Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Московский физико-технический институт
(государственный университет)»

Факультет управления и прикладной математики

Кафедра информатики

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ЭФФЕКТИВНОГО РЕШЕНИЯ ЗАДАЧИ НАВИГАЦИИ

Выпускная квалификационная работа
(магистерская диссертация)

Направление подготовки: 03.04.01 Прикладные математика и физика

Выполнил:
студент 973а группы _____Борисяк Максим Александрович

Научный руководитель:
к.ф.-м.н., доцент _____ Устюжанин Андрей Евгеньевич

Москва 2015

Содержание

1 Введение

В настоящее время с появлением новых возможностей в робототехнике становятся актуальными задачи искусственного интеллекта. Одними из самых популярных являются задачи связанные с навигацией роботов, которые интересны с практической и теоретической точек зрения и содержат большой спектр подзадач от инженерных до чисто теоретических.

Среди всего спектра можно выделить несколько классов. В первую очередь довольно успешно решаются задачи навигации на короткие расстояния — обход препятствий. Так как в этих задачах размеры препятствий сравнимы с размерами робота, они обычны формулируются в терминах механики и успешно решаются методами теорий оптимального управления и численной оптимизации. Другим похожим классом является навигация на дальние расстояния в известном окружении, которые представляют интерес только в случае непрерывных координат и также успешно решаются методами теории оптимального управления. Среди прикладных систем особенно стоит отметить Robot Operating System[1] представляющую в том числе окружение для осуществление навигации подобного рода.

В контексте данной работы, особое внимание нужно уделить генетическим алгоритмам обхода препятствий, результаты который были продемонстрированы в недавних работах ([2], [3], [4]). Следуя стандартному подходу области искусственного интеллекта к построению агентов, обобщенную задачу навигации (которая в этом случае выглядит так же как и общая задача планирования) можно сформулировать как нахождение оптимальной стратегии (либо близкой к оптимальной):

$$\begin{aligned} \Pi(\theta) &= \arg\max_{\psi \in \mathcal{M}} Q(\theta, \psi) \\ \mathcal{M} &\subseteq \mathcal{F}(\mathcal{O}, \mathcal{A}) \end{aligned}$$

где $\theta \in \Theta$ — описание окружения, $\omega \in \mathcal{O}$ — наблюдение, $a \in \mathcal{A}$ — действие, $Q(\cdot, \cdot)$ — функция качества, $\mathcal{M}$ — некая модель, $\mathcal{F}(X, Y)$ — множество всех отображений из X в Y. Учитывая постановку задачи обхода препятствий информацию об окружении θ можно однозначно восстановить из любого наблюдения ω, поэтому

$$\Pi(\theta) \equiv \psi^*$$

Так же в силу особенностей навигации алгоритм реализующий стратегию ψ^*, который в общем виде имеет вид $\psi[s, \omega] : S_\psi \times \mathcal{O} \mapsto S_\psi \times \mathcal{A}$, где S_ψ — некое пространство состояний алгоритма, без значительных потерь в производительности может быть представлен в виде алгоритма-таблицы: $\psi[\omega] : \mathcal{O} \mapsto \mathcal{A}$. Такие алгоритмические модели с одной стороны устойчивы к малым возмущениям, с другой обычно имеют высокую размерность, что делает простой генетический поиск более вычислительно эффективным по сравнению с классическими методами оптимизации. Однако в случае с известным окружением, такая модель оказывается далеко не самой эффективной, и часто такие задач успешно решаются аналитически, что делает обозначенные выше работы ценными только как демонстрация возможности применение генетический методов.

Условие частично наблюдаемого окружения (обычно, область действия сенсоров много меньше размеров самого окружения) порождает новый класс задач, начиная

[1] Главная страница проекта: `http://ros.org`

от оценки текущего местоположения и построение карты окружения до интеллектуальной прокладки маршрута. Задачи навигации в таких условиях приобретают совершенно иной характер, в силу существенной нелокальности — принятие решение в некий момент в общем случае требует оценки на всю дальнейшую стратегию, что влечет за собой проблему характерную для большинства задач планирования — экспоненциальный рост числа состояний с глубиной планирования. При этом в зависимости от особенностей окружения оптимальные стратегии могут существенно меняться, что при попытке решения в общем случае делает строгие методы нахождения стратегий вычислительно проблематичными для реализации на практике.

В данной работе делает попытка построить метод нахождения оптимальных алгоритмов навигации на дальние дистанции в частично наблюдаемом окружении с помощью генетических алгоритмов поиска, тем самым расширив область исследований заданных в обозначенных работах.

2 Постановка задачи и предварительное исследование

В отличии от большинства работ мы абстрагируемся от многих технических особенностей навигации автономных роботов, таких как: управление моторами, обход малых препятствий, учет геометрии робота, неточность сенсоров и сконцентрируемся на нахождении стратегии поведения, отдавая детали реализации заданного уровня абстракции существующим алгоритмам. Более подробно о методах построения карты, обработки показаний сенсоров и так далее можно узнать, например, из [5], [6], [7], [8], [9], [10].

В качестве окружения робота была выбрана одна из самых простых моделей — двумерная прямоугольная клеточная карта фиксированных размеров, где каждая клетка может быть либо свободной, -1, либо занятой, 1, при этом робот может находиться только в свободных клетках:

$$U = \{-1, 1\}^{W \times H} \setminus \overline{U}; \tag{1}$$

где U — множество всех карт размера $W \times H$ со значениями $-1, 1$ таких, что всегда существует хотя бы один путь между двумя различными свободными точками, требующий не менее одного действия, $\overline{U}$ — соответственно, множество карт, где любая свободная точка изолирована.

Роботу позволено совершать три действия ($\mathcal{A}$): повороты и движение на клетку вперед. Также у робота есть текущее направление из $D = \{d \mid d \in \mathbb{Z}^2, \|d\| = 1\}$ и положение $r \in R = \overline{1, W} \times \overline{1, H}$. Парой $(r, d) \in R \times D$ описывается текущее состояние робота.

$$\mathcal{A} = \{\text{TURNLEFT, FORWARD, TURNRIGHT}\}; \tag{2}$$

$$\varphi : U \times R \times D \times \mathcal{A} \to R \times D; \tag{3}$$

$$\varphi(l, r, d, \text{TURNLEFT}) = (r, \text{TURNLEFT}(d)); \tag{4}$$

$$\varphi(l, r, d, \text{TURNRIGHT}) = (r, \text{TURNRIGHT}(d)); \tag{5}$$

$$\varphi(l, r, d, \text{FORWARD}) = \begin{cases} (r, d), & l(r + d) = 1 \\ (r + d, d), & \text{иначе} \end{cases} \tag{6}$$

где φ — функция «физики» окружения l, $\mathrm{TURNLEFT}(\cdot), \mathrm{TURNRIGHT}(\cdot)$ — функции поворота вектора направления на $\pi, -\pi$ соответственно.

У робота есть только один сенсор — зрение, которое задается функцией v:

$$\begin{aligned} V &= \{-1, 0, 1\}^{W \times H}; & (7) \\ v &: U \times R \times D \to V; & (8) \end{aligned}$$

$$\forall l \in U\ \forall (r, d) \in R \times D\ \forall (x, y) \in R : v(l, r, d)(x, y) \neq 0 \Rightarrow v(l, r, d)(x, y) = l(x, y); \quad (9)$$

Здесь значение 0 в клетке выступает в роли неизвестного участка карты, а свойство (9) означает согласованность зрения с реальным окружением, то есть значение зрения в любой точке карты может быть либо 0, либо соответствовать реальному окружению l.

Мы будем использовать лишь две функции зрения:

$$v_a^r(l, x_0, y_0)(x, y) = \begin{cases} l(x, y), & (x - x_0)^2 + (y - y_0)^2 \leq r^2; \\ 0, & \text{иначе}; \end{cases} \quad (10)$$

$$v_t^r(l, x_0, y_0)(x, y) = \begin{cases} l(x, y), & 1 \notin \mathrm{Line}_l(x_0, y_0, x, y) \wedge (x - x_0)^2 + (y - y_0)^2 \leq r^2; \\ 0, & \text{иначе}; \end{cases} \quad (11)$$

где $\mathrm{Line}_l(x_0, y_0, x, y)$ — 4-х связная прямая между точками (x_0, y_0) и (x, y), не включающая точку (x, y). Как не сложно заметить, функции зрение открывают только области окружения в радиусе r текущей позиции. $r > 0$ будем называть радиусом зрения.

Для удобства включим в множество возможных наблюдений $\mathcal{O}$ текущее положении и конечную точку:

$$\mathcal{O} = R \times D \times V \times R$$

В общем случае, задача состоит в нахождении алгоритма навигации A.

Определение 1. *Алгоритмом навигации A[2] с пространством состояний S_A называется пара (ψ, s_0), такая что:*

$$\begin{aligned} \psi &: S_A \times \mathcal{O} \to S_A \times \mathcal{A}; \\ s_0 &\in S_A; \end{aligned}$$

Определение 2. *Пусть дан алгоритм навигации $A = (\psi, s_0)$ и заданы функция зрения $v(\cdot, \cdot, \cdot)$ и изначально открытая область v_0. Будем говорить, что алгоритм A проецирует путь $\pi = \varphi^*(A, l, r_0, d_0, r_f, v_0)$ в окружении l с открытой областью v_0, с*

[2]Сразу заметим, что рассматриваются только алгоритмы, которые заведомо завершают свою работу, поэтому процедура ψ определяется как отображение.

начальным состоянием (r_0, d_0) *и целевой точкой* r_f*, если:*

$$\begin{aligned}
s^0 &= s_0; \\
(s^t, a^t) &= \psi(\omega^{t-1}, s^{t-1}); \qquad (12)\\
\omega^t &= (r^t, d^t, v^t, r_f); \\
r^0, d^0 &= r_0, d_0; \\
(r^t, d^t) &= \begin{cases} \varphi(l, r^{t-1}, d^{t-1}, a^t), & r^{t-1} \neq r^f \\ (r^{t-1}, d^{t-1}), & \textit{иначе} \end{cases} \qquad (13)\\
v^0 &= v_0; \\
v^t &= v^{t-1} \otimes v(l, r^t, d^t); \qquad (14)\\
\pi^0 &= (r^0, d^0); \\
\pi^t &= (r^t, d^t); \\
t &\geq 1; \\
T &= \begin{cases} \min\left\{t \mid \forall \tau \geq t: \ r^\tau = r^f\right\}, & \exists \tau: \ r^\tau = r^f \\ \infty, & \textit{иначе} \end{cases} \qquad (15)
\end{aligned}$$

При этом T *называется длиной пути* π *или* $|\pi|$*. Аналогично* φ^* *вводятся функции* s^**,* a^**,* ω^**,* r^**,* d^* *возвращающие соответствующие последовательности. Для удобства так же введем функцию* $\Phi^*(A, l, r_0, d_0, r_f, v_0) = \{(s^t, a^t, r^t, d^t, \omega^t)\}_{t=0}^{T}$*. Если изначально открытая область* v_0 *не указана, то* $\pi = \varphi^*(A, l, r_0, d_0, r_f) = \varphi^*(A, l, r_0, d_0, r_f, v(l, r_0, d_0))$*, аналогично для* Φ^**.*

Для удобства дальнейшего рассмотрения, заметим, что алгоритму навигации на вход подается аккумулированная карта окружения ((14)):

$$(v^1 \otimes v^2)_{ij} = v^1_{ij} \otimes v^2_{ij}; \qquad (16)$$

$$x \otimes y = \begin{cases} x, & x = y; \\ x, & y = 0; \\ y, & x = 0; \end{cases} \qquad (17)$$

В силу условия согласованности зрения (9) уравнение (17) описывает все возможные случаи и просто означает добавление информации об исследованных областях. Заметим, что эта процедура не более, чем упрощение, так как в любой алгоритм можно добавить процедуру построение аккумулированной карты окружения.

Определение 3. *Кортеж* $\theta = (l, r_0, d_0, r_f) \in U \times R \times D \times R$ *будем называть задачей навигации в окружении* l *с начальной позицией* (r_0, d_0) *и целью* r_f*, если существует непрерывный путь* $\pi \in (R \times D)^*$ *(а значит и набор команд для реализации этого пути) из позиции* (r_0, d_0) *в точку* r_f*.*

Множество $\Theta \neq \varnothing$ *будем называть набором задач, если:*

$$\Theta \subseteq \{\theta \mid \theta = (l, r_0, d_0, r_f), l \in U, \ r_0, r_f \in R, \ d_0 \in D, \theta \ - \textit{задача навигации} \ \}$$

Определение 4. *Будем говорить, что наблюдение* $\omega \in \mathcal{O}$ *достижимо в задаче* θ*, если*

$$\theta \vdash \omega \Longleftrightarrow \exists a \in \mathcal{A}^*: \ \omega \in \omega^*(\theta, a) \qquad (18)$$

Наблюдение достижимо в наборе задач, если оно достижимо хотя бы для одной задачи:

$$\Theta \vdash \omega \iff \bigvee_{\theta \in \Theta} \theta \vdash \omega \tag{19}$$

Обозначит соответствующие множества как $\Omega(\theta)$ и $\Omega(\Theta)$:

$$\Omega(\theta) = \{\omega \in \mathcal{O} \mid \theta \vdash \omega\} \tag{20}$$

$$\Omega(\Theta) = \bigcup_{\theta \in \Theta} \Omega(\theta) \tag{21}$$

Аналогично определяются $l \vdash \omega$, $(r_0, d_0) \vdash \omega$, $r_f \vdash \omega$:

$$\begin{aligned} l \vdash \omega &\iff \exists \theta = (l, \cdot, \cdot, \cdot) \in \Theta : \theta \vdash \omega \\ (r_0, d_0) \vdash \omega &\iff \exists \theta = (\cdot, r_0, d_0, \cdot) \in \Theta : \theta \vdash \omega \\ r_f \vdash \omega &\iff \exists \theta = (\cdot, \cdot, \cdot, r_f) \in \Theta : \theta \vdash \omega \end{aligned}$$

2.1 Критерии качества

Определение 5. *Пусть задан набор задач Θ. Тогда два алгоритма A_1, A_2 считаются эквивалентными относительно набора Θ — $A_1 \overset{\Theta}{\sim} A_2$, если:*

$$\forall \theta \in \Theta : \ \varphi^*(A_1, \theta) = \varphi^*(A_2, \theta) \tag{22}$$

Если для любого набора Θ $A_1 \overset{\Theta}{\sim} A_2$, то будем говорить, что два алгоритма эквивалентны $A_1 \sim A_2$.

