МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

# РАЗРАБОТКА ПРОГРАММНОГО КОМПОНЕНТА МАРШРУТИЗАЦИИ СОЕДИНЕНИЙ ДЛЯ ВИЗУАЛЬНЫХ СРЕД МОДЕЛИРОВАНИЯ БЛОЧНЫХ И ФУНКЦИОНАЛЬНЫХ СХЕМ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 411 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Шестова Петра Николаевича

Научный руководитель
доцент, к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2026

# ВВЕДЕНИЕ

Структурные функциональные схемы являются одним из распространенных способов наглядного описания вычислительных, логических и управляющих процессов. Они применяются в инженерном моделировании, цифровой обработке сигналов, автоматизированных системах управления и инструментах визуального программирования. В таких схемах функциональные блоки размещаются на плоскости, а передача данных между ними задается соединительными линиями, которые связывают выходы одних элементов со входами других.

При увеличении числа блоков и связей визуальное представление схемы усложняется. Линии могут пересекаться, накладываться друг на друга, проходить слишком близко к элементам или образовывать трудновоспринимаемые человеком узлы. Из-за этого затрудняется анализ схемы, ее модификация и сопровождение, а также повышается вероятность ошибок при проектировании. При ручном построении пользователь вынужден самостоятельно следить за расположением линий, что становится трудоемким и не гарантирует устойчивого результата при перемещении блоков. Соответственно, возникает необходимость в автоматическом построении и оптимизации маршрутов соединительных линий. Целью работы является разработка программного компонента в виде динамической библиотеки (DLL) для автоматической маршрутизации соединительных линий в структурных блочных схемах, а также создание тестовой визуализации для демонстрации и проверки работы разработанного компонента.

Для достижения поставленной цели решались следующие задачи:

- проанализировать особенности представления структурных блочных схем и соединительных линий;
- спроектировать формат входных данных, включающий описание блоков, портов, соединений и параметров маршрутизации;
- спроектировать формат выходных данных, включающий координаты построенных линий и информацию о результате маршрутизации;
- разработать алгоритм построения ортогональных маршрутов с учетом геометрии блоков и существующих соединений;
- реализовать программный компонент в виде динамической библиотеки (DLL) со стандартизированным API;
- разработать адаптеры для передачи данных между внешним приложением

и DLL;

— реализовать тестовое приложение для визуализации блоков, портов и построенных соединительных линий.

**Краткое содержание работы**

Задача оптимизации соединительных линий в структурных схемах тесно связана с направлением визуализации графов и разведения связей на плоскости, и в научной литературе ей уделяется значительное внимание. Одной из ключевых теоретических работ в области ортогональной трассировки является исследование Р. Тамассиа, посвященное построению ортогональных сеточных представлений графов с минимальным числом изгибов ребер; в дальнейшем Дж. Ди Баттиста, П. Эйдс, Р. Тамассиа и И. Толлис систематизировали основные методы визуализации графов, включая алгоритмы ортогональной и иерархической отрисовки. Среди алгоритмов поиска пути на сетке классическим является волновой алгоритм Ли, применяемый при трассировке соединений в электронных схемах; его развитием стали более эффективные методы — алгоритм Хэдлока, сокращающий число рассматриваемых узлов за счет эвристики, и модификация Ф. Рубина, вводящая стоимостную функцию. Для задач с несколькими критериями качества маршрута применяются алгоритмы поиска пути с весами, такие как алгоритм Дейкстры, позволяющие задавать стоимость переходов с учетом длины маршрута, числа перегибов, близости к препятствиям и пересечений с уже существующими линиями. Поскольку алгоритм Дейкстры ведет ненаправленный поиск, на практике удобнее его модификация — алгоритм A*, формализованный в работе П. Харта, Н. Нильссона и Б. Рафаэля, который использует эвристическую оценку расстояния до цели и тем самым уменьшает количество рассматриваемых состояний, что делает его подходящим для автоматической маршрутизации линий на сетке.

