Министерство образования и науки Российской Федерации
Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет Информационных технологий и программирования
Направление (специальность) прикладная математика и информатика
Квалификация (степень) бакалавр прикладной математики и информатики
Специализация ----------
Кафедра Компьютерных технологий **Группа** 4539

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К БАКАЛАВРСКОЙ РАБОТЕ

Генерация вероятностных автоматов методами стимулирующего обучения

Автор квалификационной работы Иринёв А. В. (подпись)

Руководитель Шалыто А. А. (подпись)

Санкт-Петербург, 2008 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Вероятностные автоматы [1–3] хорошо изучены в теоретической кибернетике и используются для моделирования и синтеза сложных систем, имеющих стохастическую природу. Например, это может быть биологическая популяция в развитии, система массового обслуживания и т. п. В последнее время вероятностные автоматы получили свое распространение в таких областях, как робототехника [4, 5], построение аниматов [6] и различных мобильных агентов [7], для которых вероятностный автомат является управляющей моделью. Такой управляющий автомат полностью или частично задаёт поведение мобильного агента. С помощью вероятностной модели агент в состоянии учесть фактор неточности или неопределенности его знаний о действительных состояниях физической системы. Примером проявления данного фактора может служить невозможность мобильного робота точно интерпретировать информацию об окружающем мире, полученную с помощью сенсоров, видеокамеры и т.п.

Для построения управляющих вероятностных автоматов используются методы самообучения (методы, работающие непосредственно с автоматной моделью), которые оценивают реакцию автомата и определённым образом перераспределяют вероятности, связанные с данной реакцией. В общем случае это сводится к проверке определенного условия, на основе которого делается вывод о том, была реакция хорошей или плохой. При этом результирующий автомат будет в большей степени следовать жадной стратегии, так как алгоритм обучения не учитывает, что «хорошая» реакция на данном шаге может привести к плохому результату на следующих шагах.

Очевидно, что эту проблему можно решить с помощью использования более мощных алгоритмов, например, машинного обучения [8]. Однако эти алгоритмы не приспособлены для работы непосредственно с вероятностным автоматом. В данной работе предлагается новый подход, основанный на использовании алгоритмов стимулирующего обучения (*reinforcement learning*)

[9, 10]. Данный раздел машинного обучения предлагает гораздо более мощные средства, нежели условно-рефлекторная схема, описанная выше (по сути, описанная схема является вырожденным случаем задачи стимулирующего обучения). Однако, при таком подходе вероятностный автомат не может обучаться напрямую. Поэтому рассматривается процесс генерации (в англоязычной литературе этому понятию соответствует термин «inference») управляющего автомата на последнем этапе обучения агента. Под генерацией автомата в данном случае понимается перевод внутренней модели обучения в автоматную модель.

Существует класс задач стимулирующего обучения, которые могут выиграть от применения вероятностных автоматов. На примере одной из таких задач в работе показаны некоторые особенности процесса генерации управляющего автомата для мобильных агентов. Также на этом примере рассматривается процесс сжатия выходной автоматной модели.

Целью данной работы является разработка и исследование предлагаемого метода. В частности, в данное исследование входит определение круга задач, генерация управляющих вероятностных автоматов для которых является актуальным, а также рассмотрение особенностей, связанных с процессом генерации автоматной модели.

В первой главе вводятся основные понятия, используемые в работе – дается формальное определение вероятностного автомата и рассматривается алгоритм стимулирующего обучения. Во второй главе кратко описывается исследуемый круг задач. После этого рассматривается процесс генерации вероятностных автоматов с помощью стимулирующего обучения. Также в этой главе приведены результаты сжатия автоматной модели для выбранной задачи на двумерном поле. Наконец, в третьей главе описывается программная реализация обучающегося агента, генерирующего автоматную модель для решения задач на двумерном поле, а также реализация окружающей среды для данного мобильного агента.

ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ

В этой главе будут кратко рассмотрены основные объекты исследования – вероятностные автоматы и стимулирующее обучение.

1.1. Вероятностные автоматы

Приведём определение вероятностного автомата.

Определение 1. Конечный вероятностный автомат – это набор $A =< Q, \Sigma, \delta, I, F, P >$, где

- Q – конечное множество состояний;
- Σ – входной алфавит;
- $\delta \subseteq Q \times \Sigma \times Q$ – функция переходов автомата;
- $I : Q \to R^+$ – распределение вероятностей для начальных состояний;
- $P : \delta \to R^+$ – распределение вероятностей для переходов;
- $F : Q \to R^+$ – распределение вероятностей для сохранения автоматом своего текущего состояния.

Функции I, P, F удовлетворяют следующим условиям:

$$\sum_{q \in Q} I(q) = 1,$$

и

$$\forall q \in Q, \quad F(q) + \sum_{a \in \Sigma, q' \in Q} P(q, a, q') = 1.$$

Вероятностный автомат можно графически представить в виде ориентированного графа переходов. На рис. 1 показано графическое представление автомата с четырьмя состояниями $Q = \{q_0, q_1, q_2, q_3\}$, одним начальным состоянием q_0 (таким состоянием q, для которого $I(q) > 0$) и четырёхсимвольным алфавитом $\Sigma = \{a, b, c, d\}$. Вещественные числа внутри состояний – вероятности сохранения автоматом своего текущего состояния.

Вещественные числа на переходах – вероятности для соответствующих переходов.

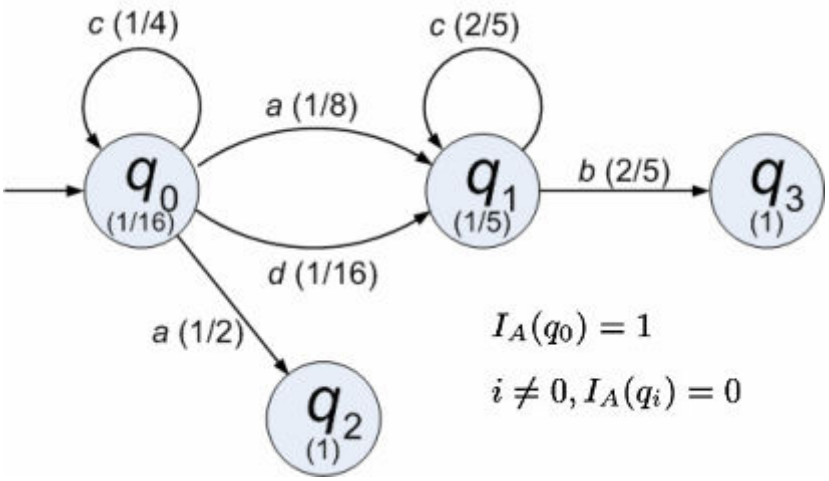

Рис. 1. Графическое представление вероятностного автомата

В данной работе будут рассматриваться только детерминированные вероятностные автоматы.

Определение 2. Конечный вероятностный автомат $A =< Q, \Sigma, \delta, I, F, P >$ называется детерминированным, если:

- $\exists q_0 \in Q$ (начальное состояние), такое, что $I(q_0) = 1$;
- $\forall q \in Q, \quad \forall a \in \Sigma, \quad \left|\{q' : (q, a, q') \in \delta\}\right| \leq 1$.

Таким образом, в случае детерминированного вероятностного автомата переход (q, a, q') полностью определяется значениями q и a.

1.2. Стимулирующее обучение

Стимулирующее обучение – один из разделов машинного обучения, ориентированный на обучение агента посредством взаимодействия с окружающей средой. Это взаимодействие происходит через поощрения (наказания), получаемые агентом в ответ на предпринятые им действия. При этом агент пытается выработать стратегию поведения, которая приносит максимальную прибыль не только здесь и сейчас, но и в долгосрочной перспективе.

1.2.1. Постановка задачи

Рассмотрим некоторую абстрактную задачу. Пусть задан агент, который взаимодействует с окружающей средой, предпринимая какие-то действия. Среда реагирует на действия агента и определенным образом поощряет, либо наказывает его. Таким образом, для того, чтобы агент смог выбрать правильную тактику поведения, ему необходимо следить за поощрениями (наказаниями) от окружающей среды. Такой вид обучения называется стимулирующим обучением (рис. 2).

