ФИЛИАЛ КАЗАНСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

В Г. НАБЕРЕЖНЫЕ ЧЕЛНЫ

ФАКУЛЬТЕТ ПМ и ИТ

Курсовая работа

за VI семестр

Моделирование поведения средствами генно-нечётких систем

Выполнил

Сафронов Марк Андревич

студент группы 4606 (III-курс)

Научный руководитель

Лернер Эдуард Юльевич

кандидат физ-мат. наук, доцент

Члены комиссии по защите курсовой работы

_____
Ф.И.О подпись

_____
Ф.И.О подпись

_____
Ф.И.О подпись

Оценка_____

Дата_____

2009

Оглавление

1 Введение

1.1 Цели

Целью работы является построение и реализация математической модели, имитирующей поведение некоторых, во всяком случае, рационально мыслящих, существ (например, человека), оценка работоспособности предположений, при которых она может быть реализована и методов, выбранных для реализации.

1.2 Постановка задачи

Модель должна быть способной *из всех возможных субъектов поведения отобрать такой, чьё поведение приведёт его к заданному желаемому состоянию* [1]. При этом предполагается, что поведение, вообще говоря, осознанно *не направлено* на достижение заданного конечного состояния.

Функционирование модели должно обеспечиваться компьютерной программой в виде Windows-приложения, реализующей модель.

1.3 Актуальность работы

Изучение поведения человека, формализация его в виде какой-либо математической модели и совершение предсказаний на её основе являются фундаментальными вопросами из нескольких смежных областей знаний, таких как психология, социология, биохимия, генетика и наука об искусственном интеллекте. Актуальность этих вопросов не исчезнет до тех пор, пока не будет найдено их гарантированное точное решение.

[1] Определение субъекта поведения и терминов «поведение», «состояние» приведено в разделе 2.1 «Необходимая терминология».

2 Теоретическая часть

2.1 Необходимая терминология

Субъект поведения будем называть «**особью**». Особь представлена в модели в виде *набора численных именованных параметров*.

Само «**поведение»** будем рассматривать как циклический процесс, заключающийся в изменении параметров особи. То, каким образом на каждом шаге итерации поведения происходит изменение параметров, предполагается зависящим от текущего значения параметров.

«**Состоянием**» особи будем называть набор текущих значений параметров особи на момент наблюдения. Таким образом, «**конечное состояние**» особи – значения параметров особи на момент окончания моделирования её поведения.

2.2 Формулировка математической модели

Поставленная задача в вышеуказанных терминах сводится к следующей задаче оптимизации:

$$\iota^* = \arg\max_{\iota} f_{\tilde{\rho}}(\rho(\iota)) \qquad (2.1)$$

Здесь $\rho(\iota)$ – это n-мерный вектор, обозначающий собой значения параметров особи на момент начала развития динамики её поведения.

$$\rho(\iota) \in \mathrm{P} = \left\{ (p_1, p_2, \ldots, p_n) \middle| p_{k_1} \in [\underline{p_k}, \overline{p_k}] \subset R, k = \overline{1,n} \right\} \qquad (2.2)$$

Каждый параметр предполагается принадлежащим отрезку действительной оси $[\underline{p_k}, \overline{p_k}]$, но это не обязательное условие[2]. Множество P называется пространством значений параметров особи. Значения вектора ρ зависят от значений вектора ι, которые, собственно, и являются аргументами функции $f_{\tilde{\rho}}(\rho(\iota))$.

[2]Причины данного ограничения указаны в разделах 2.4.2 «Применение нечёткой логики для формализации поведения» и 2.5 «Оценка приспособленности».

Введение дополнительного вектора $\imath$, который преобразуется в вектор параметров особи для дальнейших вычислений, было обусловлено в первую очередь возможным количеством параметров особи, в особенности, если особь представляет собой оцифровку особенностей *человека*.

Излишне большая размерность вектора значений параметров особи приводит к тому, что оптимизация целевой функции $f_{\tilde{\rho}}(\rho)$ будет производиться на крайне большом пространстве значений аргументов. Поэтому автор счёл уместным ввести в модель понятие «**склонностей**», основывающееся на предположении о том, что некоторая (меньшая) часть параметров особи влияет на поведение принципиально более значимо, нежели все остальные параметры. На примере поведения человека «склонности» идентичны генетическим предрасположенностям человека, заданным ему при рождении, и не меняющимся в процессе жизнедеятельности. Влияние генетических предрасположенностей на поведение человека рассматривается в работах этологов, например, [1] или [2].

Значения склонностей перед началом моделирования поведения предполагается преобразовывать в начальное состояние особи, поэтому единственное влияние, которое концепция «склонностей» оказывает на модель – уменьшение пространства поиска.

Таким образом, значения вектора склонностей определёны следующим образом:

$$\imath \in \mathrm{I} = \left\{ (i_1, i_2, ..., i_m) \middle| i_k \in [\underline{i}_k, \overline{i_k}] \subset Z^+, k = \overline{1, m} \right\} \qquad (2.3)$$

Множество I называется пространством значений склонностей особи Пространство значений склонностей особи, как несложно заметить, является ограниченным[3], так же, как и пространство значений параметров особи. Склонности в данном случае приняты положительными целочисленными значениями для ещё большего уменьшения пространства поиска. Следует заметить, что число склонностей m должно быть меньше числа параметров n, иначе введение концепции «склонностей» теряет смысл.

[3] Причина этого указана в разделе 2.4.2.1 «Имитация самооценки».

Функция $f_{\tilde{\rho}}(\rho(\imath))$ называется «функцией приспособленности», единственно по причине того, что метод, выбранный для её оптимизации[4], подразумевает её названной так. Функция приспособленности возвращает численное значение, символизирующее *степень соответствия* её текущих аргументов *ρ(ι) заданному желаемому состоянию* $\tilde{\rho}$. Она является кратким переобозначением функции $\tilde{f} : \mathrm{P} \times \mathrm{P} \to R$, которая производит *оценку отличия* конечного достигнутого особью по окончании поведения состояния от заданного желаемого[5]:

$$f_{\tilde{\rho}}(\rho(\imath)) \equiv \tilde{f}(\varsigma_r(\imath), \tilde{\rho}) \tag{2.4}$$

Функция ς_r является, собственно, «функцией поведения», с количеством шагов итерации r. Она получает значения склонностей особи, генерирует на их основе значения параметров (ещё раз следует заметить, что динамика изменения особи рассматривается только на значениях её параметров, значения склонностей используются только на этапе инициализации) и, после r шагов итерации поведения, возвращает её конечное состояние.

Таким образом, функция итерации поведения определяется рекурсивно, следующим образом:

$$\varsigma_r(\imath) = \begin{cases} u(\imath), & r = \mathrm{O} \\ \varsigma(\varsigma_{r-1}(\imath)), & r > \mathrm{O} \end{cases}, \qquad r \in Z^+ \tag{2.5}$$

$$\varsigma_r : \mathrm{I} \to \mathrm{P}, \qquad u : \mathrm{I} \to \mathrm{P}, \qquad \varsigma : \mathrm{P} \to \mathrm{P} \tag{2.6}$$

Функция u преобразует полученные значения склонностей в начальное состояние особи, от которого будет отталкиваться функция итерации поведения. Можно сказать, что, согласуясь с определением (2.1), значение функции *u(ι)* эквивалентно вектору *ρ(ι)*, от которого зависит значение функции приспособленности:

$$u(\imath) \equiv \rho(\imath) \tag{2.7}$$

Функция ς является элементарным шагом итерации поведения, который создаёт новые значения параметров особи на основе текущих значений.

[4] Рассмотрен далее, в разделе 2.3 «Метод оптимизации»

[5] Метод оценки рассмотрен в разделе 2.5 «Оценка приспособленности»

Определение этой функции основывается на предположениях о поведении, взятых, в случае поведения человека, из психологии, этологии, социологии или других гуманитарных наук. Формализация функции ς для данной реализации модели описана в разделе 2.4 «Формализация поведения».

Число итераций r задаётся вместе с желаемым конечным состоянием особи. Следует отметить, что возможно ввести число итераций в виде ещё одного аргумента целевой функции, и искать, например, последовательность изменений особи, приводящей особь к желаемому состоянию, которая была бы минимальной длины. В данной реализации модели это не используется ради упрощения оптимизации, что приводит всю модель в крайнюю зависимость от r.

2.3 Метод оптимизации

Для вычисления $argmax$ от целевой функции используются эволюционные методы в виде генетического алгоритма. Выбор метода обоснован следующим:

1) генетические алгоритмы *определены* как метод нахождения аргументов функции, максимизирующих её значение ([3, стр. 11]);
2) для данной тематики модели использование генетических алгоритмов, имитирующих эволюционные процессы в среде живых существ, является естественным действием.

2.3.1 Генетические алгоритмы

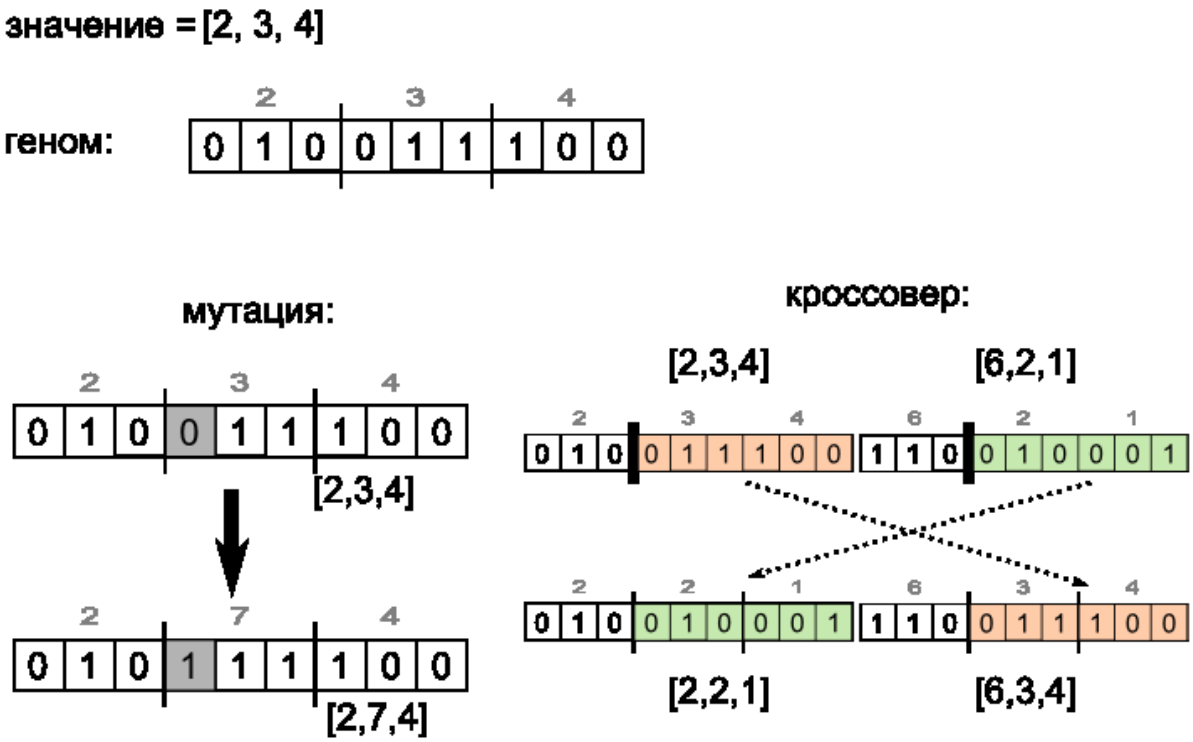
Для оптимизации целевой функции используется так называемый «классический генетический алгоритм», разобранный более подробно в [3]. Следующее краткое описание генетического алгоритма, как он применяется в данной работе, основано на описании из этой книги.

