Московский авиационный институт

(национальный исследовательский университет)

Отчёт

по курсу «Математическое программирование и оптимальное управление»

Тема: «Решение транспортной задачи»

Выполнил: студент группы

М7О-407С-20

Понкращенков Д.Б.

Принял:

Барчев Н.Б.

1. Содержательная постановка оптимизационной задачи

Система обороны располагает несколькими территориально разнесенными стартовыми площадками зенитно-ракетных комплексов (ЗРК). Каждый ЗРК располагает определенным числом зенитно-управляемых ракет (ЗУР). В налете на систему обороны участвует определенное число бомбардировщиков, являющихся целями для системы обороны. Система обороны должна решить задачу целераспределения, то есть определить, какие ЗУР направить на поражение каких целей. Решение указанной задачи разбивается на два этапа: распределение ЗУР на цели по количеству (определение общего количества ЗУР со всех стартовых площадок, выделенных для поражения конкретной цели) и задачу назначения (определение количества ЗУР с определенных стартовых площадок, назначенных для поражения конкретных целей). В определенный момент системой обороны решена задача целераспределения по количеству. После ее решения получена дополнительная информация о том, **что некоторые ЗРК выведены из строя.** Необходимо, не повторяя первый этап задачи целераспределения, решить задачу назначения так, чтобы минимизировать суммарное полетное время всех оставшихся ЗУР к точкам перехвата.

1. Формализованная постановка оптимизационной задачи и ее характеристика

m и n – количество ЗУР и бомбардировщиков.

С[i][j] – временные затраты на полет ЗУР до точек перехвата.

где: i – номер ЗУР и j – номер бомбардировщика.

A[i] – количество оставшихся ЗУР,

B[j] – количество оставшихся бомбардировщиков.

X[i][j] – переменная распределения, 1 – если ЗУРi назначена бомбардировщикуj

Целевая функция имеет вид:

Ограничения:

Тип задачи – закрытый, т.к. в условии не сказано, что после повреждений количества ЗУР недостаточно для поражения всех целей, т.е.:

3. Обоснование перехода к Т-задаче и ее характеристика

Переход к транспортной задаче имеет смысл, если нужно учитывать цену (вес) распределения ресурсов для выполнения поставленной задачи.

Характеристики:

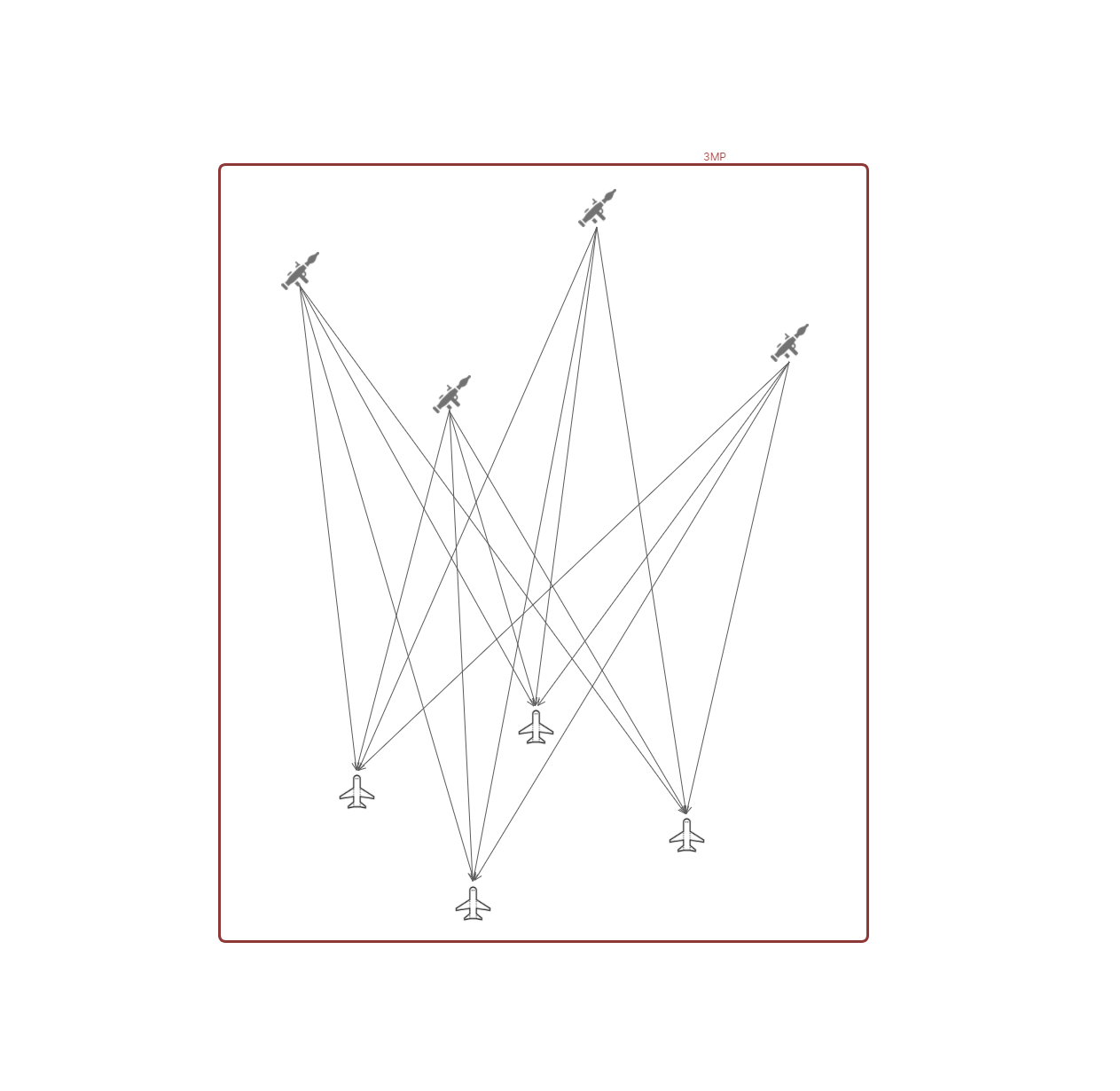
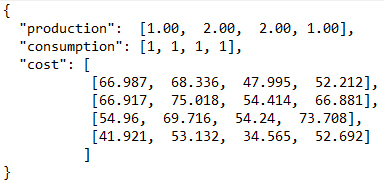
* **Источники и пункты назначения:** В Т-задаче есть набор источников (обычно обозначаются как "производство") и набор пунктов назначения (обычно обозначаются как "потребители"). Это производство может поставлять ресурсы в разные потребители.
* **Объемы поставок:** для каждой пары источник-пункт назначения определен объем, который должен быть доставлен из источника в пункт назначения. Этот объем может быть представлен в массиве, где каждый элемент описывает количество ресурсов, которые должны быть перевезены из пунктов производства в пункты потребления
* **Затраты на перевозку:** Каждая перевозка между производством и потреблением связана с определенными затратами. Затраты могут быть определены в виде матрицы, где каждый элемент представляет стоимость перевозки из источника в пункт назначения.
* **Цель:** Целью Т-задачи является минимизация общих затрат на перевозку чего-либо из пунктов производства в пункты потребления. Это аналогично цели задачи назначения, где цель - минимизировать суммарные затраты на выполнение работ.
* **Ограничения:** задача может быть, как открытой, так и закрытой. Закрытой задачей является та, у которой сумма товара в пунктах потребления равна сумме товара в пункте производства.