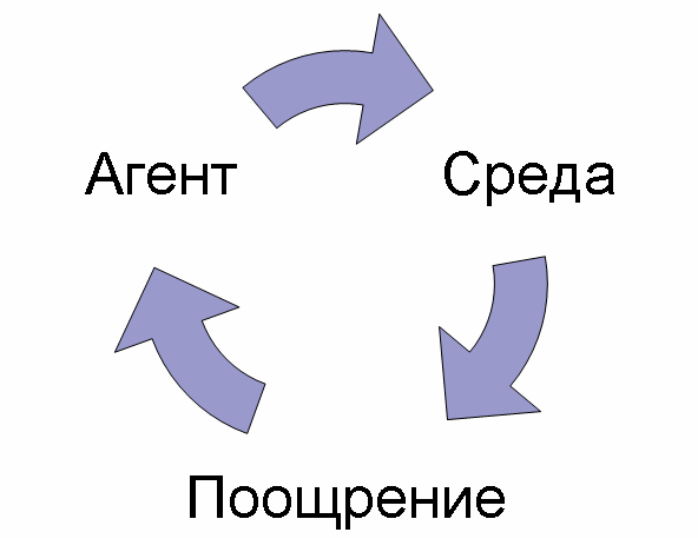

Рис. 2. Цикл обучения

Попробуем формализовать нашу абстрактную задачу:

- S – множество состояний, в которых может находиться агент;
- A – множество доступных агенту действий;
- t – дискретный шаг по времени;
- на каждом шаге агент распознаёт текущее состояние $s_t \in S$ и выбирает действие $a_t \in A$. После этого окружающая среда возвращает агенту поощрение $r_t = r(s_t, a_t)$ и переносит его в новое состояние $s_{t+1} = \delta(s_t, a_t)$.

Сделаем несколько замечаний. Во-первых, в рассматриваемом случае функции δ и r являются частью окружающей среды и не известны агенту. Во-вторых, в данной работе рассматриваются только конечные множества S и A.

Задача агента состоит в том, чтобы найти оптимальную стратегию выбора следующего действия по текущему состоянию $\pi : S \to A$. Для оценки оптимальности найденных стратегий используются различные модели, но на практике, в основном, предпочтение отдаётся модели бесконечного горизонта (*infinite horizon model*) [8–10]. Согласно этой модели, оптимальной называется стратегия, максимизирующая прибыль агента в соответствии со следующей формулы:

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i},$$

где $0 \le \gamma \le 1$ – некоторая константа (в англоязычной литературе называемая *discount factor* [8–10]), которая позволяет в большей степени учитывать награду, полученную от окружающей среды здесь и сейчас, нежели в ближайшем будущем. Это справедливо для большинства реальных задач, так как, во-первых, чем раньше агент получит награду, тем лучше. И, во-вторых, награда, которую он теоретически должен получить через двадцать шагов, может и не достаться агенту, если у него в запасе их осталось всего десять.

1.2.2. Q-learning

Оптимальную стратегию поведения агента можно выразить следующим соотношением:

$$\pi^* \equiv \arg\max_{\pi} V^{\pi}(s), \quad (\forall s).$$

Для удобства обозначим $V^*(s)$ функцию прибыли, соответствующую оптимальной стратегии. Тогда предыдущее тождество можно переписать в виде:

$$\pi^*(s) = \arg\max_a [r(s,a) + \gamma V^*(\delta(s,a))] .$$

Очевидно, что главная задача агента – найти оптимальную стратегию π^*. Однако напрямую вычислить функцию $\pi^* : S \to A$ довольно затруднительно, так как вся информация, доступная агенту – это последовательность поощрений от окружающей среды $r(s_i, a_i)$, где $i = 0, 1, 2, \ldots$.

Эту проблему можно обойти с помощью введения так называемой *Q-функции* [8, 9]:

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) .$$

Эта функция для каждой пары состояние-действие (*s, a*) возвращает максимальную прибыль, которую получит агент, если, находясь в состоянии *s*, выберет действие *a*. Это тождество для оптимальной стратегии можно переписать в виде:

$$\pi^*(s) = \arg\max_a Q(s,a) .$$

Такая перезапись важна по одной простой причине. Эта формула показывает, что для нахождения оптимальной стратегии агенту вместо функции V^* достаточно вычислить функцию *Q*. *Q-функция*, в свою очередь, позволяет выбирать оптимальное действие для текущего состояния без каких-либо знаний о функциях δ и r.

Таким образом, для получения оптимальной стратегии требуется найти значения *Q-функции*, которая вычисляется непосредственно через взаимодействие агента с окружающей средой (через последовательность поощрений $r(s_i, a_i)$, для $i = 0, 1, 2, \ldots$). Описанный метод стимулирующего обучения получил название *Q-learning* [8, 9]. В данной работе именно этот метод взят за основу исследований, так как он является одним из самых распространенных методов стимулирующего обучения.

Однако остается еще один вопрос – каким образом агент должен выбирать следующее действие? Основная задача агента – максимизировать свою прибыль. Поэтому он должен использовать лучшие из найденных стратегий чаще, нежели менее прибыльные. Таким образом, лучшие стратегии будут стабилизироваться, а полученные значения *Q-функции* смогут использоваться при поиске более выгодных стратегий. Для поиска более выгодных стратегий агент не должен следовать жадной стратегии выбора действий, иначе он не сможет выбраться из локального максимума, найденного на первых шагах обучающего алгоритма. Для того, чтобы этого избежать, агенту необходимо время от времени выбирать не самые оптимальные на текущий момент действия. С другой стороны, заниматься только поиском новых стратегий агенту тоже нельзя, так как при этом значительно возрастет время обучения. Поэтому необходимо найти некий компромисс между использованием найденных стратегий и исследованием новых. Эта очень известная проблема получила название *Exploration vs. Exploitation* [8–10]. Она присутствует в любом методе стимулирующего обучения и однозначного ее решения, конечно же, нет.

1.2.3. Пример решения задачи на двумерном поле

Рассмотрим пример, который проиллюстрирует основную идею метода *Q-learning*. На рис. 3 показано двумерное поле (*grid-world environment* [8, 9]) размера 3x3, которая определяет конфигурацию окружающей среды. При этом

- В каждый момент времени агент может находиться только в одной клетке поля, которая определяет его текущее состояние (исключая клетку, помеченную символом «X»). Таким образом, восемь клеток этого поля представляют множество всех возможных состояний агента.
- Стрелками показаны возможные действия агента. Каждая стрелка показывает возможный переход в соседнюю клетку и помечена числом, которое соответствует поощрению агента за выбранное действие.

- Символом «G» помечена клетка, которая является главной целью агента.

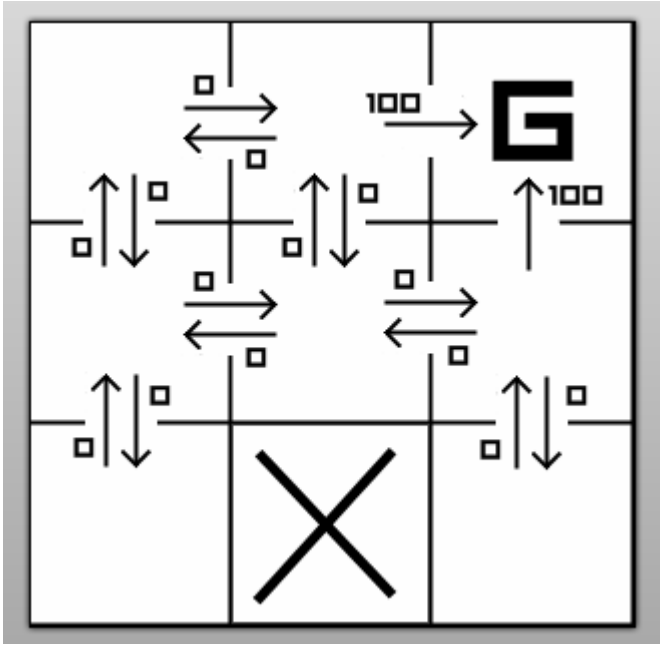

Рис. 3. Конфигурация окружающей агента среды

Из рисунка видно, что агенту требуется прийти в правую верхнюю клетку по кратчайшему пути (прибыль, полученная здесь и сейчас, важнее прибыли, полученной через N шагов). Однако агент, естественно, об этом ничего не знает. Единственная информация, которая ему доступна – это общее число состояний – первоначально агент ничего не знает ни о топологии задачи, ни о действиях, которые он может совершать, находясь в определенной клетке. Поэтому агент может обучаться только следя за поощрениями от окружающей среды. В рассматриваемом случае реакцией на поощрение является изменение *Q-функции*, описанной в предыдущем разделе.

