УДК 303.732.4

ВЫБОР АЛГОРИТМА ОПТИМИЗАЦИИ ДЛЯ РАСПРЕДЕЛЕНИЯ РАБОТ МЕЖДУ СОТРУДНИКАМИ СПЕЦИАЛИЗИРОВАННОГО МАГАЗИНА

Берсенева Валерия Александровна –
аспирант
Кубанский государственный технологический университет, Краснодар, Россия

В статье рассмотрены различные алгоритмы оптимизации, выполнен анализ полученных результатов и на его основании сделан вывод о применении определенного алгоритма. В итоге построен наглядный график работы сотрудников специализированного магазина

Ключевые слова: АЛГОРИТМЫ ОПТИМИЗАЦИИ, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, ГРАФИК РАБОТЫ ПЕРСОНАЛА, ФИТНЕС-ФУНКЦИЯ, УПРОЩЕННЫЙ АЛГОРИТМ

UDC 303.732.4

CHOICE OF OPTIMIZATION ALGORITHM FOR DISTRIBUTION OF WORK BETWEEN CO-WORKERS OF A SPECIALTY STORE

Berseneva Valeria Aleksandrovna
Post-graduate student
Kuban State Technological University, Krasnodar, Russia

In this article various algorithms of optimization are considered, the analysis of the received results and the conclusion on its basis is drawn on application of certain algorithm. As a result the evident schedule of work of employees of specialized shop is constructed

Keywords: OPTIMIZATION ALGORITHM, GENETIC ALGORITHM, SCHEDULE OF WORK OF EMPLOYEES, FITNESS FUNCTION, SIMPLIFIED ALGORITHM

Грамотное построение графика работ для сотрудников специализированного магазина требует решения оптимизационной задачи. Это проблемы оптимального распределения ресурсов и планирования.

Традиционно оптимизационные задачи решаются при помощи линейных и/или нелинейных методов оптимизации, которые обычно предполагают сведение к минимуму целевой функции. В сущности, считается, что задача оптимизации является минимизация проблемы.

На практике существует много задач, которые не могут быть описаны аналитически, например, когда целевая функция имеет несколько экстремумов. В этих случаях необходимо создавать несколько экстремумов глобальной задачи оптимизации, где традиционные методы оптимизации не применяются, а другие решения должны быть исследованы.

Целью статьи является построение наглядного графика работ сотрудников специализированного магазина. Для этого в статье рассмотрим два алгоритма оптимизации, выполним анализ полученных

результатов и на его основании сделаем вывод о применении определенного алгоритма.

Общая постановка задачи

В современном магазине при грамотном управлении в определенный момент встает задача о построении графика работ сотрудников. Как правило, магазины работают по 12-24 часа в сутки, а рабочий день сотрудника, в идеале, должен составлять не более 8 часов.

После 3-х месяцев работы магазина, управляющий уже может самостоятельно (либо с помощью программы) составить примерный рабочий график магазина наподобие таблицы 1. По горизонтали указано время работы магазина, разбитое по одному часу. По вертикали указаны виды работ, которые необходимо выполнять сотрудникам магазина. В ячейках таблицы указано количество человеко-часов.

Обычным шрифтом указано количество человеко-часов, которое можно переставить на другой час, если того требует ситуация. Жирным курсивом выделено кол-во человеко-часов в приоритетной работе. То есть, на другой час нельзя переносить этот вид работ.

Таблица 1 – Примерная таблица одного рабочего дня магазина

	Понедельник												
Время / Виды работ	9.00-10.00	10.00-11.00	11.00-12.00	12.00-13.00	13.00-14.00	14.00-15.00	15.00-16.00	16.00-17.00	17.00-18.00	18.00-19.00	19.00-20.00	20.00-21.00	21.00-22.00
1.Работа с поставщиками	***2чч***	***2чч***	***1чч***										
2. Выкладка товара		1чч	2чч		1чч	1чч		2чч				***1чч***	
3.Работа на кассе		***2чч***	***2чч***	***2чч***	***2чч***	***1чч***	***1чч***	***2чч***	***2чч***	***2чч***	***3чч***	***3чч***	***1чч***
4. Помощник покупателям		***2чч***	***2чч***	***1чч***	***1чч***	***1чч***	***1чч***	***2чч***	***2чч***	***3чч***	***3чч***	***3чч***	
5.Инкассация													***1чч***
6.Уборка	***1чч***						1чч						

