Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

НАЛАШТУВАННЯ ПАРАМЕТРА МУТАЦІЇ ГЕНЕТИЧНОГО АЛГОРИТМУ

Текстова частина
магістерської роботи
за спеціальністю «Комп'ютерні науки» 122

Керівник магістерської роботи
к-т фіз.-мат. наук, доцент
Гулаєва Н.М.

_____
(Підпис)
"___" _____ 2023 року

Виконав студент
Кобєлєв М. Д.
"___" _____ 2023 року

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Викладач кафедри інформатики,

к-т фіз.-мат. наук, доцент

_____ Гулаєва Н.М.

„____” _____ 2022р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на магістерську роботу

студенту Кобєлєву Михайлу Дмитровичу

факультету інформатики 2 курсу магістерської програми

ТЕМА: **Налаштування параметра мутації генетичного алгоритму**

Зміст ТЧ до магістерської роботи:
Індивідуальне завдання
Вступ
Розділ 1. Розробка програмного застосунку для проведення експериментів
Розділ 2. Проведення експериментів
Розділ 3. Аналіз результатів експериментів
Висновки.
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2022 р. Керівник_____

(підпис)

Завдання отримав _____

(підпис)

Тема: **Налаштування параметра мутації генетичного алгоритму**

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на магістерську роботу.	1.10.2022	
2.	Огляд літератури за темою роботи.	1.11.2022	
3.	Аналіз вимог до застосунку для проведення експериментів.	1.12.2022	
3.	Імплементація застосунку для проведення експериментів.	15.01.2023	
4.	Проведення експериментів.	19.02.2023	
5.	Об’єдання, очистка та первинний аналіз даних.	26.02.2023	
6.	Проведення додаткових експериментів за потреби.	24.03.2023	
7.	Написання модулів для візуалізації даних.	7.04.2023	
8.	Редагування та доповнення пояснювальної роботи.	8.05.2023	
9.	Оформлення слайдів для доповіді.	1.06.2023	
10.	Захист магістерської роботи	12.06.2023	

Студент _____

Керівник _____

“_____”_____ 2022 р.

Зміст

Анотація

Розроблено програмний застосунок для пошуку максимального значення ймовірності мутації Pmax, за якого відбувається збіжність алгоритму у заданий ліміт кількості ітерацій, та для проведення експериментів з ГА на різних вхідних параметрах. Із використанням застосунку знайдено значення Pmax для низки тестових задач. Проведено додаткові експерименти-прогони ГА для підтвердження або спростування гіпотез щодо поведінки алгоритму за близьких до Pmax значень ймовірності мутації.

Ключові слова:

Генетичний алгоритм, збіжність генетичного алгоритму, параметр мутації, ймовірність мутації, кросинговер.

Вступ

Інтенсивний розвиток областей штучного інтелекту та задач оптимізації вимагає пошуку нових ефективних методів для вирішення складних проблем. У цьому контексті генетичні алгоритми займають вагому позицію завдяки їх здатності працювати із складними функціями.

Ця робота присвячена дослідженню параметра мутації генетичного алгоритму та його впливу на збіжність та на якість знайдених розв’язків. Мутація в генетичних алгоритмах є одним із ключових механізмів, який забезпечує введення випадковості та різноманітності в популяцію. Цей параметр визначає ймовірність зміни генетичної інформації кожної окремої особини - потенційного розв’язку. Однак підібрати правильне значення для цього параметру залишається нетривіальною задачею.

Мета роботи - аналіз впливу параметра мутації на збіжність ГА та якість отриманих ним розв’язків. Для цього були поставлені такі задачі:

1) Розробити застосунок для проведення експериментів та аналізу зібраних даних.
2) Провести серію експериментів з підбору граничного значення мутації Pmax.
3) Провести експерименти, змінюючи значення інших параметрів ГА.
4) Сформулювати та перевірити гіпотези щодо збіжності алгоритму та точності знайдених розв’язків за різних значень параметрів. Використати візуальний та статистичний аналіз для цього.

Наукова методика дослідження передбачає зібрання емпіричних даних з прогонів генетичних алгоритмів із різними значеннями параметрів, аналіз результатів та їх порівняння з метою перевірки гіпотез щодо оптимальних значень таких параметрів. Ми будемо оцінювати

швидкість збіжності алгоритму, ймовірність знаходження оптимального розв'язку та інші характеристики.

Очікується, що результати дослідження нададуть уявлення щодо впливу параметра мутації та інших факторів на збіжність генетичного алгоритму. Це дозволить оптимізувати параметри алгоритмів для різних задач і впроваджувати їх із покращеним результатом.

Актуальність теми:

Актуальність роботи випливає з необхідності подальшого удосконалення генетичних алгоритмів та розуміння їх параметрів для вирішення складних оптимізаційних задач. Генетичні алгоритми знаходять все більше застосувань у різних галузях, проте оптимальний вибір параметрів для конкретної задачі залишається відкритою проблемою. Глибокий аналіз та експериментальне вивчення параметра мутації та його взаємозв'язку з іншими параметрами допоможе встановлювати такі налаштування генетичного алгоритму, що сприятимуть покращенню результатів оптимізації в різних доменах. Такий підхід може відкрити нові перспективи в галузі генетичних алгоритмів та допомогти ефективніше розв’язувати оптимізаційні задачі.

Об’єкт дослідження:

Збіжність генетичного алгоритму.

Предмет дослідження:

Параметр «ймовірність мутації», його вплив на збіжність генетичного алгоритму та точність знайдених розв’язків.

Методи дослідження:

Експериментальний метод: розроблено програму для підбору та тестування значень параметру Pmax. Проведено та проаналізовано додаткові експерименти для підтвердження або спростування гіпотез.

Статистичний аналіз: висунуто гіпотези щодо залежностей між параметрами алгоритму та швидкістю його збіжності. Використано статистичні критерії та підтвердження або спростування цих гіпотез.

Структура роботи:

Робота має три розділи:

1) У першому розділі описано процес розробки застосунку для проведення експериментів та особливості його імплементації.
2) У другому розділі описано проведення експериментів, надані набори вхідних параметрів до експериментів.
3) У третьому розділі аналізуються результати експериментів, перевіряються висунуті гіпотези.

Розділ 1. Розробка програмного застосунку для проведення експериментів

1.1 Алгоритм підбору граничного значення ймовірності мутації Pmax

Визначимо граничне значення параметру мутації - Pmax - як максимальне значення ймовірності мутації, за якого збіжність алгоритму відбувається у 100% прогонів. Тобто збільшивши це значення, на приклад, на 20%, ми втратимо збіжність у деяких прогонах.

Для підбору граничного значення використовуємо наступний алгоритм (також зазначено в курсовій роботі [1]):

```
pm = (початкове значення)
δ = pm * 0.5
ПОВТОРЮВАТИ 15 разів
        Провести 10 прогонів для заданого значення pm
        ЯКЩО спостерігається збіжність в 100% прогонів,
                        ТО збільшити pm += δ
                ІНАКШЕ зменшити pm -= δ
        δ *= 0.5
Pmax = (останнє pm, за якого відбувалась 100% збіжність)
```

Лістинг 1.1 Алгоритм знаходження граничного значення Pmax

Початкове значення, за можливості, ми беремо із вже знайдених за минулих досліджень або вираховуємо його з рівнянь лінійних залежностей наведених в [1].

Знайдене потенційне значення Pmax тестуємо: збіжність має відбуватись при 100% прогонів, але при збільшенні Pmax на 20%, збіжність має бути не в усіх прогонах. Якщо тестування вдале - вважаємо, що підібране значення є Pmax, якщо ні - запускаємо алгоритм з Лістингу 1.1 знову, взявши за початкове значення те, яке ми тестували.

1.2 Структура застосунку

1.2.1 Вимоги до застосунку

Основними вимогами до застосунку були:

1. Запуск експериментів із підбору значень Pmax за алгоритмом, що описано в пункті 1.1, для низки конфігурацій параметрів ГА.
2. Проведення експериментів із можливістю задання довільних значень параметрів та бажаної кількості прогонів.
3. Можливість візуалізації та статистичного аналізу отриманих в ході експериментів даних. Збереження результатів у зручній для аналізу формі.

Згідно висунутих вимог було розроблено консольний застосунок мовою Python, а для збереження конфігурацій та прогонів використовується реляційна база даних PostgresQL.

1.2.2 Архітектура застосунку

Аби забезпечити максимальну гнучкість при конфігурації експериментів, але уникнути дублювання коду, було створено наступну структуру застосунку.

- Основні функції ГА, такі як відбір, мутація, кросинговер, кодування тощо, були винесені в спільний пакет (package) `core`. А кінцевий застосунок складається з окремих пакетів, що імпортують цей пакет.
- Пакети `app_pmax` та `app_free_runs` складаються з таких компонент:
    - Модуль `main` - стартова точка програми.
    - Модуль `models` - визначення класів/таблиць ORM.

- Модуль `runner` - реалізація самого ГА.
- Пакет `helpers` - набір декількох допоміжних модулів, наприклад, із функціями для збереження або логування результатів.
- Пакет `functions` - імплементація тестових функцій.

- Пакет `app_analysis` - є набором Jupyter ноутбуків для статистичного аналізу та візуалізації даних і містить модуль `helpers` із допоміжними функціями для аналізу та роботи із БД.

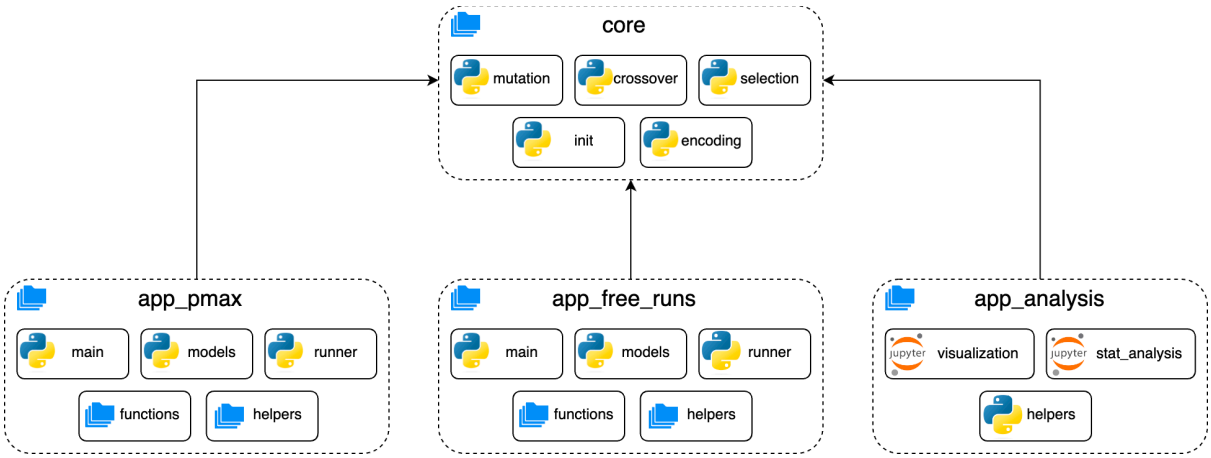


Рисунок 1.1 Діаграма архітектури застосунку

1.2.3 Схема бази даних

Для пакетів `app_pmax` та `app_free_runs` були розроблені дві окремі схеми бази даних, оскільки задачі цих пакетів сильно відрізнялись і мали різні вимоги до збереження даних.

Перша база даних - для пакету `app_pmax` - для збереження інформації про підбір значень Pmax. Було розроблено ER-модель БД та виокремлено такі сутності:

1. **Function** – тестова функція (сферична, Деба тощо).

2. **FuncParam** – параметри функції (інтервал, на якому проводимо експерименти, точність).
3. **FuncCase** – функція із точністю та інтервалом (напр., сферична на інтервалі [-1, 1] із точністю 3 знаки після коми).
4. **ParamSet** – набір параметрів генетичного алгоритму.
5. **ExperimentSuite** – експеримент з певним набором параметрів.
6. **TestSuite** – тестування значення Pmax (окремо для 80% від Pmax, Pmax та 120% від Pmax).
7. **RunSet** – і-ий крок алгоритму підбору Pmax (усього 15 на один експеримент).
8. **Run** – 1 прогін у тестах (відношення до TestSuite) або в алгоритмі пошуку Pmax (відношення до RunSet).
9. Також маємо окрему допоміжну таблицю **InitPopulation**, яка служить як seed для ініціалізації популяції. Докладніше про ініціалізацію йдеться в пункті 2.1.1.2.

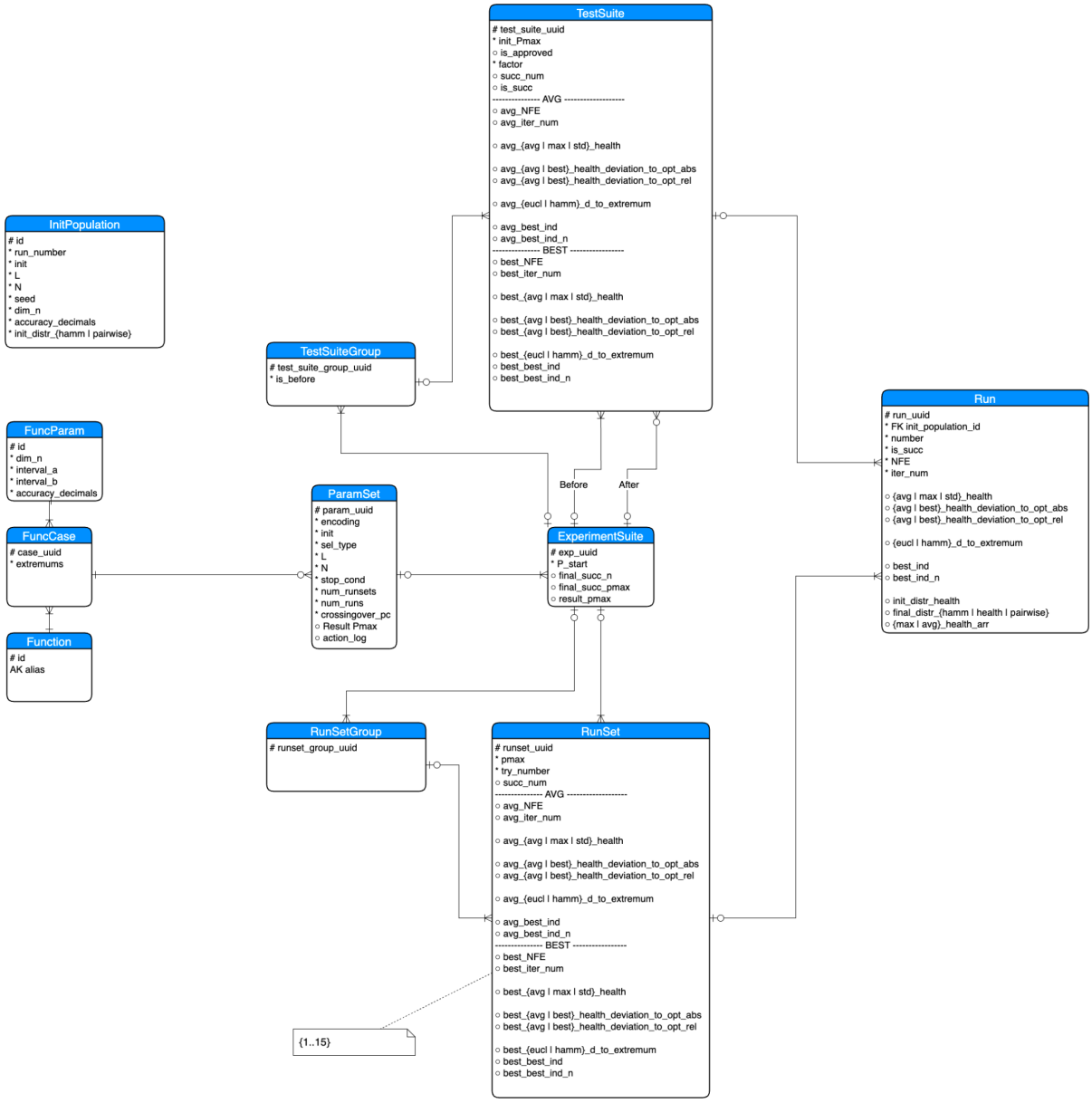


Рисунок 1.2 Діаграма схеми бази даних для пакету app_pmax

Ця база даних вже використовувалась для курсової роботи [1], але зазнала деяких змін, а саме:

1. Було вирішено змінити тип ключі з автоінкремента на UUID. Оскільки під час проведення обчислень з’явилась можливість запускати їх на декількох машинах, для зменшення мережевого навантаження, кожна машина мала свою базу даних. Згодом об’єднання таких баз в одну базу для зручності аналізу викликало складнощі через конфліктуючі ключі. Зміни ключів на UUID вирішила цю проблему.

2. Для зручності аналізу були додані надлишкові сутності - **TestSuiteGroup** - група тестів із 0.8*Pmax, Pmax, 1.2*Pmax та **RunSetGroup** - група сетів прогонів (кроків алгоритму) - від 1 до 15.

Друга база даних - для пакету `app_free_runs` - для збереження експериментів з довільною конфігурацією. Були виокремлені такі сутності:

1. **RunSuite** – допоміжна сутність для умовного розділення експериментів.
2. **RunSpec** – конфігурація алгоритму та бажана кількість прогонів на такій конфігурації.
3. **RunSpecBundle** – набір пов’язаних конфігурацій - для зручності аналізу.
4. **Run** – Один прогін.
5. **RunSampledIteration** – “знімок” популяції на заданій ітерації. Зазвичай зберігаються перші 5-10 ітерацій та фінальна популяція. За необхідності робиться знімок кожні N ітерацій.
6. І так само допоміжна таблиця **InitPopulation**.

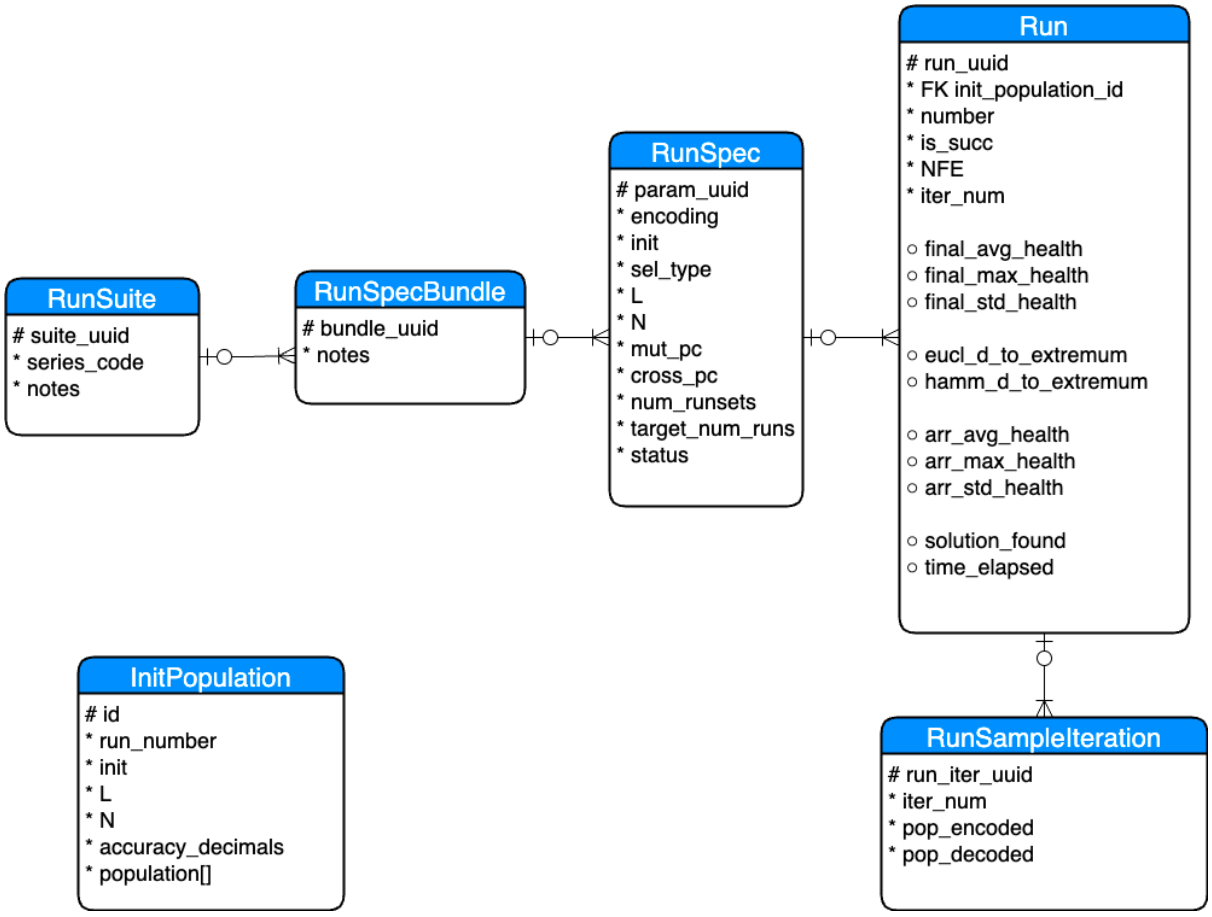


Рисунок 1.3 Діаграма схеми бази даних для пакету app_free_runs

Окремою задачею було пришвидшення роботи з базою, оскільки загальний обсяг фінальних баз був великий - 41 GB та 50 GB відповідно. Більшість місця займали масиви-знімки популяції на заданих ітераціях та масиви показників здоров'я популяції. Оскільки PostgresQL є row-oriented базою даних, то доступ до скалярних характеристик (кількість ітерацій, успішність прогону, знайдений розв'язок тощо) вимагав би від бази даних зчитування всієї реляції. Для вирішення цієї проблеми, зокрема, масиви-знімки популяції були винесені в окрему реляцію RunSampleIteration.

