ДОДАТОК А

Текст програми

Код генетичного алгоритму, файл TspGaRestrict.java

```
package core.tsp;
import core.io.ConfigDAO;
import java.io.IOException;
import java.util.*;

public class TspGaRestrict extends RestrictionChecker {

    private long workTime;
    private int bestTourCost;
    private List<Integer> bestRoute;
    private final int tournamentSize;
    private final int[][] distanceMatrix;
    private final int populationSize;
    private final int numGenerations;
    private final double mutationRate;
    private final int elitismCount;
    private final int maxStagnation;
    private int currentStagnation;
    private final int startCity;
    private final int numTraveler;
    private final boolean isRestrictionNeeded;
    public long getWorkTime() {
        return workTime;
    }
    public int getBestTourCost() {
        return bestTourCost;
    }
    public List<Integer> getBestRoute() {
        return bestRoute;
    }

    public TspGaRestrict(int[][] distanceMatrix) throws
IOException {
        ConfigDAO dao = new ConfigDAO();
        this.populationSize = dao.getPopulationSize();
        this.numGenerations = dao.getNumGenerations();
        this.mutationRate = dao.getMutationRate();
        this.elitismCount = dao.getElitismCount();
        this.maxStagnation = dao.geMaxStagnation();
        this.tournamentSize = dao.getTournamentSize();
        this.startCity = dao.getStartPoint();
        this.numTraveler = dao.getNumOfTraveler();
        this.distanceMatrix = distanceMatrix;
        this.isRestrictionNeeded = dao.isRestrictionsNeeded();
```

```
        this.currentStagnation = 0;
        runGA();
    }
    public List<Integer> solveTSP() {
        int numCities = distanceMatrix.length;
        List<List<Integer>> population =
generateRandomPopulation(numCities);
        int lastBestDistance =
calculateTotalDistance(Collections.min(population, (a1, b1) ->
calculateTotalDistance(a1) - calculateTotalDistance(b1)));
        for (int generation = 0; generation < numGenerations;
generation++) {
            population.sort((a, b) -> calculateTotalDistance(a)
- calculateTotalDistance(b));
            int currentBestDistance =
calculateTotalDistance(population.get(0));

            if (currentBestDistance == lastBestDistance) {
                currentStagnation++;
                if (currentStagnation >= maxStagnation) {
                    break;
                }
            } else {
                lastBestDistance = currentBestDistance;
                currentStagnation = 0;
            }

            List<List<Integer>> elitePopulation = new
ArrayList<>(population.subList(0, elitismCount));
            List<List<Integer>> parents = new ArrayList<>();
            List<Integer> parent1;
            List<Integer> parent2;
            for (int i = 0; i < (populationSize - elitismCount);
i++) {
                if (isRestrictionNeeded){
                    do {
                        parent1 =
tournamentSelection(population);
                        parent2 =
tournamentSelection(population);
                    } while
(!(RestrictionChecker.checkRoute(parent1) &&
RestrictionChecker.checkRoute(parent2)));

                } else {
                    parent1 = tournamentSelection(population);
                    parent2 = tournamentSelection(population);
                }
                parents.add(parent1);
                parents.add(parent2);
            }
            List<List<Integer>> newPopulation = new
```

```
ArrayList<>(elitePopulation);
            List<List<Integer>> children;
            for (int i = 0; i < parents.size(); i += 2) {
                if (isRestrictionNeeded){
                    do {
                        parent1 = parents.get(i);
                        parent2 = parents.get(i + 1);
                        children = crossover(parent1, parent2);
                        mutate(children.get(0));
                        mutate(children.get(1));
                    } while
(!(RestrictionChecker.checkRoute(children.get(0)) &&
RestrictionChecker.checkRoute(children.get(1))));
                } else {
                    parent1 = parents.get(i);
                    parent2 = parents.get(i + 1);
                    children = crossover(parent1, parent2);
                    mutate(children.get(0));
                    mutate(children.get(1));
                }
                newPopulation.add(children.get(0));
                newPopulation.add(children.get(1));
            }
            population = newPopulation;
        }
        return Collections.min(population, (a, b) ->
calculateTotalDistance(a) - calculateTotalDistance(b));
    }

    private List<List<Integer>> generateRandomPopulation(int
numCities) {
        List<List<Integer>> population = new ArrayList<>();
        for (int i = 0; i < populationSize; i++) {
            List<Integer> individual;
            if (isRestrictionNeeded){
                do {
                    individual = generateRandomRoute(numCities);
                } while
(!RestrictionChecker.checkRoute(individual));
                population.add(individual);

            } else {
                individual = generateRandomRoute(numCities);
                population.add(individual);
            }
        }
        return population;
    }

    private List<Integer> generateRandomRoute(int numCities) {
        List<Integer> route = new ArrayList<>();
        for (int i = 1; i <= numCities; i++) {
```

```
            if (i != startCity) {
                route.add(i);
            }
        }
        Collections.shuffle(route);
        route.add(0, startCity);
        route.add(startCity);
        return route;
    }

    public int calculateTotalDistance(List<Integer> path) {
        int totalDistance = 0;
        int size = path.size();
        int from = path.get(0) - 1;

        for (int i = 1; i < size; i++) {
            int to = path.get(i) - 1;
            totalDistance += distanceMatrix[from][to];
            from = to;
        }
        totalDistance += distanceMatrix[from][path.get(0) - 1];
        return totalDistance;
    }

    private List<Integer>
tournamentSelection(List<List<Integer>> population) {
        List<List<Integer>> tournament = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < tournamentSize; i++) {
            int randomIndex = random.nextInt(population.size());
            tournament.add(population.get(randomIndex));
        }
        return Collections.min(tournament, (a, b) ->
calculateTotalDistance(a) - calculateTotalDistance(b));
    }

    private List<List<Integer>> crossover(List<Integer> parent1,
List<Integer> parent2) {
        int size = parent1.size();
        Random random = new Random();
        List<List<Integer>> children = new ArrayList<>();
        int start = random.nextInt(size - 1) + 1; // Don't
change the first element
        int index = start;
        List<Integer> child1 = new
ArrayList<>(Collections.nCopies(size, -1));
        List<Integer> child2 = new
ArrayList<>(Collections.nCopies(size, -1));
        int iterations = 0;

        do {
            child1.set(index, parent1.get(index));
```

```
                child2.set(index, parent2.get(index));
                index = parent2.indexOf(parent1.get(index));
                iterations++;
                if (iterations > size) {
                    break;
                }
            } while (index != start);

            for (int i = 0; i < size; i++) {
                if (child1.get(i) == -1) {
                    child1.set(i, parent2.get(i));
                }
                if (child2.get(i) == -1) {
                    child2.set(i, parent1.get(i));
                }
            }
            child1.remove(0);
            child1.add(0, startCity);
            child1.remove(child1.size() - 1);
            child1.add(startCity);
            child2.remove(0);
            child2.add(0, startCity);
            child2.remove(child2.size() - 1);
            child2.add(startCity);
            children.add(child1);
            children.add(child2);
            return children;
        }

        private void mutate(List<Integer> individual) {
            if (Math.random() < mutationRate) {
                int size = individual.size();
                Random random = new Random();

                int pos1 = 1 + random.nextInt(size - 2);
int pos2;
                do {
                    pos2 = 1 + random.nextInt(size - 2);
} while (pos1 == pos2);

                if (pos1 > pos2) {
                    int temp = pos1;
                    pos1 = pos2;
                    pos2 = temp;
                }
                while (pos1 < pos2) {
                    int temp = individual.get(pos1);
                    individual.set(pos1, individual.get(pos2));
                    individual.set(pos2, temp);
                    pos1++;
                    pos2--;
                }
```

```
        }
    }

    public void runGA() {
        long time = System.nanoTime();
        bestRoute = solveTSP();
        time = System.nanoTime() - time;
        bestTourCost = calculateTotalDistance(bestRoute);
        for (int i = 0; i < bestRoute.size(); i++) {
            if (i > distanceMatrix.length - numTraveler + 1) {
                int index = bestRoute.indexOf(i);
                bestRoute.remove(index);
                bestRoute.add(index, startCity);
            }
        }
        workTime = time;
    }
}
```