Определение 6. *Пусть задан набор задач Θ и некий критерий качества $Q(\theta, \pi)$. Будем говорить, что алгоритм A_1 доминирует алгоритм A_2 по критерию Q относительно набора Θ, если:*

$$[\forall \theta \in \Theta : \ Q(\theta, \varphi^*(A_1, \theta)) \geq Q(\theta, \varphi^*(A_2, \theta))] \wedge [\exists \theta \in \Theta : \ Q(\theta, \varphi^*(A_1, \theta)) > Q(\theta, \varphi^*(A_2, \theta))] \tag{23}$$

Если из контекста понятно о каких Q и Θ идет речь, будем писать $A_1 \succ A_2$

Перед тем как перейти далее, заметим, что в общем случае не гарантированно, что алгоритм хоть когда-либо достигнет цели в неком окружении. Вместо того, чтобы исследовать возможность алгоритмической проверки на достижении цели, предъявим оценку сверху на количество команд, которое может потребоваться для достижения цели. Для этого рассмотрим алгоритм 1[3].

Теорема 1. *Алгоритм 1 (A^0) решает любую задачу навигации $\theta = (l, r_0, d_0, r_f)$, причем:*

$$|\varphi^*(A^0, \theta)| \leq (8HW + 4)3^{4HW} \tag{24}$$

Доказательство. Для начала заметим, что в любой задаче существует путь из $\pi \in (R \times D)^*$ из (r_0, d_0) в $(r_f, \cdot)$ такой, что все элементы пути различны, то есть путь без самопересечений, так как для из любого пути с самопересечениями отрезав «петли»

[3]Для удобства записи используется оператор **yield** — аналог **return**, но сохраняющий внутреннее состояние функции, в том числе и текущее состояние потока исполнения.

Algorithm 1 Алгоритм верхней границы

```
function GenerateActionSequence(n)
    if n ≤ 0 then
        return []
    end if

    actionSequence = GenerateActionSequence(n − 1)
    result = []

    for all a ∈ A do
        for all s ∈ actionSequence do
            result += [a] + s
        end for
    end for

    return result
end function

allActionSequences = GenerateActionSequence(4WH)

function Act((π = [], n_s = 1, n_a = 1, tostart? = FALSE), ω)
    if tostart? = TRUE AND n_a > 0 then
        if allActionSequences[n_s][n_a] = TURNLEFT then
            return ((π, n_s, n_a − 1, TRUE), TURNRIGHT)
        else if allActionSequences[n_s][n_a] = TURNRIGHT then
            return ((π, n_s, n_a − 1, TRUE), TURNLEFT)
        else
            return ((π, n_s, n_a − 1, TRUE), TURNLEFT)
        end if
    else if n_a ≤ 0 then
        yield TURNRIGHT
        yield TURNRIGHT
        return Act(([], n_s + 1, 1, FALSE), ω)
    end if

    (r, d, v, r_f) = ω
    if (r, d) ∈ π then
        yield TURNLEFT
        yield TURNLEFT
        return Act((π, n_s, n_a, TRUE), ω)
    else
        π′ = [(r, d)] + π
        s′ = (π′, n_s, n_a + 1, FALSE)
        a = allActionSequences[n_s][n_a]
        return (s′, a)
    end if
end function
```

можно получить путь без самопересечений. Так как $|R \times D| = 4HW$, то $|\pi| \leq 4HW$, а значит соответствующая этому пути последовательность команд $|a| \leq 4HW$.

Несложно заметить, что алгоритм 1 просто перебирает все возможные последовательности команд длины $4HW$, возвращаясь на исходную позицию при наличии самопересечения, а значит в какой-либо момент найдет последовательность соответствующую какому-либо пути из начальной точки в целевую. Всего возможных последовательностей команд длины $4HW$: 3^{4HW}. При этом для каждой последовательности алгоритм совершает не более $8HW + 4$ команд (путь в обе стороны, плюс 4 команды для двух разворотов), что доказывает оценку (24). □

Несмотря на то, что алгоритм 1 тривиален, не эффективен ни вычислительно, ни с точки зрения навигации, используя оценку (24) можно частично решить проблему отсечения алгоритмов, которые не всегда находят путь из начальной точки в конечную — так как множество Θ конечно, существует тривиальная алгоритмическая процедура, проверяющая алгоритм на выполнение оценки (24). Тем самым мы фактически накладываем ограничение на рассматриваемые алгоритмы.

Условие частичного наблюдаемого окружение влечет трудности при оценке качества алгоритмов. Понятно, что алгоритма, который находит оптимальный маршрут в любом окружении набора Θ может и не существовать.

Определение 7. *В случае, когда существует алгоритм, находящий кратчайшие маршруты в любом окружении набора Θ, будем называть набор Θ тривиальным.*

Теорема 2. *Если Θ — не тривиальный набор задач, то для любого критерия качества Q, достигающего единственный глобальный максимум на кратчайших маршрутах, не существует доминирующего алгоритма:*

$$\neg \left[\exists \hat{A} : \ \forall A \nsim \hat{A} : \hat{A} \succ A\right] \tag{25}$$

Доказательство. Обозначим кратчайший маршрут в окружении θ как $\hat{\pi}(\theta)$ Предположим существование доминирующего алгоритма $\hat{A}$. Так как Θ не тривиален, то

$$\exists \theta_0 \in \Theta : \ Q(\theta_0, \varphi^*(\hat{A}, \theta_0)) < \max_{\pi} Q(\theta_0, \pi) = Q(\theta_0, \hat{\pi}(\theta_0))$$

Пусть алгоритм A_0 вне зависимости от окружения следует пути $\hat{\pi}(\theta_0)$[4], после чего, если цель не достигнута, следует, например, правилу левой руки. Но тогда:

$$Q(\theta_0, \varphi^*(A_0, \theta_0)) = Q(\theta_0, \hat{\pi}(\theta_0)) = \max_{\pi} Q(\theta_0, \pi) > Q(\theta_0, \varphi^*(\hat{A}, \theta_0))$$

что противоречит определению доминирующего алгоритма. □

Понятно, что тривиальный наборы Θ редко встречаются в практических задачах и для наших целей не представляют интереса. Теорема 2 фактически означает, что в этом случае измерение качества алгоритмов должно опираться на весь набора задач Θ. Самый простой класс однозначных критериев качества — взвешенная сумма по всем окружениям набора Θ :

$$\hat{Q}(\Theta, A) = \sum_{\theta \in \Theta} c(\theta) Q(\theta, \varphi^*(A, \theta)) \tag{26}$$

[4]Это можно реализовать, например, положив $S_{A_0} = \mathbb{N}$, $s^{t+1} = s^t + 1$, $s_0 = 1$, каждый раз выдавая по номеру команду соответствующую пути $\hat{\pi}(\theta_0)$, который в силу конечности может быть просто частью алгоритма.

которому в силу конечности Θ всегда можно придать вероятностную интерпретацию. Пусть задано вероятностное пространство над Θ с вероятностной мерой $\mathbb{P}(\theta)$:

$$\hat{Q}(\Theta, A) = \mathbb{E}_\theta\, Q(\theta, \varphi^*(A, \theta)) \tag{27}$$

В данной работе будет использоваться два критерия качества:

$$I_r(\Theta, A) = \mathbb{E}_\theta\, Q_a(\theta, \varphi^*(A, \theta)) = -\mathbb{E}_\theta\, \frac{|\varphi^*(A, \theta)|}{|\hat{\pi}(\theta)|} \to \max \tag{28}$$

$$I_a(\Theta, A) = \mathbb{E}_\theta\, Q_r(\theta, \varphi^*(A, \theta)) = -\mathbb{E}_\theta\, |\varphi^*(A, \theta)| \to \max \tag{29}$$

Другим важным классом критериев являются максиминные критерии, например:

$$I_r^m(\Theta, A) = \min_{\theta\in\Theta} Q_a(\theta, \varphi^*(A, \theta)) = \min_\theta\, -\frac{|\varphi^*(A, \theta)|}{|\hat{\pi}(\theta)|} \to \max \tag{30}$$

$$I_a^m(\Theta, A) = \min_{\theta\in\Theta} Q_r(\theta, \varphi^*(A, \theta)) = \min_\theta\, -|\varphi^*(A, \theta)| \to \max \tag{31}$$

$$\tag{32}$$

2.2 Алгоритмы навигации

Заметим, что в общем случае поведение алгоритма в момент t существенно зависит от его состояния в этот момент времени, которое не всегда можно восстановить по наблюдению ω^{t-1}.

Определение 8. *Пусть задано вероятностное пространство $(\Theta, 2^\Theta, \mathbb{P})$, где Θ — набор задач. Алгоритм A существенно зависит от внутреннего состояния, если:*

$$\exists \theta_1 = (\cdot, r_0, d_0, \cdot),\ \theta_2 = (\cdot, r_0, d_0, \cdot) \in \Theta, \mathbb{P}(\theta_1)\mathbb{P}(\theta_2) > 0 :\ \exists t_1, t_2 :\ \omega_{\theta_1}^{t_1} = \omega_{\theta_2}^{t_2} \wedge a_{\theta_1}^{t_1} \neq a_{\theta_2}^{t_2} \tag{33}$$

Иными словами алгоритм существенно зависит от своего внутреннего состояния, если решение алгоритма зависит не только от текущего наблюдения и положения. Несложно заметить, что алгоритмы не зависящие существенно от внутреннего состояния эквивалентны отображениям $R \times D \times \mathcal{O} \mapsto \mathcal{A}$.

Теорема 3. *Пусть задано вероятностное пространство $(\Theta, 2^\Theta, \mathbb{P})$, где Θ — набор задач. Тогда для любого существенно зависимого от внутреннего состояния алгоритма A существует не зависимый существенно от внутреннего состояния алгоритм A' доминирующий A по некому критерию $Q(\theta, A)$, $Q(\theta, A) = Q'(\theta, |\varphi^*(\theta, A)|)$, где функция $Q'(\theta, \cdot)$ строго убывает по второму аргументу при каждом фиксированном θ.*

Доказательство. Так как в силу ограничений алгоритм A решает каждую задачу из Θ, то можно построить следующее отображение, сопоставляющее каждой тройке (r_0, d_0, ω) внутренние состояния алгоритма в момент наблюдения ω в задаче с (r_0, d_0):

$$\begin{aligned} H &: R \times D \times \mathcal{O} \to S_A^*; \\ H(r_0, d_0, \omega) &= \{s \mid \exists \theta \in \Theta :\ (s, \cdot, \cdot, \cdot, \omega) \in \Phi^*(\theta, A) \wedge (\cdot, r_0, d_0, \cdot) = \theta\} \end{aligned} \tag{34}$$

Заметим, что одни и те же наблюдения могут соответствовать различным окружениям.

На основе этого отображения построим новый алгоритм A' не зависимый существенно от внутреннего состояния. Так как $\omega^0 = (r_0, d_0, \cdot, \cdot)$, то есть доступны алгоритму изначально, для удобства включим (r_0, d_0) в состояние алгоритма — $S_{A'} = R \times D$.

Введем множество $\Theta(r_0, d_0, \omega)$:

$$\Theta(r_0, d_0, \omega) = \{\theta \in \Theta \mid \theta = (\cdot, r_0, d_0, \cdot) \wedge \theta \vdash \omega\} \tag{35}$$

Так как количество возможных наблюдений конечно, A' представим в виде отображения $S_{A'} \times \mathcal{O} \mapsto \mathcal{A}$:

$$\psi_{A'}((r_0, d_0), \omega) = ((r_0, d_0), \psi'(r_0, d_0, \omega)) \tag{36}$$

$$\psi'(r_0, d_0, \omega) = \psi(s'(r_0, d_0, \omega), \omega) \tag{37}$$

$$s'(r_0, d_0, \omega) = \underset{s \in H(r_0, d_0, \omega)}{\arg\min} \min_{\theta \in \Theta(r_0, d_0, \omega)} |\varphi^* ((\psi, s), l, r, d, r_f, v)| \tag{38}$$

Докажем, что алгоритм A' не хуже A. Заметим, что в силу свойств (9) и (14) по наблюдению ω и начальной позиции (r_0, d_0) для заданного алгоритма можно единственным образом восстановить последовательность состояний от начала до последнего получения этого наблюдения, которое всегда существует, так как алгоритм завершает любую задачу из набора. Для этого достаточно взять любое $\theta \in \Theta(r_0, d_0, \omega)$ и взять начало $\Phi^*(A, \theta)$. Рассмотрим случай $H(r_0, d_0, \omega) = \{s_1, s_2, \ldots, s_T\}$, $T \geq 2$. Пусть состояния упорядочены по номеру шага, на котором они появляются. В силу свойства (14) между первым s_1 и последним s_f состояниями, алгоритм не получал зрение отличное от $v, \omega = (r, d, v, r_f)$, а значит просто сделал петлю в известной области.

$$\forall \theta = (l, r_0, d_0, r_f) \in \Theta(r_0, d_0, \omega) : \ |\varphi^*((\psi, s_f), l, r, d, r_f, v)| < |\varphi^*((\psi, s_1), l, r, d, r_f, v)| \tag{39}$$

а значит s_f будет минимизировать (38). Обозначим путь проделанный алгоритмом до первого наблюдения ω как π. Так как (39) верно для всех возможных при (r_0, d_0) и ω задачах, то:

$$\begin{aligned}
&\forall \theta = (l, r_0, d_0, r_f) \in \Theta(r_0, d_0, \omega) : \\
&\quad Q(\theta, A') = Q'(\theta, |\pi| + |\varphi^*((\psi, s_f), l, r, d, r_f, v)|) > \\
&\qquad Q'(\theta, |\pi| + |\varphi^*((\psi, s_1), l, r, d, r_f, v)|) = Q(\theta, A)
\end{aligned}$$

Так как это верно для любых возможных ω и (r_0, d_0), для которых $|H(r_0, d_0, \omega)| \geq 2$, f по определению существует хотя бы одна такая пара, и так как для $|H(r_0, d_0, \omega)| = 1$ алгоритмы принимают одинаковые решения, то:

$$\begin{aligned}
&\forall \theta \in \Theta : \ Q(\theta, A') \geq Q(\theta, A) \\
&\exists \theta \in \Theta : \ Q(\theta, A') > Q(\theta, A)
\end{aligned}$$

что доказывает $A' \succ A$.