1.2.4 Схема роботи застосунку

Кількість запланованих експериментів та час обчислення унеможливили їх проведення на локальній машині, тому було обрано

асинхронний формат проведення експериментів, тобто присутності дослідника при виконанні обчислень не потребувалось. За такого формату проведення конфігурація експерименту вноситься в базу даних, а обчислення проводяться в тіньовому (background) режимі.

На окремому сервері запускається набір агентів (workers) у залежності від потужностей системи. Оскільки один прогін виконується в одному процесі в Python, то бажано запускати кількість агентів, що не перебільшує кількість ядер CPU, що доступні на машині.

В базі даних зберігається набір конфігурацій із певним статусом - параметрів ГА, початкової популяції, бажана кількість прогонів тощо. Агент вибирає першу доступну конфігурацію та змінює її статус. Аби уникнути проблем з паралельним доступом, коли, наприклад, два агенти беруть в роботу одну конфігурацію, необхідно впевнитися, що рівень ізоляції транзакцій в базі даних SERIALISABLE. Оскільки в роботі використовувалась СУБД PostgresQL, цю зміну необхідно було провести вручну [2]:

```
ALTER DATABASE genalgo
SET DEFAULT_TRANSACTION_ISOLATION TO SERIALIZABLE;
```

Лістинг 1.2 SQL вираз для зміни рівня ізоляції транзакцій в PostgresQL

1.3 Оптимізація швидкодії застосунку

Окрім оптимізації роботи БД, описаних у пункті 1.2.3 була необхідність пришвидшення самого програмного коду, оскільки експерименти можуть займати багато часу. Із цією метою програма активно використовує бібліотеку numpy, написану на C.

Кожна популяція моделюється двовимірним numpy масивом (саме масивом, а не Python списком) з “0” та “1”, а операції, такі як відбір, мутація, функція здоров’я тощо, представляються у вигляді композиції векторизованих функцій із бібліотеки.

Розглянемо приклад імплементації щільнісної мутації за допомогою таких операцій:

1. Генерація масиву випадкових чисел з інтервалу [0,1), що має такий самий розмір, що і популяція.
2. Порівняння - якщо число в локусі менше за P_{mut} - треба виконувати мутацію у цьому локусі.
3. Побітовий XOR - у локусах, відмічених у пункті 2, обертаємо значення “0”->”1” та “1”->”0”.

```
def mutate(population: np.ndarray, px: float) -> np.ndarray:
    places_to_mutate = np.random.rand(*population.shape) < px
    population[places_to_mutate] ^= 1
    return population
```

Лістинг 1.2 Код із імплементацією щільнісної мутації

Таким чином представлені майже всі оператори, використані в ГА. Проте залишались деякі операції, а саме кросинговер та відбір SUS (див. пункти 2.1.1.4 та 2.1.1.3), які не вдалось представити у вигляді векторизованих операцій.

Ці дві операції було вирішено пришвидшити за допомогою функції розширення Python-проекту скомпільованими модулями. Найпопулярнішою мовою для таких розширень є C (бібліотеки numpy, pandas. SciPy та інші) та C++ (TensorFlow, OpenCV). Однак, зараз набуває популярності також мова Rust через безпекові гарантії, що вона надає, та більш розвинену екосистему бібліотек та інструментів. Саме через останню причину Rust було обрано для написання розширення для застосунку.

Правильне налаштування пакетів значно спрощує утиліта maturin [3] та набір байндингів для Python PyO3 [4].

Розглянемо приклад кросинговеру. На вхід у Rust функцію надходять масиви numpy (використано numpy bindings for Rust [5]) із популяцією, індексами пар особин для схрещування та масивом точок схрещування. Ми проходимо вкладеним циклом по кожній парі та in-place замінюємо значення генів. Код для кросинговеру та SUS на Rust доступний для ознайомлення в Додатку А.

Після написання коду встановлюємо пакет-розширення до поточного віртуального середовища (venv) за допомогою maturin та можемо імпортувати модуль у Python.

Порівняємо швидкодію двох імплементацій кросинговеру на машині:

- Процесор: Apple M2 Pro, 12 ядер, 3.5 MHz
- ОП: 32GB

Програмний код у Додатку Б.

	Час на 10000 повторень, сек		
	N=100	N=500	N=1000
Python	0.48	2.4	4.8
Rust	0.15	0.6	1.3

Таблиця 1.1 Результати тесту на швидкодію двох імплементацій кросинговеру

Бачимо, що імплементація на Rust має суттєву перевагу в швидкодії в 3-4 рази. Аналогічну тенденцію має оптимізований SUS (Додаток Б), але різниця ще більша - 9-14 разів:

	Час на 10000 повторень, сек		
	N=100	N=500	N=1000
Python	0.27	1.4	2.7
Rust	0.03	0.1	0.19

Таблиця 1.2 Результати тесту на швидкодію двох імплементацій SUS

Проаналізуємо також загальне пришвидшення всього алгоритму з використанням оптимізованих і не оптимізованих операторів. Для цього зафіксуємо наступний набір параметрів (для детального опису див. пункт 2.1.1):

1. Функція: sigma_100
2. Кількість особин (N): 220
3. Довжина кожної особини (L): 50
4. Відбір: SUS
5. Кросинговер: одноточковий, P=1
6. Мутація: щільнісна, P=Pmax

Проведемо по 100 прогонів на 4 варіантах коду та фіксуватимемо час виконання одного прогону:

1. Simple cross, simple SUS
2. Simple cross, optimised SUS
3. Optimised cross, simple SUS
4. Optimised cross, optimised SUS

Оскільки ми не можемо регулювати, за скільки ітерацій алгоритм збіжиться, зобразимо час виконання в залежності від кількості ітерацій:

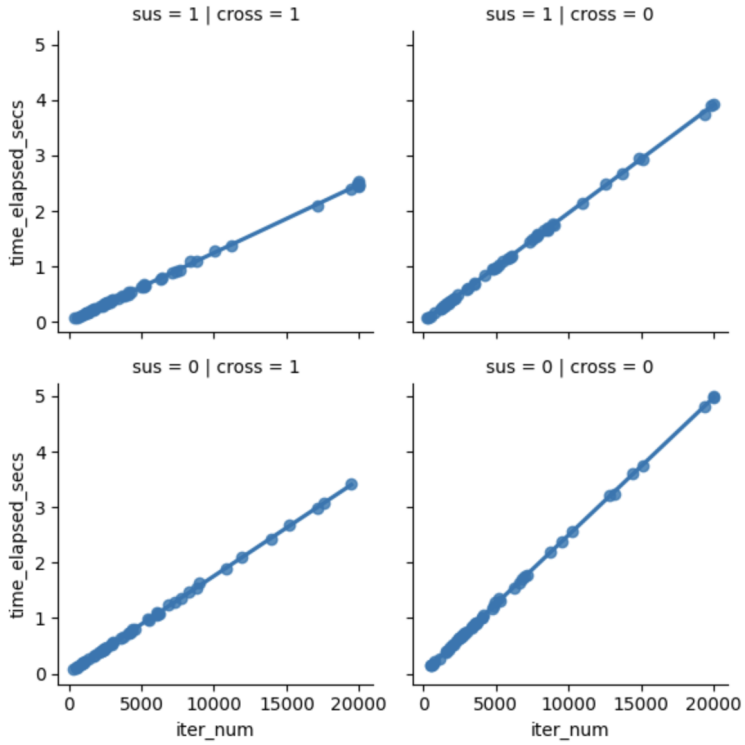


Рисунок 1.4 Графік часу виконання алгоритму (=1 оптимізований метод, =0 неоптимізований метод)

Бачимо, що за максимальної кількості ітерацій (20000) прогін виконується за 3 секунди з оптимізованими відбором і кросинговером і за 5 секунд з неоптимізованим кодом, тобто спостерігаємо загальне пришвидшення в ~1.67 разів.

Розділ 2. Проведення експериментів

2.1 Структура наборів даних

2.1.1 Параметри алгоритму

У роботі розглядається ГА з генераційним типом репродукції та сталим розміром популяції [6]. Зафіксуємо схему роботи алгоритму:

1. Ініціалізація
2. Моделювання еволюційного процесу (репродукція):
   2.1. Оцінювання.
   2.2. Відбір
   2.3. Застосування оператора мутації, кросинговеру.
   2.4. Якщо виконується умова зупинки, перехід на п.3, інакше перехід на п.2.1
3. Завершення роботи

2.1.1.1 Кодування

Функції бінарного ланцюжка не потребують додаткового кодування розв’язків, оскільки функції визначені безпосередньо на бінарних послідовностях. Відсутність кодування позначається як **(none)** на графіках.

Для функцій дійсного аргументу використовується кодування вузлами дискретизації [8], тобто пошуковий простір розділяється на вузли, та кожному вузлу надається номер, який згодом кодується або стандартним бінарним кодом - **(bin)** - або кодом Грея [9] - **(gray)**.

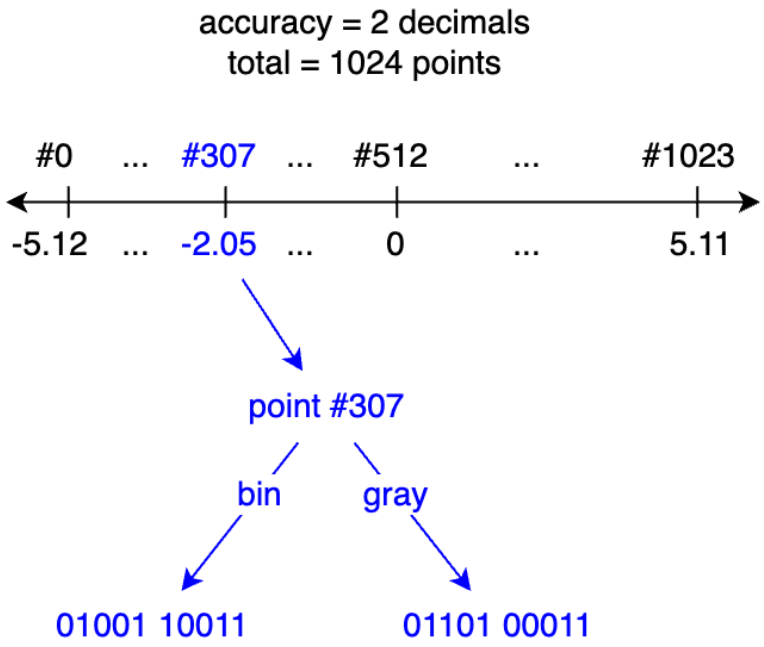


Рисунок 2.1 Приклад кодування для функцій дійсного аргументу вузлами дискретизації

У разі, якщо розмірність функції більша за 1, то всі аргументи після кодування конкатенуються в одну послідовність.

2.1.1.2 Ініціалізація

Для всіх експериментів використовується випадкова ініціалізація популяції. Для кожного гена особини рівно ймовірна ініціалізація з “0” або “1”. При цьому, розподіл відстаней Гемінга у просторі генотипів до оптимальної особини має бути нормальним згідно з центральною граничною теоремою (див. пункт 3.1). При чому, що більший L, то розподіл буде наближатись до нормального.

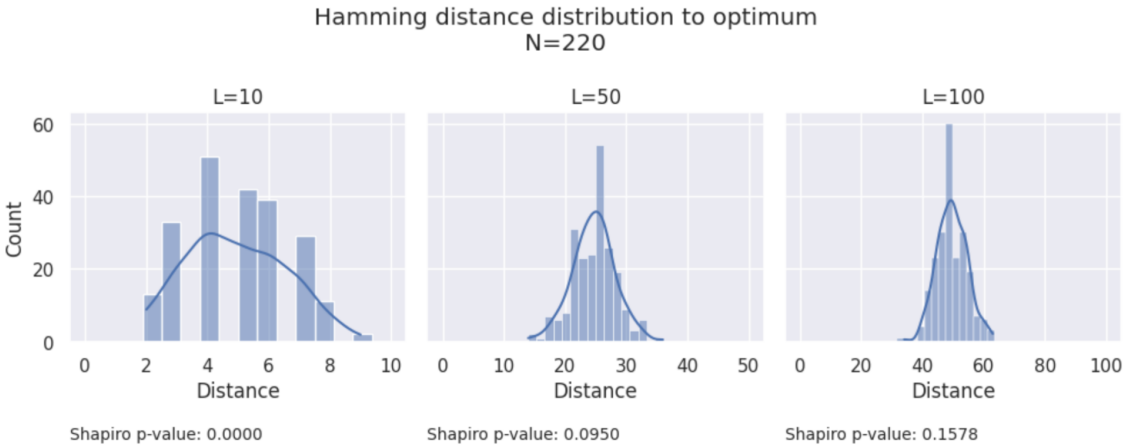


Рисунок 2.2 Графік розподілу відстані Гемінга до оптимуму

Для визначення нормальності розподілу використано критерій Шапіро-Уілка, детальніше цей підхід описано в пункті 3.1.

У той же час, розподіл у просторі фенотипів за такого способу ініціалізації є рівномірним:

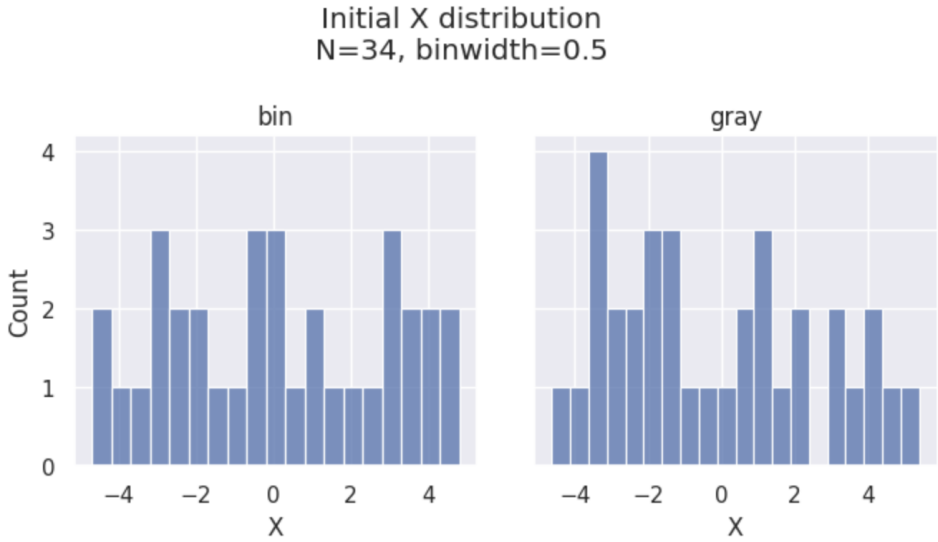


Рисунок 2.3 Графік розподілу початкового фенотипу

Для порівнюваності конкретних прогонів на різних параметрах, початкові популяції для кожного і-того прогону однакові, якщо розмір популяції N, довжина кожної особини L та тип кодування однакові.

2.1.1.3 Відбір

Використовувались такі методи відбору

1. Детермінований турнірний з поверненням [6] із параметром розміру турніру $t \in \{2,5,10,20\}$ - позначка **tournament_{t}**.
2. Рулетка [6] - позначна **rws** (roulette wheel selection).
3. Універсальний стохастичний відбір [10] - позначка **sus** (stochastic universal sampling - SUS).
4. Відбір за рангом [6] на основі SUS із параметром $\beta = 2$ - позначка **ranking**.

2.1.1.4 Генетичні оператори

Використовується щільнісна мутація [8], тобто кожний ген мутує із заданою ймовірністю $P_{mut} \in [0,1]$.

Кросинговер використовується одноточковий [8]. Для більшості прогонів він відсутній, тобто $P_{cross} = 0$, але для експериментів у пунктах 3.4 та 3.7 також використані значення $P_{cross} \in \{0, 0.25, 0.5, 0.75, 1\}$.

2.1.1.5 Умова зупинки

Для всіх експериментів та інших вхідних параметрів зафіксована така умова зупинки:

1. Протягом останніх 10 ітерацій середнє здоров’я популяції змінювалось не більше ніж на 0.0001.
2. АБО проведено 20000 ітерацій.

Якщо справджується лише умова 2, прогін є таким, що не збігся.

2.1.1.6 Успішність прогонів

Оскільки в тестових функціях шукаємо максимум, то знайденим розв'язком будемо вважати особину (одна або декілька), що на останній ітерації алгоритму мала найвище здоров'я.

Маємо два критерії успішності прогонів:

- Знайдений розв'язок є глобальним оптимумом задачі - позначка на графіках **solution_found**.
- Алгоритм збігся у визначений ліміт кількості ітерацій (20000) - позначка на графіках **convergence**.

2.1.2 Набори експериментів

В якості тестових задач в роботі використовуються відомі функції для тестування генетичних алгоритмів, а саме для вирішення задачі знаходження максимуму. Зокрема, функції бінарного ланцюжка:

1. Обернена відстань Гемінга до оптимального ланцюжка - позначка **inverted_hamming_distance**.
2. Функція селективної переваги на біт - позначки **sigma_10**, **sigma_100** - $\sigma \in \{10, 100\}$.

та функції дійсного аргументу:

3. Сферична - позначка **spherical** - на проміжку [-5.12; 5.11], точність 2 знаки після коми, розмірність $dim \in [1,10]$.
4. Зміщена вгору функція Растригіна [7] - позначка **shifted_rastrigins** - на проміжку [-5.12; 5.11], точність 2 знаки після коми, розмірність $dim \in [1,10]$.
5. Перевернутий квадратний корінь - позначка **square_root** - на проміжку [-5.12; 5.11], точність 2 знаки після коми, розмірність $dim \in [1,10]$.

Аналітичне визначення функцій та їхні графіки можна знайти в Додатку В.

Експерименти, проведені в рамках роботи, можна розділити на дві основні категорії:

1. *Підбір значень Pmax* - проведення алгоритму описаного в пункті 1.1 для подальшого аналізу та пошуку залежностей між цим значенням та іншими вхідними параметрами алгоритму. Експерименти проводились етапами. Кожний етап включав у себе певний набір параметрів та поверхнево аналізувався для того, щоб вирішити, які параметри варто запускати далі. Детально опис параметрів кожного етапу можна переглянути в Додатку Г.
2. *Додаткові експерименти* - набори прогонів на довільних параметрах для аналізу властивостей популяції та генетичного алгоритму при граничному значенні мутації Pmax. Експерименти проводились за необхідності в залежності від потреб аналізу та групувались у набори. Детальний перелік параметрів кожного набору разом із кодами експериментів та описами можна переглянути в Додатку Д.

