Федеральное агентство по образованию
Государственное образовательное учреждение высшего профессионального образования
Уральский государственный университет им. А.М. Горького

Математико-механический факультет
Кафедра алгебры и дискретной математики

Система поддержки алгоритмов коллективного разума

Допущен к защите

_____

"__" _____ 2010

Квалификационная работа на степень бакалавра наук
по направлению «Математика. Компьютерные науки.»
студента гр. КН – 401
Кононучука Дмитрий Олеговича

Научный руководитель
Окуловский Юрий Сергеевич
должность, степень

Екатеринбург
2010

Содержание

Перечень принятых в работе сокращений 3

Введение 4

Глава 1. Теоретические основы 5

1.1. Понятия искусственного и вычислительного интеллекта 5

1.2. Алгоритмы эволюционного моделирования 6

1.3. Алгоритмы коллективного разума 12

1.4. Многоагентные алгоритмы 13

1.5. Общие свойства АКР 16

Глава 2. Практическая реализация 18

2.1. Спецификация поддерживаемых АКР 18

2.2. Пояснения и уточнения свойств 22

2.3. Требования к реализации 25

2.4. Описание интерфейсов координат 28

Заключение 30

Литература 31

Перечень принятых в работе сокращений

ACO	Ant Colony Optimisation
GSA	Gravitational Search Algorithm
LEM	Learnable Evolution Model
PSO	Particle Swarm Optimization
АКР	Алгоритмы Коллективного Разума
АЭМ	Алгоритмы Эволюционного Моделирования
ВИ	Вычислительный Интеллект
ГА	Генетические Алгоритмы
ИИ	Искусственный Интеллект
МАА	Многоагентные Алгоритмы
ОО	Объектно Ориентированный
ПО	Программное Обеспечение

Введение

Глава 1

Теоретические основы

1.1. Понятия искусственного и вычислительного интеллекта

Под искусственным интеллектом понимается довольно обширный класс инструментов и технологий, границы которого во многом определяются историческими либо практическими причинами, а не общепринятыми формальными критериями, которых на данный момент не существует.

Наиболее распространенным подходом к определению интеллектуальности, а, следовательно, и ИИ , является агент-ориентированный подход, изложенный в одной из основополагающих работ в области — [1]. Ключевое понятие для этого подхода — агент. Это некоторая сущность, которая способна воспринимать окружающую среду посредством датчиков и влиять на нее посредством исполнительных механизмов. Поведение агента считается интеллектуальным, если оно рационально в т.ч. и при меняющихся условиях среды. Эта модель требует наличия свойства адаптивности у алгоритмов ИИ.

Тем не менее, существуют другие подходы, например логический [2] или основанный на поиске [3], ярким примером которого является алгоритм A-star [4], который в настоящее время не относится к интеллектуальным.

В своей работе я исследовал лишь методы, относящиеся к области вычислительного интеллекта [5], интеллектуальность которых общепризнанна. Однако, также как и для интеллекта искусственного, для вычислительного не было построено четкого определения. Обширный обзор разработанных определений можно найти в [6]. В этой-же работе автор приводит собственное, пожалуй, лучшее из представленных в литературе, хотя и излишне общее определение ВИ: «Вычислительный интеллект — это область компьютерных

наук, изучающая решение задач, для которых не существует эффективного алгоритмического решения».

Во многих работах, например [7], ВИ определяется, как совокупность различных технологий. Подобный остенсивный [8] способ определения понятия обладает очевидными недостатками, поскольку не указывает на общие свойства этих технологий и не является расширяемым. В частности, в книге [5] авторы относят к ВИ 4 типа алгоритмов, а 2 года спустя в [7] — уже 7. Работы, использующие для определения ВИ именно такой — остенсивный — подход, обычно содержат довольно подробную классификацию существующих методов.

Согласно этим классификациям алгоритмы ВИ разбиваются на три подкласса: нейронные сети [9], нечеткое управление [10] и эволюционное моделирование [11].

1.2. Алгоритмы эволюционного моделирования

Перейдем к более подробному рассмотрению третьего класса алгоритмов — алгоритмов эволюционного моделирования. Эти алгоритмы предназначены для решения задач комбинаторной оптимизации.

Исторически, первыми представителями этого класса были генетические алгоритмы, предложенные Лоуренсом Фогелем в работе [12]. В их основу положен принцип моделирования генетической эволюции в смысле синтетической теории эволюции [13]. В самом общем виде генетический алгоритм формулируется следующим образом:

1. Алгоритм оперирует особями — некоторым представлением решений задачи. Каждая особь однозначно определяет допустимое решение. Для каждой особи определена ее функция приспособленности — характеристика оптимальности решения ею определяемого.

2. Алгоритм поддерживает пул особей, называемый популяцией, который характеризует его текущее состояние.

3. При инициализации алгоритма некоторым образом создается начальная популяция.

4. На каждом шаге алгоритма производятся три процесса:

    а. Мутация: некоторым случайным образом выбираются несколько особей и для каждой из них в популяцию добавляется особь (или несколько особей), которая является ее некоторой случайной модификацией.

    б. Скрещивание: некоторым случайным (либо нет) образом выбираются пары особей.

    в. Для каждой пары некоторым случайным способом создаются и затем добавляются в популяцию некоторое количество особей являющихся производными от данной пары.

    г. Отбор: из популяции удаляются особи имеющие плохую функцию приспособления либо слишком продолжительное время жизни (возможно сочетание критериев).

5. Шаг алгоритма повторяется до тех пор пока не выполняться определенные условия. Например, пока в популяции не перестанут происходить существенные изменения.

