Санкт-Петербургский национальный исследовательский университет
информационных технологий механики и оптики
Факультет информационных технологий и программирования
кафедра «Компьютерные технологии»

Отчет по лабораторной работе «Построение управляющих автоматов с помощью генетических алгоритмов»

Вариант № 6

Выполнил студент
факультета ИТиП
группы 3539
Карпов Дмитрий Павлович

Санкт-Петербург
2011

Содержание

Введение

В лабораторной работе требуется установить влияние изменения числа потомков в популяции (λ) на эффективность работы эволюционной стратегии построения автомата, решающей задачу «О роботе, обходящем препятствия». Для решения этой задачи будет проводиться ряд экспериментов при различных λ и постоянных значениях прочих параметров эволюционной стратегии. Для этого была написана программа на языке C++, реализующая данную стратегию. В результате ее работы будет получен файл с некоторыми данными, по которым будет построен график. На основе графиков будут проанализированы общие зависимости эффективности от параметра λ.

Постановка задачи

В данной лабораторной работе требуется экспериментально исследовать влияние значения λ на эффективность работы эволюционной стратегии при построении автомата Мура из восьми состояний, решающего задачу «О роботе, обходящем препятствия». При этом сравнение эффективностей будет проведено при нескольких различных размерах популяции (μ) и при двух различных типах стратегии: «+»-стратегии и «,»-стратегии [1]. В «,»-стратегии срок жизни особи ограничивается одним поколением, то есть отбор следующего поколения производится только среди потомков. В «+»-стратегии же особь может выживать до тех пор, пока она будет достаточно приспособленной, то есть отбор осуществляется среди объединенной популяции предков и потомков (всего из $\mu+\lambda$ особей). Для краткого обозначения характеристик эволюционной стратегии далее будут использоваться следующую запись: (μ,λ) – для «,»-стратегий и $(\mu+\lambda)$ – для «+»-стратегий. В данной работе будут рассмотрены случаи $(1,\lambda),(5,\lambda),(1+\lambda),(5+\lambda)$.

В решаемой задаче робот двигается по заранее определенному полю, которое является квадратом 32×32 (рис. 1). На игровом поле содержатся клетки трех типов: выход из лабиринта, препятствие и пустая клетка. Цель робота – найти выход из лабиринта не более чем за 200 ходов. При этом он может выполнять за ход одно из четырех действий: повернуться налево, повернуться направо, идти прямо или ничего не делать. Если робот пытается пойти прямо и перед ним находится при этом клетка-препятствие, то выполняется операция «ничего не делать». Очередное действие робот может выбирать в зависимости от типа клетки перед ним. В наших экспериментах робот будет перемещаться по полю, изображенному на рисунке 1. При этом стартовая позиция робота – верхняя левая свободная клетка. Выход из лабиринта на рисунке отмечен зеленым домиком.

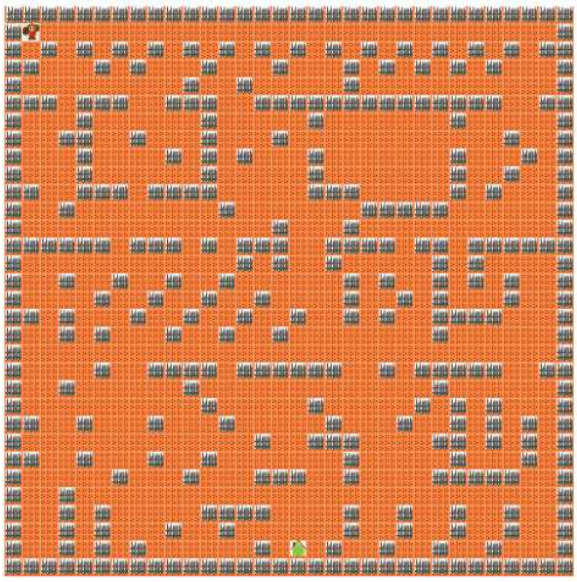

Рис.1 – Поле, по которому осуществляется движение робота.

Проведем эксперименты при постоянном размере поколения (μ) по схемам $(1,\lambda),(1+\lambda),(5,\lambda),(5+\lambda)$. Сравнение эффективности будет осуществляться за счет моделирования работы алгоритма. Будет проводиться несколько экспериментов при одинаковых значениях параметров и рассматриваться средний их результат, чтобы уменьшить влияние случайных операций (например, мутации и скрещивания) на результат эксперимента. В результате будет проведено сравнение эффективности по двум параметрам: значения функции приспособленности в конкретном поколении (максимальное, минимальное и среднее по всем экспериментам) и процент экспериментов, в которых был построен автомат, решающий поставленную задачу. На основании результатов экспериментов требуется исследовать зависимость эффективности эволюционной стратегии от параметра λ.

Описание алгоритма

Для решения задачи на языке C++ была написана программа, реализующая эволюционную стратегию, в том числе ее основные операции: мутацию, скрещивание, отбор следующего поколения, – а так же введена функция приспособленности.

1. Представление автомата

Автомат Мура [2, с. 17 – 21] представлен полной таблицей переходов и таблицей действий.

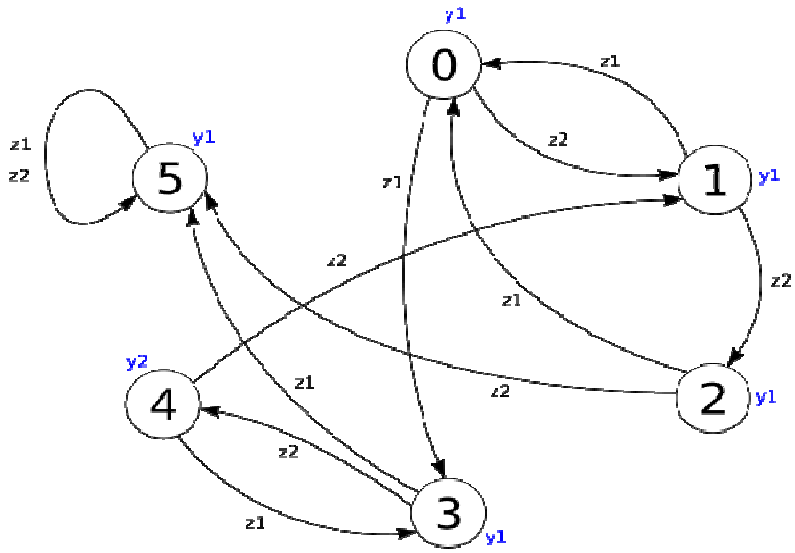

Рис.2 – Пример автомата Мура из шести состояний

В таблице переходов на пересечении номера текущего состояния и входного символа указан номер состояния, в который осуществляется переход. В таблице действий будет храниться операция, которую нужно выполнять при переходе в данное состояние: ничего не делать, повернуть налево, повернуть направо, идти вперед. Для этого был реализован класс *Automaton*, в котором двумерный массив *table* соответствует таблице переходов автомата, массив *act* – таблице действий. Также для каждого автомата будет храниться значение его функции приспособленности (*fit*), которое вычисляется в одном из конструкторов при помощи метода *calcFitness()*. Код класса *Automaton* приведен в приложении 1.