Код методу гілок та меж, файл TspBnbRestrict.java

```
package core.tsp;
import core.io.ConfigDAO;
import java.io.IOException;
import java.util.*;

public class TspBnbRestrict extends RestrictionChecker {
    private long workTime;
    private final int numTraveler;
    private final boolean isRestrictionNeeded;
    private final int startCity;
    private final int N;
    private final int[] final_path;
    private final boolean[] visited;
    private int final_res = Integer.MAX_VALUE;

    public TspBnbRestrict(int[][] costMatrix) throws IOException
{
        ConfigDAO dao = new ConfigDAO();
        this.numTraveler = dao.getNumOfTraveler();
        this.isRestrictionNeeded = dao.isRestrictionsNeeded();
        this.startCity = dao.getStartPoint() - 1;
        this.N = costMatrix.length;
        this.final_path = new int[N + 1];
        this.visited = new boolean[N];
        TSP(costMatrix);
    }

    public List<Integer> getFinal_path() {
        List<Integer> finalRoute = new
```

```
ArrayList<>(Arrays.stream(final_path)
                .boxed().toList());
        finalRoute.replaceAll(integer -> integer + 1);
        for (int i = 0; i < finalRoute.size(); i++) {
            if (i > N - numTraveler + 1) {
                int index = finalRoute.indexOf(i);
                finalRoute.remove(index);
                finalRoute.add(index, startCity + 1);
            }
        }
        return finalRoute;
    }
    private List<Integer> arrayAsList(int[] path){
        List<Integer> rote = new ArrayList<>(Arrays.stream(path)
                .boxed().toList());
        rote.replaceAll(integer -> integer + 1);
        return rote;
    }

    public long getWorkTime() {
        return workTime;
    }

    public int getFinal_res() {
        return final_res;
    }

    private void copyToFinal(int[] curr_path) {
        if (N >= 0) System.arraycopy(curr_path, 0, final_path,
0, N);
        final_path[N] = curr_path[0];
    }

    int firstMin(int[][] adj, int i) {
        int min = Integer.MAX_VALUE;
        for (int k = 0; k < N; k++)
            if (adj[i][k] < min && i != k)
                min = adj[i][k];
        return min;
    }

    private int secondMin(int[][] adj, int i) {
        int first = Integer.MAX_VALUE, second =
Integer.MAX_VALUE;
        for (int j=0; j<N; j++)
        {
            if (i == j)
                continue;

            if (adj[i][j] <= first)
            {
                second = first;
```

```
                first = adj[i][j];
            }
            else if (adj[i][j] <= second &&
                    adj[i][j] != first)
                second = adj[i][j];
        }
        return second;
    }
    private void TSPRec(int[][] adj, int curr_bound, int
curr_weight, int level, int[] curr_path) {
        if (level == N)
        {
            if (adj[curr_path[level - 1]][curr_path[0]] != 0)
            {
                int curr_res = curr_weight +
                        adj[curr_path[level-1]][curr_path[0]];
                if (isRestrictionNeeded){
                    if
(RestrictionChecker.checkRoute(arrayAsList(curr_path))){
                        if (curr_res < final_res)
                        {
                            copyToFinal(curr_path);
                            final_res = curr_res;
                        }
                    }
                } else {
                    if (curr_res < final_res)
                    {
                        copyToFinal(curr_path);
                        final_res = curr_res;
                    }
                }

            }
            return;
        }
        for (int i = 0; i < N; i++)
        {
            if (adj[curr_path[level-1]][i] != 0 && !visited[i])
{
                int temp = curr_bound;
                curr_weight += adj[curr_path[level - 1]][i];
                if (level==1)
                    curr_bound -= ((firstMin(adj,
curr_path[level - 1]) +
                            firstMin(adj, i))/2);
                else
                    curr_bound -= ((secondMin(adj,
curr_path[level - 1]) +
                            firstMin(adj, i))/2);
                if (curr_bound + curr_weight < final_res)
                {
```

```
                    curr_path[level] = i;
                    visited[i] = true;
                    TSPRec(adj, curr_bound, curr_weight, level +
1,
                            curr_path);
                }
                curr_weight -= adj[curr_path[level-1]][i];
                curr_bound = temp;
                Arrays.fill(visited,false);
                for (int j = 0; j <= level - 1; j++)
                    visited[curr_path[j]] = true;
            }
        }
    }

    private void TSP(int[][] adj) {
        long startTime = System.nanoTime();
        int[] curr_path = new int[N + 1];
        curr_path[0] = startCity;
        visited[startCity] = true;
        int curr_bound = 0;
        Arrays.fill(curr_path, -1);
        Arrays.fill(visited, false);
        for (int i = 0; i < N; i++) {
            if (i != startCity) {
                curr_bound += (firstMin(adj, i) + secondMin(adj,
i));
            }
        }
        curr_bound = (curr_bound == 1) ? 1 : curr_bound / 2;
        curr_path[0] = startCity;
        visited[startCity] = true;
        TSPRec(adj, curr_bound, 0, 1, curr_path);
        workTime = System.nanoTime() - startTime;
    }
}
```