Так как A' по построению не зависим существенно от внутреннего состояние, теорема доказана. □

Теорема 3 имеет два важных следствия. Во-первых, при поиске (или аппроксимации) оптимального алгоритма навигации для любого набора задач с функцией качества, удовлетворяющей ограничениям теоремы (которые крайне слабы с практической точки зрения), можно ограничиться классом алгоритмов эквивалентным отображениям $R \times D \times \mathcal{O} \mapsto \mathcal{A}$. Во-вторых, это позволяет улучшить оценку (24).

Теорема 4. *Пусть задан алгоритм-отображение* $A : R \times D \times \mathcal{O} \mapsto \mathcal{A}$ *и задача* $\theta = (l, r_0, d_0, r_f)$. *Алгоритм* A *либо не решает задачу* θ, *либо:*

$$|\varphi(\theta, A)| \leq HW(4HW + 1) \tag{40}$$

Доказательство. Так как r_0, d_0 фиксированы, можно рассматривать отображение $A'(\omega) = A(r_0, d_0, \omega)$. Заметим, что если путь алгоритма содержит цикл, то алгоритм не решает задачу. Для фиксированного зрения $v, \omega = (r, d, v, r_f)$ существует не более $4HW$ различных позиций (r, d). Так как алгоритм не содержит цикл, то он не встречает два одинаковых наблюдения, значит для фиксированного зрения он может совершить не более $4HW - 1$ действий. Учитывая свойства функций зрения, после $4HW$ действий к v гарантированно добавиться как минимум одна известная точка. Всего точек не более HW. Так как путь алгоритма не содержит цикл, то среди этих точек будет r_f, что доказывает теорему. □

Теорема 5. *Пусть задано вероятностное пространство* $(\Theta, 2^\Theta, \mathbb{P})$, *где* Θ *— набор задач. Предположим, что окружение* l, *стартовая позиция* (r_0, d_0) *и целевая точка* r_f *независимы в совокупности как случайные величины. Тогда для любого существенно зависимого от внутреннего состояния алгоритма* A *существует не зависимый существенно от внутреннего состояния алгоритм* A', *такой что:*

$$I_a(\Theta, A) \leq I_a(\Theta, A') \tag{41}$$

где I_a *– критерий качества, заданный уравнением* (29).

Доказательство. Для доказательства для алгоритма A для всех задач из Θ сопоставим каждому возможному наблюдению состояния в момент получения этого наблюдения:

$$\begin{aligned} H &: \mathcal{O} \to S_A^*; \\ H(\omega) &= \{s \mid \exists \theta \in \Theta : \ (s, \cdot, \cdot, \cdot, \omega) \in \Phi^*(\theta, A)\} \end{aligned} \tag{42}$$

Заметим, что одни и те же наблюдения могут соответствовать различным окружениям.

На основе этого отображения построим новый алгоритм A' с $S_{A'} = \{s_0\}$ и $\psi'(s, \omega) = \psi(\omega)$. Так как количество возможных наблюдений конечно, A' представим в виде отображения $\mathcal{O} \mapsto \mathcal{A}$. Зафиксируем $\omega = (r, d, v, r_f) \in \Omega(\Theta)$. Если $H(\omega) = \varnothing$, то полагаем $\psi'(\omega)$ равным произвольной команде. Заметим, что $\omega \notin \Omega(\Theta) \Rightarrow H(\omega) = \varnothing$. Иначе, если $\omega \in \Omega(\Theta)$, то существует апостериорное распределение $\mathbb{P}(l \mid \omega)$, определяемое по формуле Байеса. В силу независимости l, (r_0, d_0) и r_f как случайных величин:

$$\mathbb{P}(\omega \mid l) = \mathbb{I}[l \vdash \omega] \sum_{\theta \in \Theta} \mathbb{P}(\theta) \mathbb{I}[\theta \vdash \omega] \mathbb{I}[\theta = (l, \cdot, \cdot, r_f)] = \mathbb{I}[l \vdash \omega] C_\omega$$

где

$$\mathbb{I}[t] = \begin{cases} 1, & \text{если } t; \\ 0, & \text{иначе} \end{cases}$$

Обозначим множество окружений из Θ как L_θ. Тогда:

$$\mathbb{P}(l \mid \omega) = \frac{\mathbb{P}(l) \mathbb{I}[l \vdash \omega] C_\omega}{\sum\limits_{l \in L_\Theta} \mathbb{P}(l) \mathbb{I}[l \vdash \omega] C_\omega} = \frac{\mathbb{P}(l) \mathbb{I}[l \vdash \omega]}{\sum\limits_{l \in L_\Theta} \mathbb{P}(l) \mathbb{I}[l \vdash \omega]} \tag{43}$$

Сопоставим каждому наблюдение из $\Omega_A(\Theta) = \{\omega \mid H(\omega) \neq \varnothing\}$ состояние $\hat{s}_\omega$:

$$\hat{s}_\omega = \arg\min_{s \in H(\omega)} \sum_{l \in L_\Theta} \mathbb{P}(l \mid \omega) \cdot |\varphi^* ((\psi, s), l, r, d, r_f, v)| \tag{44}$$

Тогда:

$$(\cdot, \psi'(\omega)) = \psi(\hat{s}_\omega, \omega)$$

Алгоритм A' не является существенно зависимым от внутреннего состояния по построению.

Докажем, что A' не хуже алгоритма A по критерию I.

Рассмотрим качество алгоритма A. Заметим, что для заданной задачи существует только единственная последовательность наблюдений и состояний. В силу свойства (9) текущее наблюдение $\omega = (r, d, r_f, v)$ и начальная позиция (r_0, d_0) полностью определяют апостериорное распределение θ:

$$\mathbb{P}(\theta \mid \omega^*) = \mathbb{P}(\theta \mid (r_0, d_0), \omega) \tag{45}$$

где $\omega^* \in \mathcal{O}^*$ — последовательность наблюдений алгоритма. Рассмотрим промежуточное состояние: пусть алгоритма проделал путь π, наблюдая последовательность ω^* с первым наблюдением $\omega_0 = (r_0, d_0, v_0, r_f)$ и последним $\omega = (r, d, v, r_f)$. Рассмотрим «апостериорное» качество:

$$\begin{aligned}
-I_a(A, \Theta \mid \omega^*) &= -\mathbb{E}_\theta \left[Q_a(\theta, [\pi; \varphi^*(A, l, r, d, r_f, v)]) \mid (r_0, d_0), \omega \right] && (46)\\
&= \mathbb{E}_\theta \left[|\varphi^*(A, l, r, d, r_f, v)| \mid (r_0, d_0), \omega \right] + \mathbb{E}_\theta \left[|\pi| \mid (r_0, d_0), \omega \right] && (47)\\
&= \mathbb{E}_\theta \left[|\varphi^*(A, l, r, d, r_f, v)| \mid (r_0, d_0), \omega \right] - I_\pi && (48)\\
&= \sum_{\theta \in \Theta} \mathbb{P}(\theta \mid (r_0, d_0), \omega) |\varphi^*(A, l, r, d, r_f, v)| - I_\pi && (49)\\
&= -I_\omega - I_\pi && (50)
\end{aligned}$$

Заметим, что в силу независимости l и (r_0, d_0) как случайных величин:

$$\begin{aligned}
\mathbb{P}(\theta \mid (r_0, d_0), \omega) &= \mathbb{P}((l, r'_0, d'_0, r_f) \mid (r_0, d_0), \omega)\\
&= \mathbb{P}(l \mid (r_0, d_0), \omega) \, \mathbb{P}((r'_0, d'_0) \mid (r_0, d_0), \omega) \mathbb{I}[\theta \vdash \omega]\\
&= \mathbb{P}(l \mid \omega) \mathbb{I}[(r'_0, d'_0) = (r_0, d_0)] \mathbb{I}[\theta \vdash \omega]
\end{aligned}$$

Тогда продолжая уравнение (50):

$$\begin{aligned}
-I_\omega &= \sum_{\theta \in \Theta} \mathbb{P}(\theta \mid (r_0, d_0), \omega) \cdot |\varphi^*(A, l, r, d, r_f, v)\\
&= \sum_{\theta \in \Theta} \mathbb{P}(l \mid \omega) \cdot \mathbb{I}[(r'_0, d'_0) = (r_0, d_0)] \cdot \mathbb{I}[\theta \vdash \omega] \cdot |\varphi^*(A, l, r, d, r_f, v)|\\
&= C_\omega \sum_{l \in L_\Theta} \mathbb{P}(l \mid \omega) \cdot |\varphi^*(A, l, r, d, r_f, v)|\\
&\leq C_\omega \sum_{l \in L_\Theta} \mathbb{P}(l \mid \omega) \cdot |\varphi^*((\psi, \hat{s}_\omega) l, r, d, r_f, v)| && (51)
\end{aligned}$$

по построению $\hat{s}_\omega$.

Так как (51) верно для любого наблюдения и начальной позиции:

$$I_a(A, \Theta) \leq I_a(A', \Theta)$$

что доказывает уравнение (41) и теорему. □

Стоит обратить внимание, что доказательство теоремы 5 неявно использует следующее свойство критерия I_a.

Определение 9. *Пусть алгоритм проделал путь π, наблюдая последовательность ω^*, при этом отдавая команды $a_p \in \mathcal{A}$. Пусть $\Lambda = \mathcal{A}^{R\times D\times \mathcal{O}}$ — множество отображений (фактически, множество классов эквивалентности алгоритмов). Для $\lambda \in \Lambda$ можно построить составной алгоритм: $A = [a_p; \lambda]$, который в начале исполняет последовательность a_p, затем следует отображению λ. Тогда критерий I называется потенциальным, если для набора задач Θ, для любого π возможного в наборе задач Θ и соответствующих a_p и ω^* существует функция $U_\theta((r_0, d_0), \omega)$:*

$$\begin{aligned}
\hat{a} &= \operatorname*{Arg\,max}_{a\in\mathcal{A}} \max_{\lambda\in\Lambda} I([a_p, a, \lambda], \Theta \mid \pi, \omega^*) \\
&= \operatorname*{Arg\,max}_{a\in\mathcal{A}} \max_{\lambda\in\Lambda} I([a_p, a, \lambda], \Theta \mid (r_0, d_0), \omega) \\
&= \operatorname*{Arg\,min}_{a\in\mathcal{A}} \mathbb{E}_\theta \left[U((r_0, d_0), \Phi_\omega(\theta, \omega, a)) \mid (r_0, d_0), \omega\right]
\end{aligned}$$

где $\Phi_\omega(\theta, \omega, a)$ — наблюдение, которое будет получено после выполнения команды a в задаче θ при текущем наблюдении ω, следуя определению 2.

Потенциальность критерия, фактически означает, что оптимальное действие в какомлибо состоянии не зависит от способа достижения этого состояния. В данном случае состояние задается парой $((r_0, d_0), \omega)$.

Очевидно, что для потенциалов I_a и I_r (уравнения (29) и (28)):

$$\begin{aligned}
U_a((r_0, d_0), \omega) &= \mathbb{E}_{\theta\in\Theta} \left[\rho_\theta(r, d) | (r_0, d_0), \omega\right] \\
U_r((r_0, d_0), \omega) &= \mathbb{E}_{\theta\in\Theta} \left[\frac{\rho_\theta(r, d)}{\rho_\theta(r_0, d_0)} \middle| (r_0, d_0), \omega\right]
\end{aligned}$$

где $\rho_\theta(r, d)$ — длина минимального пути из позиции (r, d) в точку r_f в задаче $\theta = (l, r_0, d_0, r_f)$ ($\rho_\theta(r_0, d_0) = |\hat{\pi}(\theta)|$).

Используя потенциал, довольно просто прийти к оптимальной стратегии, пример реализации которой приведен в алгоритме 2.

2.3 Частично наблюдаемый марковский процесс принятия решений

Потенциальность критериев качества позволяет сформулировать задачу в терминологии частично наблюдаемого марковского процесса принятия решения (partially observed Markov decision process, POMDP)[11].

Algorithm 2 Оптимальный алгоритм навигации

Require: Θ — набор задач, $(\Theta, 2^{\Theta}, \mathbb{P})$ — вероятностное пространство, U — потенциал.
Ensure: Алгоритм $(\text{Act}, ((0,0), +\infty, \text{FALSE}))$ — оптимальный алгоритм для критерия, соответствующего потенциалу U.

function ACT($((r_0, d_0) = (0,0)$, $u = +\infty$, initialSet? = FALSE), ω)
  if NOT initialSet? **then**
    $(r, d, v, r_f) := \omega$
    $(r_0, d_0) := (r, f)$
    $u := U((r_0, d_0), \omega)$
  end if

  $\hat{a} :=$ NULL
  $u_a := +\infty$
  for $a \in \mathcal{A}$ **do**
    $u' := 0$
    for $\theta \in \Theta$ **do**
      $u' := u' + \Phi_\omega(\theta, \omega, a) \cdot \mathbb{P}(\theta \mid (r_0, d_0), \omega)$
    end for
    if $u' < u_a$ **then**
      $u_a := u'$
      $\hat{a} := a$
    end if
  end for

  return $(((r_0, d_0), u_a, \text{TRUE}), \hat{a})$
end function

Определение 10. *Марковский процесс принятия решения задается набором* $(\mathcal{S}, \mathcal{A}, T, G)$, *где* $\mathcal{S}$ *— конечное множество состояний системы, которые может различить агент,* $\mathcal{A}$ *— конечное множество возможных действий,* $T(s, a, s')$ *— функция перехода из состояния* s *при действии* a *агента, задающая дискретное распределение вероятностей на* $\mathcal{S}$, $G(s, a)$ *— функция награды при совершении действия* a *в состоянии* s.

Для заданной стратегии агента $\pi : \mathcal{S} \mapsto \mathcal{A}$ *определяются соответствующие последовательности* s_t, a_t.

Определение 11. *Частично наблюдаемый марковский процесс принятия решения задается набором* $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, G, O)$, *где* $\mathcal{S}$, $\mathcal{A}$, T, G *задаются так же, как и в определении 10,* $\mathcal{O}$ *— конечное множество наблюдений,* $O(s, \omega)$ *— функция, задающая распределение наблюдений в состоянии* s *системы.*

Для заданной стратегии агента $\pi : \mathcal{O} \mapsto \mathcal{A}$ *определяются соответствующие последовательности* s_t, ω_t a_t.

Определим задачу навигации в терминах POMDP.

