МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

Кафедра №22 «Кибернетика»

Направление подготовки 09.03.04 Программная инженерия

Расширенное содержание пояснительной записки

к учебно-исследовательской работе студента на тему:
«Разработка системы, реализующей генетический алгоритм и применение её к расположению вершин графа на плоскости»

Группа Б22-534

Студент _____ (подпись) Когановский Г.И. (ФИО)

Руководитель _____ (подпись) Короткова М.А. (ФИО)

Научный консультант _____ (подпись) _____ (ФИО)

Оценка руководителя _____ (0-30 баллов) Оценка консультанта _____ (0-30 баллов)

Итоговая оценка _____ (0-100 баллов) ECTS _____

Комиссия

Председатель _____ (подпись) _____ (ФИО)

_____ (подпись) _____ (ФИО)

_____ (подпись) _____ (ФИО)

_____ (подпись) _____ (ФИО)

Москва 2025

Реферат

Пояснительная записка содержит _ страниц. Количество источников – _. Количество рисунков – _. Количество таблиц - _.

Ключевые слова: генетические алгоритмы, многокритериальная оптимизация, теория графов, сравнительный анализ.

Целью данной работы является разработка системы, реализующей генетический алгоритм и применение её к расположению вершин графа на плоскости.

В первом разделе проводится исследование существующих методов и генетических алгоритмов (SGA, NSGA-II, SPEA2 и др.), применяемых для решения тестовых задач многокритериальной оптимизации, таких как задача о рюкзаке, задача коммивояжёра, поиск минимума сложных функций.

Второй раздел посвящён оценке эффективности выбранных методов и выдвижению гипотез. Также в разделе описываются модификации пространства решений поставленной задачи.

Третий раздел посвящён разработке требований к системе и алгоритмам, адаптированным к поставленной задаче размещения вершин графа на плоскости, на основе выдвинутых гипотез и результатов исследования. Также в разделе описывается проектирование программной системы и алгоритмов в соответствии с разработанными требованиями.

Четвертый раздел сфокусирован на описании программной реализации спроектированной системы и алгоритмов. Также в разделе описываются результаты тестирования разработанной системы.

В заключении приводятся задачи, решённые в процессе разработки системы, реализующей генетический алгоритм, и её применения к расположению вершин графа на плоскости. Также в этом разделе представлены перспективы дальнейшей работы над проектом.

Введение

Современные научные и технические задачи всё чаще требуют поиска оптимальных решений в условиях многокритериальности и высокой вычислительной сложности. Как отмечают авторы [1], большинство таких задач относятся к классу комбинаторных оптимизационных проблем, для которых характерно наличие множества альтернативных решений различного качества. Традиционные методы, основанные на полном или направленном переборе, сталкиваются с фундаментальными ограничениями: рост вычислительных ресурсов экспоненциально зависит от размерности задачи, а жесткие требования к математическим моделям сужают область их применимости [1, 2]. Это особенно актуально для задач, связанных с теорией графов, таких как размещение вершин на плоскости, где необходимо одновременно учитывать несколько критериев (например, минимизацию пересечений рёбер, равномерность распределения узлов и эстетическую ясность визуализации).

В этом контексте генетические алгоритмы (ГА) демонстрируют значительный потенциал как универсальный инструмент управляемого перебора. Их эволюционная природа, основанная на механизмах селекции, скрещивания и мутации, позволяет эффективно исследовать обширные пространства решений [2]. Однако эффективность ГА существенно зависит от выбора операторов, параметров адаптации и способов учёта множественности критериев. Модификации алгоритмов, такие как NSGA-II и SPEA2, предлагают решения для многокритериальных задач, но их применимость к специфическим проблемам, таким как визуализация графов, требует дополнительного исследования и адаптации.

Целью данной работы является разработка и анализ генетического алгоритма, ориентированного на решение задачи размещения вершин графа на плоскости с учётом нескольких критериев оптимизации.

1. Исследование существующих методов и генетических алгоритмов для решения задач многокритериальной оптимизации

1.1. Формулировка оптимизационных задач

Следующие оптимизационные задачи будут использованы в качестве тестовых. Они послужат основой для последующего сравнительного анализа методов.