Генетические алгоритмы копируют приёмы, которые природа использует при репродукции следующих поколений живых существ – мутацию и рекомбинацию. Абстрактно, для генетического алгоритма искомые аргументы оптимизируемой функции кодируются в виде объекта (называемого далее «**геномом**[6]»), над

[6] англ. genome

которым определяются два оператора: «**оператор мутации**[7]**»** и «**оператор кроссовера**[8]**»**, и один метод: определения численного значения «**приспособленности**[9]**»** генома условиям оптимизации. Геном сам по себе является множеством более мелких закодированных элементов, называемых далее «**генами**[10]». Каждый ген кодирует один аргумент оптимизируемой функции.

Оператор мутации случайным образом изменяет код генома. В классической схеме мутация изменяет значение одного случайно выбранного гена из генома.



Операторы мутации и оператора кроссовера

Оператор кроссовера по какому-либо принципу из двух исходных геномов получает два производных, называемых далее «дочерними». В классической схеме кроссовер выбирает в обоих геномах одну и ту же позицию между генами,

[7] англ. mutation

[8] или рекомбинации, англ. crossover

[9] англ. fitness

[10] англ. gene

называемую далее «точкой кроссовера[11]», и создаёт два дочерних генома, извлекая для первого дочернего генома цепочку генов *до* точки кроссовера из *первого* исходного генома и цепочку *после* точки кроссовера из *второго* исходного генома. Аналогично для второго дочернего генома извлекается цепочка генов *до* точки кроссовера из *второго* исходного генома и цепочка *после* точки кроссовера из *первого* исходного генома.

Далее (по классической схеме) алгоритм на первом шаге итерации генерирует случайным образом стартовое множество геномов, называемое далее «**популяцией**[12]», для каждого из которых вычисляется значение приспособленности. Затем на основании значений приспособленности каждого генома генерируется следующая популяция особей применением к геномам в текущей популяции операторов кроссовера и мутации (используя заранее установленную вероятность появления мутации). Некоторые геномы, в зависимости от их приспособленности, могут рекомбинироваться несколько раз, имитируя большую репродуктивную способность более приспособленных особей. Таким образом, следующая популяция геномов будет скорее состоять из результатов кроссовера и мутаций наиболее приспособленных геномов из текущей популяции.

Генетический алгоритм повторяет построение следующих поколений популяций геномов до тех пор, пока не будет достигнуто условие останова.

Условием остановки может быть, например:

1) перебор определённого количества поколений особей;
2) достижение популяции, среднее значение приспособленности которой превысит некоторое заданное пороговое (или достигнет максимального возможного для функции определения приспособленности);
3) достижение популяции, содержащей особь с приспособленностью, превышающей пороговую (или равной максимальной возможной для функции определения приспособленности).

[11] англ. crossover point

[12] англ. population

Результатом работы генетического алгоритма является популяция (то есть, набор геномов), удовлетворяющий только что упомянутым условиям, из которой может, например, быть выбран геном с наибольшим значением приспособленности.

2.3.2 Применение метода оптимизации

Целевая функция $f_{\tilde{\rho}}(\rho(\iota))$ (2.1) оптимизируется генетическим алгоритмом, участвуя в нём как, собственно, функция приспособленности. В классической схеме традиционно используется область значений функции приспособленности в виде отрезка действительной оси от 0 до 1; таким образом, целевая функция определяется так:

$$f : \mathrm{I} \to [0,1] \in R \qquad (2.8)$$

После завершения работы генетический алгоритм возвращает геном, декодированное значение которого представляет собой кортеж аргументов целевой функции, максимизирующий её значение.

Большую часть вычислений в целевой функции занимают вычисления функции итерации поведения ς_r. Учитывая подробности, изложенные далее в разделе 2.4 «Формализация поведения», вычисление этой функции может оказаться (и, скорее всего, окажется) трудоёмким. Забегая вперёд, можно сказать, что работа всего генетического алгоритма[13] занимает *исчезающе малое* время по сравнению со временем, требуемым для вычисления целевой функции на лишь *одном* заданном наборе аргументов, в случае оцифровки особи параметрами количества порядка двух десятков и количестве итераций порядка сотен.

[13] Имеются в виду выполнение операторов мутации и кроссовера, генерация исходной и последующих популяций, а также проверка условий остановки – все действия, не связанные с непосредственно вычислением функции приспособленности.

2.4 Формализация поведения

Формализация поведения[14] каким-либо структурированным и предсказуемым образом является значительной задачей самой по себе. Поэтому автор не стремился к строгой научной обоснованности предположений, использованных далее и в разделе 3.1.1 «Конкретизация».

Следует отметить также, что, начиная с этого этапа описание модели прекращает быть строго математическим.

Элементарный шаг итерации поведения рассматривается в виде *выбора и совершения особью действий*. Здесь «**действие**» определяется как именованное детерминированное преобразование текущего состояния особи в последующее[15]. Набор доступных для особи действий конечен и определён на этапе построения модели.

Выбор особью действия для совершения заключается в сопоставлении особью каждому доступному для выбора действию численного значения, называемого «**приоритетом**», и выборе из оценённых таким образом действий действия, получившего наибольший приоритет. Здесь основным предположением модели является предположение о том, что *особь делает выбор, основываясь только на текущих значениях своих параметров*.

Правила, по которым вычисляются приоритеты действий, предполагаются извлечёнными из психологии, социологии или иной другой эмпирической науки, связанной с изучением поведения. В данной реализации модели они построены искусственным образом на основе бытовых предположений о поведении, более подробно это рассмотрено в разделе 3.1.1 «Конкретизация». Автором делались попытки использовать более научно обоснованные предположения о поведении, например, из [4], но все найденные сведения основывались на использовании *памяти* субъекта поведения, что выходит за рамки текущих ограничений модели.

[14] Автор рассматривает поведение в широком смысле: не обязательно человеческого, но любой активной сущности, способной выбирать действие для последующего совершения. Однако, если угодно, всё данное исследование можно рассматривать в рамках человеческого поведения.

[15] Таким образом, вообще говоря, «действие» может с тем же успехом называться «событием»

Основное предположение модели – о том, что выбор действия зависит только от текущих значений параметров особи – имитирует сиюминутные стремления особи. В данной модели субъект поведения не строит далеко идущих планов и не ставит целей, которых хочет достичь.

Правила вычисления приоритета действий называются далее «**правилами выбора**».

Правила выбора *полностью* описывают *все* предположения о поведении особи в модели. Так как подобные предположения обычно записываются на естественном языке, будет удобным использовать для их формализации аппарат нечёткой логики.

2.4.1 Нечёткая логика

Более подробно о нечёткой логике можно узнать из первоисточника – работ L. A. Zadeh [7], [8] и [9] (или их перевода [16]), или из использованной автором источника: [6], раздел «Fuzzy Sets». Там же находится более подробное, строгое математическое описание работы нечёткого вывода

Ниже кратко изложены концепции нечёткой логики, необходимые для реализации модели. Даже если нижеприведённое изложение несколько отличается от приведённого в литературе, *именно в такой* интерпретации нечёткая логика применялась автором при реализации модели.

Для каждого параметра определяется набор связанных с ним т. н. «**нечётких значений**[16]». Нечёткое значение параметра, вообще говоря, является *именованной функцией*, определённой на значениях этого параметра, и возвращающей число, обычно действительное от 0 до 1, семантически представляющее собой степень соответствия текущего значения параметра некоторой лексической концепции – субъективной оценке, применимой к значениям данного параметра. Исходные значения параметра, над которым определено множество нечётких значений, называются «**чёткими значениями**» этого параметра.

[16] Иначе «**лингвистических значений**»

Именованная числовая величина, над чёткими значениями которой определен набор нечётких значений, называется далее «**нечёткой переменной**». Далее по тексту определённые над величиной нечёткие значения часто будут называться «**нечёткими оценками**» или просто «**оценками**» этой величины.

Связанная с нечётким значением переменной функция, по чёткому значению переменной возвращающая соответствие чёткого значения нечёткому, называется функцией принадлежности[17]. Нечёткая переменная, вообще говоря, при заданном чётком значении имеет одновременно *все* свои нечёткие значения, но каждое – с определённой степенью соответствия.

Нечёткие значения нечёткой переменной являются субъективными оценками чёткого значения этой переменной. В зависимости от того, как определены функции принадлежности нечётких значений, одно и то же чёткое значение нечёткой переменной может больше соответствовать различным нечётким значениям этой переменной.

Рис. 2.2. Пример графического представления нечёткой переменной.
Чёткое значение 23 является «низким» с соответствием 0.75,
и одновременно «высоким» с соответствием 0.16.

Для нечёткой переменной можно также не задать *чёткого* значения, но задать *нечёткое* определить на множестве её чётких значений произвольную интегрируемую функцию, возвращающую число от 0 до 1. В таком случае

[17] англ. membership function

возможно вычислить чёткое значение переменной на основании нечёткого. Это вычисление называется операцией дефаззификации[18]. Оператор дефаззификации может быть введён произвольно; в схеме нечёткого контроллера по Mamdani[19], которая используется в данной реализации модели, дефаззификация производится вычислением абсциссы центра масс фигуры под кривой графика функции принадлежности нечёткого значения переменной.

Нечёткий логический вывод основан на обобщении правила Modus Ponens классической логики, которое записывается так:

предпосылка 1 (факт):	*x является A*
предпосылка 2 (правило):	*если x является A, то y является B*
следствие:	*y является B*

Обобщение вышеуказанного правила выглядит так:

предпосылка 1 (факт):	*x является A^**
предпосылка 2 (правило):	*если x является A, то y является B*
следствие:	*y является B^**

где A^* это краткая запись для выражения «A со степенью соответствия α» и B^* это краткая запись для выражения «B со степенью соответствия α» (то есть, очевидно, *с той же степенью соответствия*).

Modus Ponens в классической двузначной логике, таким образом, является частным случаем, в котором существуют только две степени соответствия: 0 и 1 (да/нет, истина/ложь).

Нечёткое правило «если-то» определяется так:

если x является A, то y является B

где x и y – нечёткие переменные, A и B – нечёткие значения, причём, естественно, A определено на x, а B определено на y.

[18] англ. defuzzyfication

[19] [6, стр. 74]. Подробнее про понятие нечёткого контроллера дальше по тексту. Mamdani – фамилия автора.

Часть правила, расположенная после ключевого слова «если» и до ключевого слова «то», называется «**предпосылкой**[20]» правила. Часть правила, расположенная после ключевого слова «то», называется «**последствием**[21]» правила.

Правило может иметь несколько предпосылок, в таком случае они считаются соединёнными логическим оператором «И». Если они семантически должны быть соединены оператором «ИЛИ», то такое правило эквивалентно набору правил с разными предпосылками, но одинаковыми последствиями.

Если правило имеет несколько последствий, они считаются соединёнными логическим оператором «И», причём такое правило эквивалентно набору правил с одинаковыми предпосылками, но разными последствиями. Применение логического оператора «ИЛИ» к последствиям правила смысла не имеет.

Смысл логических операторов «И» и «ИЛИ» в нечёткой логике определяется произвольным образом[22], учитывая то, что применяться они будут к значениям соответствия чёткого значения нечёткому, то есть, к действительным числам. Таким образом, эти операторы уже не традиционные булевы функции, но какие-либо действительнозначные непрерывные. В вышеупомянутой схеме Mamdani, используемой в данной модели, оператору «И» соответствует умножение.