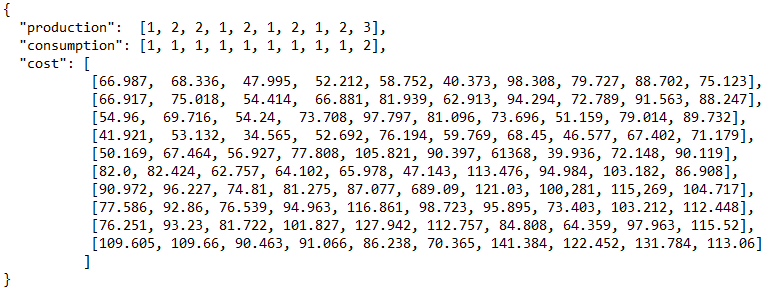
Открытой задачей является та задача, в которой по условию:

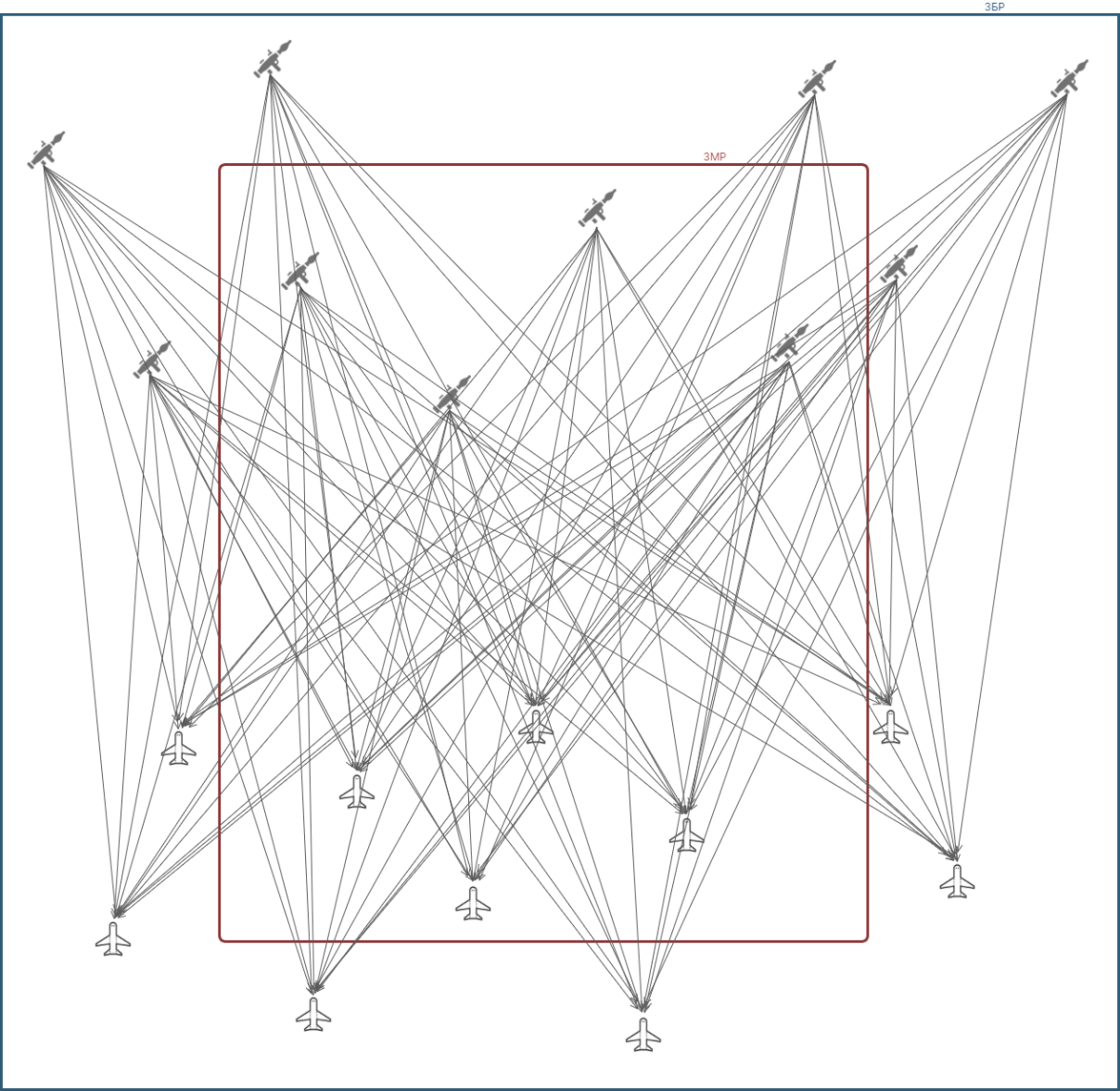
* + На 1 потребителя больше – задача с недопоставкой
  + На 1 производство больше – задача с перепроизводством

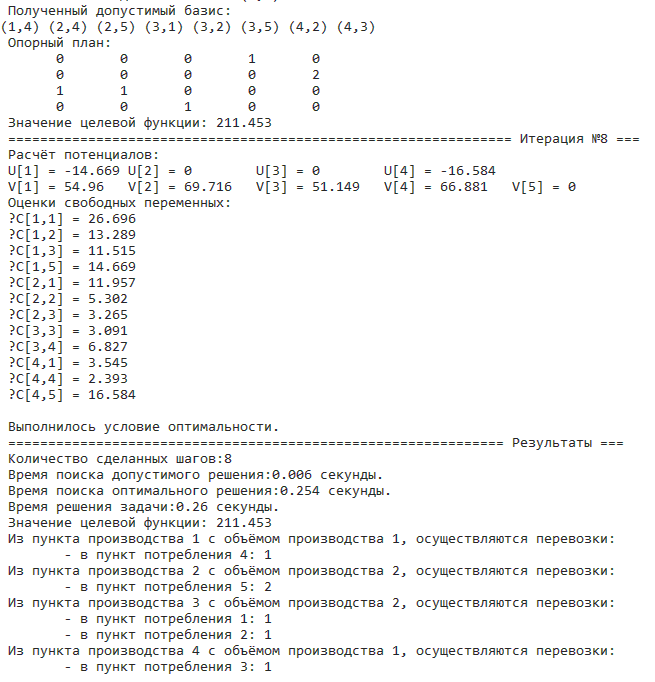
Каждая открытая задача сводится к закрытой путем добавления фиктивных пунктов потребления или производства и штрафных стоимостей в матрице затрат в зависимости от условий задачи.