1. Задача о расположении вершин графа на плоскости

   Задача заключается в назначении координат в 2-мерном пространстве вершинам невзвешенного, неориентированного графа с N вершинами и M ребрами таким образом, чтобы оптимизировать выбранные метрики. В качестве метрики предлагается взять количество пересечений рёбер.

   Качество работы метода предлагается оценивать, основываясь на значениях выбранных метрик для найденного решения и времени, затраченного компьютером на его поиск.

2. Задача о рюкзаке 0-1

   Для $N,\ D \in \mathbb{N}; N, D > 1$: дано N предметов, i-ый предмет имеет стоимость $p_i > 0$ и требуемые ресурсы $r_{ij} > 0, j \in [1, D-1]$. Необходимо выбрать из этих предметов такой набор, чтобы суммарная стоимость была максимальна, а суммарное количество каждого требуемого ресурса не превосходило заданных ограничений $C_j,\ j \in [1, D-1]$. Каждый предмет есть в единственном экземпляре.

   То есть, найти такой набор $(x_1, \ldots, x_N),\ x_i \in \{0, 1\}$, при котором $\sum_{i=1}^{N} r_{ij} x_i \le C_j,\ j \in [1, D-1]$ и $\sum_{i=1}^{N} p_i x_i$ максимально.

   Эффективность метода будет определяться максимальной суммарной стоимостью набора предметов, выбранного алгоритмом, а также временем, затраченным на поиск оптимального решения.

3. Симметричная задача о Коммивояжере

   Для данного полного взвешенного графа с N вершинами найти минимальный гамильтонов цикл. Каждой вершине в графе присвоены координаты в 2-мерном пространстве. Расстояния между вершинами вычисляются с помощью евклидовой метрики и, соответственно, симметричны.

   Оценка работы алгоритма характеризуется минимальной длиной гамильтонова цикла и временем вычислений. Для небольшого количества

городов (≤ 12) длину цикла предлагается нормировать по эталонному значению, найденному полным перебором всех перестановок вершин.

1.2. Описание и программная реализация выбранных методов решения тестовых задач

Следующие методы будут использованы для решения тестовых задач.

1. Динамическое программирование

Метод динамического программирования предлагается использовать для решения задачи о рюкзаке 0-1 при $D = 2$. 2-мерность означает, что каждый предмет обладает стоимостью и требует затрат только одного ресурса (масса).

Описание метода

Пусть $A(k,\ s)$ есть максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов, то есть $\{n_1, n_2, \ldots,\ n_k\}$, назовем этот набор допустимых предметов для $A(k,\ s)$.

$$A\left(k,\ 0\right) = 0$$

$$A(0,\ s) = 0$$

Найдем $A(k,\ s)$. Возможны 2 варианта.

Если предмет k не попал в рюкзак. Тогда $A(k,\ s)$ равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов $\{n_1, n_2, \ldots,\ n_{k-1}\}$, то есть $A(k,\ s) = A(k-1,\ s)$.

Если k попал в рюкзак. Тогда $A(k,\ s)$ равно максимальной стоимости рюкзака, где вес s уменьшаем на вес k-ого предмета и набор допустимых предметов $\left\{n_1, n_2, \ldots,\ n_{k-1}\right\}$ плюс стоимость k, то есть $A\left(k-1,\ s-r_{k1}\right) + p_k$.

$$A\left(k,\ s\right) = \begin{cases} A\left(k-1,\ s\right), & b_k = 0 \\ A\left(k-1, s-r_{k1}\right) + p_k, & b_k = 1 \end{cases}$$

То есть

$$A\left(k,\ s\right) = \max(A\left(k-1,\ s\right),\ A\left(k-1, s-r_{k1}\right) + p_k)$$

Стоимость искомого набора равна $A\left(N,\ C_1\right)$ так как нужно найти максимальную стоимость рюкзака, где все предметы допустимы и вместимость рюкзака C_1.