2.2 Запуск експериментів

Як зазначалось у пункті 1.2 експерименти проводяться в асинхронному форматі. Тобто вхідні дані для експериментів задаються окремо від безпосередніх обчислень.

2.2.1 Експерименти з підбором значень Pmax

Першим кроком цього експерименту є створення конфігурацій, а саме:

1. Вносимо в базу даних вручну інформацію про функції (Function, FuncParam, FuncCase).
2. Генеруємо початкові популяції (InitPopulation).
3. Виходячи із наборів параметрів, описаних у пункті 2.1.1, генеруємо ParamSet.

Далі запускаємо ряд агентів (workers) у залежності від потужності машини. Кожний worker бере в роботу конфігурацію та проводить експеримент.

2.2.2 Запуск довільних експериментів

Процес дещо схожий на той, що описаний у пункті 2.2.1

1. Вносимо інформацію про серію експериментів, що ми хочемо провести - RunSuite.
2. Генеруємо конфігурації прогонів - RunSpecBundle, RunSpec.

Далі запускаємо незалежні агенти (workers). Кожний агент (worker) бере в роботу конфігурацію та проводить задану кількість прогонів із нею - Run.

Результатами обчислень є збережені в базу даних прогони та характеристики популяцій, такі як кількість ітерацій, знайдений розв’язок, зміна середнього здоров’я із кожною ітерацією тощо. Ці дані вивантажуються за допомогою SQL-запитів у Jupyter-notebooks для подальшого аналізу та візуалізації.

2.3 Вивантаження результатів

Усі результати для зручності аналізу зберігаються на одному сервері СКБД. Початково планувалось, що згенеровані бази даних будуть вивантажені в dump файлі та об'єднані на локальній машині. Але при прогоні експериментів стало зрозуміло, що через об'єм даних цей підхід займав би занадто багато часу. Тому було вирішено зберігати та об'єднувати всі дані в СУБД на сервері.

Вивантаження файлів здійснювалось за допомогою утиліти pg_dump [11] наступним чином:

```
pg_dump --file=out.dump --format=c --compress=7 --dbname=genalgo
```

Аналогічно утилітою pg_restore [12] дані можна було завантажити на інший сервер СУБД:

```
pg_restore --dbname=genalgo_restored out.dump
```

Проте залишалась задача об'єднати дані в єдину базу для аналізу. Для цього потребувались додаткові зміни бази описані в пункті 1.2.3, а саме використання UUID для первинних ключів. Після цього об'єднання було можливе за допомогою розширення PostgresQL dblink [13]:

```
INSERT INTO experimentssuite(seq_num, created, p_start, final_succ_n, final_succ_pmax, ...)
SELECT seq_num,
       created,
       p_start,
       final_succ_n,
       final_succ_pmax,
       ...
FROM DBLINK('dbname=genalgo_ported', 'SELECT * FROM experimentssuite') AS t(
    seq_num INTEGER,
    created TIMESTAMP,
    p_start DOUBLE PRECISION,
    final_succ_n INTEGER,
    final_succ_pmax DOUBLE PRECISION,
    ...)
```

Лістинг 2.1 SQL код для об'єднання баз даних

Після об’єднання база даних мала такий розмір:

- БД пошуку Pmax: 41 GB
- БД довільних експериментів: 50 GB

Оскільки експерименти займають багато часу та можуть бути дорогими в плані обчислювальних ресурсів, необхідно було потурбуватись про надійне збереження згенерованих даних. З цією метою проводились регулярні бекапи бази даних, які в подальшому зберігались в об’єктному сховищі даних AWS S3 [14]. Проте за замовченням S3 виставляє високу ціну за зберігання даних. На момент написання роботи вона складала $0.023 per GB:

Рисунок 2.5 Ціни на хмарному сховищі AWS S3

За цю ціну S3 надає швидкий доступ та високу доступність збережених даних. Бекапи не потребують швидкого доступу та необхідні лише у рідких випадках збоїв. Для цього випадку використання рекомендовано використовувати сервіс AWS Glacier, ціна якого складає $0.0036 per GB, що в ~6.6 разів нище:

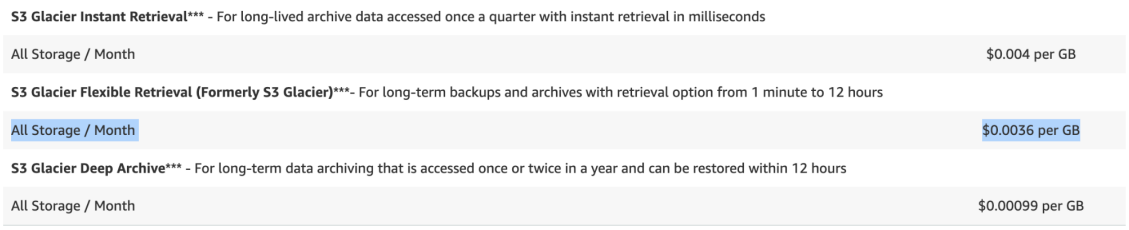


Рисунок 2.6 Ціни на хмарному сховищі AWS Glacier

Підсумовуючи, маємо наступний процес вивантаження даних:

1. Всі окремі бази переносяться у вигляді фалів на сервер із СУБД (pg_dump -> pg_restore).
2. Окремі бази об’єднуються в одну (db_link).

3. Робимо регулярні бекапи об’єднаної бази (pg_dump) та завантажуємо на AWS S3/AWS Glacier (aws cli)

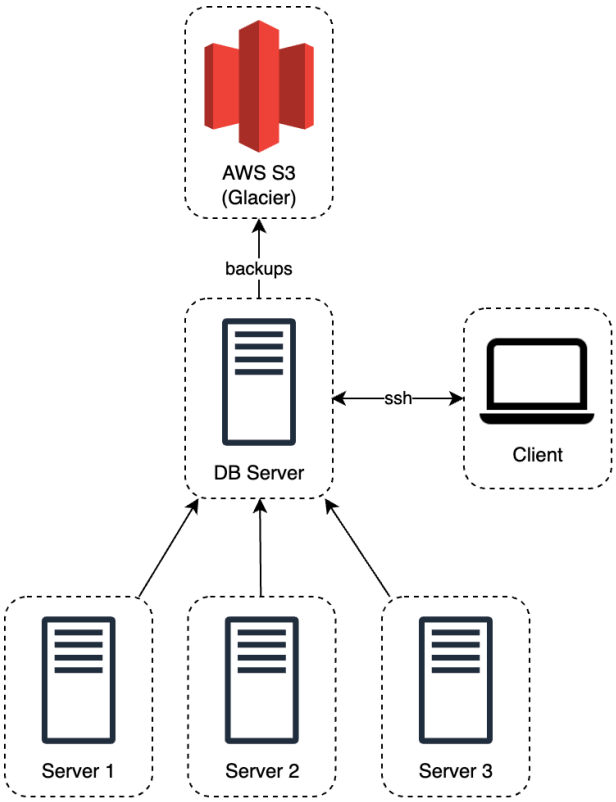


Рисунок 2.7 Схема вивантаження даних

2.4 Модулі для проведення аналізу

Для проведення аналізу даних було використано Jupyter lab [15]. Окремий Jupyter сервер було розгорнуто на хості із базами даних. Комунікація з Jupyter сервером відбувалась за допомогою HTTPS через веб-клієнт Jupyter lab.

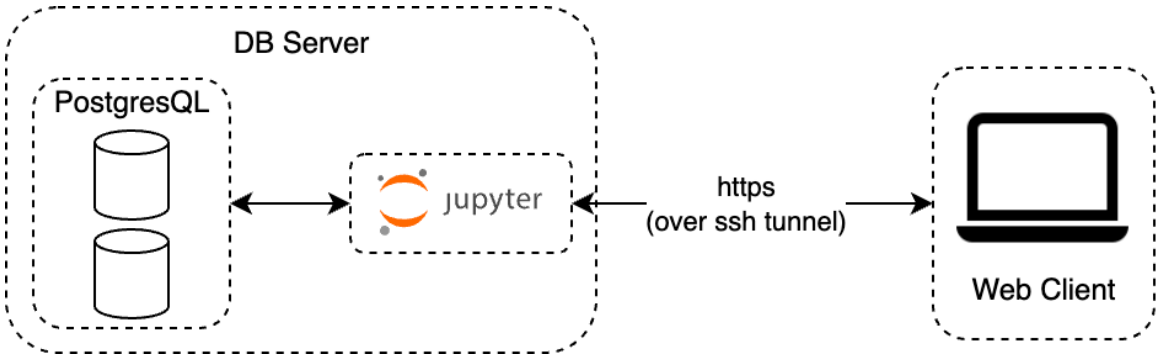


Рисунок 2.8 Схема роботи із Jupyter

Для підготовки даних до аналізу використовувались такі бібліотеки:

1. numpy [16] - основна бібліотека для роботи з числовими масивами даних. Ця бібліотека є основою для багатьох інших бібліотек і напряму майже не використовувалась при аналізі.
2. pandas [17] - робота з даними в табличному вигляді. Вивантаження даних із БД.
3. matplotlib [18] - базова бібліотека для побудови графіків.
4. seaborn [19] - надбудова над matplotlib, бібліотека надає зручний API для візуалізацій.
5. scipy [20] - статистичний аналіз.

Всі вихідні Jupyter notebooks із SQL-запитами, із кодом обробки даних та побудови графіків можна знайти в Додатку Е.

Розділ 3. Аналіз результатів експериментів

3.1 Визначення методів статистичного аналізу

Найбільше увагу в роботі приділено дослідженню впливу параметрів генетичного алгоритму на швидкість його збіжності, яка представлена кількістю ітерацій до збіжності популяції. Проаналізуємо розподіл кількості ітерацій до збіжності для того, щоб визначити які критерії та методи можна використовувати при статистичному аналізі гіпотез.

3.1.1 Визначення розподілу кількості ітерацій

Проведемо додаткові експерименти на функціях sigma_100, spherical та shifted_rastrigins - по 100 прогонів на фіксованому наборі параметрів. Всі параметри кожного експерименту перелічені в таблиці в Додатку Д. На початку кожного підрозділу та на кожному згенерованому графіку зазначається код експерименту (suite), за яким в таблиці можна знайти значення параметрів, із якими експеримент був запущений.

Код цього експерименту в Додатку Д - **custom_ni_n_fixed**.

Для прикладу, побудуємо гістограми кількості ітерацій на розмірності L=10 (для функцій дійсного аргументу dim=1) та N=100.

1. Функції sigma_100

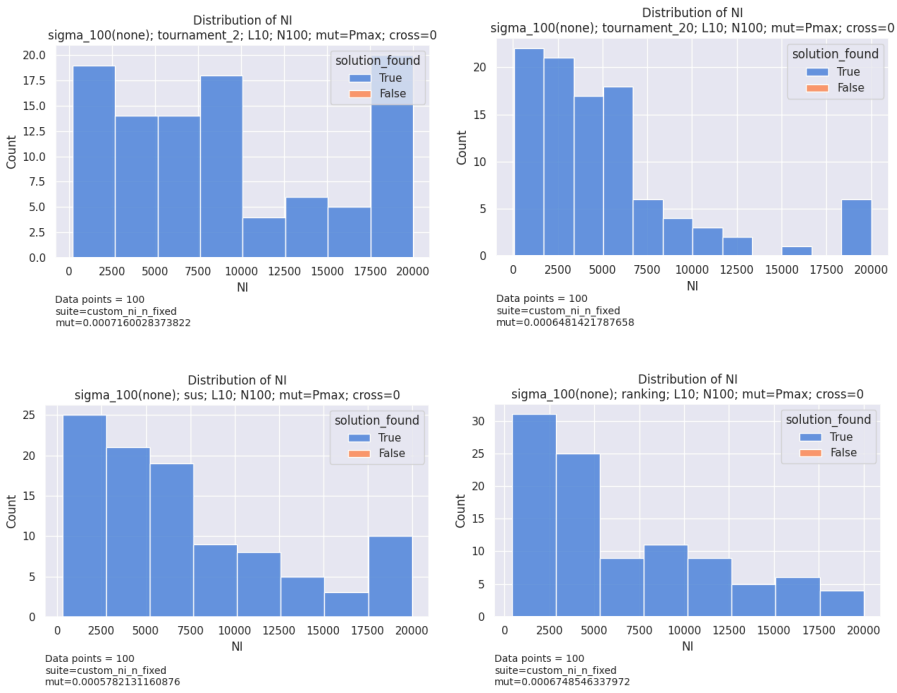


Рисунок 3.1 Гістограми кількості ітерацій для функції sigma_100

2. Функції spherical

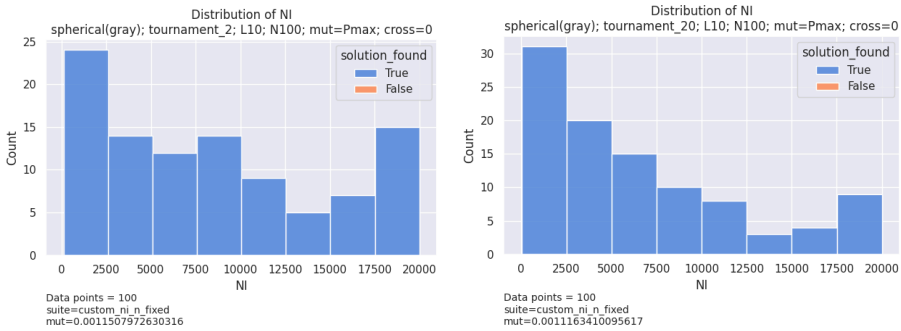


Рисунок 3.2 Гістограми кількості ітерацій для функції spherical

3. Функції shifted_rastrigins

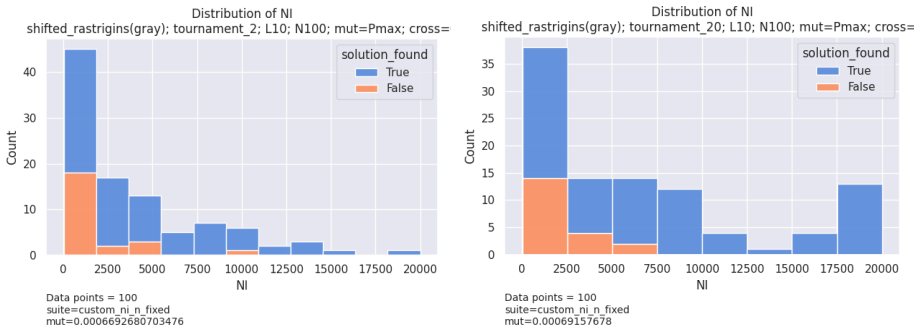


Рисунок 3.3 Гістограми кількості ітерацій для функції shifted_rastrigins

Бачимо з графіків, що розподіл не нагадує жоден із відомих ймовірнісних розподілів. Маємо аналогічну ситуацію для всіх інших комбінацій значень L та N. Отже, використовувати параметричні статистичні критерії на пряму не можна.

3.1.2 Вибірковий розподіл вибіркового середнього кількості ітерацій

Можемо використати інший підхід. Згідно з Центральною граничною теоремою [21], розподіл середніх значень випадкової вибірки (вибіркових середніх) достатньої кількості спостережень збігається до нормального, навіть якщо початкова сукупність не є нормально розподіленою. Вважатимемо, що проведена кількість прогонів на зафіксованому наборі параметрів (на приклад, 10, 50 або 100 прогонів) є випадковою вибіркою усієї нескінченної сукупності можливих прогонів на цьому наборі параметрів. Тоді згідно теоремі, середнє арифметичне кількості ітерацій всіх прогонів на цьому наборі параметрів є випадковою величиною, розподіл якої збігається до нормального. Покладемо, що розмір вибірки n=100 є достатнім для аналізу в цій роботі.

Щоб впевнитись у цьому, зобразимо графічно розподіл середніх значень вибірок розміром 100 на вхідних параметрах, описаних в пункті 3.1.1. Також проведемо статистичний тест на нормальність Шапіро-Уілка [22]. Нагадаємо, що цей статистичний тест має нульову гіпотезу:

$$H_0\text{: "Вибірка є нормально розподіленою"}$$

Також покладемо α-рівень 95%, тобто при p-value меншим за 0.05 H_0 відкидається, і вважаємо, що вибірка не є нормально розподіленою, а якщо більше, то нульову гіпотезу не можемо відкинути, і вважаємо зворотнє.

Спочатку спробуємо проаналізувати всі вибірки розміром 100 на всіх наборах параметрів:

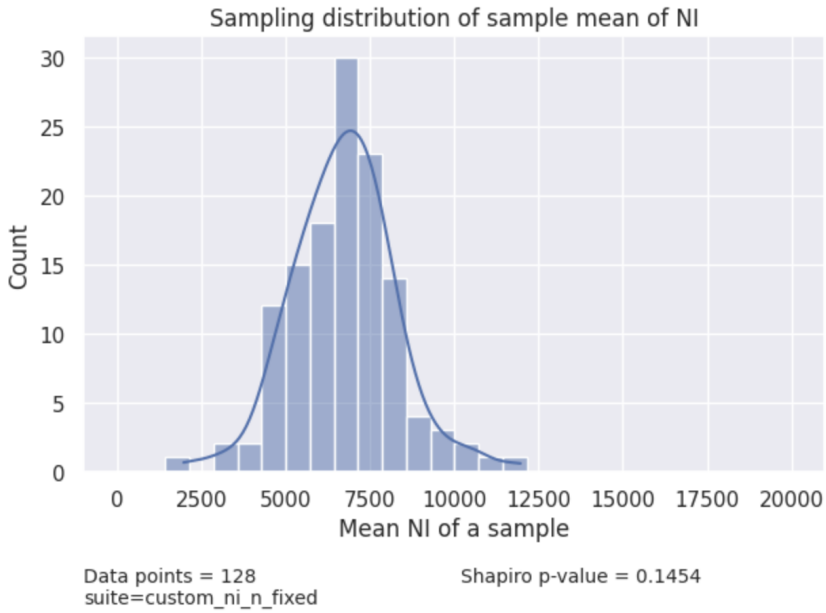


Рисунок 3.4 Графік вибіркового розподілу вибіркового середнього

Графічно бачимо, що розподіл схожий на нормальний. За результатом статистичного тесту (pvalue=0.1454) також можемо припускати для подальшого аналізу, що розподіл є нормальним.

Аналогічні результати маємо також, якщо розділимо вибірку за функціями пристосованості.

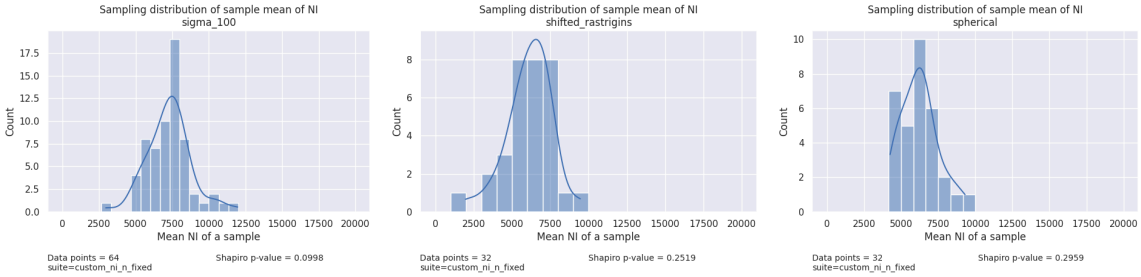


Рисунок 3.5 Графіки вибіркового розподілу вибіркового середнього за функціями пристосованості

Графіки разом із p-value можна знайти в Додатку Ж.

На жаль, подальше розбиття вибірки - наприклад, додатково за типом відбору та кодування - неможливе через невелику кількість даних. Проте в роботі будемо вважати, що теза центральної граничної теореми справджується для будь-якого набору вхідних параметрів генетичного алгоритму із кількість прогонів на такому наборі не менше 100.