2. Операция мутации

Операция мутации реализуется следующим образом. Каждый элемент таблицы переходов будет изменяться с вероятностью p на случайное состояние автомата. Аналогично, каждая операция из таблицы действий с вероятностью p будет изменяться на случайное действие. Эксперименты будут проводиться при $p = 0.1$. Код метода *mutate* приведен в приложении 2.

3. Операция скрещивания

В данной реализации с вероятностью 1 - *pcross* операция скрещивания не производится, а метод *crossover* возвращает предка с большей функцией приспособленности. С вероятностью *pcross* осуществляется непосредственно операция скрещивания. В данной реализации используется *pcross* = 0.6. При скрещивании из двух родительских особей, потомок получается по следующей схеме. Потомок получает вершину автомата со всеми переходами из нее одного из родителей, причем вероятность унаследовать вершину у родителя с большей функцией приспособленности равна 0.6. Аналогично, потомок наследует элементы таблицы действий у предка с большей функцией приспособленности с вероятностью 0.6, а у другого предка – с вероятностью 0.4. Код метода *crossover* приведен в приложении 3.

4. Функция приспособленности

Согласно условию задачи робот может сделать не более 200 ходов в поле размером 32×32. Пусть тогда $\{x_n, y_n\}_{n=0}^{200}$ – последовательность точек поля, посещенных роботом, действующим под управлением автомата A. Тогда функция приспособленности будет иметь следующий вид: $\varphi(A) = 65 - \min\limits_{0\le i\le 200}(|x_i - x'| + |y_i - y'| + \frac{i}{200})$, где (x', y') – координаты выхода из лабиринта. Значение функции можно вычислить для каждого автомата моделированием поведения робота согласно построенному автомату. Код метода *calcFitness* приведен в приложении 4.

5. Построение следующего поколения

При генерации следующего поколения сначала производятся операции скрещивания, затем особи, полученные в результате скрещивания, мутируют. Среди получившейся популяции (в случае «+»-стратегии к ней добавятся родительские особи) производится отбор особей, которые составят следующее поколение. Следующее поколение составляется из μ экземпляров с лучшими функциями приспособленности. Выбор очередной пары особей для скрещивания производится случайно. Этот процесс реализуется методом *nextGeneration* класса *Evolution*, который возвращает максимальное значение функции приспособленности автоматов полученной популяции. Код метода *nextGeneration()* приведен в приложении 5.

Результаты эксперимента

Далее отдельно будут рассмотрены результаты экспериментов для каждого из четырех случаев, указанных в постановке задачи, и проанализировано поведение функции приспособленности в зависимости от λ.

Ниже приведены результаты первого типа экспериментов. В данных запусках программы будет рассмотрена зависимость максимального, минимального и среднего значения функции приспособленности по 20 запускам для каждого $1 \le \lambda \le 100$. При этом в каждом запуске осуществляется генерация 2000 поколений автоматов для «,»-стратегий и 500 поколений автоматов для «+»-стратегий. Рассмотрим графики зависимости среднего значения фитнес-функции от номера поколения.

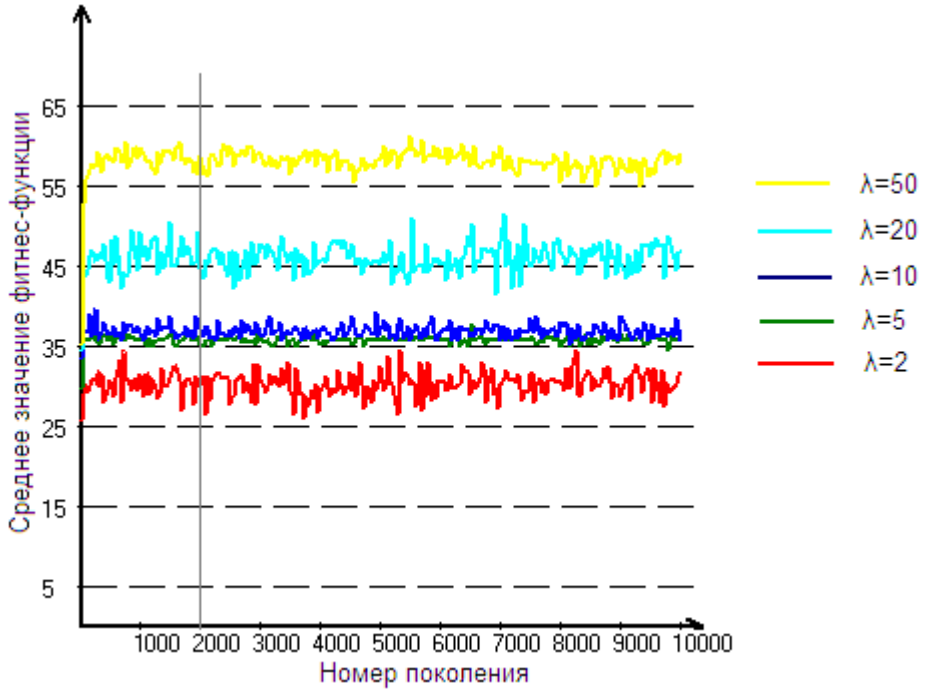

Рис.3 – Зависимость средних значений функции приспособленности от номера поколения при стратегии $(1, \lambda)$

