідно до певної ймовірності з попереднього покоління, щоб вибрати M особин як батьків, безпосередньо скопійованих у наступне покоління, хромосома не змінюється. Одна з найпоширеніших і найпростіших ймовірностей вибору: Ps (xi) = f (xi) / Eif (xi), де f (xi) — придатність моделі xi. Кросовер (Crossover) — це процес випадкового вибору двох особин із старої популяції, обмін генетичною інформацією та створення потомства. Мутація (Mutation) Тобто процес генерації нових генів, вибір групи індивідуумів (хромосом), випадковий вибір біта для інвертованої операції. Це може запобігти сходженню генетичного алгоритму до локального оптимального рішення, допомогти розширити область оптимізації та розширити можливості пошуку. Особливо в останній частині генетичного алгоритму, коли особини та значення пристосованості в популяції подібні, подальша еволюція популяції залежить від операції мутації. Підсумовуючи, процес пошуку оптимального рішення за допомогою генетичного алгоритму можна розділити на наступні етапи: 1) кодування задачі, що розв’язується; 2) випадково заданий набір початкових розв’язків X (0) = (X1, X2, ..., Xn); (3) оцінка продуктивності поточної групи та обчислення придатності f (xi) для кожного окремого x у поточній групі x (t); вибір певної кількості рішень з поточного рішення як об’єктів генетичної операції відповідно до результату оцінки; генетична операція обраного розчину для отримання нового набору розчинів; повернутися для оцінки нового рішення; якщо поточне рішення для задоволення вимог або еволюційного процесу досягає певного значення, розрахунок завершується або продовжується. Генетичний алгоритм у перших кількох ітераціях, поява індивідів добре і погано співіснують, пристосованість невисока, зі збільшенням кількості

ітерацій особи з високою пристосованістю супроводжувалися генетичними. Генетичні алгоритми мають багато переваг для вирішення задач оптимізації, таких як вибір умов. Зокрема, GA має високий порядок пошуку, а пошук має дослідницькі та саморозвиваючі можливості.

Генетичний алгоритм є результатом мультидисциплінарної інтеграції та проникнення та перетворився на самоорганізовану та адаптивну інтегровану технологію. Як ефективний глобальний метод пошуку він широко використовується в багатьох галузях, включаючи інженерне проектування, виробництво, штучний інтелект, інформатику, біоінженерію, розвідку нафти, автоматичне керування, соціальні науки, комерцію та фінанси. Розвідка нафти в основному використовується в багатьох областях, таких як прогнозування видобутку нафтових родовищ, оптимізація розробки нафтових родовищ, оптимізація інтерпретації каротажу та визначення розподілу проникності пласта. Генетичні алгоритми заповнюють недолік традиційних методів оптимізації та показують свої особливості та привабливість у вирішенні та застосуванні багатьох задач у галузі розвідки та розробки нафти та газу. Однак на даний момент все ще є деякі недоліки та недоліки як у теорії, так і в застосуванні генетичних алгоритмів. На практиці генетичні алгоритми часто схильні до передчасної конвергенції та поганої конвергенції, а генетичні алгоритми не є універсальними. З точки зору вирішення проблем вони не можуть повністю замінити існуючі методи оптимізації в певній галузі, і всі вони мають власну сферу застосування . Для конкретної області генетичні алгоритми часто не можуть зрівнятися з алгоритмами, які займаються проблемами в цій області.

Задача оптимізації компенсації реактивної потужності в енергосистемі є багатопараметричною та багатообмеженою задачею змішаного нелінійного програмування. Керовані змінні включають безперервні змінні (такі як напруга вузла та реактивна потужність генератора), а також дискретні змінні, механізм РПН перемикача навантаження, перемикальну групу компенсаційного конденсатора), що робить процес оптимізації дуже складним. Однак

традиційний метод лінійного програмування, метод нелінійного програмування та метод змішаного цілочисельного програмування, які застосовуються для компенсації оптимізації реактивної потужності, мають проблеми наближеної обробки дискретних змінних і не відповідають фактичним вимогам планування реактивної потужності. Тому, однак, все ще існують деякі загальні обмеження, які легко потрапити в локальну екстремальну область і не можуть досягти точної обробки дискретних змінних. Завдяки власним характеристикам генетичний алгоритм особливо підходить для вирішення задачі багатокритерійної гібридної оптимізації та має стабільні характеристики збіжності. Це розширений метод глобальної оптимізації. Оптимізація реактивної потужності – це набір початкових рішень за умовами енергосистеми. Генетичний алгоритм обмежений різними обмеженнями, а його переваги та недоліки оцінюються функцією придатності. Від низького значення придатності відмовилися. Лише високе значення придатності Мають можливість повторити свої характеристики для наступного раунду рішень, шляхом відбору, кросинговеру, мутації та інших генетичних операцій, популяція поступово прагне стати найкращою та, нарешті, отримати оптимальне рішення. Недоліком простого генетичного алгоритму є те, що він займає багато часу для обчислень і може легко сходитися до локального екстремуму, якщо сукупність і генетична алгебра недостатньо великі. З цієї причини пропонується вдосконалений генетичний алгоритм або комбінація генетичного алгоритму та інших алгоритмів для підвищення швидкості обчислень і точності обчислень. У цьому документі генетичний алгоритм відпалу використовується в поєднанні з імітованим алгоритмом вибору відпалу, тобто для використання вибору відпалу як індивідуальної стратегії заміни, щоб уникнути попадання в локальне оптимальне рішення. Перша частина, алгоритм реактивної оптимізації, була наведена раніше. Інша частина, тобто мета системи керування реактивною потужністю та напругою в режимі реального часу, полягає у використанні прикладного програмного забезпечення EMS для дистанційного керування відводами трансформатора та батареями

конденсаторів на підстанції в регіональному диспетчерському центрі, щоб реалізувати безпечне та економічна експлуатація обленерго. Як система управління в режимі реального часу, не тільки для забезпечення достатньо хорошого оптимального рішення, але й для забезпечення оптимального рішення, відповідного контролю виконання програми. Тому, окрім вирішення проблем теорії та алгоритму, слід також вирішити велику кількість технічних проблем у реалізації проекту, включаючи обробку обмежень, методи запуску, обробку мертвої зони, міркування безпеки та впровадження програми.

2. ПРОЕКТУВАННЯ СИСТЕМИ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА

2.1 Розробка структурної схеми застосування генетичного алгоритму в задачі комівояжера

Основою паралельного генетичного алгоритму (PGA) є генетичний алгоритм (GA), який є класом глобального, адаптованого та ймовірнісного алгоритму оптимізації та революції пошуку, отриманого з моделі еволюції органіки, а також моделює генетику та еволюцію біологічної популяції в природи. GA приймає природну еволюційну модель, таку як відбір, кросовер, мутація, видалення та перенесення. З математичної точки зору цей еволюційний процес є типовим алгоритмом пошуку оптимального рішення за допомогою ітераційного пошуку серед багатьох елементів у наборі недетермінованого поліноміального часу (NP). Простий генетичний алгоритм (SGA) можна визначити як SGA=(M, C, F, o, Ps, Pc, Pm, T), де C — фіксований рядок бітів, F — функція оцінки придатності, M — початкова сукупність біологічної колонії, а Ps, Pc, Pm – ймовірності відбору, кросинговеру та мутації відповідно.

При розв'язанні задач NP серією GA великий простір вибірки збільшить довжину хромосом. А це призводить до збільшення часової складності алгоритму.

Ми змінили послідовний генетичний алгоритм для паралельної обробки. Нарешті, часова складність зменшується. PGA використовує дві основні модифікації порівняно з генетичним алгоритмом. Спочатку розподіляється відбір для спаровування. Люди живуть у двовимірному світі. Вибір партнера здійснюється кожною особиною самостійно в її сусідстві (представлено на рис.2.1).

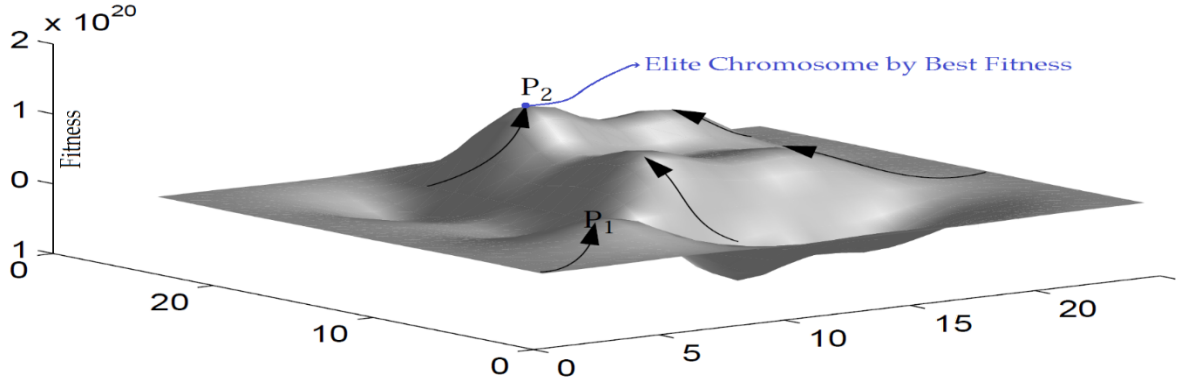

Рисунок 2.1 – Паралельність в генетичному алгоритмі

PGA є повністю асинхронним, працює з максимальною ефективністю на паралельних комп'ютерах MIMD (кілька інструкцій, багато даних — це техніка, яка використовується для досягнення паралелізму в обчисленнях). Стратегія пошуку PGA базується на невеликій кількості активних і розумних осіб, тоді як GA використовує велику кількість пасивних осіб. Абстрактно PGA — це паралельний пошук з обміном інформацією між особинами в одній популяції.

Паралельні обчислення широко використовуються сьогодні для вирішення трудомістких задач. Задача комівояжера є добре відомою комбінаторною задачею.

Ідея TSP полягає в тому, щоб знайти найкоротший тур групою міст без відвідування будь-якого міста двічі, але, практично, це передбачає побудову гамільтонового циклу в межах зваженого повнозв'язного неорієнтованого графа. Отже, це задача комбінаторного пошуку графа. TSP, можливо, є однією з найбільш досліджуваних проблем у інформатиці. Застосування проблеми TSP численні – у комп'ютерній проводці, плануванні роботи, мінімізації споживання палива в літаках, проблемі маршруту транспортного засобу, навчанні роботів тощо.

Загалом запропоновано 4 моделі реалізації паралельних генетичних алгоритмів:

Одиночне населення Господар / Раб (фітнес)

Одиночна популяція дрібнозернистих або клітинних PGA

Багатонаселення (зі рівнем міграції)

Ієрархічний

У всіх наведених вище методах паралельного генетичного алгоритму мета оптимізації дизайну з точки зору генетики вважалася кращою відповіддю, а не для швидкості обчислень. GA з кількома популяціями також широко використовуються паралельні методи, але вони є більш складними, ніж методи однієї популяції. Ключовою характеристикою багатопопуляційних PGA є міграція особин між субпопуляціями. Кожна субпопуляція управляється незалежною SGA, за винятком того, що процесори періодично обмінюються особами. Обчислювальне навантаження в такому розмірі та діапазоні призводить до зниження швидкості системи.

GA з однією популяцією зазвичай реалізуються за допомогою моделі ведучий-підлеглий. У моделі головний-підлеглий одна популяція знаходиться в головному процесорі, а головний процесор виконує відбір, кросинговер і мутацію; тільки оцінка функції придатності розподіляється між підлеглими процесорами. Глобальна модель головного-підлеглого з єдиною популяцією, яку ми використовуємо, проілюстрована на малюнку 2.2. Частина популяції розподіляється між кожним підлеглим процесором для оцінки значення придатності окремих осіб. Головний процесор також зберігає частину популяції, щоб він міг виконувати оцінку паралельно з підлеглими процесорами. Генетичні операції, крім оцінки, виконуються лише головним процесором. Головний процесор призначає частку населення кожному підпорядкованому процесору для кожного покоління.