Символьная система, использующая набор нечётких переменных в виде своих входных значений, другой набор нечётких переменных в виде своих выходных значений и набор нечётких правил, устанавливающих связь между значениями входных переменных и значениями выходных переменных, называется «**нечётким контро́ллером**». Нечёткий контроллер, при условии определения над входными и выходными переменными нечётких значений, которые соответствуют нечётким правилам, заданным для контроллера, является, вообще говоря, *функцией*, по заданному набору *чётких* значений входных переменных возвращающей набор *чётких* значений выходных переменных. Вся субъективность, подразумевающаяся

[20] англ. antecedent

[21] англ. consequence

[22] Но всё же с учётом некоторых особенностей; подробности в [6]

нечёткими правилами, приближёнными к естественным рассуждениям, скрывается внутри контроллера.

$$\varphi(\vec{\alpha}) = \vec{\beta} \tag{2.9}$$

$$\varphi : R^n \to R^m$$

$$\vec{\alpha} \in R^n, \vec{\beta} \in R^m$$

Здесь φ – функция, представляемая нечётким контроллером, вектор α – вектор (чётких) значений входных переменных, вектор β – вектор (чётких) значений выходных переменных.

Алгоритм работы нечёткого контроллера по схеме Mamdani состоит из следующих шагов, применяемых для каждой выходной переменной (далее называемой «**выводящаяся выходная переменная**»):

1) Отбираются все правила, в последствиях которых имеется упоминание выводящейся выходной переменной. Каждое правило с несколькими последствиями при этом разбивается на набор правил с разными последствиями, но одинаковыми предпосылками. Наборы правил с одинаковыми последствиями, но разными предпосылками объединяются в одно правило с общим последствием, и всеми предпосылками, связанными оператором «И».
2) Вычисляются значения соответствия для каждой предпосылки каждого отобранного на шаге 1 правила.
3) Для каждого обработанного на шаге 2 правила вычисляется т. н. **firing strength**[23] (в данном контексте скорее «**значимость**» правила) как произведение всех степеней соответствия всех предпосылок этого правила.
4) Функция принадлежности нечёткого значения последствия у каждого правила умножается на значимость этого правила.
5) Из всех правил, имеющих одинаковое последствие, выводится одно последствие в виде нечёткого значения выводящейся нечёткой переменной,

[23] Дословно «сила выстрела» или, если угодно, «огневая мощь».

функция принадлежности которого является минимумом среди функций принадлежностей этих правил[24].

6) После выполнения шага 5 для выводящейся выходной переменной оказывается сформирован набор т. н. «**уточнённых нечётких значений[25]».** Из функций принадлежности этих нечётких значений формируется единое нечёткое значение выводящейся выходной переменной в виде максимума этих функций[26].
7) Для функции принадлежности полученного нечёткого значения вычисляется абсцисса центра масс фигуры под кривой графика этой функции. Этот шаг называется дефаззификацией. Результат дефаззификации является чётким значением выводящейся выходной нечёткой переменной.

Вышеперечисленные действия выполняются для каждой выходной переменной, в результате чего вычисляются чёткие (то есть, числовые) значения всех выходных переменных нечёткого контроллера, что и требовалось найти.

2.4.2 Применение нечёткой логики для формализации поведения

Аппарат нечёткой логики позволяет определять связь между параметрами особи и приоритетами действий не в виде функциональных зависимостей[27], а на более близком к естественному языку уровне (хотя и всё равно не в произвольной форме).

При использовании нечёткой логики правила выбора будут иметь вид:

если параметр P имеет нечёткое значение Vp,

то приоритет действия A имеет нечёткое значение Va

[24] Имеется в виду минимум среди *функций*, то есть, грубо говоря, нижняя огибающая их графиков, буде они совмещены на одной координатной плоскости.

[25] англ. qualified consequent membership functions, при этом понятно, что «функции принадлежностей последствия» являются эквивалентами неименованных нечётких значений последствия.

[26] Аналогично минимуму функций – верхняя огибающая графиков функций, совмещённых на одной координатной плоскости.

[27] Которые сложно, если вообще возможно, использовать для чего бы то ни было, относящегося к живым существам.

где *Vp* и *Va* – термины естественного языка наподобие «большое», «маленькое», «среднее», или даже более того: «синее», «горячее», «быстрое» и т. п. Далее, в зависимости от контекста, нечёткие значения параметра называются «**оценками**» параметра.

Выбор действия для совершения осуществляется запуском нечёткого контроллера, который на основе заданных параметров особи и правил выбора оценивает приоритеты действий, доступных для особи (вообще говоря, известных модели), и *выбором среди оценённых действий действия, получившего наибольший приоритет*. При этом для корректного функционирования генетического алгоритма требуется, чтобы в случае получения более одного действия с вычисленным наибольшим приоритетом, модель выбирала одно из них чётко определённым, предсказуемым образом[28].

Основное преимущество, которое даёт нечёткая логика по сравнению с классической двузначной – возможность оперировать *приблизительными* оценками, причём эти оценки вводятся произвольным образом, предоставляя возможность имитировать в механизме рассуждений основную черту человеческого мышления: *субъективность*.

В данной модели автором введена имитация **субъективности оценки особью значений своих параметров**, следующим образом.

2.4.2.1 Имитация самооценки

Предполагается, что набор нечётких значений в смысле лексических термов определён на этапе построения модели. Однако функции принадлежности нечётких значений параметров являются *параметризированными* непрерывными функциями от одного действительного аргумента (чёткого значения параметра). Далее следует различать «параметры особи» и «параметры функций принадлежности нечётких значений» параметров особи.

[28] Иначе функция вычисления приспособленности перестанет быть функциональной зависимостью значения приспособленности от значений склонностей, и решение задачи потеряет смысл.

Выполнение функции u (преобразовывающей набор значений склонностей особи в начальное состояние особи) включает в себя не только формирование чётких значений параметров особи на основе значений склонностей, но и вычисление параметров функций принадлежности нечётких значений, определённых на параметрах особи.

Все нечёткие значения, определённые на параметрах особи, считаются заданными функцией принадлежности следующего вида[29]:

$$g(x,c,\sigma) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2} \tag{2.10}$$

Параметр c определяет абсциссу вершины графика этой функции, параметр σ влияет на её размах (расстояние от абсциссы вершины графика до значения аргумента, при котором значение функции будет практически неотличимо от нуля).

С помощью следующего коэффициента можно, зная параметры функции, получить абсциссы точек, в которых эта функция принадлежности будет равна ½,:

$$e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2} \Rightarrow \left[\begin{array}{l} x = c - \sqrt{\sigma^2}\sqrt{2\cdot\log(2)} \\ x = c + \sqrt{\sigma^2}\sqrt{2\cdot\log(2)} \end{array}\right.$$

$$\sqrt{2\cdot\log(2)} \approx 1.17741 \equiv \delta \tag{2.11}$$

Управляя абсциссами вершины и точек, в которых эта функция равна ½, можно контролировать вид графика функции принадлежности. Таким образом, это позволит строить различные нечёткие оценки параметров особи в зависимости, от, например, склонностей особи. Именно для этого ранее и была введена концепция влияния склонностей на параметр.

Далее вводится функция, условно называемая «**функцией распределения опорных точек**» нечётких значений.

$$f(x,a,b) = \frac{x^a}{b^{a-1}} \tag{2.12}$$

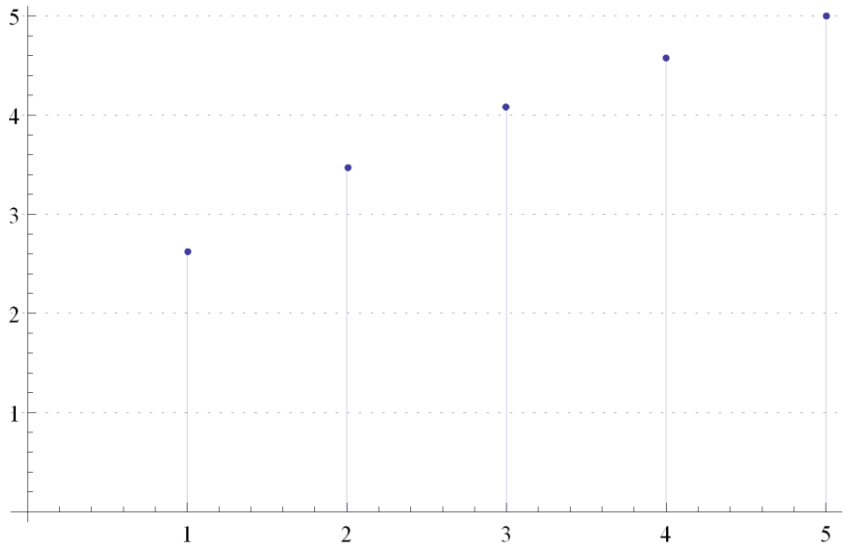
Эта функция интерпретируется следующим образом: число f является показателем положения точки номер x на некоторой числовой оси. Вычислив f для

[29] Это функция Гаусса, она также используется в теории вероятностей.

нескольких номеров, можно получить набор точек, распределённых по прямой через некоторые промежутки, длина которых зависит от a и b.

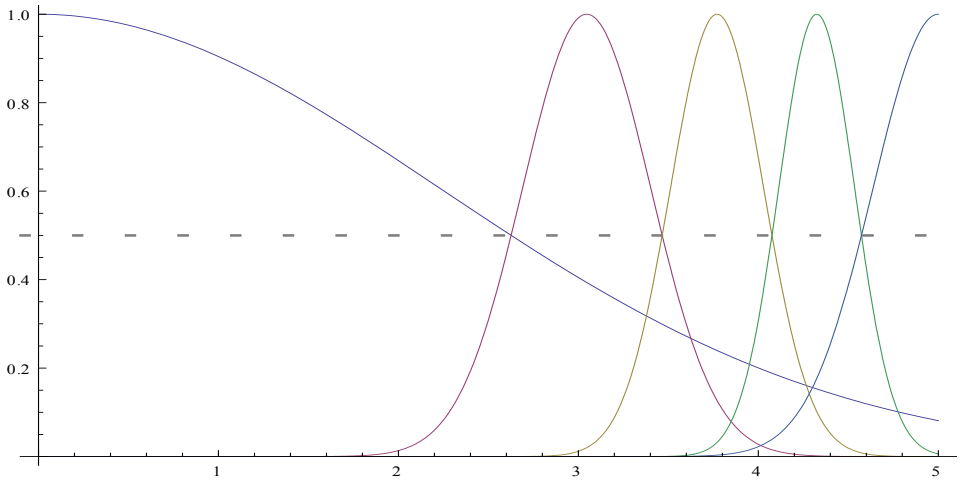
Параметр b определяет номер точки, которая, грубо говоря, фиксирует границу распределения точек.

Чем параметр a больше единицы, тем точки более сгруппированы к точке номер b (см. рис., a = 0.4).