4. Исходные данные для проведения вычислительных экспериментов

ЗМР:

ЗБР:



5. Результаты вычислительных экспериментов

|  |  |  |
| --- | --- | --- |
| Кол-во шагов АПОР/ Время работы АПДР/ Время работы всей программы | | |
|  | ММЭ | АСЗУ |
| ЗМР [4x4] | |  |  | | --- | --- | | [3]; [ 0.004 с]; [ 0.089 с] | [8]; [0.007 с]; [0.158 с] | | [8]; [ 0.007 c]; [ 0.158 c] |
| ЗБР [10x10] | [8]; [ 0.018 с]; [0.019 с] | [19]; [0.018 c]; [ 0.019 c] |

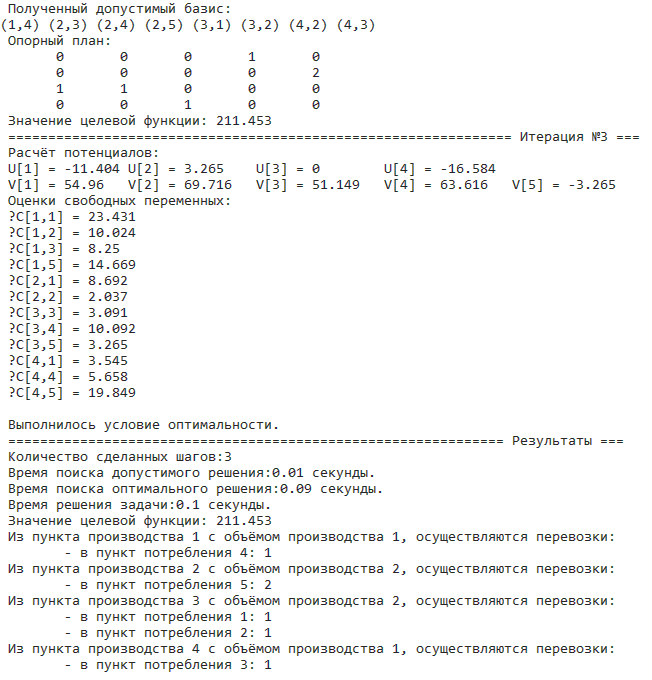
Рис 1. ЗМР АСЗУ

Рис. 2. ЗМР ММЭ

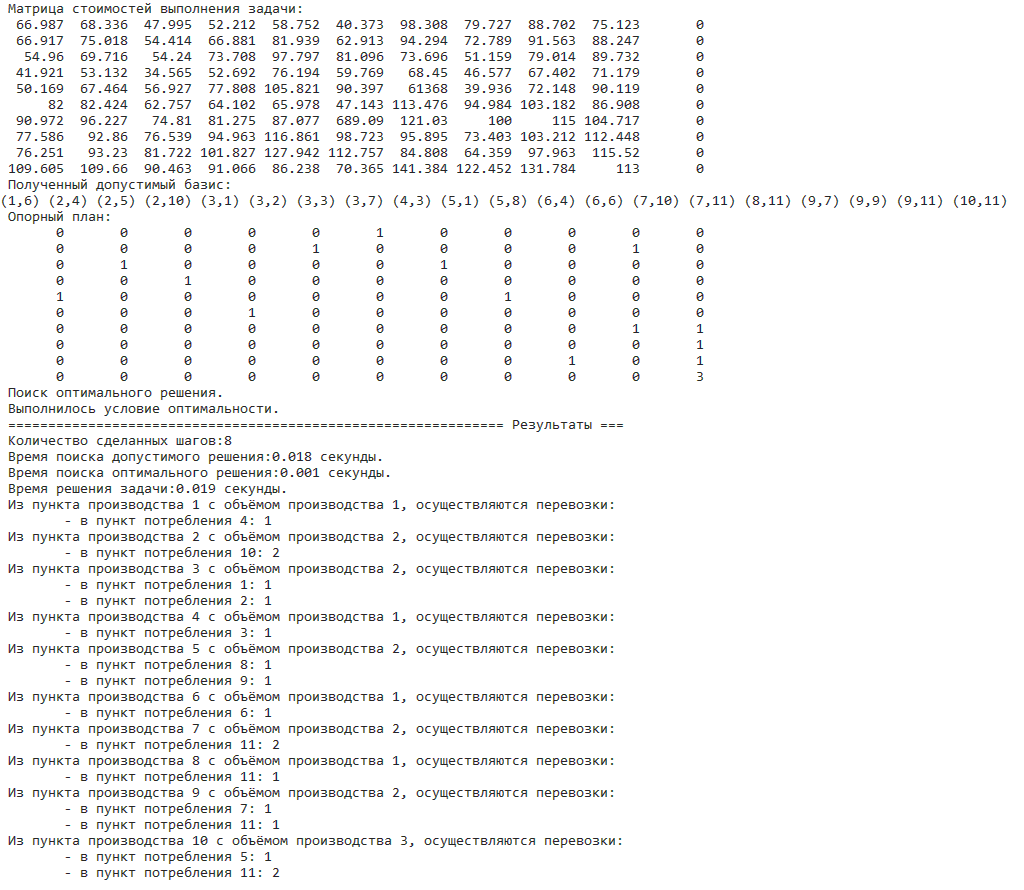
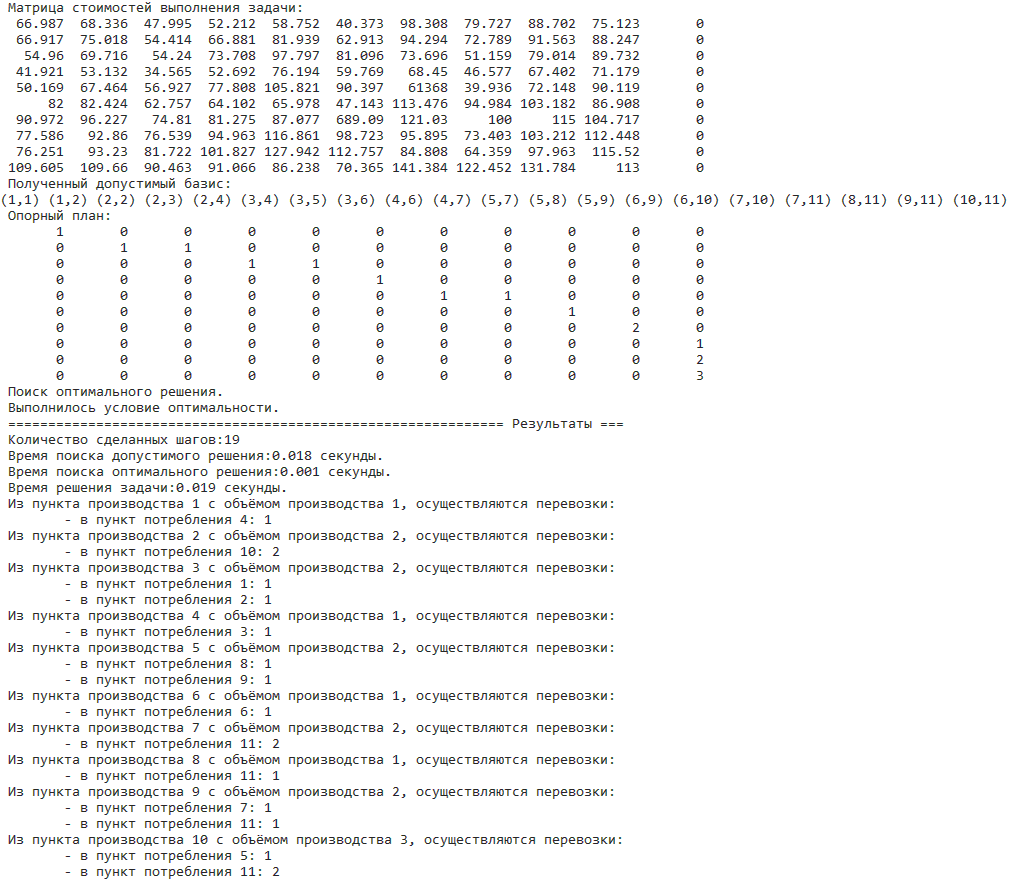
Рис 3. ЗБР ММЭ