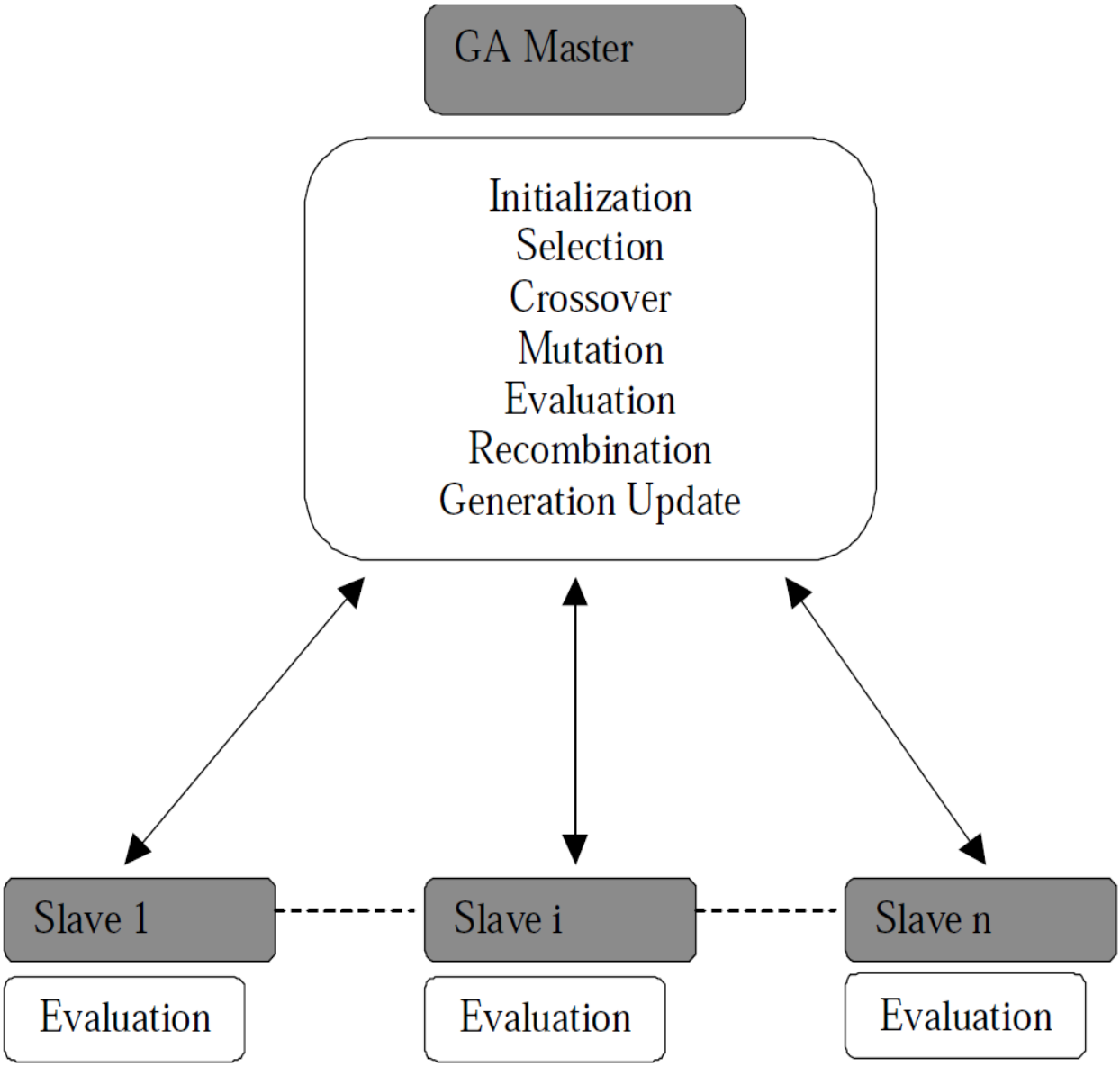

Рисунок 2.2 – Структурна схема застосування генетичного алгоритму в задачі комівояжера

2.2 Формування UML-діаграм системи застосування генетичного алгоритму в задачі комівояжера

Стандарт UML, який застосовується до об'єктно-орієнтованого підходу, забезпечує відповідні погляди на систему, так що в усіх термінах система може бути описана зі статичного (структурного) і динамічного аспектів. Він використовується для розробки програмного забезпечення, яке потребує плану, пропонує можливість візуалізації в багатьох вимірах і рівнях деталізації та

підходить для оновлення старих систем. Цілком певно, що така дія спростить процес отримання рішення.

У цьому документі представлено моделювання спочатку через статичні, а потім динамічні діаграми. Це хороший спосіб вирішення, оскільки UML описує вихідний код, моделі допомагають візуалізувати систему такою, якою вона є або якою вона має бути, і дозволяють визначити структуру та поведінку системи. Моделі документують рішення, які ми приймали, і пропонують рішення, якими ми керувалися під час побудови системи та конкретних програм.

Діаграма 2.3 випадків використання відображає функціональні вимоги, яким повинна відповідати система. Він складається з одного актора та випадків використання, що випливають з опису генетичних алгоритмів (ініціалізація, оцінка, відбір, схрещування та мутація). Усі випадки вживання у відповідному контексті, і між ними існує взаємодія.

Діаграма 2.4 – це дії, які виконуються, а саме ця діаграма дає загальне уявлення про дії, які далі декомпозуються. Потім наступні діаграми представляють декомпозицію згаданих видів діяльності на піддіяльності, що містять конкретні дії (діаграма 2.5 і 2.6). Діаграма 2.5 дає візуальне представлення того, як розраховується процес відбору, так звана функція придатності та ймовірність для кожного члена окремо, тоді як у діаграмі 2.6 ми вказали, як використання отриманих значень із попередньої діяльності та процедури відбору учасників ґрунтується на їх вірогідності.

Діаграма послідовності 2.7 представляє так званий оператор перетину GSX і спосіб його роботи. Ця діаграма представляє взаємну взаємодію між об’єктами (батьківським, дочірнім, методами GSX), яка загалом представляє серію обмінів повідомленнями між класами, з чітко позначеною послідовністю та часовим ходом надсилання та отримання повідомлень. Діаграми 2.8 і 2.9 представляють дії, які здійснюються в рамках діяльності «схрещування», і

описують оператора GSX з іншого боку. На попередній діаграмі послідовності чітко вказано порядок і взаємодію між об’єктами, які беруть участь у цих діях, і ці діаграми показують, що акцент робиться на діях та їх виконанні та на існуючих умовах. Діаграма 2.10 є декомпозицією дій «мутацій» і представляє дії, що виконуються в цьому операторі.

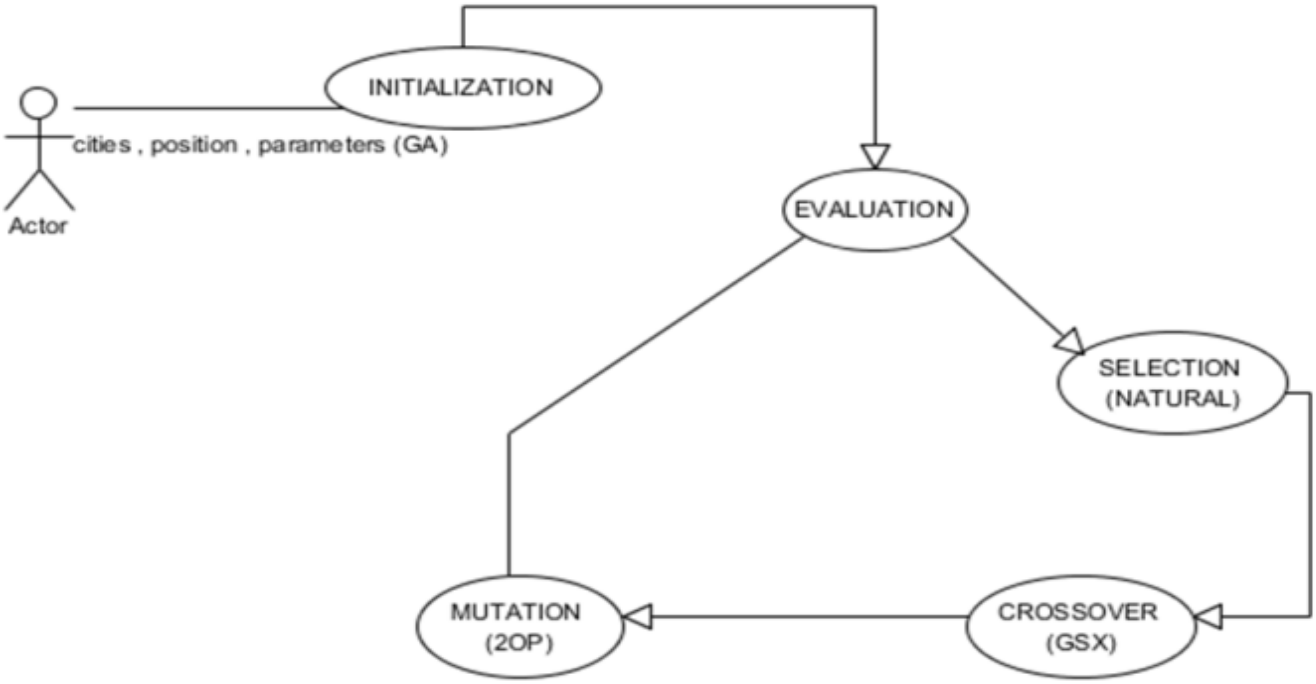

Рисунок 2.3 – Діаграма прецедентів

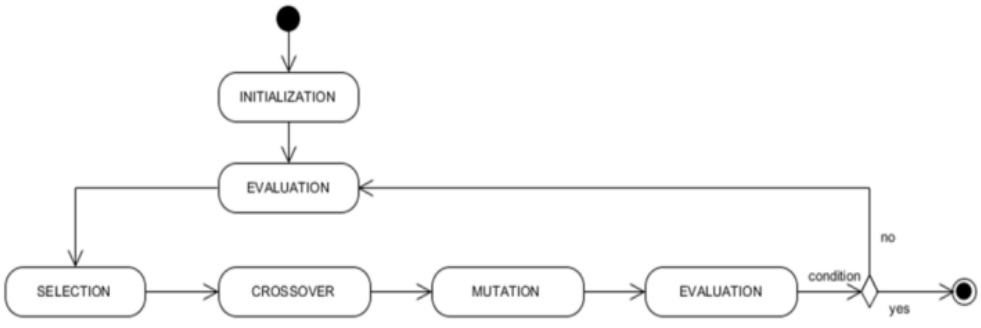

Рисунок 2.4 – Діаграма діяльності

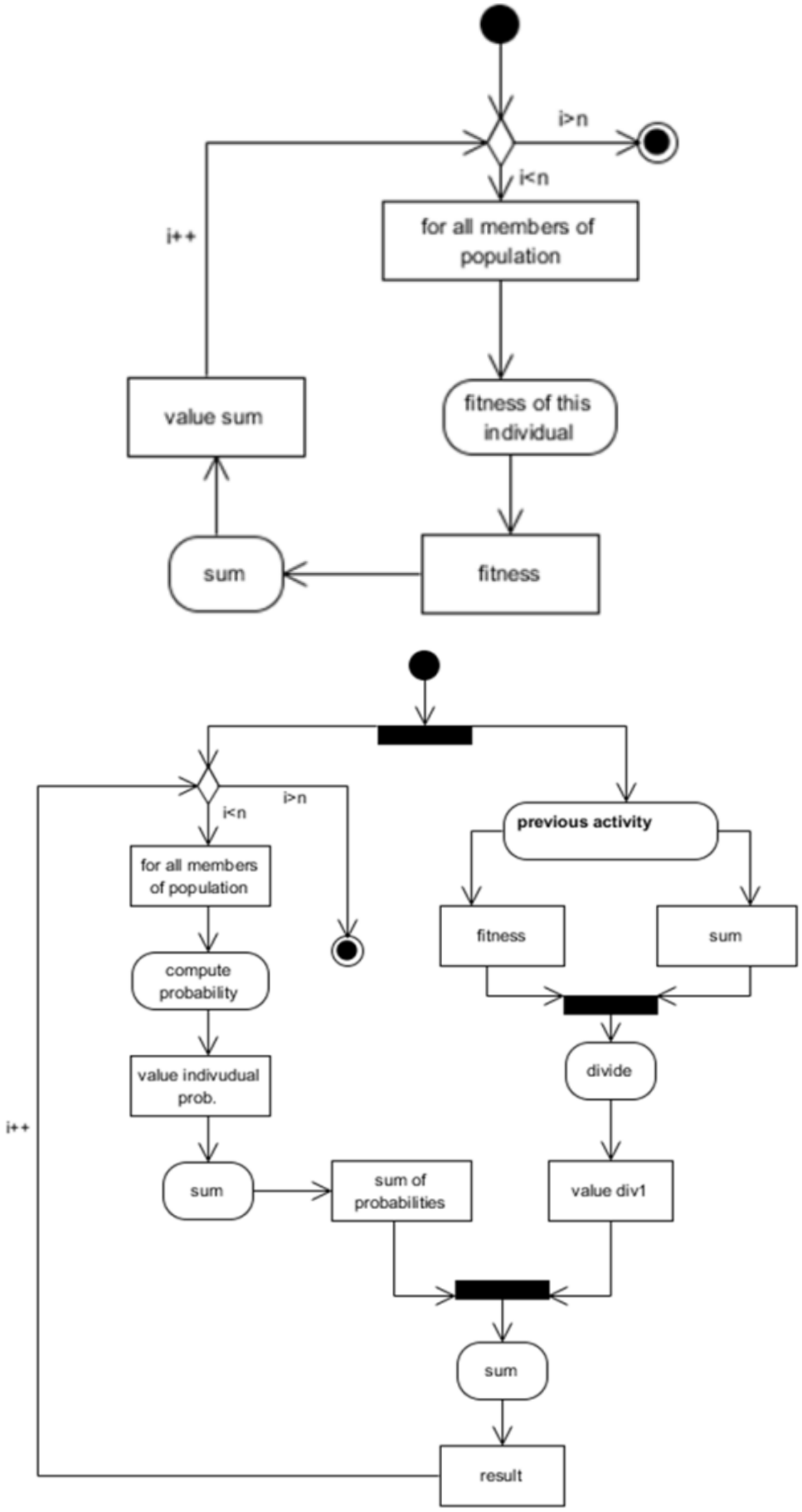

Рисунок 2.5 – Деталізовані діаграми діяльності генетичної селекції

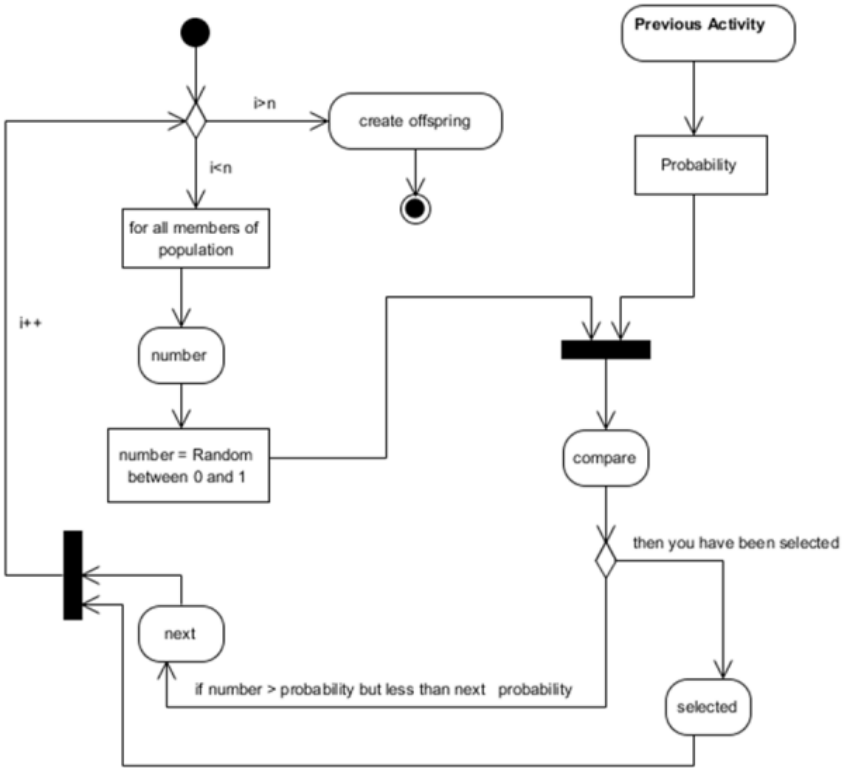

Рисунок 2.6 – Деталізована діаграма діяльності (узагальнена)