ДОДАТОК Б

Апробація наукових результатів

Міністерство освіти і науки України

NURE

Харківський національний університет радіоелектроніки

ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2023

(Випуск 2)

[електронне видання]

KITAP

http://nure.ua/department/kafedra-komp-yuterno-integrovanih-tehnologiy-avtomatizatsiyi-ta-mehatroniki-kitam

iTeD

http://itez.zntu.edu.ua/

http://kafea.kdu.edu.ua

Харків 2023

УДК 519.81:658.512

ОПТИМІЗАЦІЯ МАРШРУТІВ В ЛОГІСТИЧНИХ МЕРЕЖАХ ВИРОБНИЧОГО ПРОЦЕСУ

О.В. Чернишенко
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
E-mail: oleksandr.chernyshenko@nure.ua

Анотація: У роботі проведено аналіз проблеми оптимізації маршрутів логістичної мережі виробничого процесу. Сформульовано задачі оптимізації маршрутів як класичної задачі комівояжера (TSP) та задачі багатьох комівояжерів з обмеженнями (MTSPC). Наведено математичні співвідношення для зведення задачі MTSPC до задачі TSP. Для розв'язання задачі оптимізації маршрутів запропоновано використати метод гілок та меж.

Ключові слова: логістична мережа, моделювання, оптимізація маршрутів.

OPTIMIZATION OF ROUTES IN THE LOGISTICS NETWORKS OF THE PRODUCTION PROCESS

O. Chernyshenko
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av.,14
E-mail: oleksandr.chernyshenko@nure.ua

Annotation: The paper analyzes the problem of optimizing the routes of the logistics network of the production process. Route optimization problems are formulated as a classical traveling salesman problem (TSP) and a multiple traveling salesman problem with constraints (MTSPC). Mathematical relations for reducing the MTSPC problem to the TSP problem are given. To solve the route optimization problem, it is proposed to use the method of branches and boundaries.

Key words: logistics network, modeling, route optimization.

АКТУАЛЬНІСТЬ РОБОТИ. Ускладнення продукції сучасних виробничих підприємств, зростання вимог до її якості, розвиток технологій вимагають відповідних змін виробничих процесів (ВП). Зміни у структурі та технологіях виробничих процесів приводять до необхідності змін логістичних мереж (ЛМ), що забезпечують їх функціонування. Здатність швидко та ефективно координувати потоки сировини, матеріалів, обладнання, виробів є одним з ключових факторів конкурентоспроможності виробничих компаній. З ускладненням ланцюгів поставок та збільшенням вимог до швидкості доставки, існує необхідність у розробці та впровадженні більш ефективних стратегій маршрутизації [1].

Оптимізація маршрутів у ЛМ не тільки сприяє підвищенню ефективності виробництва, але й значно знижує експлуатаційні витрати, зокрема на транспортні послуги. У контексті глобалізації та зростаючих вимог до екологічності, оптимізація логістичних маршрутів відіграє одну з ключових ролей у мінімізації вуглецевого сліду виробничих і транспортних процесів. Враховуючи ці фактори, вивчення та розробка нових підходів до оптимізації маршрутів у ЛМ є актуальною темою для досліджень. Це допоможе підприємствам підвищити ефективність й сприятиме сталому розвитку.

АНАЛІЗ ПРОБЛЕМИ ОПТИМІЗАЦІЇ ЛОГІСТИЧНИХ МАРШРУТІВ. Оптимізація логістичних мереж у виробничих процесах охоплює цілу низку заходів, які мають на меті вдосконалення різних елементів виробничих та розподільчих систем з метою скорочення

витрат та підвищення їхньої ефективності. До цих заходів відносяться [2]:

– визначення оптимальної кількості та розташування виробничих об'єктів, складських та розподільчих центрів для мінімізації транспортних витрат і часу доставки;

– розробка та реінжиніринг маршрутів транспортування матеріалів у виробничому процесі з метою зниження логістичних витрат;

– координація роботи різних виробничих ліній для оптимального розподілу навантаження на обладнання та мінімізації простоїв;

– вдосконалення систем управління запасами з метою зниження витрат на зберігання та скорочення часу тримання сировини та готової продукції на складі.

Виробнича логістика є складовою виробничо-збутової логістики, в якій розрізняють задачі оптимізації макро- і мікрологістичних мереж [3–7]. Оптимізація макрологістичних мереж фокусується на великих компаніях які мережі яких розподілені територіально. Це включає в себе складні завдання, такі як управління міжнародними ланцюгами постачання, інтеграція різних транспортних засобів та оптимізація взаємодії між різними географічно розподіленими частинами ЛМ. З іншого боку, оптимізація мікрологістичних мереж стосується менших, часто локалізованих логістичних систем. Вона включає адаптацію та удосконалення існуючих мереж, які діють на обмежених територіях, наприклад, у межах міста, виробництва, або навіть одного виробничого цеху. Це зазвичай включає в себе розв'язання задач з оптимізації кільцевих маршрутів для ефективної доставки та розподілу матеріалів та готової продукції, а також управління локальними складськими процесами. Широке застосування при оптимізації мікрологістичних мереж знаходить задача комівояжера (TSP – Travelling Salesman Problem), що є однією з класичних проблем в теорії оптимізації. Вона виникає в ситуаціях, де необхідно визначити найефективніший маршрут для відвідування декількох точок, наприклад, складів, або різних виробничих ділянок, з мінімальними загальними витратами часу чи довжини шляху. Використання моделі TSP допомагає в розв'язанні таких задач, як оптимізація маршрутів доставки з розподільного центру до кількох точок розміщення технологічного обладнання або планування ефективних маршрутів для збору готової продукції чи відходів.

