

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования"МИРЭА - Российский технологический университет"РТУ МИРЭА |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА № 3**

**по дисциплине**

**«Системный анализ данных в системе поддержки принятия решений»**

**Роевой алгоритм**

Студент группы: ИКБО-04-21 Даурбеков М.И. *(Ф. И.О. студента)*

Руководитель Железняк Л.М.

*(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc144886867)

[1 АЛГОРИТМ ОТЖИГА 4](#_Toc144886868)

[1.1 Постановка задачи 4](#_Toc144886869)

[1.2 Теоретический раздел 4](#_Toc144886870)

[1.3 Алгоритм отжига 4](#_Toc144886871)

[ЗАКЛЮЧЕНИЕ 7](#_Toc144886872)

[СПИСОКИНФОРМАЦИОННЫХИСТОЧНИКОВ 8](#_Toc144886873)

[ПРИЛОЖЕНИЯ 9](#_Toc144886874)

# ВВЕДЕНИЕ

Алгоритм отжига - метод оптимизации, вдохновленный физическим процессом отжига и моделированием поведения металла при охлаждении. Основная идея этого алгоритма заключается в имитации процесса отжига и случайного перемещения по пространству решений с целью найти глобальный оптимум задачи оптимизации. Алгоритм отжига начинается с некоторого начального решения, которое затем постепенно модифицируется в процессе итераций. Эти модификации выполняются путем принятия новых решений, которые могут быть либо лучше, либо худшими, чем текущее решение. Однако, по мере продвижения в итерациях, вероятность принятия худшего решения (с целью избежания застревания в локальных оптимумах) уменьшается со временем, подобно охлаждению металла. Этот процесс осуществляется с помощью функции охлаждения, которая постепенно уменьшает температуру системы. Алгоритм отжига является эффективным инструментом для решения задач оптимизации в различных областях, таких как раскрой материалов, планирование расписания, поиск минимального пути и многих других.

.

# 1 РОЕВОЙ АЛГОРИТМ

## 1.1 Постановка задачи

Предметная область: поиск глобального минимума заданной функции.

В данной практической работе необходимо изучить и разработать метод, позволяющий определить глобальный минимум функции, а также направит все частицы в координатной плоскости к глобальному минимум.

## 1.2 Теоретический раздел

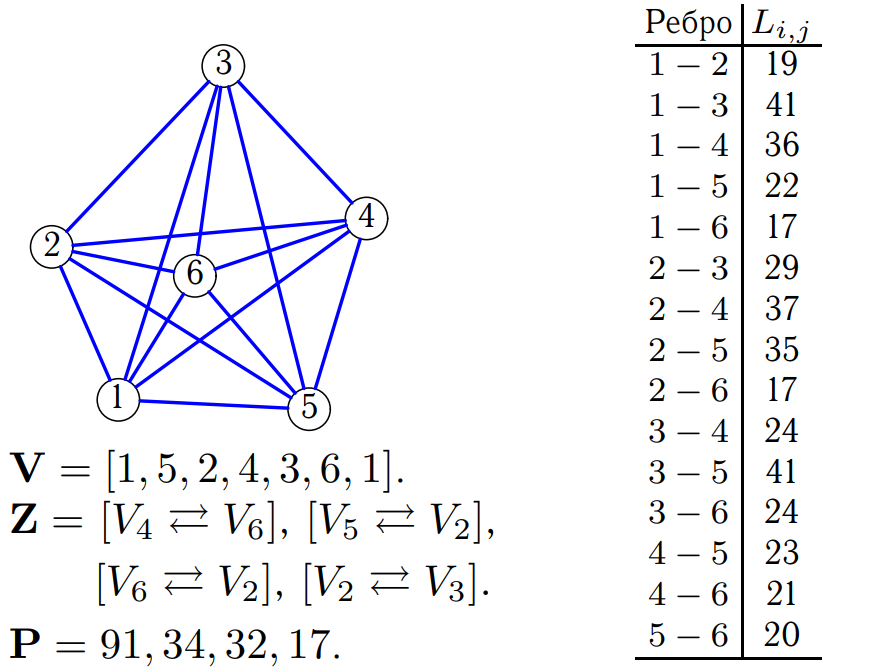
РА использует рой частиц, где каждая частица представляет потенциальное решение проблемы. Поведение частицы в гиперпространстве поиска решения все время подстраивается в соответствии со своим опытом и опытом своих соседей. Кроме этого, каждая частица помнит свою лучшую позицию с достигнутым локальным лучшим значением целевой (фитнесс-) функции и знает наилучшую позицию частиц - своих соседей, где достигнут глобальный на текущий момент оптимум. В процессе поиска частицы роя обмениваются информацией о достигнутых лучших результатах и изменяют свои позиции и скорости по определенным правилам на основе имеющейся на текущий момент информации о локальных и глобальных достижениях. При этом глобальный лучший результат известен всем частицам и немедленно корректируется в том случае, когда некоторая частица роя находит лучшую позицию с результатом, превосходящим текущий глобальный оптимум. Каждая частица сохраняет значения координат своей траектории с соответствующими лучшими значениями целевой функции, которые обозначим , которая отражает когнитивную компоненту. Аналогично значение глобального оптимума, достигнутого частицами роя, будем обозначать , которое отражает социальную компоненту. Таким образом, каждая частица роя подчиняется достаточно простым правилам поведения (изложенным ниже формально), которые учитывают локальный успех каждой особи и глобальный оптимум всех особей (или некоторого множества соседей) роя.