На рис. 4 показаны результирующие значения *Q-функции* (параметр γ принят равным 0.9), полученные в ходе обучения агента на поставленной задаче (рис. 3).

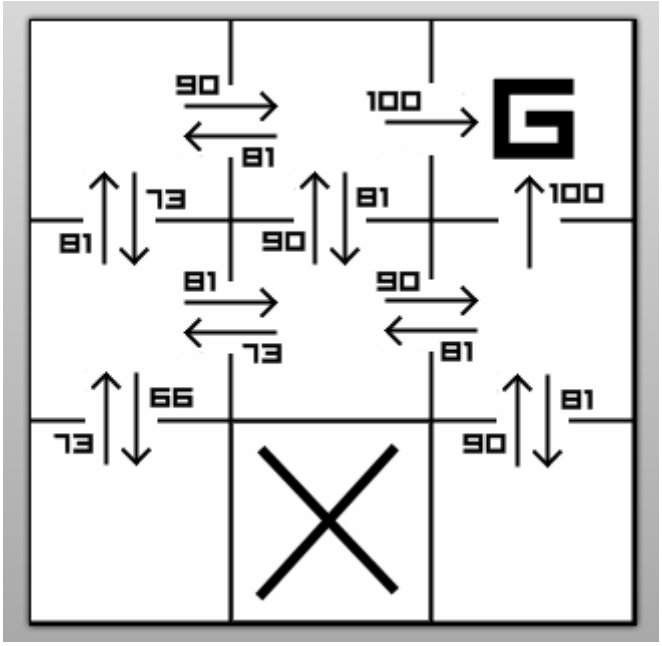

Рис. 4. Значения *Q*-функции

На рис. 5 показаны значения функции $V^*(s)$ – искомой функции прибыли, соответствующей оптимальной стратегии $\pi^*(s)$. В качестве значения функции прибыли для данного состояния берётся максимальная прибыль по всем доступным для агента в этом состоянии действиям (рис. 4).

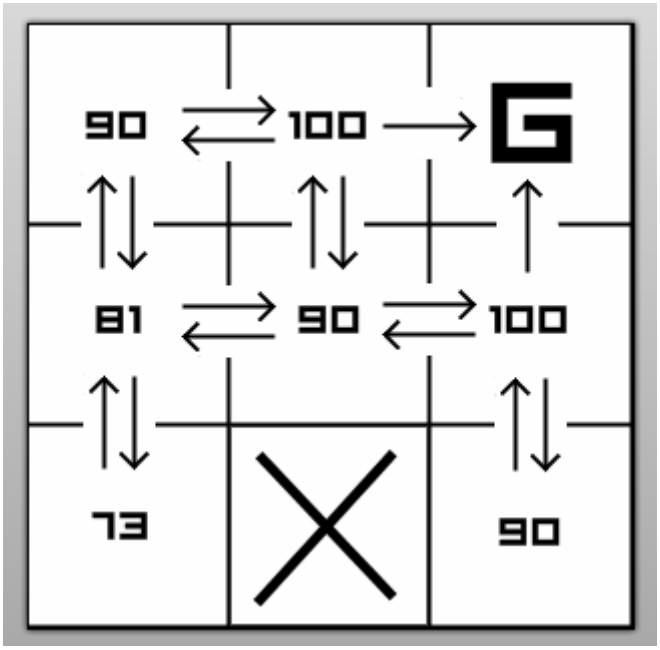

Рис. 5. Значения функции прибыли для оптимальной стратегии

Найденная оптимальная стратегия приведена на рис. 6. Здесь с каждым состоянием ассоциировано ровно одно действие, которое соответствует максимальной прибыли, показанной на рис. 5.

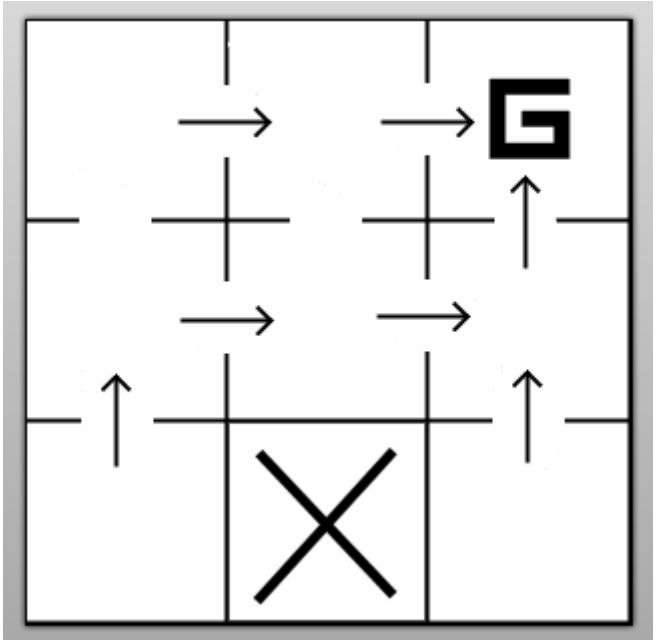

Рис. 6. Пример оптимальной стратегии

Выводы по главе 1

При постановке задачи стимулирующего обучения использовались такие понятия, как состояние агента и действия, ассоциированные с данным состоянием. Нетрудно заметить, что такая постановка задачи очень близка к автоматной терминологии, описанной в разд. 1.1. Поэтому встают два вопроса. Во-первых, как связать эти две области. И, во-вторых, какую выгоду можно получить от совместного использования стимулирующего обучения и вероятностных автоматов. Рассмотрению этих вопросов и будет посвящена оставшаяся часть данной работы.

ГЛАВА 2. ГЕНЕРАЦИЯ АВТОМАТНОЙ МОДЕЛИ

В данной главе будет рассматриваться процесс генерации вероятностного управляющего автомата для агента, обучающегося с помощью поощрений. При этом решение поставленной задачи происходит согласно следующей схеме:

1. Постановка задачи обучения.
2. Использование алгоритма стимулирующего обучения (получение внутренней модели обучения – *Q-функции*).
3. Генерация вероятностного автомата, который задаёт поведение агента.
4. Сжатие вероятностного автомата.
5. Использование вероятностного автомата для работы в среде.

В этой главе будет обозначена область задач, для которой применение такого подхода может оказаться актуальным. Также здесь будет подробно описан процесс генерации управляющего автомата и на конкретном примере будут показаны некоторые особенности, связанные с генерацией и сжатием автоматной модели.

2.1. Системы со стохастическим поведением

Прежде всего, необходимо выделить ту область задач стимулирующего обучения, применение вероятностных автоматов для которой может оказаться актуальным. В разд.1.2.3 был рассмотрен простейший пример обучения на двумерном поле и показана оптимальная стратегия поведения агента (рис. 6). Согласно найденной стратегии, с каждым состоянием ассоциировано ровно одно действие, которое соответствует максимальной прибыли. В такой постановке задачи агенту для максимизации прибыли не важна информация, ассоциированная с остальными действиями. Таким образом, распределение вероятностей по соответствующим действиям для определенного состояния также не даёт никаких преимуществ.

Поэтому применение вероятностных автоматов для таких «статических» задач максимизации себя не оправдывает – среда должна иметь стохастическое поведение. В то же время, это поведение должно подчиняться некоторым глобальным закономерностям. В таком случае, во-первых, будет важен тот факт, что реакция агента является неоднозначной. И, во-вторых, агент сможет подстроить свое поведение под некоторый случайный (или псевдослучайный) процесс, заложенный внутри задачи.

Примером такой задачи может служить система автоматического управления движением транспорта на перекрёстке двух улиц с разной интенсивностью движения. Для простоты рассмотрим систему с двумя состояниями:

- *1* – проезд по магистрали (улица с интенсивным движением) открыт;
- *2* – магистраль перекрыта, разрешено поперечное движение.

Входных сигналов тоже два:

- *S1* – на поперечной улице ждет транспорт;
- *S2* – поперечная улица пуста.