3.1.3 Статистичні критерії

Виходячи із пунктів 3.1.1 та 3.1.2 можемо використовувати два підходи для статистичного тестування гіпотез, пов’язаних із впливом вхідних параметрів на швидкість збіжності:

1. Непараметричні критерії для аналізу значень окремих прогонів або маленьких вибірок таких прогонів:
    - Критерій знакових рангів Уілкоксона [21, с. 342]
    - U-критерій Манна-Уітні [21, с. 373]
2. Параметричні критерії, лише для усереднених значень за 100 прогонами:
    - t-критерій Стьюдента [21, с. 310]
    - Дисперсний аналіз (ANOVA) [21, с. 513]
    - t-критерій Уелча, якщо не відомо, чи дисперсії вибірок рівні [21, с. 365]

Надалі, використовуватимемо t-критерій Уелча як основний спосіб статистичної перевірки гіпотез (якщо буде дозволяти розмір вибірки). Також додатково будемо вказувати критерій Уілкоксона для повноти аналізу.

Зафіксуємо α-рівень 0.05.

3.2 Вплив рівня ймовірності мутації на збіжність

Гіпотеза 1.1: «На значеннях ймовірності мутації, близьких до Pmax, збіжність алгоритму є швидшою, ніж на значеннях більше або менше за Pmax».

Гіпотеза 1.2: «На значеннях ймовірності мутації, близьких до Pmax, відсоток успішних прогонів є вищим, ніж на значеннях більше або менше за Pmax».

При прогоні алгоритму підбору Pmax (див. пункт 1.1) було помітно, що прогони тестування на 80% від Pmax виконувались значно швидше ніж прогони із значенням ймовірності Pmax або на 20% вище за Pmax.

Це дещо суперечить теорії Рехенберга про “еволюційне вікно” [23], де зазначається, що при поступовому збільшенні мутації S прогрес алгоритму Φ спочатку збільшується, а потім падає. Автор визначає цей пік як “вікно еволюції”.

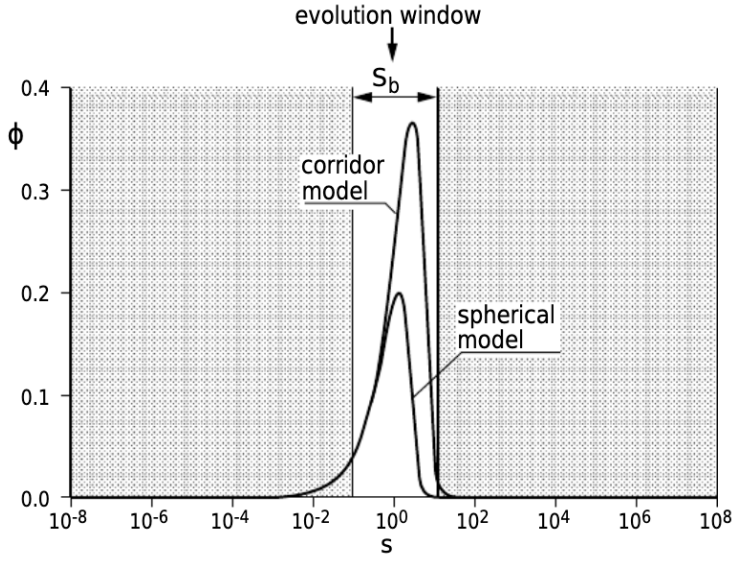


Рисунок 3.6 Еволюційне вікно[23]

Перевіримо це спостереження для нашого випадку. Для цього проведемо додаткові експерименти із значенням

$P_m = \frac{i*Pmax}{100},\ i \in \{1, 10, 20, \ldots, 90, 100, 110, 120\}$. Позначка експерименту в додатку Д - **evol_window**.

Побудуємо розподіл кількості ітерацій за допомогою діаграми розмаху. Також додано точність - частку прогонів, в яких був знайдений правильний розв'язок, та збіжність - частку прогонів, що збіглись. Всі побудовані графіки можна знайти в Додатку Ж.

Для функції бінарного ланцюжка sigma_100 для обох методів відбору SUS та tournament_10 маємо схожі графіки точності - для більших L необхідна більша ймовірність мутації для кращої точності

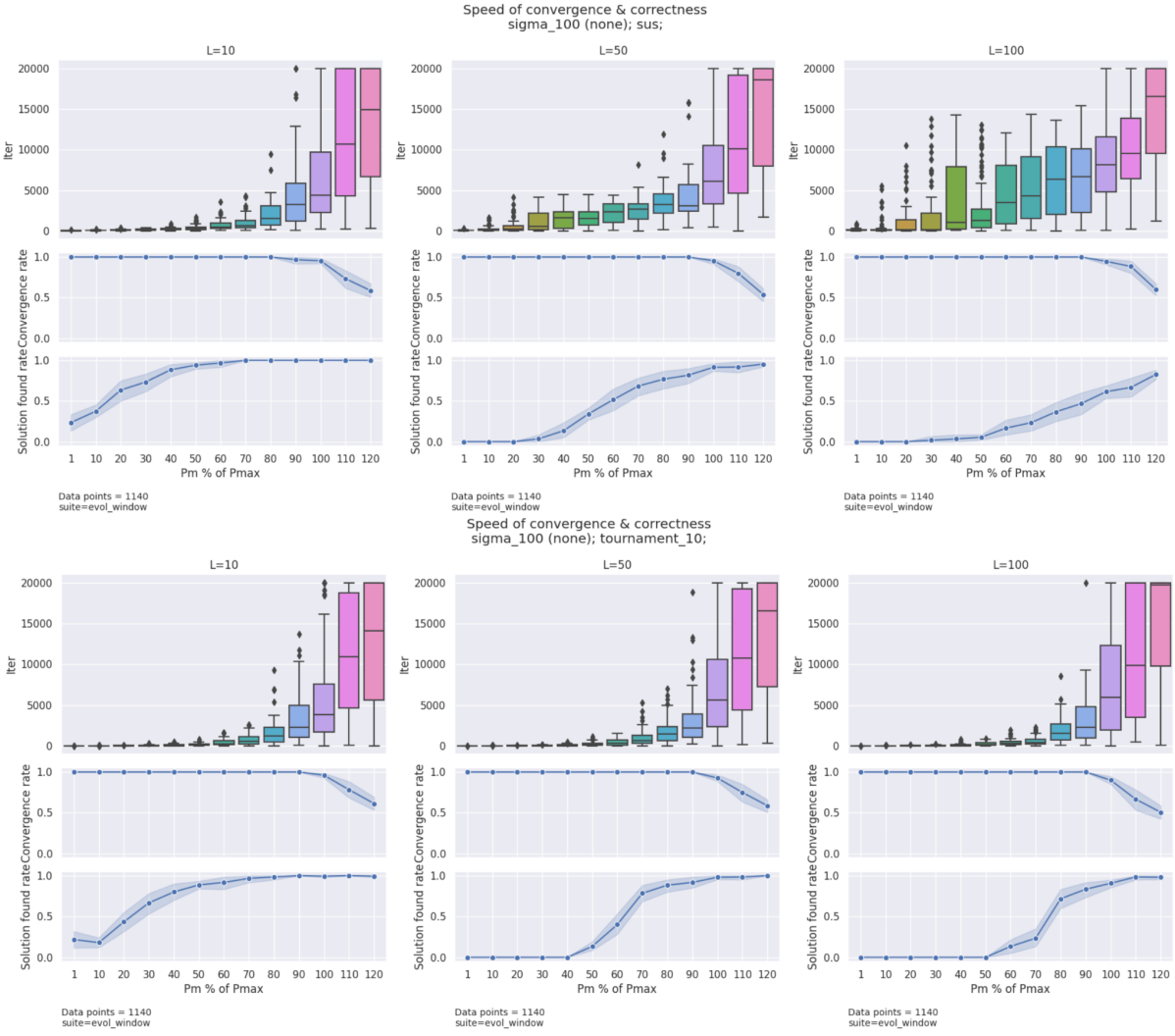
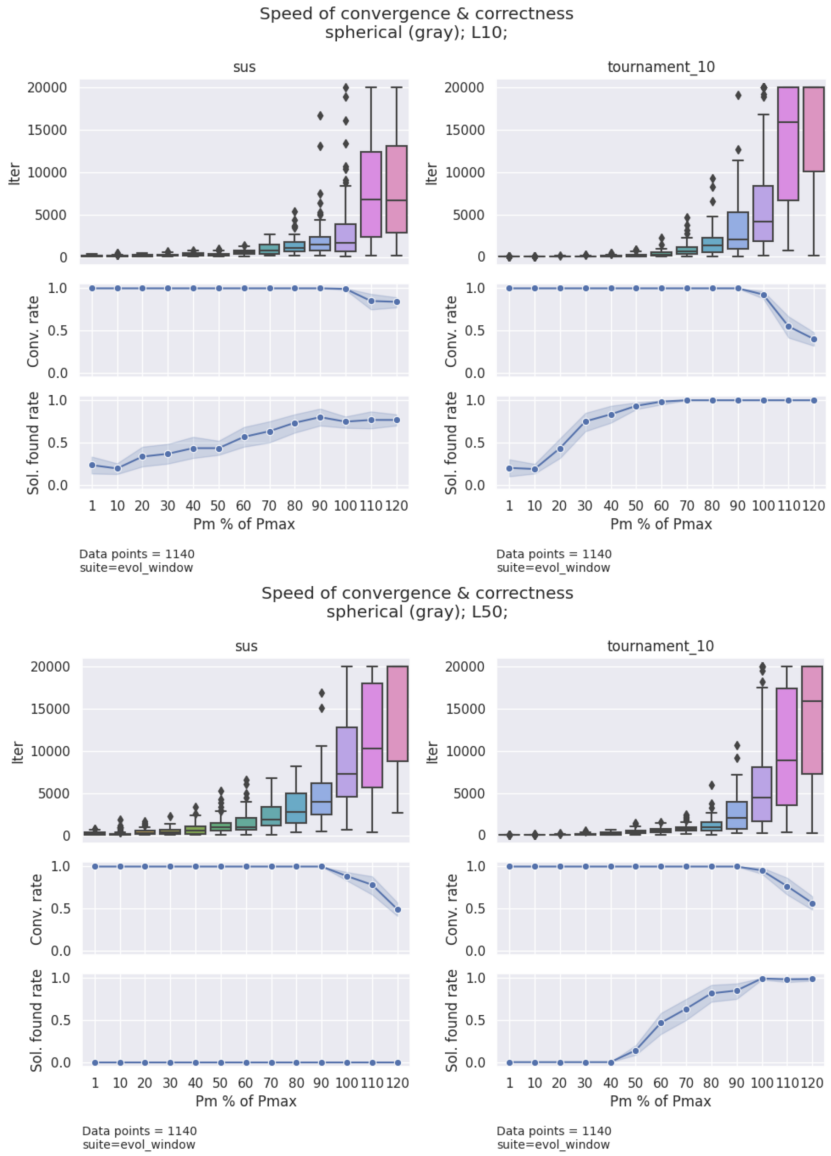


Рисунок 3.7 Графіки швидкості та точності збіжності для функції sigma_100

З графіків видно, що точність 100% досягається при менших значеннях P_m для менших L, а для L=100 такої точності взагалі не

досягається для SUS. При цьому також бачимо, що починаючи з $i = 100$ втрачаємо стовідсоткову збіжність.

Для сферичної функції - spherical - різниця між методами відбору є суттєва:



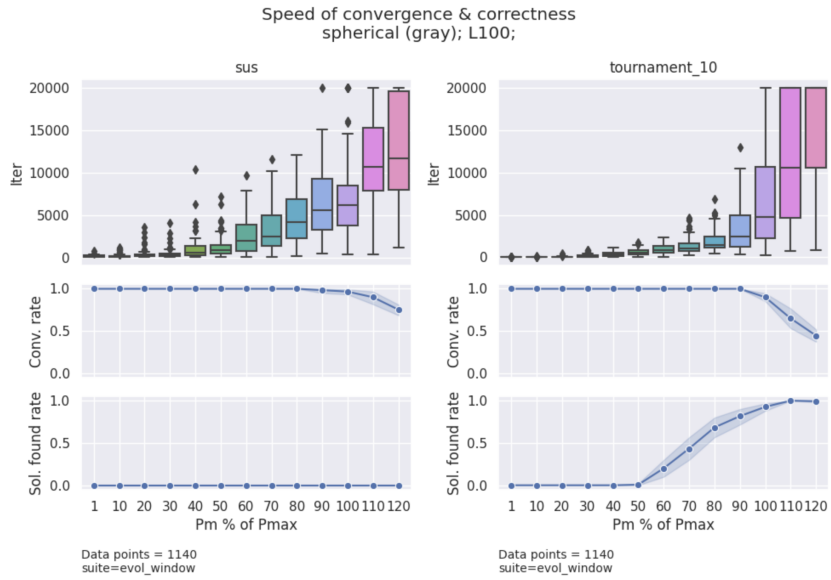
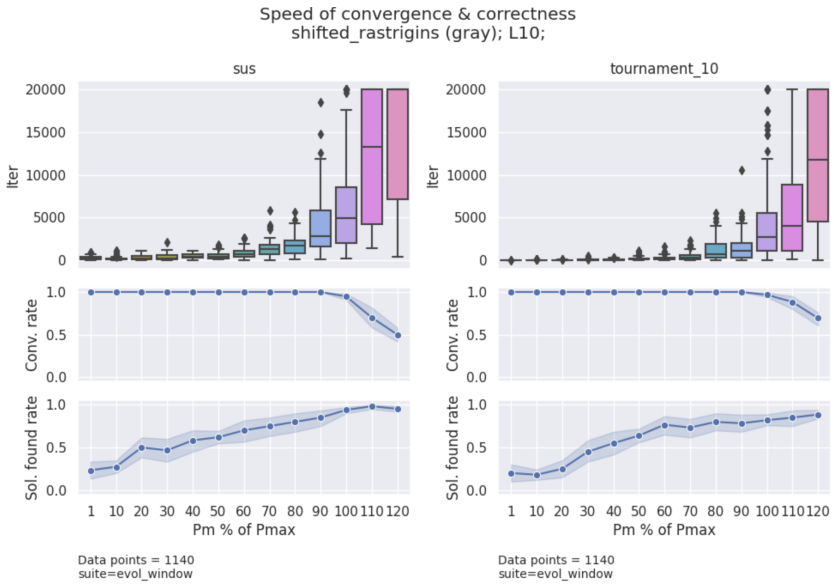


Рисунок 3.8 Графіки швидкості та точності збіжності для функції spherical для L=10, 50 та 100 (зверху вниз)

SUS є більш “консервативним” в плані збереження різноманітності популяції, тому можна припустити, що більш пристосованим особинам не вистачає часу розмножитись та дати ще більш пристосованих нащадків. Турнірний відбір навпаки - завжди обирає найсильніших.

Багатовимірна функція Растригіна (dim=5 та dim=10) - shifted_rastrigins - є занадто складною і для турнірного відбору також:



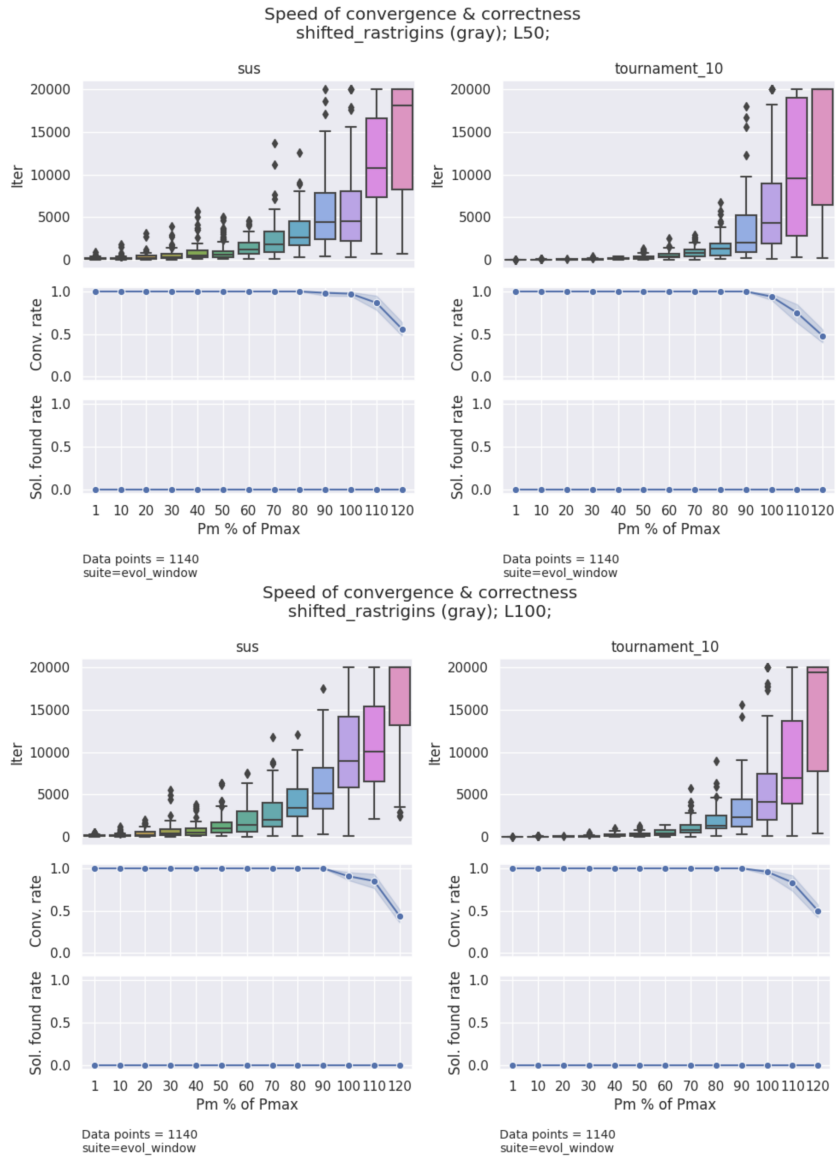


Рисунок 3.9 Графіки швидкості та точності збіжності для функції shifted_restrigins для L=10, 50 та 100 (зверху вниз)

Бачимо, що у жодному прогоні не було знайдено правильного розв'язку для L=50 (dim=5) та L=100 (dim=10). Так само частка прогонів, що збіглись, спадає, починаючи з $P_m = 100\%\ Pmax$.

Для всіх тестових функцій і методів відбору дійсно спостерігаємо спільну тенденцію, що середня кількість ітерацій сильно зростає, особливо починаючи з $P_m = 0.6Pmax$.

Проведемо статистичний аналіз.

A				B				diff B/A		pvalue	
Pm, %	**count**	**mean**	**std**	**Pm, %**	**count**	**mean**	**std**	**d_Pm**	**d_mean**	**Welch**	**Wilcoxon**
sigma_100, sus, L=50, N=100											
100	100	6609.1	4787.2	**120**	100	14142.9	6514.3	1.2	2.14	**3.90E-17**	3.50E-12
50	100	1454.5	829.9	**100**	100	6609.1	4787.2	2	4.54	**2.60E-18**	1.20E-16
50	100	1454.5	829.9	**120**	100	14142.9	6514.3	2.4	9.72	**6.70E-36**	4.30E-18
10	100	193.4	173.8	**50**	100	1454.5	829.9	5	7.52	**7.30E-28**	3.10E-17
10	100	193.4	173.8	**100**	100	6609.1	4787.2	10	34.17	**5.60E-24**	3.90E-18
10	100	193.4	173.8	**120**	100	14142.9	6514.3	12	73.13	**5.90E-39**	3.90E-18
sigma_100, tournament_10, L=50, N=100											
100	100	7131.3	5989.8	**120**	100	13177.2	7220.3	1.2	1.85	**9.20E-10**	2.40E-07
50	100	234.4	221.5	**100**	100	7131.3	5989.8	2	30.42	**5.70E-20**	5.00E-18
50	100	234.4	221.5	**120**	100	13177.2	7220.3	2.4	56.21	**7.30E-33**	3.90E-18
10	100	21.6	11.5	**50**	100	234.4	221.5	5	10.84	**8.00E-16**	1.50E-17
10	100	21.6	11.5	**100**	100	7131.3	5989.8	10	329.7	**9.70E-21**	3.90E-18
10	100	21.6	11.5	**120**	100	13177.2	7220.3	12	609.21	**2.20E-33**	3.60E-18

Таблиця 3.1 Результат статистичного аналізу різниці кількості ітерацій

Пояснення до колонок:

1. **Pm, %** - значення ймовірності мутації як % від Pmax.
2. **d_Pm** - $d_{Pm} = \frac{Pm_A}{Pm_B}$.
3. **d_mean** - $d_{mean} = \frac{mean_A}{mean_B}$. Зеленим виділені значення більші за 1.
4. **p-value** - зеленим виділені Welch pvalue нижчі за α-рівень (0.05), що позначають статистично значиму відмінність.