Программная реализация

```go
func (p *KnapsackProblem) AlgorithmicSolution() problems.AlgorithmicSolution {
    n := p.Params.ItemsNum
    capacity := p.Params.Constraints[0]

    weights := make([]int, n)
    values := make([]int, n)
    for i := range n {
        values[i] = p.Items[i].Value
        weights[i] = p.Items[i].Resources[0]
    }

    dp := make([][]int, n+1)
    for i := range dp {
        dp[i] = make([]int, capacity+1)
    }

    // solving
    for i := 1; i <= n; i++ {
        for w := 1; w <= capacity; w++ {
            if weights[i-1] <= w {
                include_item := values[i-1] + dp[i-1][w-weights[i-1]]
                exclude_item := dp[i-1][w]
                dp[i][w] = max(include_item, exclude_item)
            } else {
                dp[i][w] = dp[i-1][w]
            }
        }
    }

    // tracing solution
    selectedBits := make([]bool, n)
    for i, w := n, capacity; i > 0; i-- {
        if dp[i][w] != dp[i-1][w] {
            selectedBits[i-1] = true
            w -= weights[i-1]
        }
    }

    return problems.AlgorithmicSolution{
        Solution: &KnapsackSolution{problemParams: p.Params, items: p.Items, Bits:
selectedBits},
    }
}
```

2. Полный перебор

Метод полного перебора предлагается использовать для решения симметричной задачи Коммивояжёра для небольшого (≤ 12) количества вершин.

Описание метода

Метод генерирует все перестановки вершин в графе и для каждой из них вычисляет длину цикла, проходящего по ним в выбранном порядке, выбирая перестановку с минимальной длиной пути.

Программная реализация

```go
func (p *TSProblem) AlgorithmicSolution() problems.AlgorithmicSolution {
    cities := make([]int, p.Params.CitiesNum-1)
    for i := range p.Params.CitiesNum - 1 {
        cities[i] = i + 1
    }

    bestSolution := TSPSolution{problemParams: p.Params, cities: p.Cities,
VisitingOrder: cities}
    for _, order := range permutations(cities) {
        solution := TSPSolution{problemParams: p.Params, cities: p.Cities,
VisitingOrder: order}
        if solution.Fitness() > bestSolution.Fitness() {
            bestSolution = solution
        }
    }

    return problems.AlgorithmicSolution{
        Solution: &bestSolution,
    }
}
```

3. SGA (Simple Genetic Algorithm) [3]

   Описание метода

   SGA представляет собой базовую версию генетического алгоритма. Процесс начинается с формирования начальной популяции решений, после чего каждый кандидат оценивается по заранее определенной функции приспособленности. При отсутствии удовлетворяющего решения происходит переход к эволюционным операциям: отбору элиты, скрещиванию и мутации, что позволяет постепенно улучшать качество решений.

   Шаги алгоритма.

   1. Создаётся начальная популяция решений размером N.
   2. Оценивается качество каждого решения.

      Если найдено решение, удовлетворяющее требованиям, алгоритм завершается. В противном случае, выполняются следующие шаги.
   3. Выбирается «элита» - N_p решений из предыдущей популяции попадают в следующую без изменений.
   4. Создаётся пул размножения.
   5. Выполняется скрещивание.
   6. Выполняется мутация в решениях-потомках.
   7. Старые решения последнего поколения заменяются на вновь созданные.

      Переход к шагу (2).

