

Санкт-Петербургский национальный исследовательский университет  
информационных технологий, механики и оптики

Кафедра компьютерных технологий

А. П. Ковшаров

# **Оптимизация параметров стратегий поиска объектов на море**

Бакалаврская работа

Научный руководитель: А. С. Ковалев

Санкт-Петербург  
2015

# Содержание

Содержание . . . . .	3
Введение . . . . .	5
<b>Глава 1. Постановка задачи . . . . .</b>	<b>7</b>
1.1 Задача построения маршрута поиска в общем случае . . . . .	7
1.1.1 Расширения задачи коммивояжера . . . . .	7
1.1.2 Распределение частиц . . . . .	9
1.1.3 Формулировка задачи построения маршрута поиска в общем случае . . . . .	10
1.2 Задача построения маршрута поиска стратегией “Параллельное галсирование” . . . . .	11
1.2.1 Стратегии поиска . . . . .	11
1.2.2 Формулировка задачи построения маршрута поиска стратегией “Параллельное галсирование” . . . . .	12
1.3 Задача симуляции прохождения маршрута . . . . .	13
<b>Глава 2. Симуляция эволюции распределения . . . . .</b>	<b>15</b>
2.1 Примеры моделей и распределений . . . . .	15
2.1.1 Разновидности начальных распределений . . . . .	15
2.1.2 Разновидности моделей изменения распределения . . . . .	16
2.2 Процесс симуляции изменения распределения и прохождения маршрута . . . . .	18
2.2.1 Симуляция изменения распределения . . . . .	18
2.2.2 Симуляция сбора частиц средством поиска . . . . .	19
2.3 Корректировка области симуляции с течением времени . . . . .	20
2.4 Статистика прохождения маршрута . . . . .	22

<b>Глава 3. Алгоритм построения маршрутов согласно стратегии “Параллельное галсирование”</b>	<b>24</b>
3.1 Алгоритм построения маршрута при фиксированном распределении	24
3.1.1 Состояния алгоритма	24
3.1.2 Вспомогательные матрицы с частичными суммами весов	26
3.1.3 Переходы между состояниями	28
3.1.4 Оценка времени работы	29
3.2 Корректировка маршрута	30
<b>Глава 4. Сравнение с существующими решениями</b>	<b>33</b>
4.1 Обзор существующих решений	33
4.2 Сравнение со старым алгоритмом построения маршрутов “Кронштадт Технологии”	33
4.2.1 Тест 1	35
4.2.2 Тест 2	35
4.2.3 Тест 3	35
4.2.4 Тест 4	36
4.2.5 Результаты	41
<b>Заключение</b>	<b>42</b>
<b>Источники</b>	<b>43</b>

# Введение

Основной целью данной работы является разработка эффективного метода построения маршрутов поиска объектов согласно общепринятым стратегиям поиска. Метод должен работать с большим количеством различных моделей изменения распределения вероятности обнаружения объекта со временем.

Существующие подходы строят весь маршрут исходя из данных в начальный момент времени — информации о начальном распределении и модели его изменения. Таким образом главным недостатком существующих подходов является необходимость разработки нового алгоритма для каждой модели изменения распределения. Учитывая тот факт, что подобрать правильную модель, хорошо приближающую реальность, крайне непросто, возникает необходимость разработки алгоритма, работающего единообразно на широком классе различных моделей. Основная идея рассматриваемого подхода — использование симулятора для получения информации о распределении в любой момент времени при построении маршрута. Таким образом при планировании пользователь в первую очередь выберет модель, как можно лучше приближающую реальность в данном случае, а после запустит алгоритм построения маршрута.

Таким образом задачами данной работы являются разработка инструмента симуляции изменения вероятности во время прохождения маршрута и алгоритма построения маршрута при заданных условиях. В данной работе будет рассмотрен алгоритм построения маршрута согласно стратегии “Параллельное галсирование”.

Расчетная задача может быть использована оператором как вспомогательное средство при планировании маршрутов поиска потерпевших

бедствие объектов в реальной жизни. Визуализированный процесс изменения распределения может пригодиться при решении какие области следует исследовать более подробно, а какие можно пропустить.

Концепция данной задачи был продемонстрирована на одной из выставок. К задаче был проявлен интерес, и было решено внедрить ее в комплекс расчетных морских задач.

В главе 1 решаемая задача будет рассмотрена более подробно. Описаны классы маршрутов, получаемые при использовании стратегии “Параллельное галсирование”. Будут приведены особенности задачи, которые отличают ее от классической задачи коммивояжера и делают невозможным использование ранее разработанных методов решения для решения исходной задачи в общем случае.

В главе 2 будут рассмотрены вопросы, связанные с разработкой симулятора на CUDA. Обозначены предоставляемые симулятором сервисы.

В главе 3 будет описан алгоритм построения маршрутов согласно стратегии “Параллельное галсирование”. В первую очередь будет рассмотрен подход с динамическим программированием для определения маршрута при фиксированном распределении. Далее будет рассмотрена корректировка маршрута согласно изменениям распределения.

В главе 4 будет осуществлено сравнение с существующим алгоритмом “Кронштадт технологии”.

# Глава 1. Постановка задачи

## 1.1. ЗАДАЧА ПОСТРОЕНИЯ МАРШРУТА ПОИСКА В ОБЩЕМ СЛУЧАЕ

### 1.1.1. Расширения задачи коммивояжера

В классической формулировке задача коммивояжера (traveling salesman problem, TSP) [1] звучит так: дан взвешенный граф, необходимо найти цикл минимального веса, посещающий все его вершины. Евклидова TSP — частный случай TSP, когда весами ребер являются расстояния на плоскости. Задача построения маршрута поиска объекта на первый взгляд очень похожа на задачу коммивояжера. Однако в реальности особенности задачи оказываются существенными:

- Средство поиска в один момент времени посещает несколько вершин, а именно все вершины попадающие в круг определенного радиуса с центром в его текущем положении. Существует расширение TSP под названием GTSP [2] или обобщенная задача коммивояжера, которое решает следующую задачу: дан взвешенный граф и разбиение его вершин на **непересекающиеся** множества, необходимо найти цикл минимального веса, посещающий хотя бы одну вершину из каждого множества. Существует сведение данной задачи к TSP не увеличивающее размерности. Но к сожалению данное расширение неприменимо к нашей задаче, так как множества могут пересекаться.
- Зачастую все вершины посетить физически невозможно. Соответственно выделяют вершины в которых более вероятно обнаружить объект. Сопоставим вершине  $v$  величину  $p_v$  — вероятность обнару-

жить объект в этой вершине.  $p_{path} = \sum_{v \in path} p_v$ . На практике длина путей с  $p_{path} \geq 0.99$  может превышать длину путей с  $p_{path} \geq 0.9$  в десятки раз. То есть длина пути может расти экспоненциально в зависимости от  $p_{path}$ . Следовательно необходимым параметром задачи становится максимальная длина пути (или время поиска с физической точки зрения). Известно обобщение Profit Based TSP [3]: каждой вершине сопоставляется значение  $p_v$ , при посещении вершины к сумме призов добавляется  $(p_v - t_v)$ , где  $t_v$ —время посещения, необходимо составить маршрут с наибольшей суммой призов. К сожалению наша задача и здесь сравнительно более общая, так как величины призов могут изменяться нелинейно.

- Распределение  $p_v$  может быть не статично по времени и изменяться согласно заданной модели [4]. Следовательно время и расстояние не взаимозаменяемы в поставленной задаче и оптимальное значение  $p_{path}$  может быть различно если мы фиксируем один из параметров. В поставленной задаче будет фиксированно время. Однако все модели изменения обладают свойством: распределение изменяется непрерывно, перераспределяясь не превышая фиксированную скорость, не появляется извне и не исчезает (будем считать, что при посещении призы “собираются” и исчезают). Соответственно  $p_{v,t} = \sum_{pos(q_i,t)=v} q_i$ , где  $pos(q, t)$  — функция, возвращающая местоположение (вершину графа) приза  $q$  в момент времени  $t$ . Приз  $q_i$  в вершине  $v$  мы можем собрать лишь в определенные промежутки времени. Обобщение Time Windows TSP [5] решает соответствующую задачу: вершину  $v$  можно посетить лишь в промежуток  $[t_{v,l}; t_{v,r}]$ . Проблема сведения к этому обобщению в том, что значительно увеличивается количество вершин, а именно новая вершина будет сопоставлена  $j$ -му моменту, когда приз  $q_i$  оказался в вершине  $v$ .

- Другое обобщение для случая с движущимися вершинами — Kinetic TSP [6] или Moving Targets TSP [7]. Это классическая Евклидова TSP, в которой вершины движутся в фиксированном направлении. Однако в существующих работах были рассмотрены только случаи где вершины движутся с фиксированной скоростью, или коммивояжер должны возвращаться в стартовую вершину после посещения каждой. В исходной задаче скорость и направления перемещения призов могут изменяться. Кроме того сведение вновь значительно увеличивает число вершин.
- Фактически до текущего момента времени задача формулировалась в непрерывном случае. При разработке алгоритма для ЭВМ необходимо провести некоторую дискретизацию. Необходимое количество вершин для двумерной задачи растет квадратично и соответственно приемлемое значение может достигать миллионов вершин. Такое количество вершин велико даже для самых быстрых реализаций приближенных решений TSP.
- В реальной жизни невозможно искать согласно произвольной траектории из-за технических ограничений на передвижение поискового средства. Помимо этого недопустимо применение не общепризнанных, зарекомендовавших себя стратегий поиска.

### 1.1.2. Распределение частиц

Использование понятия распределение вероятности обнаружения противника в дальнейшем будет не совсем удобно, так как одна из основных частей задачи “сбор” вероятности — не является корректной операцией над вероятностями.

Введем альтернативное понятие — распределение “частиц”. Частица  $\pi : \Pi$  — гипотеза изначального расположения объекта и его поведения в



дальнейшем. Вес частицы  $w_\pi$  — вероятность осуществления именно этой гипотезы.  $pos(\pi, t)$  — положение частицы  $\pi$  в момент времени  $t$ . Сумма весов всех частиц изначально равна единице. Расширение понятия веса частиц на непрерывный случай аналогично расширению для вероятностного пространства. “Собрать частицу” — проверить гипотезу. После проверки частица исчезает и больше не может быть собрана. Распределение частиц  $f(dS, t, path)$  — функция, ставящая в соответствие области и времени сумму весов частиц находящихся в этой области в заданное время, с учетом частиц собранных средством поиска к данному моменту. С течением времени частицы могут перемещаться в любом направлении с ограниченной скоростью, однако не могут появляться из ниоткуда или исчезать иным способом, кроме сбора их средством поиска. Сумма собранных частиц фактически равна априорной вероятности обнаружить объект, имеющий заданное начальное распределение вероятности и закон его изменения.