Рис 4. ЗБР АСЗУ

6. Анализ результатов вычислительных экспериментов

* **по влиянию варианта АПДР на сходимость АПОР (по количеству итераций последнего);**В данном случае, ММЭ имеет меньше кол-во шагов, в обоих случая размерности задачи, на АПОР, возможно, это связано с расположением найденных базисных элементов при помощи АПДР, т.е. ММЭ смог найти лучшее расположение элементов для дальнейшего АПОР по сравнению с АСЗУ.
* **по влиянию размерности Т-задачи на относительную эффективность применения ММЭ;**

В данном анализе, размерность Т-задачи имеет два уровня: ЗМР и ЗБР.  
ММЭ показывает лучшую относительную эффективность в количестве итераций при равной размерности задачи, по сравнению с АСЗУ. При ЗМР разброс между ММЭ и АСЗУ составил 6 шагов, а при ЗБР - 11 шагов (8 против 19, что почти в два раза лучше).

* **по влиянию размерности Т-задачи на относительную сходимость АПОР, время работы АПДР и решения всей задачи применения ММЭ;**

Общий анализ показывает, что с увеличение размерности Т-задачи, кол-во шагов АПОР увеличивается, и время работы АПДР тоже увеличивается. Это обусловлено тем, что более сложные Т-задачи требуют больше итераций для АПОР и вычислительная и временная сложность растет.

7. Выводы.

Можно сделать вывод, что ММЭ является более эффективным методом, скорость сходимости выше, за счёт более оптимального опорного плана, который на ЗМР не редко может выдать сразу оптимальный результат.

Однако, текущая реализация показывает, что по времени в пределах [с] оба метода равны, если запускать программу в режиме без трассировки.

8. Листинг

#include "pch.h"

#include "TransportTask.h"

//--------------------------------------------------------------------------------||

std::vector<cells> TransportTask::Basis;

std::vector<double> TransportTask::LProduction;

std::vector<double> TransportTask::LConsumption;

std::vector<double> TransportTask::LPenalty;

int TransportTask::CountProduct = 0;

int TransportTask::CountConsumption = 0;

int TransportTask::indexPotential = 0;

int TransportTask::dummyConsum = -1;

int TransportTask::dummyProd = -1;

bool TransportTask::isPrint = true;

//--------------------------------------------------------------------------------||

void TransportTask::Calc(std::string path) {

ReadInput(path);

PrintInputs();

if (Correction()) {

PrintInputs();

}

int Tstart = clock();

//ANWA();

MME();

PrintBasis();

CheckBasis();

std::cout << "\n Поиск оптимального решения.";

int Tstart2 = clock();

int iter = 0, MaxIter = 100;

while (iter++ < MaxIter) {

if (isPrint) {

std::cout << "\n ===============================================================";

std::cout << " Итерация №" << iter << " ===";

std::cout << "\n Расчёт потенциалов:\n ";

}

potential potent = DefinePotential();

#pragma region --- ВЫВОД ПОТЕНЦИАЛОВ ---

if (isPrint) {

int i = 0;

for (auto x : \*potent.first) {

std::cout << "U[" << ++i << "] = " << x << '\t';

}

i = 0;

std::cout << "\n ";

for (auto x : \*potent.second) {

std::cout << "V[" << ++i << "] = " << x << '\t';

}

}

#pragma endregion

if (!CheckPotential(\*potent.first, \*potent.second)) {

if (isPrint) {

std::cout << "\n Условие оптимальности не выполняется";

std::cout << "\n В базис вводится X(" << indexPotential / CountConsumption + 1 << ","

<< indexPotential % CountConsumption + 1 << ")";

}

NewBasis();

if (isPrint)

PrintBasis();

CheckBasis();

}

else {

std::cout << "\n Выполнилось условие оптимальности.";

break;

}

}

int TEnd = clock();

std::cout << "\n ============================================================== Результаты ===";

std::cout << "\n Количество сделанных шагов:" << iter;

std::cout << "\n Время поиска допустимого решения:" << double(Tstart2 - Tstart) / 1000 << " секунды.";

std::cout << "\n Время поиска оптимального решения:" << double(TEnd - Tstart2) / 1000 << " секунды.";

std::cout << "\n Время решения задачи:" << double(TEnd - Tstart) / 1000 << " секунды.";

CheckBasis();

int i = 1;

int j = 1;

int jj = 1;

for (auto x : LProduction) {

std::cout << "\n Из пункта производства " << i << " с объёмом производства " << x << ", осуществляются перевозки:";

jj = 0;//(i - 1) \* CountConsumption;

for (auto b : Basis) {

int row = jj / CountConsumption;

if (row + 1 == i && b.getValue() != 0) {

std::cout << "\n\t- в пункт потребления " << jj % CountConsumption + 1 << ": " << b.getValue();

}

jj++;

}

i++;