Помимо теоретической базы, был рассмотрен ряд программных решений: библиотека автоматической компоновки графов Eclipse Layout Kernel, редактор структурных схем и UML-диаграмм draw.io, системы проектирования печатных плат KiCad и Horizon EDA с интерактивными роутерами, внешний трассировщик OrthoRoute на основе итерационного алгоритма PathFinder, а также встроенные средства оптимизации соединений в MATLAB Simulink и NI LabVIEW. Eclipse Layout Kernel поддерживает различные стили размещения и маршрутизации ребер, включая ортогональные соединения и работу с портами, однако ее интеграция требует согласования модели данных приложения с собственной моделью библиотеки, что может оказаться избыточным для задач, где нужен компактный DLL-компонент. В KiCad и Horizon EDA применяется интерактив-

ная маршрутизация, основанная на расталкивании уже проложенных дорожек и обходе препятствий в режиме реального времени, что хорошо подходит для ручной и полуавтоматической трассировки печатных плат, но ориентировано прежде всего на предметную область, где существенную роль играют физические ограничения дорожек, зазоры и слои платы. Трассировщик OrthoRoute, в свою очередь, показывает эффективность ортогональных сеточных моделей на сверхплотных платах, распределяя тысячи соединений на регулярной манхэттенской сетке с аппаратным ускорением, однако его реализация ориентирована на многослойные топологии и использует специфические программные и аппаратные механизмы. Наконец, средства оптимизации в Simulink и LabVIEW обеспечивают высокое качество представления схем, но их алгоритмы, как правило, являются закрытыми либо тесно связаны с внутренней моделью данных конкретной среды. Анализ показал, что эти решения предоставляют развитую теоретическую и практическую базу, однако многие из них либо ориентированы на конкретную предметную область, либо требуют согласования с собственной моделью данных, либо являются закрытыми и недоступными для прямой интеграции. Это обосновывает необходимость разработки независимого компонента в виде DLL, который предоставляет формализованный интерфейс входных и выходных данных и может быть встроен в различные моделирующие среды.

Формально структурная схема, используемая в визуальных средах моделирования, может быть представлена в виде графа

$$G = (V, E), \tag{1}$$

где множество вершин $V$ соответствует функциональным блокам схемы, а множество ребер $E$ задает направленные соединения между ними. Каждый блок имеет прямоугольную геометрическую область и набор портов, расположенных на его границе; порт характеризуется координатой на сцене, направлением подключения линии и типом — входным или выходным. Каждое соединение связывает выходной порт одного блока с входным портом другого, и задача маршрутизации состоит в том, чтобы для каждого соединения построить геометрический маршрут в виде ортогональной полилинии

$$R_k = \{q_1, q_2, \ldots, q_m\}, \tag{2}$$

первая точка которой совпадает с координатой начального порта, а последняя — с координатой конечного. Для каждого сегмента маршрута при этом должно выполняться условие ортогональности

$$\forall i \in \{1, \ldots, m-1\}: \quad (x_i = x_{i+1}) \vee (y_i = y_{i+1}), \tag{3}$$

а для корректного подключения линии к блоку используются вспомогательные точки выхода из портов, расположенные на небольшом расстоянии от порта в направлении его нормали, благодаря чему линия сначала выходит из блока перпендикулярно его стороне и лишь затем меняет направление.

Поиск маршрута выполняется не по всей непрерывной плоскости, а по дискретной сетке, которая строится как равномерная решетка с заданным шагом и дополняется значимыми координатами объектов схемы: координатами портов, точек выхода, границ блоков и уже построенных сегментов линий. Блоки рассматриваются как препятствия, причем их прямоугольные области могут расширяться на небольшой отступ, чтобы линия проходила не вплотную к блоку. Уже существующие линии учитываются не как абсолютно запрещенные области, а как нежелательные ситуации, увеличивающие стоимость маршрута, поскольку полное исключение пересечений в сложных схемах может привести к отсутствию маршрута; жестким ограничением является лишь запретная зона вокруг чужих портов. Само построение выполняется алгоритмом A*, в котором состояние поиска задается текущей точкой сетки и направлением прихода в нее, что необходимо для начисления штрафа за поворот. Задача формулируется как поиск маршрута с минимальной стоимостью