На рис. 7 показан пример вероятностного автомата, соответствующего данной задаче. Если проезд по магистрали открыт и на поперечной улице ждет транспорт, то автомат на следующем такте работы с вероятностью 0.3 закроет проезд по магистрали и с вероятностью 0.7 оставит проезд открытым. С другой стороны, если открыт проезд по поперечной улице и там есть транспорт, то с вероятностью 0.5 проезд останется открытым и с той же вероятностью движение возобновится по магистрали. Если же магистраль открыта и транспорта на поперечной улице нет, то автомат останется в том же положении. Также, в ситуации, когда магистраль закрыта и на поперечной улице нет транспорта, автомат на следующем такте работы откроет проезд по магистрали. Функция распределения вероятностей для сохранения автоматом своего текущего состояния равна нулю для обоих состояний.

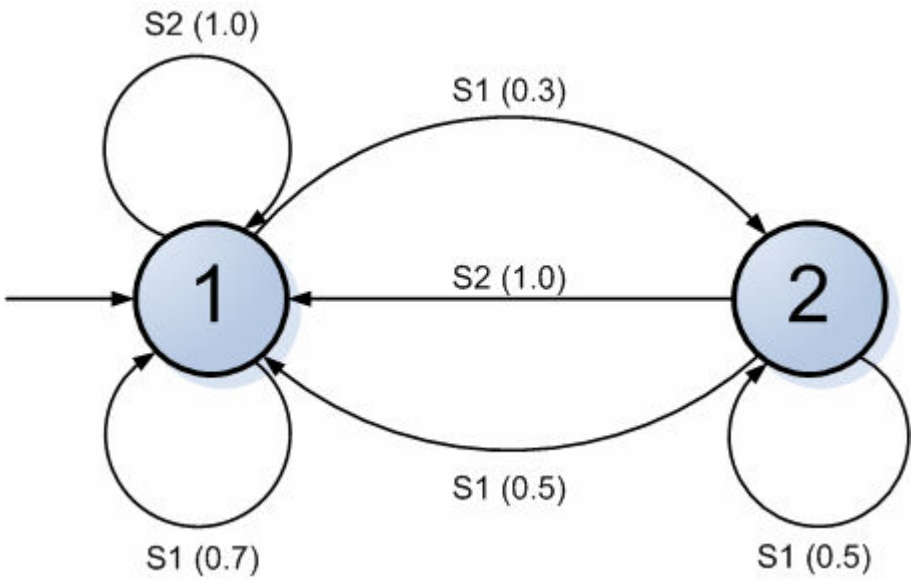

Рис. 7. Пример вероятностного автомата, решающего задачу автоматического управления движением транспорта

Такой автомат по мере надобности пропускает поперечный транспорт, но не перекрывает магистраль при появлении на поперечном направлении каждой отдельной машины. Численные значения вероятностей переходов и время основного такта работы автомата выбираются на этапе обучения агента исходя из конкретного транспортного режима. При этом поощрения могут даваться исходя из дорожной ситуации (например, отсутствие/наличие пробок на магистрали и поперечной улице, максимальное количество машин, ждущих на светофоре и т.д.).

2.2. Процесс генерации вероятностных автоматов

Под генерацией выходного автомата в данной работе понимается перевод внутренней модели, которая формируется агентом в процессе обучения, в автоматную. Термин «генерация» выбран как наиболее подходящий по смыслу перевод термина «inference», который используется в англоязычной литературе. По мнению автора, непосредственное обучение вероятностного автомата является непрактичным – при работе обучающего алгоритма напрямую с распределениями вероятностей теряется информация о количественных характеристиках полезности каждого действия. Другими

словами, если на очередном шаге алгоритма агент получает поощрение, например, в виде десяти абстрактных единиц, то агент не сможет из одной автоматной модели делать выводы о том, много это или мало. Следовательно, непонятно, каким образом эффективно перераспределять вероятности между выбранным действием и соответствующими ему альтернативами.

При постановке задачи стимулирующего обучения использовались такие понятия, как состояние агента и действия, ассоциированные с данным состоянием. Нетрудно заметить, что такая постановка задачи очень близка к автоматной терминологии. Поэтому в конце процесса обучения, когда агенту доступна определенная информация, на основе которой он может делать выбор в пользу того и или иного действия, возможен перевод внутренней модели обучения в автоматную.

Как отмечалось в разд.1.2.2, в данной работе используется метод стимулирующего обучения под названием *Q-learning*. Соответственно, внутренней моделью для данного метода является *Q-функция*, которая показывает потенциальную прибыль агента для всех пар состояние-действие. На основе этой модели для каждого состояния агента логично сопоставить вероятности переходов выходного автомата каждому из возможных действий агента пропорционально ожидаемой прибыли для этого действия. Таким образом, находясь в состоянии s_i, агент должен выбирать действие a_j $(0 \le j \le n_i)$ с вероятностью:

$$p_j = \frac{R(s_i, a_j)}{\sum_{k=0}^{n_i} R(s_i, a_k)},$$

где

$$R(s_i, a_k) = \begin{cases} Q(s_i, a_k), & при \;\; Q(s_i, a_k) \ge 0 \\ 0, & при \;\; Q(s_i, a_k) < 0 \end{cases},$$

n_i – число доступных агенту действий в состоянии s_i (при этом a_0 означает, что агент не предпринимает никаких действий и остается в текущем состоянии).

Эта формула, в частности, присваивает нулевые вероятности действиям с отрицательным значением *Q-функции*. Таким образом, исключается возможность того, что агент совершит заранее невыгодное для него действие. Заметим, что здесь речь не идет о действиях, которые просто приносят отрицательный результат – такое действие вполне может входить в оптимальную стратегию [9]. Примером заранее невыгодного действия может служить попытка мобильного робота совершить действие «идти вперед», когда перед ним находится стена. В данном случае, окружающая среда оставляет агента в том же состоянии и возвращает отрицательное число при «неправильном» действии. При традиционном подходе, когда агент следует оптимальной стратегии π^*, такое действие исключается в силу того, что потенциальная прибыль для него будет в любом случае меньше, чем для любого осмысленного действия. При использовании подхода с генерацией вероятностного автомата следует учитывать тот факт, что заранее невыгодные действия могут приводить к положительным значениям *Q-функции*. Для того, чтобы этого не случилось, среда в качестве наказания должна возвращать агенту большое отрицательное число за каждое такое действие. В противном случае, может пострадать эффективность полученного решения.

2.3. Тестовый пример

Рассмотрим подробнее процесс генерации вероятностного автомата на примере следующей задачи. Задано двумерное поле размера *NxM* и несколько мобильных роботов, которые подчиняются следующим правилам:

- В каждый момент времени каждый робот может находиться только в одной клетке поля, которая определяет его текущее состояние.

- В каждом состоянии всем роботам доступны четыре действия («идти вверх», «идти вправо», «идти вниз» и «идти влево»). За попытку выхода с поля робот получает большой штраф.
- Существуют несколько выделенных клеток, попадая в которые робот каким-то образом поощряется и переносится в новую, заранее определенную позицию на двумерном поле. При этом в течение следующих *K* шагов любой робот, пришедший в эту выделенную клетку, переносится без поощрения.
- Все роботы начинают двигаться из верхнего левого угла и у каждого есть в запасе имеется ровно *S* шагов.

  Необходимо максимизировать общую прибыль всех роботов.

Пусть на стадии обучения имеется только один мобильный робот и поле, показанное на рис. 8. Здесь символом «G» обозначены выделенные клетки, в которых робот находиться не может. Стрелка указывает, куда робот переносится, попадая в соответствующую выделенную клетку, а число рядом со стрелкой указывает размер поощрения. Серым цветом закрашена стартовая клетка, с которой мобильный робот начинает свое движение.