}

}

//--------------------------------------------------------------------------------||

void TransportTask::ReadInput(std::string path) {

std::ifstream ifile(path);

if (ifile.is\_open()) {

std::string buff;

std::cout << "\n Считывание входного файла...";

while (ifile.eof(), ifile >> buff) {

if (buff == "\"production\":") {

ifile >> buff;

buff = buff.substr(1, buff.size());

while (buff.rfind(']') == -1) {

buff = buff.substr(0, buff.size() - 1);

LProduction.push\_back(std::stod(buff));

ifile >> buff;

}

// Добавление последнего элемента

buff = buff.substr(0, buff.size() - 2);

LProduction.push\_back(std::stod(buff));

}

else if (buff == "\"consumption\":") {

ifile >> buff;

buff = buff.substr(1, buff.size());

while (buff.rfind(']') == -1) {

buff = buff.substr(0, buff.size() - 1);

LConsumption.push\_back(std::stod(buff));

ifile >> buff;

}

// Добавление последнего элемента

buff = buff.substr(0, buff.size() - 2);

LConsumption.push\_back(std::stod(buff));

}

else if (buff == "\"cost\":") {

ifile >> buff >> buff;

while (buff[0] != ']' || buff.size() > 1) {

buff = buff.substr(1, buff.size());

while (buff.rfind(']') == -1) {

buff = buff.substr(0, buff.size() - 1);

Basis.push\_back(cells(nullptr, std::stod(buff)));

ifile >> buff;

}

// Добавление последнего элемента

buff = buff.substr(0, buff.size() - 2);

Basis.push\_back(cells(nullptr, std::stod(buff)));

ifile >> buff;

}

}

else if (buff == "\"add\_cost\":") {

ifile >> buff;

buff = buff.substr(1, buff.size());

while (buff.rfind(']') == -1) {

buff = buff.substr(0, buff.size() - 1);

LPenalty.push\_back(std::stod(buff));

ifile >> buff;

}

// Добавление последнего элемента

buff = buff.substr(0, buff.size() - 2);

LPenalty.push\_back(std::stod(buff));

}

}

std::cout << "\n Считывание входного файла завершено...";

CountConsumption = LConsumption.size();

CountProduct = LProduction.size();

}

else {

std::cout << "\n Не удалось открыть входной файл: " << path;

}

}

void TransportTask::PrintBasis() {

int i = 0;

std::cout << "\n Полученный допустимый базис: \n";

for (int j = 0; j < Basis.size(); j++) {

if (Basis[j].Value != nullptr) {

std::cout << "(" << j / CountConsumption + 1 << "," << j % CountConsumption + 1 << ") ";

}

}

std::cout << "\n Опорный план: \n";

for (auto obj : Basis) {

if (i++ != CountConsumption) {

std::cout << std::setw(8) << obj.getValue();

}

else {

i = 1;

std::cout << "\n" << std::setw(8) << obj.getValue();

}

}

}

void TransportTask::PrintInputs() {

std::cout << "\n Количество пунктов производства: " << CountProduct;

std::cout << "\n Количество пунктов потребления: " << CountConsumption;

std::cout << "\n Объёмы производства в пунктах: \n ";

for (auto obj : LProduction) {

std::cout << obj << " ";

}

std::cout << "\n Объёмы потребления в пунктах: \n ";

for (auto obj : LConsumption) {

std::cout << obj << " ";

}

std::cout << "\n Матрица стоимостей выполнения задачи: \n";

int i = 0;

for (auto obj : Basis) {

if (i++ != CountConsumption) {

std::cout << std::setw(8) << obj.Cost;

}

else {

i = 1;

std::cout << "\n" << std::setw(8) << obj.Cost;

}

}

}

//--------------------------------------------------------------------------------||

bool TransportTask::Correction() {

int sumProd = 0, sumConsum = 0;

for (auto x : LProduction) {

sumProd += x;

}

for (auto x : LConsumption) {

sumConsum += x;

}

if (sumConsum == sumProd) {

std::cout << "\n Закрытая транспортная задача.";

return false;

}

else {

if (sumConsum < sumProd) {

std::cout << "\n\n Открытая транспортная задача с перепроизводством.";

dummyConsum = CountConsumption++;

std::vector<cells> locBasis;

if (LPenalty.size() != 0) {

auto iPenalty = LPenalty.begin();

int j = 0;

for (int i = 0; i < Basis.size(); i++) {

if (j++ % CountConsumption + 1 != CountConsumption)

locBasis.push\_back(Basis[i]);

else {

locBasis.push\_back(cells(nullptr, \*iPenalty));

iPenalty++;

i--;

}

}

locBasis.push\_back(cells(nullptr, \*iPenalty));

}

else {

int j = 0;

for (int i = 0; i < Basis.size(); i++) {

if (j++ % CountConsumption + 1 != CountConsumption)

locBasis.push\_back(Basis[i]);

else {

i--;

locBasis.push\_back(cells(nullptr, 0));

}

}

locBasis.push\_back(cells(nullptr, 0));

}

LConsumption.push\_back(sumProd - sumConsum);

Basis = locBasis;

}

else if (sumProd < sumConsum) {

std::cout << "\n\n Открытая транспортная задача с недопроизводством.";

if (LPenalty.size() != 0) {

for (auto x : LPenalty) {

Basis.push\_back(cells(nullptr, x));

}

}

else {

for (int i = 0; i < CountConsumption; i++)

Basis.push\_back(cells(nullptr, 0));// Заполнение стоимости нулями, если нет штрафов

}

dummyProd = CountProduct++;

LProduction.push\_back(sumConsum - sumProd);

}

std::cout << "\n ================================================================= Коррекция ===";

}

return true;

}

//--------------------------------------------------------------------------------||

void TransportTask::ANWA() {

auto locLProduction(LProduction);

auto locLConsumption(LConsumption);

auto locProd = locLProduction.begin();

auto locConsum = locLConsumption.begin();

auto obj = Basis.begin();

double remains(0);

while (obj != Basis.end()) {

remains = \*locConsum - \*locProd;

// Спрос удовлетворён

if (remains <= 0) {

obj->setValue(\*locProd + remains); // Сколько ушло от предприятия потребителю.

\*locProd -= obj->getValue();

\*locConsum = 0;

if (remains == 0) {

locProd++; // Переход к некст производству, если ресурсы закончились

// Если не последняя строка

if ((obj - Basis.begin()) / CountConsumption < CountProduct - 1) {

// Занос фиктивного 0 в матрицу для вычисления потенциалов

if (((obj + 1) - Basis.begin()) % CountConsumption >= 0)

(obj + 1)->setValue(0);

obj += CountConsumption; // Сдвиг в списке на слой ниже - слой нового потребителя

}

}

obj++;

locConsum++; // Переход к некст потребителю

}

// Спрос не удовлетворён

else {

obj->setValue(\*locProd); // Сколько ушло от предприятия потребителю.