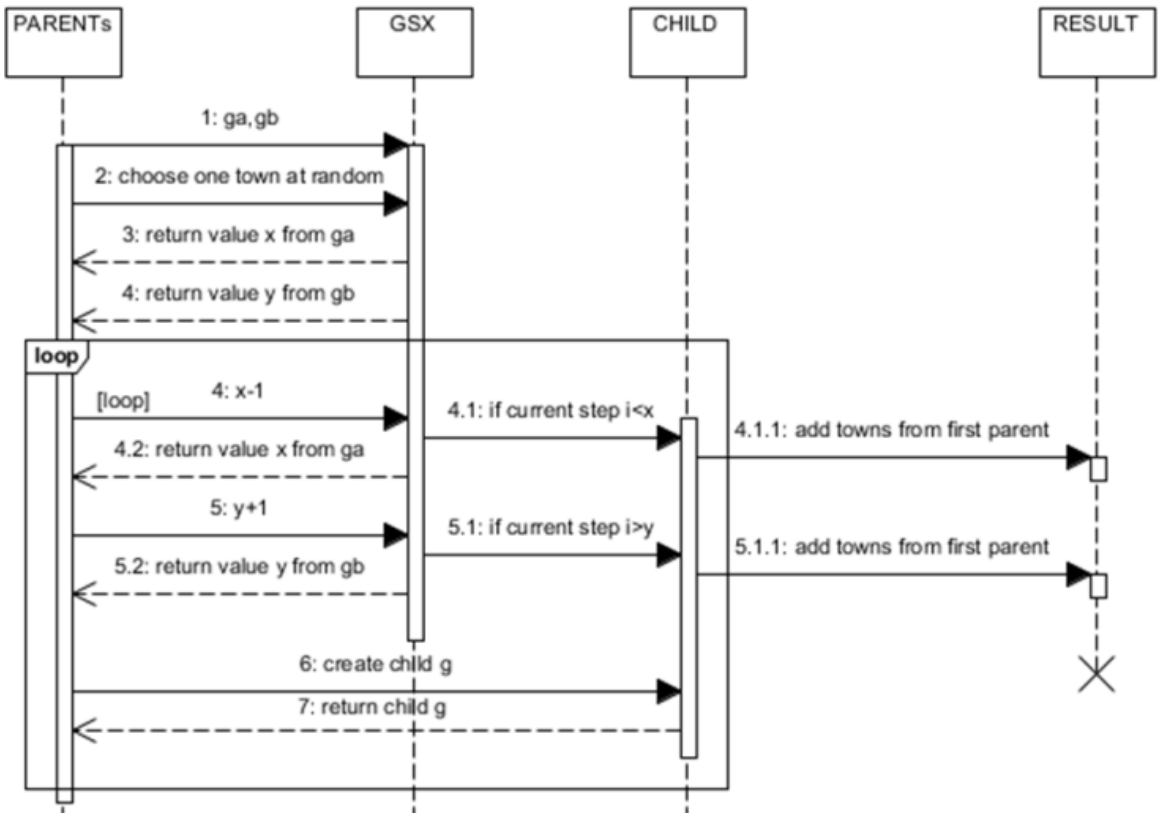

Рисунок 2.7 –Діаграма послідовності

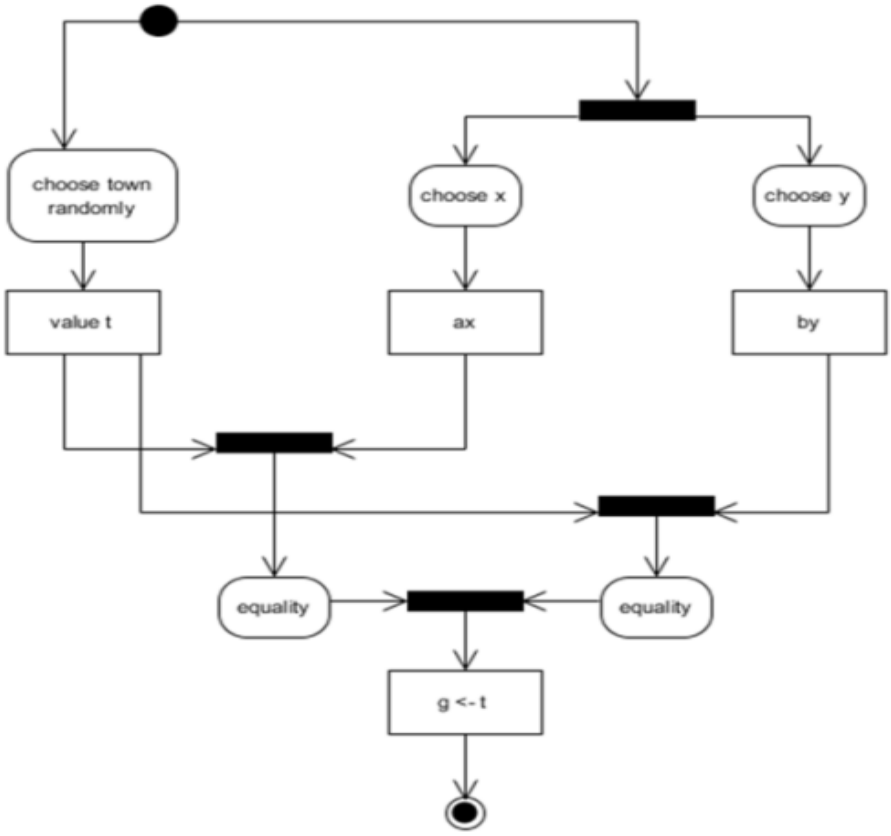

Рисунок 2.8 – Діаграма діяльності для процедури кросоверу

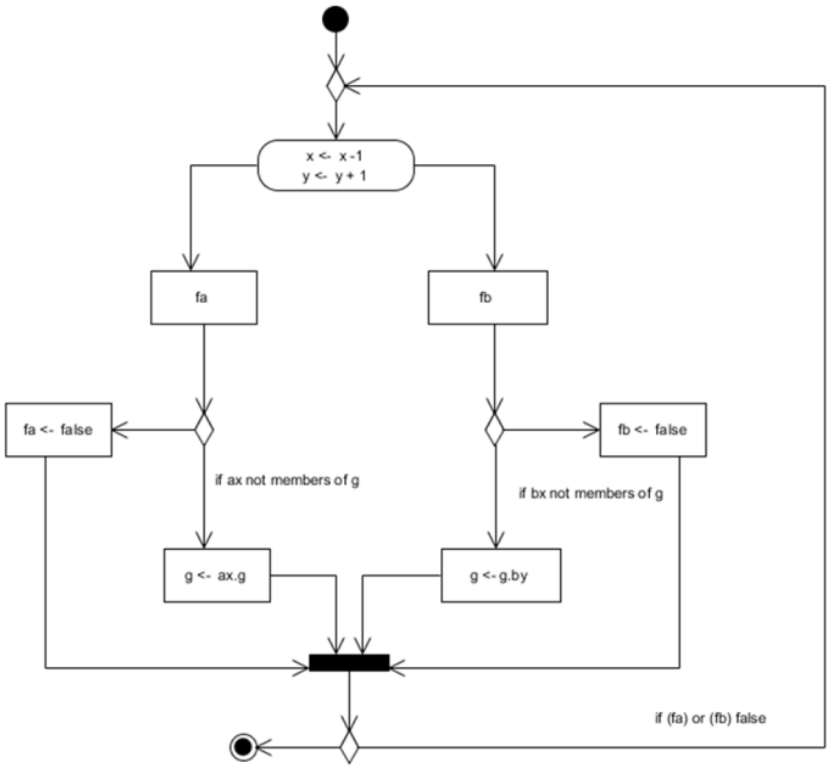

Рисунок 2.9 – Діаграма дільності для кросоверу за жадібним алгоритмом

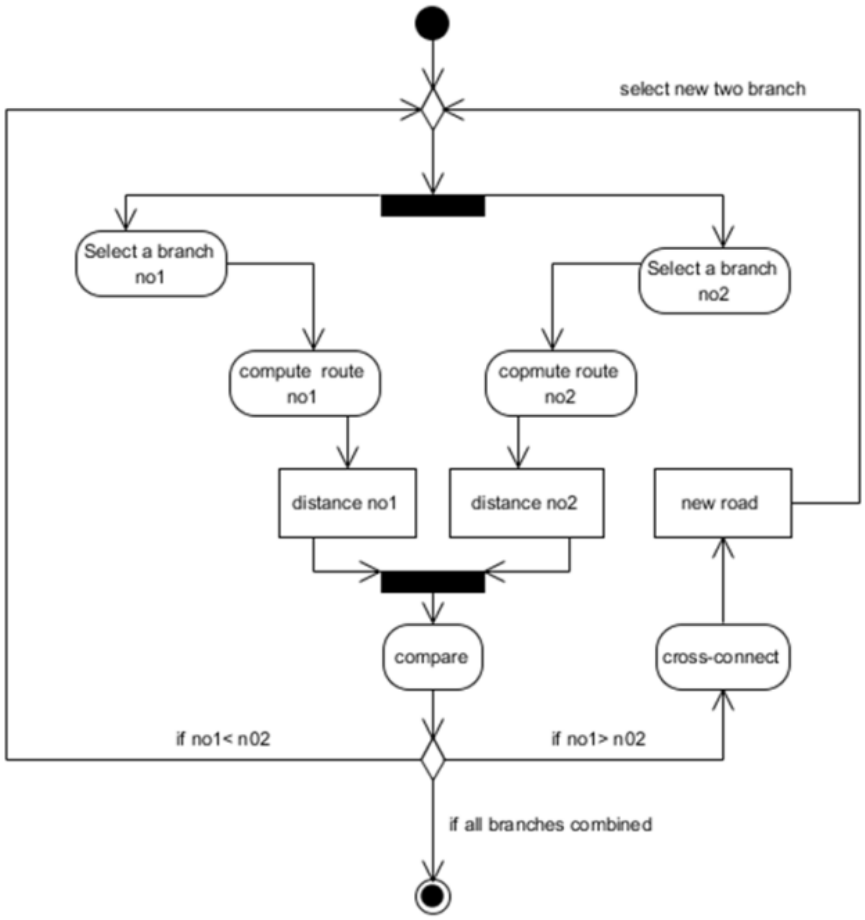

Рисунок 2.10 – Діаграма діяльності мутації

2.3 Проектування схеми роботи системи застосування генетичного алгоритму в задачі комівояжера

Керівник (майстер) після того, як рандомізує початкову популяцію, виконує масив для сортування відповідно до придатності, потім кількість гірших хромосом (відповідно до рівня відбору) у масиві видаляється, доки не буде дозволено виконати їх нову дочірню заміну. Перед тим, як розміщення спрацює, кількість хромосом, яка має нове виробництво та розміщення, визначається вибраною функцією; потім шляхом поділу заміна розраховується на кількість системних процесорів, може знати, скільки нащадків повинен створити кожен робочий елемент, і, нарешті, цей параметр із початковим індексом параметрів масиву менеджера надсилається робочим завданням.

Найважливіша частина нашої роботи виконана на цьому етапі, робітники повинні працювати самостійно і без опорних ниток. І, нарешті, процес генерації був виконаний повністю незалежно. Так само з взаємною винятковістю, коли батьки поділяються на працівників і виробляють нащадків, тим самим підвищуючи незалежність наших процесів. І лише вставка отриманих дітей у глобальний масив населення (який розташований у головному) знаходиться в стані серії. Але його також можна зберегти як відсортований масив у будь-який момент. Зрештою, масив завжди був відсортований, і алгоритм сортування в менеджері не потрібен.

Для розробки паралельного алгоритму було зроблено функціональну декомпозицію паралельної програми та застосовано парадигму менеджер/працівники. Оцінка продуктивності та профілювання паралелізму виконано на основі реалізації багатозадачної програми. Експериментальна паралельна комп’ютерна платформа — це комп’ютер із кількома процесорами, що включає вісім робочих станцій (процесор або комп’ютер), які взаємодіють через батьківське завдання (адміністратор або мережевий комутатор).

Схема роботи системи представлена на рис. 2.11.