6. Результат работы алгоритма — особь с лучшей функцией приспособленности.

Генетические алгоритмы являются крайне общими методами, обладающими огромным числом параметров, существенно влияющими на их пове-

дение. Для ГА известны способы доказательства корректности и сходимости алгоритмов [14], а также определения их эффективности [15].

Другим АЭМ, зачастую противопоставляемым генетическим, является алгоритм Learnable Evolution Model [16]. Он также работает с представлением решений в виде особей с определенной функцией приспособленности и представлением текущего состояния в виде популяции. Однако в его основе лежит другая теория эволюции — теория направленной эволюции (номогенез) [17]. В самом общем виде алгоритм формулируется так:

1. Некоторым способом создается начальная популяция.
2. На каждом шаге алгоритма выполняются следующие действия:
    а. Из популяции выделяются две группы особей: «хорошие» и «плохие» — особи соответственно с высоким и низким значением функции приспособленности.
    б. С помощью методов машинного обучения строятся гипотезы — описания отличительных черт «хороших» особей, которые не наблюдаются у «плохих». Возможно также построение обратных гипотез — черт присутствующих у «плохих», но не у «хороших» особей.
    в. Генерируются новые особи, которые удовлетворяют гипотезам и не удовлетворяют обратным гипотезам.
    г. Эти особи тем или иным способом добавляются в популяцию. Например заменяя всех особей не являющихся «хорошими».
3. Шаг алгоритма повторяется до тех пор пока не выполняются определенные условия.
4. Результат работы алгоритма — особь с лучшей функцией приспособленности.

Алгоритм Particle Swarm Optimization [18] может служить примером абсолютно другого подхода к АЭМ. В нем, как и в предыдущих алгоритмах, работа ведется с агентами (в английской литературе: particle — частица), популяцией (в англ. литературе: particle swarm — рой частиц) и метрикой неоптимальности решения (f). Однако, требуется, чтобы агент был представлен вектором в n-мерном пространстве, причем каждый вектор этого пространства является допустимым решением и, следовательно, потенциальным агентом.

1. Для каждого агента $i \in [1, S]$ (где S — количество агентов) в момент времени $t = T$ определены векторы: $x_i(T)$ — его координата; $v_i(T)$ — его скорость; $x_i^\sharp(T)$ — координата наилучшей достигнутой позиции, т.е. $x_i^\sharp(T) = \arg\min\limits_{t \leqslant T} f(x_i(t))$.
2. Для всей популяции в момент времени $t = T$ определен вектор наилучшего достигнутого решения $x^*(T) = \arg\min\limits_{i \in [1,S]} f(x_i^*)$.
3. Некоторым способом задается начальная популяция.
4. На каждом шаге алгоритма для каждой особи i выполняются следующие действия:
   а. Вычисляется новый вектор скорости:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1 (x_{ij}^\sharp(t) - x_{ij}(t)) + c_2 r_2 (x_j^*(t) - x_{ij}(t))$$

   где w, c_1, c_2 — параметры алгоритма, а r_1, r_2 — случайные, равномерно распределенные на [0; 1] числа генерируемые независимо для каждого агента.

   б. Вычисляется новая координата агента: $x_i(t+1) = x_i(t) + v_i(t+1)$ и его метрика $f(x_i(t+1))$.

   в. Определяется $x_i^\sharp(t+1)$ и если необходимо обновляется значение $x^*(t+1)$.

5. Шаг алгоритма повторяется до тех пор пока не выполняются определенные условия.

6. Результат работы алгоритма — особь с координатой $x^*(t_{\max})$.

На основании вышеперечисленных алгоритмов можно выделить общие черты, присущие всем АЭМ:

1. Состояние алгоритма представлено некоторым множеством (популяцией) некоторых объектов (особь). Каждая особь определяет решение, для которых вводится мера оптимальности. Эта мера может быть перенесена на особи (что определяет функцию приспособленности особи).

2. Алгоритм итеративный. Цель каждой итерации — увеличить максимальное значение функции приспособленности для особей из популяции. По сути — провести эволюцию популяции.

3. Начальное состояние популяции дается алгоритму извне — обычно генерируется случайно.

4. На каждой итерации алгоритм многократно (например для каждой особи, либо фиксированное число раз) выполняет некоторые действия. Причем действия эти независимые и распараллеливаемые. В некоторых алгоритмах также выделяются стадии пред- и пост-процессинга.

5. Алгоритм стохастический.

Особо отмечу, что вышеприведенные алгоритмы формируют более слабые ограничения для пунктов (1) и (4). Основываясь лишь на них, мы можем сформулировать эти пункты как: «Состояние алгоритма представлено множеством (популяцией) его допустимых решений (особь)» и «На каждой итерации алгоритм выполняет некоторые действия для каждой особи». Однако существуют

примеры АЭМ, которые не вписываются в эти, более узкие, рамки. Например, расширение пункта (1) необходимо для включения в класс АЭМ алгоритмов муравейника [19], а пункта (4) — для включения алгоритма гармонического поиска [20].

Перечисленные свойства не являются обязательными для АЭМ, поскольку этот класс сформировался не вокруг общих свойств, а вокруг общего происхождения и круга решаемых задач, однако, они наблюдаются у всех общеизвестных алгоритмов.