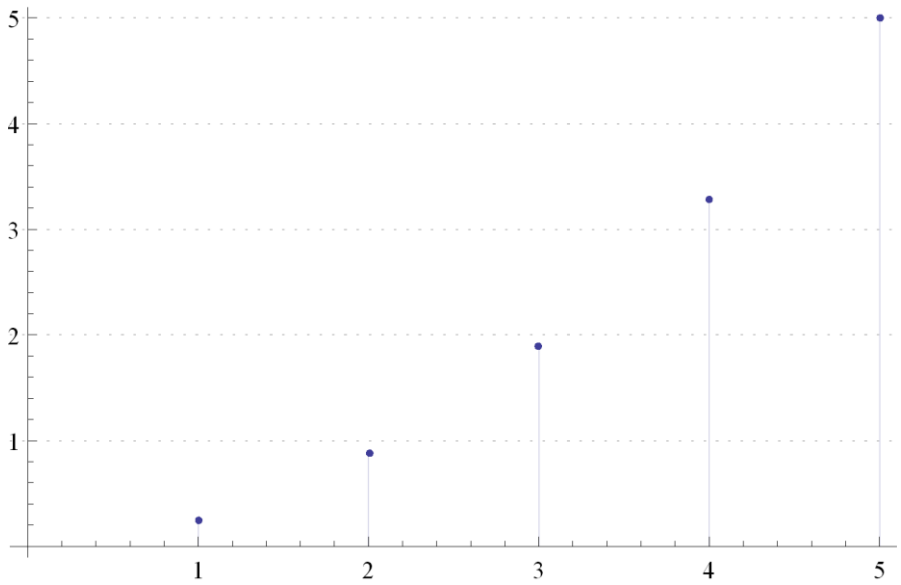


Здесь по горизонтальной оси отложены номера точек, а по вертикальной – относительные значения меры расположения точки, которые можно отмасштабировать до получения отрезка, на котором определён параметр особи.

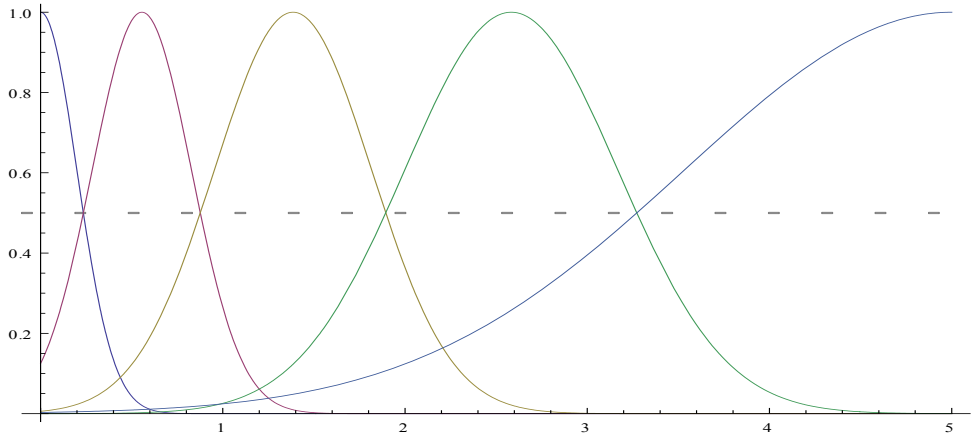
Забегая вперёд, по этому распределению точек можно получить следующее распределение функций принадлежности нечётким значениям:



Чем параметр a больше единицы, тем точки более сгруппированы к нулю (см. рис., a = 1.9).



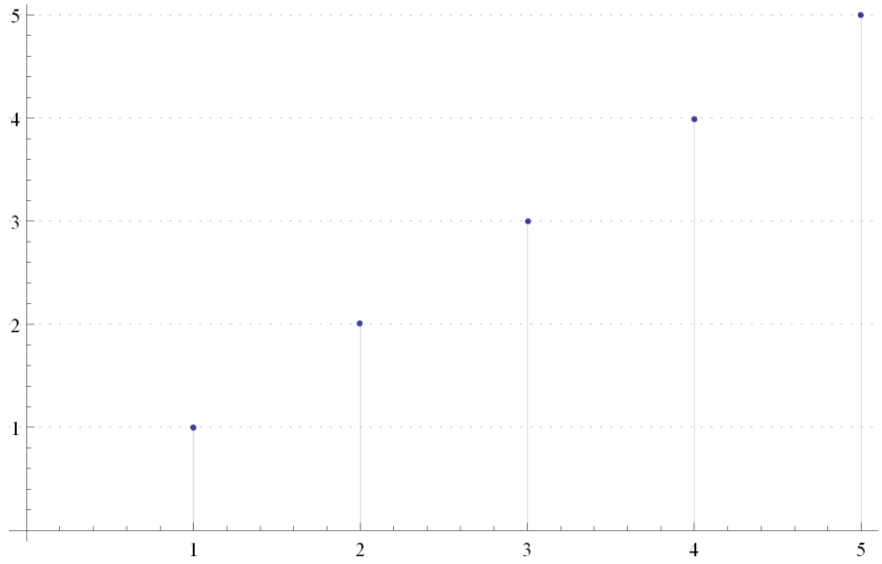
Таким образом, функции принадлежности будут распределены так:

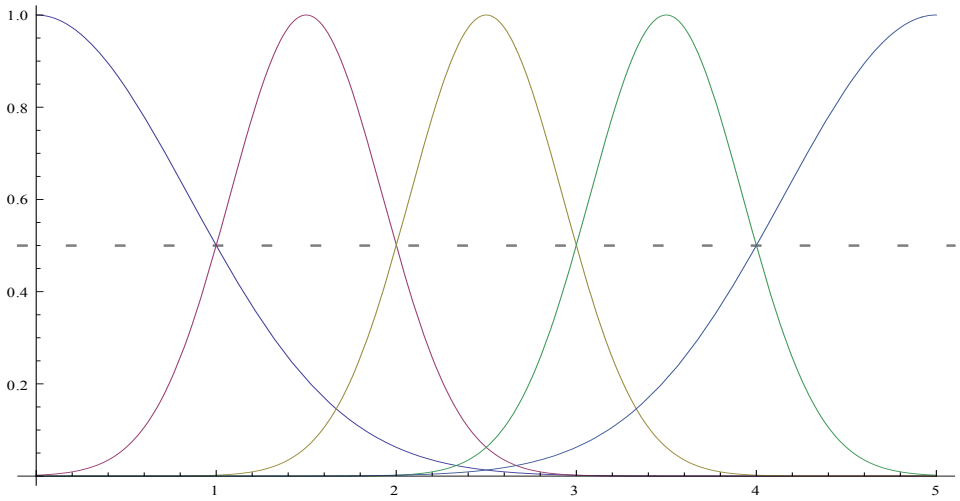


Отрицательные значения a, равно как и дробные значения b, не рассматриваются, так как не имеют смысла в данной интерпретации.

Пользуясь функцией распределения опорных точек, далее получаем абсциссы точек, которые используются как точки, в которых функции принадлежности нечётких значений равны ½. Зная коэффициент δ, и учитывая симметричность графика функции Гаусса относительно вершины, легко по имеющимся координатам восстановить абсциссы вершин этих функций. Зная абсциссу вершины и размах, вычисленный исходя из коэффициента δ и абсциссы вершины, получаем все необходимые параметры функции принадлежности, которую можно затем использовать для определения нечёткого значения на параметре.

Пример распределения опорных точек (а вместе с ними и функций принадлежности) при a = 1 показан на следующей паре рисунков.





Данное распределение функций принадлежности используется в модели при нечётких оценках приоритетов действий и конечного состояния особи, не зависящих от склонностей конкретной особи. Значения по горизонтальной оси далее масштабируются, для соответствия пределам оцениваемого параметра. Смысл используемых пяти функций принадлежности нечётким значениям определён в разделе 3.1.1 «Конкретизация».

Дабы вычислить параметр a функции распределения опорных точек, для каждого параметра особи определяется набор весовых коэффициентов, каждый из которых ставится в соответствие одной из склонностей. На этапе построения начального состояния особи при рассмотрении каждого параметра особи значения

склонностей умножаются каждый на свой весовой коэффициент (определённый для рассматриваемого параметра), и полученные значения передаются функции следующего вида:

$$\lambda(\iota, a) = \frac{(\bar{\iota} - a)^2}{a} + \frac{1}{a} \qquad (2.13)$$

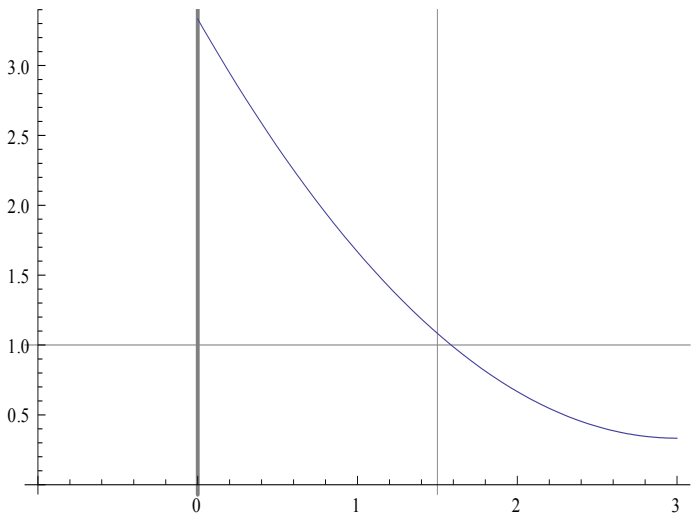
$$\bar{\iota} = \frac{\sum_{k=1}^{m} i_k}{m}$$

Где ι – кортеж значений склонностей, как он определён в формуле (2.5), раздел 2.2 «Формулировка математической модели», но эти значения домножены каждое на коэффициент, зависящий от склонности и параметра, на который эта склонность влияет. Таким образом, какие-то склонности влияют более сильно на одни параметры, а какие-то другие склонности – на другие параметры. Параметр a – максимальное значение склонностей. Благодаря этому параметру лямбда масштабируется, не меняя формы.

Эта функция (далее «**лямбда**») выбрана из следующих соображений:

1) Лямбда - положительное действительное, близкое к 1;
2) Если всех склонности имеют среднее значение, то лямбда равна 1;
3) Если все склонности выше среднего, лямбда меньше 1;
4) Если все склонности ниже среднего, лямбда больше 1;
5) Если все склонности имеют предельное значение, ламбда равна 1/a;
6) Если все склонности имеют нулевое значение, ламбда равна a;

Определение (2.14) соответствует условиям с точностью до долей единицы в упомянутых параметрах.



Таким образом, для каждого параметра особи лямбда является условно значением «**влияния склонностей**» на этот параметр. Значение лямбда передаётся в

качестве параметра a в функцию распределения опорных точек, меняя её форму и влияя на распределение функций принадлежности нечётких оценок параметра, зависимого от данного лямбда.

2.5 Оценка приспособленности

Конечное состояние каждой особи, поведение которой было сымитировано функцией поведения, сравнивается с желаемым конечным состоянием следующей функцией:

$$\tilde{f}(\rho,\tilde{\rho}) = 1 - \sqrt{\frac{\sum_{k=1}^{n}(p_k - \tilde{p}_k)^2}{\sum_{k=1}^{n}(\overline{p}_k - \underline{p}_k)^2}} \qquad (2.14)$$

где все обозначения взяты из формул (2.2) и (2.3), раздел 2.2 «Формулировка математической модели». Значение ρ является результатом выполнения функции поведения $\varsigma_r(i)$.

То есть, приспособленность определяется как отношение расстояния между текущим значением вектора параметров и желаемым значением вектора параметров к максимальному возможному в пространстве параметров расстоянию – от вектора минимальных значений параметров до вектора максимальных значений параметров. Расстояние определяется согласно метрике ρ_2. Максимальное расстояние на пространстве параметров вычислимо, так как пространство ограниченное.

Данное отношение лежит в отрезке от 0 до 1, но тем меньше, чем меньше расстояние между текущими (конечными) параметрами особи и желаемыми, что является обратным по отношению к смыслу оценки приспособленности. Поэтому значение этого отношения вычитается из 1.