Определение 12. *Обобщенной задачей навигации будет называть частично наблюдаемый марковский процесс* $(\mathcal{S}, \mathcal{A}, \mathcal{O}, T, G, O)$, *где* $\mathcal{A}$ *и* $\mathcal{O}$ *соответствуют введенным ранее множествам доступный действий и наблюдений,* $\mathcal{S} = R \times D \times \Xi(\Theta)$, *где* $\Xi(\Theta)$ *— множество возможных суперпозиций задач из* Θ, *а функции* T, G *и задаются следующим образом:*

$$
\begin{aligned}
s &= (r, d, \mathbb{P}_s) \\
T(s, a, s') &= s' = (r', d', \mathbb{P}_{s'}) \\
(r', d') &= \varphi(\omega, r, d) \\
\mathbb{P}_{s'}(\theta) &= \frac{\mathbb{P}_s(\theta)\mathbb{I}[\theta \vdash \Phi_\omega(\theta, r, d, a)]}{\sum_{\vartheta \in \Theta} \mathbb{P}_s(\vartheta)\mathbb{I}[\vartheta \vdash \Phi_\omega(\vartheta, r, d, a)]}
\end{aligned}
$$

Функция $G(s, a)$ *задается исходя из потенциала критерия качества.*

Заметим, что $\Xi(\Theta)$ — конечное множество как требуется в определениях 10 и 11, так как множество возможных наблюдений всегда конечно $|\mathcal{O}| \leq 3^{H \times W}$, а каждая суперпозиция определяется через апостериорное распределение:

$$\mathbb{P}_s(\theta) = \mathbb{P}(\theta \mid \omega)$$

Поэтому, фактически, можно рассматривать множество состояний $\mathcal{S} = \mathcal{O}$.

Наиболее популярным алгоритмом решения POMDP в общем случае является policy iteration[12] — алгоритм динамического программирования, поледовательно находящий приближения потенциала для каждого наблюдения.

Алгоритмы нахождения оптимальных стратегий для POMDP крайне популярны для решения задач навигации в неизвестных окружениях (см. например, [13], [14], [15]). Так как мы переформулировали задачу навигации через POMDP, можно рассматривать данную работу, как иной подход к решению POMDP в случае конкретной задачи. Сразу следует отметить, что при определении модели для поиска, мы абстрагируемся от многих специфических для навигации аспектов, поэтому построенные алгоритмы также можно рассматривать как демонстрацию решения POMDP с помощью генетических алгоритмов в общем случае.

2.4 Дополнительные предположения

Легко понять, что с практической точки зрения рассматривать произвольный набор задач Θ не эффективно. В реальности большинство окружений, с которыми может столкнуться автономный робот имеют определенную структуры, часто влияющую на эффективность навигации. Ярким примером являются окружения созданные человеком. Например, ширина автомобильных дорог часто определяет «ранг» дороги — оптимальный маршрут в дали от цели вероятнее пролегает по главным дорогам, однако при приближении к цели, часто оптимальнее использовать «локальную» дорожную систему, которая обычно уже. Другим примером является помещения — большая часть маршрута, скорее всего будет пролегать по коридорам.

Говоря о наличии структуры или особенностей в наборе задач Θ, мы будет сравнивать его относительно некого универсального набора задач, который постулируется как абсолютно бесструктурный. В качестве такого набора логично взять набор с максимальной энтропией.

Определение 13. *Пусть заданы набор задач Θ с вероятностной мерой $\mathbb{P}$ и критерий качества $I(\Theta, A)$. Будем говорить, что набор задач Θ обладает структурой, если:*

$$I(\Theta, \hat{A}) > I(\Theta, \hat{A}_0) + \epsilon$$

где ϵ — некая заданная константа, $\hat{A}$ — оптимальный алгоритм (например, алгоритм 2) для набора задач Θ с вероятностной мерой $\mathbb{P}$, $\hat{A}_0$ — оптимальный алгоритм для набора задач U (уравнение (1)) с вероятностной мерой $\mathbb{P}_0(\theta) \equiv \text{const}$.

Наборы задач в данной работе подбирались, исходя в первую очередь из практических соображений. К сожалению, строго проверить наличие структуры в наборе задач, следуя определению, на практике невозможно в силу чрезвычайно высокой алгоритмической сложности процедуры нахождения оптимального алгоритма. Изначально наборы задач были выбраны исходя из интуитивных и практических соображений, а проверка на наличии структуры производилась постфактум сравнением качеств построенного алгоритма и $\hat{A}_0$, либо качества $\hat{A}_0$ с максимумом для критерия ($\max_{\Theta,A} I_r(\Theta, A) = -1$).

В дополнение к наличию структуры, вводится интуитивно понятное предположение о том, что начальные и конечные точки как случайные величины не несут никакой информации об окружении (см. условия теоремы 5).

2.5 Наивный алгоритм

Фактически, алгоритм $\hat{A}_0$ (в данной работе он назван «наивным» в силу простоты предположений на вероятностную меру для набора задач) в свете определения 13 выступает в роли базового уровня. Так как сам алгоритм будет использоваться в дальнейшем, предоставим пример его реализации (алгоритм 3).

Данная реализация опирается на тот факт, что при равномерном распределении задач на U, апостериорное распределение вероятностей для каждой точки вне области зрения $\mathbb{P}(l(x, y) = -1 \mid (r_0, d_0, \omega),\ v(x, y) = 0) = \text{const}$, поэтому алгоритм 3 осуществляет команду, соответствующую кратчайшему пути по точка $v(x, y) \neq 1$.

[4]На практике для поиска кратчайшего пути эффективнее использовать, например, алгоритм A* вместо приведенного поиска в ширину. Однако в дальнейшем мы будем использовать потенциал, вычисленный поиском в ширину.

Algorithm 3 Наивный алгоритм навигации

```
1: function WAVE(v, open, closed, u)
2:     if open = ∅ then
3:         return u
4:     end if

5:     open′ := ∅

6:     for (r, d) ∈ open do
7:         states := {(r − d, d), (r, TURNLEFT(d)), (r, TURNRIGHT(d))}
8:         for (r′, d′) ∈ states do
9:             if (r′, d′) ∉ closed ∧ v[r′] ≠ 1 ∧ u[d′][r′] > u[d][r] + 1 then
10:                u[d′][r′] := u[d][r] + 1
11:                open′ := open′ ∪ {(r′, d′)}
12:            end if
13:        end for
14:    end for

15:    return Wave(v, open′, open ∪ closed, u, v)
16: end function

17: function POTENTIAL(v, r_f)
18:    u := []
19:    for (r, d) ∈ R × D do
20:        u[d][r] := +∞
21:    end for
22:    for d ∈ D do
23:        u[d][r_f] := 0
24:    end for

25:    return Wave({(r_f, d) | d ∈ D}, ∅, u, v)
26: end function

27: function ACT(ω)
28:    (r, d, v, r_f) := ω
29:    u := Potential(v, r_f)
30:    â := NULL
31:    û = +∞
32:    for a ∈ A do
33:        (r′, d′) := φ(v, r, d, a)
34:        if û′ > u[d′][r′] then
35:            û := u[d′][r′]
36:            â := a
37:        end if
38:    end for

39:    return â
40: end function
```

2.6 Мотивация

Можно заметить, что для набора задач $\Theta \sim O(2^{H \times W})$ мощность прямой модели M_0 алгоритма навигации, то есть отображения $A : R \times D \times \mathcal{O} \mapsto \mathcal{A}$, колоссальна с практической точки зрения уже при $H, W = 4$ ($|\mathcal{A}^{R \times D \times \mathcal{O}}| \sim 7 \cdot 10^{245}$) и растет быстрее, чем экспоненциально:

$$\max |M_0| = 4 \cdot (H \times W) \cdot 3^{2^{H \times W}}$$

Рассмотренный алгоритм policy iteration на каждой итерации для каждого наблюдения оперирует с распределением над возможными наблюдениями, чья мощность достигает $\sim 2^{|\Theta|}$ для начальных итераций. [16] Для решения POMDP разработаны многие эвристические методы (см., например, [17]), например, итерационный алгоритм, рассматривающий состояние системы как $\mathcal{S} = \mathcal{S}' \cup \bigcup_i \{s_i\}$, где $\mathcal{S}'$ рассматривается как единое состояние, а улучшение оценки происходит за счет добавление состояний из $\mathcal{S}'$ для явного рассмотрения; позволяющие балансировать между вычислительной сложностью и качеством. Однако часто для успешного практического применения используются малые по мощности модели POMDP, построенные за счет дополнительной предобработки (см., например, [18], [19], [15], [13]).

Несмотря на практический успех применения POMDP в навигации автономных роботов, автором было принято решение рассмотреть другой подход, основанный на применении генетических алгоритмов, показавший себя перспективным на задачах обхода препятствий (см. [2], [3], [4]). Решение было продиктовано известным свойством генетических алгоритмов — лучшая по сравнению со многими классическими оптимизационными подходами устойчивость к высоким размерностям. Например, популярно применение классического генетического поиска для отбора признаков в задачах машинного обучения — в этом случае мощность модели $|M| = |2^{\mathcal{F}}| = 2^{|\mathcal{F}|}$, где $\mathcal{F}$ — множество признаков, что в условиях высокой вычислительной стоимости функции качества (которая в этом случае включает обучение используемого метода, для которого происходит отбор признаков) дает преимущество по сравнению с методами дискретной оптимизации.

Другим важным преимуществом генетического подхода является слабые начальные предположения и соответственно общность алгоритмов, что открывает возможности применять обширные модификации, приспосабливая метод для конкретной задачи. Именно этому свойству обязаны положительные результаты эксперимента.

3 Генетические методы навигации

3.1 Модель

Теорема 3 обосновывает возможность поиска оптимального алгоритма (и его приближений) в классе эквивалентном отображениям $R \times D \times \mathcal{O} \mapsto \mathcal{A}$, а при предположениях о независимости начальной, конечной точек и окружения, теорема 5 сужает класс до $\mathcal{A}^{\mathcal{O}}$.

Понятно, что производить поиск напрямую в этом классе крайне затратно, поэтому, следуя аналогии с работами [3], [2], используется подмножество отображений, задаваемое набором шаблонов $p : [-1, 1]^{\mathcal{O}}$, задающие некое условие на наблюдение. Также можно рассматривать эти шаблоны как предикаты нечеткой логики ([20], [4]). Итоговое отображение A задается набором $G \subset \{(p, a) \mid p : \mathcal{O} \mapsto [-1, 1], a \in \mathcal{A}\}$ на основе

схемы наилучшего соответствия:

$$(\cdot, A(\omega)) = \arg\max_{(p,a)\in G} p(\omega) \tag{52}$$

либо схемы голосования:

$$A(\omega) = \arg\max_{\hat{a}\in\mathcal{A}} \frac{\sum_{(p,a)\in G} \mathbb{I}[a = \hat{a} \wedge p(\omega) > 0] p(\omega)}{\sum_{(p,a)\in G} \mathbb{I}[a = \hat{a} \wedge p(\omega) > 0]} \tag{53}$$

Следуя терминологии генетических методов, мы будем называть множество G — геномом. В качестве базовой модели для шаблона зрения используется следующая функция похожести наблюдений:

$$p((r, d, v, r_f)) = p_\nu((r, d, v, r_f)) = \nu \otimes c_{r,d}(v) \tag{54}$$

где $c_{r,d}(v)$ — преобразование центрирования по r и поворота к d наблюдения v, ν — некий шаблон зрения.

Следуя стандартному подходу ν следует задавать в как $\nu \in V$, то есть в виде дискретных значений. В работе [21] показаны преимущества распространения значений шаблонов на непрерывное множество. Мы последуем этому подходу и определим шаблон зрения как матрицу со значениями в отрезке $[-1, 1]$:

$$\nu \in \left\{ [-1, 1]^{h\times w} \mid h, w \leq \max(H, W) \right\} \tag{55}$$

$$\nu \otimes v = \frac{\sum_{x,y} \nu_{x,y} v_{x,y}}{\sum_{x,y} |\nu_{x,y}|} \tag{56}$$

где суммирование ведется по всем возможным в обеих матрицах индексам, считая отсутствующие значения равными нулю. Иными словами шаблон p задается нормированной суммой элементов поэлементного произведения фиксированного для p шаблона зрения ν и выровненного зрения v.

Легко заметить, что шаблон зрения можно представить в виде разложения:

$$\nu = v \circ c \tag{57}$$

$$v \in \{-1, 1\}^{h\times w} \tag{58}$$

$$c \in [0, 1]^{h\times w} \tag{59}$$

где $\circ$ — оператор поэлементного умножения, v — дискретный шаблон окружения, а матрицу c можно рассматривать как коэффициенты вклада каждой точки.

Определение 14. *Пусть дан вектор предикатов нечеткой логики* $\mathbf{p} \in \left([-1, 1]^{\mathcal{O}}\right)^N$ *и вектор коэффициентов* $\mathbf{c} \in \mathbb{R}^N$. *Тогда определим оператор нечеткой конъюнкции предикатов* $\mathbf{p}$ *с коэффициентами* $\mathbf{c}$ *как:*

$$q = \mathbf{c} \otimes \mathbf{p}$$

$$q(\omega) = \frac{\sum_i \mathbf{c}_i \cdot \mathbf{p}_i(\omega)}{\sum_i |\mathbf{c}_i|} \tag{60}$$

Легко видеть, что $\mathbf{c} \otimes \mathbf{p} \in [-1,1]^{\mathcal{O}}$

Для удобства дальнейшего использования переформулируем модель. Для каждой точки относительно позиции робота (x, y) зададим предикат:

$$p_{x,y}(\omega) = \dot{v}_{x,y} \tag{61}$$

где $\dot{v}$ — относительно текущей позиции робота зрение. Тогда обозначив вектор всех предикатов (61) как $\mathbf{p}_v$, легко видеть, что:

$$p_\nu = \nu \otimes \mathbf{p}_v \tag{62}$$

где ν — рассматривается как вектор коэффициентов. Вектор предикатов $\mathbf{p}_v$ будем в дальнейшем называть вектором базисных предикатов зрения.

Аналогично зрению можно расширить определение модели.