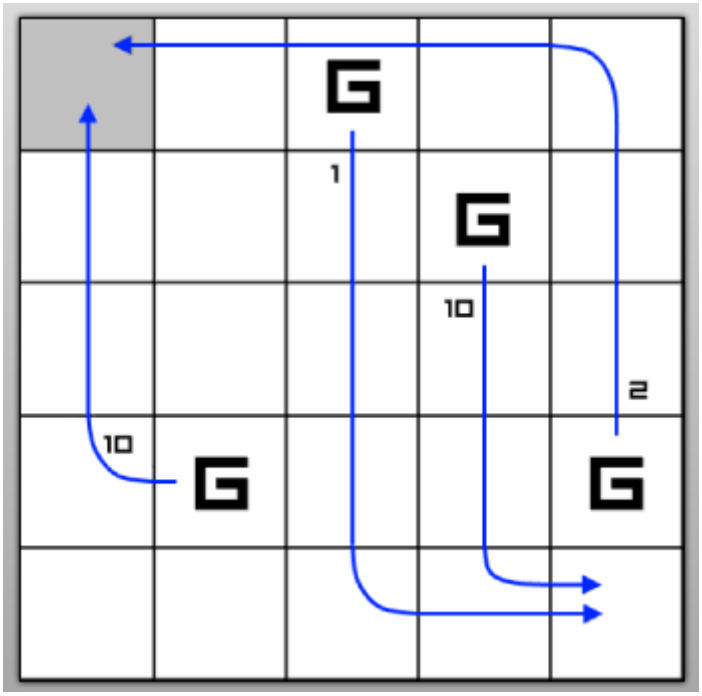

Рис. 8. Конфигурация окружающей среды

Пусть состояния нумеруются построчно слева направо. Нумерация начинается с нуля. При этом правый верхний угол тестового поля имеет номер «4», левый нижний – номер «20». Действия также нумеруются с нуля. При этом 0 – «идти вверх», 1 – «идти вправо», 2 – «идти вниз» и 3 – «идти влево».

Выполним обучение мобильного робота согласно алгоритму, описанному в разд. 1.2.2. В качестве автоматной модели для поставленной задачи обучения выбрана стохастическая матрица переходов [11]. Строки этой матрицы соответствуют состояниям мобильного робота, столбцы – возможным действиям. Таким образом, стохастическая матрица переходов сопоставляет с каждой парой состояние-действие значение вероятности данного действия. Значения соответствующих вероятностей стохастической матрицы переходов вычисляются по формуле, показанной в предыдущем разделе.

Результирующая стохастическая матрица переходов вероятностного автомата приведена в табл. 1 ($K = 10$, $S = 100$). При этом нулевые вероятности выбора действия говорят о том, что данное действие является попыткой выхода с поля и приводит к отрицательной оценке суммарной прибыли. Состояния «2», «8», «16» и «19» не входят в результирующую модель, так как

этим состояниям соответствуют выделенные клетки тестового поля – мобильный робот не может находиться в перечисленных состояниях в связи со спецификой поставленной задачи.

Таблица 1. Стохастическая матрица переходов

	0	1	2	3
0	0	0,5	0,5	0
1	0	0,32	0,38	0,3
3	0	0,32	0,4	0,28
4	0	0	0,5	0,5
5	0,29	0,36	0,35	0
6	0,22	0,28	0,28	0,22
7	0,2	0,3	0,25	0,25
9	0,31	0	0,31	0,38
10	0,29	0,36	0,35	0
11	0,24	0,24	0,29	0,23
12	0,25	0,25	0,25	0,25
13	0,29	0,24	0,24	0,23
14	0,36	0	0,29	0,35
15	0,31	0,38	0,31	0
17	0,24	0,23	0,23	0,3
18	0,28	0,22	0,22	0,28
20	0,5	0,5	0	0
21	0,38	0,31	0	0,31
22	0,36	0,29	0	0,35
23	0,36	0,29	0	0,35
24	0,5	0	0	0,5

В приведённой постановке задачи жадная стратегия поведения не применима. Все мобильные роботы будут ходить по одному маршруту, но

поощряться окружающей средой будет только один. Все остальные роботы будут «голодать», даже несмотря на то, что на поле могут присутствовать другие выделенные клетки, через которые не проходит «жадный» маршрут.

Эту проблему можно решить с помощью вероятностного автомата – роботы будут выбирать следующее действие пропорционально ожидаемой награде и с пропорциональными вероятностями выбирать маршруты ко всем выделенным клеткам поля. В такой постановке задачу можно рассматривать как систему со стохастическим поведением – если зафиксировать одного мобильного робота, то для него изменение конфигурации окружающей среды происходит с определенной закономерностью, но сама природа изменений является стохастической.

Данные предположения подтверждаются опытами, выполненными автором с помощью моделирования на компьютере. Так, для тестового поля (рис. 8), рост общей прибыли при увеличении числа роботов показан на рис. 9. Здесь значения прибыли усреднены по пятидесяти запускам сгенерированного автомата. При выборе жадной стратегии, независимо от числа роботов, общая прибыль равна ста.

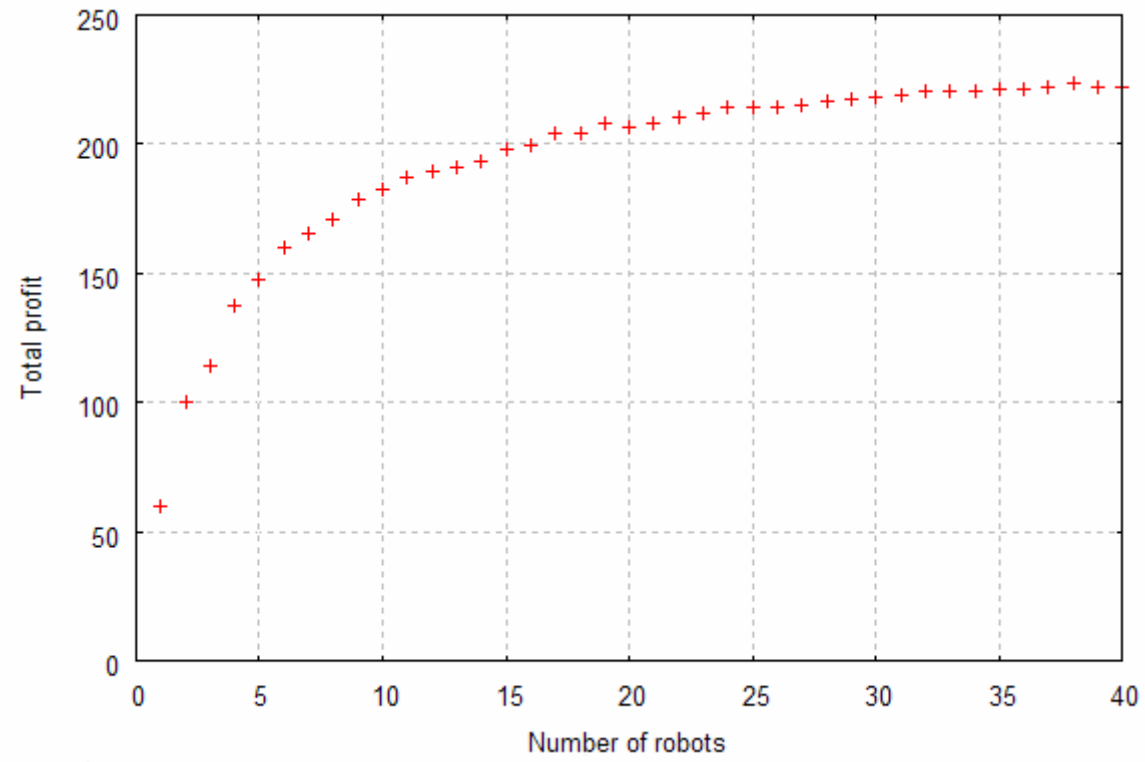

Рис. 9. Рост общей прибыли при добавлении новых роботов (по оси X отложено число роботов, одновременно находящихся на поле, по оси Y – общая прибыль, получаемая с тестового полигона), $K = 10$ и $S = 100$

2.4. Сжатие выходного автомата

При использовании метода *Q-learning* агент вынужден хранить на этапе обучения оценки для всех возможных пар состояние-действие. Так как в общем случае таких пар может быть довольно много, это негативно отразится на размере выходного автомата, который, в отличие от жадной стратегии выбора следующего действия, вынужден хранить значения вероятностей переходов, ассоциированные с каждой парой состояние-действие.

С другой стороны, вероятности посещения отдельных вершин во время работы автомата могут значительно отличаться. Аналогичные рассуждения применимы и к действиям, которые совершает агент во время работы алгоритма. Поэтому в зависимости от специфики решаемой задачи возможна ситуация, когда удаление из выходного вероятностного автомата отдельных вершин или переходов не приведет к ухудшению общей функции прибыли.