3 Практическая часть

В разделе 2 «Теоретическая часть» были рассмотрены теоретические основы методов построения модели и работы с ней. Также был определён каркас алгоритма проводимых вычислений. Теперь рассмотрим реализацию этого алгоритма на конкретном языке программирования, и проанализируем результаты, которые возвращает модель, реализованная в виде компьютерной программы.

3.1 Реализация

Целевым языком программирования был выбран язык Visual Prolog 7.1, как язык с поддержкой возможности мультипарадигменного программирования и обширной библиотекой стандартных классов. Несколько более подробно об этом языке рассказано в разделе 3.1.2 «Visual Prolog 7.1». Также на выбор языка существенно повлиял факт наличия готового пакета классов для реализации генетических алгоритмов, написанный на Visual Prolog 7.1 (об этом сказано в разделе 3.1.3.1 «Пакет классов для генетического алгоритма»).

3.1.1 Конкретизация

Прежде чем перейти к описанию конечной компьютерной программы, реализующей модель, необходимо уточнить начальные условия, определив предметную область задачи.

Для данной реализации модели будем исходить из следующих соображений[30].

Особь является человеческим существом, живущим в городе, расположенном в сказочном («фэнтезийном», если угодно) мире. Каждая итерация поведения соответствует одному прожитому дню, за который особь работает, учится или отдыхает.

[30] Предпосылки, подвигнувшие автора на то, чтобы использовать подобную предметную область в качестве примера моделирования поведения, изложены в Приложении А «Личные мотивы»

Особь описывается следующими 29 именованными числовыми параметрами.

Идентификатор параметра	Семантика параметра
InclPhys	Склонность к физическому развитию
inclCogn	Склонность к познавательной деятельности
inclCreat	Склонность к творчеству
inclMasc	Мужеподобность
inclFem	Женственность
inclIntr	Замкнутость
inclSex	Либидо
inclAggr	Агрессивность
attrSTR	Физическая сила
attrCON	Выносливость
attrINT	Интеллект
attrREF	Изящность
attrCHA	Обаяние
attrETH	Этичность (стойкость мировоззрения)
attrAMR	Аморальность (степень различия между личной этикой и общественной)
attrSEN	Чувствительность
skillDec	Навыки этикета
skillArt	Искусствознание
skillCook	Навыки приготовления пищи
skillClean	Навыки уборки по дому
skillConv	Навыки ведения беседы
skillCombSkl	Навыки ведения рукопашного боя
skillCombAtt	Навыки нападения в рукопашном бою
skillCombDef	Навыки обороны в рукопашном бою
skillMgrSkl	Навыки колдовства
skillMgrAtt	Навыки нападения в бою магией
skillMgrDef	Навыки обороны в бою магией
moneyQt	Количество денег
stressLv	Численный показатель уровня стресса

Первые 8 параметров, имеющие префикс *incl-*, выбраны в качестве склонностей особи (они, собственно, для этого и были введены в оцифровку

особи). И в качестве параметров, и в качестве склонностей они принадлежат отрезку [0, 3]. Значения параметров, имеющих префикс *attr-*, принадлежат отрезку [0, 900]. Значения параметров, имеющих префикс *skill-*, принадлежат отрезку [0, 450]. $moneyQt \in [0,10000]$, $stressLv \in [0,100]$.

Таким образом, геном особи для генетического алгоритма – битовая последовательность 8 групп по 2 бита (2 бита кодируют 4 значения 0..3), всего 16 бит. Это даёт 4^8 = 65536 вариантов особей. Несмотря на то, что пространство поиска относительно невелико, следует учитывать то, что сложность вычисления функции приспособленности для каждой особи зависит от количества итераций функции поведения.

Действия, доступные особи, делятся на три категории: различные виды профессиональной деятельности, учёба различным концепциям и единственное действие отдыха. Всего возможных действий 26.

Идентификатор действия	Семантика идентификатора
jobMaid	Работа домработницей
jobBabysitter	Работа сиделкой
jobInn	Работа прислугой в гостинице
jobFarm	Работа на ферме
jobChurch	Работа в церкви
jobRestaurant	Работа поваром в ресторане
jobLumber	Работа на лесопилке
jobSalon	Работа стилистом в парикмахерской
jobMason	Работа на стройке
jobHunter	Работа охотником
jobCemetery	Работа кладбищенским смотрителем
jobTutor	Работа репетитором
jobBar	Работа барменом
jobBordello	Работа в борделе
jobCabaret	Работа танцовщицей в кабаре
studyScience	Обучение естественным наукам
studyStrategy	Обучение стратегии
studyTeology	Обучение теологии
studyPoetry	Обучение поэзии

Идентификатор действия	Семантика идентификатора
studyFighting	Обучение рукопашному бою
studyFencing	Обучение фехтованию
studyMagic	Обучение магии
studyEtiquette	Обучение этикету
studyArts	Обучение искусствоведению
studyDancin	Обучение танцам
actLeizure	Отдых

Каждая итерация поведения соответствует одному прожитому дню, за который особь работает, учится или отдыхает. Работа значительно увеличивает значение стресса, увеличивает количество денег и увеличивает значения некоторых (зависит от вида деятельности) параметров за счёт, возможно, незначительного уменьшения других. Учёба значительно уменьшает количество денег, увеличивает значение стресса и изменяет значения параметров таким же образом, что и работа.

Последствия действий определены так, что выполнение действий в среднем *увеличивает* значения параметров (позитивные эффекты значительно превосходят негативные).

Отдых уменьшает значение стресса, увеличивает параметр attrAMR и уменьшает количество денег на значения, зависящие от текущего количества денег у особи.

Полное описание последствий всех действий приведено в Приложении В «Последствия действий»

И параметры особи, и приоритеты действий оцениваются следующим набором нечётких значений:

Идентификатор значения	Семантика идентификатора
minimal	Крайне низкое значение
low	Низкое значение
medium	Среднее значение
high	Высокое значение
overpower	Крайне высокое значение

Весовые коэффициенты для влияния склонностей на параметры приведены в Приложении С «Влияние склонностей на параметры».

Для упрощения формирования набора правил выбора действия была введены концепции «**прямой зависимости**» и «**обратной зависимости**» приоритета действия от параметра особи. Пользуясь этими концепциями, возможно определять правила на уровне высказываний

«чем выше значение параметра P, тем выше приоритет действия A»

для прямой зависимости и

«чем выше значение параметра P, тем ниже приоритет действия A»

для обратной зависимости.

Эти концепции основываются на том, что и для параметров, и для приоритетов определён одинаковый набор нечётких значений, являющийся градацией от условно «маленького» значения до «большого».

Если требуется определить прямую зависимость приоритета действия *A* от значения параметра *P*, то вводятся нечёткие правила «если-то» по числу градаций (в данном случае 5), вида

если параметр P имеет значение V,

то приоритет действия A имеет то же самое значение V.

Таким образом, прямая зависимость означает наличие нечётких правил, связывающих одинаковые нечёткие значения параметра и приоритета действия.

Если требуется определить обратную зависимость, то вводится набор правил, соотносящий нечёткие значения параметра и приоритета действия по следующей схеме:

termMinimal → termOverpower

termLow → termHigh

termMedium → termMedium

termHigh → termLow

termOverpower → termMinimal

Итоговое количество правил выбора в использованной конкретизации модели – 290, сгруппированных в 58 прямых и обратных зависимостей. Полный список правил находится в Приложении D «Правила выбора».

Правила основывались на следующих предположениях:

1. особь стремится к максимальной диверсификации своего развития: чем выше значение какого-либо параметра, тем ниже приоритеты действий, в результате выполнения которых значение этого параметра увеличивается;
2. чем выше значение стресса, тем выше приоритет отдыха и тем ниже приоритет всех остальных действий (так имитируется усталость[31]);
3. чем меньше количество денег, тем выше приоритет работы, чем больше количество денег, тем выше приоритет учёбы.

Во всех вышеперечисленных предположениях если не сказано о наличии негативного влияния на приоритет, значит, таковое отсутствует.

Кроме собственно решения задачи, поставленной в разделе 1.2 «Постановка задачи», интересно было бы получить список действий, которые совершит особь вследствие своего поведения, а также вообще получить отчёт об изменениях, произошедших с особью. Поэтому дополнительно к основной перед программой ставится задача о протоколировании своей деятельности в текстовый файл.

Ради того, чтобы следовать принципам нечёткой логики – определения значений параметров в виде субъективных оценок – желаемое конечное состояние особи будет задаваться в виде нечётких значений параметров[32]. Для получения приспособленности будут использоваться чёткие желаемые значения параметров, дефаззифицированные по Mamdani (получением абсциссы центра

[31] В случае, если выбор действия имитируется в масштабах нескольких дней, аналогом параметра stressLv будет скорее «нервное напряжение», нежели «усталость».

[32] Эти нечёткие значения не будут зависеть от склонностей особи: в данном случае подразумевается, что оценивать конечное состояние особи будет уже некая другая сущность, производящая оценку единообразно для всех особей.

масс фигуры под кривой графика функции принадлежности заданной нечёткой оценки).

Допускается указание не всех параметров из желаемого конечного состояния особи. В таком случае неуказанные параметры считаются имеющими такое же желаемые значение, как и то, которого особь достигла, то есть, неуказанные пожелания считаются удовлетворёнными.

3.1.2 Visual Prolog 7.1

Visual Prolog [10] – объектно-ориентированный язык программирования, являющийся прямым потомком известного в своё время декларативного языка ISO/PDC Prolog [14] [15]. Компилятор и интегрированная среда разработки, использующая Visual Prolog, распространяется под проприетарной лицензией (версия «для личного пользования», с несколько ограниченными возможностями, бесплатна), свободных аналогов нет. Язык полностью унаследовал все возможности декларативного ISO/PDC Prolog'a, такие, как поддержка логики предикатов на уровне языка, pattern matching и работу со списками произвольной длины. Являясь объектно-ориентированным расширением языка-предшественника, Visual Prolog поддерживает также возможность мультипарадигменного программирования – декларативно-императивного, или же полностью процедурного (с некоторыми незначительными языковыми особенностями).

3.1.3 Ядро логики

Ядро логики, согласно идеологии целевого языка, разбито на пакеты классов. Каждый пакет содержит взаимосвязанный набор классов, не обязательно замкнутый, но сама концепция пакетов классов предполагает замкнутость пакета.

Данная научная работа не предполагает написание дизайн-документа для программы, реализующей модель, поэтому ниже приведено лишь краткое описание функционала разработанных классов и связь их с каркасом модели, построенным в разделе 2 «Теоретическая часть».

Построенная автором программная реализация модели подразумевает, что перенастройка конкретизации под другую задачу заключается в работе непосредственно с исходным кодом программы. Следует заметить, что, хотя автор и объявляет свою работу принадлежащей public domain[33], но компилятор (равно как и интегрированная среда разработки) Visual Prolog 7.1 являются проприетарными.

3.1.3.1 Пакет классов каркаса логики

Содержит классы ***alife*** и ***alifeSettings***. Также содержит определение интерфейсов ***individual*** и ***reality***, классы объектов для которых определены в пакете классов конкретизации задачи (раздел 3.1.3.2).

Класс *alifeSettings* предназначен для хранения ключевых параметров самой модели и программы, её реализующей:

- желаемого конечного состояния особи;
- количества итераций функции поведения («продолжительности жизни»);
- точности вычислений в нечётком контроллере (см. раздел 3.1.3.3 «Пакет классов для нечёткой логики»);
- имена файлов отчётов, которые производит программа в процессе работы.