Стоит заметить, что эти свойства являются весьма существенными с точки зрения программных реализаций алгоритмов. Они определяют задачи, которые должны быть решены в каждой из них. Это, в первую очередь, задачи создания удобной программной абстракции, выполнения сервисных операций с популяцией, распараллеливания и распределения по данным вычислений на каждой итерации. Их решение довольно трудоемко, что приводит к мысли о создании специализированных библиотек поддержки. Однако, ввиду чрезмерной общности данных свойств и их неестественности, такие библиотеки будут недостаточно удобны в использовании, хотя их примеры существуют: EO Evolutionary Computation Frameworki [21], ECF [22] и другие.

Неестественность общих свойств АЭМ во многом объясняется тем, что данный класс состоит из двух подклассов, представители которых существенно различны. Это подклассы алгоритмов эволюционных вычислений [23] и коллективного разума [24]. У каждого из них имеется гораздо больше общих свойств, и эти свойства гораздо естественнее, благодаря чему создание библиотеки поддержки становится более целесообразным.

В своей работе я сфокусировался на разработке подобной библиотеки для класса алгоритмов коллективного разума.

1.3. Алгоритмы коллективного разума

Родоначальником этого класса являются уже упомянутый выше алгоритм муравейника (англ. Ant Colony Optimisation) [19]. Это алгоритм является более общим, нежели другие приведенные мною ранее алгоритмы, в т.ч. алгоритм PSO, который также как и алгоритм муравейника относится к АКР. Фактически, он лишь определяет общие принципы построения аналогичных алгоритмов. Эти принципы были скопированы его разработчиками с принципов поведения колонии муравьев.

Алгоритм работает на графах. В его основе лежит использование множества независимых агентов (здесь и далее «агент» — синоним «особи» в терминологии принятой для АКР) — «муравьев», каждому из которых соответствует решение, итеративно строимое им (муравей перемещается по вершинам графа; его маршрут — решение). Ключевая особенность алгоритма муравейника в использовании «феромонов» ($\tau_{i,j}(t)$) — разновидности памяти, позволяющей ассоциировать с ребрами графа значения, указывающие на историю его использования муравьями. Кроме того, алгоритм использует эвристику $\eta_{i,j}(t)$.

На каждой итерации алгоритма муравейника для каждого муравья ant_k, $k \in \{1 \dots n\}$ расположенного в вершине i производятся следующие действия:

1. Вычисляется вероятность перехода муравья в каждую из допустимых вершин. Общая формула:

$$\forall j \in N : \; p_{i,j}^k(t) = \frac{\left(\tau_{i,j}(t)\right)^\alpha \left(\eta_{i,j}(t)\right)^\beta}{\sum\limits_{l \in N} \left(\tau_{i,l}(t)\right)^\alpha \left(\eta_{i,l}(t)\right)^\beta}$$

2. Выполняется случайный переход муравья по ребру $(i,\ j)$.

3. Фиксированным для алгоритма способом определяется функция $\Delta\tau_{i,j}^k(t)$ и $\forall l \neq j$ полагаем $\Delta\tau_{i,l}^k(t) = 0$

Кроме того, на каждой итерации, для всех ребер осуществляется пересчет соответствующих феромонов:

$$\tau_{i,j}(t+1) = \rho\tau_{i,j}(t) + \sum_{k=1}^{n} \Delta\tau_{i,j}^{k}(t), \text{ где } 0 \leqslant \rho < 1$$

Алгоритм первоначально был сформулирован для решения задачи коммивояжёра [25], однако благодаря своей обобщенности и наличию параметров был адаптирован для решения многих других задач, примеры которых можно найти в [26].

1.4. Многоагентные алгоритмы

Алгоритм муравейника является не только характерным примером АКР, но и другого, чрезвычайно похожего на первый взгляд, класса алгоритмов — многоагентных алгоритмов. Границу между этими двумя классами не проводят в большей части литературы.

Дело в том, что МАА и АКР оба базируются на агентах, однако трактуют это понятие по разному. Как уже было указано выше, для АКР агент — то же самое, что и особь для АЭМ, т.е. сущность, которой он оперирует и которая представляет решение. В то время как в МАА под агентом подразумевается агент в терминах агент-ориентированного подхода к ИИ [1], т.е. независимая, решающая задачу сущность, воспринимающая окружающую среду и взаимодействующая с ней.

Поскольку сам МАА также является агентом в этих терминах, то естественно, что необходимо вводить специальные ограничения, делающие задачи решаемые агентами разных уровней (МАА в целом $\longleftrightarrow$ агент в МАА) существенно различными. Типичным ограничением является возможность работы лишь в подмножестве пространства исходной задачи.

Из определений очевидно, что МАА являются подмножеством АКР: с

точностью до формулировок МАА являются АКР, в котором каждый агент решает задачу независимо от других (он может либо игнорировать другие агенты, либо взаимодействовать с ними лишь опосредованно — через сообщения или разделяемую память). Но это означает, что многоагентные системы, которые по сути являются МАА, естественно сводятся к терминам АКР. Этот факт с практической точки зрения представляет большой интерес.

К многоагентным системам относятся, в частности, эмуляции жизни, эволюции, социума и т.п., некоторые игры и многое другое. Все эти системы могут интерпретироваться как алгоритмы АКР и реализовываться поверх библиотеки их поддержки, что существенно расширяет область применимости такой библиотеки.

Примером АКР, которые не является МАА, может служить алгоритм гравитационного поиска (англ. Gravitational Search Algorithm) [27]. Агентами в этом алгоритме обладают координатой $X_i(t)$ в пространстве решений, скоростью перемещения в этом пространстве — $V_i(t)$ и массой (авторы предлагают использовать три вида масс: инерционную $M_{ii}(t)$, активную $M_{ai}(t)$ и пассивную $M_{pi}(t)$ гравитационные, однако способы обновления и вычисления их приводят только для ситуации, когда $M_i(t) = M_{ai}(t) = M_{pi}(t) = M_{ii}(t)$). Для агента определена его оценка — функция качества решения им определенного $f_i(t) = f(X_i(t))$.