\*locConsum -= \*locProd;

\*locProd = 0; locProd++;

obj += CountConsumption; // Сдвиг в списке на слой ниже - слой нового поставщика

}

}

}

void TransportTask::MME() {

auto locLProduction(LProduction);

auto locLConsumption(LConsumption);

auto locProd = locLProduction.begin();

auto locConsum = locLConsumption.begin();

auto locBasis(Basis);

int iter = 0;

while (iter < CountConsumption + CountProduct - 1) {

int index = getMinCost(locBasis, locLProduction, locLConsumption);

int row = getRow(index);

int column = getColumn(index);

int a = locLConsumption[column] - locLProduction[row];

// И потребитель и производитель кончились

if (a == 0) {

locLConsumption[column] = 0;

Basis[index].setValue(locLProduction[row]);

locLProduction[row] = 0;

int iter = CountConsumption \* row;

while (getRow(iter) == row) {

locBasis[iter].Cost = -1;

iter++;

}

}

// остаток у потребителя - удаляется строке

else if (a > 0) {

locLConsumption[column] = a;

Basis[index].setValue(locLProduction[row]);

locLProduction[row] = 0;

int iter = CountConsumption \* row;

while (getRow(iter) == row) {

locBasis[iter].Cost = -1;

iter++;

}

}

// Остаток у производителя - удаляется столбец

else if (a < 0) {

locLProduction[row] = -a;

Basis[index].setValue(locLConsumption[column]);

locLConsumption[column] = 0;

int i = column;

while (getColumn(i) == column && i < locBasis.size()) {

locBasis[i].Cost = -1;

i += CountConsumption;

}

}

iter++;

}

}

//--------------------------------------------------------------------------------||

int TransportTask::getRow(int x) {

return x / CountConsumption;

}

int TransportTask::getColumn(int x) {

return x % CountConsumption;

}

int TransportTask::getMinCost(std::vector<cells>& locBasis, std::vector<double>& locProd, std::vector<double>& locConsum) {

double min = locBasis[0].Cost;

int index = 0, iter = 0;

bool f1 = false;

for (auto x : locBasis) {

if ((min > x.Cost || min == -1) && x.Cost > -1 && getRow(iter) != dummyProd && getColumn(iter) != dummyConsum) {

min = x.Cost;

index = iter;

f1 = true;

}

iter++;

}

// Если мнимальный элемент в базисе не была найден

// Поиск среди фиктивных столбцов/строк

if (!f1 && min == -1) {

// Поиск по фиктивному столбцу

if (dummyConsum != -1) {

iter = dummyConsum;

while (iter < locBasis.size()) {

if ((min > locBasis[iter].Cost || min == -1) && locBasis[iter].Cost > -1) {

min = locBasis[iter].Cost;

index = iter;

f1 = true;

}

iter += CountConsumption;

}

}

// Поиск по фиктивной строке

else if (dummyProd != -1) {

iter = (dummyProd + 1) \* CountConsumption;

while (iter < locBasis.size()) {

if ((min > locBasis[iter].Cost || min == -1) && locBasis[iter].Cost > -1) {

min = locBasis[iter].Cost;

index = iter;

f1 = true;

}

iter++;

}

}

}

return index;

}

template <typename T>

bool contains(std::vector<T> vec, T value) {

auto iter = vec.begin();

while (++iter != vec.end()) {

if (\*iter == value)

return true;

}

return false;

}

//--------------------------------------------------------------------------------||

// Запоминать для каждой строки и столбца сколько у него уже встречено вершин

void TransportTask::NewBasis() {

std::vector<int> chain = getChain();

if (isPrint) {

std::cout << "\n Цепочка:\n ";

for (int x : chain) {

std::cout << "(" << getRow(x) + 1 << ", " << getColumn(x) + 1 << ")";

}

}

#pragma region --- Поиск минимального элемента и сдвиг по цепочке ---

int min = INT32\_MAX;

int indexMin = CountConsumption \* CountProduct;

// Поиск минимального элемента в цепочке

int i = 0;

for (auto cell = (chain.begin()); cell != chain.end(); cell++, i++) {

// Если элемент в цепочке нечётный - и меньше минимального

if (i % 2 > 0 && (min > Basis[\*cell].getValue())) {

min = Basis[\*cell].getValue();

indexMin = i;

}

}

// Сдвиг

i = 0;

bool isFirts = false;

for (int x : chain) {

Basis[x].setValue((i % 2 > 0) ? Basis[x].getValue() - min : Basis[x].getValue() + min);

if (indexMin == i) { //&& chain[i] != indexPotential //!isFirts &&

if (isPrint)

std::cout << "\n Из базиса выводится элемент X(" << getRow(x) + 1 << "," << getColumn(x) + 1 << ") = " << min;

delete Basis[x].Value;

Basis[x].Value = nullptr;

isFirts = true;

}

i++;

}

#pragma endregion

}

std::vector<int> TransportTask::getChain() {

if (isPrint)

std::cout << "\n Поиск цепочки.";

auto locBasis(Basis);

std::vector<int> Chain; // Список индексов из цепочки

std::stack<std::array<int, 5>> state; // Стек состояний

Chain.push\_back(indexPotential);

int i = indexPotential;

bool isChain = false;

Direction fromDir = Non;

int row = getRow(i);

int rowIter = 0;

int column = getColumn(i);

int columnIter = 0;

int counterIter = 0;

// Вектор с базисными индексами в различных направлениях и направлением откуда пришли

std::array<int, 5> ListDir{ -1, -1, -1, -1, Non };

/// <summary>

/// Цикл перехода к новой базисной ячейке

/// </summary>

while (!isChain && counterIter < CountConsumption \* CountConsumption) {

counterIter++;

column = getColumn(i);

row = getRow(i);

// Добавить проверку на не более двух вершин в строке и столбце, что ещё раз позволит скипать шаги.