помещения													

Для управляющего магазина нужна другая, более наглядная форма:

Таблица 2 – Желаемая примерная выходная форма графика

	Понедельник												
Время / Абстрактный сотрудник	9.00-10.00	10.00-11.00	11.00-12.00	12.00-13.00	13.00-14.00	14.00-15.00	15.00-16.00	16.00-17.00	17.00-18.00	18.00-19.00	19.00-20.00	20.00-21.00	21.00-22.00
	1	***1***	***1***	***3***	2	2	***3***	2					
	1	***1***	***3***	***3***	***3***	***3***	***4***	2					
	6	***3***	***3***	***4***	***3***	***4***	6	***3***				**2**	
		3	***4***	2	***4***	***2***		***3***	***3***		***3***	***3***	
		4	***4***	2				***4***	***3***	3	***3***	***3***	
		4						***4***	***4***	***3***	***3***	***3***	
									4	***3***	***4***	***4***	
										4	***4***	***4***	***3***
										4	***4***	***4***	***5***

Итак, основной задачей статьи является программная реализация перехода от первой таблицы ко второй. Представим таблицу 1 в более наглядном виде:

Таблица 3 — Виды работ на каждый час

	Понедельник												
Время / Абстрактный сотрудник	9.00-10.00	10.00-11.00	11.00-12.00	12.00-13.00	13.00-14.00	14.00-15.00	15.00-16.00	16.00-17.00	17.00-18.00	18.00-19.00	19.00-20.00	20.00-21.00	21.00-22.00
		1	1									2	
		1	2					2			3	3	
		2	2					2		3	3	3	
		3	3		2			3	3	3	3	3	
	1	3	3	3	3	2	3	3	3	4	4	4	
	1	4	4	3	3	3	4	4	4	4	4	4	3
	6	4	4	4	4	4	6	4	4	4	4	4	5

Где закрашенная ячейка соответствует одному человеко-часу, а номер в ячейке — виду работ из табл. 1. Ячейки с тёмным фоном переносить по времени нельзя, со светлым фоном — можно.

Таблица 3 отражает виды работ на конкретные часы. Очевидно, что равномерное распределение работ в таблице и будет наиболее оптимальным графиком работ.

Максимальное приближение к равномерному распределению есть первая задача. Для более наглядного отображения разделяем приоритетные и не приоритетные работы. В данном случае наиболее оптимальный график будет выглядеть так:

Таблица 4 — График работ с разделением по приоритету

	Понедельник												
Время	9.00-10.00	10.00-11.00	11.00-12.00	12.00-13.00	13.00-14.00	14.00-15.00	15.00-16.00	16.00-17.00	17.00-18.00	18.00-19.00	19.00-20.00	20.00-21.00	21.00-22.00
Абстрактный сотрудник													
		2	2									2	
		1	2					2			3	3	
		1	1					2		3	3	3	
		3	3		2			3	3	3	3	3	
	1	3	3	3	3	2	6	3	3	4	4	4	
	1	4	4	3	3	3	3	4	4	4	4	4	3
	6	4	4	4	4	4	4	4	4	4	4	4	5

Формулировка задачи оптимизации

Описание задачи:

$$\begin{matrix} N_d \quad min \\ T_e \leq 8 \\ Tfull_e \leq 10 \end{matrix} \qquad (1)$$

В данном случае 8 и 10 – это не фиксированные значения, а наиболее предпочтительные. Допускается отклонение на 1 час.

Где:

- N_d - количество сотрудников на текущий день
- T_e - время работы сотрудника без учёта перерывов
- $Tfull_e$ - время работы сотрудника с учётом перерывов. Может меняться в зависимости от политики компании.