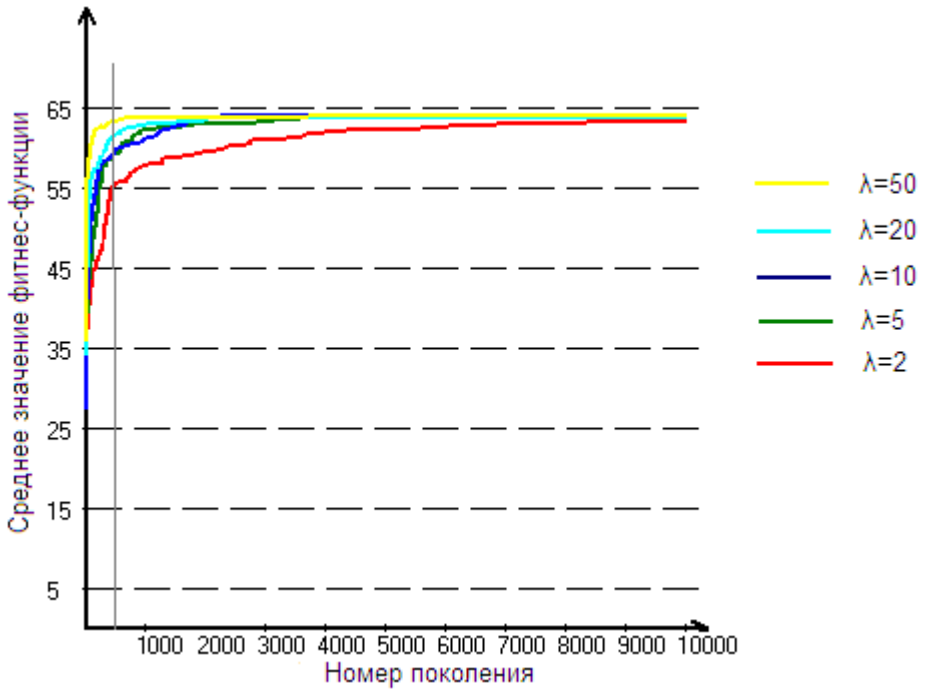

Рис.4 – Зависимость средних значений функции приспособленности от номера поколения при стратегии $(1+\lambda)$

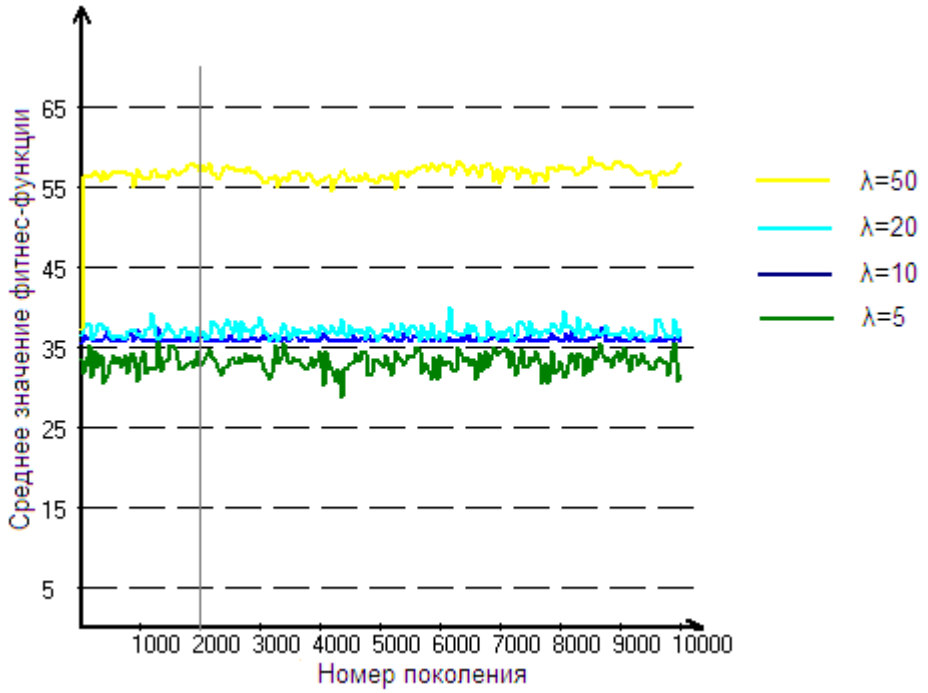

Рис.5 – Зависимость средних значений функции приспособленности от номера поколения при стратегии $(5,\lambda)$

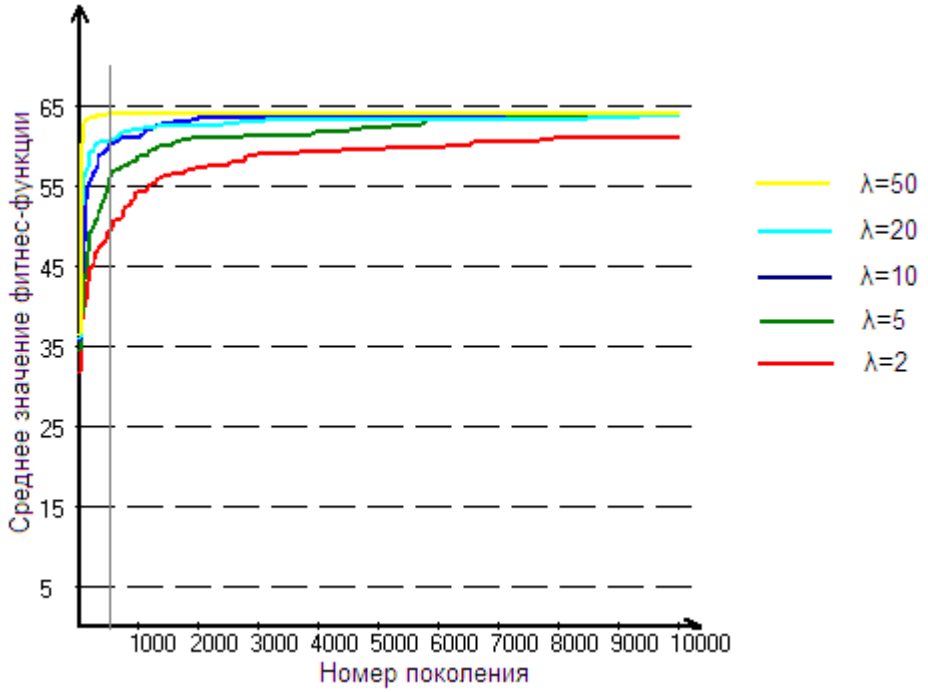

Рис.6 – Зависимость средних значений функции приспособленности от номера поколения при стратегии $(5+\lambda)$

При «,»-стратегиях (рис. 2, 4) с ростом номера поколения не происходит существенного изменения функции приспособленности, поэтому использование 2000 поколений допустимо для сравнения эффективности при разных λ. При таком количестве поколений средние значения фитнес-функций достаточно различаются. При «+»-стратегиях (рис. 3, 5) использование 2000 поколений недопустимо, поскольку при данной стратегии среднее значение функции приспособленности слишком быстро возрастает и различия между эффективностями алгоритмов уменьшаются. Из графиков видно, что подходящим решением является генерация в «+»-стратегия 500 поколений автоматов. Таким образом, сравнение эффективности алгоритмов можно проводить, генерируя 2000 поколений автоматов для «,»-стратегий и 500 поколений автоматов для «+»-стратегий при каждом запуске алгоритма. Также для уменьшения времени вычисления при «,»-стратегиях будем рассматривать только нечетные λ.