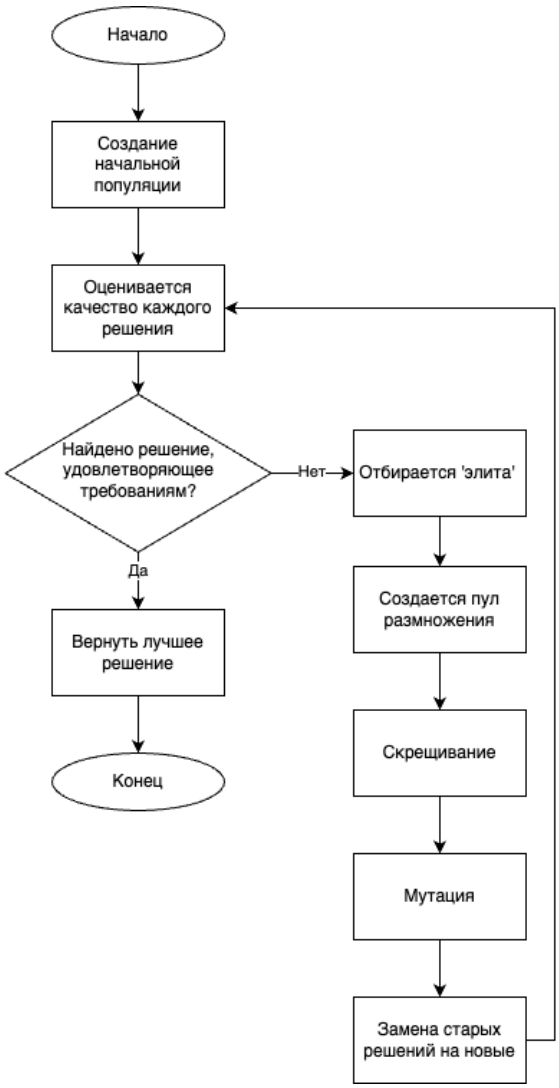

Рисунок 1 Блок-схема SGA

Программная реализация

```go
func (sga *SimpleGeneticAlgorithm) Evolve() {
    sga.evaluateGeneration()

    newPopulation := make([]problems.Solution, 0, sga.params.PopulationSize)

    // perform elitism
    newPopulation = append(newPopulation, sga.population[:sga.eliteSize]...)

    // generate rest of the population from mating pool
    for len(newPopulation) < sga.params.PopulationSize {
        p1Ind := rand.Intn(sga.matingPoolSize)
        p2Ind := rand.Intn(sga.matingPoolSize)
        if p1Ind == p2Ind {
            continue
        }
        parent1 := sga.population[p1Ind]
        parent2 := sga.population[p2Ind]
        children := parent1.Crossover(parent2)
        for i := range children {
            children[i] = children[i].Mutate(sga.params.MutationRate)
        }
        newPopulation = append(newPopulation, children...)
    }

    sga.population = newPopulation
}
```

4. SSGA (Steady State Genetic Algorithm) [4]

Описание метода

«Steady State» означает отсутствие поколений. Отличается от SGA тем, что вместо добавления решений-потомков в популяцию следующего поколения заменяет два старых решения на два лучших решения из группы двух родителей и двух их потомков, сохраняя размер популяции.

Шаги алгоритма.

1. Создаётся начальная популяция решений размером N.
2. Оценивается качество каждого решения.
3. Выбираются 2 решения-родителя без повторов.
4. Выполняется скрещивание и мутация, получается 2 потомка.
5. Если потомки повторны переход к шагу (3).
6. Оценивается качество потомков.
7. Если качество потомков выше качества худших решений, новые заменяют старые.
8. Если найдено решение, удовлетворяющее требованиям, алгоритм завершается. В противном случае, переход к шагу (3).

Рисунок 2 Блок-схема SSGA

5. NSGA-II (Non-dominated Sorting Genetic Algorithm II) [5]

Описание метода

NSGA-II является одним из самых известных и эффективных алгоритмов для решения многокритериальных оптимизационных задач. Его ключевыми особенностями являются быстрое ранжирование по принципу недоминирования и механизм сохранения разнообразия в популяции.

Шаги алгоритма.

1. Инициализация

   Создается начальная популяция P_0, содержащая N индивидуумов, случайно распределённых по пространству решений.

2. Объединение популяций

На каждом поколении формируется объединённая популяция $R = P \cup Q$, где P – текущая популяция, а Q – порожденные потомки.

3. Сортировка по недоминированию

   Решения сортируются на несколько фронтов. Первый фронт включает недоминированные решения, второй – решения, доминируемые только элементами первого фронта, и так далее.