$$R_k = \arg\min_{R} F(R), \tag{4}$$

где функция стоимости $F(R)$ объединяет несколько требований к качеству маршрута: длину линии, число перегибов, пересечения с уже существующими соединениями, прохождение через области препятствий и близкое параллельное наложение на другие линии. В качестве эвристики используется манхэттенское расстояние до целевой точки

$$h(q_i) = |x_i - x_t| + |y_i - y_t|, \tag{5}$$

которое соответствует ортогональному характеру маршрута и направляет поиск в сторону конечного порта. Для множества соединений требуется построить набор маршрутов

$$\mathcal{R} = \{R_1, R_2, \ldots, R_{|E|}\}, \tag{6}$$

причем соединения строятся не одновременно, а последовательно в порядке возрастания манхэттенского расстояния между их портами, и весь набор обрабатывается за несколько проходов, на каждом из которых линии перестраиваются с учетом текущей геометрии остальных, пока расположение не стабилизируется.

Следует отметить, что в общем случае задача поиска ортогонального вложения графа на сетке с минимальным числом изгибов является NP-трудной, что исключает построение точного оптимального решения за полиномиальное время и обосновывает применение эвристических методов. Если область маршрутизации имеет ширину $W$, высоту $H$, а шаг регулярной части сетки равен $g$, то число узлов оценивается как

$$N \approx \frac{WH}{g^2}, \tag{7}$$

откуда видно, что уменьшение шага сетки повышает точность, но квадратично увеличивает число узлов поиска. Поскольку состояние задается парой «узел сетки — направление», число состояний и переходов имеет порядок $O(N)$, а при реализации A* на основе бинарной кучи базовая сложность построения одного маршрута составляет $O(N \log N)$. При наивной проверке взаимодействия нового маршрута с $M$ существующими сегментами на каждом переходе сложность возросла бы до $O(N \log N + NM)$. В разработанном решении эта проверка вынесена в отдельный этап предварительной разметки карт ребер сетки, поэтому стоимость одного перехода становится постоянной, а сложность построения одного маршрута принимает вид

$$O\big(N \log N + (B + M)K\big), \tag{8}$$

где $B$ — число блоков, а $K$ — среднее число ребер сетки, затрагиваемых одним объектом при разметке. С учетом числа проходов $p_{\max}$ и среднего числа сегментов в маршруте $s$ итоговая оценка одного полного цикла маршрутизации всех

соединений имеет вид

$$O\big(p_{\max}(|E|N\log N + |E|^2 sK)\big), \tag{9}$$

и ключевое отличие от наивной оценки состоит в том, что число существующих сегментов не умножается на число узлов поиска на каждом шаге алгоритма.

Разработанное решение построено как двухкомпонентная система, состоящая из тестового графического приложения и динамической библиотеки маршрутизации: тестовое приложение отвечает за отображение схемы, взаимодействие с пользователем и подготовку геометрических данных, а библиотека выполняет построение маршрутов соединительных линий. В архитектуре выделяются четыре уровня — уровень пользовательского интерфейса с главным окном, графической сценой и инструментами взаимодействия; уровень модели и визуализации схемы с блоками, портами и соединительными линиями; сервисный уровень, выполняющий сбор данных сцены, управление соединениями и вызов библиотеки; и алгоритмический уровень DLL, реализующий обработку препятствий, учет существующих линий и построение ортогонального маршрута. Пользовательский интерфейс реализован с помощью модулей Qt Widgets и Qt Graphics View, причем геометрия линии хранится отдельно от ее отображения: объект `Link` содержит последовательность точек маршрута, а объект `LinkItem` отвечает за ее отрисовку, включая дуги в точках пересечения с другими линиями. Связующим компонентом между графической частью и библиотекой является класс `SocketManager`, который хранит список соединений, формирует описания линий и прямоугольники блоков-препятствий и передает эти данные в библиотеку, а загрузка и вызов библиотеки инкапсулированы в классе `DllOptimizerLoader`, выполняющем динамическое подключение DLL, получение адресов функций C-API и формирование структуры запроса. Благодаря этому основная часть приложения не зависит от внутренней реализации библиотеки.