### 1.1.3. Формулировка задачи построения маршрута поиска в общем случае

Задано распределение частиц  $f(dS, t, path)$ ,  $t_{search}$  — время поиска и параметры средства поиска:  $pos_0$  — начальная позиция,  $r$  — радиус обнаружения и  $v_{max}$  — максимальная скорость передвижения. Необходимо найти маршрут  $path(t)$ , определяющий позицию средства поиска в любой момент времени и максимизирующий сумму весов собранных частиц  $sum_{res}$ . Более формально  $sum_{res} = 1 - \int_S f(dS, t_{search}, path)$ ,  $path(0) = pos_0$ ,  $|path'(t)| \leq v_{max}$ , собраны те и только те частицы  $\pi$  для которых  $\exists \tau \leq t_{search} ||pos(\pi, \tau) - path(\tau)|| \leq r$ .

## 1.2. ЗАДАЧА ПОСТРОЕНИЯ МАРШРУТА ПОИСКА СТРАТЕГИЕЙ “ПАРАЛЛЕЛЬНОЕ ГАЛСИРОВАНИЕ”

### 1.2.1. Стратегии поиска

Однако, помимо того, что решения общей задачи сложно найти, построенные маршруты могут быть практически неприменимы из-за ограничений на передвижение средства поиска. В реальной жизни поиск объектов осуществляется общепринятыми стратегиями, такими как “заданный маршрут”, “гребенка”, “расширяющийся квадрат”, “параллельное галсирование”(рис. 1.1). **Галс** — линия пути воздушного судна от поворота до поворота.

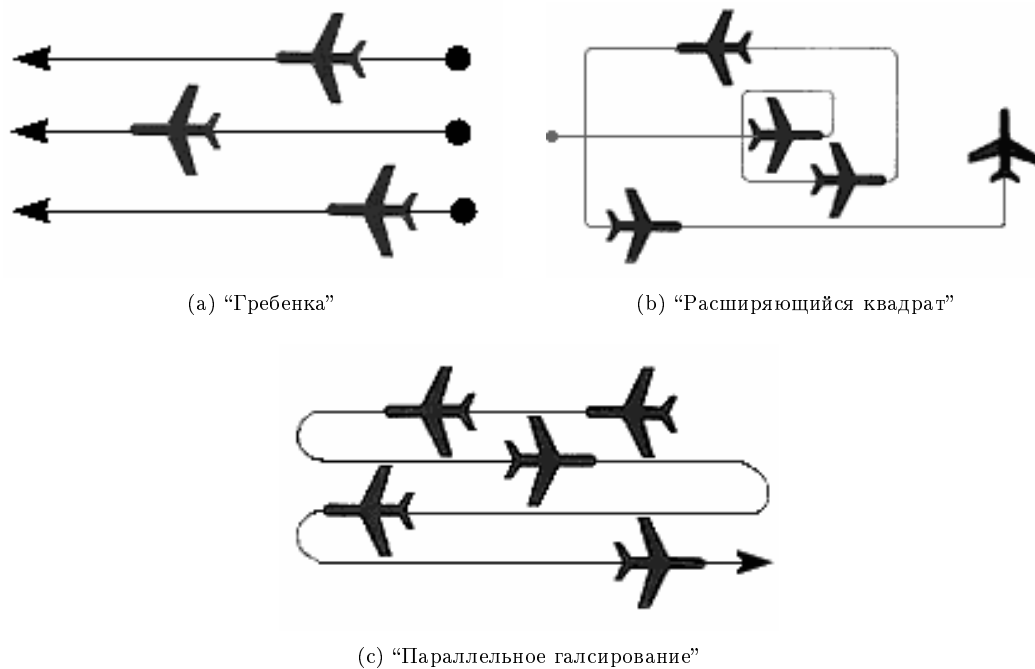


Рис. 1.1: Стратегии поиска

Согласно приказу об утверждении правил проведения авиационных поисково-спасательных работ [8]:

“Поиск способом “Заданный маршрут” выполняется по линии заданного пути, проходящего вдоль участка маршрута воздушного судна, потер-

певшего бедствие.”

“Способ “Гребенка” заключается в одновременном обследовании района поиска группой воздушных судов путем совместного полета по параллельным прямолинейным маршрутам на интервалах, составляющих примерно 75% визуальной видимости или дальности действия поисковой аппаратуры.”

“Поиск [способом “Расширяющийся квадрат”] состоит в обследовании одиночным воздушным судном района вокруг известной точки, в которой предполагается нахождение потерпевшего бедствия экипажа.”

“Поиск способом “Параллельное галсирование” применяется при недостаточном количестве имеющихся поисково-спасательных воздушных судов и для обследования значительной площади. ... При этом способе район поиска разделен на несколько участков поиска (полос), которые просматриваются одновременно несколькими одиночными воздушными судами или последовательно одним воздушным судном. ... Расстояние между галсами устанавливается таким же, как и интервал между воздушными судами при поиске способом “Гребенка”. Для сокращения количества разворотов прямолинейные участки галсов целесообразно ориентировать вдоль полос обследования.” Следуют отметить, что для разных местностей рекомендуется разное расстояние между галсами.

В данной работе будет рассмотрено построение маршрута поиска одиночным самолетом согласно стратегии “Параллельное галсирование”.

### **1.2.2. Формулировка задачи построения маршрута поиска стратегией “Параллельное галсирование”**

Задано распределение частиц  $f(dS, t, path)$ ;  $t_{search}$  — время поиска; параметры средства поиска:  $pos_0$  — начальная позиция,  $r$  — радиус обнаружения и  $v_{max}$  — максимальная скорость передвижения; параметры стратегии поиска:  $l$  — прямая параллельная направлению гал-

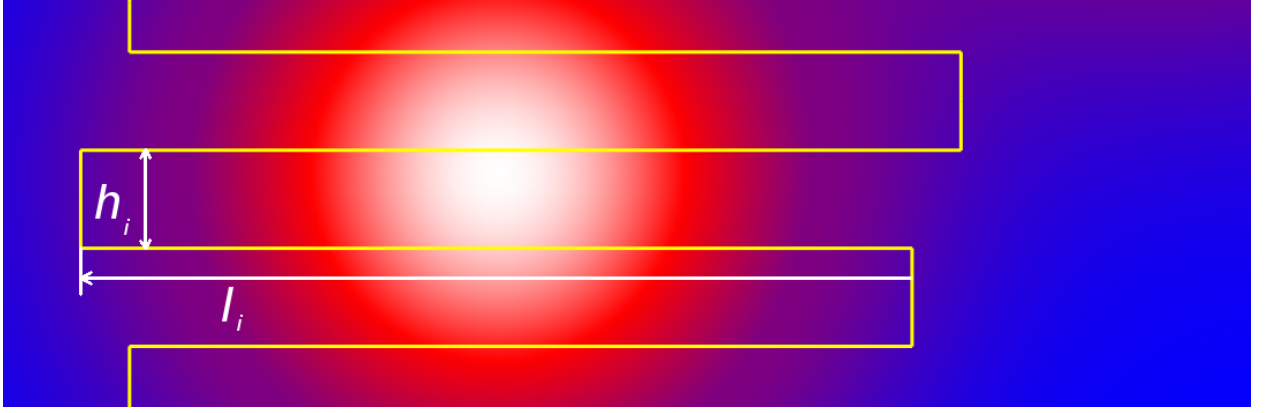


Рис. 1.2: Параметры “параллельного галсирования”

сов(полосе обследования). Необходимо найти маршрут  $path(t)$ , определяющий позицию средства поиска в любой момент времени и максимизирующий сумму весов собранных частиц  $sum_{res}$ .  $path(t)$  должен состоять из прямолинейных отрезков, удовлетворяющих стратегии поиска “параллельное галсирование” с галсами параллельными прямой  $l$ .  $path(t)$  должен быть задан количеством галсов  $n$ , а также последовательностями чисел  $l_i, i \in 1..n$  — величина проекции на прямую  $l$  (может быть отрицательной),  $h_i, i \in 1..(n-1)$  — расстояние между галсами с номерами  $i$  и  $i+1$ . (рис. 1.2)

$$\sum_{i=1}^{n-1} (|l_i| + h_i) + |l_n| \leq v_{max} \cdot t_{search}.$$

Средство поиска перемещается по пути со скоростью  $v_{max}$ . Более формально  $sum_{res} = 1 - \int_S f(dS, t_{search}, path)$ ,  $path(0) = pos_0$ ,  $|path'(t)| = v_{max}$ , собраны те и только те частицы  $\pi$  для которых  $\exists \tau \leq t_{search} ||pos(\pi, \tau) - path(\tau)|| \leq r$ .

### 1.3. ЗАДАЧА СИМУЛЯЦИИ ПРОХОЖДЕНИЯ МАРШРУТА

Для визуализации распределений использована аналогия с температурой: наиболее вероятные участки обнаружения противника отображаются белым цветом (наиболее горячие), менее вероятные красным, самые маловероятные синим (рис. 1.3). Промежуточные состояния изображены градиентом между этими цветами.

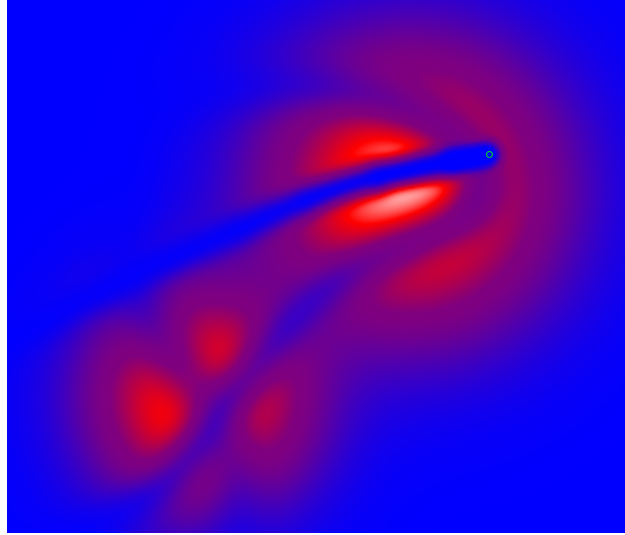


Рис. 1.3: Пример визуализации

Инструмент, проводящий симуляцию изменения распределения вероятности в условиях сбора ее средством поиска, помимо визуализации должен предоставлять статистические данные для возможности его использования в алгоритмах оптимизирующих параметры маршрутов поиска. Размеры данных, которые должны быть обработаны в реальном времени оказываются слишком велики, и использование лишь мощностей CPU оказывается недостаточным. Поэтому для разработки были задействованы мощности GPU и технология CUDA.

Благодаря тому, что большинство моделей изменения распределения можно представить **ядром** (Kernel, Convolution Matrix, матрица свертки — небольшая матрица используемая для создания блюра, резкости и тому подобных эффектов, посредством свертки с изображением) и фактически производить Image Convolution (свертку изображения) [9, 10] с заданным ядром на каждой итерации — задача хорошо подходит для архитектуры CUDA [11]. Более сложные модели удастся симулировать посредством нескольких запусков с различными ядрами и последующей их композицией.