Таким образом, в некоторых случаях можно получить более компактную автоматную модель с теми же оценочными характеристиками.

Сделаем следующее замечание. Тот факт, что автомат редко приходит в данное состояние, не всегда означает, что это состояние является хорошим кандидатом на удаление. Например, два состояния могут быть промежуточными на пути к третьему состоянию, за приход в которое автомат награждается большим поощрением. Следовательно, эти два состояния будут посещаться довольно часто, хотя удаление одного из них никак не повредит общей функции прибыли. С другой стороны, может иметь место четвертое состояние, которое тоже приносит определённую прибыль, но не такую большую, как третье. Поэтому возможна ситуация, в которой вероятность прийти в такое состояние будет меньше, чем вероятность прийти в одно из промежуточных состояний.

Для того чтобы проиллюстрировать это, вернемся к задаче, рассмотренной в предыдущем разделе. Будем удалять состояния управляющего автомата в порядке возрастания вероятностей посещения автоматом данного состояния. В табл. 2 показано число посещений управляющим автоматом для каждой клетки поля после стадии обучения.

Таблица 2. Статистика посещений клеток поля

	0	1	2	3	4
0	641360	436366	169553	1622	3408
1	516152	442230	154678	72009	8861
2	280563	254605	139459	73699	19933
3	129343	182162	109706	117067	180532
4	62413	74430	122599	210100	301406

Согласно собранной статистике, удаление вершин следует производить в следующем порядке: (0; 3), (0; 4), (1; 4), (2; 4), (4; 0), (1; 3), (2; 3), (4; 1), (3; 2),

(3; 3), (4; 2), (3; 0), (2; 2), (1; 2), (0; 2), (3; 4), (3; 1), (4; 3), (2; 1), (2; 0), (4; 4), (0; 1), (1; 1), (1; 0), (0; 0).

На рис. 10 показан график изменения функций общей прибыли для шести управляющих автоматов, которые получены путем удаления первых шести состояний выходного вероятностного автомата в порядке, показанном выше. Значения прибыли усреднены по пятидесяти запускам сгенерированного автомата. При удалении пяти самых «непопулярных» состояний общая функция прибыли практически не изменяется – на рисунке соответствующие графики с незначительным отклонением идут вдоль кривой, образованной точками исходного управляющего автомата (рис. 9). При удалении шестого состояния происходит заметное ухудшение оценочной функции прибыли – на рисунке соответствующий график расположен значительно ниже исходного. Такое падение производительности автомата происходит потому, что последнее удаленное состояние – одна из выделенных клеток тестового поля.

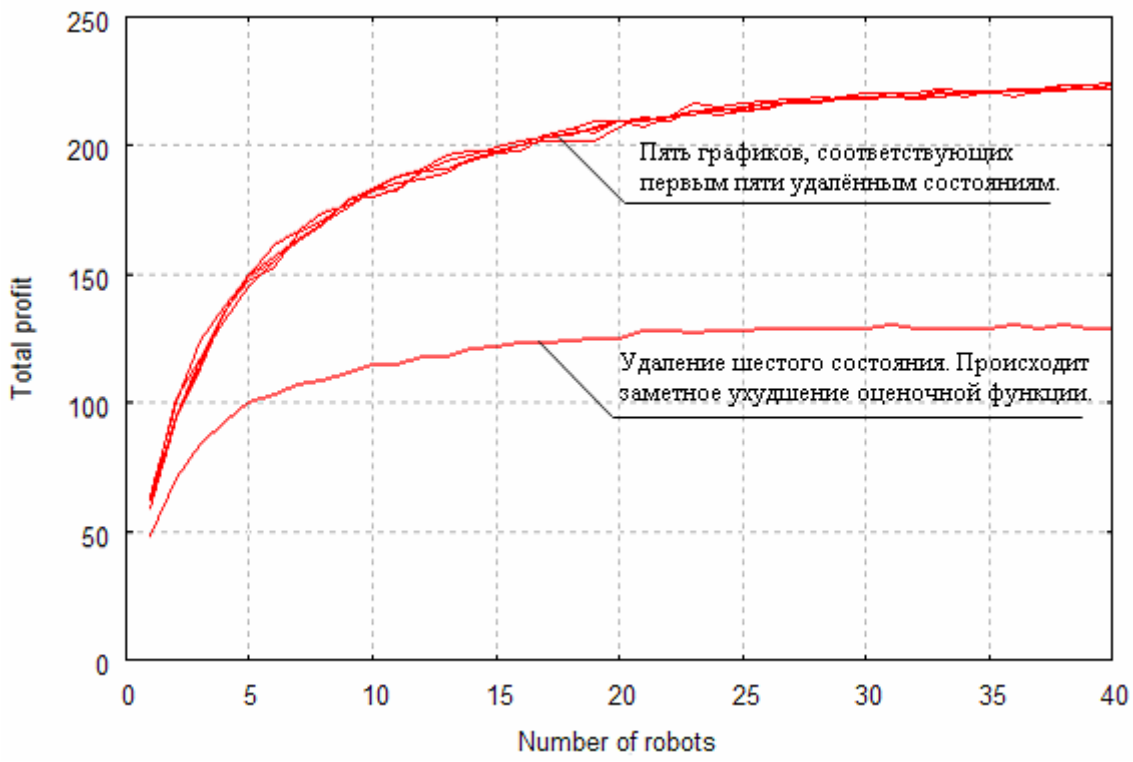

Рис. 10. Изменение общей функции прибыли при удалении состояний управляющего автомата

На рис. 11 показан график изменения функций общей прибыли для пятнадцати управляющих автоматов, которые получены путем удаления первых пятнадцати самых «непопулярных» состояний выходного вероятностного автомата, исключая выделенные клетки тестового поля. На рисунке видно, что удаление первых тринадцати вершин приводит к постепенному спаду производительности, в то время как удаление только четырнадцатого по счету приводить к серьезному ухудшению общей функции прибыли.

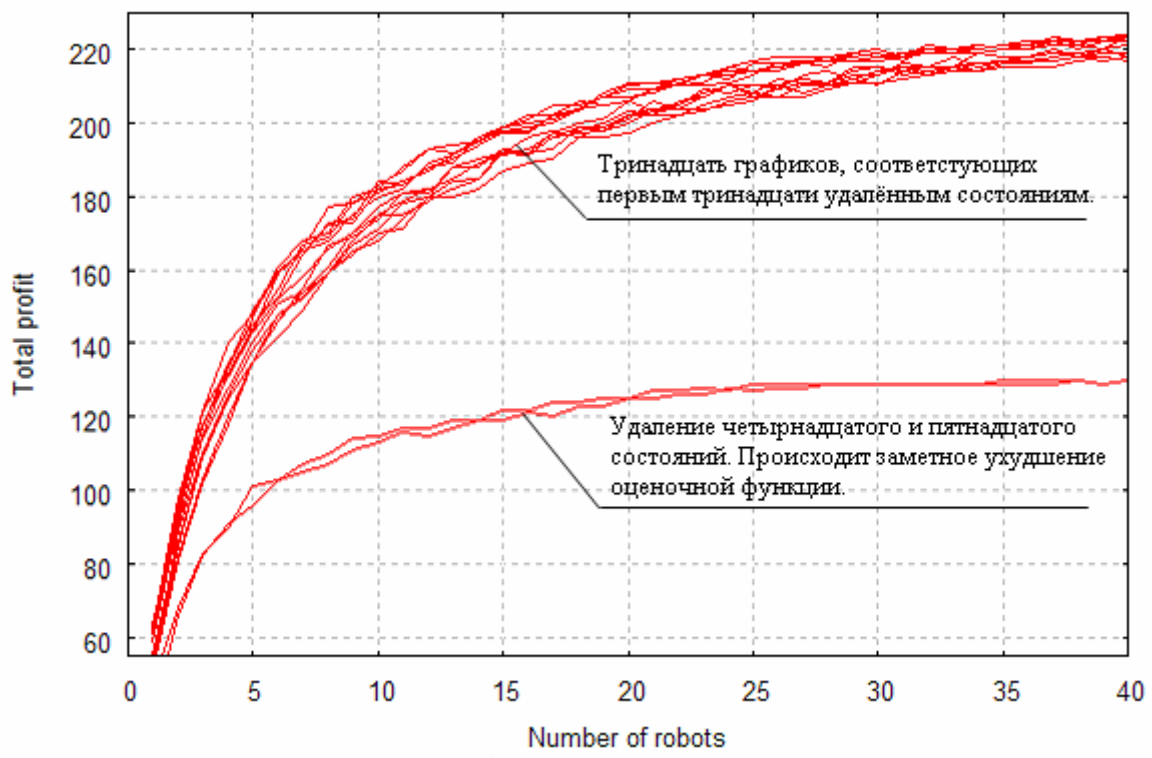

Рис. 11. Изменение общей функции прибыли при удалении состояний управляющего автомата, исключая выделенные клетки

Выводы по главе 2

Непосредственное обучение вероятностного автомата является непрактичным, так как при работе обучающего алгоритма напрямую с распределениями вероятностей теряется информация о количественных характеристиках полезности каждого действия.

