УДК 519.6

РЕШЕНИЕ ЗАДАЧИ ШТЕЙНЕРА С ПОМОЩЬЮ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

А.И. Ольшевский, М.Ю. Починский

Государственный университет информатики и искусственного интеллекта,
г. Донецк, Украина info@iai.donetsk.ua

Проведен анализ алгоритмов решения задачи Штейнера. Рассмотрено решение задачи с помощью модифицируемого генетического алгоритма. Показано, что для повышения эффективности работы алгоритма целесообразно выделять области исходных данных (создания начальных популяций) с возможностью распараллеливания процессов вычислений.

ЗАДАЧИ ШТЕЙНЕРА, СИНТЕЗ, АЛГОРИТМЫ РЕШЕНИЯ, КРАТЧАЙШАЯ СЕТЬ, ПОПУЛЯЦИЯ, ОБЛАСТЬ, БЛОК ВЫЧИСЛЕНИЙ

Введение

Важными оптимизационными задачами на графах являются задачи построения кратчайших связывающих деревьев. Для минимизации суммарной длины ребер графа в кратчайших связывающих деревьях предложено при соединении множества деревьев использовать дополнительные точки (вершины). Задачу построения минимального дерева при помощи введения дополнительных точек называют задачей Штейнера[1]. Задача Штейнера является NP-полной.

Задача Штейнера имеет практические приложения – конструирование интегральных электронных схем. Более короткая сеть проводящих линий на интегральной схеме требует меньшего времени зарядки-разрядки по сравнению с более длинной сетью и повышает, таким образом, быстродействие схемы. Однако задача отыскания кратчайшей сети на интегральной схеме имеет другую геометрию, так как проводники на ней обычно проходят лишь в двух направлениях – горизонтальном и вертикальном.

Разновидности задачи о кратчайшей сети из вертикальных и горизонтальных соединений применимы и при проектировании локальных компьютерных сетей.

Прямоугольная версия задачи поиска минимального остовного дерева может быть эффективно решена алгоритмом, выбирающим на каждом шаге кратчайшее соединение, если это соединение не образует замкнутого пути.

В связи с возможностью быстрого получения набора различных деревьев Штейнера наиболее эффективным представляется применение методов генетического поиска для решения поставленной задачи.

В данной статье рассматриваются существующие алгоритмы решения задачи Штейнера и на основе их анализа разработан генетический алгоритм синтеза кратчайшей сети.

1. Генетический алгоритм

Генетический алгоритм (ГА) – это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Является разновидностью эволюционных вычислений. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Задача кодируется таким образом, чтобы её решение могло быть представлено в виде вектора («хромосома»). Случайным образом создаётся некоторое количество начальных векторов («начальная популяция»). Они оцениваются с использованием «функции приспособленности», в результате чего каждому вектору присваивается определённое значение («приспособленность»), которое определяет вероятность выживания организма, представленного данным вектором. После этого с использованием полученных значений приспособленности выбираются векторы (селекция), допущенные к «скрещиванию». К этим векторам применяются «генетические операторы» (в большинстве случаев «скрещивание» – crossover и «мутация» – mutation), создавая таким образом следующее «поколение». Особи следующего поколения также оцениваются, затем производится селекция, применяются генетические операторы и так далее. Так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий останова алгоритма. Таким критерием может быть:

– нахождение глобального, либо субоптимального решения;

– исчерпание числа поколений, отпущенных на эволюцию;

– исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в очень больших, сложных пространствах поиска.

Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Создание начальной популяции.
2. Вычисление функций приспособленности для особей популяции (оценивание).
3. Начало цикла.
4. Выбор индивидов из текущей популяции (селекция).
5. Скрещивание и\или мутация.
6. Вычисление функций приспособленности для всех особей.
7. Формирование нового поколения.
8. Если выполняются условия останова, то конец цикла, иначе – в начало цикла на шаг 3.

Достоинством эвристических алгоритмов является их способность находить оптимальные или же близкие к оптимальным решения задачи большого числа точек за разумное время вычисления. Недостатком является то, что относительно найденного решения нельзя утверждать, что оно оптимально [2]. Двумя основными критериями, оценивающими эвристический алгоритм, будут скорость сходимости и близость получаемого решения к оптимальному или же к лучшему решению. ГА обладает механизмом обмена информацией между особями, который порождает из родительских решений решения потомков [3].