МОДЕЛІ ОПТИМІЗАЦІЇ ЛОГІСТИЧНИХ МАРШРУТІВ. Пропонується розглядати завдання оптимізації мікрологістичних маршрутів як узагальнену форму задачі комівояжера. Класична TSP-задача полягає у визначенні найбільш оптимального маршруту X^o, який проходить через певний набір точок ВП, відвідуючи кожну точку лише один раз і повертаючись у точку старту.

Модель оптимізації мікрологістичних маршрутів як TSP-задачу можна формалізувати у такому вигляді [8]:

$$X^o = arg\ \min_X \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}x_{ij}\,, \tag{1}$$

$$\sum_{i=1}^{N} x_{ij} = 1,\ \ j=\overline{1,N}\,,\ \sum_{j=1}^{N} x_{ij} = 1,\ \ i=\overline{1,N}\,, \tag{2}$$

де X – матриця, що визначає маршрут обходу пунктів (x_{ij} – наявність безпосереднього переходу з i-го пункту в j-й, $x_{ij} \in \{0,1\}$); N – кількість пунктів мережі; d_{ij} – вага (відстань, вартість) переходу з i-го пункту в j-й.

Формула (1) визначає цільову функцію – пошук маршруту з мінімальною вартістю. Обмеження (2) задають умову відвідування кожного пункту тільки один раз. Проте, якщо використовувати лише обмеження (2), то розв'язок задачі не обов'язково буде циклічним. Тобто маршрут може розпастися на декілька незв'язних між собою циклів. Для забезпечення

умови цілісності маршруту введемо наступне обмеження:

$$u_i - u_j + Nx_{ij} \le N-1,\ i,j=\overline{1,N}, i \ne j, \tag{3}$$

де u_j – номер кроку, на якому було відвідано j-й пункт; u_i – номер кроку, на якому було відвідано i-й пункт.

Модель (1) з обмеженнями (2)-(3) адаптована до ситуацій з порівняно невеликою кількістю точок доставки та не враховує різноманітні обмеження, такі як обсяги поставок, вантажопідйомність транспортних засобів, часові рамки доставки та інші. Для врахування цих факторів, більш доцільно представити задачу оптимізації логістичних мереж як задачу багатьох комівояжерів з обмеженнями (MTSPC – Multiple Traveling Salesman Problem with Constraints). У цьому варіанті m комівояжерів повинні відвідати всі точки ВП по одному разу, при цьому забезпечуючи повернення до стартової точки, враховуючи при цьому встановлені обмеження.

Модель оптимізації мікрологістичних маршрутів як MTSPC-задачу можна формалізувати у такому вигляді [8]:

$$X^o = arg\ \min_X \sum_{k=1}^{m}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}x_{ijk}, \tag{4}$$

$$\sum_{k=1}^{m}\sum_{i=1}^{N} x_{ij} = 1,\ j=\overline{1,N},\ \sum_{k=1}^{m}\sum_{j=1}^{N} x_{ij} = 1,\ i=\overline{1,N},\ \sum_{k=1}^{m} x_{ijk} \ge 1,\ i=\overline{1,N}, j=\overline{1,N}, \tag{5}$$

$$u_i - u_j + Nx_{ij} \le N-1,\ i,j=\overline{1,N}, i \ne j,$$

$$0 \le l_k \le Q,\ l_k = \sum_{i=1}^{N} q_i x_{ijk},\ j=\overline{1,N},\ k=\overline{1,m}, \tag{6}$$

де x_{ijk} – безпосереднього наявність переходу з i-го пункту в j-й для k-го комівояжера, $x_{ij} \in \{0,1\}$; Q – однакова для всіх вантажопідйомність комівояжера; l_k – накопичена вага вантажу, яку несе k-й комівояжер; q_i – запит на товар для i-го пункту.

Обмеження (5) задає умову того, щоб кожний комівояжер відвідав хоча б однин пункт, а (6) обмежує максимальну довжину маршруту за накопиченою вагою вантажу.

Для оптимізаційних моделей (1)–(3) і (4)–(6) множина пунктів ЛМ подається [9] у вигляді повнозв'язного графа, вага ребер якого задається квадратною матрицею $D=[d_{ij}]$ розміром $N \times N$ (де d_{ij} – відстань, витрати або час проїзду між i-м та j-м пунктами, $d_{ij} \ge 0,\ i,j=\overline{1,N}$, $d_{ii}=0\ \forall i=\overline{1,N}$). Матриця $D=[d_{ij}]$ може бути як симетричною ($d_{ij}=d_{ji}\ \forall i,j=\overline{1,N}$), так і несиметричною ($d_{ij} \ne d_{ji}\ \forall i,j=\overline{1,N}$).

Якщо вершини графа задані точками на площині з координатами x_i та y_i, $i=\overline{1,N}$, то, у залежності від особливості території та приміщень ВП, вага ребер $D=[d_{ij}]$ визначається за допомогою евклідової (7), манхетенської (8) або інших метрик відстані:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, \tag{7}$$

$$d_{ij} = |x_i - x_j| + |y_i - y_j|. \tag{8}$$

Задачу MTSPC (4)-(6) можна звести до задачі TSP (1)–(3), змінивши вхідну матрицю ваг наступним чином [3]. Нехай $D'=[d'_{ij}],\ i,j=\overline{1,N+1}$ – матриця ваг задачі m комівояжерів, у

якій пункт виїзду комівояжера має номер $N+1$. Введемо $m-1$ фіктивну вершину з номерами $N+2,\ldots,N+m$ та приймемо значення d'_{ij} рівними:

$$d'_{ij}=d_{ij},\ i,j=\overline{1,N+m},\quad d'_{i,N+j}=d_{i,N+1},\ i=\overline{1,N+m},\ j=\overline{2,m},\qquad (9)$$
$$d'_{N+i,j}=d_{N+1,j},\ j=\overline{1,N+m},\ i=\overline{2,m},\quad d'_{N+i,N+j}=\infty,\ i,j=\overline{2,m}.$$

Таким чином, вирішивши задачу TSP для матриці $D'=[d'_{ij}],\ i,j=\overline{1,N+m}$ вирішимо задачу MTSP з m комівояжерами. В кінці необхідно змінити всі пункти маршруту з номерами, які більше N, на один з номером $N+1$.