// Обход вверх

if (fromDir != up) {

bool isBasis = false;

int ii = i - CountConsumption; // Временный итератор по столбцу

columnIter = getColumn(ii);

// Пока не достигнут потолок

while (columnIter == column && ii >= 0) {

// Автоматически исключается случай хождения в обратном направлении благодаря contains

if (contains(Chain, ii))

break;

isChain = ii == indexPotential;

// Если встречена базисная переменная.

if (locBasis[ii].Value != nullptr || isChain) {

isBasis = true;

break;

}

ii -= CountConsumption; // Шаг по матрице вверх

rowIter = getRow(ii);

columnIter = getColumn(ii);

}

// Если базис был встречен

if (isBasis) {

if (isChain) {

i = ii; // Цепочка найдена - остановка поисков

Chain.push\_back(i);

break;

}

else {

ListDir[up] = ii; // В направлении вверх записываем индекс следующего шага

}

}

}

// Обход вниз

if (fromDir != down) {

bool isBasis = false;

int ii = i + CountConsumption; // Временный итератор по столбцу

columnIter = getColumn(ii);

// Пока не достигнут пол

while (columnIter == column && ii < locBasis.size()) {

// Автоматически исключается случай хождения в обратном направлении благодаря contains

if (contains(Chain, ii))

break;

isChain = ii == indexPotential;

// Если встречена базисная переменная.

if (locBasis[ii].Value != nullptr || isChain) {

isBasis = true;

break;

}

ii += CountConsumption; // Шаг по матрице вверх

rowIter = getRow(ii);

columnIter = getColumn(ii);

}

// Если базис был встречен

if (isBasis) {

if (isChain) {

i = ii; // Цепочка найдена - остановка поисков

Chain.push\_back(i);

break;

}

else {

ListDir[down] = ii; // В направлении вверх записываем индекс следующего шага

}

}

}

// Обход влево

if (fromDir != left) {

bool isBasis = false;

int ii = i - 1; // Временный итератор по строке

rowIter = getRow(ii);

// Пока не достигнута левая границы

while (rowIter == row && ii >= 0) {

// Автоматически исключается случай хождения в обратном направлении благодаря contains

if (contains(Chain, ii))

break;

isChain = ii == indexPotential;

// Если встречена базисная переменная.

if (locBasis[ii].Value != nullptr || isChain) {

isBasis = true;

break;

}

ii--; // Шаг по матрице вверх

columnIter = getColumn(ii);

rowIter = getRow(ii);

}

// Если базис был встречен

if (isBasis) {

if (isChain) {

i = ii; // Цепочка найдена - остановка поисков

Chain.push\_back(i);

break;

}

else {

ListDir[left] = ii; // В направлении вверх записываем индекс следующего шага

}

}

}

// Обход вправо

if (fromDir != right) {

bool isBasis = false;

int ii = i + 1; // Временный итератор по строке

rowIter = getRow(ii);

// Пока не достигнута левая границы

while (rowIter == row && ii < locBasis.size()) {

// Автоматически исключается случай хождения в обратном направлении благодаря contains

if (contains(Chain, ii))

break;

isChain = ii == indexPotential;

// Если встречена базисная переменная.

if (locBasis[ii].Value != nullptr || isChain) {

isBasis = true;

break;

}

ii++; // Шаг по матрице вверх

columnIter = getColumn(ii);

rowIter = getRow(ii);

}

// Если базис был встречен

if (isBasis) {

if (isChain) {

i = ii; // Цепочка найдена - остановка поисков

Chain.push\_back(i);

break;

}

else {

ListDir[right] = ii; // В направлении вверх записываем индекс следующего шага

}

}

}

// Если цепочка не найдена - но есть куда ходить

if (ListDir[0] != -1 || ListDir[1] != -1 || ListDir[2] != -1 || ListDir[3] != -1) {

Direction j = up;

while (j < 4) {

if (ListDir[j] != -1) {

Chain.push\_back(ListDir[j]);

i = ListDir[j];

ListDir[j] = -1; // Отмечается направление, в котором будет шаг

break;

}

j = (Direction)(j + 1);

}

// Костыли

// На следующую итерацию блокируется ход назад.

switch (j)

{

case up:

fromDir = down;

break;

case down:

fromDir = up;

break;

case left:

fromDir = right;

break;

case right:

fromDir = left;

break;

default:

break;

}

state.push(ListDir);

ListDir = { -1, -1, -1, -1, fromDir };

}

// Если все пути пройдены - шаг назад

else {

//delete locBasis[i].Value;

locBasis[i].Value = nullptr;

Chain.pop\_back(); // Удаление из цепочки

i = Chain.back();

ListDir[up] = state.top()[up]; // Возврат предыдущего состояния обхода.

ListDir[down] = state.top()[down]; // Возврат предыдущего состояния обхода.

ListDir[left] = state.top()[left]; // Возврат предыдущего состояния обхода.

ListDir[right] = state.top()[right]; // Возврат предыдущего состояния обхода.

fromDir = (Direction)state.top()[4];

ListDir[4] = fromDir;

state.pop();

}

}

#pragma region --- Избавление от нескольких вершин на одной прямой ---

std::vector<int> newChain;

newChain.push\_back(Chain[0]);

newChain.push\_back(Chain[1]);

for (int i = 2; i < Chain.size(); i++) {

// Если на одной прямой лежит подряд нескольо вершин - удалить среднюю.

if (getColumn(Chain[i - 1]) == getColumn(Chain[i]) && // По столбцу

getColumn(Chain[i - 2]) == getColumn(Chain[i]) ||

getRow(Chain[i - 1]) == getRow(Chain[i]) && // По строке

getRow(Chain[i - 2]) == getRow(Chain[i])) {

newChain.back() = Chain[i]; // Перезапись дубля

}

else {

newChain.push\_back(Chain[i]);

}

}

newChain.pop\_back(); // Удаление дубля индекса добавления базисной переменной.

Chain = newChain;

#pragma endregion

if (isPrint)

std::cout << "\n Цепочка найдена за " << counterIter << " шагов.";

return Chain;

}

bool TransportTask::CheckBasis() {

double ansrew = 0;

for (auto obj : Basis) {

ansrew += obj.Cost \* obj.getValue();

}

if (isPrint)

std::cout << "\n Значение целевой функции: " << ansrew;

return 0;

}

//--------------------------------------------------------------------------------||

potential TransportTask::DefinePotential() {

double\* u;

double\* v;

bool isFirst = true;

int initPotentU = 1,initPotentV = 0,row = 0, column = 0,iter = 0;

auto locBasis(Basis);

// Вектор с базисными индексами в различных направлениях

std::array<int, 4> ListDir{ -1, -1, -1, -1 };

std::vector<int> indexes;

// Стек состояний

std::stack<std::array<int, 4>> stateMashine;

#pragma region --- Инициализация списков потенциалов ---

auto\* U = new std::vector<double>();

for (size\_t i = 0; i < CountProduct; i++) {

U->push\_back(LONG\_MAX);

}

// Инициализация нулем потенциала строки, где есть поставка.

for (int iter = 0; iter < locBasis.size(); iter += CountConsumption) {

if (locBasis[iter].Value != nullptr) {

(\*U)[getRow(iter)] = 0;

break;

}

}

auto\* V = new std::vector<double>();

for (size\_t i = 0; i < CountConsumption; i++) {

V->push\_back(LONG\_MAX);

}

#pragma endregion

while (initPotentU != CountProduct || initPotentV != CountConsumption) {

// Если обход закончен, но все потенциалы так и не были найдены - последний обход в тупую.