У обычного алгоритма генетического поиска есть существенный недостаток – очень важна последовательность, в которой будут заданы исходные точки. Исходя из этого, предлагается алгоритм с изменяющейся последовательностью точек.

На рис. 1 изображена двухуровневая общая схема функционирования алгоритма.

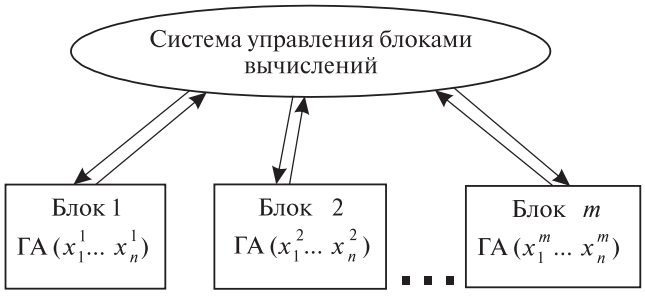


Рис. 1. Структурная схема алгоритма

Блок вычислений – это генетический алгоритм с заданным набором входных параметров и результатом каждого этапа эволюции. Фактически блоки представляют собой отдельные программы, работающие по модифицированному алгоритму генетического поиска с постоянной последовательностью соединения исходных точек. Сами соединения могут быть разными.

Система управления блоками вычислений (СУБВ) – это программа, которая подает на блоки вычислений последовательность точек и производит анализ результатов работы блоков вычислений.

2. Алгоритм работы СУБВ

Алгоритм состоит из 6 основных шагов.

1. Введем кодировку хромосом. Ген будет иметь вид числа от 1 до n, где n – количество исходных точек.

2. Случайным образом (путем генерации случайных последовательностей чисел от 1 до n) создаем начальную популяцию для системы управления блоками вычислений. Обнуляем счетчик популяций: $t = 0$.

3. Рассчитываем значение фитнес-функции. Для этого передаем каждую хромосому в блок вычислений, и он возвращает значение фитнес-функции.

4. Создаем новую популяцию.

4.1. С помощью оператора селекции выбирается 2 родителя.

4.2. Оператор кроссинговера скрещивает родителей и получается потомок (новая хромосома).

4.3. Для того чтобы алгоритм не вырождался, после кроссинговера запускается оператор мутации.

4.4. Если сгенерирована не вся популяция, то переходим на шаг 4.1.

5. Рассчитываем фитнес-функцию для хромосом новой популяции. Для этого передаем каждую хромосому в блок вычислений, и он возвращает значение фитнес-функции.

6. $t = t+1$. Если $t = t_{max}$, где t_{max} – ограничение на количество популяций, алгоритм прекращает работу и решением будет лучшая хромосома последней популяции, иначе на шаг 4.

После того, как ввели кодировку хромосом, создается начальная популяция и рассчитывается фитнес-функции. Для этого генерируется m последовательностей неповторяющихся случайных чисел от 1 до n, где m – количество блоков вычислений, а n – количество исходных точек.

Таким образом, имеем N хромосом, где N может быть задано произвольно.

Пусть $m = 4$ и заданы 4 исходные точки:

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

Предположим, что в результате генерации последовательностей были получены следующие хромосомы:

3, 4, 1, 2;
1, 2, 3, 4;
1, 2, 4, 3;
3, 4, 2, 1.

После этого для каждой хромосомы рассчитываем фитнес-функцию. Передаем хромосому в блок вычислений, и он возвращает значение фитнес-функции.

Обнуляем счетчик популяций: $t = 0$.

Создание новой популяции. Для генерации одной хромосомы новой популяции необходимо последовательно запустить 3 оператора:

1) оператор селекции;
2) оператор кроссинговера;
3) оператор мутации.

Первым запускается оператор селекции. Оператор селекции осуществляет отбор хромосом в соответствии со значениями их функции приспособленности. Существуют как минимум три популярных типа оператора селекции: рулетка, турнирный и равномерный отбор.

Турнирный отбор реализует m турниров, чтобы выбрать m особей. Каждый турнир построен на выборке k элементов из популяции, и выбора лучшей особи среди них. Наиболее распространен турнирный отбор с k=2.

