Звіт з лабораторної роботи №2
Застосування генетичного алгоритму для зламу RSA-шифрування

28 листопада 2025 р.

Зміст

1 Мета роботи

Метою роботи є реалізація простого генетичного алгоритму (за моделлю Холланда та Голдберга) для апроксимації невідомого відкритого тексту a таким чином, щоб результат шифрування `rsa_lib.encrypt_text(candidate, public_key, n)` був якомога ближчим до заданого шифротексту b.

2 Постановка задачі

2.1 Вхідні дані

Дано:

- Шифротекст: масив цілих чисел довжиною 42 елементи
- Публічний ключ: $e = 65537$
- Модуль: $n = 33227$
- Алфавіт: `"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ, "`
- Довжина тексту: 42 символи

2.2 Задача оптимізації

Необхідно знайти текст-кандидат x довжиною 42 символи з заданого алфавіту, який мінімізує функцію помилки:

$$P(x, b) = \frac{1}{L} \sum_{i=1}^{L} (b'_i - b_i)^2 \tag{1}$$

де:

- b — цільовий шифротекст
- b' — шифротекст, отриманий при шифруванні кандидата x
- L — довжина шифротексту

2.3 Функція пристосованості

Для застосування генетичного алгоритму функція помилки перетворюється у функцію пристосованості (fitness), яку необхідно максимізувати:

$$F(x) = \frac{1}{1 + P(x, b)} \tag{2}$$

Таким чином, менша помилка відповідає більшій пристосованості.

3 Опис генетичного алгоритму

3.1 Представлення індивіда

Кожен індивід у популяції представлений рядком довжиною 42 символи з алфавіту розміром 55 символів (52 літери англійського алфавіту + кома + пробіл).

3.2 Параметри алгоритму

Основні параметри генетичного алгоритму:

- Розмір популяції: `POP_SIZE = 100`
- Ймовірність мутації: `MUTATION_RATE = 0.02`
- Ймовірність кросоверу: `CROSSOVER_RATE = 0.8`
- Кількість елітних особин: `ELITE_COUNT = 2`
- Розмір турніру: `tournament_size = 3`
- Максимальна кількість поколінь: `MAX_GENERATIONS = 20000`
- Поріг помилки для зупинки: `TARGET_ERROR_THRESHOLD = 0.0`

3.3 Ініціалізація популяції

Початкова популяція генерується випадковим чином: кожен індивід складається з 42 випадково обраних символів з алфавіту.

```
def random_char():
    return random.choice(ALPHABET)

def random_text(length=TEXT_LENGTH):
    return "".join(random_char() for _ in range(length))

def initialize_population():
    return [random_text(TEXT_LENGTH) for _ in range(POP_SIZE)]
```

3.4 Оператор селекції

Використовується турнірна селекція з розміром турніру 3. Алгоритм:

1. Випадково обирається перший індивід
2. Додатково обираються ще 2 випадкові індивіди
3. Повертається індивід з найбільшою пристосованістю серед трьох

```python
def tournament_select(population, fitnesses, tournament_size=3):
    selected_idx = random.randrange(len(population))
    for _ in range(tournament_size - 1):
        i = random.randrange(len(population))
        if fitnesses[i] > fitnesses[selected_idx]:
            selected_idx = i
    return population[selected_idx]
```

3.5 Оператор кросоверу

Застосовується одноточковий кросовер:

1. Випадково обирається точка розрізу від 1 до $L-1$ (де L — довжина рядка)
2. Перша дитина: початок від першого батька + кінець від другого батька
3. Друга дитина: початок від другого батька + кінець від першого батька

```python
def one_point_crossover(parent1, parent2):
    L = len(parent1)
    if L < 2:
        return parent1, parent2
    cut = random.randint(1, L - 1)
    child1 = parent1[:cut] + parent2[cut:]
    child2 = parent2[:cut] + parent1[cut:]
    return child1, child2
```

3.6 Оператор мутації

Для кожного символу в рядку з ймовірністю `MUTATION_RATE` (0.02) символ замінюється на випадковий символ з алфавіту.

```python
def mutate(text):
    text_list = list(text)
    for i in range(len(text_list)):
        if random.random() < MUTATION_RATE:
            text_list[i] = random_char()
    return "".join(text_list)
```

3.7 Елітизм

На кожній ітерації 2 найкращі індивіди (елітні) автоматично переходять до наступного покоління без змін, що гарантує збереження найкращих знайдених рішень.

4 Алгоритм створення нового покоління

Процедура формування нового покоління:

1. **Оцінка:** Обчислюється пристосованість кожного індивіда в поточній популяції
2. **Сортування:** Популяція сортується за спаданням пристосованості
3. **Елітизм:** Перші `ELITE_COUNT` особин додаються до нової популяції
4. **Генерація нащадків:** До досягнення розміру `POP_SIZE`:
   - Вибираються два батьки за допомогою турнірної селекції
   - З ймовірністю `CROSSOVER_RATE` застосовується кросовер
   - До обох нащадків застосовується мутація
   - Нащадки додаються до нової популяції
5. **Заміна:** Нова популяція замінює стару

5 Критерії зупинки

Алгоритм зупиняється при виконанні однієї з умов:

- Знайдено точний збіг: $P(x, b) \leq 0.0$
- Досягнуто максимальну кількість поколінь: 20000

6 Результати роботи

Алгоритм виконується протягом максимум 20000 поколінь. На кожному поколінні виводиться інформація про:

- Номер покоління
- Мінімальна помилка P для найкращого індивіда
- Текст найкращого індивіда

Формат виведення:

```
Generation XXXX | best error P = Y.YYYYYY | best text = 'текст'
```

6.1 Перевірка результату

Для перевірки правильності знайденого рішення використовується дешифрування з відомими параметрами:

- Факторизація $n = 149 \times 223 = 33227$
- Обчислення $\phi(n) = (p - 1)(q - 1) = 148 \times 222 = 32856$
- Знаходження приватного експоненти d як модульного оберненого до e за модулем $\phi(n)$
- Дешифрування: $m = c^d \bmod n$ для кожного символу

```python
# Факторизація n
p, q = 149, 223
phi = (p - 1) * (q - 1)

# Знаходження d (розширений алгоритм Евкліда
def egcd(a, b):
    if b == 0:
        return (1, 0, a)
    x, y, g = egcd(b, a % b)
    return (y, x - (a // b) * y, g)

x, y, g = egcd(public_key, phi)
d = x % phi

# Дешифрування
plaintext_chars = [chr(pow(c, d, n)) for c in ciphertext]
plaintext = "".join(plaintext_chars)
```

7 Висновки

У даній роботі було реалізовано простий генетичний алгоритм для задачі апроксимації відкритого тексту за шифротекстом RSA. Алгоритм використовує класичні генетичні оператори:

- Турнірну селекцію для вибору батьків
- Одноточковий кросовер для рекомбінації генетичного матеріалу
- Точкову мутацію для підтримки різноманітності популяції
- Елітизм для збереження найкращих рішень

Функція пристосованості базується на квадратичній відстані між шифротекстами, що дозволяє ефективно направляти пошук до оптимального рішення.

Алгоритм демонструє типовий підхід еволюційних обчислень до задач комбінаторної оптимізації, де простір пошуку є дискретним і надзвичайно великим ($55^{42} \approx 10^{73}$ можливих комбінацій).