Определение 15. *Конечное множество предикатов нечеткой логики $\mathcal{G}$ будем называть моделью:*

$$\mathcal{G} \subset \mathcal{P} = [-1,1]^{\mathcal{O}}$$

Каждой модели $\mathcal{G}$ соответствует вектор базисных предикатов $\mathbf{p}_\mathcal{G} \in \mathcal{G}^{|\mathcal{G}|}$. Модель порождает множество выражаемых в модели предикатов:

$$\mathcal{P}_\mathcal{G} = \left\{ \mathbf{c} \otimes \mathbf{p}_\mathcal{G} \mid \mathbf{c} \in \mathbb{R}^{|\mathcal{G}|} \right\}$$

Будем говорить, что модель $\mathcal{G}$ содержит геном G если:

$$\mathcal{G} \vdash G \Longleftrightarrow G \in (\mathcal{P}_\mathcal{G} \times \mathcal{A})^*$$

Пусть задана схема принятия решений $\Lambda : (\mathcal{P} \times \mathcal{A})^* \to (\mathcal{O} \mapsto \mathcal{A})$. Будем говорить, что модель $\mathcal{G}$ содержит отображение $A : \mathcal{O} \mapsto \mathcal{A}$, если:

$$\mathcal{G} \vdash A \Longleftrightarrow \exists G : \ \mathcal{G} \vdash G : \ \Lambda(G) \equiv A$$

Зададим предикаты текущей точки, поворота и целевой точки:

$$\omega = (r, d, \cdot, r_f)$$

$$p_\rho(\omega) = 2\left(\frac{1}{1 + \|r - \rho\|}\right) - 1 \tag{63}$$

$$p_\delta(\omega) = \delta \cdot d \tag{64}$$

$$p_{\rho_f}(\omega) = 2\left(\frac{1}{1 + \|r_f - \rho_f\|}\right) - 1 \tag{65}$$

$$\tag{66}$$

Теорема 6. *Для схемы принятия решений* (52) *модель:*

$$\begin{aligned} \mathcal{G}_0 = &\{p_\nu \mid \nu \in \left\{[-1,1]^{h \times w} \mid h, w \leq \max(H, W)\right\}\} \cup \{p_\delta \mid \delta \in D\} \cup \\ &\{p_{\rho_f} \mid \rho_f \in R\} \cup \{p_\rho \mid \rho \in R\} \end{aligned} \tag{67}$$

содержит все возможные отображения $A : \mathcal{O} \mapsto \mathcal{A}$.

Доказательство. Для доказательства достаточно заметить, что каждый из предикатов (62), (63), (64), (65) задается через константное значение соответствующей компоненты ω при этом:

$$\operatorname*{Arg\,max}_{\omega\in\mathcal{O}} \; p_x(\omega) = \{\omega \mid x = \omega_x\}$$

где ω_x — соответствующая x компонента ω. Пусть задано $\omega = (r, d, v, r_f)$, и вектор $\mathbf{c}_\omega$ состоит из нулей, кроме позиций соответствующих предикатам p_r, p_d, p_v, p_{r_f}. Так как значения каждого типа предикатов зависят только от соответствующей компоненты:

$$\begin{aligned}
&\operatorname*{Arg\,max}_{\omega'\in\mathcal{O}} \; (\mathbf{c}_\omega \otimes \mathbf{p}_0)(\omega') = \\
&\qquad \operatorname*{Arg\,max}_{\omega'\in\mathcal{O}} \; \left[(1,1,1,1) \otimes (p_r, p_d, p_v, p_{r_f})\right](\omega') = \\
&\qquad \{\omega' \mid \omega'_r = r\} \cap \{\omega' \mid \omega'_d = d\} \cap \{\omega' \mid \omega'_v = v\} \cap \{\omega' \mid \omega'_{r_f} = r_f\} = \{\omega\}
\end{aligned}$$

Пусть задано отображение $A : \mathcal{O} \mapsto \mathcal{A}$. Построим эквивалентный геном G_A:

$$G_A = \{(p_\omega, A(\omega)) \mid p_\omega = \mathbf{c}_\omega \otimes \mathbf{p}_0,\ \omega \in \mathcal{O}\}$$

В силу предыдущих рассуждений:

$$\operatorname*{Arg\,max}_{(p,a)\in G_A} \; p(\omega) = \{(p_\omega, A(\omega))\}$$

а в силу определения схемы принятия решений Λ (52):

$$\left[\forall\omega \in \mathcal{O} : \; \operatorname*{arg\,max}_{(p,a)\in G_A} p(\omega) = (p_\omega, A(\omega))\right] \Longrightarrow \Lambda(G_A) \equiv A$$

что завершает доказательство. □

$\mathcal{G}_0$ со схемой голосования (53) покрывает только часть отображений.

3.2 Эволюционный подход

Определив модель, перейдем к рассмотрению алгоритмов оптимизации. Все рассматриваемые в работе генетические алгоритмы поиска можно по аналогии с эволюционной теорией разделить на два класса: эволюционные и адаптационные. Простейшим примером эволюционного алгоритма является алгоритм 4, на который часто ссылаются как на классический генетический алгоритм. Каждая итерация алгоритма состоит из трех шагов:

1. формирование популяции, процедура Evolve
2. оценка качества каждого генома, процедура Evaluate
3. отбор геномов, процедура Select

Algorithm 4 Классический генетический алгоритм

Require: Generation(), Mutation($\cdot$), Crossover($\cdot, \cdot$) — функции генерации, мутации и скрещивания, Evaluate($\cdot$) — процедура оценки качества, populationSize — размер популяции, $\alpha > 0$ — доля отбора, β — доля скрещивания, $\alpha + \beta < 1$

function SELECT(population, quality)
    $N := \alpha|\text{population}|$
    return $\arg \text{top}_N$ quality
end function

function EVOLVE(population)
    $N_{\text{crossover}} = \beta|\text{populationSize}|$
    mutated := $\varnothing$
    for $G \in$ population **do**
        mutated := mutated $\cup$ {Mutation(G)}
    end for

    crossovered := $\varnothing$
    while |crossovered| $< N_{\text{crossover}} \wedge$ |population| ≥ 2 **do**
        G := randomSample(population)
        G' := randomSample(population $\setminus$ {G})
        crossovered := crossovered $\cup$ {Crossover(G, G')}
    end while

    generated = $\varnothing$
    while |generated| $<$ populationSize $-$ |mutated| $-$ |crossovered| **do**
        generated := generated $\cup$ {*Generation*()}
    end while

    return mutated $\cup$ crossovered $\cup$ generated
end function

function CLASSICGENETICSEARCH(population, n)
    population$'$ = Evolve(population)

    quality := []
    for $G \in$ population$'$ **do**
        quality[G] = Evaluate(G)
    end for

    if $n > 0$ **then**
        population$''$ = Select(population$'$, quality)
        return ClassicGeneticSearch(population$''$, $n - 1$)
    else
        return $\arg \max_{G \in \text{population}}$ quality[G]
    end if
end function

Процедура Evolve определяется как минимум двумя (иначе генетический поиск вырождается в случайный), а чаще всего тремя функциями генерации, каждая из которых задает процедуру мутацию определенной арности, начиная с нулевой: порождение нового генома — Generation(), мутация — Mutation($\cdot$), скрещивание — Crossover($\cdot, \cdot$) и так далее. Для нашей модели соответствующие процедуры можно задать как показано в листинге 5.

Процедура Evaluate соответствует выражению:

$$\text{Evaluate}(G) = \hat{\mathbb{E}}_\theta \, Q(\theta, G) \tag{68}$$

где $Q(\theta, G)$ — выбранный критерий качества, $\hat{\mathbb{E}}_\theta$ — математическое ожидание качества либо его оценка.

Эксперимент с применением алгоритма 4 не дал никаких положительных результатов. Причиной этому является проблема начального приближения — попадание случайным начальным приближением в окрестность, в которой алгоритмы завершают все задачи из набора, маловероятно, поэтому на стадии Evaluate все алгоритмы популяции отсекались оценкой (40), превращая генетический поиск либо в случайное блуждание, либо в случайный поиск, в зависимости от способа обработки данного случая.

3.3 Адаптационный подход

Корень проблемы предыдущего подхода заключается в том, что алгоритм абстрагируется от поведения генома на шаге Evaluate, тем самым не учитывая некоторые особенности. Идея адаптационного похода состоит в приспособлении (адаптации) модели к наблюдениям, встречающимся при оценке качества, что легко укладывается в наш случай, так как модель $\mathcal{G}_0$ строится на основе наблюдений, что позволяет оценивать ее качество более детально.

Алгоритм навигации выдает последовательность команд, что позволяет пойти дальше, применяя процедуру адаптации на каждом шаге алгоритма, применяя техники из области обучения с подкреплением (reinforcement learning)[22], [23].

Мы рассмотрим два алгоритма: алгоритм обучающейся классифицирующей системы (learning classifier system, LCS)[24] и его частный случай, на который мы будем ссылать как на однослойную LCS[2]. Так как процедура адаптации применяется на каждом шаге, требуется обратная связь (feedback) в виде оценки качества одной команды.

В зависимости от типа обратной связи, алгоритм будет принадлежать к двум разным классам: обучение с учителем (supervised learning) и без учителя (unsupervised learning). Рассмотрение начинается с первого класса.

Однослойная LCS

Алгоритм 6 представляет упрощенную версию LCS, использующую модель $\mathcal{G}_0$ практически на прямую. Единственное отличие заключается в сопоставлению каждому гену $g = (p(\cdot), a)$ числового значения $\rho[g] > 0$ — веса, с помощью которого происходит обучение модели и определение общего ответа генома:

$$\hat{g} = \arg\max_{g \in G} p(\omega) w(\rho[g]) \tag{69}$$

$$(\cdot, \hat{a}) = \hat{g} \tag{70}$$

Algorithm 5 Пример реализация функций генетического поиска для задачи навигации

Require: genomeSize — размер одного генома (количество предикатов); $\alpha \in [0, 1]$ — интенсивность мутаций; DrawNormal(μ, Σ) — процедура генерации векторов, распределенных нормально: $\mathcal{N}(\mu, \Sigma)$, где μ — вектор средних, Σ — матрица ковариаций; DrawUniform(X) — процедура выбора элемента из конечного множества X в соответствии с равномерным распределением на этом множестве; DrawPoisson(λ) — процедура генерации значений, распределенных по Пуассону с интенсивностью λ; $\mathbf{p}_v$ — вектор базовых предикатов зрения.

function GENERATEGENE( )
    predicate := DrawNormal$(0_{HW}, E_{HW \times HW}) \otimes \mathbf{p}_v$
    action := DrawUniform$(\mathcal{A})$
    return (predicate, action)
end function

function GENERATION( )
    $G := \varnothing$
    while $|G| <$ genomeSize **do**
        $G := G \cup \{$GenerateGene()$\}$
    end while

    return G
end function

function MUTATION(G)
    $\lambda := \alpha \cdot |G|$
    $N :=$ DrawPoisson(λ)
    $G' := G$
    for $i = 1, \ldots, N$ **do**
        $n :=$ DrawUniform$(\{1, \ldots, |G|\})$
        $G'[n] :=$ GenerateGene()
    end for

    return G'
end function

function CROSSOVER(G_1, G_2)
    $G' := \varnothing$
    for $i = 1, \ldots,$ genomeSize **do**
        $n =$ DrawUniform$(\{1, 2\})$
        $G' := G' \cup \{G_n[i]\}$
    end for

    return G'
end function

где функция w — монотонно возрастающая весовая функция. В данной работе, в силу нормировки $p(\cdot) \in [-1, 1]$ используется логистическая функция:

$$w(\rho) = \frac{2}{1 + \exp(-\rho)} - 1 \tag{71}$$

Для однораноговой LCS наличие весов позволяет применять схему голосования:

$$\hat{a} = \arg\max_{a \in \mathcal{A}} \frac{\sum_{g \in G} \mathbb{I}[g = (p, a)] \left[p(\omega)\right]_+ w(\rho[g])}{\sum_{g \in G} \mathbb{I}[g = (p, a)] \mathbb{I}[p(\omega) > 0]} \tag{72}$$

при которой процедуры Feedback и Response алгоритма 6 заменяются на соответствующие процедуры алгоритма 9. Стоит обратить внимание, что суммирование ведется по генам для которых $p(\omega) > 0$, то есть учитываются только те гены, которые считают ω «похожей» на свой шаблон.

Процедура обучения похожа по своей структуре на общую схему работы алгоритма навигации и фактически является надстройкой над ней. Главное отличие состоит в добавлении нового этапа обратной связи, Feedback идущего после вычисление ответа системы. На этом этапе алгоритму передается некая оценка качества действия $f \in \mathbb{R}$, с помощью которой происходит пересчет весов после каждого действия. В формулировке POMDP обратная связь соответствует функции награды G, а значит в задаче навигации эквивалентна уменьшению потенциала критерия качества. Более подробно об обратной связи в LCS описано в [24]. Заметим, что в функции EvaluateAction для 6 потенциал считается известным, что соответствует обучению с учителем.

Детали реализации представлены в алгоритмах 6, 7, 8, 9.

Заметим, что несмотря на то, что схема голосования покрывает только подмножество отображений $\mathcal{O} \mapsto \mathcal{A}$, что может уменьшить качество на определенных наборах задач, она обладает важным с практической точки зрения преимуществом: так как в формировании решения принимают участия множество генов, пересчет весов на каждом шаге происходит более, чем для одного гена, что дает большую скорость сходимости по сравнению с схемой наибольшего отклика (52).