Внешний интерфейс библиотеки оформлен как C-совместимый API, что позволяет передавать данные через простые структуры, числовые константы и одномерные массивы и упрощает подключение компонента к различным средам. Для обращения к библиотеке используется непрозрачный дескриптор оптимизатора, скрывающий внутреннюю реализацию объекта, а геометрия блоков и соединений передается через структуру запроса, содержащую массив описа-

ний соединений, массив прямоугольных препятствий и указатель на параметры маршрутизации. Каждое соединение задается координатами и направлениями начального и конечного портов, препятствие — координатами противоположных углов прямоугольника, а точка маршрута — парой целочисленных координат в пикселях. Параметры маршрутизации объединены в отдельную структуру и включают штраф за поворот, штрафы за пересечение линий и прохождение через блок, штраф и минимальное расстояние для близкого параллельного прохождения, радиус обхода чужих портов, шаг сетки, длину начального участка выхода из порта, отступ вокруг препятствий и максимальное число проходов. Результат возвращается через структуру, в которой все построенные маршруты записаны подряд в один массив точек, а массивы индексов начала и длины задают срез для каждой линии, причем нулевая длина означает, что маршрут не построен. Аналогично возвращаются точки пересечений линий, время выполнения и требуемые размеры буферов, а основная функция маршрутизации возвращает код результата — успех, некорректные аргументы, отсутствие маршрутов или недостаточный размер буфера.

Построение маршрута отдельной линии выполняется во внутреннем модуле `LinkRouterInternal` на основе алгоритма A*, работающего в дискретном пространстве координат. Сначала из координат портов формируются точки выхода: точка порта смещается в направлении его нормали на заданное расстояние, поэтому линия сначала выходит из блока перпендикулярно его стороне; затем определяется область поиска, а каждое препятствие предварительно расширяется на величину отступа. Вместо сплошной пиксельной сетки используется сетка значимых координат, в которой массивы координат с заданным шагом дополняются координатами портов, точек выхода, границ препятствий и уже построенных сегментов, после чего сортируются и очищаются от повторений, что уменьшает число узлов поиска без потери важных геометрических элементов. Перед запуском поиска выполняется предварительная разметка ограничений и штрафов: для горизонтальных и вертикальных ребер сетки формируются битовые карты, отмечающие пересечение с существующей линией и прохождение через расширенную область блока, а отдельные массивы хранят величину штрафа за параллельное прохождение рядом с другой линией. Сегменты линий, имеющих общий порт с текущей линией, при этом не учитываются как препятствие, что позволяет нескольким соединениям свободно выходить из одного сокета,

а для обхода чужих портов дополнительно формируется карта запрещенных клеток, через которые маршрут проходить не может.

Сам поиск выполняется алгоритмом A* с использованием очереди с приоритетами. Рассматриваются переходы в четыре стороны, кроме обратного предыдущему, и запрещен переход в клетку области чужого порта, за исключением конечной. Стоимость перехода складывается из длины ребра, штрафа за поворот и штрафов за пересечение линии, прохождение через блок и близкое параллельное наложение, причем благодаря предварительной разметке все эти значения считываются из подготовленных массивов, и на этапе поиска не требуется заново перебирать все препятствия и линии. После достижения конечной клетки путь восстанавливается обратным проходом, к нему добавляются реальные координаты портов, а на заключительном этапе маршрут упрощается: удаляются лишние коллинеарные точки, а зигзагообразные участки по возможности заменяются более короткими угловыми соединениями при условии, что новый участок не пересекает препятствия и не увеличивает число пересечений с другими линиями.