Бачимо, що різниця ітерацій до збіжності для різних пар значень мутації є статистично значимою для функції sigma_100, L=50 для обох методів відбору. Аналогічними є результати і для інших L, функцій та методів відбору.

Таким чином, Гіпотеза 1.1 не підтвердилась, оскільки швидкість алгоритму *сповільнюється* із збільшенням ймовірності мутації. Беручи до уваги два критерії успішності - збіжність та точність - Гіпотеза 1.2 підтвердилась: незважаючи на те, що точність прогонів зростає із збільшенням ймовірності мутації, ми втрачаємо збіжність на значеннях $\mathrm{P}_{mut} > Pmax$.

Отже, можемо зробити висновок, що збільшення значення параметру мутації збільшує кількість ітерацій до збіжності. Проте також збільшується і ймовірність знайти точний розв’язок до задачі. Значення Pmax виступає вигідним компромісом, коли ми можемо розраховувати як на високу точність, так і на мінімальну кількість ітерацій, необхідну для цієї точності.

3.3 Вплив розміру популяції на збіжність за Pmax

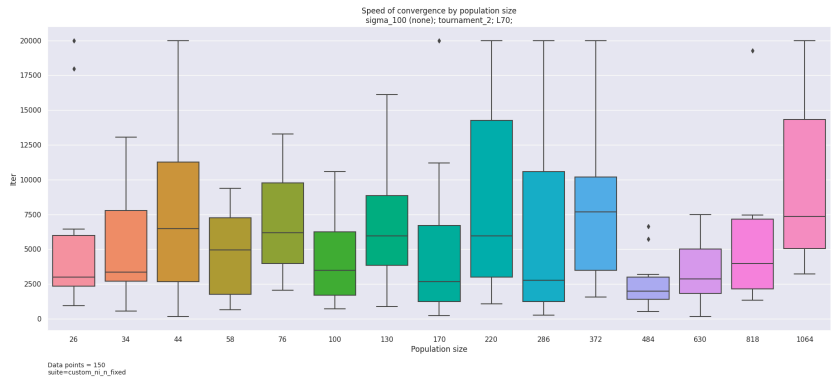
Гіпотеза 2: «За значення ймовірності мутації близькому до Pmax, збіжність алгоритму сповільнюється із збільшенням розміру популяції».

3.3.1 Експерименти з підібраним Pmax

Для початку проведемо експерименти з $L \in \{10,30,70,100\}$ та розміром популяції від 26 до 1064, де кожне наступне значення на ~30% більше за попереднє. Для кожного набору параметрів {функція, відбір, L, N} ймовірність мутації своя, при чому $P_m = Pmax_{f,sel,L,N}$. Позначка експерименту в Додатку Д - **custom_ni_n_fixed**.

Побудуємо діаграми з розмахом (коробкові діаграми) [19] для різних L, де на осі Ox маємо розмір популяції, а на осі Oy кількість ітерацій до збіжності.

Для функції sigma_100 для відборів tournament_2, tournament_20 та ranking очевидних залежностей не видно:



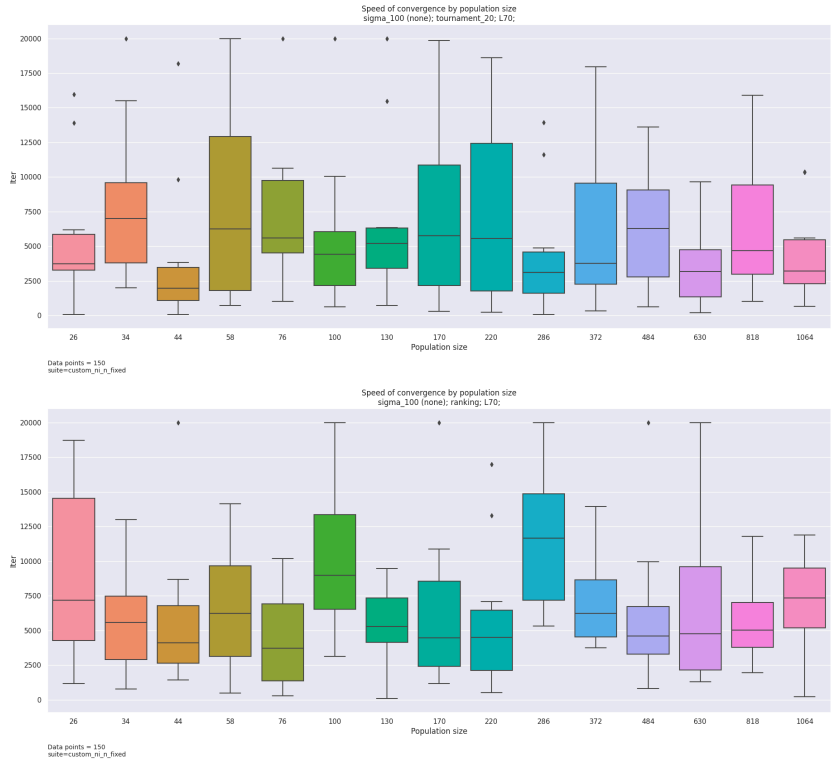


Рисунок 3.10 Графіки швидкості збіжності за розміром популяції для функції sigma_100 та різних типів відбору

Для інших значень L (10, 30 та 100) ситуація така сама. Для відбору SUS спостерігається невелика залежність - із зростанням N зростає кількість ітерацій:

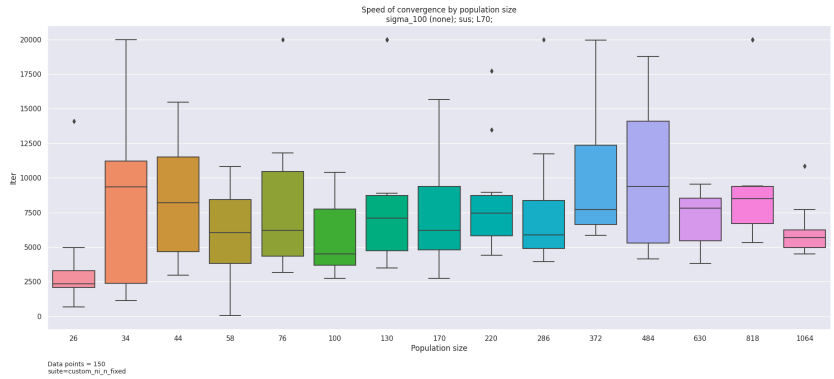


Рисунок 3.11 Графік швидкості збіжності за розміром популяції для функції sigma_100, L=70 та SUS

Ще більше це помітно для L=100:

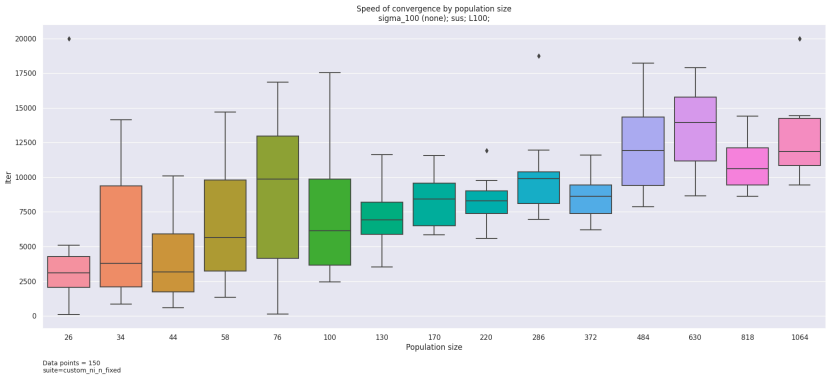


Рисунок 3.12 Графік швидкості збіжності за розміром популяції для функції sigma_100, L=100 та SUS

Для функцій дійсного аргументу spherical та shifted_rastrigins маємо лише прогони із турнірним відбором. Так само як і для sigma_100 на цих відборах залежностей не спостерігається для будь-яких значень L. Усі згенеровані графіки можна знайти в Додатку Ж.

Проведемо статистичний аналіз різниці кількості ітерацій для розмірностей 100, 220 та 484 попарно, при цьому:

H_0: Кількість ітерацій для N_a та N_b однакова.

H_A: Кількість ітерацій різна.

A				B				diff B/A		pvalue	
N	**count**	**mean**	**std**	**N**	**count**	**mean**	**std**	**d_N**	**d_mean**	**Welch**	**Wilcoxon**
sigma_100, tournament_2, L=70											
58	100	5707.5	4720.4	**100**	100	8405.0	6427.8	1.72	1.47	**0.0009**	0.001
58	100	5707.5	4720.4	**220**	100	7598.3	6538.8	3.79	1.33	**0.0201**	0.0716
58	100	5707.5	4720.4	**484**	100	4715.6	4280.1	8.34	0.83	**0.1212**	0.2222
100	100	8405.0	6427.8	**220**	100	7598.3	6538.8	2.2	0.9	**0.38**	0.3504
100	100	8405.0	6427.8	**484**	100	4715.6	4280.1	4.84	0.56	**3.8E-06**	5.1E-06
220	100	7598.3	6538.8	**484**	100	4715.6	4280.1	2.2	0.62	**0.0003**	0.0006
sigma_100, tournament_20, L=70											
58	100	5991.3	5735.6	**100**	100	8388.8	6894.6	1.72	1.4	**0.0082**	0.0053
58	100	5991.3	5735.6	**220**	100	7967.7	6551.7	3.79	1.33	**0.0243**	0.0239
58	100	5991.3	5735.6	**484**	100	7612.1	6234.5	8.34	1.27	**0.0572**	0.0212

100	100	8388.8	6894.6	**220**	100	7967.7	6551.7	2.2	0.95	**0.6584**	0.6303
100	100	8388.8	6894.6	**484**	100	7612.1	6234.5	4.84	0.91	**0.4044**	0.3944
220	100	7967.7	6551.7	**484**	100	7612.1	6234.5	2.2	0.96	**0.6946**	0.6037
sigma_100, ranking, L=70											
58	100	6656.1	5383.9	**100**	100	8922.5	6499.4	1.72	1.34	**0.0079**	0.0183
58	100	6656.1	5383.9	**220**	100	7027.8	5935.6	3.79	1.06	**0.6433**	0.9722
58	100	6656.1	5383.9	**484**	100	6443.7	5281.1	8.34	0.97	**0.7785**	0.8554
100	100	8922.5	6499.4	**220**	100	7027.8	5935.6	2.2	0.79	**0.0326**	0.0345
100	100	8922.5	6499.4	**484**	100	6443.7	5281.1	4.84	0.72	**0.0035**	0.014
220	100	7027.8	5935.6	**484**	100	6443.7	5281.1	2.2	0.92	**0.4631**	0.3976
sigma_100, sus, L=70											
58	100	5590.8	4075.6	**100**	100	6690.9	4384.1	1.72	1.2	**0.0676**	0.0743
58	100	5590.8	4075.6	**220**	100	8537.7	4231.1	3.79	1.53	**1.2E-06**	2.6E-06
58	100	5590.8	4075.6	**484**	100	10759.0	4823.8	8.34	1.92	**3.7E-14**	1.6E-10
100	100	6690.9	4384.1	**220**	100	8537.7	4231.1	2.2	1.28	**0.0028**	0.0005
100	100	6690.9	4384.1	**484**	100	10759.0	4823.8	4.84	1.61	**2.6E-09**	3.8E-07
220	100	8537.7	4231.1	**484**	100	10759.0	4823.8	2.2	1.26	**0.0007**	0.0019

Таблиця 3.1 Результат статистичного аналізу різниці кількості ітерацій

Пояснення до колонок:

1. **d_N** - $d_N = \frac{N_A}{N_B}$.
2. **d_mean** - $d_{mean} = \frac{mean_A}{mean_B}$. Зеленим виділені значення більші за 1, тобто середня кількість ітерацій більша із більшим N, і навпаки - виділено червоним.
3. **p-value** - зеленим виділені Welch pvalue нижчі за α-рівень (0.05), що позначають статистично значиму відмінність.

Бачимо, що тільки для SUS для всіх проаналізованих пар кількість ітерацій дійсно більша із збільшенням N (d_mean > 1) і спостерігаємо статистичну значимість для значень d_N > 2, а для d_N = 1.2 значення p-value = 0.0676, що є вищим на порогове значення 0.05.

Для tournament_2 та tournamen_20 маємо схожі результати також на функціях spherical та shifted_rastrigins.

3.3.2 Підібраний зменшений Pmax

З графіків можна бачити, що деяка частина прогонів не збігаються за 20000 ітерацій, проте вони могли б збігтись за більшу кількість ітерацій, якщо ліміт було б збільшено. Це може бути спричинено стохастичністю алгоритму підбору Pmax.

З пункту 3.2 ми знаємо, що зменшення ймовірності мутації веде до зменшення кількості ітерацій разом із втратою точності. Спробуємо також провести аналогічні експерименти до тих, що описані в пункті 3.3.1, але зменшивши ймовірність мутації на 90%, 50% та 10% для наочності, тобто $P_m \in \{0.1Pmax_{f,sel,L,N}, 0.5Pmax_{f,sel,L,N}, 0.9Pmax_{f,sel,L,N}\}$.

Побудуємо аналогічні графіки до тих, що представлені в пункті 3.3.1. Для функції sigma_100 різниця для SUS ще посилюється із зменшенням P_{mut}:

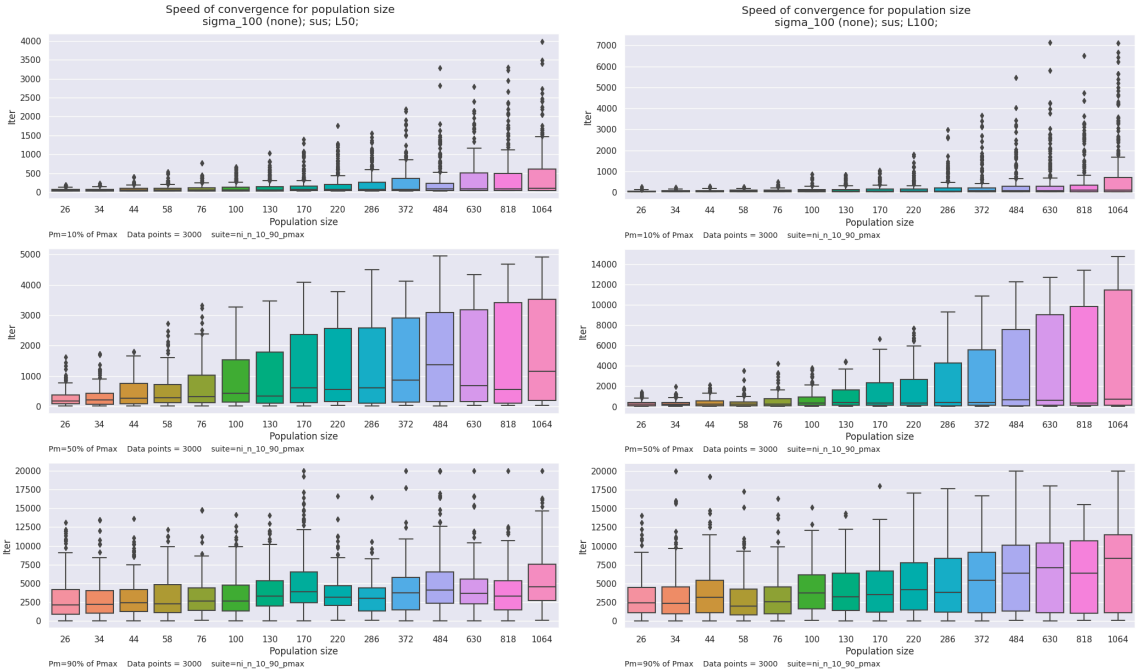


Рисунок 3.13 Графіки швидкості збіжності за розміром популяції для функції sigma_100 і SUS

Разом із зменшенням середнього також спостерігаємо збільшений розкид значень.

Ані для tournament_10, ні для ranking такої закономірності для середнього значення або для розкиду не спостерігається для різної довжини особини L.

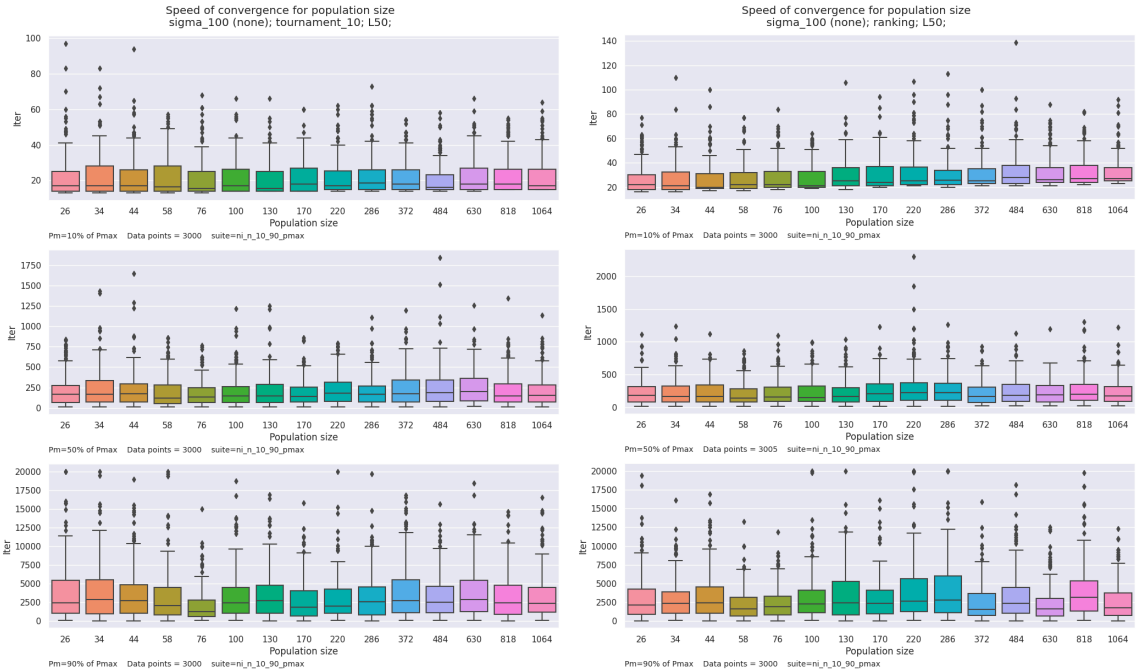


Рисунок 3.14 Графіки швидкості збіжності за розміром популяції для функції sigma_100 і tournament_10

Для інших функцій spherical та shifted_rastrigins ми так само маємо лише прогони із турнірним відбором tournament_10 і залежність відсутня.

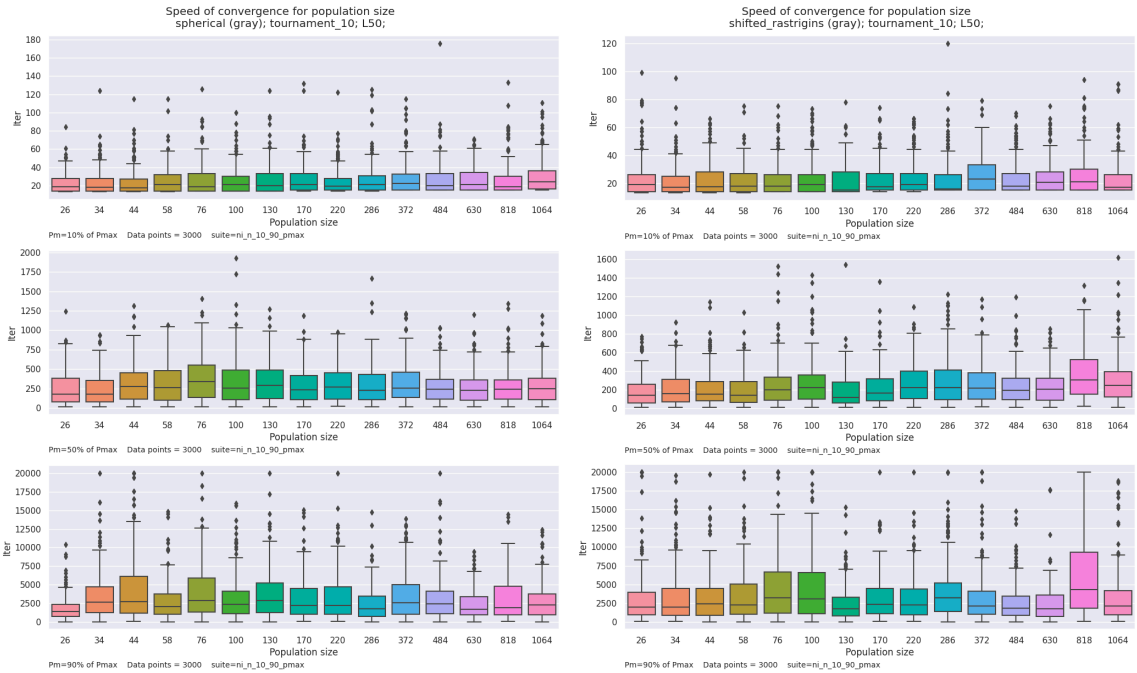


Рисунок 3.14 Графіки швидкості збіжності за розміром популяції для функції spherical, shifted_rastrigins і tournament_10

Для перевірки значимості різниці кількості ітерацій для різних розмірів популяції для SUS та відсутності цієї різниці для інших методів відбору проведемо статистичний аналіз.