Тогда алгоритм каждой итерации следующий:

1. Вычислить силы взаимодействия для каждой пары объектов. Для d координаты они вычисляются по формуле:

$$F_{ij}(t) = G(t)\frac{M_{pi}(t)M_{aj}(t)}{R_{ij}(t)}\left(x_i^d(t) - x_j^d(t)\right)$$

где $R_{ij}(t) = \left\|X_i(t),\, X_j(t)\right\|_2$ — евклидово расстояние.

Это правило является видоизмененным законом всемирного тяготения. Отличия заключаются во-первых в том, что множитель $R_{ij}(t)$ имеет сте-

пень -1, а не -3, а во-вторых, в том что гравитационная постоянная G введена как функция от времени. Первый изменение было выведено авторами экспериментально — так алгоритм быстрее сходился. Второе является следствием поисковой природы алгоритма — для нахождения точных решений необходимо повышать «аккуратность» алгоритма со временем, для чего необходимо уменьшить силы взаимодействия.

2. С помощью независимых случайных величин $rnd_i \in [0, 1]$, вычисляются $F_i(t) = \sum\limits_{j=1,\, j\neq i}^{N} rnd_j F_{ij}(t)$ и $a_i(t) = F_i(t)/M_{ii}(t)$. Это, соответственно, правило сложения сил, в которое добавили элемент случайности, и второй закон Ньютона.

3. Производится перемещение объекта:

$$V_i(t+1) = rand_i V_i(t) + a_i(t)$$
$$X_i(t+1) = X_i(t) + V_i(t+1)$$

где $rand_i$ — независимые случайные величины равномерно распределенные на $[0, 1]$. Затем пересчитывается его оценка.

4. Обновляются массы объектов:

$$w(t) = \min_{i\in\{1\ldots N\}} f_i(t)$$
$$b(t) = \max_{i\in\{1\ldots N\}} f_i(t)$$
$$m_i(t) = \frac{f_i(t) - w(t)}{b(t) - w(t)}$$
$$M_i(t+1) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}$$

На примере трех вышеприведенных алгоритмов можно рассмотреть еще одну важную характеристику АКР — тип пространства агентов. В части алгоритмов, в т.ч. в PSO и в GSA, используются агенты располагаемые (обладающие координатой) в пространстве допустимых решений, в то время как в

остальных алгоритмах (из приведенных — в ACO) — в пространстве задачи. Агенты первого типа являются агентами, оптимизирующими решения. Они аналогичны особям, используемым в ГА, LEM и других эволюционных алгоритмах [28]. Агенты второго типа являются агентами, строящими решение. Они могут быть использованы только в АКР и, более того, только в МАА. Большинство МАА (в т.ч. многоагентные системы) основаны на них. Однако, существуют исключения: упомянутый выше PSO, Stochastic diffusion search [29] и другие.

1.5. Общие свойства АКР

Обобщив вышесказанное, можно представить класс АКР в следующем виде: (картинка АКР — неМАА + МАА — МАА со строящими агентами)

Сформулируем общие свойства АКР:

1. Алгоритм работает в некотором пространстве с координатами (карта).
2. Работа алгоритма сводится к оперированию множеством (рой) сущностей (агентами) в этом пространстве.
3. Состояние алгоритма определяется состоянием роя. Реже также используется память ассоциированная с точками пространства и/или небольшое количество глобальных переменных.
4. Алгоритм итеративный. Одну итерацию будем называть ходом.
5. В течении хода алгоритм производит независимые действия над каждым агентом роя (обработка). Эти действия не образуют сторонних эффектов друг на друга: т.е. результат обработки для каждого агента не зависит от того, были ли обработаны другие агенты, в каком порядке они были

обработаны, как они изменились и изменили хранимые данные в ходе обработки.

6. Условие остановки алгоритмом не специфицируется.

7. Начальное состояние данных и роя дается алгоритму извне. Координаты агентов в начальном рое обычно случайны (реже при их генерации используется эвристика).

8. Алгоритм стохастический. Генерация случайных чисел (обычно многократная) необходима при обработке каждого агента.

МАА в дополнение к вышеперечисленным также обладает следующими свойствами:

1. Его агенты независимы. Результат обработки любого агента не зависит от состояния других агентов в любой момент времени. Взаимодействие агентов может быть лишь опосредованным, например, через совместный доступ к памяти.

2. Агенты могут действовать лишь в своей ограниченной окрестности.

АКР с оптимизирующими агентами обладает лишь одним специфичным свойством: карта является пространством допустимых решений, таким образом каждому агенту соответствует решение. Меру оптимальности (или отношение «быть оптимальнее») определенную на решениях можно распространить на агентов и затем использовать для их сравнения.

В АКР со строящими агентами карта является пространством задачи. Таким образом каждый агент не является полноценным решением, а значит невозможно без введения эвристики производить их сравнение.

Глава 2

Практическая реализация

2.1. Спецификация поддерживаемых АКР

Поскольку, как было сказано в разделе (1.5), существует большое количество подклассов и разновидностей АКР, то первой задачей при разработке системы их поддержки становится задача фиксации требований к ней — разработка спецификации поддерживаемого типа АКР.