Каждая i-я частица характеризуется в момент времени t своей позицией в гиперпространстве и скоростью движения . Позиция частицы изменяется в соответствии со следующей формулой:

Вектор скорости управляет процессом поиска решения и его компоненты определяются с учетом когнитивной и социальной составляющей следующим образом:

## 1.3 Алгоритм отжига

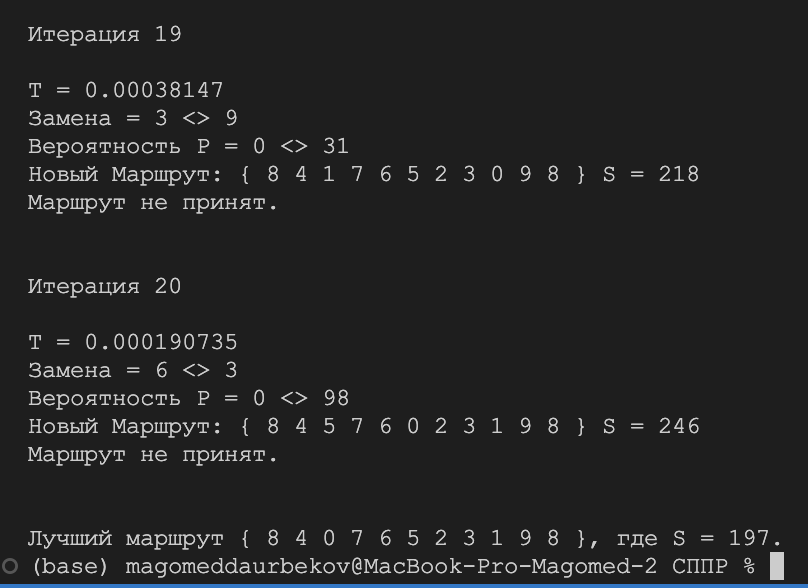
Исходные данные представлены на Рисунке 1. Даны исходные данные, где V – начальный маршрут, Z – перестановка данных вершин между собой, P – случайные числа. Первоначальная температура равна 100 градусам. Снижение происходит по закону . Количество итераций 100, максимальное количество 720 (так как у нас 6 городов которые можно переставлять).



**Рисунок 1 – Исходные данные**

При первой итерации найдём длину первоначального маршрута [1, 5, 2, 4, 3, 6, 1], сумма будет равна S0 =159. Поменяем местами первый список вершин (V4 и V6) и получим [1, 5, 2, 6, 3, 4, 1]. Сумма S1 будет равна 158. Так как у нас следующий маршрут лучше, чем старый, то понижаем температуру на 0.5. Переходим к следующей итерации (V5 и V2). Маршрут стал [1, 3, 2, 6, 5, 4, 1], Его сумма равна S2 = 166, видим, что маршрут стал длиннее. Из-за удлинения маршрута найдём приращение . Если приращение отрицательное, то длина уменьшилась (решение улучшилось) и после этого понижаем температуру. Если приращение положительное, то маршрут длиннее. Брать этот маршрут в качестве очередного в процедуре поиска минимума или не брать, зависит от вероятности перехода, вычисленной по формуле и от выпавшего в генераторе случайных чисел значения P. Если число, получившееся в формуле больше, чем число из генератора случайных чисел, то мы оставляем данный маршрут, иначе мы отбрасываем данный вариант. В нашем случае, мы получили Pk = 85,21. Из исходных данных мы видим, что 85,21 меньше, чем 91. Это значит, что мы не выбираем данный маршрут Результат работы программы представлен на Рисунке 2.

На итерации 100 длинна маршрута будет ровняться 127, что является оптимальным решением.