A				B				diff B/A		pvalue	
N	**count**	**mean**	**std**	**N**	**count**	**mean**	**std**	**d_N**	**d_mean**	**Welch**	**Wilcoxon**
sigma_100, tournament_10, L=50, Pm=50% Pmax											
58	100	204.3	183.9	**100**	100	214.6	222.3	1.72	1.05	**0.7212**	0.9833
58	100	204.3	183.9	**220**	100	213.2	169.9	3.79	1.04	**0.7217**	0.8519
58	100	204.3	183.9	**484**	100	269.0	243.0	8.34	1.32	**0.035**	0.054
100	100	214.6	222.3	**220**	100	213.2	169.9	2.2	0.99	**0.9607**	0.7992
100	100	214.6	222.3	**484**	100	269.0	243.0	4.84	1.25	**0.1**	0.036
220	100	213.2	169.9	**484**	100	269.0	243.0	2.2	1.26	**0.0615**	0.1177
sigma_100, ranking, L=50, Pm=50% Pmax											
58	100	238.0	187.1	**100**	100	281.5	203.9	1.72	1.18	**0.1182**	0.1799
58	100	238.0	187.1	**220**	100	370.5	366.8	3.79	1.56	**0.0016**	0.0035
58	100	238.0	187.1	**484**	100	290.5	199.9	8.34	1.22	**0.0569**	0.0401
100	100	281.5	203.9	**220**	100	370.5	366.8	2.2	1.32	**0.0355**	0.1057

100	100	281.5	203.9	**484**	100	290.5	199.9	4.84	1.03	**0.7527**	0.846
220	100	370.5	366.8	**484**	100	290.5	199.9	2.2	0.78	**0.0574**	0.2459
sigma_100, sus, L=50, Pm=50% Pmax											
58	100	812.9	589.1	**100**	100	1509	760	1.72	1.86	**1.10E-11**	1.90E-09
58	100	812.9	589.1	**220**	100	2427	736	3.79	2.99	**2.60E-40**	2.70E-17
58	100	812.9	589.1	**484**	100	3058	700	8.34	3.76	**3.90E-61**	4.30E-18
100	100	1509.2	759.9	**220**	100	2427	736	2.2	1.61	**1.40E-15**	1.70E-10
100	100	1509.2	759.9	**484**	100	3058	700	4.84	2.03	**2.30E-34**	1.10E-16
220	100	2427.3	735.6	**484**	100	3058	700	2.2	1.26	**3.10E-09**	2.20E-08

Таблиця 3.2 Результат статистичного аналізу різниці кількості ітерацій

Так само бачимо, що статистична значима відмінність присутня лише для відбору SUS.

Отже, можемо зробити висновок, що Гіпотеза 2 у загальному випадку не підтвердилась. Збільшення кількості ітерацій до збіжності присутнє лише для методу відбору SUS з тих методів, що були розглянуті.

3.4 Вплив розміру популяції на збіжність за відсутності мутації

Гіпотеза 3: «За відсутності мутації, збіжність алгоритму сповільнюється із збільшенням розміру популяції».

Також перевіримо, чи зберігається залежність, описана в пункті 3.3, за відсутності мутації. Для цього проведемо експерименти із $P_m = 0$ та $P_{cross} = 1$. Позначка експерименту в Додатку Д - **cross_no_mut**.

Для цього випадку для функції sigma_100 спостерігаємо ріст кількості ітерацій із збільшенням розміру популяції для всіх методів відбору, хоча для турнірного відбору зріст сповільнюється/зупиняється на N=170.

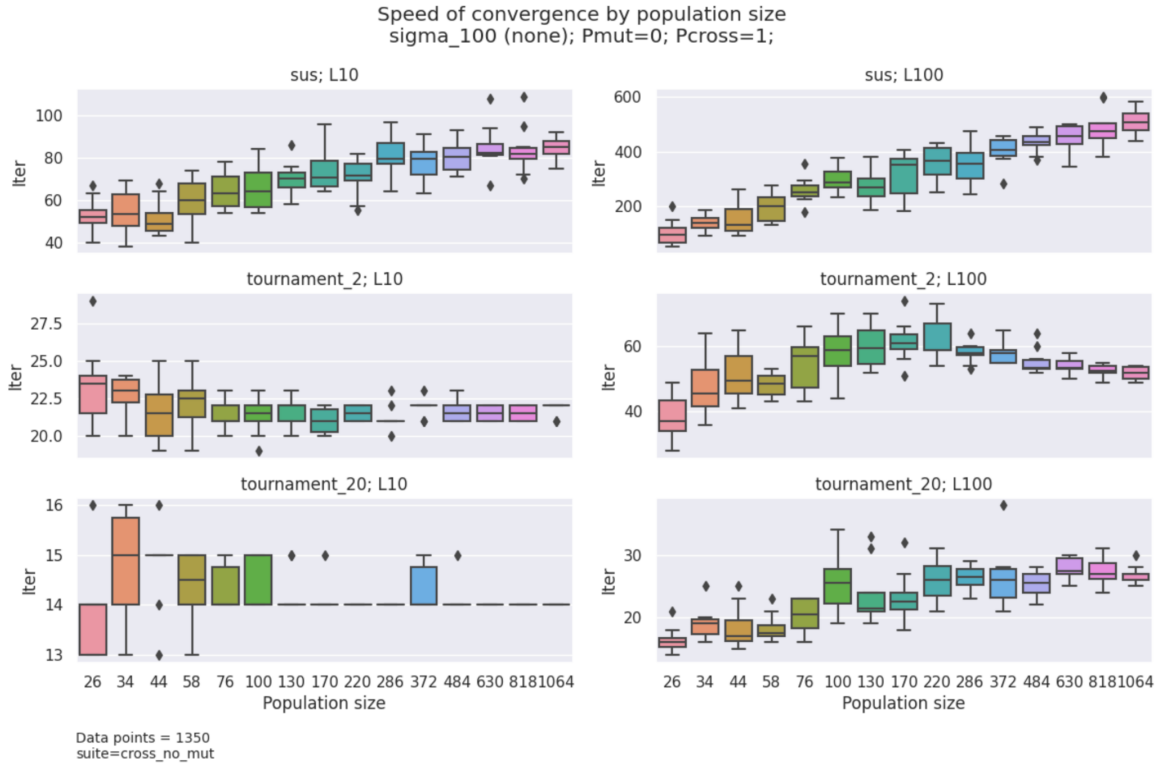


Рисунок 3.16 Графіки швидкості збіжності за розміром популяції для функції sigma_100

Для функцій spherical та shifted_rastrigins графіки для турнірних відборів дещо схожі:

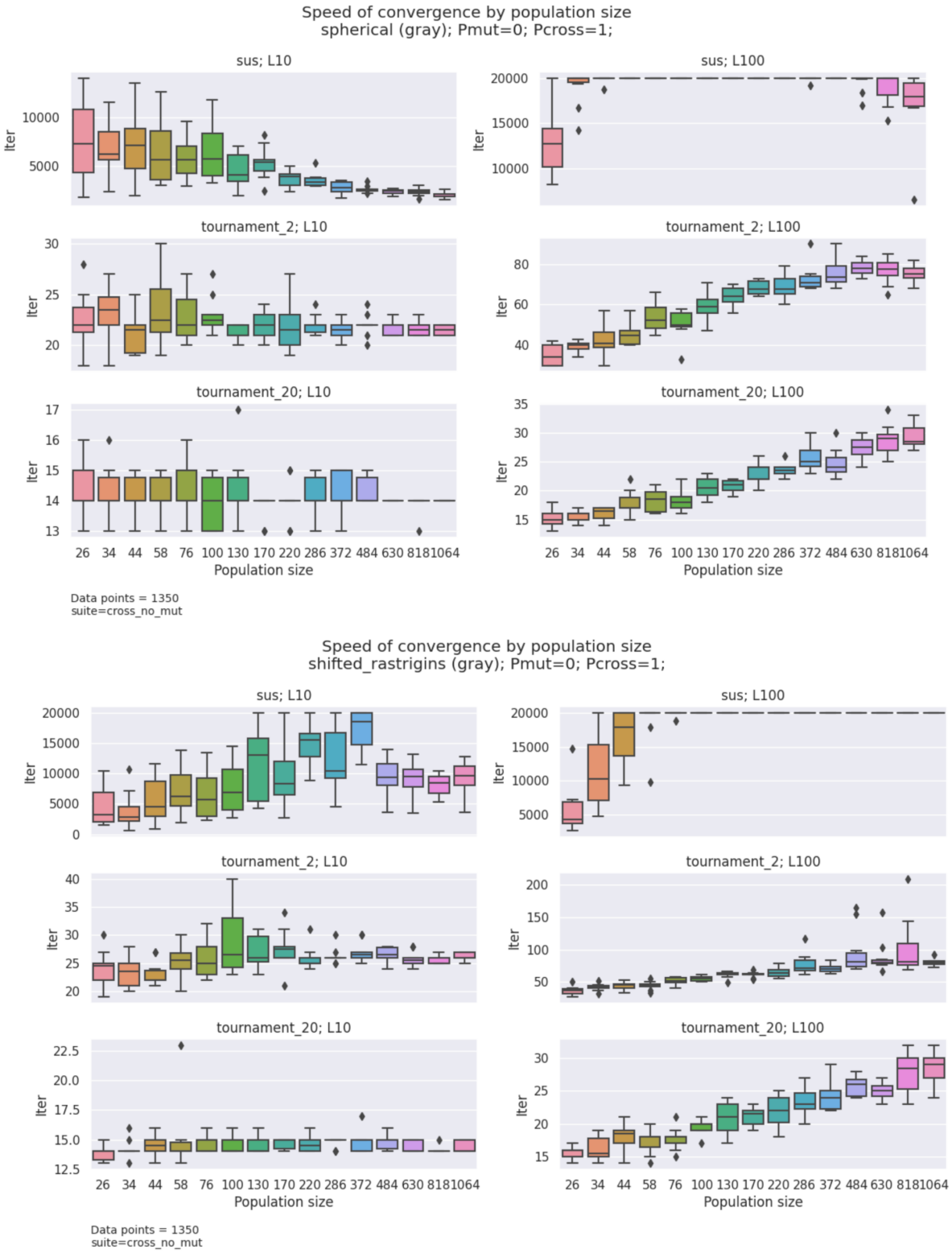


Рисунок 3.17 Графіки швидкості збіжності за розміром популяції для функції spherical, shifted_rastrigins та різних типів відбору

Для відбору SUS на функції дійсного аргументу збіжність майже відсутня для L=100 (також для L=50), але для L=10 бачимо, що збіжність пришвидшується.

Проведемо статистичний аналіз для функцій sigma_100, spherical для L=100 та різних типів відбору.

A				B				diff B/A		pvalue	
N	**count**	**mean**	**std**	**N**	**count**	**mean**	**std**	**d_N**	**d_mean**	**Welch**	**Wilcoxon**
sigma_100, tournament_2, L=100											
100	100	57.6	5.9	**220**	100	60.1	4.6	2.2	1.04	**0.0014**	0.0023
220	100	60.1	4.6	**484**	100	55.6	3.2	2.2	0.93	**1.60E-13**	7.90E-10
26	100	37.9	6.5	**100**	100	57.6	5.9	3.85	1.52	**3.10E-56**	4.60E-18
100	100	57.6	5.9	**484**	100	55.6	3.2	4.84	0.97	**0.0033**	0.0039
26	100	37.9	6.5	**220**	100	60.1	4.6	8.46	1.58	**3.40E-67**	3.80E-18
26	100	37.9	6.5	**484**	100	55.6	3.2	18.62	1.47	**1.70E-53**	3.80E-18
sigma_100, tournament_20, L=100											
100	100	22.5	3.8	**220**	100	25.3	3.3	2.2	1.13	**6.60E-08**	2.70E-07
220	100	25.3	3.3	**484**	100	26.5	2.3	2.2	1.05	**0.0035**	0.0013
26	100	16.4	2.1	**100**	100	22.5	3.8	3.85	1.37	**7.00E-30**	4.50E-16
100	100	22.5	3.8	**484**	100	26.5	2.3	4.84	1.18	**2.50E-16**	5.40E-11
26	100	16.4	2.1	**220**	100	25.3	3.3	8.46	1.55	**1.10E-52**	6.70E-18
26	100	16.4	2.1	**484**	100	26.5	2.3	18.62	1.62	**2.10E-80**	3.90E-18
sigma_100, sus, L=100											
100	100	280.7	54.6	**220**	100	357.1	56.2	2.2	1.27	**1.40E-18**	8.90E-13
220	100	357.1	56.2	**484**	100	442.2	53.5	2.2	1.24	**3.80E-22**	5.30E-14
26	100	105.5	43.9	**100**	100	280.7	54.6	3.85	2.66	**7.40E-62**	4.00E-18
100	100	280.7	54.6	**484**	100	442.2	53.5	4.84	1.58	**1.30E-52**	9.20E-18
26	100	105.5	43.9	**220**	100	357.1	56.2	8.46	3.38	**1.50E-84**	3.90E-18
26	100	105.5	43.9	**484**	100	442.2	53.5	18.62	4.19	**1.90E-109**	3.90E-18
spherical, tournament_2, L=100											
100	100	54.2	6.1	**220**	100	67.8	5.1	2.2	1.25	**1.70E-40**	1.60E-17
220	100	67.8	5.1	**484**	100	75.4	5.8	2.2	1.11	**8.20E-19**	6.30E-14
26	100	34.8	5.3	**100**	100	54.2	6.1	3.85	1.56	**1.40E-60**	8.30E-18
100	100	54.2	6.1	**484**	100	75.4	5.8	4.84	1.39	**8.40E-64**	5.60E-18
26	100	34.8	5.3	**220**	100	67.8	5.1	8.46	1.95	**8.70E-106**	3.80E-18
26	100	34.8	5.3	**484**	100	75.4	5.8	18.62	2.17	**1.60E-116**	3.80E-18

spherical, tournament_20, L=100											
100	100	19	1.6	**220**	100	21.9	2.2	2.2	1.15	**3.00E-20**	4.20E-14
220	100	21.9	2.2	**484**	100	25.6	2.4	2.2	1.17	**1.60E-24**	4.40E-15
26	100	15.5	1.3	**100**	100	19	1.6	3.85	1.23	**2.80E-39**	2.90E-17
100	100	19	1.6	**484**	100	25.6	2.4	4.84	1.35	**1.70E-54**	7.20E-18
26	100	15.5	1.3	**220**	100	21.9	2.2	8.46	1.41	**2.20E-58**	3.80E-18
26	100	15.5	1.3	**484**	100	25.6	2.4	18.62	1.66	**1.90E-79**	3.40E-18

Таблиця 3.2 Результат статистичного аналізу різниці кількості ітерацій

Бачимо, що всі відмінності у кількості ітерацій є статистично значимими. Загалом, бачимо, що із збільшенням N збільшується і кількість ітерацій. Тільки для турнірного відбору tournament_2 на sigma_100 помічаємо, що із збільшенням N в 2.2 та 4.84 рази, кількість ітерацій *зменшується* на 7% та 3% відповідно. Ця відмінність може бути спричинена шумом відбору. Наприклад, таке саме збільшення N в 2.2 рази з 100 до 220 (перший рядок таблиці), спричинило *збільшення* середньої кількості ітерацій на 4%.

Отже, можемо підтвердити Гіпотезу 3 для випадку турнірного відбору. Для методу відбору SUS гіпотезу не можемо ні спростувати, ні підтвердити.

3.5 Вплив розміру популяції на збіжність за фіксованого значення Pm

Гіпотеза 4: «За довільно обраного фіксованого значення ймовірності мутації, збіжність алгоритму сповільнюється або втрачається із збільшенням розміру популяції».

Зафіксуємо ймовірність мутації для всіх розмірів популяції N на одному Pmax, який було підібрано для одного розміру, наприклад 100, тобто $P_m = Pmax_{f,sel,L,N=100}$.

Побудуємо аналогічні графіки до тих, що представлені в попередніх пунктах, розділивши в окремі стовпці прогони, в який було і не було знайдено розв'язок. Для функції sigma_100 і L=50 графіки виглядають так:

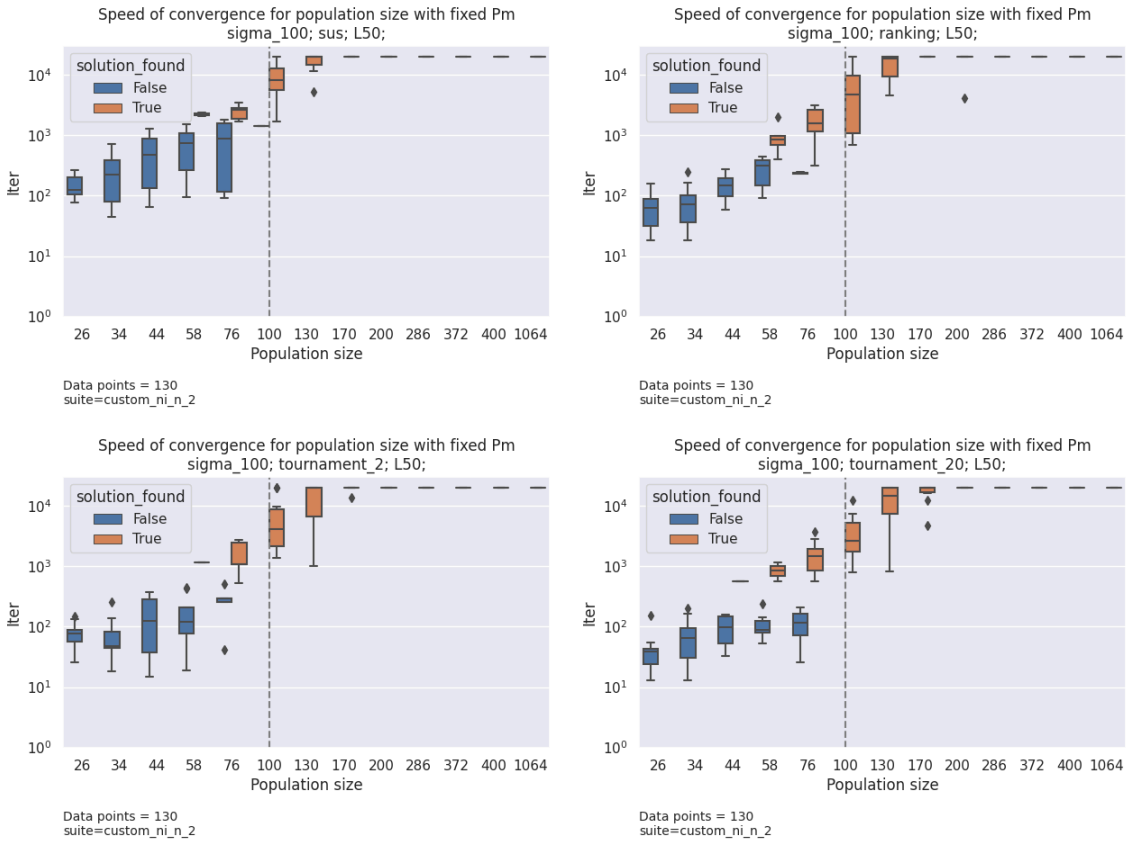


Рисунок 3.15 Графіки швидкості збіжності за розміром популяції для функції sigma_100 та різних типів відбору

Звернемо увагу, що вісь Oy має логарифмічну шкалу на цьому типі графіків.