Необходима была такая формулировка АКР, которая обладает следующими свойствами: наибольшей общностью, хорошей приспособленностью для использования большинства известных алгоритмов (в т.ч. наиболее популярных — в первую очередь, алгоритма муравейника [19]), высокой выразительностью для прикладных задач, близостью к формулировкам многоагентных систем. Кроме того, эта формулировка должна быть легко выразима в терминах ОО [**?**] архитектуры и удобна для программной реализации.

В связи с последними двумя требованиями, а также в связи с естественным для всех спецификаций ПО требованием максимальной полноты и формальности, конечная формулировка получилась иерархичной и объемной. Приведем её последовательно, начиная от общих понятий.

АКР — это изменяющаяся во времени система, обладающая следующими свойствами:

1. Система функционирует в некотором адресуемом пространстве (карте) с координатами.

2. Каждой точке этого пространства (ячейке) соответствует некоторый набор данных (фон ячейки).

3. Кроме того, в ячейках могут располагаться объекты карты (агенты). Количество агентов в ячейке не ограниченно. Каждый агент располагается в некоторой ячейке, и притом только в одной.

4. Система изменяется итеративно. Все итерации равнозначны. Одна итерация называется ходом.

На карту и координаты накладываются следующие требования:

5. Карта является, фактически, сюръективной не всюду определенной функцией отображающей пространство координат в множество ячеек.

$$Map: Coord \to Cell$$

6. Будем называть координаты, для которых функция карты определена (т.е. те координаты, которым соответствует ячейка карты), достижимыми.

$$Reach \subseteq Coord: \forall c \in Reach\ \exists Map(c)$$

7. Для каждой пары координат определен куб ими задаваемый — некоторая последовательность координат, про которые мы можем неформально сказать, что «они лежат в n-мерном кубе, верхний левый и нижний правый угол которого заданы исходной парой координат». Причем эта пара координат входит в свой собственный куб.

$$Cube: Coord \times Coord \to 2^{Coord};\ \forall a, b \in Coord: a, b \in Cube(a, b)$$

8. Для каждой пары координат верно, что если они достижимы, то и все координаты входящие в куб ими заданный также достижимы.

$$\forall a, b \in Reach: \forall c \in Cube(a, b)\ c \in Reach$$

9. Для каждой достижимой координаты и неотрицательного числа определено непустое множество координат, называемых окрестностью данной координаты с данным радиусом. Причем каждая достижимая координата лежит в любой своей окрестности.

$$Suburb: Reach \times \mathbb{R}_+ \to 2^{Reach};\ \forall c \in Reach, \forall r \in \mathbb{R}_+ : c \in Suburb(c, r)$$

10. В дальнейшем мы будем рассматривать только достижимые координаты, если иное не отмечено особо.

Фон ячейки можно рассматривать либо как неотъемлемое свойство самих ячеек, либо как некоторое множество (фон карты) биективно отображаемое на множество ячеек. Определение очевидно эквивалентны. При описании формулировки будет удобнее пользоваться первым, однако при описании модели программной реализации — вторым. Данные хранимые в фоне ячеек в общем определении АКР не специфицируются и зависят от конкретного алгоритма.

Агент — это единственная активная сущность в системе АКР. На каждом ходе он выполняет некоторый набор действий, изменяющих его и всю систему в целом. Эти действия определяются заложенным в агента алгоритмом на основе его состояния и состояния всей системы. Для агента могут быть постулированы следующие свойства:

11. Агент расположен в некоторой ячейке, т.е. для него определена координата.

12. Агент на каждом ходе обладает неизменной характеристикой — радиусом его области видимости, являющейся неотрицательным числом.

13. Агент в течении хода имеет доступ ко всем ячейкам, координаты которых входят в окрестность его координаты с радиусом, равным радиусу области его видимости. Очевидно, что ячейка, в которой расположен агент, входит в эту окрестность.

14. Под доступом к ячейке подразумевается:

    а. чтение ее координаты,

    б. доступ к данным ее фона на чтение и запись,

    в. перечисление всех агентов в ней расположенных,

    г. отправка сообщения любому из этих агентов,

    д. добавление в нее нового агента,

    е. перемещение самого себя в эту ячейку.

15. Агент может принимать сообщения (то есть иметь доступ на чтение к их содержимому), которые отправили ему другие агенты. Сообщения являются однонаправленными, не несущими информации об отправителе, локальными во времени (сообщения отправленные в течении хода будут получены в начале следующего, а затем более не будут доступны). Содержимое сообщений зависит от конкретных алгоритмов и не специфицируется.

16. Агент может отправлять широковещательные сообщения. Подобные сообщения будут получены всеми агентами в начале следующего хода, включая тех агентов, которые будут созданы в течении текущего хода, а также включая отправителя.

17. Агент может удалить себя с карты: перестать быть расположенным в некоторой ее ячейке и перестать получать широковещательные сообщения. Такой агент более не может изменять состояние системы и самого себя.

18. Действия агента на каждом ходе не зависят от действия других агентов в том-же ходе: все сообщения, изменения фона карты, добавление/перемещение/удаление агентов — результаты всех действий других агентов

не будут доступны ему. Фактически, если мы определим текущее состояние системы, как функцию от времени (роль времени играет номер текущего хода), тогда каждый агент будет по текущему значению этой функции строить ее преобразование, а значение функции в следующий момент времени будет результатом применения композиции всех таких преобразований.