В данной главе был подробно рассмотрен процесс генерации вероятностного автомата из обучающей модели для задачи стимулирующего обучения. Если в постановке задачи заложены некоторые стохастические процессы (элемент неопределённости), рассмотренный подход может давать определённые преумущества перед непосредственным использованием методов стимулирующего обучения.

Однако при использовании подхода с генерацией вероятностного управляющего автомата необходимо учитывать возможность использования агентом заранее невыгодных действий, которые при обычной схеме использования могут приводить к положительным значениям *Q-функции*. Для того, чтобы этого не случилось, среда в качестве наказания должна возвращать агенту большое отрицательное число за каждое такое действие. В противном случае может пострадать эффективность полученного решения.

В зависимости от специфики решаемой задачи, предлагается проводить (если это возможно) сжатие выходной автоматной модели. При этом меньшая вероятность посещения данного состояния в процессе работы автомата ещё не является достаточным условием для выбора этого действия в качестве кандидата на удаление.

ГЛАВА 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

В этой главе будет описывается реализация универсального *Q-learning* агента, решающего задачи на двумерной поле. Также будет приведен пример реализации класса, описывающего функциональность окружающей среды. Данный класс полностью определяет задачу обучения, поставленную перед агентом. Программный код написан на языке *C#* [12] и может быть запущен на платформах *.NET v1.0-3.5* [13].

3.1. Реализация агента

Все практические эксперименты, результаты которых были приведены в предыдущей главе, были получены в ходе моделирования на компьютере. Для этого был написан универсальный агент, решающий задачи на двумерном поле с помощью алгоритма *Q-learning*. Агент является универсальным в том смысле, что задачи, которые он решает, никак не зависят от самого агента и могут быть реализованы отдельно. Для этого как класс, реализующий функциональность агента, так и класс, описывающий задачу с помощью моделирования условий окружающей среды, должны взаимодействовать согласно правилам, описанным ниже.

Прежде всего, каждый класс, реализующий функциональность агента, наследуется от общего интерфейса:

```
interface IAgent
{
    /// <summary>
    /// Performs one iteration of training phase.
    /// </summary>
    void Train();

    /// <summary>
    /// Generates resulting stochastic matrix.
    /// Should be called after training phase was completed.
    /// </summary>
    /// <returns>Resulting stochastic matrix</returns>
    double[,] GenerateStochasticMatrix();
}
```

Метод `Train()` выполняет обучение агента. Им же определяется число действий, совершаемых агентом в процессе обучения. Метод `GenerateStochasticMatrix()` вызывается после окончания обучения агента и возвращает стохастическую матрицу, которая является результирующей автоматной моделью.

Несмотря на то, что в рамках данной работы реализован только один агент – `QLearningAgent`, ничто не мешает использовать вместо него любого другого агента, написанного согласно приведенному интерфейсу.

Сам класс, реализующий функциональность агента, выглядит следующим образом:

```
class QLearningAgent : IAgent
{
    // current task
    private readonly IEnvironment env;
    // number of lines in Grid-world environment task
    private readonly int numLines;
    // number of columns in Grid-world environment task
    private readonly int numColumns;
    // q-function, represents potential rewards
    // for all <state, action> pairs
    private double[,] q;
    // zero-based state of agent
    private int state;

    // number of moves for one training phase
    private static readonly int numOfMoves;
    // q-learning step-size parameter
    private static readonly double stepSize;
    // number of actions for each state
    private const int numOfActions = 4;

    private static Random rnd = new Random();

    #region Constructors       ... #endregion

    #region Supporting Methods ... #endregion

    #region IAgent Members     ... #endregion
}
```

Здесь обьект класса `Environment` инкапсулирует в себе текущую задачу, решаемую агентом (этот класс будет подробнее описан в следующем

разделе). Любое взаимодействие с окружающей средой моделируется посредством вызова методов этого обьекта.

Также стоит отметить, что вся доступная агенту информация об окружающей среде – это размер тестового поля и число доступных действия для каждого состояния. Рассматриваемый агент специализирован в решении задач на двумерном поле, поэтому доступных действий всего четыре. В то же время агент не обязан интерпретировать эти действия как «идти вверх», «идти вправо», «идти вниз» и «идти влево». Эти действия интерпретирует окружающая среда, которая получает соотвествующий действию численный эквивалент. Размер тестового поля определяется параметрами `numLines` и `numColumns`, которые содержат число строк и столбцов двумерного поля. Таким образом, никакая информация (например, связанная с топологией решаемой задачи), из которой агент может непосредственно сделать некоторые выводы об оптимальности той или иной стратегии, ему недоступна. Эта постановка задачи полностью соотвествует определению стимулирующего обучения – агент находит оптимальную стратегию из последовательности поощрений и наказаний, полученных от окружающей среды.

Внутренняя модель обучения агента хранится в матрице `q`, которая является программным представлением *Q-функции* и каждой паре состояние-действие сопоставляет некоторое вещественное число – потенциальную награду за выбор данного действия в данном состоянии. Текущее состояние агента хранится в переменной `state`. Для удобства программирования состояния нумеруются от нуля до `numLines * numColumns - 1`. Интерпретация числового значения, соответствующего состоянию, также целиком зависит от обьекта, описывающего функциональность окружающей среды.

Значение `numOfMoves` соотвествует параметру *S*, описанному при постановке задачи обучения в разд. 2.3, и используется для определения длительности обучения.

Значение `stepSize` соотвествует константе $0 \le \gamma \le 1$ (разд. 1.2.1), используемой в процессе обучения модели. Эта константа позволяет в большей степени учитывать награду, полученную от окружающей среды здесь и сейчас, нежели в ближайшем будущем.

Инициализация агента происходит через вызов конструкторов. При инициализации класса параметрам `numOfMoves` и `stepSize` задаются значения, прочитанные из конфигурационного файла. При инициализации объекта, параметру, отвечающему за текущее состояние агента, присваивается ноль, что соответствует верхнему левому углу тестового поля. Таким образом, все агенты начинают свое обучение из одного и того же состояния.

Самая важная часть класса – реализация методов, описанных в интерфейсе `IAgent`. Метод `Train()` отвечает непосредственно за процесс обучения агента:

```
/// <summary>
/// Performs one iteration of training phase.
/// </summary>
public void Train()
{
    const double explorationRate = 0.3;

    for (int i = 0; i < numOfMoves; i++)
    {
        int oldState = state;

        int action = (rnd.NextDouble() <= explorationRate) ?
            rnd.Next(numOfActions) :
            GetActionWithMaximumQ(oldState);
        int reward = env.PerformAction(ref state, action);
        q[oldState, action] =
            reward + stepSize * GetMaximumQByState(state);
    }
}
```

В разд. 1.2.2 была описана проблема под названием *Exploration vs. Exploitation* и отмечалось, что однозначного ответа на вопрос, каким образом при обучении агент должен выбирать следующее действие, не существует. В данном случае выбор очередного действия происходит следующим образом: с вероятностью 30% агент выбирает случайное действие из четырех доступных

на данном шаге, во всех остальных случаях он выбирает действие, которое на данном этапе работы алгоритма имеет максимальное значение *Q-функции*. Это значение вероятности выбора случайного действия получено из экспериментов.