Более подробно подход для симуляции описан в главе 2.

# Глава 2. Симуляция эволюции распределения

## 2.1. ПРИМЕРЫ МОДЕЛЕЙ И РАСПРЕДЕЛЕНИЙ



(a) Композиция стандартных распределений для задания начального распределения (b) Изменения распределения согласно модели случайного блуждания

Рис. 2.1: Пример симуляции изменения распределения без маршрута

### 2.1.1. Разновидности начальных распределений

Начальное распределение частиц фактически может быть представлено любой двумерной функцией, интеграл которой по всей плоскости равен единице. Однако для большинства применений кажется достаточным задание начального распределения как композиции стандартных двумерных распределений (рис. 2.1(a)), таких как равномерное распределение в произвольной области или нормальное распределение задаваемое эллипсом, содержащим  $3\sigma$  вероятности и тому подобное. Инструмент симуляции предоставляет удобный инструмент их задания.

На протяжении всей симуляции распределение хранится в текстуре (матрице). Значение, которое записано в каждом пикселе, означает сум-

марный вес частиц, находящихся под этим пикселем, если его нарисовать в мире. Пиксель текстуры имеет достаточно малый размер при рисовании в мире, чтобы достаточно точно различать положения различных частиц, при этом достаточно большой, чтобы текстура покрывающая всю вероятность влезла в память GPU, и ее обработка занимала приемлимое время.

### 2.1.2. Разновидности моделей изменения распределения

Изменения распределений происходят с фиксированным временем дискретизации  $\Delta t$ . Для симуляции изменений в большинстве моделей используется свертка с ядром с периодом  $\Delta t$ .

Процесс свертки с ядром осуществляется следующим образом, пусть  $D_t, D_{t+\Delta t}$  — последовательные распределения, а  $K$  — ядро размера  $(2k + 1) \times (2k + 1)$  (Пусть обращение к несуществующим элементам возвращает 0):

$$D_{t+\Delta t}[i][j] = \sum_{di=-k}^k \sum_{dj=-k}^k D_t[i + di][j + dj] \cdot K[di][dj] \quad (2.1)$$

Размер матрицы свертки выбирается такого размера, чтобы дать возможность симулировать передвижение в произвольном направлении с достаточной точностью. На данный момент выбрано значение  $k = 5$ .

Время  $\Delta t$  выбирается исходя из размера ядра и физического размера пикселя. Если размер пикселя  $h \times h$ , максимальная скорость перемещения объекта  $v_{max}$ , тогда (2.2).

$$\Delta t = \frac{k \cdot h}{v_{max}} \quad (2.2)$$

- **Модель случайных блужданий**

Простейший пример модели изменения распределения — модель случайных блужданий (рис. 2.1(b)). Ядро для модели случайных блужданий представляет собой матрицу со значениями равными

площади пересечения текущей ячейки с кругом вписанным в квадрат размера  $2kh \times 2kh$ . Ячейки соответственно отнормированы для равенства суммы единице.

Посредством изменения ядра (замены круга на кольца), можно изменять интенсивность блужданий.

- **Модель движения в одном направлении**

Пусть заданы последовательности  $\alpha_i, p_i, 1 \leq i \leq n$  — углов и вероятностей движения в направлении каждого из них. Для симулирования заданной модели можно использовать  $n$  экземпляров ядер и текстур для симуляции движения распределений соответственно в каждом из направлений. На выходе для получения итогового распределения достаточно скомбинировать полученные результаты с соответствующими весами  $p_i$ .

- **Модель притяжения(отталкивания) к заданным точкам**

Иногда уместной оказывается модель избегания некоторых точек (например точек вблизи текущей позиции поискового средства, если объект является вражеским и не желает быть обнаруженным) или напротив приближения к ним (например в случае если объект знает ближайшее положение суши, он может стремиться двигаться в ее направлении). Здесь удобно воспользоваться аналогией с электрическими зарядами, однако действие электрических сил для создания ускорения не кажется естественным в данной задаче. Значение функции аналогичной силе Кулона будет задавать направление и скорость (вместо ускорения) движения при нахождении в данной точке. Таким образом каждая из точек имеет свой заряд  $q_i$ , позицию  $pos_i$ , радиус действия  $R_i$  и закон убывания влияния, для примера используем Кулоновские  $r^{-2}$ . Пусть максимальная скорость перемещения объекта  $v_{max}$ . Таким образом вектор скорости объекта в



точке  $p$  равен (2.5).

$$\chi_i(p) = \begin{cases} 1 & \text{если } (p - pos_i)^2 \leq R_i \\ 0 & \text{иначе} \end{cases} \quad (2.3)$$

$$v' = \sum_{i=1}^n \frac{\chi_i(p) \cdot q_i \cdot (pos_i - p)}{|pos_i - p|^3} \quad (2.4)$$

$$v = \min(|v'|, v_{max}) \cdot \frac{v'}{|v'|} \quad (2.5)$$

При реализации данной модели, во время свертки ядро в каждой конкретной ячейке будет различным. Таким образом приходится работать с текстурой скоростей из глобальной памяти вместо ядра в константной памяти как в модели со случайными блужданиями, что в несколько раз замедляет время симуляции.

## 2.2. ПРОЦЕСС СИМУЛЯЦИИ ИЗМЕНЕНИЯ РАСПРЕДЕЛЕНИЯ И ПРОХОЖДЕНИЯ МАРШРУТА

### 2.2.1. Симуляция изменения распределения

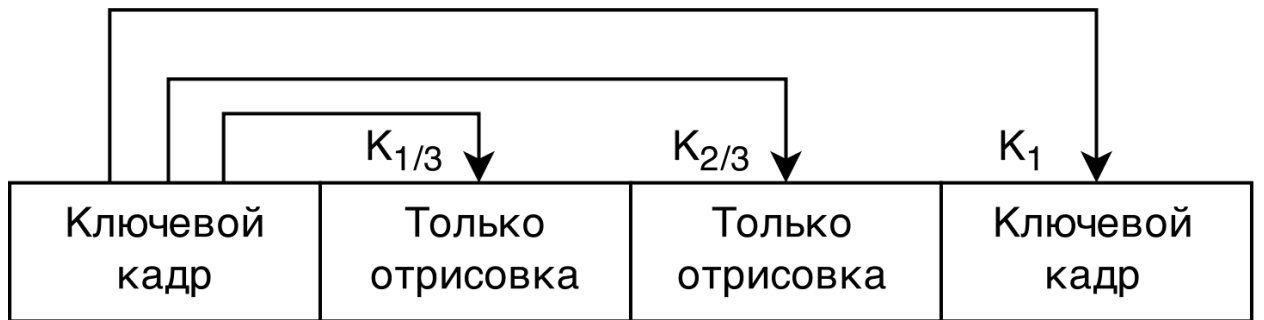


Рис. 2.2: Схема применения частичных ядер

Симуляция изменения распределения производится путем применения свертки текущего распределения с ядром с промежутком времени

$\Delta t_{frame}$ . Если  $\frac{1}{\Delta t} \geq 60FPS$ , где  $\Delta t$  — время с которым необходимо применять заданное ядро, тогда  $\Delta t_{frame} = \Delta t$ . Иначе время применения ядра слишком велико и не обеспечивает необходимой динамичности.

Для обеспечения актуальной информации в промежутках между ключевыми кадрами, идущими с интервалом  $\Delta t$ , необходимо добавить  $m$  промежуточных, так чтобы  $\Delta t_{frame} = \frac{\Delta t}{m+1} \geq 60FPS$ . Таким образом из оригинального ядра  $K_1$  нам необходимо получить  $m$  промежуточных с некоторой точностью приближающих реальность  $K_{\frac{1}{m+1}}, K_{\frac{2}{m+1}}, \dots, K_{\frac{m}{m+1}}$ . Помощью применения заданных ядер к последнему ключевому кадру, будут получены приближенные распределения в промежуточные моменты времени (рис. 2.2), однако полученные с помощью этих ядер текстуры не будут использованы для построения следующих, что избавляет симулятор от накопления погрешности от применения приближенных ядер.

Далее описан один из вариантов получения ядра  $K_{\frac{i}{m+1}}$ , которой достаточно хорошо демонстрирует себя на практике. Необходимо элементы ядра  $K_1$  домножить на  $\alpha = 1 + c_0 - c_0^{\frac{i}{m+1}}$ , где  $c_0 = K_1[k][k]$ , после чего добавить остаток  $1 - \alpha$  к стационарной ячейке получившегося ядра  $K_{\frac{i}{m+1}}[k][k]$ .

### 2.2.2. Симуляция сбора частиц средством поиска

Сбор частиц производится с интервалом  $\Delta t_{frame}$  в текстуре соответствующей последнему на данный момент ключевому кадру. После сбора к ключевому кадру применяются ядра, рассчитывающие перемещение оставшихся частиц, для получения кадра, соответствующего текущему времени.

Сбор частиц осуществляется посредством домножения значения каждого пикселя на долю его площади, не покрытой зоной видимости ни в один из моментов времени в течении текущего интервала  $\Delta t_{frame}$ . То есть подразумевается, что вес частицы распределен равномерно по площади пикселя.

Для получения суммарной зоны видимости во все моменты времени текущего интервала маршрут средства поиска делится на прямолинейные участки. После чего на каждом участке берется объединение  $n = \beta \frac{dist}{r}$ , взятых через равные промежутки, зон видимости, где  $dist$  — длина прямолинейного участка,  $r$  — радиус видимости средства поиска,  $\beta$  — коэффициент точности края. Выбрано  $\beta = 100$ , что для всех разумных размеров текстур дает достаточно точные края объединения зон видимости.