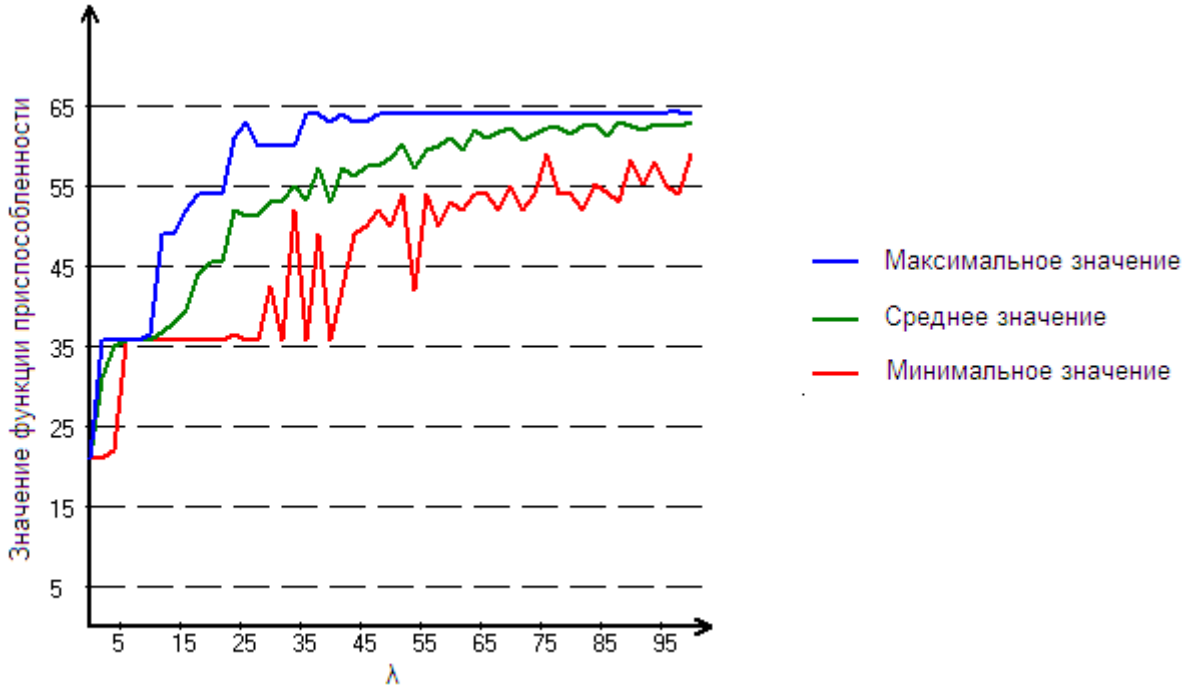

Рис.7 – Значение функции приспособленности в зависимости от λ при стратегии $(1,\lambda)$

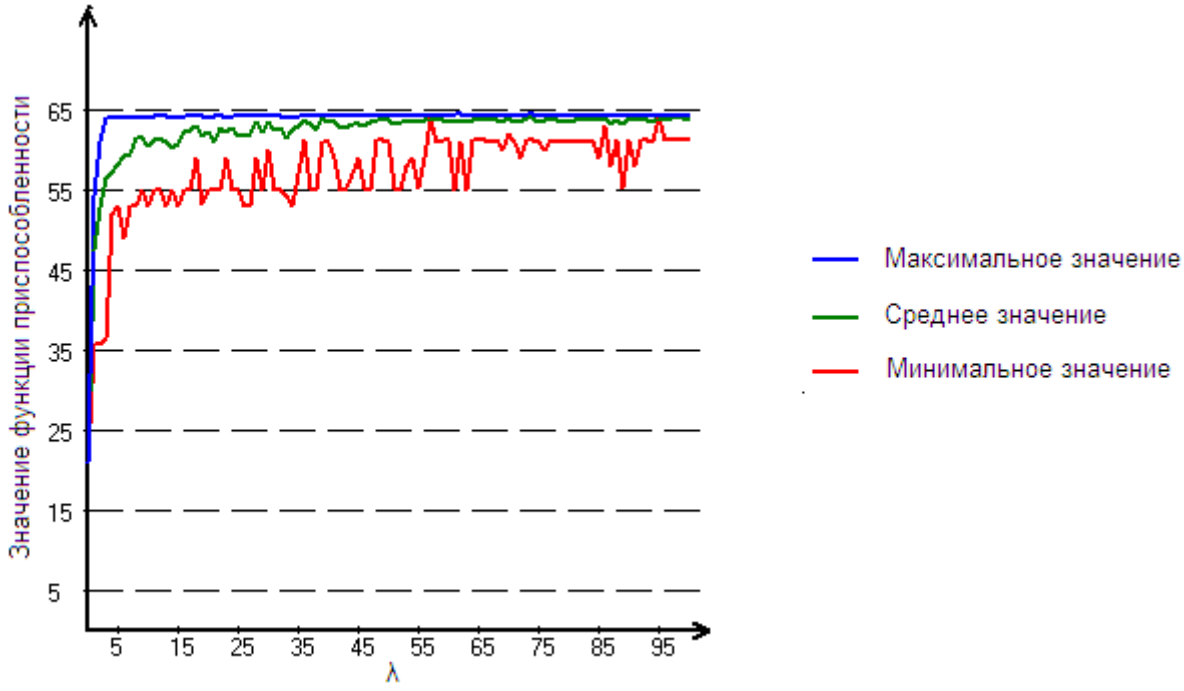

Рис.8 – Значение функции приспособленности в зависимости от λ при стратегии $(1+\lambda)$