19. Более того, действия всех агентов (в терминах преобразования системы) инвариантны относительно порядка приминения. То есть система будет преведена в одно и тоже состояние в независимости от того, в каком порядке будут применены эти преобразования. Причем очевидно, что не обязательно груперовать все действия одного агента в течении хода в одно преобразование, а можно представлять каждое действие отдельным.

Мы закончили формулировку выбранной мной вариации АКР. Очевидно, что работа любого из алгоритмов обладающих свойствами, описанными в разделе (1.5), может быть эмулированна им.

Система поддержки АКР должна брать на себя все сервистные функции алгоритма с вышеописанными свойствами, оставляя разработчику конечного алгоритма лишь работу по реализации логики агентов, спецификации типов данных сообщений и фона карты, а также описнаия координат карты. Причем, очевидно, что для такого разработчика наибольший интерес будет представлять создание агента, поэтому все остальные задачи нужно максимально упростить.

2.2. Пояснения и уточнения свойств

Перед тем как переходить к следующим вопросам, уточним причины использования подобных формулировок некоторых свойств.

Свойство (9) необходимо для поддержки МАА системой. Стоит отметить особо, что система ориентирована в первую очередь именно на них, поскольку МАА алгоритмов изветсно больше чем просто АКР, а также поскольку все неалгоритмические приложения системы используют многоагентный подход.

Свойства (7) и (8) черезвычайно удобны для введения возможности разбиения карты на некоторое колличество непересекающихся частей, которая необходима для оптимальной организации распределения вычислений в системе поддержки АКР. Поскольку эти свойства дровольно естественны и легко реализуемы, то их было решено ввести в общую модель.

Свойства (18) и (19) агента являются одними из наиболее важных. Фактически они представляют собой другую формулировку свойства (5) АКР (стр. 16) и являются существенными с точки зрения реализации, поскольку создают идеальные условия для распараллеливания и распределения вычислений действий агентов и их приминения на каждом ходе.

Из-за наличия этих свойств становится невозможным введение операций удаления или перемщения одного агента другим (эти операции были бы довольно полезны в некоторых многоагентных системах). А также невозможно непосредственное взаимодействие агентов, то есть, если возвращаться к терминам програмной модели, вызов методов одного агента другим, что и привело к необходимости введения понятия сообщений и операций их отправки и получения.

Теоретически возможно организовать работу АКР и без обмена сообщениями. В этом случае, взаимодействие агентов будет основано на передаче информации через фон карты. То есть для того, чтобы агент *A* передал некоторую информацию агенту *B* необходимо, чтобы агент *A* записал ее в фон клетки, в которой находится агент *B*. Однако, очевидно, что такой подход содержит «подводные камни»: если агент *B* переместился в другую ячейку на том же ходе, когда агент *A* передал ему информацию, то он ее не получит; если агенты

A и *C* передали информацию агенту *B* таким способом одновременно, то одна информация перекроет другую и возникнет ситуация «потерянной записи».

Вышеописанная техника применялась в прототипе существующей системы, в котором не требовалось выполнение свойства (19). Она показала себя как черезвычайно неудобная для реализации и использования. В том-же прототипе вместо широковещательных сообщений (свойство (16)) использовались глобальные переменные, которые не смотря на то, что нарушают свойство (19), являются более удобными для использования.

Свойство (15) требует, чтобы отправленные в текущем ходе сообщения были приняты лишь в следующем. Это требование является следствием свойства (18), однако оно крайне неудобно с практической точки зрения. Как показала практика работы с прототипом системы и анализ возможных сценариев ее приминения, чаще всего сообщения отправленные в текущем ходе содержат данные актуальные именно в нем.

Поскольку, чтобы на момент получения сообщения было возможно восстановить контекст его отправки и правильно его интерпретировать, приходится сохранять довольно много информации в состоянии агента, то почти постоянно при разработке агента тратиться множество времени на эти вспомогательные операции, которые, теоретически, должна целиком брать на себя система поддержки. В худших, но нередких, случаях приходится даже разделять логически цельный ход на два хода системы: в первом производится отправка сообщений, а во втором — некоторые действия зависящие от сообщений. Очевидно, что необходимо обоих ситуаций избегать.

В связи с вышеуказанными проблемами в системе использовался несколько видоизменнный ход, разбитый на две стадии: начальная и конечная. В начальной стадии хода агент может посылать сообщения и читать сообщения, принятые в предыдущем ходе. В конечной стадии агент не может отсылать сообщения, но может принимать сообщения, посланные на начальной стадии.

В остальном стадии равноправны. Подобное разделение позволяет агентам сделать последовательные ходы максимально независимыми друг от друга и существенно сократить количество информации хранимой в их состоянии (в лучшем случае — избавиться от хранимого состояния полностью). Агенты, для которых «актуальность» сообщений не критична, могут реализовывать только начальную стадию хода.

Для улучшения удобства работы с широковещательными сообщениями были введены несколько средств: именнованые сообщения, постоянные сообщения и обработчики сообщений.

Именнование сообщений означает, что каждому широковещательному сообщению будет присвоенно имя (по умолчанию — пустое), и агенты смогут фильтровать сообщения по этому имени.

Постоянные сообщения — это широковещательные сообщения, которые будут доступны для чтения во всех последующих ходах, а не только в одном.

Обработчики сообщений — это некоторые функции, которые могут модифицировать список широковещательных сообщений с определенным именем, например оставлять в нем только сообщение с самыми большими значениями или заменять все сообщения одним, содержащим сумму их данных.

Сочетание двух последних техник позволяет эмулировать глобальные переменные не нарушающие свойства (19).