Класс *alife* содержит реализацию вычислений функции приспособленности. В нём активно используются классы из пакета классов для нечёткой логики (см. раздел 3.1.3.3). *alife* экспортирует две функции:

- initReality – предназначена для инициализации объекта класса *reality*, инкапсулирующего в себе определения правил выбора и оценки конечного состояния особи (класс *reality* определён в пакете классов для конкретизации задачи, раздел 3.1.3.2)
- simulateLife – является собственно функцией приспособленности, получающей в качестве входного аргумента кортеж значений

[33] То есть отказывается от любых притязаний, основанных на понятии «интеллектуальная собственность» и связанных с его работой.

склонностей особи и возвращающей через своё имя числовое значение приспособленности рассмотренной особи сохранённому в классе *alifeSettings* желаемому конечному состоянию.

Интерфейс *individual* определяет требования к классу объектов *individual*, инкапсулирующих в себе описание субъектов поведения (собственно, особей) в терминах предметной области. Согласно определению интерфейса, класс *individual* должен предоставлять возможность

- инициализировать параметры особи по заданному набору склонностей
- получить сведения о параметрах особи
- получить от особи нечёткую самооценку своих параметров
- выполнить особью какое-либо действие по указанному идентификатору действия.

Интерфейс *reality* определяет требования к классу объектов *reality*, инкапсулирующих в себе описание предметной области, в которой рассматривается поведение особи (далее условно «**реальность**»). Согласно определению интерфейса, класс *reality* должен предоставлять возможность:

- получить список правил выбора особью действий
- получить список сгенерированных нечётких переменных, представляющих собой приоритеты действий, для дальнейшего использования его в нечётком выводе
- произвести нечёткую оценку текущих параметров особи для использования этой оценки в вычислении приспособленности особи к желаемому конечному состоянию.

Предполагается, что до вычисления приспособленностей при помощи класса alife будет сгенерирован единственный объект класса reality. Функция simulateLife производит следующие действия.

1. Инициализирует файлы отчётов при помощи класса *logger* (раздел 3.1.3.5 «Пакет классов для формирования отчётов»).
2. Создаёт объект класса *individual* и инициализирует его, используя полученный кортеж значений склонностей.

3. Получает от созданного объекта особи нечёткие переменные – параметры особи, оценённые ею самой.
4. Получает от существующего объекта реальности правила выбора и нечёткие переменные – приоритеты действий, оценённые реальностью.
5. Создаёт нечёткий контроллер в виде объекта класса *fuzzyInference* (раздел 3.1.3.3 «Пакет классов для нечёткой логики»), инициализируя его полученными только что параметрами особи, правилами выбора и приоритетами действий.
6. Получает из класса *alifeSettings* количество итераций выбора и запускает внутреннюю рекурсивную функцию repeatChoice (определение которой здесь не имеет значения), которая будет итерировать процесс выбора и совершения особью действий. Этой функции передаются созданные нечёткий контроллер, особь и количество итераций. На выходе repeatChoice возвращает достигшую своего конечного состояния особь.
7. Передаёт полученный объект особи существующему объекту реальности, который самостоятельно производит нечёткую оценку конечных значений параметров особи.
8. Получает из класса *alifeSettings* желаемое конечное состояние особи и передаёт объект реальности, оценивший текущее конечное состояние особи, функции evaluateResults (её дальнейшая детализация тоже не имеет значения), которая производит сравнение и вычисляет значение приспособленности особи.

Таким образом, сама функция simulateLife является реализацией целевой функции модели $f_{\tilde{\rho}}(\rho(\iota))$, функция repeatChoice является реализацией функции поведения $\varsigma_r(\iota)$, а функция evaluateResults является реализацией функции приспособленности $f_{\tilde{\rho}}(\rho(\iota)) \equiv \widetilde{f}(\varsigma_r(\iota), \widetilde{\rho})$.

3.1.3.2 Пакет классов для конкретизации задачи

Содержит определения классов ***cell***, ***individual*** и ***reality***, наследующих одноимённые интерфейсы из пакетов классов для генетического алгоритма (раздел 3.1.3.4) и каркаса логики (раздел 3.1.3.1).

Класс *cell* представляет собой формализацию объекта генома, использующегося в генетическом алгоритме. Согласно описанию своего интерфейса, предоставляет функции

- мутации генома
- кроссовера с иным геномом
- вычисления приспособленности
- замены генома на иной
- присвоения геному случайного значения
- декодирования генома с целью получения кортежа значений склонностей соответствующей геному особи

Класс *individual* представляет собой формализацию объекта особи, использующегося при вычислении приспособленности (итерировании процесса выбора/совершения действий) особи. Согласно описанию своего интерфейса, предоставляет функции

- совершения действия по его идентификатору
- получения набора нечётких переменных, представляющих собой параметры особи с определёнными на них нечёткими значениями, вычисленными в зависимости от склонностей особи (см. раздел 2.4.2.1 «Имитация самооценки»)
- получения сведений о параметрах особи: идентификатор /минимальное/максимальное/текущее чёткое значение

Класс *reality* представляет собой удобную обёртку для хранения правил выбора, одинаковых для всех особей, и осуществления независимой от склонностей особи оценки её параметров, а также оценки приоритетов действий. Согласно описанию своего интерфейса, класс reality предоставляет функции

- получения списка правил выбора
- осуществления независимой от самой особи нечёткой оценки значений параметров особи
- получения ранее оценённых реальностью параметров особи (для дальнейшего сравнения с желаемым конечным состоянием)
- получения единых для всех особей нечётких переменных – приоритетов действий, оценённых реальностью.

3.1.3.3 Пакет классов для нечёткой логики

Содержит интерфейс ***fset***, классы ***fset_ByArray*** и ***fset_LR***, наследующие интерфейсу ***fset***, а также классы ***fuzzyRule***, ***fvar***, ***fuzzyInference*** и ***fuzzySupport***.

Класс *fset* является базовым классом объектов нечётких множеств, которыми определяются нечёткие значения. Нечёткие значения отображены в программе в виде объектов классов, наследующих классу *fset*.

fset обязует классы, наследующие ему, определять следующие методы и свойства:

- свойство «идентификатор»
- метод определения принадлежности заданного чёткого значения данному нечёткому множеству
- метод дефаззификации: определения абсциссы центра масс фигуры под кривой графика функции принадлежности данного нечёткого множества

Объекты класса *fset_ByArray* являются нечёткими множествами, функция принадлежности которых задаётся по точкам. В дополнение к реализации свойства и методов, объявленных в интерфейсе *fset*, предоставляет методы для добавления/изменения/удаления точек функции принадлежности.

Объекты класса *fset_LR* являются нечёткими множествами, функция принадлежности которых задаётся ссылкой на функцию, определённую в исходном коде программы, и списком параметров, относящихся к этой функции.

Для объектов этого класса не доопределён метод дефаззификации[34], зато, соответственно, предоставлено свойство, позволяющее присвоить объекту класса *fset_LR* какую-либо функцию принадлежности.

Объекты классов *fset_ByArray* и *fset_LR* хороши каждый в своей области: для *fset_ByArray* относительно просто осуществить дефаззификацию, но приходится применять сложную интерполяцию для вычисления принадлежности; с другой стороны, для *fset_LR* дефаззификация вообще не определена, но функция принадлежности описана явно, вследствие чего вычисляется элементарным образом.

Объекты класса *fvar* представляют собой формализацию нечётких переменных. Имеют методы и свойства:

- идентификатор
- текущее чёткое значение
- текущее нечёткое значение[35]
- минимальное допустимое чёткое значение
- максимальное допустимое чёткое значение
- методы для добавления/изменения/получения по идентификатору/удаления нечётких оценок, определённых для переменной
- метод для получения значения принадлежности текущего чёткого значения переменной указанной нечёткой оценке

На этом этапе следует сделать некоторое уточнение. Над нечёткой переменной подразумеваются определёнными набор именованных нечётких значений, каждое из которых связано с некоторой лексической концепцией, имеющей смысл субъективной оценки. И одновременно с этим нечёткой переменной можно присвоить условно называемое «нечёткое значение», которое не имеет с этими определёнными ранее оценками ничего общего. Это специально искусственным

[34] Так как он крайне сложен для произвольных интегрируемых функций. Подробности в литературе по нечёткой логике, например, в [6]

[35] ни в коем случае не обязательно дефаззифицируется в текущее чёткое значение

образом определённая над чёткими значениями переменной функция, предназначенная для дефаззификации в чёткое значение переменной. Если угодно, это значение можно считать *субъективным описанием чёткого значения переменной в терминах нечётких оценок, подправленных выражениями вида* «почти», «около», «чуть более/менее чем…», etc.

Это уточнение станет понятно далее по тексту, после описания класса *fuzzyInference*.

Объекты класса *fuzzyRule* являются обёртками для описания нечётких правил «если-то». Для них имеют место понятия «левой стороны» правила – всех его предпосылок, и «правой стороны» правила – всех его последствий. Объекты класса *fuzzyRule* предоставляют возможность:

- конструировать нечёткие правила «если-то», добавляя предпосылки и последствия;
- извлекать из правила предпосылки и последствия.

Для класса *fuzzyRule* для простоты не предусмотрены методы удаления/замены предпосылок или последствий – для этого предполагается удаление текущего правила и создание нового. Объекты класса *fuzzyRule* используются объектами класса *fuzzyInference*.

Объекты класса *fuzzyInference* являются формализацией понятия нечёткого контроллера по схеме Mamdani.

Они предоставляют возможность:

- устанавливать для контроллера входные нечёткие переменные, выходные нечёткие переменные и правила вывода;
- изменять чёткие значения конкретных входных переменных без пересборки всего нечёткого контроллера;
- дефаззифицировать нечёткое значение переданной в качестве аргумента метода нечёткой переменной или дефаззифицировать набор выходных переменных по их идентификаторам;

- произвести нечёткий вывод для набора выходных переменных, заданных списком идентификаторов, и присвоить им на основе вывода нечёткие значения.

В результате нечёткого вывода выходные переменные получают нечёткие значения в виде объектов класса *fset_ByArray*, над которыми определена операция дефаззификации. Объекту класса *fuzzyInference* возможно указать количество точек, которыми будет заданы нечёткие значения выходных переменных, этот параметр условно называется «точность нечёткого контроллера».

Класс *fuzzySupport*, не создающий объектов и являющийся, по сути, аналогом программного модуля в других языках программирования, содержит:

- определения типов данных, использующихся другими классами пакета классов для нечёткой логики;
- функции преобразования используемых объектов в их текстовое описание для вывода на экран;
- предопределённые параметризованные функции, которые можно использовать как функции принадлежности нечётких множеств, представленных объектами класса *fset_LR*.

3.1.3.4 Пакет классов для генетического алгоритма

Содержит классы ***geneticProcess*** и ***geneticSettings***, интерфейс ***cell***, наследуемый классом *cell* из пакета классов для конкретизации задачи (см. раздел 3.1.3.2), и пакет классов ***vp_genetic***, который представляет собой реализацию генетического алгоритма по классической схеме, взятую из [5].

vp_genetic – самодостаточный пакет классов, при условии, что содержащийся в нём интерфейс ***geneticCell*** будет унаследован каким-либо классом, который определит требуемые интерфейсом методы и свойства объекта класса cell.

Интерфейс *geneticCell* подразумевает, что наследующий его класс определит следующие методы:

- вычисление приспособленности
- копирование

- кроссовер
- мутацию
- присвоению геному случайного значения

В данной реализации модели интерфейс *geneticCell* наследуется интерфейсом *cell*, который добавляет объявление следующих методов:

- декодирование значения
- установку/получение генома из объекта класса *cell*.