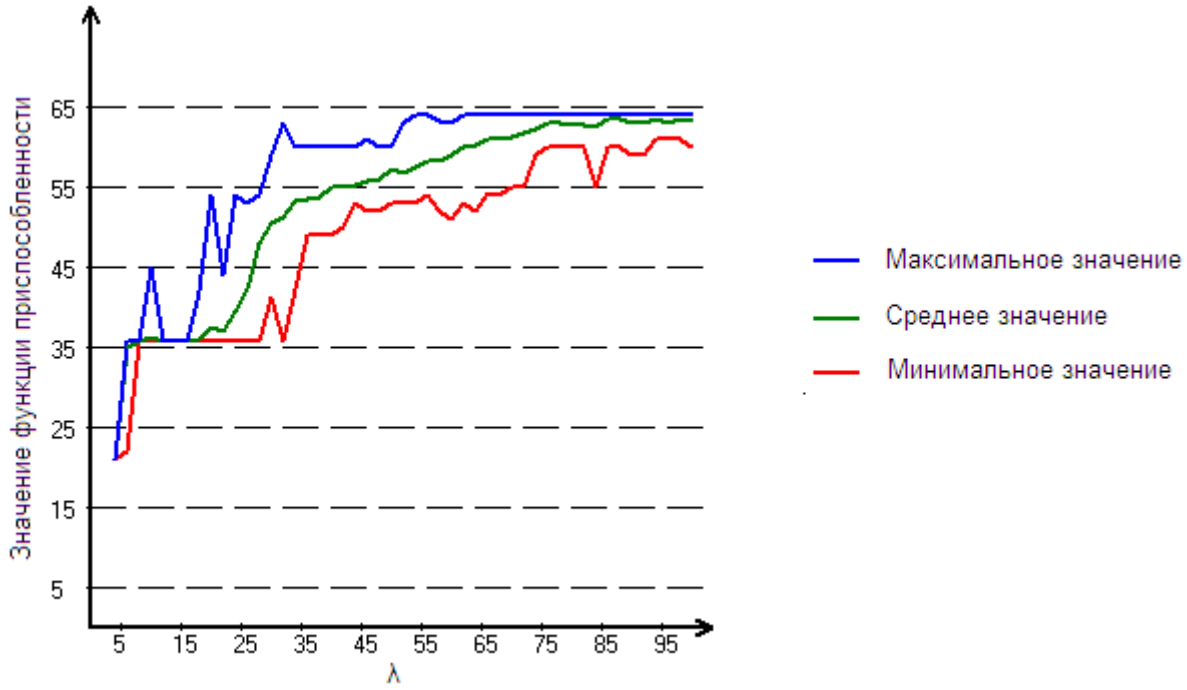

Рис.9 – Значение функции приспособленности в зависимости от λ при стратегии $(5,\lambda)$

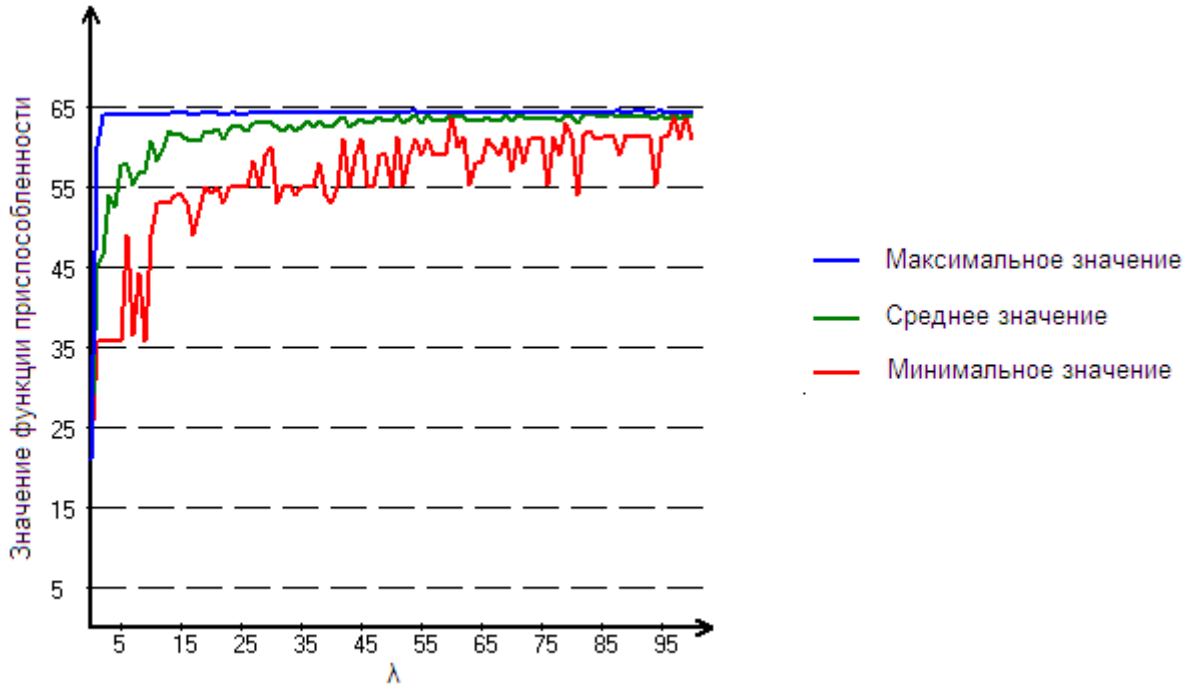

Рис.10 – Значение функции приспособленности в зависимости от λ при стратегии $(5+\lambda)$

Из данных графиков видно, что при росте λ можно выделить два процесса: рост среднего значения фитнес-функции и уменьшение ее дисперсии. Стоит отметить, что при стратегии $(1,\lambda)$ (рис. 6), даже при достаточно большом числе потомков, функция приспособленности может принимать малые значения, что, возможно, при одиночном запуске не даст хорошего результата. Также в «+»-стратегиях (рис. 7, 9) при больших

λ среднее значение фитнес-функции асимптотически приближается к максимальному значению. Естественно предположить, что этот процесс будет продолжаться с увеличением λ. В «,»-стратегиях (рис. 6, 8) также наблюдается рост значений фитнес-функции, но со значительно меньшей скоростью. В «+»-стратегиях при $\lambda > 25$ рост становится практически несущественным. При таких λ алгоритм может работать более эффективно при меньшем числе потомков, так как в таком случае для его реализации требуется меньше ресурсов (прежде всего позволит сократить затраты времени на обработку каждого поколения).

В следующем типе экспериментов рассмотрим зависимость количества успешных автоматов от λ $(1 \le \lambda \le 50)$ в 100 генерациях автоматов. В каждом опыте будет генерироваться не более 500 поколений автоматов. При этом процесс генерации может быть остановлен раньше: при построении успешного автомата. Назовем автомат «успешным», если он решает поставленную задачу, то есть робот под управлением из автомата находит выход из лабиринта. Для этого значение фитнес-функции особи должно быть не меньше 64.