2.3. Требования к реализации

Поскольку планируется будующее использование системы поддержки АКР в работе лаборатории РВИиМАП, в том числе в рамках мроводимого моим научным руководителем спецкурса Нейронные Сети, то к ее реализации были предьявленны некоторые дополнительные требования. Во-первых, система должна быть написанна на языке C# и работать на платформе .NET,

поскольку остальные разработки лабборатории выполняются с помощью этих технологий. Во-вторых, было необходимо достичь максимальной простоты решения задач, следовательно, система поддержки должна иметь средства облегчающие решения типовых задач «из коробки». В идеале небольшие задачи-примеры должны занимать всего несколько строк, причем легко-читаемых и понятных. В-третьих, система должна иметь документацию, а ее код должен быть прокомментирован. В-четвертых, система должна быть легко расширяема и модифицируема и, при этом, показывать неплохую производительность.

Систему было решено писать на C# 4.0 под фреймворк .NET 4.0 [**?**]. Выбор столь новых инструментов обусловлен тем, что в фреймворке 4.0 существует большое количество механизмов для удобной работы с рсапараллеливанием [**?**], а также с асинхронными вычислениями [**?**], которые неизбежно возникнут при реализации распределения вычислений. Также .NET 4.0 содержит набор библиотек WCF [**?**], с помощью которых можно легко организовать обмен сообщениями между вычислительными узлами, необходимый для распределения вычислений (подобное решение является временным и будет использовтся лишь на начальном этапе).

Также среди инструментов стоит отметить особо библиотеку Code Contracts [**?**] для .NET 4.0, реализующую идеалогию контрактного программирования [**?**].

При реализации необходимо учитывать следующие моменты:

1. Размер карты может быть очень большим. Поэтому хранение каких-либо сервистных структур данных для каждой ячейки невозможно.

2. Имеет смысл хранить сервистные структуры данные лишь для ячеек, в которых расположен хотябы один агент.

3. Организация данных фона карты существенно зависит от специфики

этих данных. Поэтому фон карты воспринимается как отдельный объект, используемый для получения данных каждой конкретной ячейки.

4. При перемещении агентов из одних ячеек в другие возникает необходимость удалять ячейки, в которых нет больше агентов, и добавлять ячейки, в которые агенты переместились. Поскольку вышеописанные операции происходят многократно в течении каждого хода, а операции с выделением/освобождением памяти в .NET довольно медленные, то необходимо минимизировать их количество, что возможно сделать с помощью реализации пула сервистных объектов ячеек [**?**].

5. В ходе работы системы будет черезвычайно активно вестись работа с координатами. Поэтому координаты необходимо реализовать как структуры (т.к. операции с ними несколько быстрее) реализующие некоторый интерфейс. Кроме того, использовать координаты надо не как этот интерфейс, поскольку подобное использование приведет к боксингу структур в объекты с существенной потерей производительности, а как генрик-тип реализующий интерфейс.

6. Вызов виртуального метода занимает больше времени, чем вызов обычного. Таким образом надо максимально отказатся от их использования. *Однако при использовании абстрактных методов вышеуказанного падения производительности не наблюдается.*

7. Работа со свойствами медленнее, чем с полями. Наличие блоков try-catch также существенно снижает производитеьность. Вызов делегата работает быстрее вызова виртуаьлного метода. Стандартные field-like события неэффективно реализуют операции добавления и удаления делегата — эффективнее использовать вместо них List<T>. Все методы синхронизации работают очень медленно.

8. Существуют способы избегания проблем производительности, описанных в предыдущем пункте, однако эти способы приводят к нетривиальным решениям, которыми сложно пользовтся и сопровождать. Поэтому нужно сохранять баланс между производительностью кода и его качеством.

При проектирвоании системы, для достижения ее расширяемости и модифицируемости, было решено следовать набору принципов SOLID [30]. И в первую очередь — принципу SRP (Single Responsibility Principle — англ., принцип единственной обязанности), согластно которому каджый объект должен выполнять только одну задачу [31]. Благодаря использованию SRP удается создать систему, каждый из аспектов поведения которой можно расширять или модифицировать заменяя одни ее компоненты аналогами. К сожалению, принципы SRP и ISP [32] из набора SOLID не применимы к некоторым объектам в системе, поскольку эти объекты исполняют роль тех или иных сущностей в АКР и потому их интерфейсы логически неделимы.

Использование SOLID приводит к созданию многокомпонентных програмных модулей, причем «сборка» из этих компонент работоспособной системы зачастую является трудоемкой задачей [**?**]. Часто для облегчения этой задачи используются т.н. DI-контейнеры [33], однако на начальном этапе от их использования в проекте решено отказатся — добавление зависимости от большой библиотеки при том, что выгода от ее использования не столь очевидна, было призванно нецелесообразным. Но ни что не мешает конечным пользователям библиотеки использовать их.

2.4. Описание интерфейсов координат

Начнем описание програмной модели с описания наиболее базовой ее части — интерфейсов определеяющих координаты. Как уже было сказанно выше

реализациями этих интерфейсов должны быть структуры, причем используемые как генрик-параметры.

Сущность кооррдинаты была разбита на две: собственно координаты и «измеритель» – вспомогательная сущность, через которую реализуется свойство (9) координат. Подобное разбиение необходимо, поскольку результат операции «взятия окрестности» зависит от топологии текущей карты, а значит и сущность реализующая эту операцию должна использовать данные о карте и иногда реализовывать нетривиальную логику. Добавление ссылки на карту в координату нарушает логическую иерархию зависимостей объектов, а добавление логики — принцип SRP.