**Рисунок 2 – Результат работы программы**

# ЗАКЛЮЧЕНИЕ

В процессе выполнения практической работы был изучен и разработан метод оптимального выбора вершин. В заключение, алгоритм отжига является мощным инструментом оптимизации, который может быть успешно применен для разнообразных задач. За счет своей гибкости и способности исследовать пространство решений, он позволяет найти глобальный оптимум, обойдя локальные экстремумы. Одним из ключевых преимуществ алгоритма отжига является его способность принимать худшие решения на начальных стадиях и постепенно сходиться к лучшим решениям с учетом функции охлаждения. Это помогает избежать застревания в локальных оптимумах и повышает вероятность обнаружения глобального оптимума. Тем не менее, эффективное применение алгоритма отжига требует правильной настройки параметров, таких как начальная температура, функция охлаждения и число итераций. Экспериментирование и адаптация этих параметров в соответствии с конкретной задачей являются ключевыми факторами успеха.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Болотова Л. С. Многокритериальная оптимизация. Болотова Л. С., Сорокин А. Б. [Электронный ресурс] / Метод. указания по вып. курсовой работы — М.: МИРЭА, 2015.
2. Сорокин А. Б. Методы оптимизации: гибридные генетические алгоритмы. Сорокин А. Б. [Электронный ресурс] / Учебно-метод. пособие — М.: МИРЭА, 2016.
3. Сорокин А. Б. Линейное программирование: практикум. Сорокин А. Б., Бражникова Е. В., Платонова О. В. [Электронный ресурс] / Учебно-метод. пособие — М.: МИРЭА, 2017.

# ПРИЛОЖЕНИЯ

Приложение А – Код практической работы № 2

**Приложение А**

*Листинг 5 - Код практической работы № 2*

|  |
| --- |
| #include <algorithm>  #include <random>  #include <iostream>  #include <vector>  #include <numeric>  #include <ctime>  struct SwapIter { int from; int to; };  double rd() { return double(rand()) / RAND\_MAX; }  std::vector<int> SetupPorbabilities(int k)  {  std::vector<int> probabilities(k);  for(int i = 0; i < k; i++)  probabilities[i] = (rd() \* 100);  return probabilities;  }  std::vector<SwapIter> SetupSwapTable(int n, int k)  {  std::vector<SwapIter> swap\_table(k);  for(int i = 0; i < k; i++)  swap\_table[i] = {1 + (rand() % (n - 2)), 1 + (rand() % (n - 2))};  return swap\_table;  }  std::vector<std::vector<int>> SetupEdges(int n)  {  std::vector<std::vector<int>> l(n);  for(int i = 0; i < n; i++)  {  l[i].reserve(n);  for(int j = 0; j < n; j++)  {  if(j < i)  {  l[i][j] = l[j][i];  continue;  }  else if(j == i)  {  l[i][j] = 0;  continue;  }  l[i][j] = 50 \* (rd() + 0.1);  }  }  return l;  }  int f(const std::vector<int>& path, const std::vector<std::vector<int>> &l)  {  int res = 0;  for(int i = 1; i < path.size(); i++)  res += l[path[i - 1]][path[i]];  return res;  }  int main()  {  system("clear");  //srand(time(0));  const int n = 10;  const int k = 20;  const double y = 0.5;  double t = 100.0;  std::vector<std::vector<int>> l = SetupEdges(n);  std::vector<int> probabilities = SetupPorbabilities(k);  std::vector<SwapIter> swap\_table = SetupSwapTable(n, k);  std::vector<int> v(n); std::iota(v.begin(), v.end(), 0);  std::shuffle(v.begin(), v.end(), std::default\_random\_engine(time(0)));  v.push\_back(v[0]);  std::cout << "\nДлины граней:\n";  for(int i = 0; i < n; i++)  for(int j = i + 1; j < n; j++)  std::cout << i + 1 << " -> " << j + 1 << " = " << l[i][j] << "\n";  std::cout << "\nВероятности: {";  for(int prob : probabilities)  std::cout << " " << prob;  std::cout << " }\n";  std::cout << "\nЗамены:\n";  for(const auto& iter : swap\_table)  std::cout << iter.from + 1 << " <> " << iter.to + 1 << "\n";  int best\_s = f(v, l);  std::cout << "\nНачальный Маршрут: {";  for(auto i : v)  std::cout << " " << i;  std::cout << " } S = " << best\_s << "\n";  for(int i = 0; i < k; i++)  {  std::cout << "\nИтерация " << i + 1 << "\n\n";  std::cout << "T = " << t << "\n";    std::vector<int> u = v;  SwapIter swap\_iter = swap\_table[i];  std::cout << "Замена = " << swap\_iter.from + 1 << " <> " << swap\_iter.to + 1 << "\n";  std::swap(u[swap\_iter.from], u[swap\_iter.to]);  int s = f(u, l);  double ds = s - best\_s;  double p = 100.0 \* exp(-ds / t);  std::cout << "Вероятность P = " << p << " <> " << probabilities[i] << "\n";  std::cout << "Новый Маршрут: {";  for(auto idx : u)  std::cout << " " << idx;  std::cout << " } S = " << s << "\n";  if(ds < 0 || p > probabilities[i])  {  best\_s = s;  v = u;  if(ds < 0)  std::cout << "Маршрут принят т.к. dS = " << ds << " < 0.\n\n";  else  std::cout << "Маршрут принят т.к. P = " << p << " > " << probabilities[i] <<".\n\n";  }  else  std::cout << "Маршрут не принят.\n\n";  t \*= y;  }  std::cout << "\nЛучший маршрут {";  for(auto idx : v)  std::cout << " " << idx;  std::cout << " }, где S = " << best\_s << ".\n";  return 0;  } |