ЗАСТОСУВАННЯ МЕТОДУ ГІЛОК ТА МЕЖ ДЛЯ ВИРІШЕННЯ TSP-ЗАДАЧІ. Методи розв'язання задач (1)–(3) та (4)–(6) класифікуються як точні та евристичні (наближені) [3, 10]. Точні методи гарантують знаходження оптимального маршруту з мінімальною загальною вартістю, однак їх використання може бути обчислювально високо вимогливим, особливо для великих наборів даних. Евристичні методи, у свою чергу, здатні розв'язувати задачі великих масштабів з урахуванням багатьох додаткових обмежень, проте вони не завжди забезпечують високу точність отриманих рішень. Для розв'язання задач виробничої мікрологістики пропонується використати метод гілок та меж (Branch and Bound, B&B). Він передбачає реалізацію універсального підходу до вирішення оптимізаційних задач. У ньому здійснюється систематичний перегляд усіх потенційних рішень, одночасно звужуючи простір пошуку шляхом оцінювання верхньої та нижньої допустимої межі. Існує адаптована версія методу гілок та меж [11], для ефективного вирішення TSP-задач. Цей варіант алгоритму має назву «метод Літтла» і може бути представлений послідовністю наступних кроків.

Крок 1. Виконуємо приведення матриці $D=[d_{ij}]$ по рядкам: для кожного рядка матриці знаходимо мінімальний елемент і віднімаємо його значення від всіх елементів цього рядка. У разі, якщо у приведеній матриці є стовпці без нульових елементів, тоді приводимо її по стовпцях.

Крок 2. Обчислюємо константу приведення (10), яка буде нижньою межею множини усіх гамільтонових циклів.

$$Lb=\sum_{i=1}^{N}\min(row_i)+\sum_{j=1}^{N}\min(column_j),\qquad (10)$$

де Lb – нижня межа для вузла; $\min(row_i)$ – мінімальний елемент i-го рядка після редукції; $\min(column_j)$ – мінімальний елемент j-го рядка після редукції.

Крок 3. Для кожного нульового елемента приведеної матриці розрахуємо степінь нульового елемента. Для цього нуль в матриці замінюємо на знак «$*$» і знаходимо суму мінімальних елементів рядка і стовпця, які відповідають цьому нулю.

Крок 4. Вибираємо ребро (i_0, j_0), для якого степінь нульового елемента досягає максимального значення.

Крок 5. Розбиваємо множину всіх гамільтонових циклів Ω^0 на дві підмножини Ω_{i_0,j_0} та $\Omega^*_{i_0,j_0}$. Підмножина Ω_{i_0,j_0} включає в себе ребро (i_0, j_0), а $\Omega^*_{i_0,j_0}$ його не містить. Для отримання матриці вартості для підмножини Ω_{i_0,j_0} викреслюємо рядок i_0 та стовпець j_0.

Крок 6. Приводимо матрицю гамільтонових циклів Ω_{i_0,j_0} та розраховуємо нижню межу даної множини $\varphi(\Omega_{i_0,j_0})$. Аналогічні дії виконуємо для множини гамільтонових циклів $\Omega^*_{i_0,j_0}$.

Крок 7. Порівнюємо нижні межі підмножин гамільтонових циклів Ω_{i,j_0} і Ω^*_{i,j_0}. Якщо $\varphi(\Omega_{i,j_0}) < \varphi(\Omega^*_{i,j_0})$, то подальшому галуженню підлягає множина Ω_{i,j_0}, інакше – Ω^*_{i,j_0}.

Крок 8. Якщо в результаті розгалужень отримуємо матрицю 2×2, то визначаємо гамільтоновий цикл, який отримано розгалуженням, та розраховуємо його довжину $f(x)$.

Крок 9. Порівнюємо довжину гамільтонового циклу $f(x)$ з нижніми межами обірваних гілок. Якщо довжина $f(x)$ не перевищує їхніх нижніх меж, то задача вирішена. В іншому випадку розбиваємо гілки підмножин з нижньою межею, меншою отриманого шляху, до тих пір, поки не отримаємо маршрут з меншою довжиною або не переконаємося, що такого не існує.

Перевага цього методу полягає у забезпеченні точного рішення TSP-задачі, при цьому враховуючи її специфіку. Проте, основним недоліком методу є його висока обчислювальна вимогливість при розв'язанні задач з відносно великою розмірністю.

ВИСНОВКИ. У рамках даної роботи було проведено аналіз задачі оптимізації маршрутів у логістичних мережах виробничих процесів, з особливим акцентом на мікрологістичні мережі. Наведено оптимізаційні моделі для TSP- та MTSPC-задач, які можуть бути використані для реінжинірингу маршрутів логістичних мереж на виробництві. Оптимізація логістичних мереж в цілому дозволить підвищити ефективність виробництва шляхом забезпечення координації процесів та зниження витрат на логістику. Особлива увага була приділена аналізу алгоритму зведення MTSP- до TSP-задачі, що дозволяє ефективніше вирішувати складні логістичні задачі. Розгляд методу гілок та меж як способу розв'язання TSP-задачі підкреслив його ефективність для знаходження точного розв'язку, проте з високими обчислювальними вимогами для задач з великою розмірністю.

У подальших дослідженнях доцільним є дослідження та використання наближених (вставки, генетичних алгоритмів тощо) й інтерактивного методів розв'язання TSP- та MTSP-задач з великої розмірності для нових задач маршрутизації [12].

ЛІТЕРАТУРА

1. Roghanian E., Pazhoheshfar P. An optimization model for reverse logistics network under stochastic environment by using genetic algorithm. Journal of manufacturing systems. 2014. Vol. 33, no. 3. P. 348–356.

2. Логістика : навчальний посібник / О. В. Безсмертна, О. О. Мороз, Т. М. Білоконь та І. В. Шварц. Вінниця: ВНТУ, 2018.