Классическая LCS

Предикаты G модели для классической обучающейся классифицирующей системы или просто LCS отличается от стандартной модели $\mathcal{G}_0$:

$$G = G_i \cup G_m \cup G_o \cup G_{io} \tag{73}$$
$$G_i \subset \{(p, m) \mid p = \mathbf{c} \otimes \mathbf{p}_0,\ m[0, 1]^+\} \tag{74}$$
$$G_m \subset \{(m_1, m_2) \mid m_1, m_2 \in [0, 1]^+\} \tag{75}$$
$$G_o \subset \{(m, a) \mid m \in [0, 1]^+,\ a \in \mathcal{A}\} \tag{76}$$
$$G_{io} \subset \{(p, a) \mid p = \mathbf{c} \otimes \mathbf{p}_0,\ a \in \mathcal{A}\} \tag{77}$$
$$(|G_i| > 0 \wedge |G_o| > 0) \vee (|G_{io}| > 0) \tag{78}$$

Все гены G делятся на четыре класса: входные (сенсорный слой), промежуточные, выходные (актуаторный слой), входные-выходные. Последние соответствуют модели $\mathcal{G}_0$. Основная концепция остается прежней — каждый ген представляет в виде пары условие-действие. Главное отличие состоит в добавлении внутренних действий и условий, которые следуя работе [24] представляются в виде не пустой последовательности

Algorithm 6 Однослойная LCS

Require: α — скорость обучения; $w(\cdot)$ — весовая функция; $\theta = (l, r_0, d_0, r_f)$ — текущая задача; $U_\theta(\cdot, \cdot)$ — потенциал функции качества для текущей задачи.

function EvaluateAction(ω, a)
    $(r, d, \cdot, \cdot) := \omega$
    $(r', d') := \varphi(l, r, d, a)$
    $\Delta U := U_\theta(r, d) - U_\theta(r', d')$
end function

function Response(G, ρ, ω)
    maxResponse $:= -\infty$
    $\hat{g} =$ NULL
    for $g \in G$ **do**
        $(p, a) := g$
        $r := p(\omega)w(\rho[g])$
        if maxResponse $< r$ **then**
            maxResponse $:= r$
            $\hat{g} := g$
        end if
    end for

    return $\hat{g}$
end function

function Feedback($(G, \rho, \hat{g}, \omega_p)$, f)
    $(\hat{p}, \cdot) = \hat{g}$
    $\rho' = []$
    for $g \in G$ **do**
        $(p, a) := g$
        $\rho'[g] := \rho[g] + \mathbb{I}[\hat{g} = g] \cdot \alpha w(\rho[g]) p(\omega_p) f$
    end for

    $(G', \rho'') :=$ Select(G, ρ')
    $(G''', \rho''') :=$ Evolve$(G', \omega_p, \rho'', f)$
    return (G'', ρ''')
end function

function Act($(G, \rho = 0_{|G|}, \hat{g} =$ NULL, $\omega_p =$ NULL$)$, ω)
    $\hat{g} :=$ Response(G, ρ)
    $(\cdot, \hat{a}) := \hat{g}$

    $f :=$ EvaluateAction$(\hat{a})$
    $(G', \rho') :=$ Feedback$((G, \rho, \hat{g}, \omega_p), f)$
    $s' := (G', \rho', \hat{g}, \omega)$
    return $(s', \hat{a})$
end function

Algorithm 7 Функция Select для алгоритма LCS

Require: ρ_0 — вес отсечения

function SELECT(G, ρ)
    $G' := \varnothing$
    $\rho' := []$
    for $g \in G$ **do**
        if $\rho[g] > \rho_0$ **then**
            $G' := G' \cup \{g\}$
            $\rho'[g] := \rho[g]$
        end if
    end for

    return (G', ρ')
end function

чисел. При совершении внутреннего действия оно становится внутренним наблюдением. Процесс повторяется до срабатывания гена из G_o, то есть гена с внешним действием.

Для генов из G_i и G_{io} отклик $p(\omega)$ вычисляется по правилам модели $\mathcal{G}_0$. Для удобства сформулируем отклик внутреннего условия μ в виде аналога предиката нечеткой логики $p_\mu : \mathbb{R}^+ \to \mathbb{R}$:

$$p_\mu(m) = \sum_k \left[\left(\sum_n |m[n+k] - \mu[n]| \right) - s \right]_+ \tag{79}$$

Процедуры генерации, мутации, скрещивания как и общая процедура эволюции полностью совпадают с процедурами в однослойной LCS.

Подробное описание алгоритма LCS и его свойств можно найти в [24].

3.4 Мета-модель

В этой части мы рассмотрим дополнение к модели $\mathcal{G}_0$, основанное на следующей неформальной гипотезе: для встречающихся на практике наборов задач отображения близкие к оптимальному являются слабым возмущением «наивного» алгоритма.

Вспомним, что предикаты базовой модели формулируются в виде:

$$p = \mathbf{c} \otimes \mathbf{p}$$

Для того, чтобы учесть гипотезу мы рассмотрим предикаты базовой системы как отдельные эвристики, а предикаты гена как композицию эвристик. Исходя из этой интерпретации несложно расширить модель:

Определение 16. *Пусть задан набор отображений $\{A_i \in \mathcal{A}^{\mathcal{O}}\}_{i=1}^N$. Каждой паре (A_i, a), где $a \in \mathcal{A}$, сопоставим предикат:*

$$p^a_{A_i}(\omega) = 2\mathbb{I}[A_i(\omega) = a] - 1 \tag{80}$$

Модель $\mathcal{G}_m$ будем называть мета-расширением модели $\mathcal{G}$ эвристиками A_i:

$$\mathcal{G}_m = \mathcal{G}_0 \cup \left\{ p^a_{A_i} \mid a \in \mathcal{A},\ i = 1, \ldots, N \right\} \tag{81}$$

Algorithm 8 Функции Evolve для алгоритма LCS

Require: ρ_1 — начальный вес генов; N_{max} — максимальный размер генома; Generation(), Generation($\cdot$), Mutation($\cdot$), Mutation($\cdot, \cdot$), Crossover($\cdot, \cdot$) — функции генерации, мутации, их адаптивные версии и функция скрещивания, α_g, α_m, α_c — доли генераций, мутаций и скрещиваний, DrawDescrete(X, p) — функция генерации реализаций случайных величин со значениями из X с распределением пропорциональным вектору p.

1: **function** EVOLVE(G, ω, ρ, f)
2: $G' := G$
3: $N = N_{max} - |G|$
4: $\rho' := \rho$
5: **for** $i = 1, \ldots, \alpha_g N$ **do**
6: **if** $\omega =$ NULL **then**
7: $g =$ Generation()
8: **else**
9: $g =$ Generation(ω)
10: **end if**
11: $G' := G' \cup \{g\}$
12: $\rho[g] = \rho_1$
13: **end for**
14: response := []
15: **for** $g \in G$ **do**
16: $(p, \cdot) = g$
17: **if** $\omega \neq$ NULL **then**
18: response$[g] = [p(\omega)]_+$
19: **else**
20: response$[g] = 1$
21: **end if**
22: **end for**
23: **for** $i = 1, \ldots, \alpha_m N$ **do**
24: $g_0 =$ DrawDescrete(G, response)
25: **if** $\omega =$ NULL **then**
26: $g =$ Mutation(g_0)
27: **else**
28: $g =$ Mutation(g_0, ω)
29: **end if**
30: $G' := G' \cup \{g\}$
31: $\rho[g] = \rho_1$
32: **end for**
33: **for** $i = 1, \ldots, \alpha_c N$ **do**
34: $g_1 =$ DrawDiscrete(G, response)
35: $g_2 =$ DrawDiscrete(G, response)
36: $g =$ Crossover(g_1, g_2)
37: $G' := G' \cup \{g\}$
38: $\rho[g] = \rho_1$
39: **end for**
40: **return** (G', ρ')
41: **end function**

Algorithm 9 Однослойная LCS со схемой голосования

Require: α — скорость обучения; $w(\cdot)$ — весовая функция;

function FEEDBACK($(G,\ \rho,\ \hat{g},\ \omega_p),\ f$)
    $(\cdot, \hat{a}) = \hat{g}$
    $\rho' = []$
    for $g \in G$ **do**
        $(p, a) := g$
        $\rho'[g] := \rho[g] + \mathbb{I}[a = \hat{a} \wedge p(\omega_p) > 0] \cdot \alpha w(\rho[g]) p(\omega_p) f$
    end for

    $(G', \rho'') := \text{Select}(G, \rho')$
    $(G'', \rho''') := \text{Evolve}(G', \omega_p, \rho'', f)$
    return $(G''\text{, } \rho''')$
end function

function RESPONSE($G,\ \rho,\ \omega$)
    response := []
    for $a \in \mathcal{A}$ **do**
        response$[a] := 0$
    end for

    for $g \in G$ **do**
        $(p, a) := g$
        $r := p(\omega) w(\rho[g])$
        if $r > 0$ **then**
            response$[a]$:= response$[a] + r$
        end if
    end for

    maxResponse := $-\infty$
    $\hat{a}$:= NULL
    for $a \in \mathcal{A}$ **do**
        if response$[a]$ > maxResponse **then**
            $\hat{a} := a$
            maxResponse := response$[a]$
        end if
    end for

    return $\hat{a}$
end function

Аналогичным образом задается расширение просто конечным набором предикатов $P \subset [-1,1]^{\mathcal{O}}$.

Очевидно, что расширение модели не может повлиять на предельное качество, однако может повлиять на скорость сходимости поиска.

Предикаты базовой модели напрямую соответствуют шаблонам наблюдений. По аналогии можно рассматривать алгоритмы расширения базовой системы $\mathcal{G}_0$ как дополнительные алгоритмические сенсоры:

$$\omega' = (r, d, v, r_f, \mathbb{I}[A_1(\omega) = \text{FORWARD}], \ldots, \mathbb{I}[A_N(\omega) = \text{TURNRIGHT}]) \tag{82}$$

Так как любой предикат p определяется вектором чисел $\mathbf{c}$, $p = \mathbf{c} \otimes \mathbf{p}_m$, все сформулированные для базовой модели алгоритмы переносятся без изменений на мета-модель.

Пусть задано вероятностное пространство $(\Theta, 2^{\Theta}, \mathbb{P})$ и критерий качества I. Рассмотрим следующую величину:

$$K(\mathcal{G}, \zeta) = \min\{|G| \mid \mathcal{G} \vdash G, I(G, \Theta) \geq \zeta\} \tag{83}$$

Величина K в некотором смысле является аналогом Колмогоровской сложности — минимальный размер генома, который достигает определенного уровня качества.

С помощью величины K можно сформулировать гипотезу более строго.

Гипотеза 1. *Пусть $\hat{\zeta}$ — уровень качества, достигаемый оптимальным алгоритмом, $\mathcal{G}_m$ мета-расширение $\mathcal{G}_0$ «наивным» алгоритмом. Тогда:*

$$\exists \zeta_0 < \hat{\zeta} : \ \forall \zeta \geq \zeta_0 : \ K(\mathcal{G}_m, \zeta) \ll K(\mathcal{G}_0, \zeta) \tag{84}$$

Важным следствием выполнения этой гипотезы является ослабление проблемы начального приближения, которое уже было упомянуто при обсуждении эволюционного подхода — несмотря на увеличение размерности гена, значительное уменьшение размера генома ведет к ощутимому увеличению вероятности попасть в область с отображениями, удовлетворяющим оценке (40) при генерации начального приближения.

Более того, можно легко явно получить начальное приближении, которое заведомо удовлетворяет оценке (40). Обозначит наивный алгоритм как B, тогда геном эквивалентный B записывается как:

$$G_0 = \{(p_B^a, a) \mid a \in \mathcal{A}\} \tag{85}$$

В численном эксперименте помимо «наивного» алгоритма модель расширялась следующими эвристиками:

$$p_1((r, d, v, r_f)) = 1 - 2\frac{\|r - r_f\|}{\|r_0 - r_f\|} \tag{86}$$

$$p_2((r, d, v, r_f)) = \frac{(r_f - r) \cdot d}{\|r_f - r\|} \tag{87}$$

$$p_3((r, d, v, r_f)) = \mathbb{H}[v[r_f] \neq 0] \tag{88}$$

$$p_4((r, d, v, r_f)) = \mathbb{H}[v[r + d] \neq 1] \tag{89}$$

$$p_6((r, d, v, r_f)) = \mathbb{H}[(r_f - r) \cdot \text{TURNLEFT}(d) - (r_f - r) \cdot d > 0] \tag{90}$$

$$p_7((r, d, v, r_f)) = \mathbb{H}[(r_f - r) \cdot \text{TURNRIGHT}(d) - (r_f - r) \cdot d > 0] \tag{91}$$

$$\tag{92}$$

где $\mathbb{H}[t] = 2\mathbb{I}[t] - 1$. Первый две эвристики соответствуют расстоянию и косинусу угла до цели. Эвристика (88) показывает наличие целевой точки в области зрения. Последние три эвристики соответствуют предсказаниям на один шаг вперед: отсутствие «стены» впереди, уменьшение угла до цели при командах TURNLEFT, TURNRIGHT.

Заметим, что с практической точки зрения обычная модель уже состоит из алгоритмических сенсоров, включая алгоритм построение карты окружения, которые формально заложены в определение наблюдения. Уже здесь можно понять, как изменение расширение базовой модели (то есть той, которая опирается на предикаты сенсоров) может повлиять на сложность 84 — в случае некумулятивного зрения теоремы 3 и 5 должны быть переформулированы для множеств отображений $R \times D \times \mathcal{O}^+ \mapsto \mathcal{A}$ и $R \times D \times \mathcal{O}^+ \mapsto \mathcal{A}$, мощность которых значительно выше.

3.5 Обучение без учителя

Вернемся обратно к алгоритмам LCS. Напомним, что в алгоритме 6 обратная связь требует знания потенциала критерия качества для текущей задачи, что в общем случае требует знания параметров задачи.

Так как зрение дает некоторую информацию о текущем окружении (остальные параметры задачи известны), можно перейти к оценке потенциала на основе зрения.

Самой простой оценкой является потенциал задаваемый «наивным» алгоритмом, то есть минимальное количество команд до целевой точки, считая неизвестные точки окружения свободными. Однако подменяя таким образом потенциал, мы принуждаем алгоритм следовать поведению «наивного» алгоритма. Так как на практике вычислительная стоимость ответа «наивного» алгоритма меньше, чем генома (причем значительно), и более того, ответ «наивного» алгоритма включен в мета-модель, такой бессмыслен.

Вместо того, чтобы получать более совершенные оценки на потенциал (что в простом случае требует знаний набора задач и распределения на нем), предлагается пожертвовать скоростью сходимости и воспользоваться более простым методом, требующим минимальных модификаций. Заметим, что оценка потенциала для зрения на шаге $t+N$ заведомо не хуже оценки на шаге t в силу свойства функции зрения (14), то есть перерасчет весов генов эффективнее, имея знания о основе будущих наблюдениях. Эту идею просто осуществить отложив этап Feedback для шага t на N шагов. Так как перерасчет весов происходит для действий в прошлом, такой подход в дальнейшем называется ретроспективным (retrospective).

Такой подход имеет два недостатка. Во-первых, даже для больших N оценка потенциал не обязана соответствовать истинному потенциалу. Полностью устранить эту проблему невозможно. Во-первых, этот подход может замедлить сходимость поиска, особенно для N сравнимых с размерами находимых путей. Однако, численные эксперименты показывают, что алгоритмы на начальных шагах имеют качество намного хуже, чем «наивный» алгоритм, поэтому различие между его потенциалом и истинным слабо влияют на скорость сходимости. К моменту же приближения качества алгоритма к «наивному», скорость сходимости существенно падает, что позволяет улучшению качества от улучшения оценки потенциала компенсировать замедление, вызванное большими N. Поэтому в данной работе предлагается постепенно увеличивать N, начиная с $N = 0$ для случайно сгенерированных геномов. Когда N становиться больше длины найденных маршрутов, алгоритм начинает напоминать эволюционный подход, в котором, оценка генома производиться после завершения задач, однако, в отличии от последнего при

достаточно плавном увеличении N, геном к этому времени будет находиться в области, в которой алгоритмы завершают задачи.