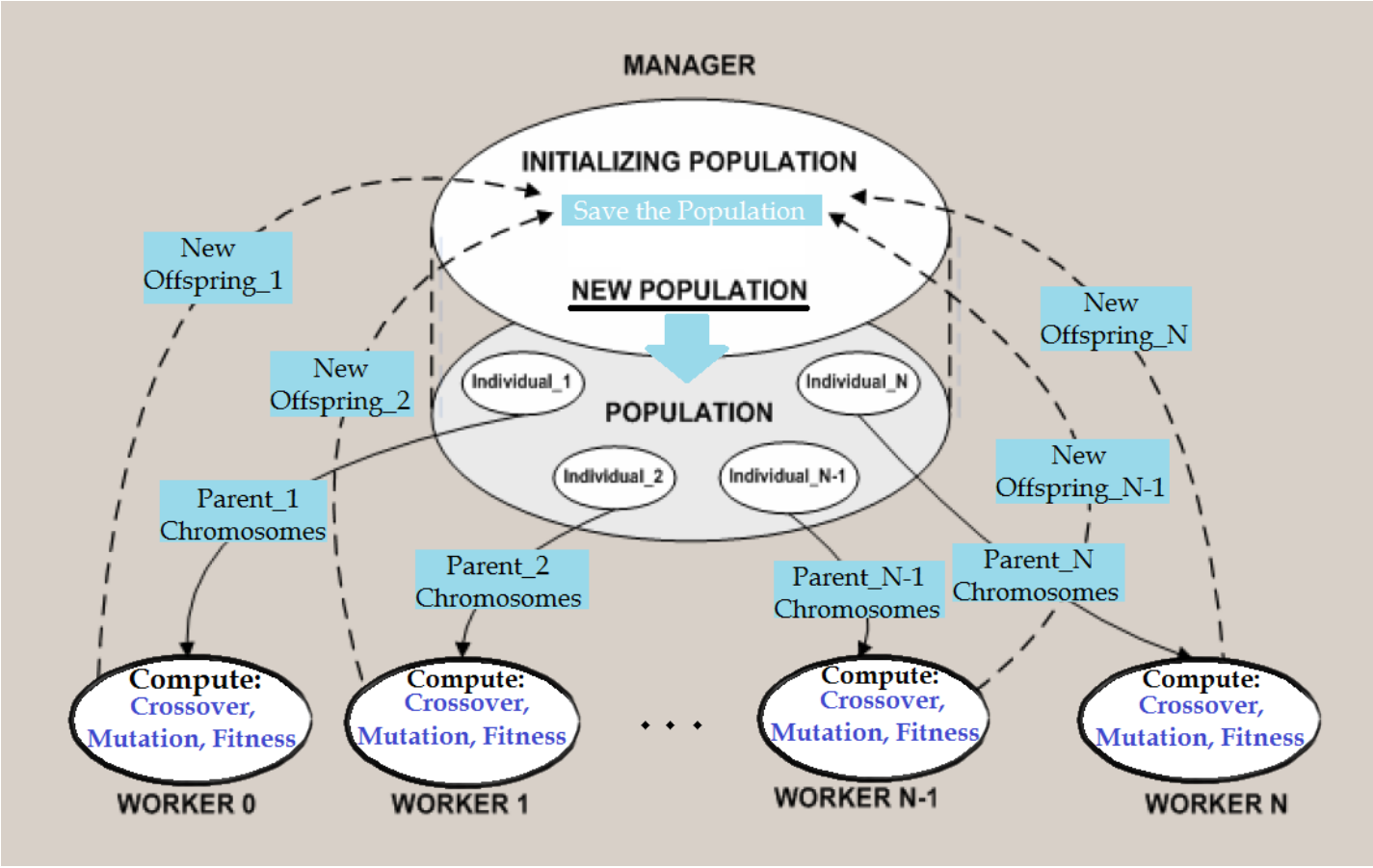

Рисунок 2.11 – Схема роботи паралельної моделі обчислення TSP за генетичним алгоритмом

Керуючий процес (ранг 0) виконує всі генетичні операції для першої популяції та розподіляє обчислювальне навантаження між робочими процесами. Він виконує такі види діяльності:

Ініціалізація популяції (рандомізація)

Сортувати хромосоми за значенням відповідності (хромосома з найнижчим значенням розміщується на першому місці масиву)

Елітність хромосом і виберіть найгіршу хромосому для видалення

Визначення ймовірності відбору відповідно до відповідності хромосом

Будівельники до кількості ядер для виконання генетичних операторів разом виробляють нове покоління

Отримує оцінене потомство від працівників і зберігає нові хромосоми за формулою, проіндексованою в масиві

Створює нову популяцію, поєднуючи будь-яке отримане потомство від робочих у масиві

Перевірити умовне завершення для розрізання циклу

Друкує обчислений найкоротший шлях

Операції робочого процесу такі:

Виконує вибір колеса рулетки

Виконує рекомбінацію (виберіть 2 точки для кросовера)

Виконує звичайну випадкову мутацію (вибираються та обмінюються два міста)

Оцінка придатностінової хромосоми.

3. РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ КОМІВОЯЖЕРА З ГЕНЕТИЧНИМ АЛГОРИТМОМ

3.1 Обгрунтування вибору мови програмування

Розглянемо декілька мов програмування для створення інформаційної технології вибору напряму навчання.

Порівняльна характеристика цих мов подана у таблиці 3.1.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Назва	Опис	Переваги	Недоліки
C#	Розробка ОС та офісних рішень для платформи Microsoft, розробка веб-додатків, настільних графічних додатків (використовуються платформи Windows Forms и WPF), серверних додатків в сфері створення динамічного веб-	Висока швидкість розробки в малобюджетних проектах Простота і лаконічність коду Наявність власного середовища розробки Можливості спадкування й універсалізації Типи, вбудовані в мову, представлені класами	Проблеми при розробці додатків під не-Windows платформи Сильна прив'язка до Microsoft Нема самодостатньості додатків, так як використовується .net framework

	контенту за допомогою платформи ASP.NET, мобільних додатків для операційної системи Windows Phone, що розроблена компанією Microsoft.	Збереження і об’єднання кращих рис мов C і C++ Підвищення ефективності коду, оскільки виконавче середовище CLR являє собою компілятор проміжної мови Зручність побудови різних типів додатків дозволяє легко розробляти Web-служби	Відсутність повноцінних деструкторів, повноцінних макросів, дуже обмежена підтримка шаблонів
C++	Створення ОС, прикладних програм, драйверів приладів, додатків для вбудованих систем, високопродуктивних серверів, а також програмування ігор.	Висока сумісність з мовою C Підтримка різних стилів програмування Можливість побудови загальних контейнерів і алгоритмів для різних типів даних Кросплатформенність Доступність і популярність	Погано продуманий синтаксис робить мову незручною в деяких задачах Нема самодостатньості додатків, для цього використовується runtime Продуктивність праці

			програмістів на мові виявляється невиправдано низькою, а продукт праці є низькоякісним
Java	Створення десктопів і аплетів, мобільних додатків, серверних додатків, що орієнтовані на роботу з мережею, програмування ігор.	Універсальність і широта застосування Кросплатформенність Відкритий вихідний код і велика кількість бібліотек Гнучка система безпеки Висока продуктивність Багатозадачність	Мова не має таких властивостей об'єктно-орієнтованої мови, як індивідуальні змінні та множинне спадкування Відсутність спливаючих підказок до методів, які можна примінити до певного об’єкта
Delphi	Об'єктно-орієнтована мова програмування, нащадок Pascal, більш відома як	Підтримка багатьох компіляторів Зручність в створенні експертних систем	Консервативність

	основна мова програмування середовища Delphi.	Якісний графічний інтерфейс Простота	

З усіх перерахованих мов програмування, було вирішено обрати C# як оптимальну мову розробки додатку.

3.2 Обгрунтування вибору середовища розробки

Вибір середовища розробки є дуже важливим етапом у створенні будь-якої програми. Одним з найбільш популярних середовищ розробки є Visual Studio від Microsoft. Це інтегроване середовище розробки (IDE), яке надає розробникам зручні інструменти для програмування, тестування та налагодження програм.

Переваги Visual Studio перед іншими середовищами розробки полягають в наступному:

1. Багата функціональність: Visual Studio містить багато інструментів, таких як автодоповнення коду, відладчик, систему контролю версій та багато іншого. Ці інструменти забезпечують розробників зручною роботою з кодом.
2. Мови програмування: Visual Studio підтримує багато мов програмування, включаючи C#, C++, Visual Basic і багато інших. Це дає можливість розробникам вибрати мову програмування, яка найкраще підходить для їх проекту.
3. Інтеграція з іншими продуктами Microsoft: Visual Studio добре інтегрується з іншими продуктами Microsoft, такими як Microsoft Azure,

Microsoft SQL Server і Microsoft Office. Це дозволяє розробникам легко створювати програми, які взаємодіють з іншими продуктами Microsoft.

4. Спільнота: Visual Studio має велику спільноту розробників, які допомагають один одному з розв'язанням проблем і вирішенням питань. Це дає можливість отримувати швидку підтримку та знайти відповіді на більшість запитань.

3.3 Тестування розробленого додатку

Реалізація паралельної програми (Visual C# + .Net 4 MultiTasking) виконується в середовищі програмування Visual Studio 2010. Профілювання паралелізму виконано для випадку 500 міст. Результати обчислень паралельно та в серії трьох графіків: час-придатність, час-генерація, фітнес-генерація.

Очевидно, що комунікація є досить інтенсивною між менеджером і працівниками, які обмінюються даними про людей (хромосоми та фізичну форму). Комунікаційні транзакції виконуються через Task, тому затримка зв’язку дуже мала. Раніше основним недоліком паралельного генетичного підходу до обчислення TSP з алгоритмічною парадигмою менеджер/працівники є послідовне обчислення генетичних операцій – відбору, мутації та відтворення – процесом менеджера. Але тепер ці операції виконуються на робітниках.

При оцінці запропонованого паралельного алгоритму з послідовним алгоритмом, згідно з малюнками 3.1 і 3.2, можна побачити, що навіть придатність елітних хромосом у послідовних поколіннях покращується швидше. Тому що незалежність у процесах відтворення (батьківська несхожість) у дітей є недостатньою конвергенцією. Результат покращується, щоб швидко досягти бажаної форми. Звичайно,

враховуючи цифри оцінки на прикладі TSP, слід враховувати, що Fitness – це відстань між містами, тож наскільки нижча, ефективніша та краща вартість.

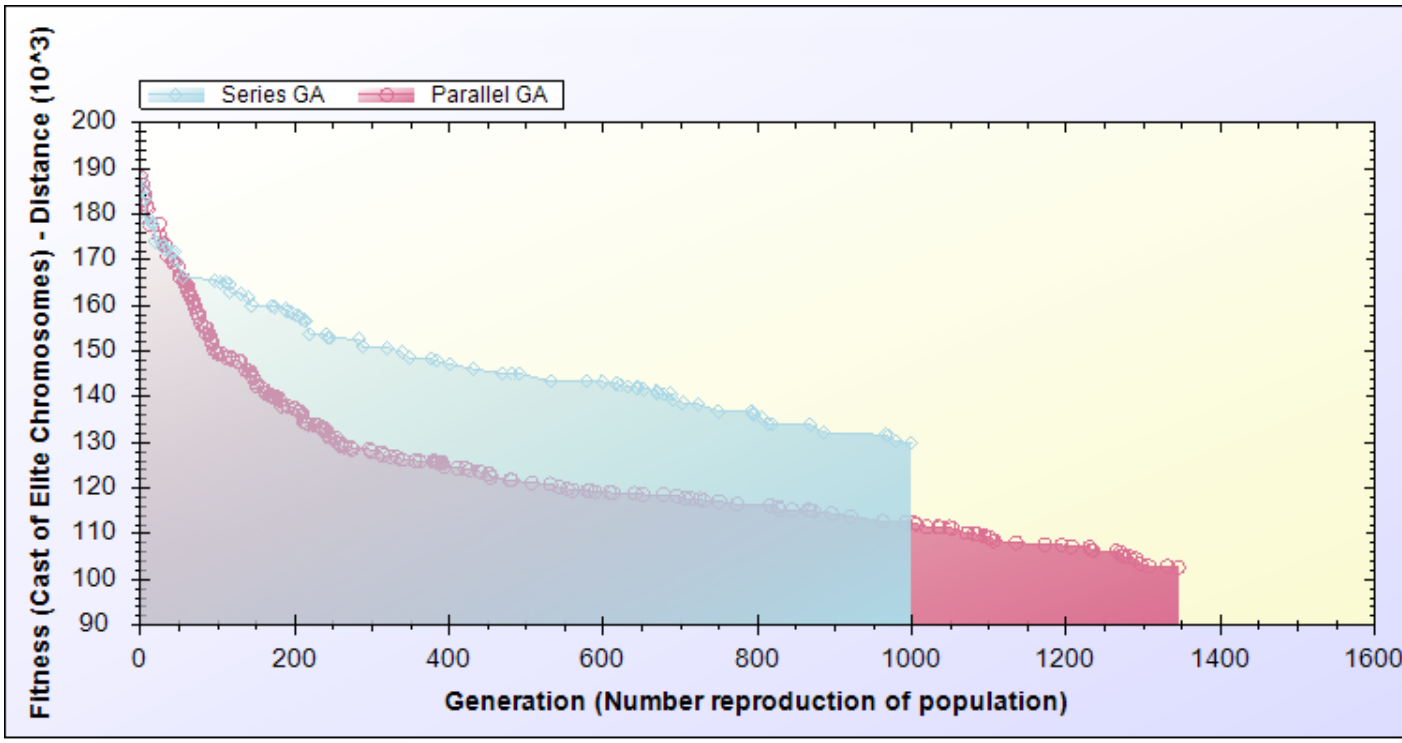

Рисунок 3.1 – Порівняння найкращої придатності будь-якого покоління як у стані послідовної, так і в паралельній обробці