### 2.3. КОРРЕКТИРОВКА ОБЛАСТИ СИМУЛЯЦИИ С ТЕЧЕНИЕМ ВРЕМЕНИ

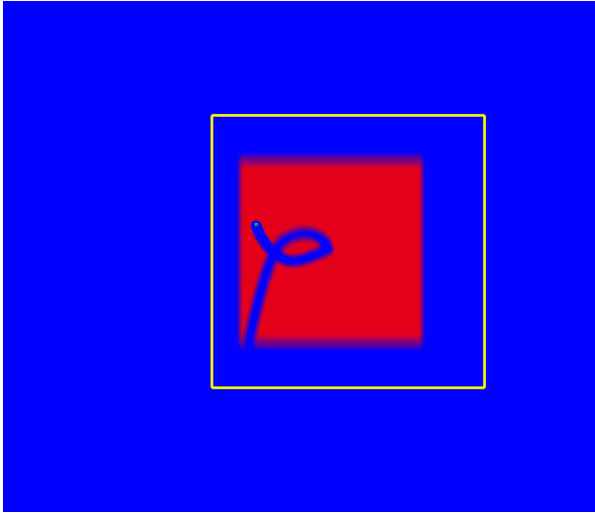


Рис. 2.3: Начальная область симуляции

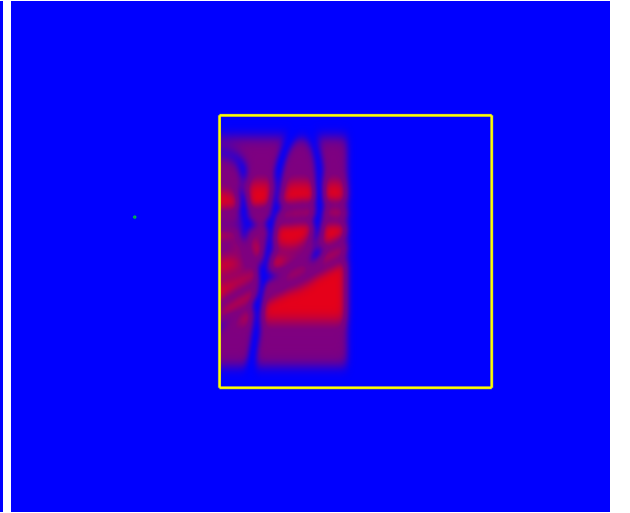


Рис. 2.4: Выход распределения за границы области, без корректировки

Одна из сторон прямоугольника  $rect$ , охватывающего интересующий район поиска, параллельна прямой галсирования (в случае построения маршрутов стратегией “Параллельное галсирование”). В начальный момент времени район задается вручную пользователем (рис. 2.3). Частицы, оказавшиеся снаружи начального района учитываться не будут.

Разрешение текстуры выбирается таким образом, чтобы размер пикселя был квадратным (при необходимости немного изменяется

выбранный район для симуляции), а количество пикселей было максимально возможным, при котором можно обеспечить приемлимую производительность. Сейчас выбрано количество пикселей порядка  $800^2 = 6.4 \cdot 10^5$ .

С течением времени рассматриваемые частицы могут выходить за пределы текущего рассматриваемого района. Без корректировки района информация о местоположении этих частиц будет потеряна (рис. 2.4).

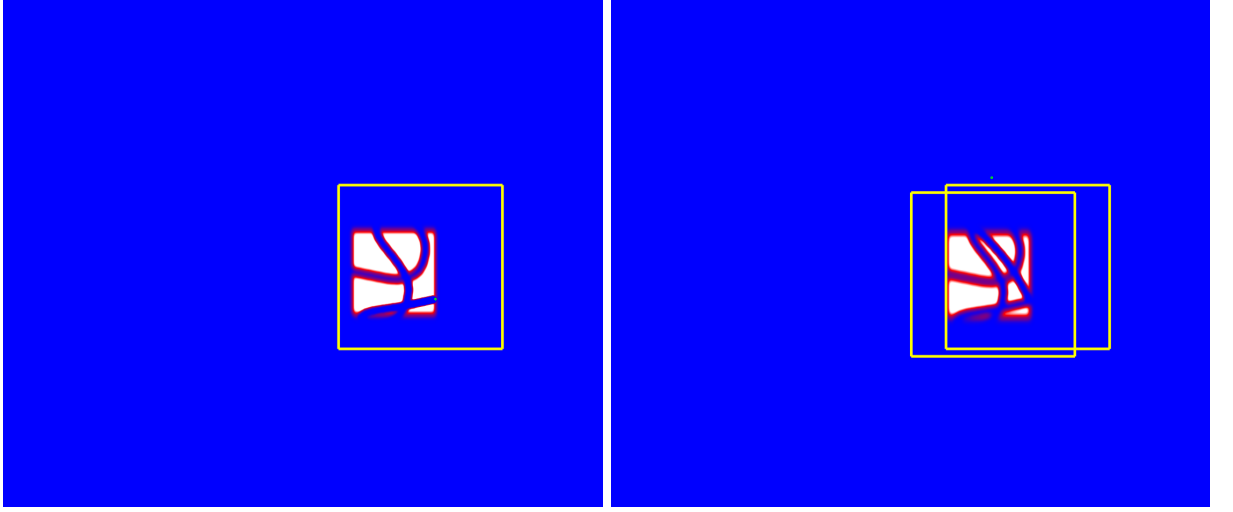


Рис. 2.5: Приближение распределения к границе области симуляции

Рис. 2.6: Изменение центра области симуляции согласно новому положению распределения

Корректировка рассматриваемого района осуществляется, когда в одном из пикселей на границе шириной  $2k$ , где  $k$  — параметр ядра, находятся частицы с суммарным весом более  $\epsilon$ . (рис. 2.5) В реализации выбрано  $\epsilon = 5 \cdot 10^{-3}$ .

Для корректировки находится прямоугольник *rect*, охватывающий все пиксели суммарный вес в которых больше  $\epsilon$ , причем длина и ширина прямоугольника содержат четное число пикселей. После чего выделяется новая текстура с центром, совпадающим с центром полученного прямоугольника (рис. 2.6) (благодаря четности, стороны пиксели новой и старой текстуры выровнены), после чего данные копируются в новую текстуру.

Так как разрешение новой текстуры всегда остается прежним, воз-

можно необходимо увеличить прямоугольник покрываемый текстурой для покрытия прямоугольника *rect*. Размер покрываемого прямоугольника последовательно увеличивается в два раза, равномерно расширяясь относительно центра, до тех пор пока не покроет прямоугольник *rect*, (рис. 2.7) после чего его положение немного корректируется, чтобы в каждый новый пиксель (если увеличили в  $2^m$  раз) попало ровно  $2^{2m}$  целых старых пикселей.

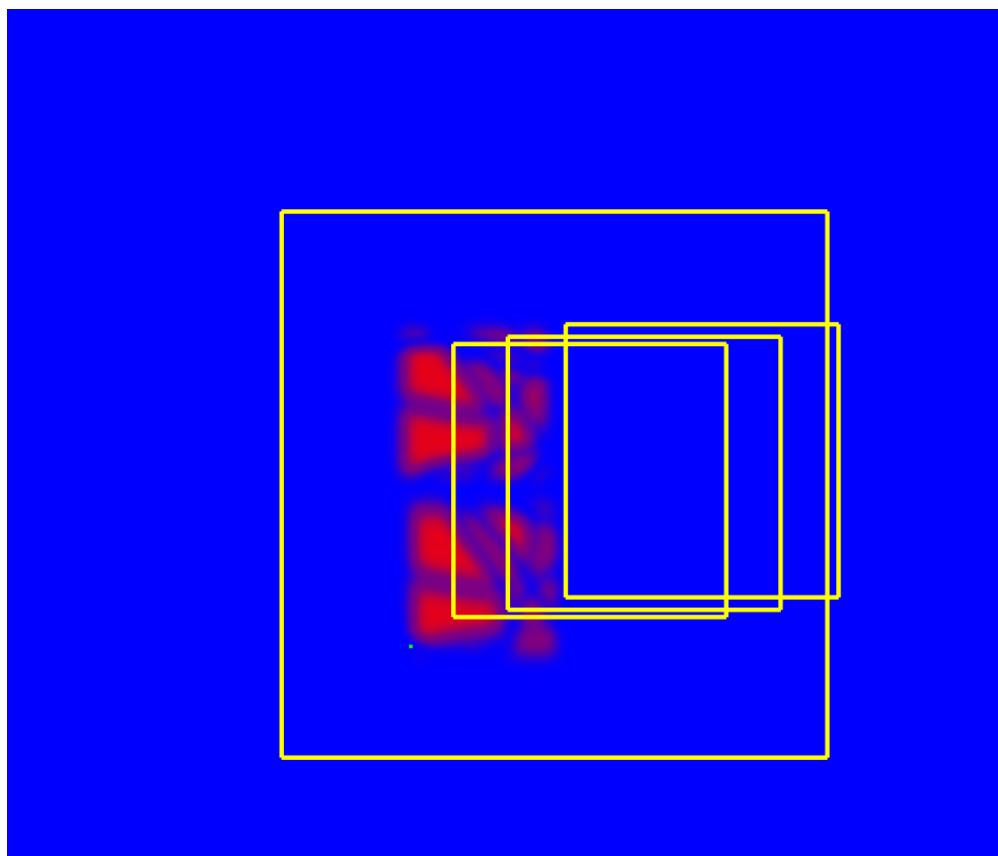


Рис. 2.7: Расширение и изменение центра области симуляции

## 2.4. СТАТИСТИКА ПРОХОЖДЕНИЯ МАРШРУТА

Симулятор предоставляет статистику о прохождении маршрута, по которой строятся графики прогресса поиска и поисковой производительности. Оба графика отображают зависимости от времени, поэтому изобра-

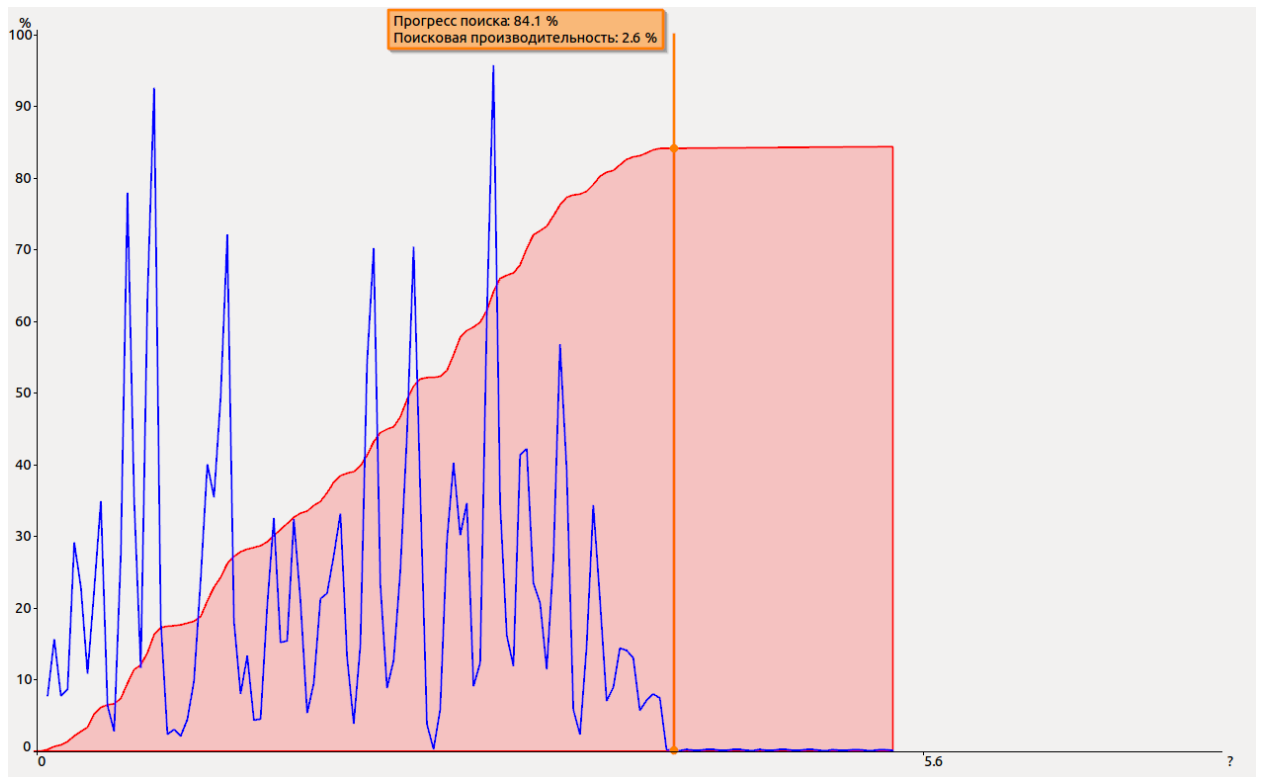


Рис. 2.8: Окно с отображением статистики

жены в одном окне с разными шкалами по оси ординат. (рис. 2.8)

Прогресс поиска (красный график) — отображает текущий суммарный вес собранных частиц в процентах от общего веса в начальный момент времени.