4. Расчет расстояния скученности

   Для каждого решения вычисляется показатель «crowding distance», характеризующий плотность расположения решений в окрестности. Это позволяет сохранить разнообразие решений, отбирая из более «разрежённых» областей.

5. Отбор

   Формируется новая популяция путём последовательного добавления фронтов из объединённой популяции до достижения требуемого размера. Если добавление целого фронта приводит к превышению размера, то решения из этого фронта отбираются с наибольшим расстоянием скученности.

6. Применение генетических операторов

   Для создания новой популяции применяются стандартные операторы генетических алгоритмов – отбор, скрещивание и мутация.

7. Повторение

   Шаги 2–6 повторяются до выполнения условия остановки (например, достижение заданного числа поколений или стабильности значений метрик).

6. SPEA2 (Strength Pareto Evolutionary Algorithm 2) [6]

Описание метода

SPEA2 является улучшенной версией оригинального алгоритма SPEA и предназначен для эффективного решения многокритериальных оптимизационных задач. Одной из ключевых особенностей SPEA2 является использование внешнего архива для сохранения лучших недоминированных решений, а также детальная оценка приспособленности с учётом как доминирования, так и плотности расположения решений.

Шаги алгоритма.

1. Инициализация

Создается начальная популяция P_0 и пустой внешний архив A.

2. Объединение популяций

   На каждом поколении объединяются текущая популяция P и архив A, формируя совокупное множество $R = P \cup A$.

3. Оценка приспособленности

   Каждому решению в R присваивается значение силы, которое определяется количеством решений, доминируемых данным решением. Помимо этого, рассчитывается мера плотности – зачастую используется расстояние до k-го ближайшего соседа. Итоговая приспособленность решения зависит от его доминирующей способности и локальной плотности, что позволяет учитывать как качество, так и разнообразие решений.

4. Обновление архива

   Из объединённого множества R отбираются недоминированные решения для формирования нового архива. Если число решений в архиве превышает заданное ограничение, то применяется процедура редукции с учётом плотности – сохраняются решения, находящиеся в более разрежённых областях пространства решений.

5. Отбор родителей

   На основании рассчитанных значений приспособленности производится отбор родителей (обычно с помощью турнирного отбора) для генетических операций.

6. Генетические операторы

   Применяются операторы скрещивания и мутации для формирования новой популяции P следующего поколения.

7. Повторение

   Процесс (объединение, оценка, обновление архива и генетические операции) повторяется до достижения условия остановки (например, по числу поколений или по сходимости результатов).

1.3. Выводы

1. Сформулированы оптимизационные задачи.
2. Описаны и реализованы методы решения тестовых задач.

1.4. Цели и задачи на УИР

Основной задачей данной работы является разработка системы, реализующей генетический алгоритм и применение её к расположению вершин графа на плоскости.

Для решения данной задачи необходимо решить следующие подзадачи.

1. Формулировка оптимизационных задач.
2. Описание и программная реализация выбранных методов решения тестовых задач.
3. Оценка эффективности выбранных методов и выдвижение гипотез.
4. Модификация пространства решений задачи о расположении вершин графа на плоскости.
5. Разработка требований к создаваемой системе.
6. Проектирование системы на основе требований.
7. Программная реализация системы и алгоритмов.
8. Тестирование разработанной системы и алгоритмов.

2. Разработка требований и проектирование системы, адаптированной к поставленной задаче

В данном разделе будут разработаны требования к системе и алгоритмам, адаптированным к поставленной задаче. Также в разделе будет спроектирована программная система и алгоритмы в соответствии с требованиями.

2.1. Оценка эффективности выбранных методов и выдвижение гипотез

Будет проведена оценка эффективности выбранных методов и выдвинуты гипотезы.

2.2. Модификация пространства решений задачи о расположении вершин графа на плоскости

Будут разработаны модификации пространства решений задачи о расположении вершин графа на плоскости.

2.3. Выводы

1. Будет проведена оценка эффективности выбранных методов и выдвинуты гипотезы.
2. Будут разработаны модификации пространства решений поставленной задачи.