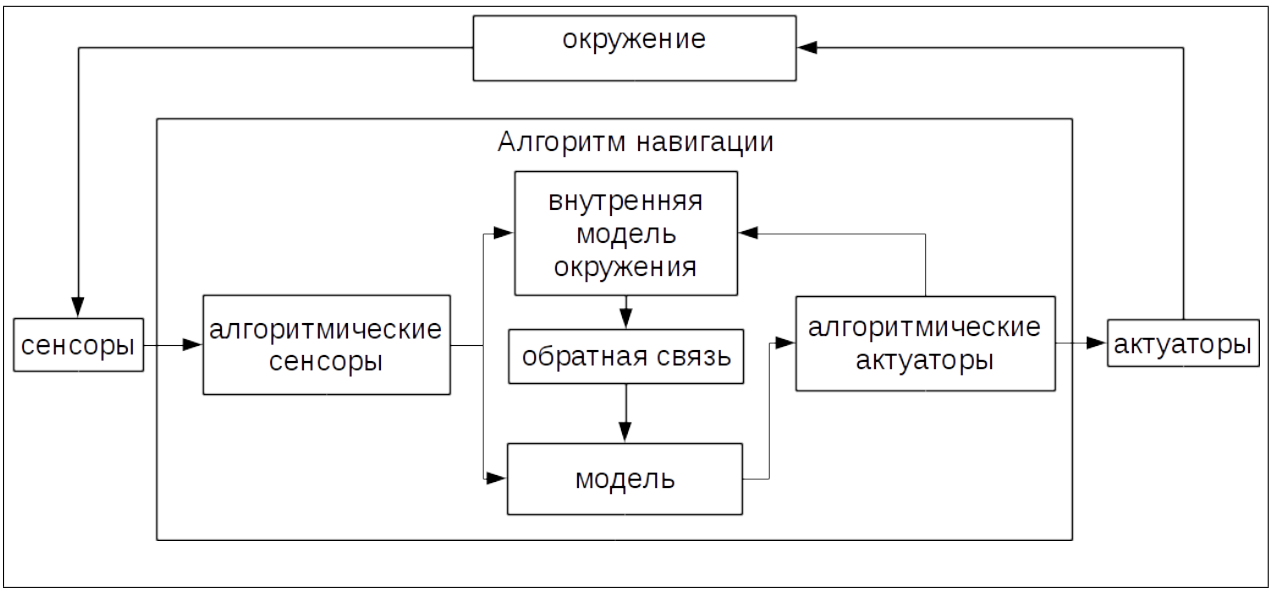


Рис. 1: Схема обобщенного генетического алгоритма с алгоритмическими сенсорами.

4 Численный эксперимент

Численный эксперимент проводился для всех описанных генетических алгоритмов, а именно:

- классический генетический алгоритм 4.
- однослойная LCS со схемой наилучшего соответствия 6
- однослойная LCS со схемой голосования 9
- классическая LCS

Для каждого алгоритма использовалась базовая и мета-модель.

Алгоритмы были реализованы на языке Scala и находятся в открытом доступе[5].

Использовались два набора задач с $H, W \sim 100$. При таких значениях провести расчет оптимального алгоритма (например, по алгоритму policy iteration) практически невозможно.

Для первого набора задач использовался случайный генератор, который в окружении $l = 1_{H \times W}$ (то есть все точки заняты), начиная с точки $r_0 = (0, W/2)$ и произвольного направления d, продвигался по этому направлении на $N[5, 10]$ шагов, случайно меняя направление после, до тех пор, пока не достигнет точки $r_f = (H, W/2)$. Каждая посещенная точка отмечалась как свободная. Распределение на задачах определяется вероятностью получить соответствующее окружение, то есть задается генератором. Этот набор задач представляет интерес только как пример набора без структуры (определение 13), так как судя по оценке качества, «наивный» алгоритм вероятно является оптимальным, либо, как минимум, близок к нему.

[5]Git-репозиторий можно найти по адресу `https://github.com/genetic-machine/genetic-machine`

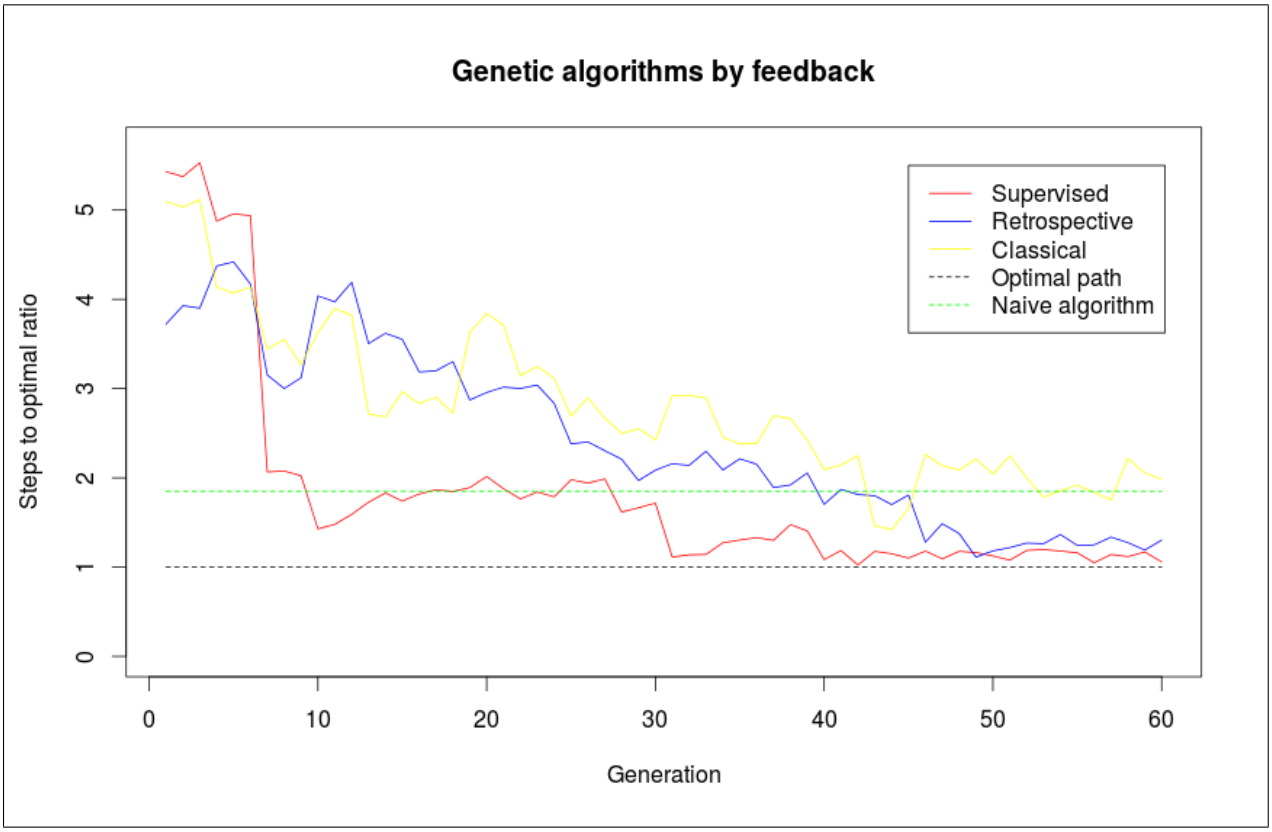


Рис. 3: Алгоритм однослойной LCS со схемой голосования. Сравнение сходимости для различных типов обратной связи: supervised — истинный потенциал (обучение с учителем), classical — потенциал «наивного» алгоритма, retrospective — ретроспективный с повышением шага задержки на 2 каждое поколение (generation), начиная с нулевой задержки. Приведены типичные кривые качества. Вертикальная ось соответствует $-I(\Theta, A)$. Эксперимент на одной задаче, generation соответствует одному проходу окружения.

Второй набор состоял из 20 предварительно сгенерированных и отобранных окружений, напоминающий план одного этажа здания — по сторонам главного широкого коридора расположены «комнаты», начальные и конечные точки находятся на разных концах коридора, причем любой путь заходя в комнату, выходит через ту же точку (или окрестность в несколько точек). Вероятностное распределение на задачах равномерное. Интуитивно понятно, что такой набор обладает структурой — простое правило, сдерживающее прокладку маршрута по комнатам, может дать значительный прирост в качестве.

В качестве функции зрения использовалась (10) с радиусом в 5 клеток.

Так как разброс длин оптимальных маршрутов относительно велик $\hat{\pi} \in [150, 200]$, для результатов представленных в данной роботе использовался нормированный критерий качества (28). Качественных выводов, отличных от представленных, анализ значений других критериев не дает. Качество «наивного» алгоритма $I(\Theta, A_n) \approx -1.8$, для конкретных задач находится в пределах $Q(\theta, A_n) \in [-2.0, -1.7]$.

Положительным результатом считалось, во-первых, нахождение отображения, способного завершить все задачи из набора, во-вторых, предельное качество превосходящее «наивный» алгоритм.

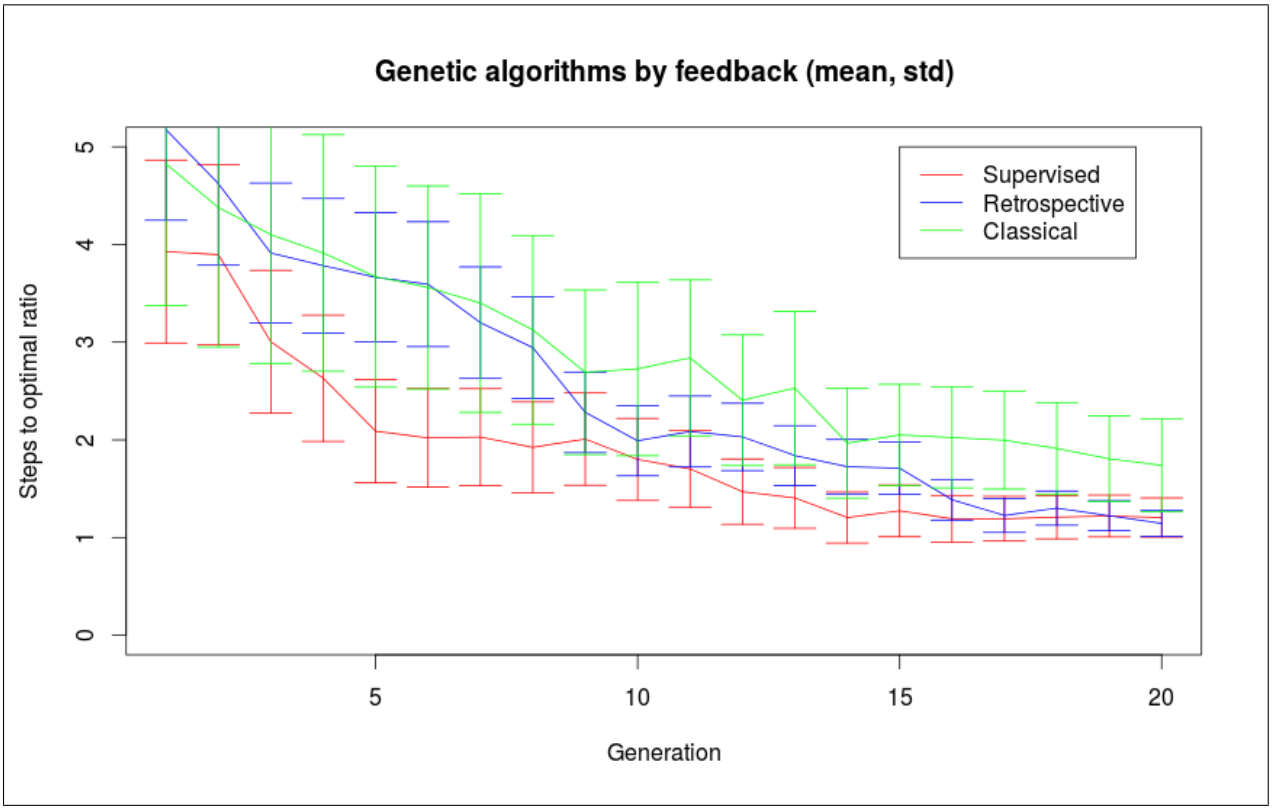


Рис. 5: Алгоритм однослойной LCS со схемой голосования. Сравнение сходимости для различных типов обратной связи: supervised — истинный потенциал (обучение с учителем), classical — потенциал «наивного» алгоритма, retrospective — ретроспективный с повышением шага задержки на 2 каждое поколение (generation), начиная с нулевой задержки. Вертикальная ось соответствует $-I(\Theta, A)$. Эксперимент на втором наборе задач, generation соответствует завершению всех 20 задач, приведены усредненные значения и стандартное отклонение.

Классический генетический алгоритм, как уже было упомянуто, столкнулся с проблемой начального приближения — даже для мета-модели, в которой начальное приближение указано явно, поиск не смог найти отображение завершающее хотя бы одну задачу, все попытки были отброшены оценкой (40).

Другие алгоритмы находили отображения, которые были способны завершить хотя бы одну задачу. Однако низкая скорость сходимости однослойной LCS со схемой наилучшего соответствия за время (как реальное, так и по количеству задач) превышающее время, понадобившееся LCS с мета-моделью со схемой голосования для опережения «наивного» алгоритма по качеству, не позволила найти отображение (даже в мета-модели), которое способно завершить все задачи набора, поэтому была вычеркнута из рассмотрения. Аналогичное поведение демонстрирует классический алгоритм LCS, который к удивлению по субъективной оценке автора (объективная оценка практически невозможна) показал скорость ниже, чем однослойная LCS.

Алгоритмы использующие обычную модель, начиная с некоторого момента, не показывали признаков заметного улучшения даже на задачах, для которых находили отображение завершающее задачу.