Графіки для інших функцій - spherical, shifted_rastrigins - та типів відбору - tournament_2, tournament_20 - виглядають аналогічно. Провести статистичний аналіз не можемо через замалу кількість даних.

Отже дійсно за фіксованого значення ймовірності мутації, із збільшенням популяції збіжність сповільнюється або втрачається. Тобто, Гіпотеза 4 підтвердилась графічно.

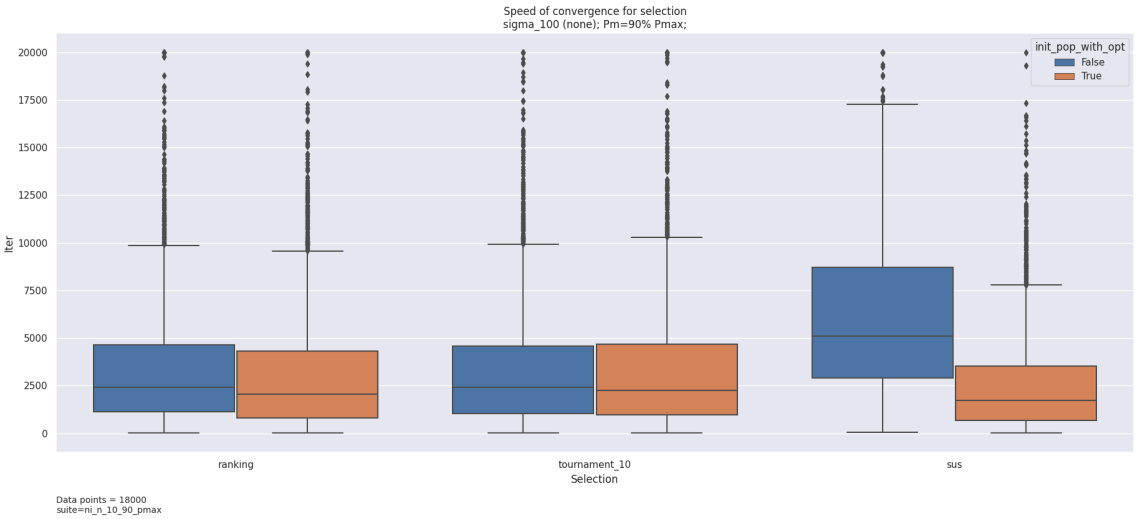
3.6 Вплив ініціалізації з оптимальною особиною на збіжність

Гіпотеза 5: «На значеннях ймовірності мутації близьких до Pmax, збіжність алгоритму є швидшою, якщо в початковій популяції присутня оптимальна особина, ніж коли вона відсутня».

При аналізі впливу інших факторів на швидкість збіжності, ми не враховували той факт, що початкова популяція могла мати вже оптимальну особину. Наявність такої особини теоретично може зменшувати кількість ітерацій до збіжності, оскільки при відборі вона завжди матиме перевагу.

Щоб перевірити цю гіпотезу, проведемо експерименти з різними типами відбору та параметрами L, N, по два прогони на кожний такий набір, де в одному прогоні штучно додано оптимальну особину до початкової популяції. Так само, як і в пункті 3.3.2, зменшимо Pm на 10%, щоб збільшити кількість прогонів, що збіглись до 20000 ітерацій.

Побудуємо графік розмаху окремо для кожного типу відбору для всіх комбінацій L, N разом.



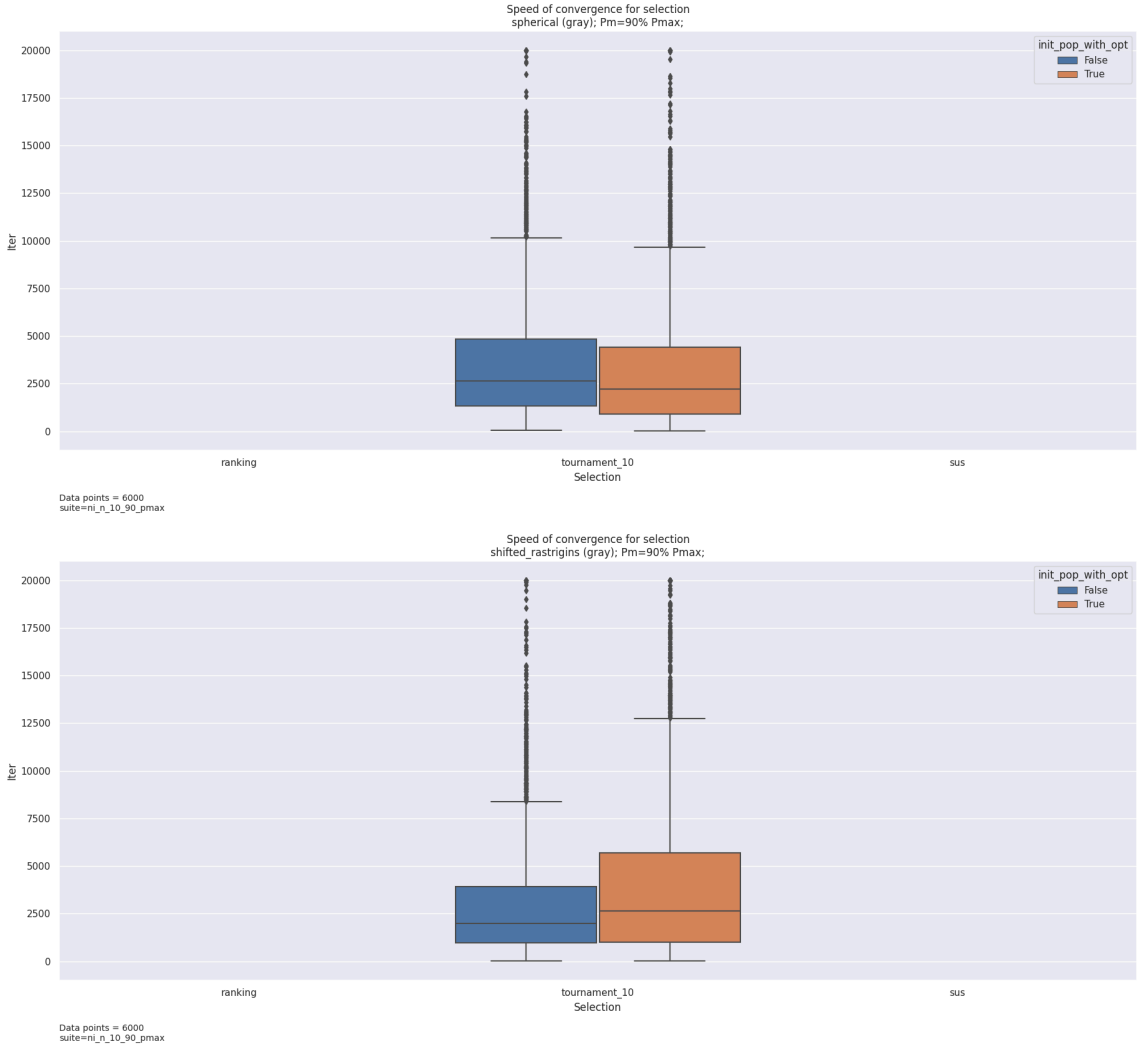


Рисунок 3.18 Графіки швидкості збіжності по типу відбору

Одразу можемо бачити, що наявність оптимальної особини сильно впливає лише на метод відбору SUS. Така сама тенденція зберігається і окремо на кожній комбінації L/N.

Перевіримо, чи є різниця в кількості ітерацій статистично значимою.

N	A (init with opt)			B (init without opt)			d_mean	pvalue	
	count	mean	std	count	mean	std		Welch	Wilcoxon
sigma_100, tournament_10, L=50									
58	100	3213.7	3210.7	100	3392.6	3884.1	1.06	**0.7229**	0.6549
100	100	2792	2945.6	100	3938.5	3667.6	1.41	**0.0157**	0.0046
220	100	3317	3289.5	100	2798.7	2580.7	0.84	**0.2167**	0.3221

484	100	3356.7	3090.5	100	3164.6	2676.5	0.94	**0.6391**	0.9507
sigma_100, ranking, L=50									
58	100	2066.1	1952.3	100	2328.1	2205.1	1.13	**0.3749**	0.4971
100	100	3405.8	3351.8	100	3079	3112.1	0.9	**0.4757**	0.731
220	100	3677.8	3862.9	100	4443.9	4296.4	1.21	**0.1864**	0.207
484	100	3204.4	3158.2	100	3411.8	3381.9	1.06	**0.6545**	0.778
sigma_100, sus, L=50									
58	100	2375.9	2313.3	100	3854.1	2718	1.62	**5.20E-05**	1.10E-05
100	100	2534.8	2709.1	100	4225.3	2679.9	1.67	**1.50E-05**	2.50E-06
220	100	2984	2667.2	100	4317.9	2281.2	1.45	**0.0002**	4.30E-05
484	100	3250.2	3264.1	100	6776.9	4220.4	2.09	**3.90E-10**	4.00E-09
spherical, tournament_10, L=50									
58	100	2913.4	3031.4	100	2777.6	2287.8	0.95	**0.721**	0.9315
100	100	3292.6	3123.1	100	3282.4	3007.9	1	**0.9812**	0.8554
220	100	3307.4	3060.8	100	3569.9	3623.8	1.08	**0.5807**	0.7544
484	100	3155.2	3073	100	3099.8	2870.7	0.98	**0.8952**	0.9849
shifted_rastrigins, tournament_10, L=50									
58	100	4390.1	4131.1	100	2595.2	2746.9	0.59	**0.0004**	0.0021
100	100	5608.2	5500.4	100	3871.1	3837.9	0.69	**0.0104**	0.0559
220	100	3524.2	3558.3	100	2860	2333.7	0.81	**0.1204**	0.212
484	100	3343.7	3128.8	100	1855.3	1559.3	0.55	**3.7E-05**	0.0001

Таблиця 3.3 Результат статистичного аналізу різниці кількості ітерацій

Дійсно, для SUS, L=50 на всіх значеннях N кількість ітерацій значимо більша у випадку ініціалізації без оптимальної особини. Такий же результат і для L=100. Для всіх інших методів відбору статистичної значимості немає.

Варто зазначити, що для функції sifted_rastrigins, L=50, ініціалізація без оптимальною особиною статистично значимо *зменшує* кількість ітерацій, окрім N=220.

Ще більше різниця для SUS на sigma_100 стає помітною, якщо ми виведемо кількість ітерацій в залежності від розміру популяції:

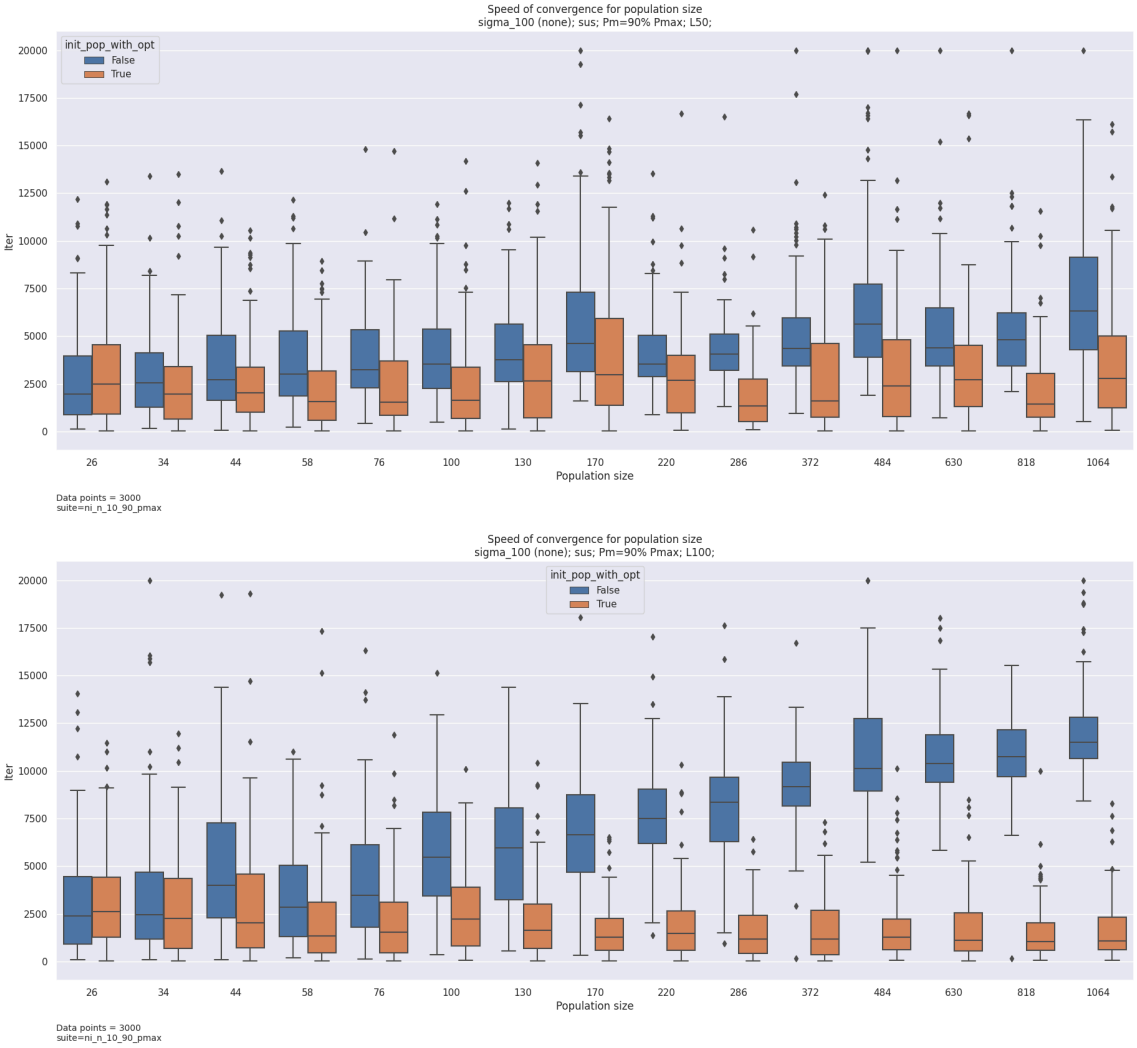


Рисунок 3.19 Графіки швидкості збіжності за розміром популяції для функції sigma_100 та SUS

Бачимо, що зі збільшенням кількості особин, кількість ітерацій, необхідна для збіжності, також збільшується, але лише тоді, коли популяція була ініціалізована без оптимальної особини.

Можемо зробити висновок, що Гіпотеза 5 справджується лише для методу відбору SUS, а для турнірного відбору та відбору за рангом різниці немає.

3.7 Вплив кросинговеру на збіжність та успішність

Гіпотеза 6.1: «На значеннях ймовірності мутації, близьких до Pmax, кросинговер пришвидшує збіжність алгоритму».

Гіпотеза 6.2: «На значеннях ймовірності мутації, близьких до Pmax, кросинговер покращує успішність прогонів».

Проаналізуємо також вплив кросинговеру на кількість ітерацій до збіжності та точність розв'язку. Для цього проведемо експерименти зі значенням ймовірності кросинговеру $P_{cross} \in \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}$. Значення ймовірності мутації - підібране $P_{mut} = Pmax_{f,sel,L,N}$. Позначка експерименту в Додатку Д - **cross**.

Для функції sigma_100 загалом не спостерігається зміни кількості ітерацій при зміні P_{cross}. В той же час, бачимо, що при підвищенні ймовірності кросинговеру точність алгоритму зростає, особливо це помітно на $L \in \{50, 100\}$:

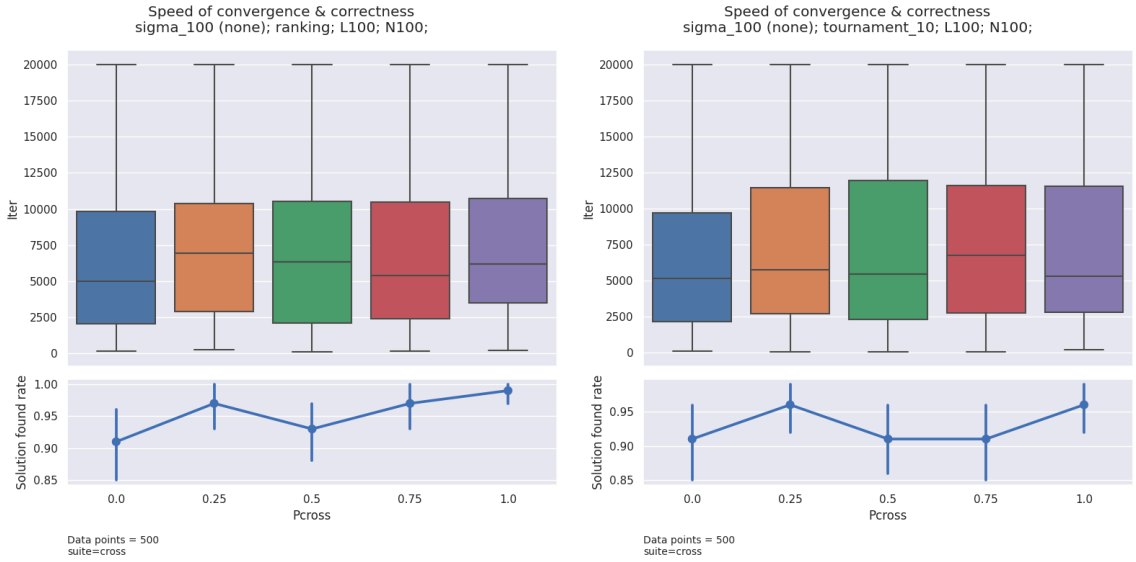


Рисунок 3.20 Графіки швидкості збіжності за розміром популяції для функції sigma_100

Для SUS спостерігається аналогічна поведінка для точності, але також фіксуємо суттєво збільшену кількість ітерацій при відсутності кросинговеру:

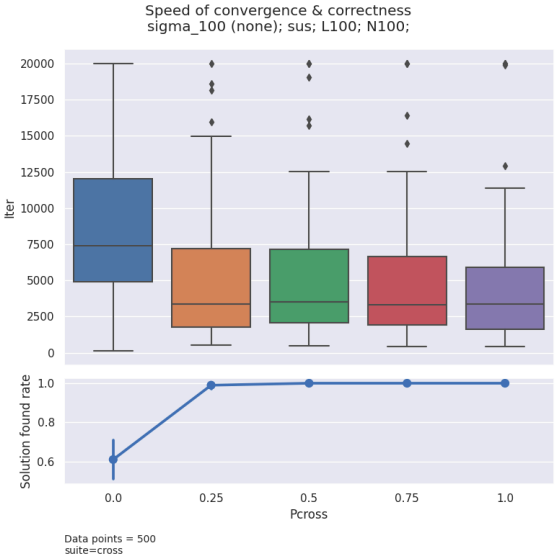


Рисунок 3.21 Графік швидкості та точності збіжності за параметром Pcross для функції sigma_100

Для функції spherical бачимо аналогічну ситуацію, але для L=50 точність трохи знизилась при $P_{cross} = 1$.