Есть еще условие, которое рекомендуется выполнять:

Количество перерывов — минимально. Стараться сделать не больше одного.

Ввиду сложности представления функции, и похожести на задачу о ранце, выберем для реализации нашей задачи генетический алгоритм.

Формулировка задачи для генетических алгоритмов

Для решения данной комбинаторной задачи сначала представим график в самом худшем виде, когда на каждого сотрудника выделяется всего один час работы (см. таблицу 5).

Таблица 5 — График работ в наихудшем случае.

	Понедельник												
Время / Абстрактный сотрудник	9.00-10.00	10.00-11.00	11.00-12.00	12.00-13.00	13.00-14.00	14.00-15.00	15.00-16.00	16.00-17.00	17.00-18.00	18.00-19.00	19.00-20.00	20.00-21.00	21.00-22.00
	1												
	1												
	6												
		2											
		1											
…													
												4	
												4	
													3
													5

Позволим блокам в таблице перемещаться вертикально (для всех блоков) и горизонтально (только для светлых блоков). Таким образом

получим хромосому из ряда смещений по координатным осям, где для тёмных блоков используется одна координата, а для светлых две.

Фитнес-функция будет показывать б**о**льшие значения для меньших N_d и 0 при превышении T_e и $Tfull_e$.

Формулировка задачи для упрощенного алгоритма

Упрощенный алгоритм — расстановка часов в таблице каждому сотруднику вручную. Главная задача оптимизации — разбиение таблицы 4 на 8-ми часовые периоды для приведения к таблице 2.

Возьмем таблицу 4 и снизу будем брать по 8 клеток горизонтально. И так для каждого сотрудника. Если не хватает до 8 клеток, то берем сверху рядом.

Результаты применения генетического алгоритма

Были проведены исследования с помощью следующих параметров:

•численность популяции: 40000 хромосом

•количество брачных пар: 10000

•вероятность побитового скрещивания: 0.04

•вероятность мутации бита: 0.01

•вероятность инверсии: 0.01

•дистанция поиска хромосомы при инбридинге : 10 хромосом

•дистанция поиска хромосомы при аутбридинге: 50 хромосом

•период использования аутбридинговых операций:15 поколений

Экспериментально выяснено, что на текущих данных, при популяции численности менее 30 000 хромосом улучшение результатов не происходит. Алгоритм расчёта по генетическому алгоритму показан в листинге 1:

Листинг 1: Описание фитнес-функции (на языке Java).

```
/**
 * фитнес-функция
 * @param numbers chromosome numbers список значений хромосомы
 * @return приспособленность данной хромосомы
 */
public double f(char[] numbers) {
    // промежуточная таблица графика дня
    final ArrayList<int[]> intermediateData = new ArrayList<int[]>();

    // заполняем промежуточную таблицу по данным хромосомы
```

```
    fillData(numbers, intermediateData);

    double totalFitness = 0;
    for (final int[] employeeDay : intermediateData) {

        int workHours = 0;                          // количество рабочих часов сотрудника
        int firstHour = MAX_COLUMN_COUNT;           // начало рабочего дня у сотрудника
        int endHour = 0;                            // последний час рабочего дня сотрудника
        int holes = 0;                              // количество перерывов во время рабочего
дня

        {// вычисляем часовые данные
            boolean wasBreak = false;
            for (int hourIdx = 0; hourIdx < employeeDay.length; ++hourIdx) {
                final int cell = Math.abs(employeeDay[hourIdx]);
                if (cell != 0) {
                    ++workHours;

                    if (wasBreak) {
                        wasBreak = false;
                        ++holes;
                    }

                    if (firstHour > hourIdx) firstHour = hourIdx;
                    if (endHour < hourIdx) endHour = hourIdx;
                } else if (workHours > 0) {
                    wasBreak = true;
                }
            }
        }

        // вычисляем значение фитнес-функции для одного сотрудника
        double lineValue = (double) workHours / MAX_VALUES_IN_ROW;

        // неприоритетное ограничение: количество перерывов должно быть меньше 2
        if (holes > 1) lineValue *= HOLES_INFLUENCE_K;

        // приоритетное ограничение: количество рабочих часов должно быть не более 8
        final int workHoursDiff = workHours - MAX_VALUES_IN_ROW;
        if (workHoursDiff > 0) lineValue /= OVER_WORK_DAY_BASE_K + workHoursDiff *
workHoursDiff;

        // приоритетное ограничение: рабочий день сотрудника не более 10 часов
        final int totalHoursDiff = endHour - firstHour + 1 - MAX_VALUES_RANGE;
        if (totalHoursDiff > 0) lineValue /= OVER_TOTAL_DAY_BASE_K + totalHoursDiff *
totalHoursDiff;

        // "мини-поощрение" функции за достижение оптимального рабочего дня для
сотрудника
        if (workHours == MAX_VALUES_IN_ROW && totalHoursDiff <= 0)
            lineValue *= FULL_DAY_INFLUENCE_K;
        else if (workHours == MAX_VALUES_IN_ROW - 1 && totalHoursDiff <= 0)
            lineValue *= ALMOST_FULL_DAY_INFLUENCE_K;

        // добавляем полученный результат к общим данным
        totalFitness += lineValue;
    }

    // усредняем данные, делаем зависимость от количества сотрудников,
    // добавляем настроечные коэффициенты
    // и возвращаем результат
    return (EMPLOYEES_INFLUENCE + totalFitness) /
            (intermediateData.size() * OPTIMIZE_EMPLOYEES_K);
```

```
}

// базовые коэффициенты
private static final double HOLES_INFLUENCE_K = 0.8;
private static final double ALMOST_FULL_DAY_INFLUENCE_K = 1.1;
private static final double FULL_DAY_INFLUENCE_K = 1.25;
private static final double OPTIMIZE_EMPLOYEES_K = 2.0;
private static final double EMPLOYEES_INFLUENCE = 8;
private static final double OVER_WORK_DAY_BASE_K = 1.0;
private static final double OVER_TOTAL_DAY_BASE_K = 3.0;

// максимальное количество колонок
private static final int MAX_COLUMN_COUNT = 13;
// максимальное время работы без учёта перерывов
private static final int MAX_VALUES_IN_ROW = 8;
// максимальное время работы сотрудника с учётом перерывов
private static final int MAX_VALUES_RANGE = 10;
// максимальное смещение по времени для задач с низким приоритетом
private static final int MAX_LOW_SHIFT = 4;
```

Результирующая таблица выглядит таким образом.

Таблица 6 — График работы персонала, полученный при помощи генетического алгоритма

	Понедельник												
Время	9.00-10.00	10.00-11.00	11.00-12.00	12.00-13.00	13.00-14.00	14.00-15.00	15.00-16.00	16.00-17.00	17.00-18.00	18.00-19.00	19.00-20.00	20.00-21.00	21.00-22.00
Номер сотрудника													
1	***1***	***3***	***4***	***3***	***3***		2	***3***	***3***				
2		***3***						***4***			***4***	***4***	
3	***6***	***4***	***3***	***4***		***4***	2			***4***		***3***	
4			***1***	***3***			2	***3***	6	***4***	***3***	***2***	
5						2	***3***			***3***	2	***3***	
6		***1***			***3***		***4***	2			***3***		
7			***3***		***4***			***4***	***4***		***3***		
8		***1***							***4***		***4***	***4***	***3***
9	***1***	***4***	***4***			***3***				***4***	***4***	***4***	
10								2	***3***	***3***		***4***	***5***

Итак, общее количество сотрудников = 10. Полученный график близок к оптимальному результату, но не оптимален. Много перерывов. Сотрудникам будет неудобно приходить на работу по 3-4 раза и работать по часу-двум, к примеру.

Результирующую таблицу 6 можно рассматривать как график работ, но прежде чем брать ее за основу, нужно подкорректировать, сделать ее более удобной для сотрудников.