Только один из алгоритмов показал положительные результаты — однослойная LCS с мета-моделью со схемой голосования. Для этого алгоритма был проведен эксперимент

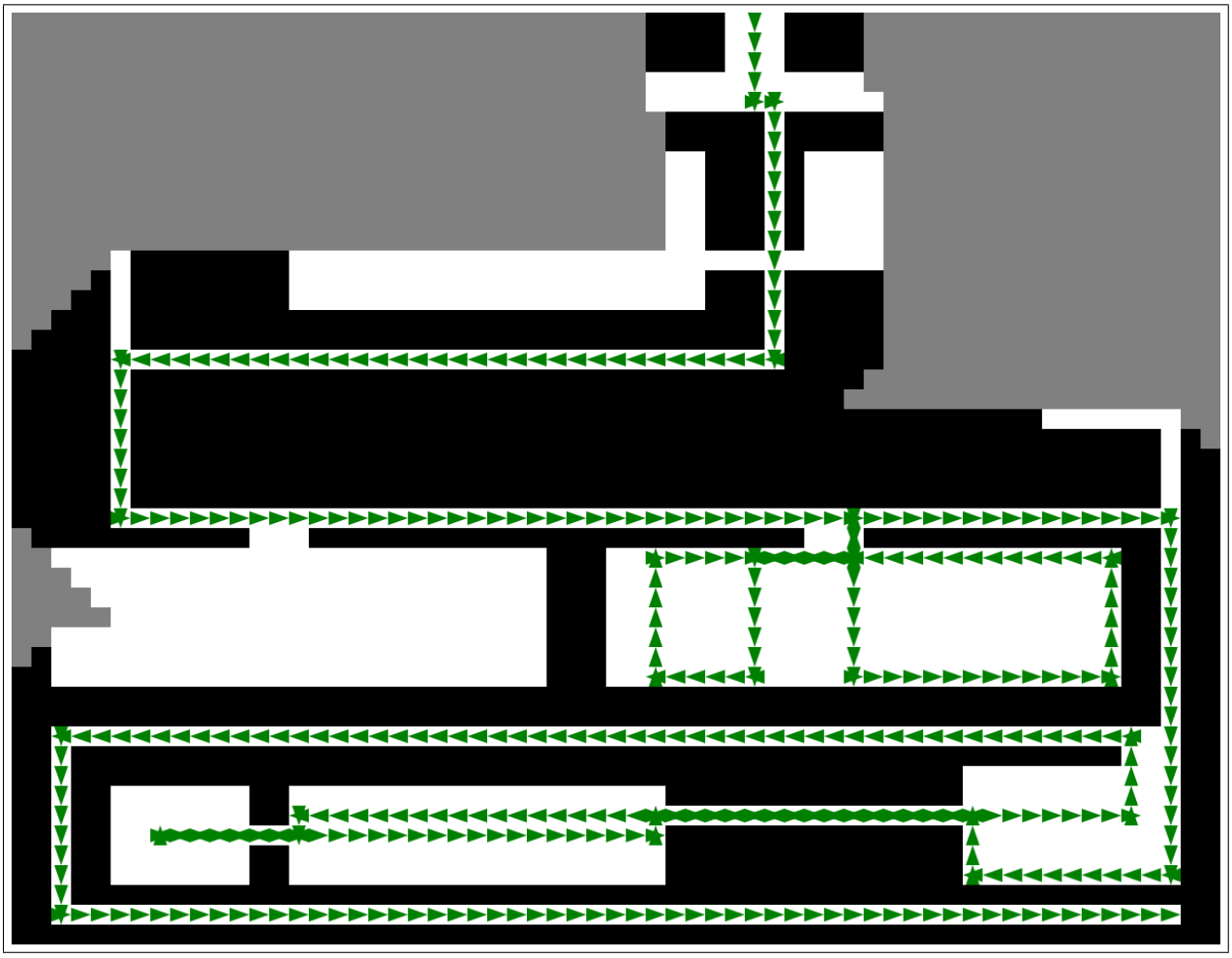
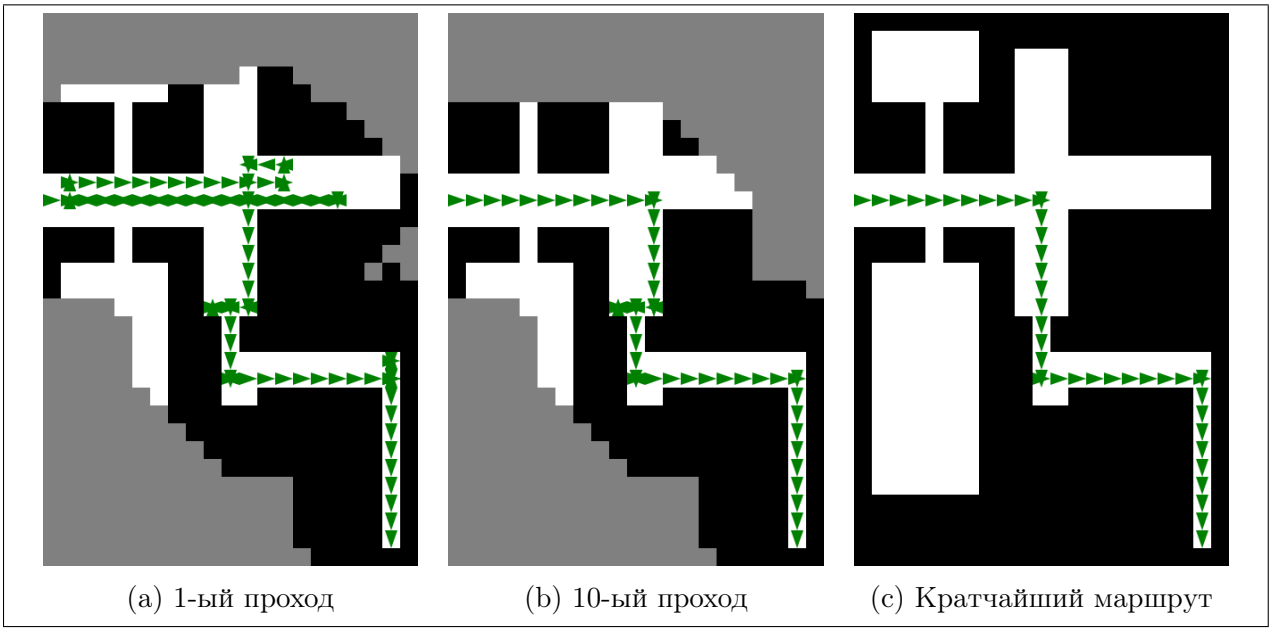


Рис. 6: Пример окружения из второго набора. Для демонстрации наличия структуры, показан путь совершаемый «наивным алгоритмом».

по сравнению различных типов обратной связи. Результаты показаны на графиках 2 и 4 (на графике для удобства показана функция $-I(\Theta, A)$). Как можно заметить, обучение с учителем приблизилось по качеству к абсолютному максимуму, при этом алгоритм с обратной связью на основе потенциала «наивного» алгоритма, как и предсказывалось, не превзошло качество характерное для «наивного алгоритма». Самым главным результатом является то, что ретроспективная обратная связь, начиная с уровня потенциала «наивного» наивного алгоритма, постепенно достигла уровня обучения с учителем. Это демонстрирует возможность успешного самообучения для задачи навигации, без необходимости в предварительной оценке/разведке окружений. Ценой этому были незначительные потери в скорости сходимости.

5 Заключение

В результате данного исследования, проведя анализ задачи была выбрана модель алгоритма навигации и доказаны ее свойства, позволяющие проводить поиск приближение к оптимальному алгоритму в этой модели. Была также сформулировано ее расширение — мета-модель, дополняющая наблюдения результатами эвристик/алгоритмических сенсоров. Было проведено исследование четырех различных генетических алгоритмов, сре-



(a) 1-ый проход (b) 10-ый проход (c) Кратчайший маршрут

Рис. 7: Демонстрация скорости сходимости. Набор состоял из единственной задачи.

ди которых был выбран один, дающие положительные результаты в численном эксперименте. В результате, получен адаптационный алгоритм навигации способный к самообучению по описанной технике, с одной стороны доступный для применения на практике с вычислительной точки зрения, с другой стороны показывающий результаты близкие к оптимальным.

6 Дальнейшая работа

Мета-модель

Единственный алгоритм, продемонстрировавший положительный результаты, был построен на основе мета-модели, которая расширяет наблюдения результатами эвристик. По гипотезе автора, для широкого с точки зрения практики класса наборов задач существует набор эвристик, позволяющих полностью заменить базовую модель, фактически превращая метод в boosting алгоритм в классическом определении. Если мощность это набора значительно меньше длины гена в стандартной модели, то возможно значительное ускорение сходимости описанных алгоритмов. Более того, существенное понижение размерности делает возможным применение классических алгоритмов оптимизации и boosting методов машинного обучения, что может повысить как вычислительную эффективность, так и качество итоговых отображений.

Определить эффективно полные системы эвристик можно анализируя близкие к оптимальным отображения полученные в результате обучения модели, которая включает в себя обычную (в том числе, описанную мета-модель).

Проблема начального приближения

Алгоритм однослойной LCS может переобучаться (излишне адаптироваться к текущей задаче), что негативно сказывается на качестве. Классический генетический алгоритм

4 при адекватной оценке качества лишен такого недостатка, но страдает от проблемы начального приближения. При этом по результатам эксперимента LCS достигает качества близкого к максимальному, а значит велика вероятность, что для других похожих наборов задач как минимум находит хорошее начальное приближение. Более того, регулировкой параметра скорости обучения, можно добиться быстрого достижения положительных результатов ценой их большего разброса. Возможно, последовательно применение этих алгоритмов может дать результаты лучше, чем показала однослойная LCS.

Другой возможный подход состоит в использовании алгоритмов приближенных решений POMDP для той же цели.

Проблема похожих генов

Схема принятие решения голосованием имеет один недостаток по сравнению со схемой лучшего соответствия, а именно слабый механизм борьбы с похожими генами, которые всегда присутствуют как результат процедур мутации и скрещивания. С одной стороны, схема голосования, в отличии от схемы наилучшего соответствия, за счет пересчета весов всех генов, которые принимали участие в голосовании, позволяет быстро наращивать веса эффективных генов вне зависимости от наличия похожих. С другой стороны, то же свойство удерживает менее эффективные похожие гены от потери веса. Выявляя такие группы генов и оставляя только наиболее эффективные, можно повысить скорость сходимости и предел качества.

Анализ близости генов по их численным представлениям крайне затруднителен, так как близость коэффициентов в общем случае не означает близость поведения и наоборот. Было начато исследование методов выявления таких групп. Предлагаемый метод основывается на метрике близости поведения, которая выражается через относительную энтропию Кульбака-Лейблера. Изначально разрабатываемый для данной работы, упрощенный метод продемонстрировал высокие результаты в похожей задаче для рекомендательных систем[25], однако для применения в алгоритме однослойной LCS требует доработки.

Список литературы

[1] Борисяк М.А. Устюжанин А.Е. Применение генетических алгоритмов для эффективного решения задачи навигации. *Труды 57-ой научной конференции МФТИ*, 2, 2014.

[2] Amanda Whitbrook, Uwe Aickelin, and Jonathan Garibaldi. Genetic-algorithm seeding of idiotypic networks for mobile-robot navigation. *arXiv preprint arXiv:0803.1626*, 2008.

[3] Ashwin Ram, Gary Boone, Ronald Arkin, and Michael Pearce. Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation. *Adaptive behavior*, 2(3):277–305, 1994.

[4] Dilip Kumar Pratihar, Kalyanmoy Deb, and Amitabha Ghosh. A genetic-fuzzy approach for mobile robot navigation among moving obstacles. *International Journal of Approximate Reasoning*, 20(2):145–172, 1999.

[5] Richard Szabo. Topological navigation of simulated robots using occupancy grid. *arXiv preprint cs/0411022*, 2004.

[6] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[7] Alberto Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. *arXiv preprint arXiv:1304.1098*, 2013.

[8] Muhammad Zohaib, Syed Mustafa Pasha, Nadeem Javaid, and Jamshed Iqbal. Intelligent bug algorithm (iba): A novel strategy to navigate mobile robots autonomously. *arXiv preprint arXiv:1312.4552*, 2013.

[9] David Kortenkamp and Terry Weymouth. Topological mapping for mobile robots using a combination of sonar and vision sensing. In *AAAI*, volume 94, pages 979–984, 1994.

[10] Sebastian Thrun, Dieter Fox, and Wolfram Burgard. Probabilistic mapping of an environment by a mobile robot. In *ICRA*, pages 1546–1551. Citeseer, 1998.

[11] Anthony R Cassandra, Leslie Pack Kaelbling, and Michael L Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, volume 94, pages 1023–1028, 1994.

[12] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25, 1995.

[13] Reid Simmons and Sven Koenig. Probabilistic robot navigation in partially observable environments. In *IJCAI*, volume 95, pages 1080–1087, 1995.

[14] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic robotics. 2005, 2005.

[15] Amalia F Foka and Panos E Trahanias. Predictive autonomous robot navigation. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 490–495. IEEE, 2002.

[16] Christos H Papadimitriou and John N Tsitsiklis. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450, 1987.

[17] Thomas L Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann E Nicholson. Planning with deadlines in stochastic domains. In *AAAI*, volume 93, pages 574–579, 1993.

[18] Sven Koenig and Reid Simmons. Xavier: A robot navigation architecture based on partially observable markov decision process models. *Artificial Intelligence Based Mobile Robotics: Case Studies of Successful Robot Systems*, pages 91–122, 1998.

[19] Sebastian Thrun. Probabilistic algorithms in robotics. *Ai Magazine*, 21(4):93, 2000.

[20] Alessandro Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180–197, 1997.

[21] Olfa Nasraoui, Fabio Gonzalez, Cesar Cardona, Carlos Rojas, and Dipankar Dasgupta. A scalable artificial immune system model for dynamic unsupervised learning. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 219–230. Springer, 2003.

[22] Wendelin Böhmer. Robot navigation using reinforcement learning and slow feature analysis. *arXiv preprint arXiv:1205.0986*, 2012.

[23] Andrew G Barto. *Reinforcement learning: An introduction*. 1998.

[24] J Doyne Farmer, Norman H Packard, and Alan S Perelson. The immune system, adaptation, and machine learning. *Physica D: Nonlinear Phenomena*, 22(1):187–204, 1986.

[25] Borisyak Maxim, Zykov Roman, and Noskov Artem. Application of relative information gain for short-term user interest detection in recommender systems. In *Proceedings of the 9th ACM Conference on Recommender systems*. ACM, 2015.