Поисковая производительность (синий график) — отображает текущую эффективность поиска. Фактически является производной прогресса поиска по времени, однако также измеряется в процентах. Поисковая производительность  $x\%$  — означает, что если мы будем искать с такой поисковой производительностью в течении всего времени поиска  $t_{search}$ , то сможем собрать  $x\%$  веса частиц.

# Глава 3. Алгоритм построения маршрутов согласно стратегии “Параллельное галсирование”

## 3.1. АЛГОРИТМ ПОСТРОЕНИЯ МАРШРУТА ПРИ ФИКСИРОВАННОМ РАСПРЕДЕЛЕНИИ

Для построения маршрута в случае сохранения частицами их положений будет использован метод динамического программирования. Далее будут описаны состояния алгоритма, его переходы и вспомогательные значения посчитанные на матрице распределения.

Прежде всего введем сетку на прямоугольнике на котором рассматривается распределение. Пусть размер сетки будет  $H \times W$ , размеры ячеек будут одинаковыми, но не обязаны быть квадратными или выровненными по пикселям текстуры. В частности мы будем стремиться к очень высокому разрешению по строчкам для более точного определения расстояния между соседними галсами и нам будет достаточно небольшого разрешения по столбцам. Например если размеры текстуры  $T_H \times T_W$ , а размеры прямоугольника в мире  $R_H \times R_W$ , то мы хотим иметь размеры сетки  $H = T_H$  и  $W \approx \frac{R_W}{1000}$ .

### 3.1.1. Состояния алгоритма

Значением динамического программирования  $dp(state)$  для определенного состояния будет суммарный вес собранных частиц к моменту времени определяемому состоянием. На самом деле храниться будет весь суммарный вес за исключением покрытого лишь передним полукругом текущего положения.

Состоянием динамического программирования является кортеж

(*cntHor*, *row*, *col*, *move*, *notCleared*).

- (*row*, *col*) — строка и столбец ячейки в которой закончен маршрут к текущему моменту времени соответственно.
- *cntHor* — количество горизонтальных перемещений (между столбцами). Если обозначить за  $time_w$  — время перемещения между двумя соседними ячейками по горизонтали, а за  $time_H$  — по вертикали, то текущее время можно посчитать как  $curTime = time_H \cdot row + time_w \cdot cntHor$ . Здесь использованы предположения, что средство поиска никогда не возвращается назад по строчками (исходя из выбранной стратегии поиска), всегда двигаемся с максимальной скоростью и в начальный момент времени находимся в нулевой строке.
- *move* — тип текущего хода (который завершится в этом состоянии). Принимает значение из множества  $\{L, R, LU, RU\}$ .  $L/R$  — означает, что мы галсируем в текущей строчке влево или вправо соответственно (галсировать в обе стороны в одной строчке нельзя).  $LU/RU$  — означает, что мы поднимаемся по строчкам, причем в последней строчке в которой мы галсировали мы двигались влево или вправо соответственно. Уточнение в какую сторону мы двигались последний раз необходимо для более точного учета собранных нами частиц во время подъема по строчкам.
- *notCleared* — количество пропущенных строк без галсирования. Главное предположение, позволяющее значительно сократить количество состояний, состоит в том, что для понимания какие из частиц еще не были собраны в текущей полосе галсирования, нам достаточно знать когда был предыдущий галс или другими словами сколько строк мы пропустили без галсирования. Зная какие из частиц не собраны, мы сможем рассчитать сколько частиц мы соберем в теку-



щий момент прохождения маршрута. Данный параметр принимает  $N$  значений. 0 означает, что мы галсировали прямо в предыдущей строчке,  $(N - 1)$  — последний раз мы галсировали  $N$  строк назад либо еще дальше.  $N$  выбирается таким образом, чтобы галсирование  $N$  строк назад не оказывало значительного влияния на текущее.

### 3.1.2. Вспомогательные матрицы с частичными суммами весов

Обозначим радиус зоны видимости средства поиска за  $r$ . Обозначим функции, переводящие из системы координат построенной сетки в мировую систему координат, за  $w_x(col)$ ,  $w_y(row)$ .

Предположим, что вес частиц в каждом пикселе распределен равномерно. Тогда с помощью сумм на префиксе, построенных на текстуре за время  $O(T_W \cdot T_H)$ , можно за  $O(1)$  отвечать на запросы вида сумма на прямоугольнике  $[x_l; x_r] \times [y_l; y_r]$ . При вычислении суммы используются линейная интерполяция на пикселях частично попавших в прямоугольник запроса.

Для подсчета веса частиц, собранных при прохождении маршрута, маршрут будет разделен на части разного типа (рис. 3.1). Как видно из картинки некоторые части могут быть частично учтены ранее, поэтому важно знать значение *notCleared* — сколько строк было пропущено без галсирования. Таким образом все предсчитанные матрицы будут возвращать значения с учетом *notCleared*. Самое большое значение *notCleared* равное  $N - 1$  будет использовано также в случаях, когда требуется считать, что последний галс был достаточно далеко, чтобы не оказывать никакого влияния в текущем положении.

Для примера рассмотрен переход между двумя галсами, содержащий части всех возможных типов. В примере ширина ячейки в два раза больше ее высоты. Примеры ячеек сетки динамики изображены красными прямоугольниками. Красная точка в центре ячейки сетки обозначает поло-



Рис. 3.1: Типы зон при прохождении маршрута

жения средства поиска при нахождении в этом состоянии динамики. Для упрощения пусть точка с координатами  $(row, col)$  совпадает с центром соответствующей ячейки. Тогда область определения  $w_x \in [-0.5; M - 0.5]$ , а  $w_y \in [-0.5; N - 0.5]$ . Желтым отмечен маршрут. Разные типы частей помечены различными цветами. Черная сетка на фоне может интерпретироваться как сетка пикселей.

Рассмотрим вертикальные части синего цвета. Значение матрицы  $vl[row][col][notCleared]$  — будет содержать сумму в прямоугольнике  $[w_x(col - 1), w_x(col)] \times [w_y(row) - r, w_y(row) + r]$ . Без учета частиц собранных ниже прямой  $w_y(row - 1 - notCleared) + r$ . А  $vr[row][col][notCleared]$  — будет содержать  $[w_x(col), w_x(col + 1)] \times [w_y(row) - r, w_y(row) + r]$ .

Розовый и оранжевый тип отличаются лишь положе-

нием относительно маршрута, справа или слева соответственно. Матрица для оранжевого  $hl[row][col][notCleared]$  — содержит сумму в  $[w_x(col) - r, w_x(col)] \times [w_y(row - 1), w_y(row)]$ . Матрица для розового  $hr[row][col][notCleared]$  — содержит  $[w_x(col), w_x(col) + r] \times [w_y(row - 1), w_y(row)]$ .

Обратим внимание на область, покрытую одновременно оранжевым и синим цветом в верхней части картинки. До тех пор пока мы не повернули налево для прохождения нового галса — эта область будет учтена как оранжевого типа. После поворота она будет переучтена как область синего. Для исключения первого ошибочно посчитанного раза необходимо предсчитать матрицу  $ol[row][col][notCleared]$  — область  $[w_x(col) - r, w_x(col)] \times [w_y(row) - r, w_y(row)]$  и матрицу  $or[row][col][notCleared]$  — область  $[w_x(col), w_x(col) + r] \times [w_y(row) - r, w_y(row)]$ .

Последним типом являются четверти круга (зеленый цвет), которые должны быть учтены при поворотах. Суммарный вес под ними будет хранить матрица  $cs[row][col][quarter][notCleared]$ . *quarter* — обозначает четверть и будет принимать значения  $Q_{RU}, Q_{LU}, Q_{LD}, Q_{RD}$  соответственно для четвертей в порядке их стандартной нумерации. Для вычисления суммарного веса, находящегося под четвертью круга, используется численное интегрирование посредством вычисления сумм на прямоугольниках.

### 3.1.3. Переходы между состояниями

Переходы из состояния динамики

$$state = (cntHor, notCleared, row, col, curMove)$$

будут описаны в таблице 3.1 следующим образом. Текущий ход *curMove* будет зафиксирован в первом столбце, следующий выбранный во втором. *newState* — состояние в которое мы перейдем, выбрав такой ход, указано в четвертом столбце. Все величины, которые нужно добавить при переходе

к величине  $dp(state)$  перед релаксацией значения  $dp(newState)$  указаны в третьем столбце. Если обозначить эти величина за  $a_i, i \in 1..k$ , то более формально  $dp(newState) = \min(dp(newState), dp(state) + \sum_{i=1}^k a_i)$ .

Текущий ход	Следующий ход	Собрано на текущем ходу	Новое состояние
$L$	$L$	$vr[row][col - 1][notCleared]$	$(cntHor + 1, notCleared, row, col - 1, L)$
$L$	$LU$	$hl[row + 1][col][N - 1]$ $hr[row + 1][col][0]$ $cs[row][col][Q_{LD}][notCleared]$	$(cntHor, 0, row + 1, col, LU)$
$R$	$R$	$vl[row][col + 1][notCleared]$	$(cntHor + 1, notCleared, row, col + 1, R)$
$R$	$RU$	$hr[row + 1][col][N - 1]$ $hl[row + 1][col][0]$ $cs[row][col][Q_{RD}][notCleared]$	$(cntHor, 0, row + 1, col, RU)$
$LU$	$LU$	$hl[row + 1][col][N - 1]$ $hr[row + 1][col][notCleared + 1]$	$(cntHor, notCleared + 1, row + 1, col, LU)$
$LU$	$L$	$vr[row][col - 1][notCleared]$ $cs[row][col][Q_{RU}][notCleared]$ $-ol[row][col][notCleared]$	$(cntHor + 1, notCleared, row, col - 1, L)$
$LU$	$R$	$vl[row][col + 1][notCleared]$ $cs[row][col][Q_{LU}][N - 1]$ $-or[row][col][notCleared]$	$(cntHor + 1, notCleared, row, col + 1, R)$
$RU$	$RU$	$hr[row + 1][col][N - 1]$ $hl[row + 1][col][notCleared + 1]$	$(cntHor, notCleared + 1, row + 1, col, RU)$
$RU$	$L$	$vr[row][col - 1][notCleared]$ $cs[row][col][Q_{RU}][N - 1]$ $-ol[row][col][notCleared]$	$(cntHor + 1, notCleared, row, col - 1, L)$
$RU$	$R$	$vl[row][col + 1][notCleared]$ $cs[row][col][Q_{LU}][notCleared]$ $-or[row][col][notCleared]$	$(cntHor + 1, notCleared, row, col + 1, R)$

Таблица 3.1: Переходы из состояния  $(cntHor, notCleared, row, col, curMove)$

### 3.1.4. Оценка времени работы

Оценим количество операция для построения вспомогательных матриц. Подсчет значений в матрицах  $hl, hr, ol, or, vl, vr$  можно осуществить

за  $O(1)$  операций, если предсчитать префикс суммы на текстуре за  $O(T_W \cdot T_H)$ . Количество значений в каждой из этих матриц  $O(W \cdot H \cdot N)$  или при грубой оценке  $N = O(W^2 \cdot H)$ .