Метод `GenerateStochasticMatrix()` возвращает результирующую стохастическую матрицу, которая является автоматной моделью для нашей задачи на двумерном поле:

```
/// <summary>
/// Generates resulting stochastic matrix.
/// Should be called after training phase was completed.
/// </summary>
/// <returns>Resulting stochastic matrix</returns>
public double[,] GenerateStochasticMatrix()
{
    double[,] matrix =
        new double[numLines * numColumns, numOfActions];

    double sum;
    for (int i = 0; i < numLines * numColumns; i++)
    {
        sum = 0;
        for (int j = 0; j < numOfActions; j++)
        {
            if (q[i, j] > 0) sum += q[i, j];
        }
        for (int j = 0; j < numOfActions; j++)
        {
            matrix[i, j] = (q[i, j] > 0) ? (q[i, j] / sum) : 0;
        }
    }

    return matrix;
}
```

Здесь действиям с отрицательными значениями *Q-функции* присваиваются нулевые вероятности переходов. Таким образом, исключается возможность того, что агент совершит заранее невыгодное для него действие. Также, в данном случае используется соглашение о том, что окружающая среда в качестве наказания за заранее невыгодное действие возвращает агенту большое отрицательное число. Эта проблема была подробно описана в разд. 2.2.

3.2. Реализация окружающей среды

Класс, реализующий функциональность окружающей среды, выглядит следующим образом:

```
class Environment : IEnvironment
{
    private Cell[,] field;
    private int numLines;
    private int numColumns;

    #region Initialization        ... #endregion

    #region IEnvironment Members ... #endregion

    ...
}
```

Обьект `Cell` представляет клетку тестового поля и содержит координаты этой клетки, а также любую другую информацию, связанную со спецификой обработки данной клетки в условиях поставленной агенту задачи. Например, обьект может содержать логическую переменную, характеризующую данную клетку, как транспортирующую агента в заранее определенное место тестового поля, и координаты пункта назначения.

Основной метод, представляющий интерес при реализации класса – `PerformAction(ref int state, int action)`. Любое взаимодействие агента и окружающей среды ограничивается вызовом данного метода. Поэтому он вынесен в отдельный интерфейс `IEnvironment`, с которым работает класс, реализующий функциональность агента. Данный метод принимает ссылку на текущее состояние агента и числовой идентификатор выполняемого действия, изменяет переданное ему состояние согласно правилам моделируемой среды и возвращает соотвествующее выбранному действию поощрение. Реализация метода для рассмотренного в разд. 2.3 примера выглядит следующим образом:

```
/// <summary>
/// Performs agent's action which changes current state
/// under environment rules.
/// </summary>
/// <param name="state">Current state</param>
```

```
/// <param name="action">Chosen action</param>
/// <returns>Reward for performed action</returns>
public int PerformAction(ref int state, int action)
{
    int i = state / numColumns + 1;
    int j = state % numColumns + 1;
    int absorbingReward = 0;
    const int wallPenalty = -1000;

    switch (action)
    {
        case 0: // up
            if (i == 1)
                return wallPenalty;
            i = (i - 1);
            absorbingReward = CheckAbsorbingState(ref i, ref j);
            break;
        case 1: // right
            if (j == numColumns)
                return wallPenalty;
            j = (j + 1);
            absorbingReward = CheckAbsorbingState(ref i, ref j);
            break;
        case 2: // down
            if (i == numLines)
                return wallPenalty;
            i = (i + 1);
            absorbingReward = CheckAbsorbingState(ref i, ref j);
            break;
        case 3: // left
            if (j == 1)
                return wallPenalty;
            j = (j - 1);
            absorbingReward = CheckAbsorbingState(ref i, ref j);
            break;
        default:
            throw new EnvironmentException("Incorrect action.");
    }

    state = (i - 1) * numColumns + j - 1;
    return cells[i - 1, j - 1].reward + absorbingReward;
}
```

Здесь агент обучается в так называемой закрытой комнате – благодаря специфике реализации окружающей среды, агент не может выйти за пределы двумерного поля. При этом за попытку выхода с поля среда в качестве штрафа возвращает большое отрицательное число.

Выводы по главе 3

Реализацию агента и класса, описывающего функциональность окружающей среды, следует отделять друг от друга. Это позволяет создавать универсальных агентов (в рассмотренном случае агент специализируется в решении задач максимизации на двумерном поле), программная реализация которых не зависит от конкретных реализаций класса окружающей среды. Такой подход способствует повторному использованию программного кода и облегчает его применение для решения новых задач.

ЗАКЛЮЧЕНИЕ

В данной работе был подробно описан процесс генерации вероятностного автомата из обучающей модели для задач стимулирующего обучения. Также было показано, что предложенный подход, исходя из специфики решаемой задачи, может давать определённые преимущества перед традиционными методами. Проведённые исследования позволяют сделать следующие выводы:

1. Предложенный подход позволяет решать задачи оптимизации для систем, в которых заложены определённые стохастические процессы, при условии, что такая система допускает обучение с помощью поощрений.
2. Непосредственное обучение вероятностного автомата является непрактичным, так как при работе обучающего алгоритма напрямую с распределениями вероятностей теряется информация о количественных характеристиках полезности каждого действия. Поэтому автором был предложен последующий перевод внутренней модели, которая формируется агентом в процессе обучения, в автоматную модель, что является эффективным методом для решения описанных задач.
3. При использовании подхода с генерацией вероятностного управляющего автомата методами стимулирующего обучения необходимо учитывать возможность использования агентом заранее невыгодных действий, которые при обычной схеме использования могут приводить к положительным значениям *Q-функции*. Для того, чтобы этого не случилось, среда в качестве наказания должна возвращать агенту большое отрицательное число за каждое такое действие. В противном случае может пострадать эффективность полученного решения.

4. Существует класс задач, генерация вероятностных автоматов для которых даёт определённые преимущества перед стандартным подходом использования методов стимулирующего обучения, когда применяется жадная стратегия выбора действий.
5. В зависимости от специфики решаемой задачи, предлагается проводить (если это возможно) сжатие выходной автоматной модели. При этом меньшая вероятность посещения данного состояния в процессе работы автомата ещё не является достаточным условием для выбора этого действия в качестве кандидата на удаление.

ИСТОЧНИКИ

1. *Бухараев Р. Г.* Основы теории вероятностных автоматов. М.: Наука., 1985.
2. *Поспелов Д. А*. Вероятностные автоматы. М.: Энергия, 1970.
3. *Vidal E., Thollard F., Colin de la Higuera, Casacuberta F., Carrasco R.* Probabilistic Finite-State Machines. Part I //IEEE Transactions on pattern analysis and machine intelligence. 2005. № 7.
4. *Карпов В. Э.* К вопросу об управлении мобильным роботом в условиях общей постановки задачи //Вестник компьютерных и информационных технологий. 2008. № 1. www.raai.org/about/persons/karpov/pages/umobrob/umobrob.doc
5. *Добрынин Д. А., Карпов В. Э.* Моделирование некоторых форм адаптивного поведения интеллектуальных роботов //Информационные технологии и вычислительные системы. 2006. № 2.
6. *Мосалов О. П, Редько В. Г., Непомнящих В. А.* Модель поискового поведения анимата. www.niisi.ru/iont/projects/rfbr/90197/Preprint.pdf
7. *Ghnemat R.* Agent-based Modeling for Spatial self-organization /Complex Systems – Computer Sciences Implementation Applications to Engineering. 2007.
8. *Mitchel T.* Machine Learning. McGraw-Hill Education (ISE Editions). 1997.
9. *Sutton R., Barto A.* Reinforcement Learning: An Introduction. MIT Press. Cambridge, MA. 1998.
10. *Николенко С. И.* Лекция по стимулирующему обучению. www.logic.pdmi.ras.ru/~sergey/teaching/ml/notes-13-reinforcement.pdf
11. *Выдрин А. С., Михалёв А. В.* Стохастическкие матрицы и анализ защищённости автоматизированных систем. www.mech.math.msu.su/~fpm/ps/k07/k071/k07105.pdf

12. *Richter J.* CLR Via C#: Applied .NET Framework 2.0 Programming. Microsoft Press, 2006.

13. *Troelsen A.* Pro C# 2008 and the .NET 3.5 Platform. APRESS, 2007.