3. V. Beskorovainyi, O. Kuropatenko and D. Gobov. Optimization of transportation routes in a closed logistics system. Innovative Technologies and Scientific Solutions for Industries, 2019. No. 4 (10). P. 24-32.

4. Beskorovainyi V., Sudik A. Optimization of topological structures of centralized logistics networks in the process of reengineering. Innovative Technologies and Scientific Solutions for Industries. 2021. No. 1 (15). P. 23–31.

5. Безкоровайний В. В., Нефьодов Л. І., Русскін В. М. Математична модель структурно-топологічної оптимізації логістичних мереж // Вісник Харківського національного автомобільно-дорожнього університету, 2021. Вип. 95. С. 178–184.

6. Beskorovainyi V., Draz O. Mathematical models of decision support in the problems of logistics networks optimization. Innovative Technologies and Scientific Solutions for Industries, 2021, No. 4 (18). P. 5–14.

7. Systemomological analysis of the problem of optimizing the network of main center of the special monitoring objects / V. V. Bezkorvayny et al. Проблеми створення, випробування,

застосування та експлуатації складних інформаційних систем. 2018. P. 50–62.

8. Семаньків М., Париляк О. NP-повні задачі та їх алгоритми. Інформаційні технології та комп'ютерне моделювання: матеріали міжнар. науково-практ. конф., Івано-Франківськ, 2023. С. 51-53.

9. Бевзюк А., Гончар В. Побудова і використання матриць суміжності і матриць відстаней. Прикладні інформаційні технології: Матеріали IV Всеукр. науково-практ. конф. студентів, аспірантів та молодих вчен., Вінниця, 2023. С. 105-110.

10. Zhang J. Comparison of various algorithms based on TSP solving», Journal of physics: conference series, 2021. Vol. 2083. P. 116-119,

11. Little J. D. C. et al. An algorithm for the traveling salesman problem. Operations research. 1963. Vol. 11. No. 6. P. 972–989.

12. Poikonen S., Golden B., Wasil E. A. A branch-and-bound approach to the traveling salesman problem with a drone. INFORMS journal on computing. 2019. Vol. 31. No. 2. P. 335–346.

Науковий керівник: *Безкоровайний Володимир Валентинович, професор, д.т.н., професор кафедри КІТАР Харківського національного університету радіоелектроніки.*

Міністерство освіти і науки України

Харківський національний автомобільно-дорожній університет

КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ АВТОМАТИЗАЦІЇ ТЕХНОЛОГІЧНИХ ПРОЦЕСІВ НА ТРАНСПОРТІ ТА У ВИРОБНИЦТВІ

МАТЕРИАЛИ ВСЕУКРАЇНСЬКОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ ЗДОБУВАЧІВ ВИЩОЇ ОСВІТИ І МОЛОДИХ УЧЕНИХ

22 листопада 2023 р.

Харків 2023

УДК 519.81:658.512

ФОРМАЛІЗАЦІЯ ЗАДАЧІ ОПТИМІЗАЦІЇ МАРШРУТІВ ЛОГІСТИЧНОЇ МЕРЕЖІ ВИРОБНИЧОГО ПРОЦЕСУ

Чернишенко О. В., Безкоровайний В. В.

Харківський національний університет радіоелектроніки, Харків

Виробнича логістика на сучасних підприємствах виступає одним з ключових елементів для створення ефективних та гнучких технологічних процесів. Розвиток технологій і зміни попиту споживачів вимагають відповідної адаптації виробничих процесів (ВП) та оптимізації їх логістичних мереж (ЛМ). З огляду на це, пошук шляхів оптимізації логістичних мереж стає не тільки шляхом покращення оперативної діяльності, але й необхідністю для забезпечення довгострокової конкурентоспроможності виробника на ринку.

Оптимізація ЛМ виробничого процесу включає комплекс заходів, спрямованих на поліпшення різних аспектів виробничої та розподільчої підсистем з метою зниження витрат та підвищення їх ефективності [1]:

– визначення оптимальної кількості та місцезнаходження виробничих підприємств, складів та розподільних центрів, що забезпечує мінімізацію транспортних витрат та часу доставки;

– планування та оптимізація маршрутів перевезень матеріалів на виробництві з метою зменшення логістичних витрат;

– синхронізація роботи різних виробничих ліній для оптимізації навантаження на обладнання та уникнення простоїв;

– оптимізація системи управління запасами для зниження витрат на зберігання і зменшення часу зберігання сировини та готової продукції.

Розрізняють задачі оптимізації макро- і мікрологістичних мереж [2]. Задачі макрологістики виникають при оптимізації ЛМ великомасштабних компаній. При розв'язанні задач мікрологістики здійснюється адаптація існуючих мереж, організованих на відносно невеликих територіях, шляхом оптимізації множини кільцевих маршрутів.

Задачу оптимізації маршрутів мікрологістики пропонується узагальнити та подати як задачу комівояжера (TSP – Travelling Salesman Problem). Суть класичної TSP-задачі полягає у знаходженні маршруту мінімальної ваги X^o, який проходитиме через задану множину пунктів ВП, відвідуючи кожен пункт лише один раз і повертаючись до початкового пункту.

Модель оптимізації маршрутів мікрологістики як TSP-задачі [3]:

$$X^o = arg\ \min_X \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}x_{ij},\quad \sum_{i=1}^{N} x_{ij} = 1,\ \ j=\overline{1,N},\quad \sum_{j=1}^{N} x_{ij} = 1,\ \ i=\overline{1,N}, \qquad (1)$$

$$u_i - u_j + N\cdot x_{ij} \le N-1,\ \ i,j=\overline{2,N},\ \ i\ne j, \qquad (2)$$

де X – матриця, що визначає маршрут обходу пунктів ВП (x_{ij} – наявність безпосереднього переходу з i-го пункту в j-й, $x_{ij}\in\{0,1\}$); N – кількість пунктів мережі; d_{ij} – вага (відстань, вартість) переходу з i-го пункту в j-й; u_i – допоміжні змінні, $u_i \ge 0$, $i=\overline{1,\ N}$.