Из каждого состояния динамики осуществляется  $O(1)$  переходов, таким образом для оценки времени работы алгоритма достаточно посчитать количество состояний. Если обозначить за  $T = \frac{t_{search}}{time_W}$ , то количество состояний составит  $O(T \cdot W \cdot H \cdot N)$ . Однако в большинстве случаев время достаточное для обследования всей территории с вероятностью близкой к 100% составляет  $W \cdot \frac{H}{N}$ . Таким образом количество состояний составит  $O(W^2 \cdot H^2)$ .

Максимальную точность позволяет получить значение  $H \approx T_H$ , однако на практике оказывается достаточным  $H = \frac{T_H}{4}$ , что также позволяет алгоритму работать достаточно быстро.

На практике достижимых состояний оказывается втрое меньше их общего числа. Одна из причин этого то, что не имеет смысла проводить больше времени чем нужно на каком-либо префиксе пути. Если все частицы на этом префиксе можно собрать за определенное время  $\tau$ , состояния, которые тратят больше времени, оказываются лишними. Вторая причина противоположная — до некоторых положений невозможно дойти за время меньше некоторого определенного.

## 3.2. КОРРЕКТИРОВКА МАРШРУТА

Вернемся к изначальной постановке задачи, когда частицы могут изменять свое местоположение согласно некоторой модели. На рис. 3.2 изображен маршрут построенный рассмотренным алгоритмом. Изменение распределения следует комбинации моделей случайного блуждания и приближения к вертикальной прямой (берегу), находящейся слева от распределения. Таким образом к середине прохождения маршрута становится заметно

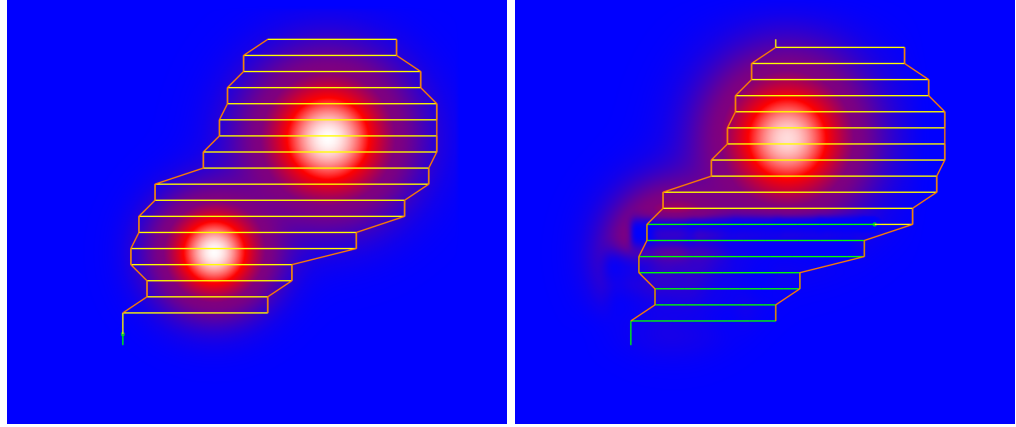


Рис. 3.2: Маршрут построенный на начальном распределении

Рис. 3.3: Маршрут оказался неудовлетворительным с течением времени

(рис. 3.3), что положения частиц изменились настолько, что текущий планируемый маршрут оказался неудовлетворительным.

Предлагаемым решением является перестройка оставшейся части маршрута (рис. 3.4), в момент времени когда становится понятно, что текущий маршрут не получит достаточного суммарного веса.

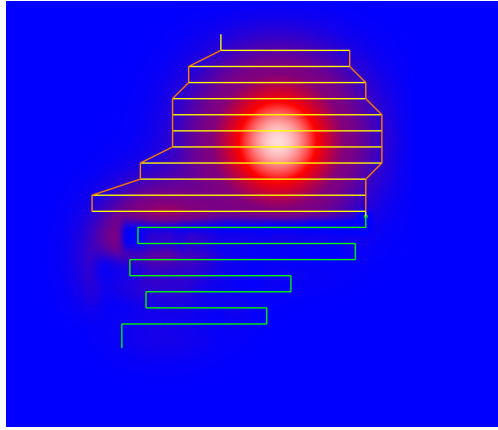


Рис. 3.4: Оставшаяся часть маршрута была перестроена

Рассмотрим критерий необходимости перестройки маршрута в момент времени  $t_i$ , если последний раз маршрут перестраивался в момент времени  $t_{i-1}$ . Разделим построенный маршрут на две части:  $path_{t_{i-1} \leq t \leq t_i}$  — уже пройденная часть маршрута,  $path_{t \geq t_i}$  — запланированная часть маршрута

рута. Далее рассмотрим два симулятора прохождения маршрута в которых частицы не изменяют своего местоположения со временем. Симулятор  $D_0$  содержит распределение в момент времени  $t_{i-1}$  на котором был просимулирован маршрут  $path_{t_{i-1} \leq t \leq t_i}$ . Симулятор  $D_1$  содержит распределение в момент времени  $t_i$ . На каждом из симуляторов оценивается суммарный вес, который будет собран на маршруте  $path_{\geq t_i}$ . Пусть это будут величины  $sum_0$  и  $sum_1$  соответственно. Маршрут должен быть перестроен в случае, если  $sum_1 \leq p \cdot sum_0$ . В реализации выбрано значение  $p = 0.98$ .

Данный критерий корректировки, как будет показано далее, позволяет достаточно успешно корректировать маршрут для сбора достаточного веса частиц при активно изменяющемся положении частиц.

# Глава 4. Сравнение с существующими решениями

## 4.1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Был разработан алгоритм строящий маршруты поиска, которые удовлетворяют фиксированным паттернам стратегии “Параллельное галсирование”. Задача построения маршрутов имеющих определенную структуру достаточно специфична. К сожалению, методы, решающие поставленную задачу (или ее частные случаи) не были найдены в открытых источниках.

Однако существует комплекс инструментов, ранее разработанный в “Кронштадт Технологии”, который предоставляет возможность строить маршруты разными стратегиями поиска, несколькими поисковыми средствами, оптимизировать параметры совместного поиска и некоторые другие возможности. Рассмотренный в данной работе метод должен прийти на замену существующему в данном комплексе, поэтому в первую очередь должно быть осуществлено сравнение с этим методом.

Далее будет произведено сравнение с существующим алгоритмом построения маршрутов стратегией “Параллельное галсирование”.

## 4.2. СРАВНЕНИЕ СО СТАРЫМ АЛГОРИТМОМ ПОСТРОЕНИЯ МАРШРУТОВ “КРОНШТАДТ ТЕХНОЛОГИИ”

Старый алгоритм имеет немного другой интерфейс. На вход принимается район в котором необходимо осуществить галсирование. Единственная поддерживаемая модель изменения распределения — случайное



блуждание с определенной интенсивностью  $I$ . Интенсивность  $I$  определяет сколько времени потребуется частицам, чтобы не менее  $p\%$  отдалились на единицу длины. Исходя из интенсивности рассчитывается расстояние между галсами — одинаковое на всем протяжении маршрута. Причем допускается пропуск частиц суммарного веса  $\epsilon$  на каждом галсе.

Сравнение будет происходить следующим образом. Тест 1 продемонстрирует несостоятельность старого метода в условии другой модели изменения распределения. Тест 2 покажет преимущества нового метода при неравномерных распределениях. Все последующие тесты будут иметь равномерное распределение и модель изменения — случайное блуждание. Тест 3 покажет важность корректировки маршрута со временем, так как район расширяется и ширина галса со временем должны быть увеличена. Тест 4 проверит способность нового метода подстраивать расстояние между галсами в простейшем случае, когда частицы не передвигаются. Кроме того тест 4 покажет способность нового метода учитывать зоны, досматриваемые во время движения вверх.

В старом методе время поиска не фиксируется, а возвращается по результату работы алгоритма. Таким образом для каждого теста новым алгоритм будет построено два маршрута. Первый маршрут будет демонстрировать лучший результат, занимая столько же времени сколько маршрут старого алгоритма. Если времени данного старым алгоритмом недостаточно для построения хорошего маршрута в данном случае — будет построен маршрут с большим временем поиска. Второй маршрут будет демонстрировать лучшее время, при этом не теряя в результативности по сравнению со старым алгоритмом.

Для каждого маршрута будет показана статистика — прогресс и поисковая производительность на протяжении всего времени поиска. Синий график поисковой производительности имеет относительный масштаб и может быть использован только для сравнения поисковой производитель-

ности в рамках одного поиска (если его абсолютная величина на одном из двух графиков меньше — это не значит, что поисковая производительность в этом запуске хуже чем во втором).

#### **4.2.1. Тест 1**

Воспользуемся моделью изменения распределения — притяжение к определенным точкам. Пусть частицы стремиться приблизиться к берегу, расположенному слева от района поиска. Новый алгоритм скорректирует маршрут, в то время как старый продолжит обследовать область их начально местоположения.

Старый алгоритм запросил 3.6 часа и собрал 85.7% суммарного веса. За это же время новый алгоритм собрал 94.7%, а для сбора 85.7% ему потребовалось менее 2.9 часа.

#### **4.2.2. Тест 2**

Модель изменения распределения — случайное блуждание. В левом нижнем и правом верхнем углу поля расположены нормальные распределения. Вес частиц значительно убывает при удалении от центров распределений, следовательно поиск в тех местах не является целесообразным. Старый алгоритм этого не учитывает.

Таким образом старый алгоритм по прежнему тратит 3.6 часа, собирая при этом 83.3%. За это же время более аккуратный новый алгоритм собирает 92.6%. Для сбора старого результата новому будет достаточно почти вдвое меньшего времени 1.9 часа.