Еще один недостаток генетического алгоритма — требует много ресурсов и времени на расчет графика. Данный график рассчитывался 1 час на процессоре AMD Athlon(tm) 64 Processor 3500+ 2.20 GHz. Соответственно, если сеть состоит из 10 магазинов, то раз в неделю на 10 часов будет происходить расчет графиков. Если 100 магазинов — то 100 часов при такой же мощности процессора.

Результаты применения упрощенного алгоритма

Алгоритм расчёта по упрощённому алгоритму показан в листинге 2:

Листинг 2: Описание упрощенного алгоритма (на языке Java)

```
/**
 * Рассчитывает конечный результат
 *
 * @return рассчитанные данные
 */
public ArrayList<int[]> calc() {
    // заполняем список сотрудников, пока есть свободные часы
    while (intermediateData.size() > 0) {
        addEmployee();
    }

    // заполненную конечную таблицу
    return outputData;
}

/**
 * добавляет график сотрудника к конечным данным
 */
private void addEmployee() {
    // график сотрудника для заполнения
    int[] newLine = new int[MAX_COLUMN_COUNT];

    // количество рабочих часов сотрудника
    int workHours = 0;
    // начало рабочего дня у сотрудника
```

```
        int firstHour = 0;

        // пропускаем пустые ячейки
        for (int skip = 0; skip < MAX_COLUMN_COUNT && intermediateData.get(0)[skip] == 0; ++skip)
{
            firstHour = skip + 1;
        }
        // количество часов в строек должно быть гарантированно больше 0,
        // т.к. пустые строки удаляются сразу после переноса ячеек.
        // Поэтому начальный час будет найден
        assert (firstHour < MAX_COLUMN_COUNT);

        // последний час рабочего дня сотрудника
        int endHour = firstHour;

        // сначала заполняем часы по порядку, пропустив самый малочисленный час
        final int emptiestHour = calcEmptiestHour(firstHour + 1, firstHour + MAX_VALUES_RANGE -
1);
        for (int hour = firstHour; hour < firstHour + MAX_VALUES_RANGE
                    && hour < MAX_COLUMN_COUNT
                    && workHours < MAX_VALUES_IN_ROW; ++hour) {

            if (hour == emptiestHour) continue;     // нашли самый малочисленный час. пропускаем

            // убираем ячейку из временных данных и помещаем её в данные сотрудника
            newLine[hour] = removeCell(hour);
            if (newLine[hour] != 0) {
                endHour = hour;
                ++workHours;
            }
        }

        // пытаемся добавить часы снаружи
        if (workHours < MAX_VALUES_IN_ROW && endHour - firstHour + 1 <
MAX_VALUES_RANGE) {
            int leftExpand = firstHour, rightExpand = endHour;

            // итеративно расширяем область поиска свободного часа
            for (int expand = 1, maxExpand = MAX_VALUES_RANGE - workHours;
                    expand <= maxExpand
                    && workHours < MAX_VALUES_IN_ROW
                    && endHour - firstHour + 1 < MAX_VALUES_RANGE; ++expand) {

                // двигаемся в ширину, если это возможно
                if (leftExpand - 1 >= 0) --leftExpand;
                if (rightExpand + 1 < MAX_COLUMN_COUNT) ++rightExpand;

                // проверяем левую границу
                if (newLine[leftExpand] == 0 && endHour - leftExpand < MAX_VALUES_RANGE) {
                    // расширив границу остались в заданных параметрах.
                    // Пытаемся получить свободный час
                    newLine[leftExpand] = removeCell(leftExpand);
                    if (newLine[leftExpand] != 0) {
                        ++workHours;
                        firstHour = leftExpand;
                    }
                }

                // проверяем правую границу
                if (newLine[leftExpand] == 0
                    && workHours < MAX_VALUES_IN_ROW
                    && rightExpand - firstHour < MAX_COLUMN_COUNT) {
                    // расширив границу остались в заданных параметрах.
```

```
                    // Пытаемся получить свободный час
                    newLine[rightExpand] = removeCell(rightExpand);
                    if (newLine[rightExpand] != 0) {
                        ++workHours;
                        endHour = rightExpand;
                    }
                }
            }
        }

        // не удалось полностью заполнить рабочий день. Попытаемся ещё раз внутри и
снаружи
        if (workHours < MAX_VALUES_IN_ROW) {
            // пытаемся заполнить пустоты внутри
            for (int hour = firstHour+1; hour < endHour
                && workHours <MAX_VALUES_IN_ROW; ++hour) {
                if (newLine[hour] != 0) continue;

                // пытаемся получить свободный час для сотрудника
                newLine[hour] = removeCell(hour);
                if (newLine[hour] != 0) {
                    ++workHours;
                }
            }
        }

        // рабочий день для сотрудника не может быть пустым,
        // т.к. при входе в функцию во временной таблице всегда есть ячейки.
        assert(workHours > 0);

        outputData.add(newLine);
    }
```