Модель (1)-(2) зорієнтована на задачі з відносно невеликою кількістю пунктів і не враховує обмежень (обсяги поставок, ємність транспортних засобів, час доставки тощо). При необхідності врахування таких обмежень пропонується подавати задачу оптимізації ЛМ як задачу декількох комівояжерів з обмеженнями (MTSPC – Multiple TSP with Constraint). У цій задачі m комівояжерів мають обійти всі пункти по одному разу з обов`язковим поверненням до старту, дотримуючись заданих обмежень.

Модель оптимізації маршрутів мікрологістики як MTSPC-задачі [3]:

$$X^o = arg\ \min_X \sum_{k=1}^{m}\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}x_{ijk}, \qquad (3)$$

$$\sum_{k=1}^{m}\sum_{i=1}^{N} x_{ij} = 1,\ \ j=\overline{1,N},\quad \sum_{k=1}^{m}\sum_{j=1}^{N} x_{ij} = 1,\ \ i=\overline{1,N},\quad \sum_{k=1}^{m} x_{ijk} \ge 1,\ \ i=\overline{1,N},\ j=\overline{1,N}, \qquad (4)$$

$$0 \le \sum_{i=1}^{N} q_i x_{ijk} \le Q,\ j=\overline{1,N},\ \ k=\overline{1,m};\ u_i - u_j + N\cdot x_{ij} \le N-1,\ \ i,j=\overline{2,N},\ \ i\ne j,$$

де x_{ijk} – безпосереднього наявність переходу з i-го пункту в j-й для k-го

комівояжера, $x_{ij} \in \{0,1\}$; Q – однакова для всіх вантажопідйомність комівояжера; l_k – накопичена вага товару, яку несе k-й комівояжер; q_i – запит на товар для i-го пункту.

У моделях (1)-(2) і (3)-(4) враховуються такі припущення [4]: множина пунктів ЛМ подається у вигляді повнозв'язного графа; вага ребер задається квадратною матрицею $D=[d_{ij}]$ розміром $N \times N$ (де d_{ij} – відстань (витрати, час проїзду) між i-м та j-м пунктами, $d_{ij} \geq 0$, $i,j=\overline{1,N}$, $d_{ii}=0$ $\forall i=\overline{1,N}$); матриця $D=[d_{ij}]$ може бути як симетричною ($d_{ij}=d_{ji}$ $\forall i,j=\overline{1,N}$), так і несиметричною ($d_{ij} \neq d_{ji}$ $\forall i,j=\overline{1,N}$).

Якщо вершини графа подаються точками на площині з координатами x_i та y_i, $i=\overline{1,N}$, то вага ребер $D=[d_{ij}]$ визначається за допомогою евклідової (5), манхетенської (6) або інших метрик відстані:

$$d_{ij}=\sqrt{(x_i-x_j)^2+(y_i-y_j)^2}, \tag{5}$$

$$d_{ij}=|x_i-x_j|+|y_i-y_j|. \tag{6}$$

Задачу MTSPC (3)-(4) можна звести до задачі TSP (1)-(2), змінивши вхідну матрицю ваг наступним чином [4-5]. Нехай $D'=[d'_{ij}]$, $i,j=\overline{1,N+1}$ – матриця ваг задачі m комівояжерів, у якій пункт виїзду комівояжера має номер $N+1$. Введемо $m-1$ фіктивну вершину з номерами $N+2,\ldots,N+m$ та приймемо значення d'_{ij} рівними:

$$d'_{ij}=d_{ij},\ i,j=\overline{1,N+m},\quad d'_{i,N+j}=d_{i,N+1},\ i=\overline{1,N+m},\ j=\overline{2,m}, \tag{7}$$

$$d'_{N+i,j}=d_{N+1,j},\ j=\overline{1,N+m},\ i=\overline{2,m},\quad d'_{N+i,N+j}=\infty,\ i,j=\overline{2,m}.$$

Таким чином, вирішивши задачу TSP для матриці $D'=[d'_{ij}]$, $i,j=\overline{1,N+m}$ вирішимо задачу MTSP з m комівояжерами. В кінці необхідно змінити всі пункти маршруту з номерами, які більше N, на один з номером $N+1$.

Методи розв'язання задач (1)-(2) і (3)-(4) поділяють на точні та наближені [5-6]. Точні методи забезпечують визначення ідеального маршруту з найменшою сумарною вартістю, проте їх застосування може бути обчислювально складним при великих обсягах даних. Наближені методи дозволяють розв'язувати задачі великих розмірів з урахуванням числених додаткових обмежень, проте не гарантують достатньої точності

розв'язків.

Запропонована формалізація задач оптимізації маршрутів дозволить підвищувати ефективність виробничих процесів за рахунок забезпечення їх ритмічності та скорочення логістичних витрат.

Література:

1. Логістика : навчальний посібник / О. В. Безсмертна, О. О. Мороз, Т. М. Білоконь та І. В. Шварц. Вінниця: ВНТУ, 2018.
2. В. В. Безкоровайний, Л. І. Нефьодов, В. М. Русскін, «Математична модель структурно-топологічної оптимізації логістичних мереж», Вісник Харківського національного автомобільно-дорожнього університету, вип. 95, с. 178-184, 2021.
3. М. Семаньків та О. Париляк «NP-повні задачі та їх алгоритми», Інформаційні технології та комп'ютерне моделювання: матеріали міжнар. науково-практ. конф., Івано-Франківськ, 2023, с. 51-53.
4. А. Бевзюк та В. Гончар, «Побудова і використання матриць суміжності і матриць відстаней», Прикладні інформаційні технології: Матеріали IV Всеукр. науково-практ. конф. студентів, аспірантів та молодих вчен., Вінниця, 2023, с. 105-110.
5. V. Beskorovainyi, O. Kuropatenko and D. Gobov, "Optimization of transportation routes in a closed logistics system", Innovative Technologies and Scientific Solutions for Industries, no. 4 (10), pp. 24-32, 2019.
6. J. Zhang, "Comparison of various algorithms based on TSP solving", Journal of physics: conference series, vol. 2083, pp. 116-119, 2021.