При равномерном отборе все хромосомы имеют равную вероятность быть выбранными. Выбор родительских хромосом производится случайно в независимости от приспособленности хромосом.

Метод рулетки – отбирает особей с помощью n «запусков» рулетки. Колесо рулетки содержит по одному сектору для каждого члена популяции. Размер i-го сектора пропорционален соответствующей величине $P_{sel}(i)$ вычисляемой по формуле:

$$P_{sel}(i) = \frac{f(i)}{\sum_{i=1}^{n} f(i)}.$$

При таком отборе члены популяции с более высокой приспособленностью с большей вероятностью будут чаще выбираться, чем особи с низкой приспособленностью. На рис. 2 показан пример распределения вероятности выбора особи.

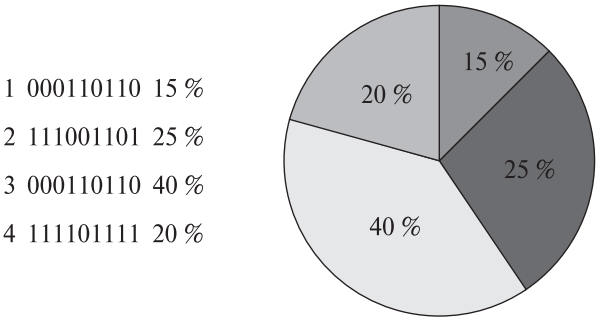


Рис. 2. Оператор селекции типа колеса рулетки с пропорциональными секторами функции приспособленности

Для того чтобы увеличить скорость работы алгоритма будет использоваться равномерный отбор, так как для его осуществления потребуется наименьшее количество математических операций по сравнению с другими видами отбора.

Таким образом, после оператора селекции имеется два числа, представляющие собой порядковые номера хромосом, которые станут родителями.

Оператор кроссинговера скрещивает родителей и получает потомка.

Далее начинает свою работу оператор кроссинговера или скрещивания, как его еще называют.

Оператор скрещивания (crossover) осуществляет обмен частями хромосом между двумя (может быть и больше) хромосомами в популяции. Может быть одноточечным или многоточечным. Одноточечный кроссовер работает следующим образом. Сначала случайным образом выбирается одна из l-1 точек разрыва. Точка разрыва – участок между соседними битами в строке. Обе родительские структуры разрываются на два сегмента по этой точке. Затем соответствующие сегменты различных родителей склеиваются и получаются два генотипа потомков.

Многоточечный кроссовер отличается только тем, что он имеет несколько точек разрыва вместо одной.

Одноточечный оператор кроссинговера требует меньшего количества операций, поэтому использоваться в алгоритме будет он.

Для того чтобы алгоритм не вырождался, после кроссинговера запускается оператор мутации.

Мутация – стохастическое изменение части хромосом. Каждый ген строки, которая подвергается мутации, с вероятностью P_{mut} (обычно очень маленькой) меняется на другой ген.

На рис. 3 изображен пример мутации.

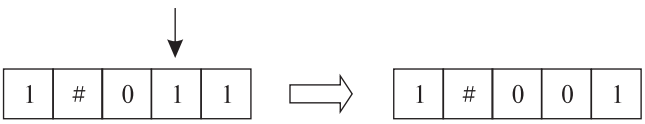


Рис. 3. Оператор мутации

Оператор мутации необходим для «выбивания» популяции из локального экстремума и способствует защите от преждевременной сходимости.

Так же как и кроссинговер, мутация проводится не только по одной случайной точке. Можно выбирать некоторое количество точек в хромосоме для изменения, причем их число также может быть случайным. Также можно мутировать сразу некоторую группу подряд идущих точек. Вероятность мутации значительно меньше вероятности кроссинговера и редко превышает 1%. Среди рекомендаций по выбору вероятности мутации нередко можно встретить варианты $1/L$ или $1/N$, где L – длина хромосомы, N – размер популяции.

Таким образом, в результате работы операторов селекции, кроссинговера и мутации получается новая хромосома.

Новая хромосома подается на вход блока вычислений, и тот возвращает значение фитнес-функции.

После запуска этих 3 операторов m раз, где m – число блоков вычислений, генерация новой популяции закончена.

Счетчик популяций t увеличиваем на 1. $t = t+1$.

В случае, если выполняется условие окончания алгоритма, программа выдает лучшую хромосому из последней популяции.