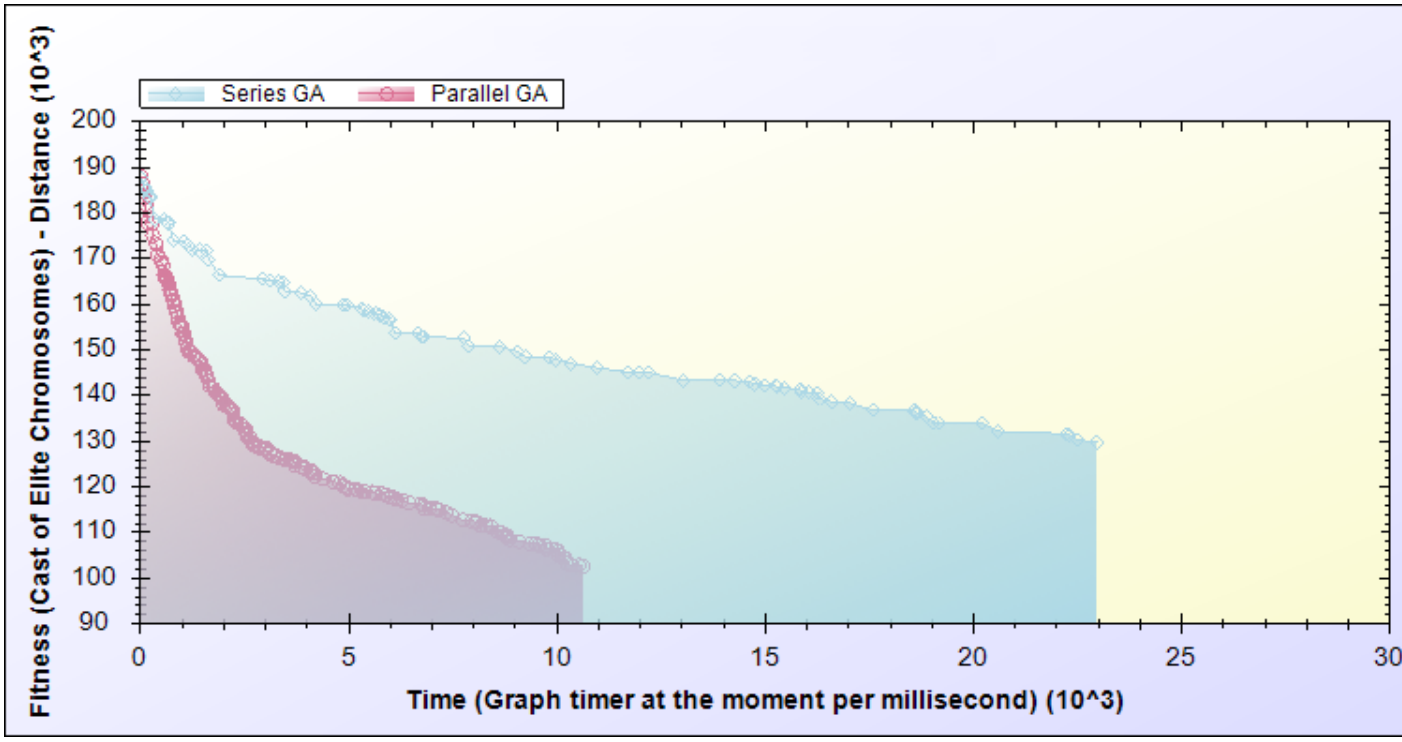

Рисунок 3.2 – Порівняння придатності елітних хромосом

Наступний результат оцінки результатів згідно з рисунком 3.3 полягає в тому, що темпи виробництва продукції є набагато швидшими, ніж стан серії.

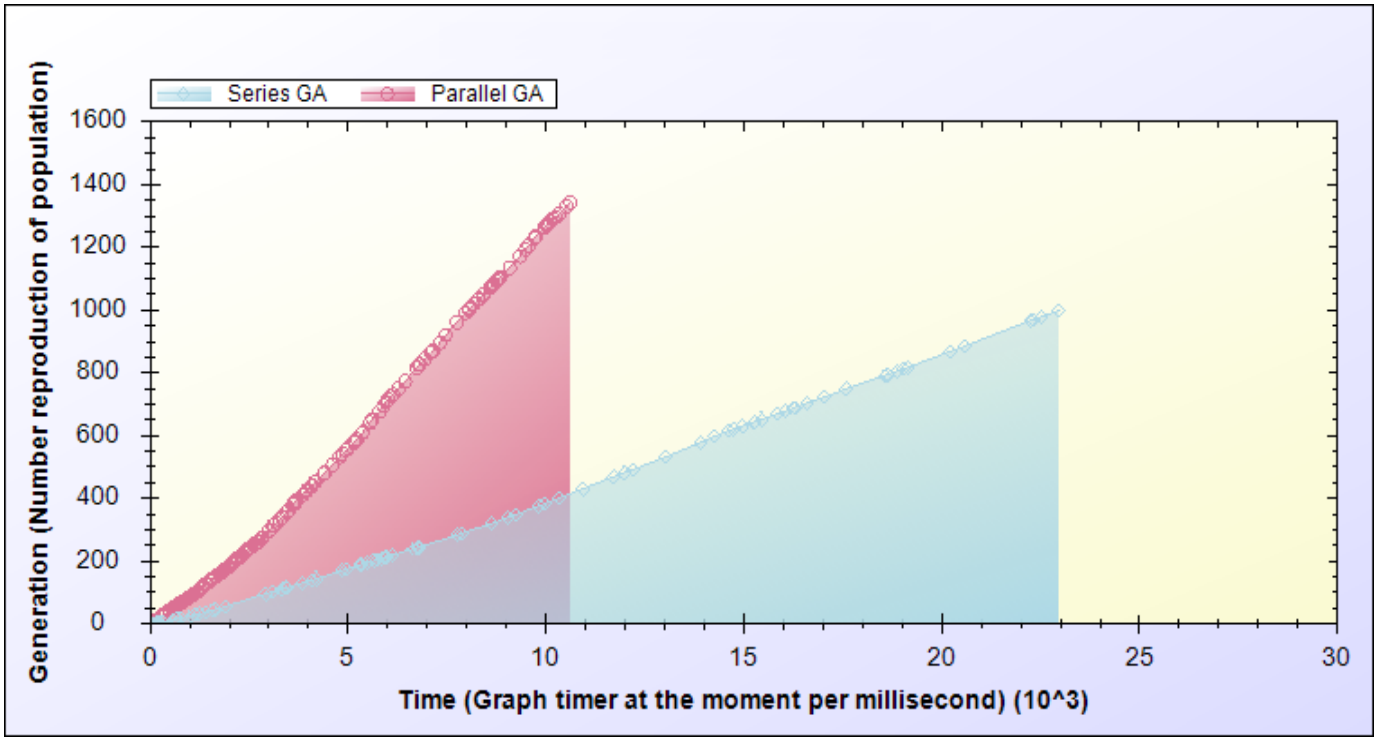

Рисунок 3.3 – Порівняння кількості поколінь, створених у кожен момент серії та паралельної обробки

ВИСНОВКИ

У даній магістерській роботі було проведено дослідження застосування генетичних алгоритмів для розв'язання задачі комівояжера. Було проаналізовано предметну область задачі, обгрунтовано застосування генетичних алгоритмів в розв'язанні даної задачі та розроблено систему, що реалізує задачу комівояжера з генетичним алгоритмом.

У процесі дослідження було проведено порівняльний аналіз програм-аналогів, що розв'язують задачу комівояжера, та вибрано оптимальний підхід до реалізації генетичного алгоритму. Було обрано мову програмування та середовище розробки, що найкраще підходять для розробки даної системи.

В результаті дослідження було розроблено та реалізовано систему з використанням генетичних алгоритмів для розв'язання задачі комівояжера. Було проведено тестування розробленого додатку та отримано результати, що демонструють ефективність та точність роботи системи.

Наукова новизна даної роботи полягає в дослідженні та застосуванні генетичних алгоритмів для розв'язання задачі комівояжера. Отримані результати можуть бути використані в галузі логістики, транспорту та виробництва для вирішення практичних задач.

Отже, дана магістерська робота має велике значення для розвитку досліджень в галузі оптимізації та логістики, а також має практичне значення для вирішення задач виробництва, транспорту та інших галузей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. I. Berninger, “Проблема маршрутизації транспортних засобів на Android/iOS,” Бакалаврська робота, Інститут комп’ютерних наук, дослідницька група DPS, Університет Інсбрука, 2014.
2. A. Homaifar, S. Guan, and G. E. Liepins, “Аналіз схеми проблеми комівояжера за допомогою генетичних алгоритмів,” Складні системи 6, с. 533-552, 1992.
3. A. Reese, “Генератори випадкових чисел в генетичних алгоритмах для безобмеженої та обмеженої оптимізації,” J Нелінійний аналіз, 71, 679–692, 2009.
4. Y. Yun, C. Moon, and D. Kim. “Гібридний генетичний алгоритм з адаптивною схемою локального пошуку для вирішення проблем у сфері багаторівневого постачання,” Comput Ind Eng 56, 821–838, 2009.
5. N. M. Razali and J. Geraghty, “Ефективність генетичного алгоритму з різними стратегіями вибору при розв’язанні проблеми комівояжера,” Процедури Всесвітнього конгресу з інженерії 2011, Том II WCE 2011, 6 - 8 липня 2011, Лондон, Великобританія.
6. K. Rani and V. Kumar, Int. J. Res.Eng. Tech. 2, 27-34. (2014)
7. Z. H. Ahmed, “Експериментальне дослідження гібридного генетичного алгоритму для проблеми максимального комівояжера,” Springer open journal, 2013.
8. K. Bryant, “Генетичні алгоритми та проблема комівояжера,” Магістерська робота, Коледж Харві Мадда, 2000.
9. Y. Wang, “Гібридний генетичний алгоритм з двома стратегіями локальної оптимізації для проблеми комівояжера,” Comput. Ind. Eng. 70, с. 124–133 (2014).