МАТЕРІАЛИ XXVII
МІЖНАРОДНОГО
МОЛОДІЖНОГО ФОРУМУ

МІНІСТЕРСТВО
ОСВІТИ І НАУКИ
УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ

2023

ТОМ 2

ХАРКІВ

УДК 681.518

ФОРМАЛІЗАЦІЯ ЗАДАЧІ РЕІНЖИНІРИНГУ ВНУТРІШНЬОЗАВОДСЬКОЇ ЛОГІСТИКИ

Чернишенко О.В.
Науковий керівник – д.т.н., проф. Безкоровайний В.В.
Харківський національний університет радіоелектроніки, каф. КІТАМ, м. Харків, Україна
тел. +38(099) 478-19-67, e-mail: oleksandr.chernyshenko@nure.ua.

The work involves conducting a comprehensive examination of the logistics operations within the enterprise. This enables the identification and formalization of tasks for managing the enterprise's transportation services, which are designed to establish more efficient cargo transportation schemes. By implementing a well-structured in-house logistics system, businesses can effectively minimize their production expenses while simultaneously enhancing their overall productivity levels. In this regard, the choice of ways and means of improving the management of intra-plant logistics processes is of particular relevance.

Умови конкуренції орієнтують сучасні виробничі компанії на необхідність швидких змін номенклатури й обсягів виробництва. Це потребує швидкого переналагодження технологічних процесів і відповідних змін внутрішньозаводської логістики. При виникненні суттєвої невідповідності поточним вимогам виробництва здійснюється реінжиніринг логістичної системи підприємства. Метою реінжинірингу є оптимізація логістичних процесів підприємства з урахуванням існуючих ресурсів для зберігання і транспортування сировини, комплектуючих, готової продукції. Реінжиніринг логістичної мережі передбачає вирішення комплексу задач її структурної, параметричної, топологічної, технологічної, оптимізації, включаючи багатофакторну оцінку та вибір системних рішень [1]. Метою дослідження є формалізація задачі реінжинірингу внутрішньозаводської логістики.

Введемо позначення [2]: X – множина вхідних змінних; Y – множина вихідних змінних; U – множина змінних керуючого впливу. Замовлення на перевезення вантажів будемо розглядати як складові множини вхідних змінних: $X = \{z_i\}, i = \overline{1,n}$ (де z_i – i-те замовлення на перевезення; n – кількість замовлень). Кожне замовлення z_i характеризується: часом доставки t_i; обсягом вантажу v_i; типом вантажу q_i; пунктом відправлення a_i, кінцевим пунктом перевезення b_i. З урахуванням цього інформація щодо i-го замовлення може бути подана у вигляді: $z_i = (t_i, v_i, q_i, a_i, b_i)$.

Позначимо множину видів транспорту, що можуть бути використані для перевезення різних вантажів як $T = \{T_j\}$, $j = \overline{1,l}$; $T_j = \{t_{jg}\}$, $g = \overline{1,d}$ (де

T_j – множина транспортних засобів j-го виду; t_{jg} – g-й транспортний засіб j-го виду; l – кількість видів транспорту; d – кількість транспортних засобів j-го виду).

Зазвичай існують різні способи перевезення вантажу з пункту a в пункт b. Позначимо множину маршрутів, що сполучають початковий і кінцевий пункти як $M_{ab} = \{m_c^{ab}\}$, $c = \overline{1,k}$ (де m_c^{ab} – c-й маршрут, що сполучає пункти a та b; k – кількість маршрутів). Як складові множини вихідних змінних виступають результати роботи відділу транспортного забезпечення з пошуку оптимального маршруту $Y = \{h, s\}$ (де h – вартість перевезення вантажу; s – сумарний чи максимальний час перевезення).

В якості керуючих змінних в моделі системи керування перевезеннями виступають вид транспортного засобу t_j, що використовується для перевезення, та маршрут m_c^{ab} транспортування з пункту a в пункт b: $U = \{t_j, m_c^{ab}\}$.

Задача оперативного керування внутрішньозаводською логістикою передбачає своєчасне і безперебійне матеріальне забезпечення виробничих цехів та складських служб таким чином, щоб витрати на транспортування h та час на доставку вантажу s були мінімальними [2]:

$$k_1(X,T,M_{ab}) = h(X,T,M_{ab}) \to \min;\ k_2(X,T,M_{ab}) = s(X,T,M_{ab}) \to \min. \quad (1)$$

Для одночасної оптимізації за двома показниками (1) пропонується використати адитивну модель якості варіантів, у якій використовуються нормовані значення витрат $\overline{k}_1(X,T,M_{ab})$ і часу $\overline{k}_2(X,T,M_{ab})$:

$$P(X,T,M_{ab}) = \sum_{i=1}^{2} \lambda_i \overline{k}_i(X,T,M_{ab}) \to \max. \quad (2)$$

де λ_1 і λ_2 – вагові коефіцієнти показників витрат і часу транспортування.

В задачі (2) враховуються обмеження на максимальний час транспортування, вартість перевезення та кількість транспортних засобів.

Запропонована формалізація двокритеріальної задачі реінжинірингу системи перевезень дозволить підвищити ефективність системи внутрішньозаводської логістики. Напрямом подальших досліджень може бути розробка методів оптимізації маршрутів внутрішньозаводських перевезень.

Список використаних джерел:

1. Beskorovainyi, V., & Sudik, A. (2021). Optimization of topological structures of centralized logistics networks in the process of reengineering. Innovative Technologies and Scientific Solutions for Industries, (1 (15)), 23-31.

2. Шептура, О. О., & Дядик, Л. С. (2012). Задачі управління транспортним забезпеченням підприємства. Штучний інтелект, 387-391.

ДОДАТОК В

Демонстраційний матеріал

№	Позначення	Найменування	Примітки
		Текстові документи	
1	ГЮИК 504310.012 ПЗ	Пояснювальна записка	105 с., ф. А4
2		Додаток А	
		Текст програми	
		Графічні документи	
3		Додаток Б	
		Апробація наукових результатів	
4		Додаток В	
		Демонстраційний матеріал	
5		Презентація	у форматі
			*.ppt

					ГЮИК 504310.012 ВД			
Змін.	Арк.	№ документа	Підпис	Дата				
Розроб.		Чернишенко О. В.			Розроблення підсистеми підтримки прийняття рішень для оптимізації маршрутів внутрішньозаводської логістики	Літ.	Аркуш	Аркушів
Перевір.		Безкоровайний В. В.					1	1
Н. контр.		Бронніков А. І.						
Затв.		Невлюдов І. Ш.						