Условие окончания может быть как ограничением на количество популяций, так и любым другим.

3. Алгоритм работы блока вычислений

Алгоритм блока вычислений можно разбить на 6 основных шагов.

1. Введем кодировку хромосом.

Используется следующая кодировка хромосом: П – вправо; Н – вниз; В – вверх.

Таким образом, указывается направление, в котором стоит следовать от начала до конца дерева.

То есть для заданных точек в пункте 4.3 может получиться следующая хромосома:

1П, 1П, 1Н, 1П.

Здесь гены разделены запятыми. Графически хромосому можно представить, как показано на рис. 4.

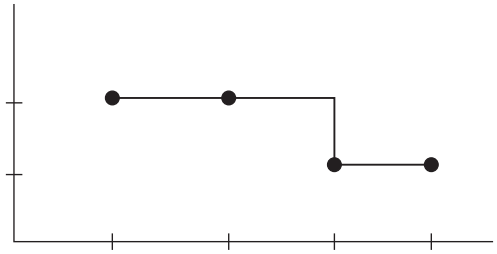


Рис. 4. Пример хромосомы

2. Создаем начальную популяцию для блока вычислений. Хромосомы строятся через всевозможные столбы Штейнера. Обнуляем счетчик популяций: $u = 0$.

Столб Штейнера – это прямая, проведенная через любую точку из исходных, такая, что из каждой другой из исходных точек опускается перпендикуляр до пересечения с этой прямой.

3. Рассчитываем фитнес-функцию для всех хромосом блока вычислений. Фитнес-функция – это длина дерева.

4. Создаем новую популяцию.

4.1. С помощью оператора селекции выбирается 2 родителя.

4.2. Оператор кроссинговера скрещивает родителей и получается потомок (новая хромосома).

4.3. Для того чтобы алгоритм не вырождался, после кроссинговера запускается оператор мутации.

4.4. Если сгенерирована не вся популяция, то – на шаг 4.1.

5. Рассчитываем фитнес-функцию для хромосом новой популяции.

6. $u = u+1$. Если $u = u_{max}$, где u_{max} – ограничение на количество популяций, алгоритм прекращает свою работу, и решением будет лучшая хромосома последней популяции, иначе – на шаг 4.

Создание начальной популяции и расчет фитнес-функции. Хромосомы строятся через всевозможные вертикальные или горизонтальные столбы Штейнера. На рис. 5 приведен пример двух деревьев, построенных на начальной популяции с помощью горизонтальных столбов Штейнера.

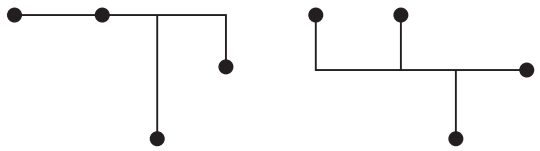


Рис. 5. Деревья с горизонтальными столбами Штейнера

На рис. 6 приведен пример деревьев, построенных с помощью вертикальных столбов Штейнера.

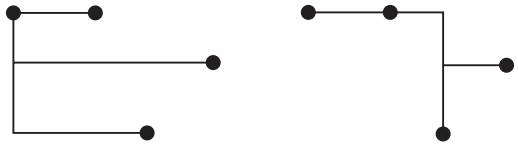


Рис. 6. Деревья с вертикаьными столбами Штейнера

После создания нового поколения для каждой хромосомы рассчитывается фитнес-функция или длина дерева.

Создание новой популяции. Для генерации одной хромосомы новой популяции необходимо последовательно запустить 3 оператора:

1) оператор селекции;

2) оператор кроссинговера;

3) оператор мутации.

С помощью равномерного оператора селекции выбирается два родителя.

Одноточечный оператор кроссинговера скрещивает родителей и получает потомка.

Скрещивание производится одноточечным кроссинговером. На рис. 7 приведен пример скрещивания двух деревьев.