Результирующая таблица выглядит таким образом:

Таблица 7 — График работы персонала, полученный при помощи упрощенного алгоритма

	Понедельник												
Время	9.00-10.00	10.00-11.00	11.00-12.00	12.00-13.00	13.00-14.00	14.00-15.00	15.00-16.00	16.00-17.00	17.00-18.00	18.00-19.00	19.00-20.00	20.00-21.00	21.00-22.00
Номер сотрудника													
1	***1***	***1***	***1***	***3***	***3***		3	***3***	***3***				
2					***3***	3		***3***	3	3	***3***	***2***	3
3	***1***	***1***	***3***	***3***		***4***	4	4	***4***				
4					4	***2***	6	***4***	4	***3***	***3***	***3***	
5						2	***2***	***2***	2	***4***	3	***3***	5
6	**6**	3	3	4	2	2				***4***			
7		***3***	***4***							4	***4***		
8			***4***								***4***	3	
9		***4***									***4***		
10												***4***	
11		4											
12												***4***	
13												***4***	

Время работы алгоритма — доли секунды. Также очевидны и минусы: алгоритм является жёстким и не может оптимизировать график небольшим уходом от ограничений, как это делают генетические алгоритмы. В результате получаем график на б**о**льшее число сотрудников,

половина из которых работает идеально, а вторая — не отрабатывает и пол-ставки.

Сравнительный анализ алгоритмов

Сравнение двух алгоритмов представим в виде таблицы 8, где параметрами сравнения будут служить те факторы, которые играют наиболее важную роль в представлении оптимального графика.

Таблица 8 — Сравнение двух алгоритмов

Вид алгоритма	**Генетический алгоритм**	**Упрощенный алгоритм**
Параметры сравнения		
Выходное количество сотрудников	10	14
Время реализации	1 час	1 секунда
Максимальное кол-во перерывов между работами одного сотрудника	3	1
Визуальное представление (макс — 5 баллов)	3	4
Удобство использования/редактирования (макс — 5 баллов)	3	4

Таким образом, исходя из сравнительной таблицы, можно сделать вывод, что лучше взять упрощенный алгоритм, добавить еще несколько условий, исходя из предпочтений руководства и непосредственно работников, чтобы получился оптимальный график работы выхода сотрудников на работу. К примеру, заложить в начале, какой график каждый сотрудник планирует, когда у кого выходной, какое количество

часов планирует отработать, какой график работы предпочтительнее и так далее.

Литература

1. http://math.nsc.ru/AP/benchmarks/UFLP/uflp_ga.html — Генетические алгоритмы.

2. Р. М. Ларин, А.В. Плясунов, А.В. Пяткин Методы оптимизации. Примеры и задачи. учебное пособие Новосибирск: Новосибирский государственный университет, 2003. 120 с.

3. А.Ю. Чирков, В.Н. Шевченко О приближении оптимального решения целочисленной задачи о ранце оптимальными решениями целочисленной задачи о ранце с ограничением на мощность. Статья *Дискретн. анализ и исслед. опер., сер. 2, 2006, том 13, номер 2, страницы 56–73*(Mida6)