#### **4.2.3. Тест 3**

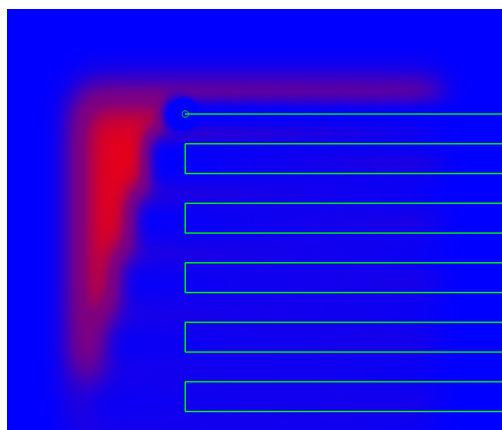
Модель изменения распределения — интенсивное случайное блуждание. Таким образом район увеличивается со временем и построенный

изначально маршрут будет покрывать все меньше частиц с течением времени.

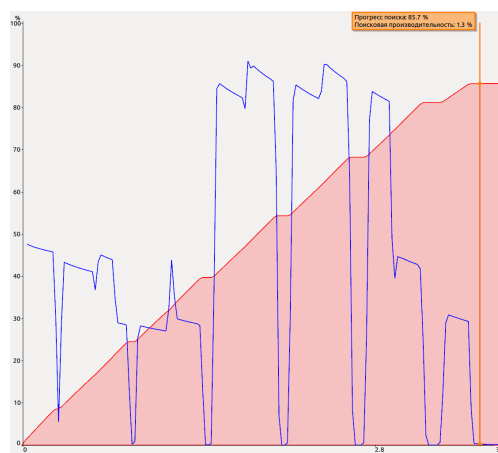
Старый алгоритм за 3.6 часа собирает 83.3%, проходя по наиболее вероятным местам. Новый алгоритм за то же время может собрать такой же вес. Однако если увеличить время поиска для нового алгоритма, он начнет собирать частицы, отдаляющиеся от начального района.

#### 4.2.4. Тест 4

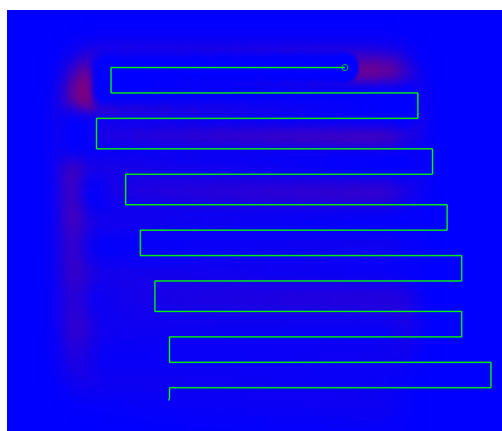
В данном тесте частицы не передвигаются со временем. Старый алгоритм, ожидаемо, рассчитывает верное расстояние между галсами и собирает все частицы за время 3.6 часа. За это же время новый алгоритм при разрешении по строкам  $T_H$ , правильно выбрав расстояние, также собирает все частицы. Однако с незначительными потерями (0.3%) новый алгоритм может сэкономить много времени на поворотах (до четверти часа), учитывая то, что на поворотах частицы также собираются.



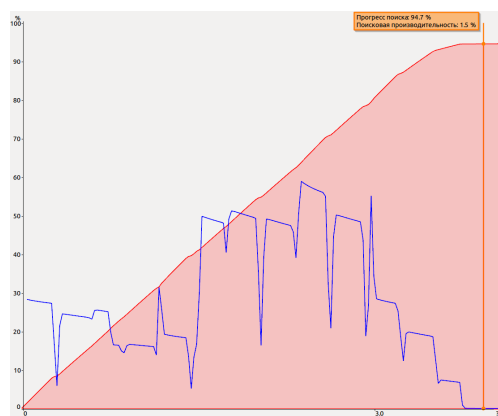
(a) Старый алгоритм



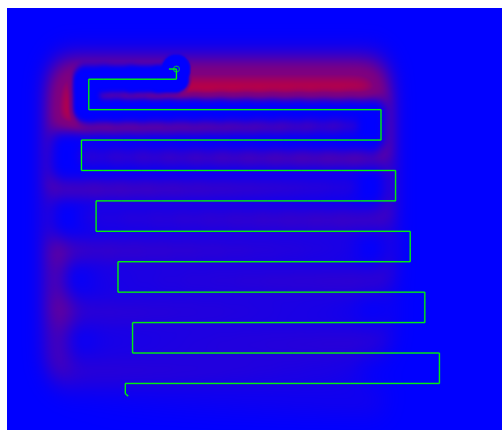
(b) Статистика



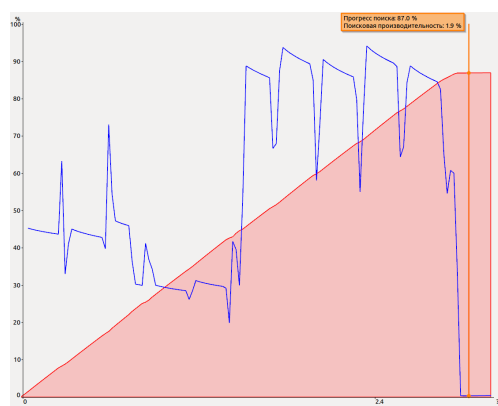
(c) Новый — сравнение по результату



(d) Статистика

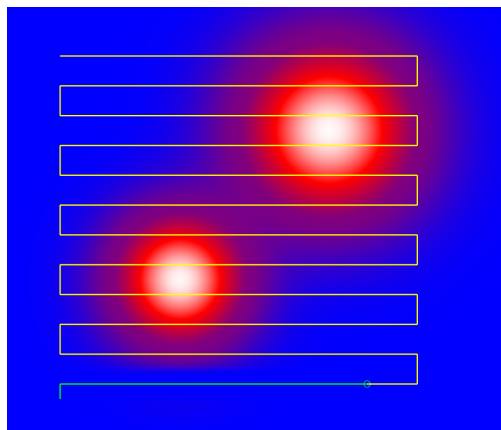


(e) Новый — сравнение по времени

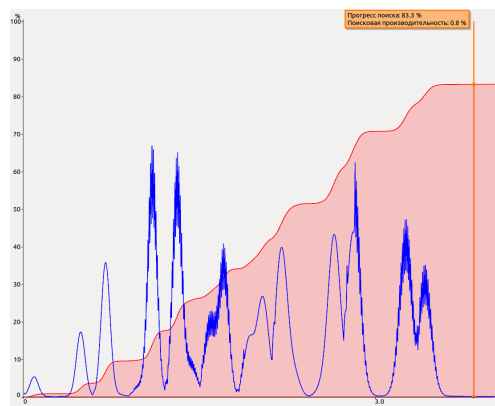


(f) Статистика

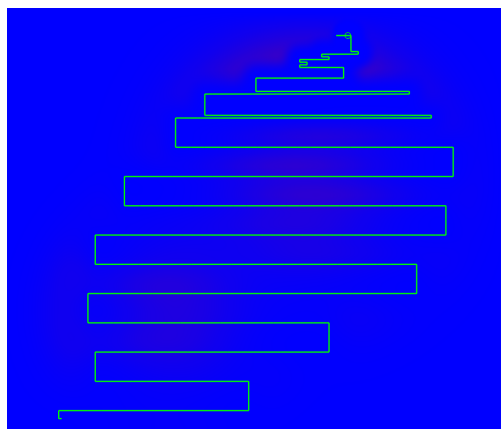
Рис. 4.1: Тест 1. Модель изменения — приближение к прямой слева