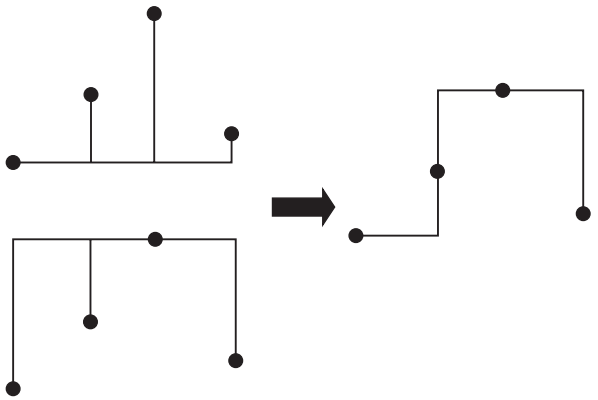


Рис. 7. Скрещивание двух деревьев

Запускается оператор мутации, который изменяет потомка с заданной вероятностью.

Для мутации был выбран отрезок между 2 и 3 точками. На рис. 8 приведена хромосома до и после работы оператора мутации.

Рис. 8. Хромосома до мутации и после

В данном случае мутация проводится между двумя исходными точками с одной дополнительной точкой и просто меняет координаты по осям X и Y местами. Но может возникнуть и другая ситуация. Например, когда дополнительных точек две и они заменяются одной (рис. 9).

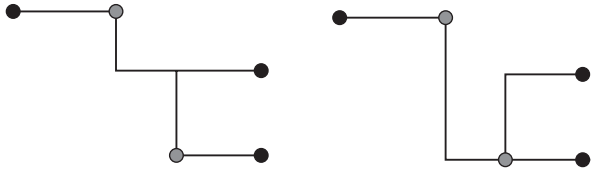


Рис. 9. Хромосома до мутации и после

Конечно, в приведенном примере эта мутация не вызывает никакого сокращения дерева Штейнера, но когда количество точек большее, это может приводить к заметному уменьшению длины дерева.

Но бывают ситуации, когда имеется две дополнительные точки. В этом случае имеет смысл уменьшить общую длину дерева, убрав обе дополнительные точки. В результате мутации будут удалены обе дополнительные точки и исходные точки будут соединены прямой линией (рис. 10).

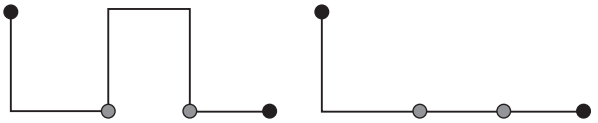


Рис. 10. Хромосома до мутации и после

Существует также другая ситуация, когда подвергаются мутации точки, соединенные прямой линией. Здесь имеет смысл, наоборот, добавить две дополнительные точки, так как есть вероятность, что линии между этими точками совпадут с уже существующими, и в итоге сократится дерево. Только добавление точек происходит в согласовании с ближайшими точками (рис. 11).

Рис. 11. Хромосома до мутации и после

Рассчитываем фитнес-функцию для хромосом новой популяции.

Счетчик популяций *и* увеличиваем на 1. $и=и+1$.

В случае, если выполняется условие окончания алгоритма, программа выдает лучшую хромосому из последней популяции.

Условие окончания может быть как ограничением на количество популяций, так и любым другим.

Опционально, в зависимости от исходных данных, несколько хромосом остаются без изменений. Эти хромосомы называются лидерами популяции.

4. Анализ особенностей формирования исходных данных

Анализ проводился на основании результатов работы программы модифицированного генетического алгоритма.

Последовательно были заданы 6, 20 и 60 точек. Для каждой точки программа запускалась по 20 раз. После чего можно попытаться выявить закономерность, в которой должны соединяться исходные точки.

В табл. 1 приведены координаты 6-ти исходных точек.

Таблица 1

Исходные точки

№	1	2	3	4	5	6
X	3	2	4	1	5	3
Y	6	8	3	9	4	3

В результате запуска алгоритма 16 раз из 20 получилось дерево, изображенное на рис. 12(*а*), длиной 11 единиц при порядке исходных точек – 3, 6, 5, 1, 2, 4 (8 раз из 16) и 5, 6, 3, 1, 2, 4 (7 раза из 16), и 1 раз, при последовательности точек 3, 6, 5, 1, 2, 4.

Остальные 2 раза было получено дерево такой же длины, но с другим порядком – 3, 6, 4, 2, 1, 5. Дерево изображено на рис. 12(*б*).