Класс *geneticProcess* является обёрткой для инкапсуляции подпрограммы, инициализирующей и выполняющей работу генетического алгоритма, заключённого в пакете классов *vp_genetic*. *geneticProcess* экспортирует одну функцию, которая выполняет следующие действия.

1. Инициализирует запись отчёта о работе с помощью объекта класса logger (см. раздел 3.1.3.5 «Пакет классов для формирования отчётов»)
2. Создаёт объект класса *geneticAlgorithm*, определённого в пакете классов vp_genetic.
3. Получает из класса *geneticSettings* значения размера популяции и вероятности мутации, и устанавливает их для созданного объекта класса *geneticAlgorithm*.
4. Вызывает функцию initReality из класса *alife*, которая создаёт и инициализирует общий для всех особей объект реальности.
5. Создаёт начальное поколение особей, присваивая им случайные значения геномов.
6. Итерирует создание следующих поколений до тех пор, пока не будет достигнуто одно из условий остановки, перечисленных в разделе 2.3.1 «Генетические алгоритмы». Численные значения для условий остановки извлекает из класса *geneticSettings*.
7. Достигнув условия остановки, выводит особь из последнего поколения, имеющую максимальное значение приспособленности, на экран и в файл отчёта.

Класс *geneticSettings*, подобно аналогичному классу *alifeSettings*, предназначен для хранения и изменения ключевых параметров работы генетического алгоритма:

- Критическое значение приспособленности генома, получив которое, генетический алгоритм прекращает работу
- Критическое значение средней приспособленности среди всех особей текущей популяции, получив которое, генетический алгоритм прекращает работу
- Количество поколений популяций особей, перебрав которые, генетический алгоритм прекращает работу
- Количество особей в популяции
- Вероятность мутации

3.1.3.5 Пакет классов для формирования отчётов

Содержит классы ***logger*** и ***errorHandler***.

Класс *logger* – самодостаточный класс, предназначенный для оформления отчётов о работе программы. Он предоставляет возможность:

- закрепить за объектом класса *logger*, постепенно получающим от программы записи о ходе своей работы, текстовый файл непосредственно в файловой системе компьютера, в который время от времени логгер будет записывать собранные записи
- вести отчёты в форматах XML (см. [12]), XHTML 1.0 Strict (см. [13]) или plain text с минимальным форматированием

Таким образом, протоколирование программой хода своей работы осуществляется созданием объекта класса *logger*, с указанием пути к файлу отчёта и типа форматирования, и затем вызовом время от времени метода для сохранения в логгере записи о текущем событии в программе. Объект класса *logger* создаётся с указанием, при каком количестве сохранённых записей о событиях переписывать их в файл, что позволяет экономить на количестве обращений к диску и главное – не держать файл открытым (а значит, заблокированным) всё время работы логгера.

Класс *errorHandler* является обёрткой для упрощения написания в коде программы обработчиков исключений, он экспортирует три функции, которые:

- принудительно создают исключительную ситуацию, с указанным сообщением об ошибке;
- продолжают уже возникшую исключительную ситуацию, передавая её более вышестоящему обработчику и приписывая к ней указанное сообщение об ошибке;
- обрабатывают исключения, выводя полный текст всех сообщений обо всех ошибках, возникших на момент начала обработки.

3.1.4 Обзор интерфейса

Интерфейс программы не предполагает её использование без знакомства с документацией или же с исходным кодом. Он принципиально разбит на две части, условно названными «верхним уровнем логики» и «нижним уровнем логики».

На нижнем уровне интерфейс позволяет задать значения склонностей, желаемое конечное состояние особи и получить оценку приспособленности для данной особи, достигнутое ею конечное состояние и отчёты о совершённых особью действиях и произошедших с нею изменениях. То есть, нижний уровень логики относится к вычислению функции приспособленности $f_{\tilde{\rho}}(\rho(\iota))$.

На верхнем уровне интерфейс позволяет задать желаемое конечное состояние, ключевые параметры генетического алгоритма и получить результат работы генетического алгоритма в виде кортежа значений склонностей особи из последнего рассмотренного поколения, на заданном количестве итераций более других приблизившейся к желаемому конечному состоянию.

3.1.4.1 Пакет классов интерфейса

Содержит описания диалоговых окон для нижнего и верхнего уровней интерфейса, а также классы элементов управления, ответственных за несколько более наглядное представление параметров и склонностей особи.

Элемент управления ***paramMeter*** получает сведения о параметрах особи из её текущего состояния: минимальное, исходное, текущее и максимальное значения, а также идентификатор параметра, и выводит их один за другим в виде вертикального списка узких длинных горизонтальных полосок, закрашенных от левого края на некоторую длину, относящуюся к длине полоски так же, как достигнутое особью значение параметра относится к максимальному значению (см. илл. из раздела 3.1.4.2 «Внешний вид интерфейса»).

Элемент управления ***polygonalMeter*** получает сведения о значениях и идентификаторах склонностей особи и изображает их в виде многоугольника, в котором каждый угол соответствует некоторой склонности, а расстояние от центра многоугольника до каждого угла тем больше, чем больше значение склонности.

3.1.4.2 Внешний вид интерфейса

Сразу после запуска приложения в главном окне программы открываются окна (рис. 3.1):

- formBasic, управляющее нижним уровнем логики
- fmViewIndividual, более-менее наглядно отображающее склонности и конечные значения параметров рассматриваемой особи
- Messages, в который будут выводиться различные сообщения от программы, в том числе сообщения об ошибках.

Из меню «Основные формы»→«Верхний уровень логики» или по нажатию F1 доступно окно fmMain, изображённое на рис. 3.2, которое управляет верхним уровнем логики. Большую часть окна fmMain занимает элемент управления genDraw, входящий в состав пакета классов vp_genetic (раздел 3.1.3.4 «Пакет классов для генетического алгоритма»). Этот элемент управления наглядно отображает ход вычислений в генетическом алгоритме. На графике по горизонтальной оси отложены поколения популяций, по вертикальной – достигнутые популяциями значения приспособленностей. Толстая чёрная линия – график значений максимальных приспособленностей среди геномов популяции, тонкая красная – график средних значений приспособленности по популяции. Особь, обладающая самым приспособленным геномом из последнего на момент остановки генетического алгоритма поколения, будет отображена в окне fmViewIndividual справа.

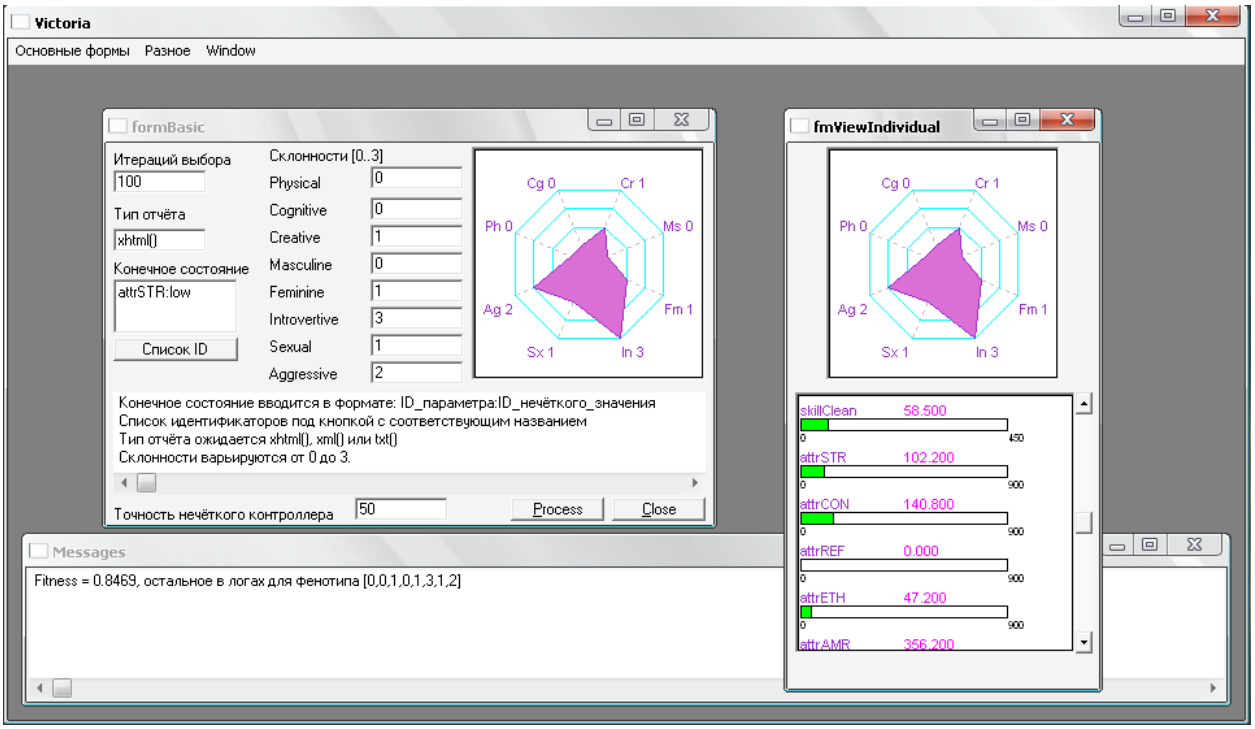


Рис. 3.1 Внешний вид программы после запуска нижнего уровня логики. Справа на форме fmViewIndividual: в верхней части элемент управления polygonalMeter, в нижней части элемент управления paramMeter

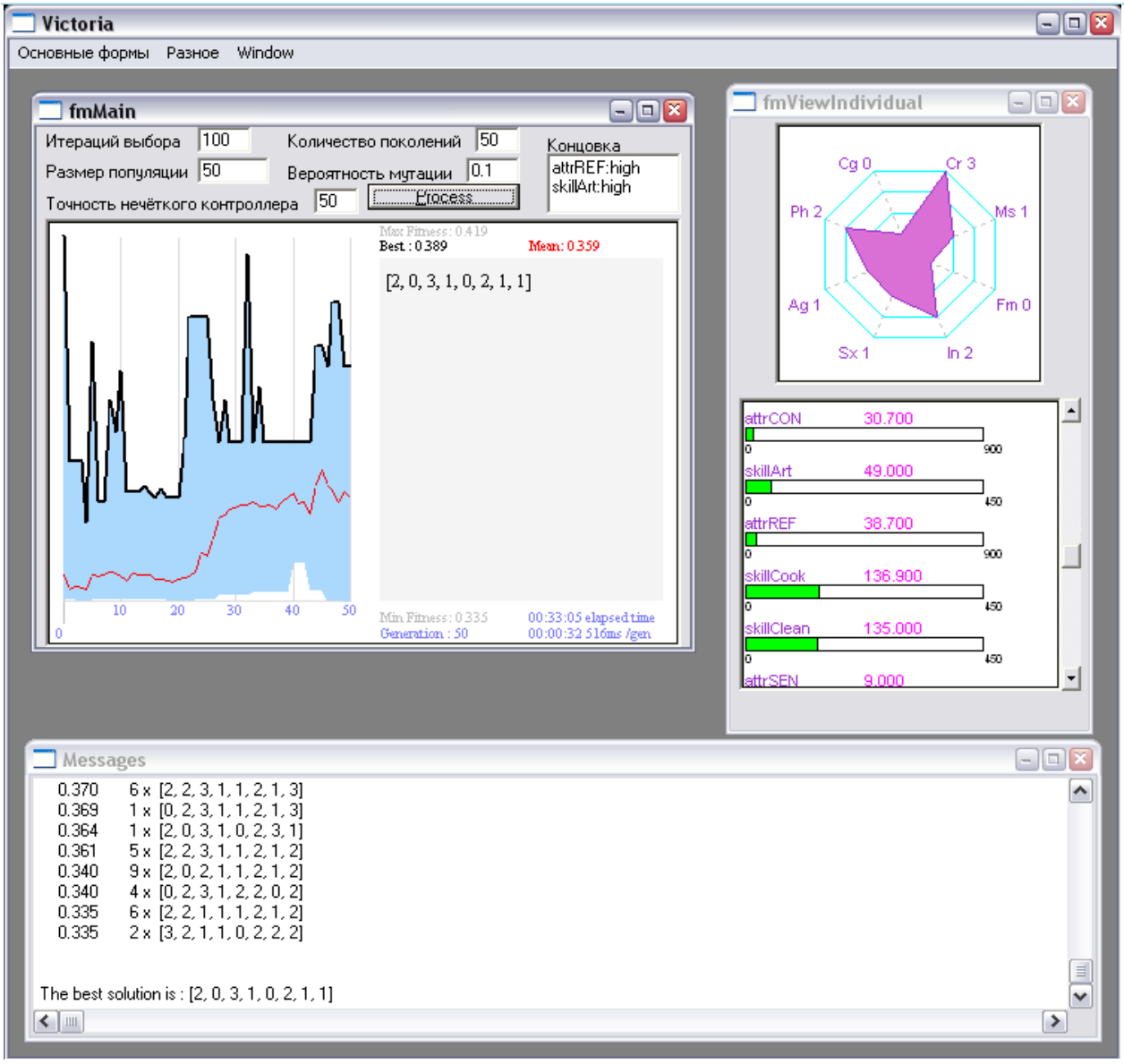


Рис. 3.2 Внешний вид программы после запуска верхнего уровня логики.