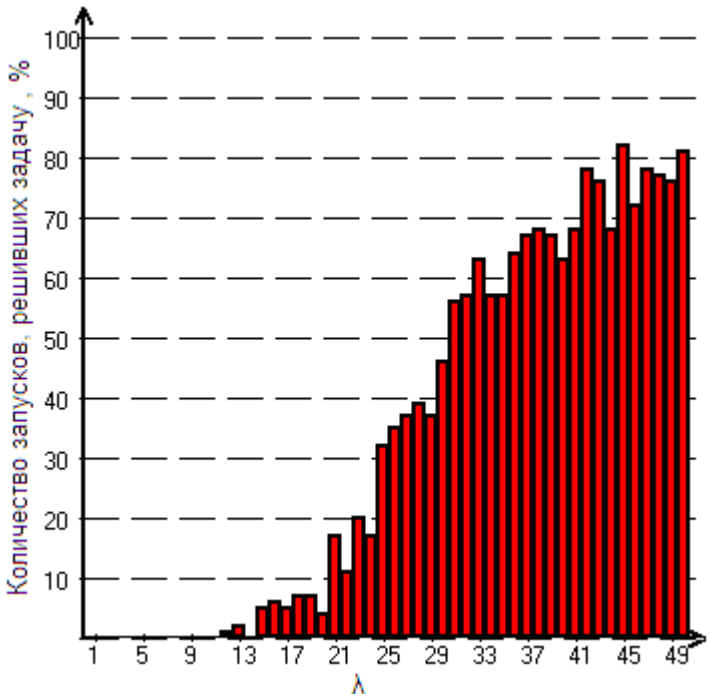

Рис.11 – Зависимости среднего времени генерации успешных автоматов и их количества при стратегии $(1, \lambda)$

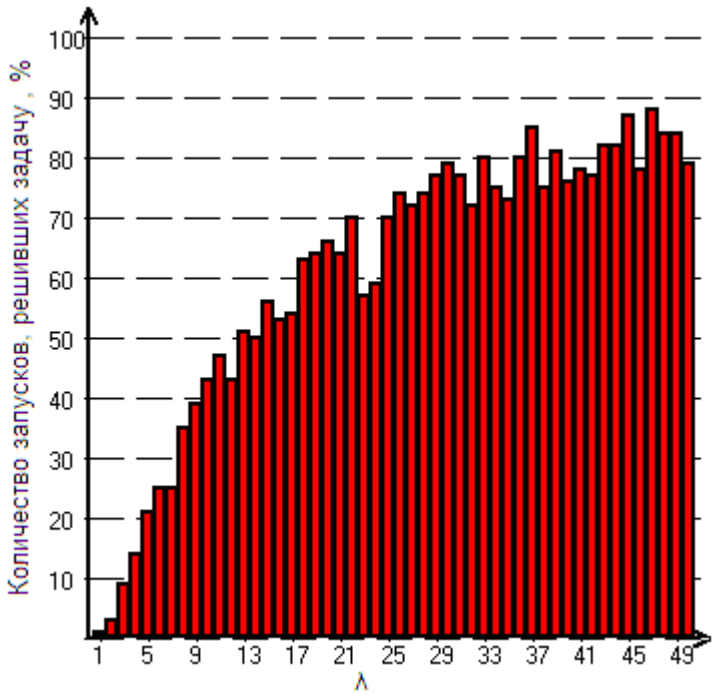

Рис.12 – Зависимости среднего времени генерации успешных автоматов и их количества при стратегии $(1+\lambda)$

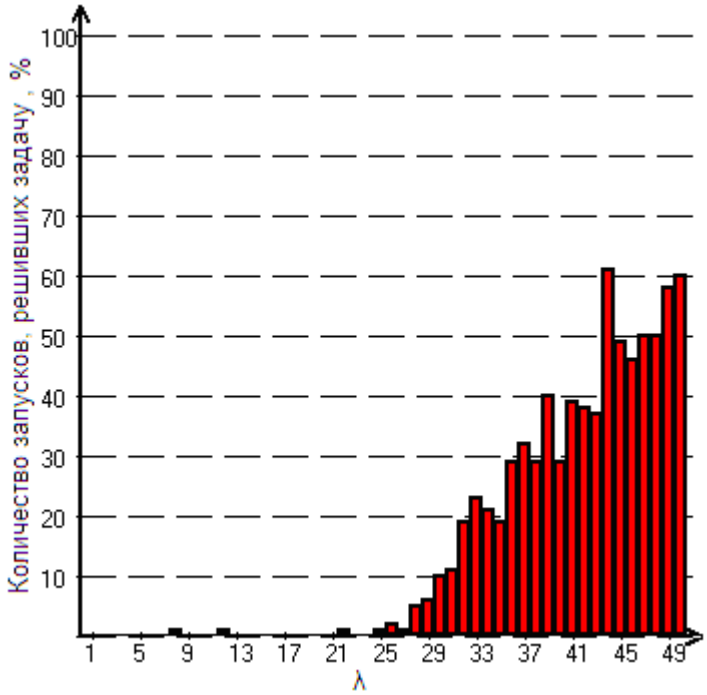

Рис.13 – Зависимости среднего времени генерации успешных автоматов и их количества при стратегии $(5,\lambda)$

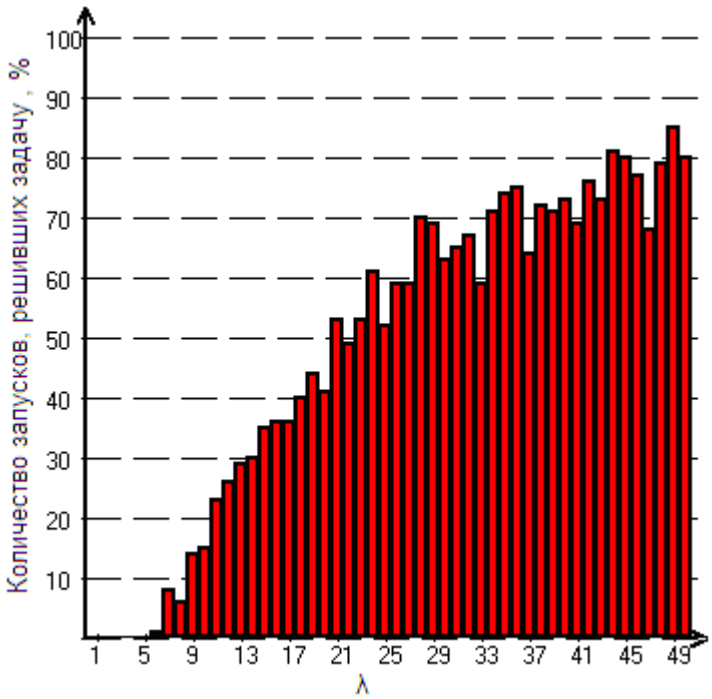

Рис.14 – Зависимости среднего времени генерации успешных автоматов и их количества при стратегии $(5+\lambda)$