if (iter == -1) {

while (++iter < locBasis.size()) {

// Если выбрано направление для шага

if (locBasis[iter].Value != nullptr) {

row = getRow(iter);

column = getColumn(iter);

u = &(\*U)[row];

v = &(\*V)[column];

// Шаг по столбцу - заполнение потенц поставщиков

if (\*u == LONG\_MAX && \*v != LONG\_MAX) {

\*u = Basis[iter].Cost - \*v;

initPotentU++;

locBasis[iter].Value = nullptr;

}

// Шаг по строке - заполнение потенц потребителей

else if (\*v == LONG\_MAX && \*u != LONG\_MAX) {

\*v = Basis[iter].Cost - \*u;

initPotentV++;

locBasis[iter].Value = nullptr;

}

}

}

}

// Если выбрано направление для шага

if (locBasis[iter].Value != nullptr) {

row = getRow(iter);

column = getColumn(iter);

u = &(\*U)[row];

v = &(\*V)[column];

// Шаг по столбцу - заполнение потенц поставщиков

if (\*u == LONG\_MAX && \*v != LONG\_MAX) {

\*u = Basis[iter].Cost - \*v;

initPotentU++;

locBasis[iter].Value = nullptr;

}

// Шаг по строке - заполнение потенц потребителей

else if (\*v == LONG\_MAX && \*u != LONG\_MAX) {

\*v = Basis[iter].Cost - \*u;

initPotentV++;

locBasis[iter].Value = nullptr;

}

}

//-------------------------------------------------------------------------||

// Сканирование направления - поиск базисных элементов

if (ListDir[down] == Non) {

if (row + 1 < CountProduct) {

int locIter = iter + CountConsumption;

// Шаг по столбцу - пока не конец и не встречена бизасная ячейка

while (getRow(locIter + CountConsumption) < CountProduct && locBasis[locIter].Value == nullptr) {

locIter += CountConsumption;

//if (getColumn(locIter) != column) {

// locIter -= CountConsumption;

// break;

//}

}

if (locIter < Basis.size() && locBasis[locIter].Value != nullptr) {

ListDir[down] = locIter;

}

else {

ListDir[down] = notFound; // Не найдено

}

}

else {

ListDir[down] = notFound; // Не найдено

}

}

if (ListDir[right] == Non) {

if (column + 1 < CountConsumption) {

int locIter = iter + 1;

while (getRow(locIter + 1) == row && locBasis[locIter].Value == nullptr) {

locIter++;

//if (getRow(locIter) != row) {

// locIter--;

// break;

//}

}

if (locIter < Basis.size() && locBasis[locIter].Value != nullptr) {

ListDir[right] = locIter;

}

else {

ListDir[right] = notFound; // Не найдено

}

}

else {

ListDir[right] = notFound; // Не найдено

}

}

if (ListDir[up] == Non) {

if (row - 1 >= 0) {

int locIter = iter - CountConsumption;

// Шаг по столбцу - пока не конец и не встречена бизасная ячейка

while (getRow(locIter - CountConsumption) > 0 && locBasis[locIter].Value == nullptr && locIter - CountConsumption >= 0) {

locIter -= CountConsumption;

/\*if (getColumn(locIter) != column || locIter < 0) {

locIter+= CountConsumption;

break;

}\*/

}

if (locIter < Basis.size() && locBasis[locIter].Value != nullptr) {

ListDir[up] = locIter;

}

else {

ListDir[up] = notFound; // Не найдено

}

}

else {

ListDir[up] = notFound; // Не найдено

}

}

if (ListDir[left] == Non) {

if (column - 1 >= 0) {

int locIter = iter - 1;

while (getRow(locIter - 1) == row && locBasis[locIter].Value == nullptr && locIter - 1 >= 0) {

locIter--;

/\*if (getRow(locIter) != row || locIter < 0) {

locIter++;

break;

}\*/

}

if (locIter < Basis.size() && locBasis[locIter].Value != nullptr) {

ListDir[left] = locIter;

}

else {

ListDir[left] = notFound; // Не найдено

}

}

else {

ListDir[left] = notFound; // Не найдено

}

}

//-------------------------------------------------------------------------||

// Cелектор направлений из уже отсканированных ячеек.

while (!stateMashine.empty() || isFirst || iter != -1) {

// Если есть куда ходить

if (ListDir[0] > -1 || ListDir[1] > -1 || ListDir[2] > -1 || ListDir[3] > -1) {

isFirst = false;

//int fromCell = -1;

bool is = false;

Direction j = up;

Direction k = up;

while (j < 4) {

// Если направление задано и не указывает само на себя,

// или значения потенциалов в этой точке всё ещё не были расчитаны..

// ||

// (\*U)[getRow(iter)] == LONG\_MAX ||

// (\*V)[getColumn(iter)] == LONG\_MAX)

if (ListDir[j] > -1 && ListDir[j] != iter) {

indexes.push\_back(iter);

iter = ListDir[j];

ListDir[j] = notFound; // Сбрасывается направление, в котором будет шаг

//if (!is) {

k = j;

//}

is = true;

break;

}

j = (Direction)(j + 1);

}

switch (k)

{

case up:

k = down;

break;

case down:

k = up;

break;

case left:

k = right;

break;

case right:

k = left;

break;

}

// Если есть куда шагать

if (is) {

stateMashine.push(ListDir);

ListDir = { -1, -1, -1, -1 };

ListDir[k] = iter; //Пометка направления из которого пришли - номером текущей ячейки.

break;

}

}

if (indexes.empty()) {

iter = -1;

break;

}

// Иначе - обновление stateMashine и новая проверка

ListDir = stateMashine.top();

stateMashine.pop();

iter = indexes.back();

indexes.pop\_back();

}

}

return potential(U, V);

}

bool TransportTask::CheckPotential(std::vector<double>& U, std::vector<double>& V) {

double\* u;

double\* v;

int iter = 0;

double min = LONG\_MAX;

int iterMin;

if (isPrint)

std::cout << "\n Оценки свободных переменных:\n ";

while (iter < Basis.size()) {

if (Basis[iter].Value == nullptr) {

u = &U[getRow(iter)];

v = &V[getColumn(iter)];

if (min > Basis[iter].Cost - \*u - \*v) {

iterMin = iter;

min = Basis[iter].Cost - \*u - \*v;

}

if (isPrint)

std::cout << "ΔC[" << getRow(iter) + 1 << "," << getColumn(iter) + 1 << "] = " << Basis[iter].Cost - \*u - \*v << "\n ";

iter++;

}

else

iter++;

}

delete& U;

delete& V;

indexPotential = iterMin;

// Если все оценки положительны - базис оптимален.

return min >= 0;

}

//--------------------------------------------------------------------------------||