Слева на графике чёрным показано изменение лучших значений приспособленности на каждом поколении геномов, красным – изменение средней приспособленности по популяции на каждом поколении геномов.

3.2 Результаты

3.2.1 Описание тестовых условий

Верхний уровень логики запускался при указанных на рис. 3.2 параметрах:

- итераций выбора ***100***
- размер популяции ***50***
- предельное количество поколений ***50***

- вероятность мутации ***0.1***
- точность нечёткого контроллера ***50***
- желаемое конечное состояние особи: ***attrREF:high, skillArt:high***
- критическое среднее значение приспособленности по популяции (скрыто в исходном коде) ***0.75***
- критическое значение приспособленности для конкретного генома (скрыто в исходном коде) ***0.9***

Конечный график показан на рис. 3.2. Файлы отчётов находятся в электронном виде в архиве, приложенном к электронной версии данного текста. Вычисления заняли 30 минут 16.140 секунд времени процессора, по окончании вычислений программа занимала 9.7 MB оперативной памяти, при этом до начала вычислений она занимала 5.1 MB[36], что составляет 4.6 MB потребовавшегося на вычисления объема памяти.

Для вычислений использовался персональный компьютер следующей конфигурации: процессор – AMD Athlon(tm) 64 X2 Dual Core Processor 5200+ действительной частоты 2612.03 МГц, оперативная память – 2 планки OSZ DDR2-SDRAM PC2-6400 (399 МГц) - [DDR2-800] общим объёмом 2048 Мб, чипсет nVidia nForce 570 SLI, частота шины северного моста 200 МГц[37].

3.2.2 Итоги

Оптимизация целевой функции генетическим алгоритмом, как явно видно из анализа значений приспособленности перебираемых популяций особей, показала свою неэффективность. Виден небольшой тренд, характерный для генетических алгоритмов в целом, выражающийся в постепенном увеличении средней приспособленности по популяции, независимо от значений максимальной приспособленности среди геномов популяции. Однако, прирост средней приспособленности слишком незначителен, к тому же вследствие стремления к

[36] По всей видимости, на графический интерфейс.

[37] На начало 2006 года – лучшее, что можно было найти в качестве домашнего компьютера.

увеличению средней приспособленности генетический алгоритм сойдётся к популяции, состоящей из одинаковых геномов, и вариативность исчезнет.

В конкретно случае, изображённом на рис. 3.2, возможно, генетическому алгоритму не хватило поколений, потому что, когда на 50 поколении его вычисление прекратилось, состав популяции определённо указывал на возможные значительные изменения популяции в дальнейших поколениях.

В отличие от задачи оптимизации, остальные предположения модели показали свою работоспособность. Ожидаемое после введения концепции «стресса» поведение особи, заключающееся в выполнении нескольких содержательных действий, а затем действия «отдыха», обнаружилось как естественное, равно как и случающиеся время от времени занятия особью учёбой после набора значительного количества денег.

Выполнив несколько запусков нижнего уровня логики, становится окончательно ясным лежащий на поверхности недостаток схемы: определение действий так, что их выполнение постоянно повышает характеристики особи, приводит к тому, что на определённых количествах итераций становится вообще невозможным для особи достичь некоторых состояний. Например, при большом количестве итераций особь может поднять значения своих характеристик намного выше, чем требует желаемое состояние, что крайне ограничит возможности генетического алгоритма. Кроме того, из-за существования верхнего предела значений параметров особи, при данных правилах выбора, направленных на диверсификацию развития особи, определённо существует такое значение количества итераций совершения действий, после которого *любая* особь сможет максимизировать значения *всех* своих параметров, после чего задача оптимизации потеряет смысл.

Крайне заметна требовательность программной реализации к вычислительным мощностям компьютера, причём, в отличие от прочих задач оптимизации, в данной задаче вычисление оптимизируемой функции на одном кортеже аргументов занимает на порядок больше времени, чем сами операции по оптимизации.

4 Заключение

Следует заметить, что данная модель разрабатывалась в первую очередь как проверка предположений, поэтому отрицательные результаты можно считать столь же полезными, как и положительные.

Даже после весьма беглого анализа, основанного на паре запусков программы, можно сделать вывод, что, по крайней мере, в данном виде, генетический алгоритм в целом малопригоден для решения поставленной задачи.

Однако реализация алгоритма показала, что поведение, заданное в терминах *выбора* и последующего за ним *совершения действия* может быть сымитировано циклическим запуском нечёткого контроллера. При этом возможно даже имитировать такую черту сознания человека, как самооценку.

При определении предметной области для задачи требуется крайне внимательно учитывать взаимосвязь между определением последствий действий, предположениями о поведении, на которых основываются правила выбора, и заданным количеством итерации функции поведения.

Определённо, имеет смысл расширить модель, например, вводя количество итераций функции поведения как аргумент целевой функции, а не как её фиксированный параметр. Или доопределить возможность задавать желаемое конечное состояние неравенствами вида $p_k > \alpha$.

Автор позволяет себе предположить, что данная модель может быть использована при маркетинговых исследованиях и вообще в социологии. В первую очередь потому что она представляет собой имитацию естественного вида группы существ, каждое из которых ведёт себя в зависимости от своих сиюминутных личных интересов. На нижнем уровне, например, для непосредственно определения поведения заданных групп населения. На верхнем уровне, например, для построения целевых аудиторий (с учётом недостатков метода оптимизации).

Большой интерес вызывает обобщение данной модели, в котором рассматривалось бы не поведение человека, а функционирование любого

абстрактного объекта (к которому вообще применимо понятие «функционировать»). Используя модель «не по назначению» можно, вероятно, моделировать многоступенчатые химические реакции (для точного ответа требуются знания химических процессов). Требуется отдельное исследование для того, чтобы определить, в каких областях может быть применено обобщение данной модели; на момент написания данного доклада автор делает неутешительный вывод о том, что её нельзя использовать ни для чего иного, кроме как собственно моделирования человеческого поведения. Этот вывод основывается на том, что функционирование модели заключается в *итеративном выборе некоего события на основе набора численных параметров, за которым следует совершение этого события, заключающееся в изменении этих параметров*. При этом все зависимости выбора события от значений параметров должны быть определены.

Следует также обратить внимание на очевидный факт: модель не предполагает никакой адаптивности, решая задачу только для полностью определённых данных.

5 Список литературы

1. **Genes and Social Behavior** [Статья] / авт. Robinson Gene E., Russell Russell D. и Clayton David F. // Science 7. - November 2008 г.. - 5903 : Т. 322. - стр. 896-900.
2. **Основы этологии и генетики поведения** [Книга] / авт. Зорина З. А., Полетаева И. И. и Резникова Ж. И.. - Москва : Издательство Московского университета "Высшая школа", 2002. - 2-е издание.
3. **An Introduction to Genetic Algorithms** [Книга] / авт. Mitchell Melanie. - London : A Bradford Book The MIT Press, 1999. - ISBN 0-262-13316-4 (HB), 0-262-63185-7 (PB) .
4. **Социология. Анализ современного общества** [Книга] / авт. Штомпка П. – Москва : Издательство Логос, 2005. – ISBN 5-98704-024-8.
5. **Genetic Algorithm** [В Интернете] / авт. Menier Gildas // ARSANiiT. - 12 Апрель 2009 г.. - http://www.arsaniit.com/prolog-tools/menu-extension/menu-genetic-algorithm.
6. **Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence** [Book] / auth. Jyh-Shing Roger Jang, Chuen-Tsai Sun and Mizutani Eiji. - [б.м.] : Prentice-Hall, Inc, 1997. - ISBN 0-13-261066-3.
7. **The Concept of a Linguistic Variable and its Application to Approximate Reasoning - I** [Статья] / авт. Zadeh L. A. // Information Sciences. - [б.м.] : American Elsevier Publishing Company, Inc., 1975 г.. - 8. - стр. 199-249.
8. **The Concept of a Linguistic Variable and its Application to Approximate Reasoning - II** [Article] / auth. Zadeh L. A. // Information Sciences. - [б.м.] : American Elsevier Publishing Company, Inc., 1975. - 8. - pp. 301-357.
9. **The Concept of a Linguistic Variable and its Application to Approximate Reasoning - III** [Article] / auth. Zadeh L. A. // Information Sciences. - [б.м.] : American Elsevier Publishing Company, Inc., 1975. - 9. - pp. 43-80.

10. **Visual Prolog: Products, Compiler, IDE, Download Free Personal Edition** [В Интернете] / авт. Prolog Development Center // Веб-узел Visual Prolog. - 12 Апрель 2009 г.. - http://www.visual-prolog.com.
11. **Нейронные сети, генетические алгоритмы и нечеткие системы** [Книга] / авт. Рутковская Д., Пилиньский М. и Рутковский Л. / перев. Рудинский И. Д.. - Москва : Горячая линия - Телеком, 2006. - ISBN 5-93517-103-1.
12. **Extensible Markup Language (XML) 1.0 (Fifth Edition)** [В Интернете] / авт. World Wide Web Consortium // Портал World Wide Web Consortium: спецификация XML. - 12 Апрель 2009 г.. - http://www.w3.org/TR/2008/REC-xml-20081126/.
13. **XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition)** [В Интернете] / авт. World Wide Web Consortium // Портал World Wide Web Consortium: спецификация XHTML. - 12 Апрель 2009 г.. - http://www.w3.org/TR/xhtml1/.
14. **Information technology - Programming languages - Prolog - Part 1: General Core.** ISO/IEC 13211-1, – 1995.
15. **Prolog: The ISO Standard Documents** [В Интернете] / авт. J.P.E. Hodgson // Веб-узел Saint Joseph's University: стандарты ISO Prolog. - 13 Мая 2009 г.. - http://pauillac.inria.fr/~deransar/prolog/docs.html.
16. **Понятие лингвистической переменной и его применение к принятию приближенных решений.** [Книга] / авт. Заде Л.А. / перев. с англ. – М.: Мир. – 1976.