Для проверки работы библиотеки было разработано тестовое графическое приложение на Qt, которое используется не только для отображения результата, но и для подготовки данных, передаваемых в библиотеку. Центральным его компонентом является класс `SocketManager`: он хранит созданные соединения, связывает каждую линию с начальным и конечным сокетом, обновляет геометрию линий при перемещении блоков и запускает оптимизацию через библиотеку. При создании соединения проверяется корректность выбранной пары сокетов — линия может быть создана только от выходного сокета к входному, запрещено соединять сокет сам с собой и подключать несколько линий к одному входному сокету. Ручное создание соединения выполняется инструментом `WireDrawingTool`, который определяет сокет под курсором, строит предварительную пунктирную линию и передает готовый набор точек в `SocketManager`, а если пользователь рисовал линию в обратном направлении, список промежуточных точек разворачивается. При перемещении блоков крайние точки линии синхронизируются с текущими координатами сокетов, а точки выхода пересчитываются по нормали к стороне блока, после чего заново вычисляются точки взаимного пересечения линий и обновляются дуги без повторного запуска полной маршрутизации. Интеграция приложения с библиотекой выполняется через

загрузчик, который динамически подключает DLL, получает адреса функций и преобразует данные сцены в формат интерфейса. Маршрутизация всех соединений производится за один вызов библиотеки, а полученные маршруты преобразуются в модели линий и отображаются на графической сцене в виде ортогональных соединений, причем точки пересечений используются для отрисовки дуг, позволяющих визуально отличать пересечение линий от их соединения.

Для проверки работоспособности алгоритма было подготовлено тестовое расположение элементов схемы, в котором в исходном варианте линии имели неудовлетворительную с точки зрения восприятия геометрию. После применения алгоритма была выполнена оптимизация геометрии соединений с учетом прямоугольных препятствий, существующих линий и заданных параметров отступов, и сравнение исходного и итогового состояний показывает, что применение разработанного компонента повышает читаемость схемы: после маршрутизации соединения представлены ортогональными полилиниями, их участки обходят области блоков и сохраняют заданный отступ от препятствий, количество наложений и пересечений между линиями уменьшается, а оставшиеся пересечения отображаются специальными дугами. Используемый алгоритм не решает задачу глобальной оптимизации всей схемы в строгом математическом смысле — маршруты строятся с учетом текущего расположения блоков, ранее построенных соединений и заданных штрафов, поэтому в схемах с большим количеством линий отдельные пересечения могут сохраняться, — однако результат демонстрирует существенное улучшение структуры соединений по сравнению с исходным вариантом и подтверждает практическую применимость компонента для визуальных сред моделирования схем.

## ЗАКЛЮЧЕНИЕ

Таким образом, в ходе работы был разработан программный компонент в виде динамической библиотеки, предназначенный для автоматической маршрутизации соединительных линий в структурных блочных схемах и позволяющий отделить алгоритмическую часть построения маршрутов от графического приложения и использовать библиотеку как самостоятельный подключаемый модуль. Были рассмотрены особенности представления структурных схем, и на их основе сформирована модель данных, в которой блоки рассматриваются как прямоугольные препятствия, порты — как ориентированные точки подключения, а соединения — как ортогональные полилинии. Для построения маршрутов реализован алгоритм ортогональной трассировки на основе A*, учитывающий геометрию блоков, направления портов, шаг сетки и штрафы за повороты, пересечения и параллельное прохождение, с последующей постобработкой маршрута. Был разработан C-совместимый API библиотеки, через который внешнее приложение передает описания соединений, координаты препятствий и параметры маршрутизации и получает результат в виде одномерных массивов точек полилиний и пересечений, а для проверки работы компонента реализовано тестовое приложение, выполняющее маршрутизацию всех соединений за один вызов библиотеки и отображающее построенные линии на графической сцене. Полученные результаты подтверждают, что разработанный компонент повышает читаемость схемы и обладает практической применимостью. Это доказывает, что все поставленные задачи решены, а цель работы достигнута. Разработанное решение может служить основой для дальнейшего развития средств автоматической оптимизации схем, включая улучшение качества маршрутов, расширение параметров настройки и интеграцию с другими системами моделирования.