Таким образом интерфейс координат будет следующим:

```
public interface ICoordinate<C>: ICloneable, IEquatable<C>
    where C: struct, ICoordinate<C>
{
    // Вспомогательный метод:
    C Cast { get; }
    // Строит куб координат, где this — нижняя граница:
    IEnumerable<C> Range(C upperBound);
    // Проверяет, лежит ли текущая координата в кубе заданном аргументами:
    bool IsInRange(C lowerBound, C upperBound);
}
```

Особый интерес в нем вызывает описание генерик-параметра C.

Заключение

Литература

1. Russell S. J., Norvig P. Artificial Intelligence: A Modern Approach. 2nd edition edition. Englewood Cliffs, NJ: Prentice-Hall, 2003.

2. McCarthy J. Artificial intelligence, logic and formalizing common sense // Philosophical Logic and Artificial Intelligence. Dordrecht: Kluwer Academic, 1989. Pp. 161–190.

3. N.J. N. Principles of Artificial Intelligence. Palo Alto, California: Tioga Publishing Company, 1979.

4. Hart P. E., Nilsson N. J., Raphael B. Correction to «A Formal Basis for the Heuristic Determination of Minimum Cost Paths» // SIGART Bull. 1972. no. 37. Pp. 28–29.

5. Engelbrecht A. P. Computational Intelligence: An Introduction. 2nd edition. John Wiley and Sons, 2007.

6. Wlodzislaw D. What is Computational Intelligence and where is it going? // Challenges for Computational Intelligence. 2007. URL: `http://cogprints.org/5358/`.

7. Konar A. Computational Intelligence: Principles, Techniques and Applications. Secaucus, NJ, USA: Springer–Verlag New York, Inc., 2005. ISBN: 3540208984.

8. Гастев Ю. А. Определение // БСЭ. 3-е изд. М., 1988.

9. Хайкин С. Нейронные сети: полный курс. 2е изд. Москва: Вильямс, 2006.

10. Passino K. M., Yurkovich S. Fuzzy Control. Menlo Park, CA: Addison-Wesley-Longman, 1998. P. 475.

11. Емельянов В. В., Курейчик В. В., Курейчик В. М. Теория и практика эволюционного моделирования. Москва: Физматлит, 2003. С. 432.

12. Fogel L. J., Owens A. J., Walsh M. J. Artificial intelligence through simulated evolution. Chichester, WS, UK: Wiley, 1966.

13. Галал Я. М. Эволюционное учение // Энциклопедия Кирилла и Мефодия. М., 2003.

14. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. Москва: Горячая Линия–Телеком, 2007. С. 452.

15. Jong K. A. D. An analysis of the behavior of a class of genetic adaptive systems: Ph. D. thesis / University of Michigan. Ann Arbor, MI, USA, 1975.

16. Michalski R. S. Learnable Evolution Model: Evolutionary Processes Guided by Machine Learning // Machine Learning. 2000. Vol. 38, no. 1-2. Pp. 9–40.

17. Берг Л. С. Номогенез или Эволюция на основе закономерности. Петергоф, 1922. С. 306.

18. Kennedy J., Eberhart R. C. Swarm intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

19. Dorigo M., Maniezzo V., Colorni A. The Ant System: Optimization by a colony of cooperating agents // IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics. 1996. Vol. 26, no. 1. Pp. 29–41.

20. Geem Z. W., Kim J.-H., Loganathan G. V. A New Heuristic Optimization Algorithm: Harmony Search. // Simulation. 2001. Vol. 76, no. 2. Pp. 60–68.

21. EO Evolutionary Computation Framework. URL: `http://eodev.sourceforge.net/`.

22. Evolutionary Computation Framework. URL: http://gp.zemris.fer.hr/ecf/.

23. DeJong K. A. Evolutionary Computation: A Unified Approach. The MIT Press, 2006.

24. Bonabeau E., Dorigo M., Theraulaz G. Swarm Intelligence: From Natural to Artificial Systems. New York, NY: Oxford University Press, 1999.

25. Асанов М. О., Баранский В. А., Расин В. В. Дискретная математика: графы, матроиды, алгоритмы. Ижевск: НИЦ «РХД», 2001. С. 288.

26. Swarm Intelligence in Data Mining, Ed. by A. Abraham, C. Grosan, V. Ramos. Springer, 2006. Vol. 34 of Studies in Computational Intelligence.

27. Rashedi E., Nezamabadi-pour H., Saryazdi S. GSA: A Gravitational Search Algorithm. // Inf. Sci. 2009. Vol. 179, no. 13. Pp. 2232–2248.

28. Baykasoglu A. Evolutionary Computation for Modeling and Optimization. // Comput. J. 2008. Vol. 51, no. 6. P. 743.

29. Hurley S., Whitaker R. M. An agent based approach to site selection for wireless networks // SAC '02: Proceedings of the 2002 ACM symposium on Applied computing. New York, NY, USA: ACM, 2002. Pp. 574–577.

30. Metz S. SOLID Object-Oriented Design. 2009.

31. McLaughlin B. D., Pollice G., West D. Head First Object-Oriented Analysis and Design: A Brain Friendly Guide to OOA&D (Head First). O'Reilly Media, Inc., 2006. ISBN: 0596008678.

32. Martin R. C. The Interface Segregation Principle // cppreport. 1996. — aug. Vol. 8.

33. Fowler M. Inversion of Control Containers and the Dependency Injection pattern. 2004. — January. URL: http://www.itu.dk/courses/VOP/E2005/VOP2005E/8_injection.pdf.