3. Программная реализация спроектированной системы

В данном разделе будет описана программная реализации спроектированной системы и алгоритмов.

3.1. Разработка требований к создаваемой системе

Будут сформулированы требования к создаваемой системе.

3.2. Проектирование системы на основе требований

Будут спроектирована и описана система, реализующая новые ГА, удовлетворяющие разработанным требованиям. Разработаны схемы, описывающие работу новых ГА (блок-схемы, UML-диаграммы).

3.3. Выводы

1. Будут разработаны требования к системе.
2. Система будет спроектирована на основе требований.

4. Оценка эффективности системы и анализ результатов

В данном разделе будет представлена оценка эффективности разработанной системы и алгоритмов. Также в разделе будут проанализированы полученные результаты.

4.1. Программная реализация системы и алгоритмов

Будут описаны этапы кодирования: реализация генетических операторов, интеграция с графическими библиотеками, настройка параметров алгоритма. Приведены примеры фрагментов кода для ключевых модулей.

4.2. Тестирование разработанной системы и алгоритмов

Сравнение эффективности разработанного ГА с существующими подходами. Выявлены зависимости качества решений от параметров алгоритма (размер популяции, вероятность мутации и т.п.).

4.3. Выводы

1. Будет разработана программная реализация системы и алгоритмов.
2. Будет протестирована разработанная система.

Заключение

В результате выполнения данной работы ожидается разработка эффективной системы, реализующей генетический алгоритм для оптимизации расположения вершин графа на плоскости.

Следующие задания будут решены для достижения данной цели.

1. Выбраны и сформулированы оптимизационные задачи.
2. Выбраны, описаны и реализованы методы решения тестовых задач.
3. Будут проанализированы полученные результаты и выдвинуты гипотезы.
4. Будут разработаны требования к системе.
5. Система будет спроектирована на основе требований.
6. Будет разработана программная реализация системы и алгоритмов.
7. Будет протестирована разработанная система.

В дальнейшем планируется обобщение разработанных алгоритмов для решения более

широкого класса задач, а также проведение дополнительного анализа параметров генетических алгоритмов с целью повышения качества получаемых решений.

Список литературы

1. Гладков Л. А., Курейчик В. В., Курейчик В. М. Генетические алгоритмы, учебник, Москва: Физматлит, 2010. 368с.
2. Цой Ю. Р., Спицын В. Г. Исследование генетического алгоритма с динамически изменяемым размером популяции //Труды Международной научно-технической конференции “Интеллектуальные системы (IEEE AIS’05)”. Научное издание. М.: Изд. физико-математической литературы. – 2005. – С. 241-246.
3. Vose M. D. The simple genetic algorithm: foundations and theory. – MIT press, 1999.
4. Agapie A., Wright A. H. Theoretical analysis of steady state genetic algorithms // Applications of mathematics. – 2014. – Т. 59. – №. 5. – С. 509-525.
5. Deb K. et al. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II //International conference on parallel problem solving from nature. – Berlin, Heidelberg : Springer Berlin Heidelberg, 2000. – С. 849-858.
6. Zitzler E., Laumanns M., Thiele L. SPEA2: Improving the strength Pareto evolutionary algorithm //TIK report. – 2001. – Т. 103.
7. Акопов А. С. и др. Разработка адаптивного генетического оптимизационного алгоритма с использованием методов агентного моделирования //Информационные технологии. – 2018. – Т. 24. – №. 5. – С. 321-329.
8. Акопов А. С. и др. Многоагентный генетический алгоритм на основе нечёткой кластеризации при решении многокритериальных задач //Искусственные общества. – 2020. – Т. 15. – №. 2. – С. 1-1
9. Чеканин В. А., Куликова М. Ю. Адаптивная настройка параметров генетического алгоритма //Вестник МГТУ Станкин. – 2017. – №. 3. – С. 85-89.
10. Акопов А. С. и др. Система поддержки принятия решений для рационального озеленения города на примере г. Ереван, Республика Армения //Программная инженерия. – 2019. – Т. 10. – №. 2. – С. 87-96