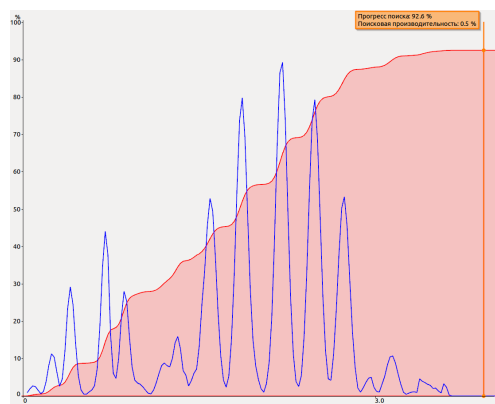
(a) Старый алгоритм



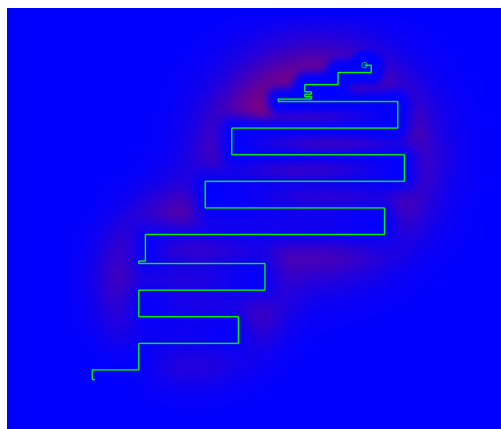
(b) Статистика



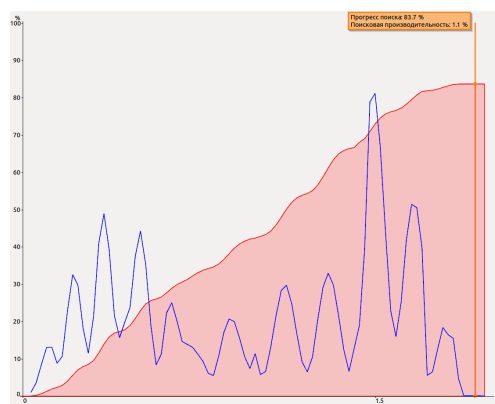
(c) Новый — сравнение по результату



(d) Статистика



(e) Новый — сравнение по времени

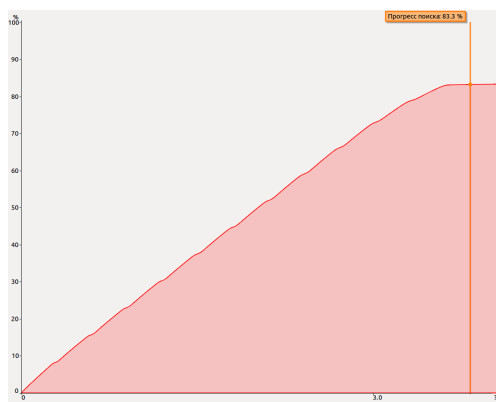


(f) Статистика

Рис. 4.2: Тест 2. Начальное распределение — композиция нормальных



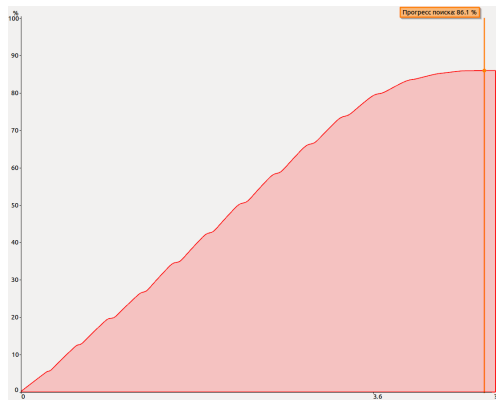
(a) Старый алгоритм



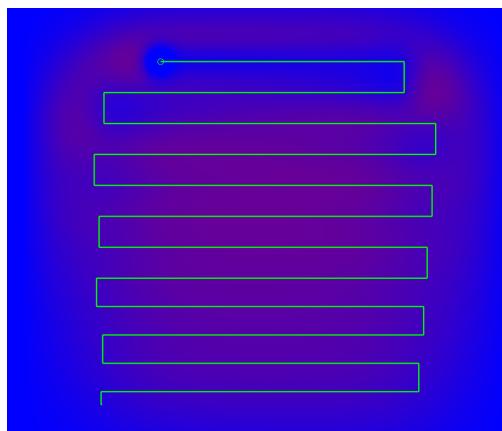
(b) Статистика



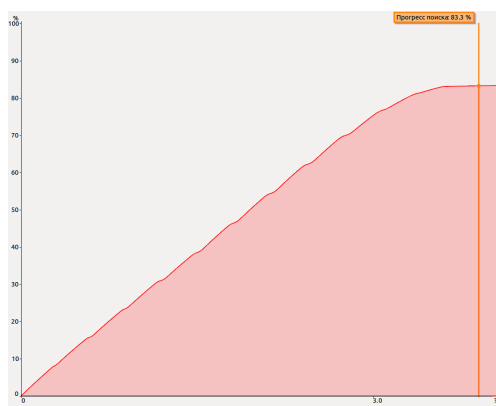
(c) Новый — сравнение по результату



(d) Статистика

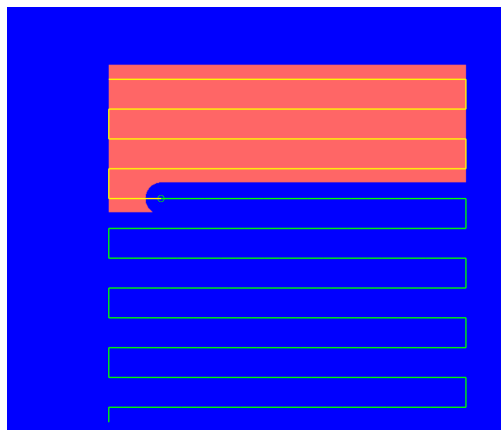


(e) Новый — сравнение по времени

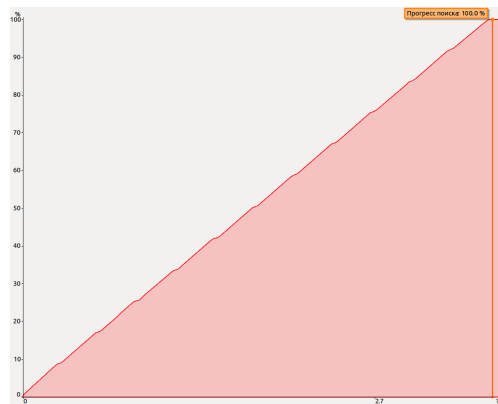


(f) Статистика

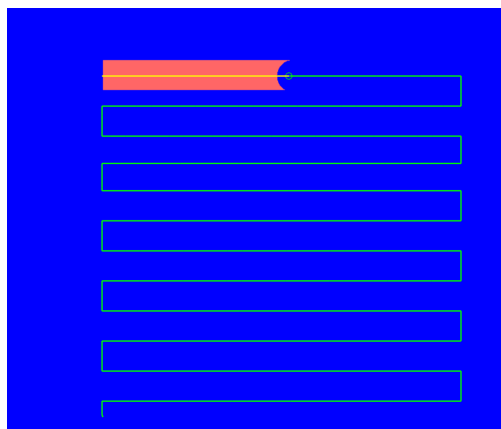
Рис. 4.3: Тест 3. Случайное блуждание — район увеличивается со временем



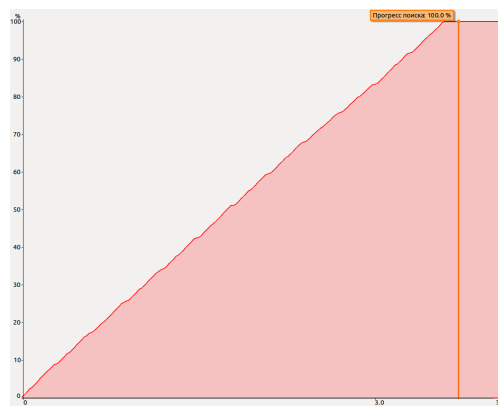
(a) Старый алгоритм



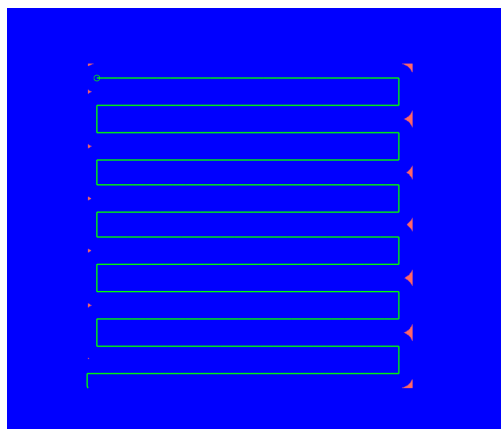
(b) Статистика



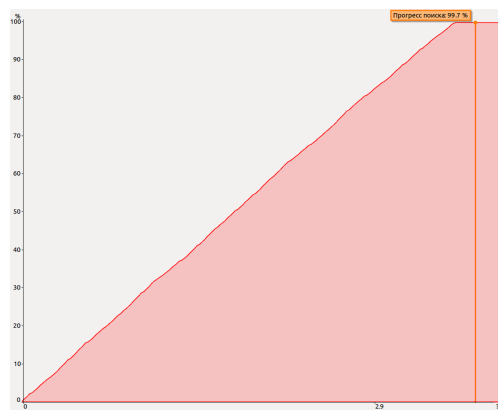
(c) Новый — сравнение по результату



(d) Статистика



(e) Новый — сравнение по времени



(f) Статистика

Рис. 4.4: Тест 4. Частицы не передвигаются с течением времени

#### 4.2.5. Результаты

Далее приведена сводная таблица результатов тестов. Во втором столбце указано время, которое занимает прохождение маршрута, построенного старым алгоритмом. В третьем столбце указан результат, получаемый при прохождении старого алгоритма. В четвертом столбце указан результат нового алгоритма при оптимизации по результату. Если не указано значения времени в скобках — значит данный результат получен за то же время что и результат старого алгоритма. В последнем столбце указан результат нового алгоритма при минимизации времени. Если не указано значения в скобках — значит за данное время получен результат не хуже результата старого алгоритма.

Номер теста	Время старого	Результат старого	Сравнение по результату	Сравнение по времени
1	3.6	87.5%	94.7%	2.9
2	3.6	83.3%	92.6%	1.9
3	3.6	83.3%	86.1% (4.5)	3.6
4	3.6	100.0%	100.0%	3.3 (99.7%)

Таблица 4.1: Сводная таблица результатов работы на тестах



# Заключение

В данной работе был разработан алгоритм, построения маршрутов поиска стратегией “Параллельное галсирование”. Алгоритм позволяет работать с неравномерными распределениями и различными моделями изменения распределений со временем.

В качестве недостатков алгоритма, которые подлежат дальнейшему улучшению, нужно отметить большое время работы при максимальном разрешении сетки по строкам  $T_H$ . Хотя такое разрешение необходимо для наиболее точной подстройки расстояния между галсами, на практике удастся обойтись меньшим разрешением без значительных потерь в результативности, при этом время работы снижается до приемлимого уровня.

Для целей визуализации и в качестве вспомогательного средства для корректировки маршрутов со временем был разработан симулятор прохождения маршрутов. Симулятор основан на технологии CUDA, таким образом при высоком разрешении поля симуляции удастся поддерживать достаточную скорость работы для визуализации прохождения маршрута в ускоренном режиме.

В качестве дальнейших улучшений симулятора можно рассмотреть повышение разрешения, посредством более продвинутых техник осуществления свертки изображения (Image Convolution) на GPU.

Разработанный алгоритм должен заменить существующий алгоритм “Кронштадт технологии”, сравнение с которым было осуществлено в главе 4. Было показано, что разработанный алгоритм не уступает старому и значительно превосходит его в случае неравномерных распределений и моделей их изменения отличных от случайных блужданий.

# Источники

- [1] The Traveling Salesman Problem / D. L. Applegate, R. E. Bixby, V. Chvátal et al. Princeton University Press, 2007.
- [2] Transformations of Generalized ATSP into ATSP / D.Ben-Arieh, G.Gutin, M.Penn et al. // Elsevier Science. Operations Research Letters. 2003. Vol. 31. P. 357–365.
- [3] Heuristics for the traveling repairman problem with profit / T.Dewilde, D.Cattrysse, S.Coene et al. // Computers & Operations Research. 2013. Vol. 40. P. 1700–1707.
- [4] Li C., Yang M., Kang L. A New Approach to Solving Dynamic Traveling Salesman Problems // Springer Science. Lecture Notes in Computer Science. 2006. Vol. 4247. P. 236–243.
- [5] The Travelling Salesman Problem with Time Windows: Adapting Algorithms from Travel-time to Makespan Optimization: Tech. Rep.: TR/IRIDIA/2013-011 / M. Lopez-Ib´a´nez, C. Blumb, J. W. Ohlmann et al.: IRIDIA, Institut de Recherches Interdisciplinaires et de D´eveloppements en Intelligence Artificielle. Universite Libre de Bruxelles, 2013.
- [6] Hammar M., Nilsson B. J. Approximation Results for Kinetic Variants of TSP // Springer Science. Lecture Notes in Computer Science. 1999. Vol. 1644. P. 392–401.
- [7] Helvig C., Robins G., Zelikovskiy A. The moving-target traveling salesman problem // Elsevier Science. Journal of Algorithms. 2003. Vol. 49. P. 153–174.
- [8] (Выписка из НАПСС-90) Основные принципы организации поисково-спасательного обеспечения полетов авиации. [http://aviaclub.ru/uploads/media/PSO\\_poljotov.pdf](http://aviaclub.ru/uploads/media/PSO_poljotov.pdf). 1990.
- [9] Novak Jan, Kaplanyan Anton, Liktov Gabor [и др.]. GPU Computing: Image Convolution. 2012.
- [10] R.C.Gonzalez, R.E.Woods. Digital Image Processing. Pearson Education Inc., 2002.
- [11] Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA / S. Ryoo, C. Rodrigues, S. Bagsorkhi et al. // PPOPP’08. 2008. February. P. 73–82.