Додаток А Скріншот роботи

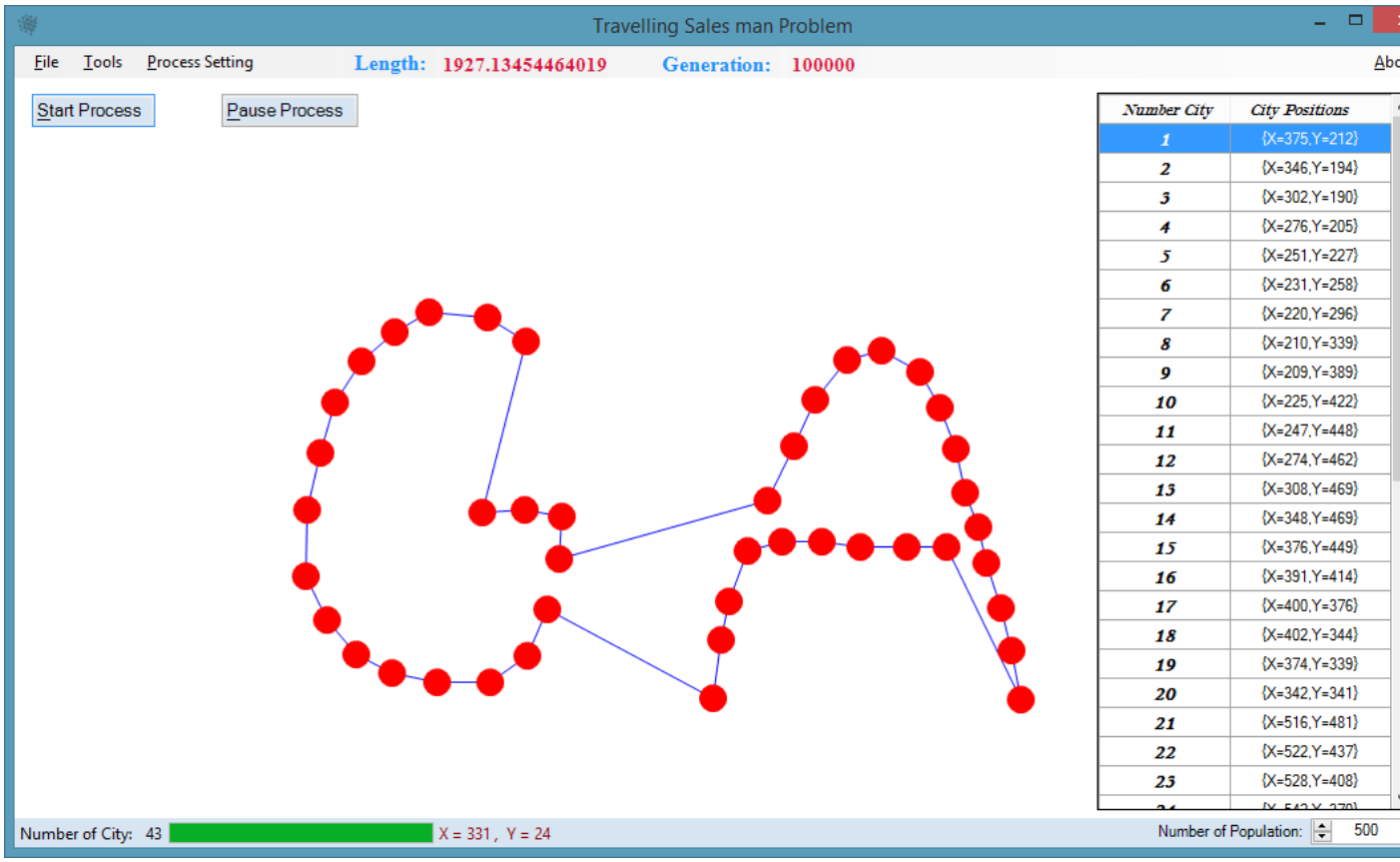

Рисунок А.1 – Скріншот роботи програми

Додаток Б Лістинг програми

```
using
System;
          using System.Collections.Generic;
          using System.Diagnostics;
          using System.Drawing;
          using System.IO;
          using System.Linq;
          using System.Threading;
          using System.Threading.Tasks;
          using System.Windows.Forms;
          using Microsoft.VisualBasic.PowerPacks;
          using TSP.Core;
          using TSP.TimerGraphs;
          using ZedGraph;
          using ThreadState = System.Threading.ThreadState;

          namespace TSP
          {
              public partial class MainForm : Form
              {
                  // Properties
                  [STAThread]
                  static void Main()
                  {
                      Process.GetCurrentProcess().PriorityClass = ProcessPriorityClass.High;
                      Application.EnableVisualStyles();
                      Application.Run(new MainForm());
                  }

                  #region Define Varibale
                  PointPairList[] _pPlistTfg;
                  PointPairList[] _pPlistTgg;
                  PointPairList[] _pPlistGfg;

                  TimeFitnessGraph _tfg;
                  TimeGenerationGraph _tgg;
                  GenerationFitnessGraph _gfg;

                  CancellationTokenSource _tokenSource;
                  int _startedTick = 0;

                  public int CountCpuCore { get; set; } = 1;

                  // Number Population
```

```
        public int PopulationNumber { get; set; } = 500;

        // Number Keep Chromosome Size
        int _nKeep = 0;

        // Double Array Pn for save Rank
        double[] _pn;

        private ShapeContainer ShapeContainerAllCityShape { get; set; }
        private List<LineShape> LineShapeWay { get; set; } = new List<LineShape>();
        public List<OvalShape> OvalShapeCity { get; set; } = new List<OvalShape>();

        // save number of all city
        public int CounterCity { get; private set; }
        public GeneticAlgorithm Genetic { get; private set; }

        // create new process or for end process
        Thread _runTime;

        #endregion

        public MainForm()
        {
            InitializeComponent();

            //
            // shapeContainer_allCityShape
            //
            ShapeContainerAllCityShape = new ShapeContainer
            {
                Location = new Point(0, 0),
                Margin = new Padding(0),
                Size = new Size(Width, Height),
                TabIndex = 0,
                TabStop = false
            };

            Controls.Add(ShapeContainerAllCityShape);
            //
            // Make up some data points from the Sine function
            _pPlistTfg = new PointPairList[2]; // [0] for Series GA and [1] for PGA
(Time-Fitness) data
            _pPlistTfg[0] = new PointPairList(); // For Series GA (Time-Fitness) Data's
            _pPlistTfg[1] = new PointPairList(); // For Parallel GA (Time-Fitness)
Data's

            _pPlistTgg = new PointPairList[2]; // [0] for Series GA and [1] for PGA
(Time-Fitness) data
```

```
            _pPlistTgg[0] = new PointPairList(); // For Series GA (Time-Fitness) Data's
            _pPlistTgg[1] = new PointPairList(); // For Parallel GA (Time-Fitness)
Data's

            _pPlistGfg = new PointPairList[2]; // [0] for Series GA and [1] for PGA
(Time-Fitness) data
            _pPlistGfg[0] = new PointPairList(); // For Series GA (Time-Fitness) Data's
            _pPlistGfg[1] = new PointPairList(); // For Parallel GA (Time-Fitness)
Data's
        }

        #region Thread Invoked
        public static void UiInvoke(Control uiControl, Action action)
        {
            if (!uiControl.IsDisposed)
            {
                if (uiControl.InvokeRequired)
                {
                    uiControl.BeginInvoke(action);
                }
                else
                {
                    action();
                }
            }
        }

        // ------------------------------------------------------------
        // This delegate enables asynchronous calls for setting
        // the Value property on a toolStripProgressBar control.
        delegate void SetValueCallback(int v);
        private void SetValue(int v)
        {
            // InvokeRequired required compares the thread ID of the
            // calling thread to the thread ID of the creating thread.
            // If these threads are different, it returns true.
            try
            {
                if (statusStrip1.InvokeRequired)
                {
                    var d = new SetValueCallback(SetValue);
                    Invoke(d, new object[] { v });
                }
                else
                {
                    toolStripProgressBar1.Value = v;
                }
            }
```

```
    catch { }
}

// ------------------------------------------------------------
// This delegate enables asynchronous calls for setting
// the Maximum.Value property on a toolStripProgressBar control.
delegate void SetMaxValueCallback(int v);
private void SetMaxValue(int v)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    try
    {
        if (statusStrip1.InvokeRequired)
        {
            var d = new SetMaxValueCallback(SetMaxValue);
            Invoke(d, new object[] { v });
        }
        else
        {
            toolStripProgressBar1.Maximum = v;
        }
    }
    catch { }
}

// ------------------------------------------------------------
// This delegate enables asynchronous calls for setting
// the Value property on a toolStripProgressBar control.
private void SetGenerationText(string v)
{
    // InvokeRequired required compares the thread ID of the
    // calling thread to the thread ID of the creating thread.
    // If these threads are different, it returns true.
    try
    {
        UiInvoke(lblGeneration, delegate ()
        {
            lblGeneration.Text = v;
        });
    }
    catch { }
}
// ------------------------------------------------------------
private void SetLenghtText(string v)
{
    try
```

```
    {
        try
        {
            UiInvoke(lblLenght, delegate ()
            {
                lblLenght.Text = v;
            });
        }
        catch { }
    }
    catch { }
}
// ------------------------------------------------------------
delegate void AddShapeCallback(LineShape l);
private void AddLineShape(LineShape l)
{
    try
    {
        if (ShapeContainerAllCityShape.InvokeRequired)
        {
            var d = new AddShapeCallback(AddLineShape);
            Invoke(d, new object[] { l });
        }
        else
        {
            ShapeContainerAllCityShape.Shapes.Add(l);
        }
    }
    catch
    {
        // ignored
    }
}
// ------------------------------------------------------------
delegate void RemoveShapeCallback(LineShape l);
private void RemoveLineShape(LineShape l)
{
    try
    {
        if (ShapeContainerAllCityShape.InvokeRequired)
        {
            var d = new RemoveShapeCallback(RemoveLineShape);
            Invoke(d, new object[] { l });
        }
        else
        {
            ShapeContainerAllCityShape.Shapes.Remove(l);
        }
```

```
            }
            catch { }
        }
        // ------------------------------------------------------------
        delegate void SetPointCallback(int l, Point p0, Point p1);
        private void SetPoint(int l, Point p0, Point p1)
        {
            try
            {
                LineShapeWay[l].X1 = p0.X + 10;
                LineShapeWay[l].X2 = p1.X + 10;
                LineShapeWay[l].Y1 = p0.Y + 10;
                LineShapeWay[l].Y2 = p1.Y + 10;
            }
            catch { }
        }
        // ------------------------------------------------------------
        private void SetNumPopEnable(bool en)
        {
            try
            {
                try
                {
                    UiInvoke(numPopulation, delegate ()
                    {
                        numPopulation.Enabled = en;
                    });
                }
                catch { }
            }
            catch { }
        }
        #endregion

        /// <summary>
        /// Genetic Algorithm for TSP
        /// </summary>
        public void Ga()
        {
            var rand = new Random(DateTime.Now.GetHashCode());
            var eliteFitness = double.MaxValue;
            //
            // set cities position
            SetCitiesPosition(OvalShapeCity);
            //
            // initialize Parallel Computing for GA
            CountCpuCore = CalcCountOfCpu(); // Calculate number of active core or CPU
for this app
```

```
            _tokenSource = new CancellationTokenSource();
            //
            // set Start TickTime
            _startedTick = Environment.TickCount;

            if (pGAToolStripMenuItem.Checked)  // clear Parallel points
            {
                _pPlistTfg[1].Clear();
                _pPlistGfg[1].Clear();
                _pPlistTgg[1].Clear();
            }
            else // clear Series points
            {
                _pPlistTfg[0].Clear();
                _pPlistGfg[0].Clear();
                _pPlistTgg[0].Clear();
            }
            //
            //               N , Pop, SR, MR, ReGen, CR
            Genetic = new GeneticAlgorithm(CounterCity, PopulationNumber, 10, 50,
10000, 75);

            var count = 0;
            SetValue(0);
            //toolStripProgressBar1.Value = 0;
            //
            SetGenerationText("0000");
            //lblGeneration.Text = "0000";
            //
            if (CounterCity <= 5)
                SetMaxValue(100);
            //toolStripProgressBar1.Maximum = 100;
            //
            else if (CounterCity <= 15)
                SetMaxValue(1000);
            //toolStripProgressBar1.Maximum = 1000;
            //
            else if (CounterCity <= 30)
                SetMaxValue(10000);
            //toolStripProgressBar1.Maximum = 10000;
            //
            else if (CounterCity <= 40)
                SetMaxValue(51000);
            //toolStripProgressBar1.Maximum = 51000;
            //
            else if (CounterCity <= 60)
                SetMaxValue(100000);
            //toolStripProgressBar1.Maximum = 100000;
```

```
            //
            else
                SetMaxValue(1000000);
            //toolStripProgressBar1.Maximum = 1000000;
            //
            //

            do
            {
                #region Selection
                #region Bubble Sort all chromosome by fitness
                //
                for (var i = PopulationNumber - 1; i > 0; i--)
                    for (var j = 1; j <= i; j++)
                        if (Genetic.Population[j - 1].Fitness >
Genetic.Population[j].Fitness)
                        {
                            var ch = Genetic.Population[j - 1];
                            Genetic.Population[j - 1] = Genetic.Population[j];
                            Genetic.Population[j] = ch;
                        }
                //
                #endregion

                #region Elitism
                if (eliteFitness > Genetic.Population[0].Fitness)
                {
                    eliteFitness = Genetic.Population[0].Fitness;
                    SetTimeGraph(eliteFitness, count, true);

                    if (dynamicalGraphicToolStripMenuItem.Checked) // Design if
Graphically is ON
                    {
                        RefreshTour();
                    }
                    //
                    //-----------------------------------------------------------------
------------
                    SetLenghtText(Genetic.Population[0].Fitness.ToString());
                    //
                }
                //else setTimeGraph(EliteFitness, count, false); // just refresh
Generation Graph's

                #endregion
                x_Rate(); // Selection any worst chromosome for clear and ...
                #endregion
```

```
                #region Reproduction
                // Definition Probability According by chromosome fitness
                // create Pn[N_keep];
                Rank_Trim();

                if (pGAToolStripMenuItem.Checked) // Parallel Genetic Algorithm
                {
                    if (threadParallelismToolStripMenuItem.Checked) // PGA by
MultiThreading
                    {
                        ReproduceByParallelThreads();
                    }
                    else if (taskParallelismToolStripMenuItem.Checked) // PGA by Task
Parallelism
                    {
                        ReproduceByParallelTask();
                    }
                    else if (parallelForToolStripMenuItem.Checked) // PGA by
Parallel.For ...
                    {
                        PReproduction(rand);
                    }
                }
                else // Series Genetic Algorithm
                {
                    #region Series Reproduct Code
                    Reproduction(rand);
                    #endregion
                }
                #endregion

                count++;
                SetGenerationText(count.ToString());
                //lblGeneration.Text = count.ToString();
                //
                SetValue(toolStripProgressBar1.Value + 1);
                //toolStripProgressBar1.Value++;
                //
            }
            while (count < toolStripProgressBar1.Maximum && Isotropy_Evaluatuon());

            //
            //toolStripProgressBar1.Value = toolStripProgressBar1.Maximum;
            SetValue(toolStripProgressBar1.Maximum);
            //
            // UnLock numUpDownPop
            SetNumPopEnable(true);
            //
```

```
            // The END
            Stop();
        }

        #region Generation Tools

        //find percent of All chromosome rate for delete Amiss(xRate) or Useful(Nkeep)
chromosome
        //x_Rate According by chromosome fitness Average
        private void x_Rate()
        {
            // calculate Addition of all fitness
            double sumFitness = 0;
            for (var i = 0; i < PopulationNumber; i++)
                sumFitness += Genetic.Population[i].Fitness;
            // calculate Average of All chromosome fitness
            var aveFitness = sumFitness / PopulationNumber; //Average of all chromosome
fitness
            _nKeep = 0; // N_keep start at 0 till Average fitness chromosome
            for (var i = 0; i < PopulationNumber; i++)
                if (aveFitness >= Genetic.Population[i].Fitness)
                {
                    _nKeep++; // counter as 0 ~ fitness Average + 1
                }
            if (_nKeep <= 0) _nKeep = 2;
        }

        // Definition Probability According by chromosome fitness
        private void Rank_Trim()
        {
            // First Reserve Possibility Number for every Remnant chromosome
            // chromosome Possibility Function is:
            // (1 + N_keep - No.chromosome) / ( ∑ No.chromosome)
            // Where as at this program No.chromosome Of Array begin as Number 0
            // There for No.chromosome in Formula = No.chromosome + 1
            // then function is: if (n == N_keep)
            // Possibility[No.chromosome] = (n - No.chromosome) / (n(n+1) / 2)
            //
            _pn = new double[_nKeep]; // Create chromosome possibility Array Cell as
N_keep
            double sum = ((_nKeep * (_nKeep + 1)) / 2); // (∑ No.chromosome) == (n(n+1)
/ 2)
            _pn[0] = _nKeep / sum; // Father (Best - Elite) chromosome Possibility
            for (var i = 1; i < _nKeep; i++)
            {
                // Example: if ( Pn[Elite] = 0.4  &  Pn[Elite +1] = 0.2  &  Pn[Elite
+2]  = 0.1 )
                // Then Own:          0 <= R <= 0.4 ===> Select chromosome[Elite]
```

```
                //                  0.4 <  R <= 0.6 ===> Select chromosome[Elite +1]
                //                  0.6 <  R <= 0.7 ===> Select chromosome[Elite +2]
                // etc ...
                _pn[i] = ((_nKeep - i) / sum) + _pn[i - 1];
            }
        }

        // Return Father and Mather chromosome with Probability of chromosome fitness
        private Chromosome Rank(System.Random rand)
        {
            var r = rand.NextDouble();
            for (var i = 0; i < _nKeep; i++)
            {
                // Example: if ( Pn[Elite] = 0.6  &  Pn[Elite+1] = 0.3  &  Pn[Elite+2]
= 0.1 )
                // Then Own:          0 <= R <= 0.6  ===> Select chromosome[Elite]
                //                  0.6 <  R <= 0.9  ===> Select chromosome[Elite +1]
                //                  0.9 <  R <= 1    ===> Select chromosome[Elite +2]
                //
                if (r <= _pn[i]) return Genetic.Population[i];
            }
            return Genetic.Population[0]; // if don't run Modality of 'for' then return