Из данных графиков видно, что $\mu = 5$ (рис. 12, 13) рост числа успехов происходит медленнее, чем при $\mu = 1$ (рис. 10, 11) для $1 \le \lambda \le 50$. Это обусловлено тем, что при достаточно малых μ операция скрещивания, которая отсутствует при $\mu = 1$ и появляется при $\mu = 5$, уменьшает эффективность. Это связано с тем, что из каждого поколения отбирается слишком мало особей и, как следствие, разнообразие особей, получаемых при скрещивании, не велико. Как итог, вероятность увеличения фитнес-функции в результате скрещивания также достаточно мала. Улучшить эти показатели можно либо существенным увеличением λ, либо (возможно меньшим) увеличением μ. При этом, при «+»-стратегиях наблюдается большая статистическая стабильность (меньшая дисперсия). При $\lambda > 40$ не меньше половины автоматов генерируются успешными при всех стратегиях.

Примеры построенных автоматов, решающих задачу

Приведем примеры автоматов, решающих задачу «О роботе, обходящем препятствия», построенные при помощи описанного выше алгоритма. Ниже приведены таблицы, представляющие автоматы, (табл. 1, 2) и изображения путей, по которым робот движется к выходу под управлением этих автоматов (рис. 14, 15). Задача решалась при помощи стратегий $(1,20)$ и $(5+5)$. В описанных ниже автоматах состояние с номером ноль считаем стартовым. Так же считаем, что действие выполняется при переходе в новое состояние, после чего робот просматривает клетку перед ним. При первом переходе в начальное состояние будем считать, что никакое действие не выполняется. Стоит отметить, что построенный стратегией $(1,20)$ автомат, по сути, содержит семь состояний, так как состояние номер пять является недостижимым из стартового. Аналогично, автомат, построенный $(5+5)$ стратегией, содержит по сути семь состояний.

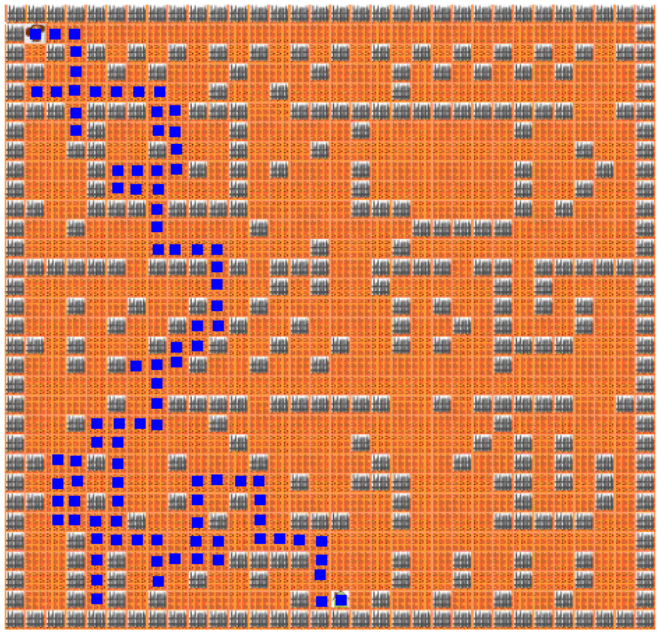

Рис. 14 – Путь, найденный автоматом при стратегии $(1,20)$

Таблица 1 – Автомат, построенный при стратегии $(1,20)$

Номер состояний	Пустая клетка	Препятствие	Выход	Действие
0	3	2	1	Повернуть налево
1	7	6	4	Идти прямо
2	4	4	1	Повернуть налево
3	6	0	2	Идти прямо
4	3	0	0	Повернуть налево
5	7	4	2	Повернуть направо
6	1	0	1	Повернуть направо
7	3	2	2	Идти прямо

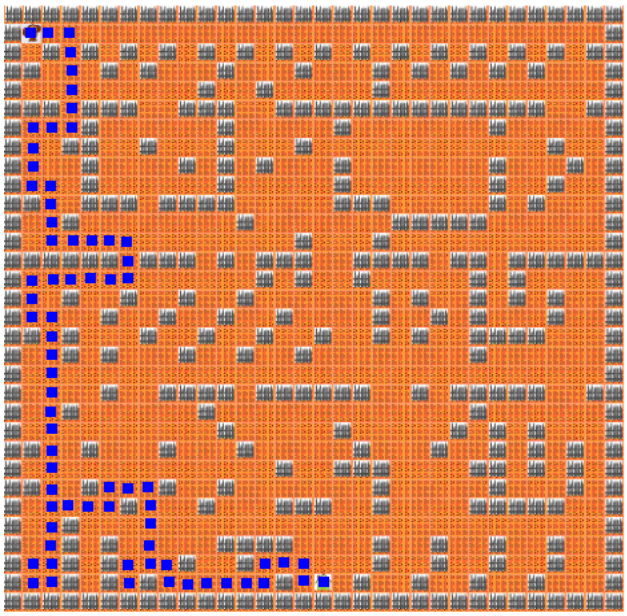

Рис. 15 – Путь, найденный автоматом при стратегии (5 + 5)

Таблица 2 – Автомат, построенный при стратегии (5 + 5)

Номер состояний	Пустая клетка	Препятствие	Выход	Действие
0	7	2	7	Повернуть налево
1	4	5	6	Идти прямо
2	7	2	6	Повернуть направо
3	1	3	5	Идти прямо
4	6	1	7	Идти прямо
5	6	0	4	Повернуть направо
6	4	0	5	Идти прямо
7	5	6	2	Идти прямо

Заключение

Наибольшая скорость роста функции приспособленности в «+»-стратегиях наблюдается при $5 \le \lambda \le 15$. В «,»-стратегиях увеличение среднего значения функции приспособленности происходит с меньшей скоростью на протяжении всего интервала $1 \le \lambda \le 100$. В «+»-стратегиях при недостаточной производительности системы не имеет смысла увеличивать λ, поскольку это не позволит добиться большого увеличения функции. Также в «+»-стратегиях увеличение λ позволяет добиться стабильного роста среднего (по запускам) значения функции приспособленности. Это заметно проявляется в первой тысяче поколений. Также было установлено, что стратегии с $\mu = 5$ менее эффективны, чем при $\mu = 1$. Это связано с тем, что при $\mu = 5$ в генерации новых поколений существенную роль играет операция скрещивания. Но так как число потомков достаточно мало, то вероятность скрещивания, повлекшего увеличение фитнес-функции, незначительна.