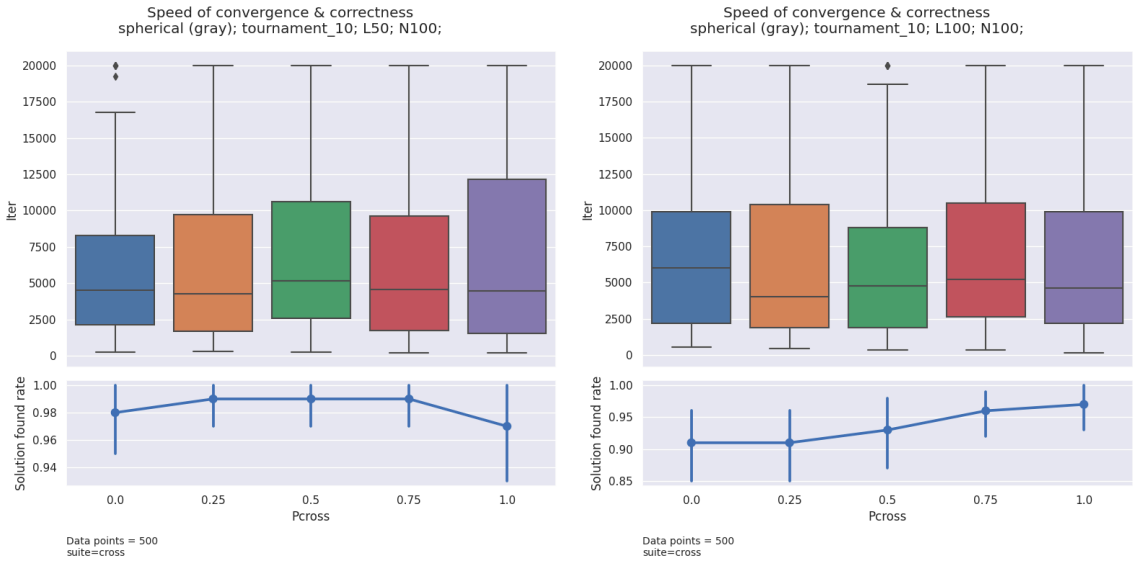


Рисунок 3.22 Графіки швидкості та точності збіжності за параметром Pcross для функції spherical

Для функції shifted_rastrigins бачимо, що кількість ітерацій так само незмінна. Проте наявність агресивного кросинговеру - $P_{cross} = 1$ - дає

можливість підняти ймовірність знайти точний розв'язок для такої складної функції на розмірності 10 ($L = 100$):

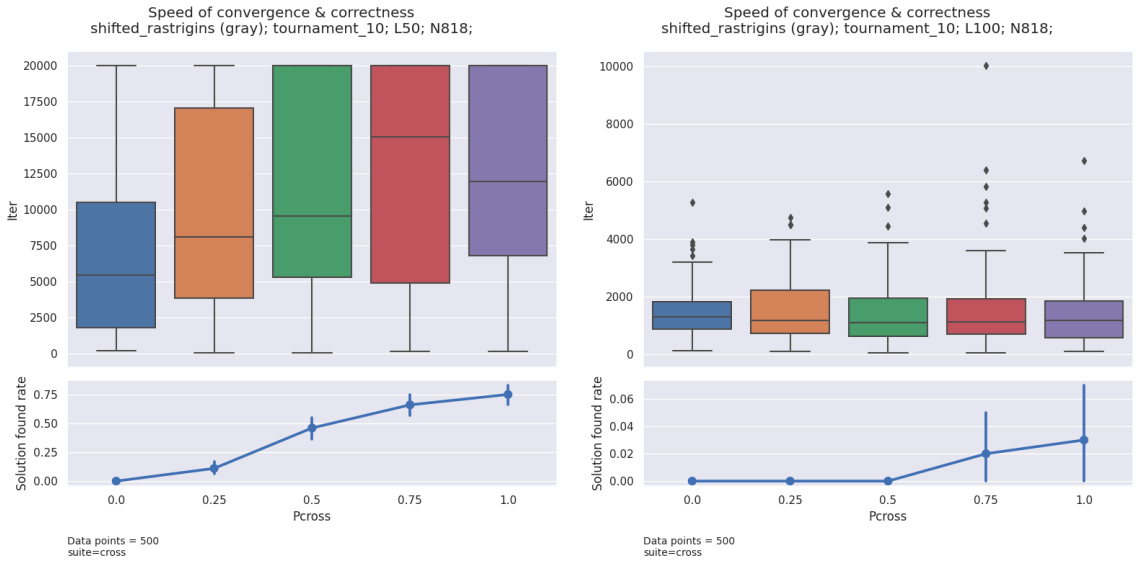


Рисунок 3.23 Графіки швидкості та точності збіжності за параметром Pcross для функції shifted_rastrigins

Проведемо статистичний аналіз кількості ітерацій для прогонів із L=50 та $P_{cross} \in \{0, \frac{1}{2}, 1\}$.

A				B				d_mean	pvalue	
P_cross	**count**	**mean**	**std**	**P_cross**	**count**	**mean**	**std**		**Welch**	**Wilcoxon**
sigma_100, tournament_10, L=50, N=100										
0	100	6906	5505.3	**0.5**	100	6897.2	5974	1	**0.9913**	0.9485
0	100	6906	5505.3	**1**	100	6128.5	5508.4	0.89	**0.3193**	0.2396
0.5	100	6897.2	5974	**1**	100	6128.5	5508.4	0.89	**0.3453**	0.3732
sigma_100, ranking, L=50, N=100										
0	100	8021.1	6252.1	**0.5**	100	5620	5251.9	0.7	**0.0037**	0.006
0	100	8021.1	6252.1	**1**	100	7144.2	5717.6	0.89	**0.3019**	0.4221
0.5	100	5620	5251.9	**1**	100	7144.2	5717.6	1.27	**0.051**	0.076
sigma_100, sus, L=50, N=100										
0	100	7414.6	5003.3	**0.5**	100	5852.8	4814.6	0.79	**0.0256**	0.021
0	100	7414.6	5003.3	**1**	100	4533.5	3631.2	0.61	**6.1E-06**	1.6E-05
0.5	100	5852.8	4814.6	**1**	100	4533.5	3631.2	0.77	**0.03**	0.0283

spherical, tournament_10, L=50, N=818										
0	100	7809.2	6353.4	**0.5**	100	6279.5	5650.5	0.8	**0.0736**	0.1002
0	100	7809.2	6353.4	**1**	100	6517.1	5458.5	0.83	**0.1246**	0.1712
0.5	100	6279.5	5650.5	**1**	100	6517.1	5458.5	1.04	**0.7626**	0.3532
shifted_rastrigins, tournament_10, L=50, N=818										
0	100	6990.8	5903.3	**0.5**	100	10859.5	6985.2	1.55	**3.6E-05**	0.0001
0	100	6990.8	5903.3	**1**	100	12318.3	6713.6	1.76	**1.2E-08**	1.6E-07
0.5	100	10859.5	6985.2	**1**	100	12318.3	6713.6	1.13	**0.1337**	0.1497

Статистичну значимість маємо лише для методу відбору SUS - кросинговер дійсно зменшив кількість ітерацій на 39%. Для функції shifted_rastrigins кількість ітерацій значимо зросла на 76%.

Отже, можемо зробити припущення, що кросинговер “не заважає” мутації за граничного значення Pmax, він не змінює кількість ітерацій, окрім випадку із SUS. Тому Гіпотеза 6.1 справджується лише для SUS.

У той же час агресивний кросинговер із $P_{cross} = 1$ покращує точність знайденого розв’язку, особливо для складних багатовимірних функцій. Тобто, Гіпотеза 6.2 підтвердилась.

Висновки

В ході роботи було проведено дослідження впливу параметра мутації на збіжність генетичного алгоритму. Для цього було розроблено програмний застосунок мовою Python: висунуті вимоги до функціональних можливостей застосунку та його швидкодії, розроблено схему бази даних для збереження інформації.

За допомогою розробленого застосунку проведено серію експериментів із підбору граничних значень ймовірності мутації Pmax, а також інші експерименти із різними наборами параметрів.

Використано модулі для візуального та статистичного аналізу згенерованих даних. Були висунуті та перевірені гіпотези щодо залежностей між параметрами алгоритму та характеристиками його збіжності - швидкістю та успішністю. Коротко ці гіпотези та результат їх перевірки представлені в цій таблиці:

№	Гіпотеза	Результат
1.1	«На значеннях ймовірності мутації, близьких до Pmax, збіжність алгоритму є швидшою, ніж на значеннях більше або менше за Pmax»	Не підтверджено
1.2	«На значеннях ймовірності мутації, близьких до Pmax, відсоток успішних прогонів є вищим, ніж на значеннях більше або менше за Pmax»	Підтверджено
2	«За значення ймовірності мутації близькому до Pmax, збіжність алгоритму сповільнюється із збільшенням розміру популяції»	Підтверджено лише для відбору SUS
3	«За відсутності мутації, збіжність алгоритму сповільнюється із збільшенням розміру популяції»	Підтверджено лише для турнірного відбору

4	«За довільно обраного фіксованого значення ймовірності мутації, збіжність алгоритму сповільнюється або втрачається із збільшенням розміру популяції»	Підтверджено (графічно)
5	«На значеннях ймовірності мутації близьких до Pmax, збіжність алгоритму є швидшою, якщо в початковій популяції присутня оптимальна особина, ніж коли вона відсутня»	Підтверджено лише для відбору SUS
6.1	«На значеннях ймовірності мутації, близьких до Pmax, кросинговер пришвидшує збіжність алгоритму»	Підтверджено лише для відбору SUS
6.2	«На значеннях ймовірності мутації, близьких до Pmax, кросинговер покращує успішність прогонів»	Підтверджено

Підсумовуючи все вищенаведене, можна зробити наступні висновки.

Хоча вибір більших значень ймовірності, але менших за граничне значення Pmax, і сповільнює швидкість, при цьому також зростає і точність знайдених розв’язків. Тобто, обрання Pmax для ймовірності мутації є компромісом, який забезпечує як достатню точність, так і низьку швидкість до збіжності, необхідну для цієї точності.

Із проведених експериментів також можна припустити, що деяка залежність між розміром популяції та кількістю ітерацій існує, а саме, що при збільшенні N може збільшуватись середня кількість ітерацій до збіжності. Проте бачимо, що на цю залежність сильно впливає ряд чинників, таких як тип відбору, ймовірність мутації та наявність кросинговеру. Консистентної залежності для різних наборів параметрів не знайдено.

Також спостерігаємо, що за значення ймовірності мутації на рівні Pmax наявність кросинговеру хоч і не завжди зменшує кількість ітерацій

до збіжності, але надає алгоритму властивість знаходити глобальний оптимум задачі частіше. При розв’язанні складних до оптимізації проблем, використання кросинговеру з $P_{cross} = 1$ залишається рекомендованим, оскільки дозволяє покращити результат, не сповільнивши при цьому збіжність.

Список використаної літератури

1. Кобєлєв М. Д. Налаштування параметра мутації генетичного алгоритму – Курсова робота 4 р. н. .– Нац. ун-т "Києво-Могилян. акад.", Київ 2021
2. PostgreSQL: Documentation: 12: 13.2. Transaction Isolation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/12/transaction-iso.html.
3. Introduction - Maturin User Guide [Електронний ресурс] – Режим доступу до ресурсу: https://www.maturin.rs/index.html.
4. Introduction - PyO3 user guide [Електронний ресурс] – Режим доступу до ресурсу: https://pyo3.rs/v0.12.3/.
5. PyO3/rust-numpy: PyO3-based Rust bindings of the NumPy C-API [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/PyO3/rust-numpy.
6. Eiben, Agoston E., and James E. Smith. “Fitness, Selection, and Population Management” *Introduction to evolutionary computing*. Springer-Verlag Berlin Heidelberg, 2015.
7. A. Törn. Global Optimization / A. Törn, A. Zilinskas // Lecture Notes in Computer Science / A. Törn, A. Zilinskas. – Berlin: Springer-Verlag, 1989.
8. Глибовець М. М. Еволюційні алгоритми : підручник / М. М. Глибовець, Н. М. Гулаєва ; Нац. ун-т "Києво-Могилян. акад.". - Київ : [НаУКМА], 2013. - 826 с. : іл. - Містить покажчик.
9. Frank, Gray. "Pulse code communication." U.S. Patent No. 2,632,058. 17 Mar. 1953.
10. Baker, James E. (1987). "Reducing Bias and Inefficiency in the Selection Algorithm". *Proceedings of the Second International*

Conference on Genetic Algorithms and Their Application. Hillsdale, New Jersey: L. Erlbaum Associates: 14–21.

11. PostgreSQL: Documentation: 12: pg_dump [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/12/app-pgdump.html.
12. PostgreSQL: Documentation: 12: pg_restore [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/12/app-pgrestore.html.
13. PostgreSQL: Documentation: 12: dblink [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/12/contrib-dblink-function.html.
14. Cloud Object Storage – Amazon S3 – Amazon Web Services [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/s3/.
15. Project Jupyter | Home [Електронний ресурс] – Режим доступу до ресурсу: https://jupyter.org/.
16. NumPy documentation — NumPy v1.24 Manual [Електронний ресурс] – Режим доступу до ресурсу: https://numpy.org/doc/1.24/index.html.
17. API reference — pandas 2.0.2 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://pandas.pydata.org/pandas-docs/version/2.0.2/reference/index.html.
18. API Reference — Matplotlib 3.7.1 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://matplotlib.org/3.7.1/api/index.html.
19. API reference — seaborn 0.12.2 documentation [Електронний ресурс] – Режим доступу до ресурсу: https://seaborn.pydata.org/api.html.

20. SciPy documentation — SciPy v1.9.2 Manual[Електронний ресурс] – Режим доступу до ресурсу: https://docs.scipy.org/doc/scipy-1.9.2/index.html.
21. Montgomery, Douglas C., and George C. Runger. Applied statistics and probability for engineers. John wiley & sons, 2010.
22. Shapiro, Samuel Sanford, and Martin B. Wilk. "An analysis of variance test for normality (complete samples)." Biometrika 52.3/4 (1965): 591-611.
23. Rechenberg, Ingo. *Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Diss. Technische Universität, Fakultät für Maschinenwissenschaft, 1970.

ДОДАТКИ

Додаток А

(довідниковий)

Розширення Python на Rust для відбору SUS та кросинговеру

Файл: `1_lib.rs`

Додаток Б

(довідниковий)

Імплементація кросинговеру та SUS

Файл: `2_crossover.py`

- crossover - із використанням звичайного циклу на Python.
- crossover_opt - із використанням розширення на Rust.

Файл: `2_sus.py`

- sus - із використанням звичайного циклу на Python.
- sus_opt - із використанням розширення на Rust.

Додаток В

(обов'язковий)

Тестові функції

Обернена відстань Гемінга

(на графіках позначається inverted_hamming_distance)

$f(x_L) = L - k, k -$ кількість нулів в послідовності x_L довжини L

Функція селективної переваги на біт

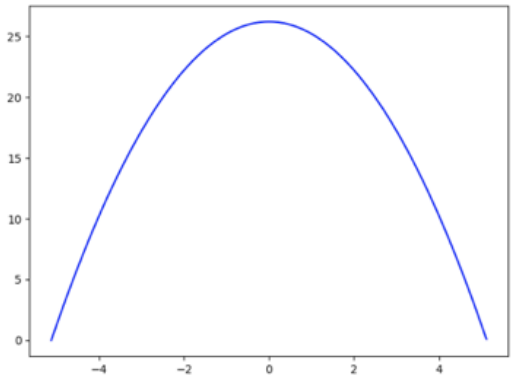
(на графіках позначається sigma_10, sigma_100)

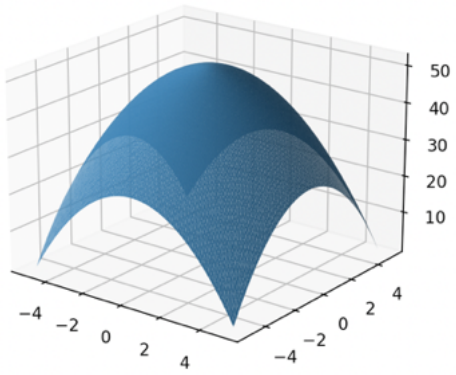
$f(x_L) = L - k + \sigma k, k -$ кількість нулів в послідовності x_L довжини L

Сферична функція (тестова ф-я Де Йонга)

(на графіках позначається sphercical)

$$f(x_1, \dots, x_n) = (5.12)^2 n - \sum_{i=1}^{n} x_i^2$$

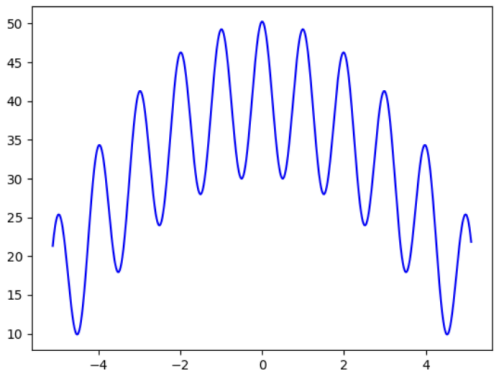


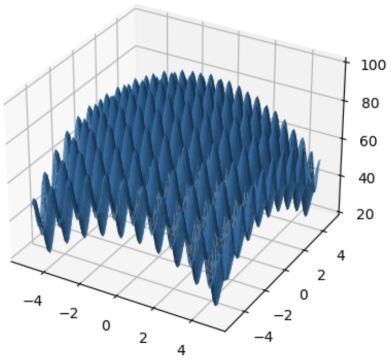


Зміщена вгору функція Растригіна [7]

(на графіках позначається shifted_rastrigins)

$$f(x_1, \dots, x_n) = n\,|10cos\,(2\pi a) - a^2| + \sum_{i=1}^{n} (10cos\,(2\pi x_i) - {x_i}^2), a = 5.5$$

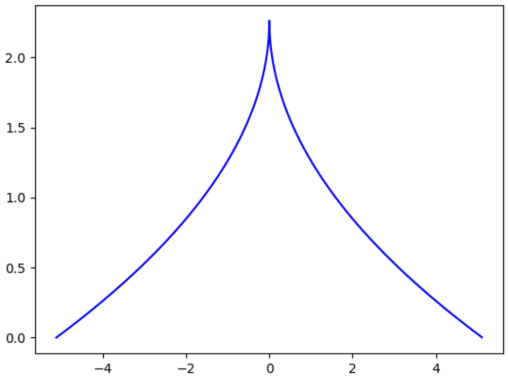


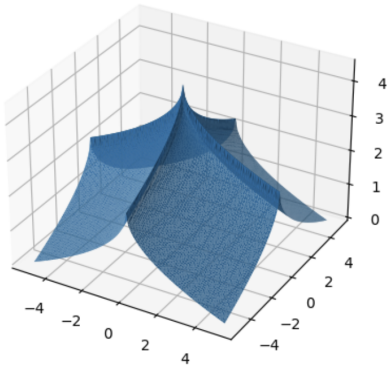


Перевернутий квадратний корінь

(на графіках позначається square_root)

$$f(x_1, \dots, x_n) = n\sqrt{|a|} - \sum_{i=1}^{n} \sqrt{|x_i|}, a = -5.12$$





Додаток Г

(довідниковий)

Перелік параметрів ГА для експериментів із підбору Pmax

Файл: `4_experiments_plan_pmax.xlsx`

Додаток Д

(обов’язковий)

Перелік параметрів ГА для довільних експериментів для аналізу

Файл: `5_experiments_plan_analyse.xlsx`

Додаток Е

(довідниковий)

Jupyter-notbooks

Notebook для побудови графіків

Файл: `6_graphs.ipynb`

Notebook для обчислення статистичних критеріїв

Файл: `6_stats.ipynb`

Додаток Ж

(обов'язковий)

Побудовані графіки

Файл: `7_all_graphs.zip`

Структура архіву:

- `3_1_ni_distribution` - графіки підрозділу 3.1
  - `1_analyse_100_runs` - пункт 3.1.1 - розподіл за 100 ітераціями
  - `2_sample_dist` - пункт 3.1.2 - вибірковий розподіл вибіркового середнього
- `3_2_evol_window` - графіки підрозділу 3.2
  - `1_by_L_N` - без групування
  - `2_by_L` - згруповані за L (графік для одного L але з усіма N)
  - `3_by_N` - згруповані за N (графік для одного N але з усіма L)
- `3_3_iter_num_N` - графіки підрозділу 3.3
  - `1_100_Pmax` - пункт 3.3.1.
  - `2_reduce_Pmax` - пункт 3.3.2.
- `3_4_iter_num_N_no_mut` - графіки підрозділу 3.4.
- `3_5_iter_num_N_fixed_Pm` - графіки підрозділу 3.5.
- `3_6_init_with_opt` - графіки підрозділу 3.6.
- `3_7_crossover` - графіки підрозділу 3.7