Elite chromosome
        }

        // Check the isotropy All REMNANT chromosome (N_keep)
        public bool Isotropy_Evaluatuon()
        {
            // Isotropy percent is 50% of All chromosome Fitness
            var perIso = Convert.ToInt32(Math.Truncate(Convert.ToDouble(50 * _nKeep /
100)));
            var counterIsotropy = 0;
            var bestFitness = Genetic.Population[0].Fitness;
            //
            // i start at 1 because DNA_Array[0] is self BestFitness
            for (var i = 1; i < _nKeep; i++)
                if (bestFitness >= Genetic.Population[i].Fitness) counterIsotropy++;

            // G.A Algorithm did isotropy and app Stopped
            if (counterIsotropy >= perIso) return false;
            else return true; // G.A Algorithm didn't isotropy and app will continued
        }

        private void ReproduceByParallelThreads()
        {
            #region Parallel Reproduct Code
            var th = new Thread[CountCpuCore];
```

```
            // Create a semaphore that can satisfy up to three
            // concurrent requests. Use an initial count of zero,
            // so that the entire semaphore count is initially
            // owned by the main program thread.
            //
            var sem = new Semaphore(CountCpuCore, CountCpuCore);
            var isAlive = new bool[CountCpuCore];
            var isCompleted = new bool[CountCpuCore];

            var length = (PopulationNumber - _nKeep) / CountCpuCore;
            var divideReminder = (PopulationNumber - _nKeep) % CountCpuCore;

            for (var proc = 0; proc < th.Length; proc++)
            {
                var tt = new ThreadToken(proc,
                    length + ((proc == CountCpuCore - 1) ? divideReminder : 0),
                    _nKeep + (proc * length));

                th[proc] = new Thread((x) =>
                {
                    // Entered
                    sem.WaitOne();
                    isAlive[((ThreadToken)x).No] = true;
                    // work ...
                    PReproduction(((ThreadToken)x).StartIndex, ((ThreadToken)x).Length,
((ThreadToken)x).Rand);
                    // We have finished our job, so release the semaphore
                    isCompleted[((ThreadToken)x).No] = true;
                    sem.Release();
                });
                SetThreadPriority(th[proc]);
                th[proc].Start(tt);
            }

        startloop:
            foreach (var alive in isAlive) // wait parent starter for start all
children.
                if (!alive)
                    goto startloop;

                endLoop:
            sem.WaitOne();
            foreach (var complete in isCompleted) // wait parent to interrupt for
finishes all of children jobs.
                if (!complete)
                    goto endLoop;

            // Continue Parent Work
```

```
            sem.Close();
            #endregion
        }
        private void ReproduceByParallelTask()
        {
            #region Parallel Reproduct Code
            var tasks = new Task[CountCpuCore];

            var length = (PopulationNumber - _nKeep) / CountCpuCore;
            var divideReminder = (PopulationNumber - _nKeep) % CountCpuCore;

            for (var proc = 0; proc < tasks.Length; proc++)
            {
                var tt = new ThreadToken(proc,
                    length + ((proc == CountCpuCore - 1) ? divideReminder : 0),
                    _nKeep + (proc * length));

                tasks[proc] = Task.Factory.StartNew(x =>
                {
                    // work ...
                    PReproduction(((ThreadToken)x).StartIndex, ((ThreadToken)x).Length,
((ThreadToken)x).Rand);
                }, tt, _tokenSource.Token);// TaskCreationOptions.AttachedToParent);
            }

            // When user code that is running in a task creates a task with the
AttachedToParent option,
            // the new task is known as a child task of the originating task,
            // which is known as the parent task.
            // You can use the AttachedToParent option to express structured task
parallelism,
            // because the parent task implicitly waits for all child tasks to
complete.
            // The following example shows a task that creates one child task:
            Task.WaitAll(tasks);

            // or

            //Block until all tasks complete.
            //Parent.Wait(); // when all task are into a parent task
            #endregion
        }
        /// <summary>
        /// Series Create New chromosome with Father & Mather Chromosome Instead of
deleted chromosomes
        /// </summary>
        /// <param name="rand"></param>
        public void Reproduction(System.Random rand) // Series
```

```
        {
            for (var i = _nKeep; i < PopulationNumber; i++)
            {
                //
                // for send and check Father & Mather chromosome
                Chromosome rankFather, rankMather;

                // have a problem (maybe Rank_1() == Rank_2()) then Father == Mather
                // Solve Problem by Loop checker
                do
                {
                    rankFather = Rank(rand);
                    rankMather = Rank(rand);
                }
                while (rankFather == rankMather);
                //
                // CrossoverHelper
                var child = rankMather.Crossover(rankFather, new System.Random());
                //
                //  run MutationHelper
                //
                child.Mutation(new System.Random());
                //
                // calculate children chromosome fitness
                //
                child.Evaluate();

                Interlocked.Exchange(ref Genetic.Population[i], child); // atomic
operation between multiple Thread shared
            }
        }
        /// <summary>
        /// Parallel Create New chromosome with Father & Mather Chromosome Instead of
deleted chromosomes
        /// </summary>
        public void PReproduction(int startIndex, int length, System.Random rand) //
Parallel
        {
            for (var i = startIndex; i < (startIndex + length) && i < PopulationNumber;
i++)
            {
                //
                // for send and check Father & Mather chromosome
                Chromosome rankFather, rankMather;

                // have a problem (maybe Rank_1() == Rank_2()) then Father == Mather
                // Solve Problem by Loop checker
                do
```

```
                {
                    rankFather = Rank(rand);
                    rankMather = Rank(rand);
                }
                while (rankFather == rankMather);
                //
                // CrossoverHelper
                var child = rankMather.Crossover(rankFather, new System.Random());
                //
                //  run MutationHelper
                //
                child.Mutation(new System.Random());
                //
                // calculate children chromosome fitness
                //
                child.Evaluate();

                Interlocked.Exchange(ref Genetic.Population[i], child); // atomic
operation between multiple Thread shared
            }
        }

        /// <summary>
        /// Parallel Create New chromosome with Father & Mather Chromosome Instead of
deleted chromosomes
        /// </summary>
        /// <param name="rand"></param>
        /// <returns></returns>
        public void PReproduction(System.Random rand) // Parallel.For
        {
            Parallel.For(_nKeep, PopulationNumber,
                        new ParallelOptions() { MaxDegreeOfParallelism = CountCpuCore,
CancellationToken = _tokenSource.Token },
                        (i, loopState) =>
                        {
                            // have a problem (maybe Rank_1() == Rank_2()) then Father
== Mather
                            // Solve Problem by Loop checker
                            Chromosome rankFather, rankMather;
                            do
                            {
                                Monitor.Enter(rand);
                                rankFather = Rank(rand);
                                rankMather = Rank(rand);
                                Monitor.Exit(rand);
                            }
                            while (rankFather == rankMather);
                            //
```

```
                            // CrossoverHelper
                            var child = rankMather.Crossover(rankFather, new
System.Random());
                            //
                            //  run MutationHelper
                            //
                            child.Mutation(new System.Random());
                            //
                            // calculate children chromosome fitness
                            //
                            child.Evaluate();
                            Interlocked.Exchange(ref Genetic.Population[i], child); //
atomic operation between multiple Thread shared
                            if (_tokenSource.IsCancellationRequested ||
_tokenSource.Token.IsCancellationRequested)
                            {
                                loopState.Stop();
                                loopState.Break();
                                return;
                            }
                        });
        }

        #endregion

        private void SetCitiesPosition(List<OvalShape> ovalShapeCity)
        {
            Chromosome.CitiesPosition.Clear();
            foreach (var city in ovalShapeCity)
                Chromosome.CitiesPosition.Add(city.Location);
        }

        private void Stop()
        {
            if (_runTime != null)
            {
                if (_runTime.IsAlive)
                {
                    SetNumPopEnable(true); // Enable population numUpDown
                    UiInvoke(btnStartStop, delegate ()
                    {
                        btnStartStop.Checked = false;
                    });
                    UiInvoke(btnPauseResume, delegate ()
                    {
                        btnPauseResume.Checked = false;
                    });
                    try
```

```
            {
                if (pGAToolStripMenuItem.Checked)
                {
                    _tokenSource.Cancel();
                }
                _runTime.Abort();
            }
            catch { }
            RefreshTour();
        }
    }
}

private int CalcCountOfCpu()
{
    var numCore = 0;

    #region Find number of Active CPU or CPU core's for this Programs

    var affinityDec = Process.GetCurrentProcess().ProcessorAffinity.ToInt64();
    var affinityBin = Convert.ToString(affinityDec, 2); // toBase 2
    foreach (var anyOne in affinityBin.ToCharArray())
        if (anyOne == '1') numCore++;

    #endregion

    //if (numCore > 2) return --numCore;
    return numCore;
}

private void SetThreadPriority(Thread th)
{
    if (th != null)
    {
        if (th.ThreadState != ThreadState.Aborted &&
           th.ThreadState != ThreadState.AbortRequested &&
           th.ThreadState != ThreadState.Stopped &&
           th.ThreadState != ThreadState.StopRequested)
        {
            switch (Process.GetCurrentProcess().PriorityClass)
            {
                case ProcessPriorityClass.AboveNormal:
                    th.Priority = ThreadPriority.AboveNormal;
                    break;
                case ProcessPriorityClass.BelowNormal:
                    th.Priority = ThreadPriority.BelowNormal;
                    break;
                case ProcessPriorityClass.High:
```

```
                        th.Priority = ThreadPriority.Highest;
                        break;
                    case ProcessPriorityClass.Idle:
                        th.Priority = ThreadPriority.Lowest;
                        break;
                    case ProcessPriorityClass.Normal:
                        th.Priority = ThreadPriority.Normal;
                        break;
                    case ProcessPriorityClass.RealTime:
                        th.Priority = ThreadPriority.Highest;
                        break;
                }
                //
                // Set Thread Affinity
                //
                Thread.BeginThreadAffinity();
            }
        }
    }

    private void SetTimeGraph(double eliteFitness, long generation, bool
fitnessRefreshed)
    {
        var timeLenght = (Environment.TickCount - _startedTick) / 10; // Convert to
MiliSecond
        if (pGAToolStripMenuItem.Checked)
        {
            if (fitnessRefreshed)
            {
                _pPlistTfg[1].Add(timeLenght, eliteFitness);
                _pPlistGfg[1].Add(generation, eliteFitness);
            }
            _pPlistTgg[1].Add(timeLenght, generation);
        }
        else
        {
            if (fitnessRefreshed)
            {
                _pPlistTfg[0].Add(timeLenght, eliteFitness);
                _pPlistGfg[0].Add(generation, eliteFitness);
            }
            _pPlistTgg[0].Add(timeLenght, generation);
        }
    }

    private void refreshDGV_CityPositions()
    {
        dgvCity.Rows.Clear();
```

```
            for (var count = 0; count < OvalShapeCity.Count; count++)
            {
                dgvCity.Rows.Add(new object[] { count + 1,
OvalShapeCity[count].Location.ToString() });
            }
        }

        private void create_City(Point e)
        {
            CounterCity++;
            toolStripStatuslblNumCity.Text = CounterCity.ToString();
            var newCity = new OvalShape();
            //
            // newCity
            //
            newCity.BackColor = Color.Red;
            newCity.BackStyle = BackStyle.Opaque;
            newCity.BorderColor = Color.Red;
            newCity.Cursor = Cursors.Hand;
            newCity.Location = new Point(e.X, e.Y);
            newCity.Size = new Size(20, 20);
            newCity.Click += ovalShape_Click;

            OvalShapeCity.Add(newCity);
            ShapeContainerAllCityShape.Shapes.Add(newCity);
        }

        private void RefreshTour()
        {
            try
            {
                Point point1, point0;
                for (var c = 1; c <= CounterCity; c++)
                    try
                    {

//this.shapeContainer_allCityShape.Shapes.Remove(lineShape_Way[c]);
                        RemoveLineShape(LineShapeWay[c]);
                        //
                    }
                    catch { break; }

                for (var c = 1; c < CounterCity; c++)
                {
                    // pop[0] is Elite chromosome or best less Distance ---------------
--------
                    point1 = OvalShapeCity[Genetic.Population[0].Genome[c]].Location;
```

```
                    point0 = OvalShapeCity[Genetic.Population[0].Genome[c -
1]].Location;

                    try
                    {
                        var d = new SetPointCallback(SetPoint);
                        BeginInvoke(d, new object[] { c, point0, point1 });
                    }
                    catch { }

                    //this.shapeContainer_allCityShape.Shapes.Add(lineShape_Way[c]);
                    AddLineShape(LineShapeWay[c]);
                    //
                }
                // design line between city 0 & last city
                // pop[0] is Elite chromosome or best less Distance
                point1 = OvalShapeCity[Genetic.Population[0].Genome[CounterCity -
1]].Location;
                point0 = OvalShapeCity[Genetic.Population[0].Genome[0]].Location;

                try
                {
                    var d2 = new SetPointCallback(SetPoint);
                    BeginInvoke(d2, new object[] { 0, point0, point1 });
                }
                catch { }

                //this.shapeContainer_allCityShape.Shapes.Add(lineShape_Way[0]);
                AddLineShape(LineShapeWay[0]);
            }
            catch { }
        }

        private void ovalShape_Click(object sender, EventArgs e)
        {
            CounterCity--;
            OvalShapeCity.Remove((OvalShape)sender);
            ShapeContainerAllCityShape.Shapes.Remove(((OvalShape)sender)); // Remove
Selected Shape
            // Minus 1 as City Number's
            toolStripStatuslblNumCity.Text = CounterCity.ToString();
            //
            // Refresh City Positions List
            refreshDGV_CityPositions();
        }

        private void MainForm_MouseMove(object sender, MouseEventArgs e)
        {
```

```
            toolStripStatuslblLocate.Text = "X = " + e.X.ToString() + " ,  Y = " +
e.Y.ToString();
        }

        private void MainForm_MouseClick(object sender, MouseEventArgs e)
        {
            var mPosition = new Point(e.X - 10, e.Y - 10);
            if (mPosition.X > 1 && mPosition.X < Width - 300 && mPosition.Y > 65 &&
mPosition.Y < Height - 85)
            {
                Stop();
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
                LineShapeWay.Clear();