Список литературы

[1] Back T., Hoffmeister F., Schwefel H. P. A Survey of Evolutionary Strategies. Proceeding of the Fourth International Conference on Genetic Algorithms, San Mateo, 1992.

[2] Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – Санкт-Петербург, 2008.

Приложение

Приложение 1. Класс, представляющий автомат

```
class Automaton{
public:
	Automaton(int,int);
	Automaton(const vector<vector<int> > &, const vector<int> &);
	Automaton(const Automaton&);
	bool operator=(const Automaton&);
	double fitness() const;
	void calcFitness();
	static void init();
	Automaton mutate(double);
	Automaton crossover(Automaton &x, double pcross = 0.6);
	bool operator<(const Automaton&);
private:
	vector<vector<int> > table;
	vector<int> act;
	double fit;
};
```

Приложение 2. Реализация операции мутации

```
Automaton Automaton::mutate(double p)
{
	double pr = p;
	vector<vector<int> > tmp;
	vector<int> a;
	int tot = 1;
	while (pr < 1)
	{
		pr *= 10;
		tot *= 10;
	}
	int k;
	for (int i=0;i<table.size();i++)
	{
		tmp.push_back(vector<int>());
		for (int j=0;j<table[i].size();j++)
		{
			k=abs(rand()%tot);
			if (k<=pr)
				tmp[i].push_back(abs((int)(rand()%table.size())));
			else
				tmp[i].push_back(table[i][j]);
		}
	}
	for (int i=0;i<tmp.size();i++)
	{
		k=rand()%tot;
		if (k<=pr)
			a.push_back(abs(rand()%4));
		else
			a.push_back(act[i]);
	}
	return Automaton(tmp,a);
}
```

Приложение 3. Реализация операции скрещивания

```
Automaton Automaton::crossover(Automaton &x, double pcross)
{
```

```
	vector<vector<int> > v;
	vector<int> a;
	int tmp=1.;
	while (pcross<1)
	{
		pcross*=10;
		tmp*=10;
	}
	if (rand()%tmp<=pcross)
	{
		int key;
		if (this->fitness()>x.fitness())
			key=1;
		else
			key=3;
		for (int i=0;i<table.size();i++)
		{
			if (rand()%5>=key)
			   v.push_back(table[i]);
			else
			   v.push_back(x.table[i]);
		}
		for (int i=0;i<act.size();i++)
			if (rand()%5>=key)
				a.push_back(act[i]);
			else
				a.push_back(x.act[i]);
		return Automaton(v,a);
	}
	else
	{
		if (this->fitness()>x.fitness())
			return (*this);
		else
		    return x;
	}
}
```

Приложение 4. Реализация вычисления функции приспособленности

```
void Automaton::calcFitness()
{
	int cur=0;
	int t=0;
	int sz=act.size();
	int x=1,y=1;
	double res=0,tmp;
	for (int q=0;q<STEP;q++)
	{
		switch (myMap[x+sh[t][0]][y+sh[t][1]])
		{
		case 0:
			cur=table[cur][0];
			break;
		case 1:
			cur=table[cur][1];
			break;
		default:
			cur=table[cur][2];
		}
		switch (act[cur]){
			case 0:
				;
			break;
```

```
                case 1:
                      t=(t+1)%4;
                break;
                case 2:
                      t=t-1;
                      if (t<0)
                            t=3;
                break;
                case 3:
                      if (myMap[x+sh[t][0]][y+sh[t][1]]!=1)
                      {
                            x=x+sh[t][0];
                            y=y+sh[t][1];
                      }
                break;
          }
          tmp=2*MAX_SZ+1-abs(x-ex.first)-abs(y-ex.second)-q/200.;
          if (res<tmp)
             res=tmp;
          if (x==ex.first && y==ex.second)
                fit = res;
    }
    fit = res;
}
```

Константа *STEP* – максимальное число действий, совершаемое роботом на поле, *MAX _ SZ* – размер поля. В массиве *sh* описан закона перемещения робота в зависимости от направления, в которое он повернут.

Приложение 5. Реализация построения следующего поколения

```
double Evolution::nextGeneration()
{
    if (mu==1)
    {
          vector<Automaton> v,res;
          for (int i=0;i<lambda;i++)
                v.push_back(obj.get(0).mutate(mutProb));
          if (! fullChange)
                v.push_back(obj.get(0));
          int pos=0;
          double max=0.0;
          double tmp;
          for (int i=0;i<v.size();i++)
          {
                tmp=v[i].fitness();
                if (tmp>max)
                {
                   max=tmp;
                   pos=i;
                }
          }
          res.push_back(v[pos]);
          obj=Population(res);
          return max;
    }
    else if (mu>1)
    {
          if (lambda<mu && fullChange)
          {
                cout<<"Так нельзя!!!";
                exit(0);
```

```
            }
            vector<Automaton> children,res;
          vector<pair<int,int> > pii;
            for (int i=0;i<obj.size();i++)
                  for (int j=i;j<obj.size();j++)
                        pii.push_back(make_pair(i,j));
            int r;
            int sz=pii.size();
            for (int i=0;i<lambda;i++)
            {
                  r=rand()%sz;

      children.push_back(obj.get(pii[r].first).crossover(obj.get(pii[r].second
)));
            }
            for (int i=pii.size();i<lambda;i++)
              children.push_back(obj.get(rand()%mu).mutate(2*mutProb));
            for (int i=0;i<lambda;i++)
                  children[i]=children[i].mutate(mutProb);
            if (! fullChange)
                  for (int i=0;i<mu;i++)
                        children.push_back(obj.get(i));
            sort(children.begin(),children.end());
            sz=children.size();
            for (int i=sz-1;i>=sz-mu;i--)
                  res.push_back(children[i]);
            obj=Population(res);
            return children[sz-1].fitness();
      }
}
```