                create_City(mPosition);
                //
                // Refresh City Positions List
                refreshDGV_CityPositions();
            }
        }

        private void numPopulation_ValueChanged(object sender, EventArgs e)
        {
            PopulationNumber = (int)numPopulation.Value;
        }

        private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
        {
            Stop();
        }

        private void newToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Stop();
            //
            // Remove Old City and road
            //
            foreach (var city in OvalShapeCity)
                ShapeContainerAllCityShape.Shapes.Remove(city);
            foreach (var anyLine in LineShapeWay)
                ShapeContainerAllCityShape.Shapes.Remove(anyLine);
            OvalShapeCity.Clear();
            CounterCity = 0;
            //
            // Refresh City Position List
            //
            refreshDGV_CityPositions();
            //
```

```
            // Clear All Label
            //
            toolStripProgressBar1.ProgressBar.Value = 0;
            lblGeneration.Text = "0000";
            lblLenght.Text = "0000";
            toolStripStatuslblNumCity.Text = "0";
        }

        private void importToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var ofd = new OpenFileDialog();
            ofd.Title = "Open City Positions";
            ofd.RestoreDirectory = true;
            ofd.Filter = "Text files|*.txt";
            ofd.DefaultExt = "CityPositions.txt";

            if (ofd.ShowDialog() == DialogResult.OK)
            {
                try
                {
                    _runTime.Abort();
                }
                catch { }

                //
                // Remove Old City and road
                //
                foreach (var city in OvalShapeCity)
                    ShapeContainerAllCityShape.Shapes.Remove(city);
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
                OvalShapeCity.Clear();
                CounterCity = 0;
                //
                // Create New City
                //
                var cityPositions = File.ReadAllLines(ofd.FileName);
                foreach (var cityP in cityPositions)
                {
                    var startIndexX = cityP.IndexOf("{X=",
StringComparison.CurrentCultureIgnoreCase) + 3;
                    var endIndexX = cityP.IndexOf(",",
StringComparison.CurrentCultureIgnoreCase);
                    var x = int.Parse(cityP.Substring(startIndexX, endIndexX -
startIndexX));

                    var startIndexY = cityP.IndexOf(",Y=",
StringComparison.CurrentCultureIgnoreCase) + 3;
```

```
                    var endIndexY = cityP.IndexOf("}",
StringComparison.CurrentCultureIgnoreCase);
                    var y = int.Parse(cityP.Substring(startIndexY, endIndexY -
startIndexY));
                    create_City(new Point(x, y));
                }
                //
                // Refresh City Position List
                //
                refreshDGV_CityPositions();
            }
        }

        private void exportToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var sfd = new SaveFileDialog
            {
                RestoreDirectory = true,
                Title = @"Save City Positions",
                Filter = @"Text files|*.txt",
                DefaultExt = "CityPositions.txt"
            };

            if (sfd.ShowDialog() == DialogResult.OK)
            {
                var postions = new List<string>();
                foreach (var city in OvalShapeCity)
                {
                    postions.Add(city.Location.ToString());
                }
                File.WriteAllLines(sfd.FileName, postions.ToArray());
            }
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Stop();
            for (var th = 0; th < Process.GetCurrentProcess().Threads.Count; th++)
                Process.GetCurrentProcess().Threads[th].Dispose();
            Application.Exit();
        }

        private void dynamicalGraphicToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            dynamicalGraphicToolStripMenuItem.Checked =
!dynamicalGraphicToolStripMenuItem.Checked;
```

```
            if (dynamicalGraphicToolStripMenuItem.Checked) RefreshTour();

            toolsToolStripMenuItem.ShowDropDown();
        }

        private void timerFitnessGraphToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (timerFitnessGraphToolStripMenuItem.Checked)
            {
                _tfg.Dispose();
                timerFitnessGraphToolStripMenuItem.Checked = false;
            }
            else if (_pPlistTfg != null)
            {
                _tfg = new TimeFitnessGraph();
                _tfg.timerGraphToolStripMenuItem = timerFitnessGraphToolStripMenuItem;
                _tfg.PPlist = _pPlistTfg;
                _tfg.Show();
            }
            toolsToolStripMenuItem.ShowDropDown();
        }

        private void generationFitnessGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (generationFitnessGraphToolStripMenuItem.Checked)
            {
                _gfg.Dispose();
                generationFitnessGraphToolStripMenuItem.Checked = false;
            }
            else if (_pPlistGfg != null)
            {
                _gfg = new GenerationFitnessGraph();
                _gfg.timerGraphToolStripMenuItem =
generationFitnessGraphToolStripMenuItem;
                _gfg.PPlist = _pPlistGfg;
                _gfg.Show();
            }
            toolsToolStripMenuItem.ShowDropDown();
        }

        private void timerGenerationGraphToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (timerGenerationGraphToolStripMenuItem.Checked)
            {
                _tgg.Dispose();
```

```
                timerGenerationGraphToolStripMenuItem.Checked = false;
            }
            else if (_pPlistTgg != null)
            {
                _tgg = new TimeGenerationGraph();
                _tgg.timerGraphToolStripMenuItem =
timerGenerationGraphToolStripMenuItem;
                _tgg.PPlist = _pPlistTgg;
                _tgg.Show();
            }
            toolsToolStripMenuItem.ShowDropDown();
        }

        private void newRandomCitiesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var enrpForm = new EnterNumberRandomPointForm();
            if (enrpForm.ShowDialog() == DialogResult.OK)
            {
                //
                // Clear old data
                //
                newToolStripMenuItem_Click(sender, e);
                //
                // Create Random Points between Parent Form Size
                // Min X=1 , Y=84
                // Max X=FormSize.X-300 , Y=FormSize.Y-85
                // ...
                var rand = new System.Random();
                for (var citiesNo = 0; citiesNo < enrpForm.NumberOfCities; citiesNo++)
                {
                    //
                    // find new safely points according by contact rate:
                    Point newPoint;
                    bool safely;
                    var maxSafelyPoint = new Point(); // save best safely point if do
not found any safely Points
                    double bestFitness = 0; /// save distance between maxSafelyPoint
and newPoint
                    var maxSafelyLoopsNo = enrpForm.NumberSafety; // Probability for
find new Safely points
                    do
                    {
                        newPoint = new Point(rand.Next(1, Width - 300), rand.Next(65,
Height - 85));
                        safely = true;
                        foreach (var otherCity in OvalShapeCity) // Check Safety!
                        {
                            if (Math.Abs(otherCity.Location.X - newPoint.X) < 24 &&
```

```
                                Math.Abs(otherCity.Location.Y - newPoint.Y) < 24)
                            {
                                safely = false;
                                var fitness = Math.Sqrt(Math.Pow((newPoint.X -
otherCity.Location.X), 2) + Math.Pow((newPoint.Y - otherCity.Location.Y), 2));
                                if (fitness > bestFitness)
                                {
                                    bestFitness = fitness;
                                    maxSafelyPoint = newPoint;
                                }
                                break;
                            }
                        }
                        maxSafelyLoopsNo--;
                    } while (!safely && maxSafelyLoopsNo > 0);
                    if (!safely) newPoint = maxSafelyPoint;
                    //
                    create_City(newPoint);
                }
                //
                // Refresh City Position List
                //
                refreshDGV_CityPositions();
            }
        }

        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var about = new FormAbout();
            about.ShowDialog();
        }

        private void btnStartStop_CheckedChanged(object sender, EventArgs e)
        {
            if (btnStartStop.Checked)
            {
                btnPauseResume.Enabled = true;
                btnStartStop.Text = "Stop Process ×";
                #region Start
                if (OvalShapeCity.Count <= 1)
                {
                    btnStartStop.Checked = false;
                    MessageBox.Show("Please Create some cities for a tour!", "Empty
Genome Exception",
                        MessageBoxButtons.OK, MessageBoxIcon.Error);

                    return;
                }
```

```
//
#region Graphical Works
//
// Disable Population numUpDown
//
numPopulation.Enabled = false;
//
// lineShape_Way
//
foreach (var anyLine in LineShapeWay)
    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
LineShapeWay.Clear();

foreach (var shape in ShapeContainerAllCityShape.Shapes)
{
    if (shape.GetType() != typeof(OvalShape) && shape is Shape s)
    {
        ShapeContainerAllCityShape.Shapes.Remove(s);
    }
}

ShapeContainerAllCityShape.Refresh();

for (var c = 0; c < OvalShapeCity.Count; c++)
{
    var newLine = new LineShape
    {
        BorderColor = Color.Blue,
        Cursor = Cursors.Default,
        Enabled = false
    };
    LineShapeWay.Add(newLine);
}
//
//
#endregion
//
// Solve();
try
{
    if (_runTime?.IsAlive != true)
    {
        _runTime = new Thread(Ga);
        SetThreadPriority(_runTime);
        _runTime.Start();
    }
}
catch
```

```
                {
                    _runTime = new Thread(Ga);
                    SetThreadPriority(_runTime);
                    _runTime.Start();
                }
                #endregion
            }
            else
            {
                if (btnPauseResume.Checked)
                {
                    btnPauseResume.Checked = false;
                }
                btnStartStop.Text = @"&Start Process";
                Stop();
            }
        }

        private void btnPauseResume_CheckedChanged(object sender, EventArgs e)
        {
            if (btnPauseResume.Checked)
            {
                btnPauseResume.Text = @"&Resume Process";
                try
                {
                    if (_runTime.IsAlive)
                        _runTime.Suspend();
                }
                catch { }
            }
            else
            {
                btnPauseResume.Text = @"&Pause Process";
                try { if (_runTime.ThreadState == ThreadState.Suspended)
_runTime.Resume(); }
                catch { }

                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Remove(anyLine);
                foreach (var anyLine in LineShapeWay)
                    ShapeContainerAllCityShape.Shapes.Add(anyLine);
            }
        }

        private void RealtimeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
```

```
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.RealTime;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }

        private void HighToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass = ProcessPriorityClass.High;
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }

        private void AboveNormalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
```

```
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.AboveNormal;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }

        private void NormalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.Normal;
                BelowNormalToolStripMenuItem.Checked = false;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }

        private void BelowNormalToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
```

```
                Process.GetCurrentProcess().PriorityClass =
ProcessPriorityClass.BelowNormal;
                LowToolStripMenuItem.Checked = false;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }

        private void LowToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (((ToolStripMenuItem)sender).Checked)
            {
                RealtimeToolStripMenuItem.Checked = false;
                HighToolStripMenuItem.Checked = false;
                AboveNormalToolStripMenuItem.Checked = false;
                NormalToolStripMenuItem.Checked = false;
                BelowNormalToolStripMenuItem.Checked = false;
                Process.GetCurrentProcess().PriorityClass = ProcessPriorityClass.Idle;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            else
            {
                ((ToolStripMenuItem)sender).Checked = true;
                ProcessPriorityToolStripMenuItem.ShowDropDown();
                SetPriorityToolStripMenuItem.ShowDropDown();
            }
            SetThreadPriority(_runTime);
        }

        private void SetAffinityToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var paf = new ProcessorAffinityForm();
            paf.ShowDialog();
            CountCpuCore = CalcCountOfCpu();
        }

        private void pGAToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // if checked then  Parallel Genetic Algorithm Enable
            pGAToolStripMenuItem.Checked = !pGAToolStripMenuItem.Checked;
```

```
            if (pGAToolStripMenuItem.Checked) taskParallelismToolStripMenuItem.Checked
= true;
            else
            {
                taskParallelismToolStripMenuItem.Checked = false;
                threadParallelismToolStripMenuItem.Checked = false;
                parallelForToolStripMenuItem.Checked = false;
            }
            // show Panel
            ProcessPriorityToolStripMenuItem.ShowDropDown();
            pGAToolStripMenuItem.ShowDropDown();
        }

        private void taskParallelismToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // First change self check to unOlder self check's
            taskParallelismToolStripMenuItem.Checked =
!taskParallelismToolStripMenuItem.Checked;

            // Then check PGA by self
            pGAToolStripMenuItem.Checked = taskParallelismToolStripMenuItem.Checked;

            // and check other by !self
            threadParallelismToolStripMenuItem.Checked = false;
            parallelForToolStripMenuItem.Checked = false;

            // show Panel
            ProcessPriorityToolStripMenuItem.ShowDropDown();
            pGAToolStripMenuItem.ShowDropDown();
        }

        private void threadParallelismToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            // First change self check to unOlder self check's
            threadParallelismToolStripMenuItem.Checked =
!threadParallelismToolStripMenuItem.Checked;

            // Then check PGA by self
            pGAToolStripMenuItem.Checked = threadParallelismToolStripMenuItem.Checked;

            // and check other by !self
            taskParallelismToolStripMenuItem.Checked = false;
            parallelForToolStripMenuItem.Checked = false;

            // show Panel
            ProcessPriorityToolStripMenuItem.ShowDropDown();
            pGAToolStripMenuItem.ShowDropDown();
```

```
        }

        private void parallelForToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // First change self check to unOlder self check's
            parallelForToolStripMenuItem.Checked =
!parallelForToolStripMenuItem.Checked;

            // Then check PGA by self
            pGAToolStripMenuItem.Checked = parallelForToolStripMenuItem.Checked;

            // and check other by !self
            taskParallelismToolStripMenuItem.Checked = false;
            threadParallelismToolStripMenuItem.Checked = false;

            // show Panel
            ProcessPriorityToolStripMenuItem.ShowDropDown();
            pGAToolStripMenuItem.ShowDropDown();
        }

    }

    public struct ThreadToken
    {
        public ThreadToken(int threadNo, int length, int startIndex)
        {
            No = threadNo;
            Length = length;
            StartIndex = startIndex;
            Rand = new System.Random();
        }
        public int No;
        public int Length;
        public int StartIndex;
        public System.Random Rand;
    };
}
```