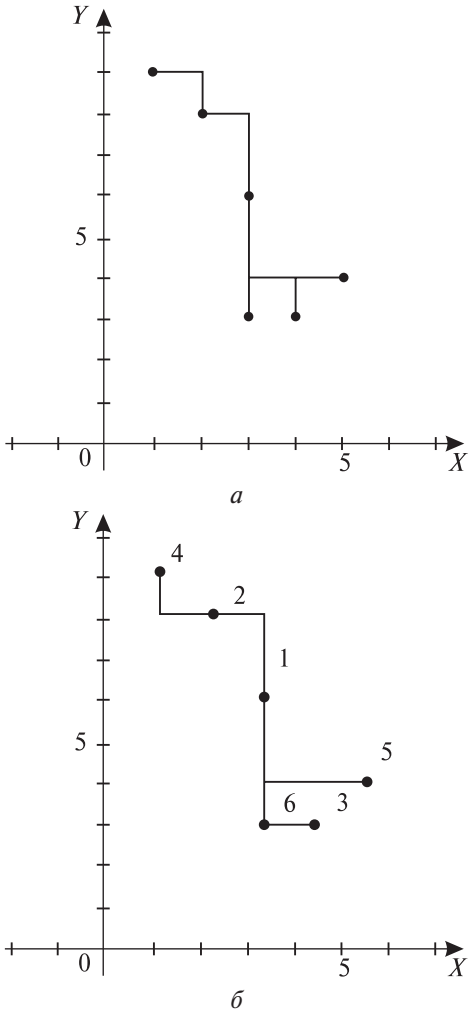


Рис. 12 Деревья для 6 точек, полученные при разном порядке данных

Порядок точек можно представить в виде табл. 2.

Таблица 2

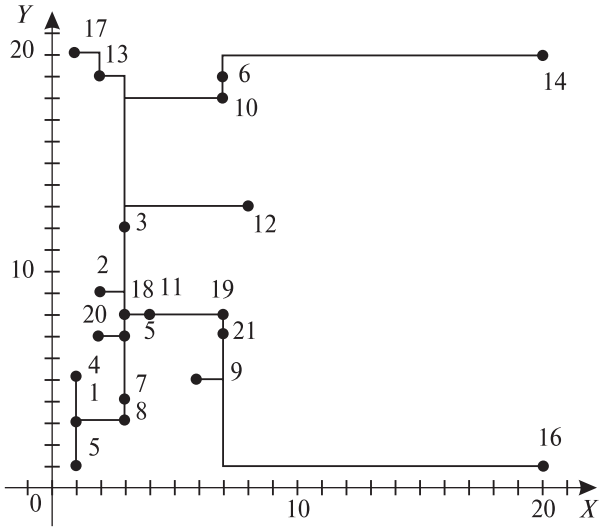
Результат работы программы

№	Порядок точек	Доля от общего числа запусков	Длина дерева
1	3, 6, 5, 1, 2, 4	45%	11
2	5, 6, 3, 1, 2, 4	35%	11
3	3, 6, 4, 2, 1, 5	20%	11

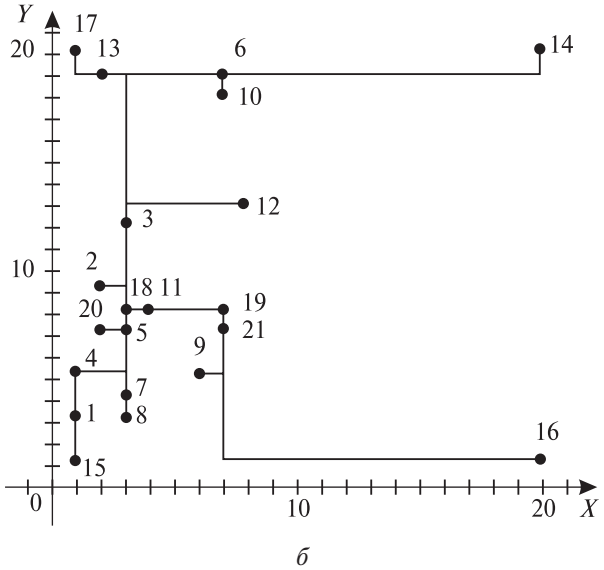
Во всех трех последовательностях точек можно видеть, что точки 1, 2 и 4 все время стоят рядом и образуют область, которая обязательно присутствует во всех лучших решениях.

Также в двух первых случаях область образуют точки 3, 6 и 5, но в третьем случае эта область разбивается на две.

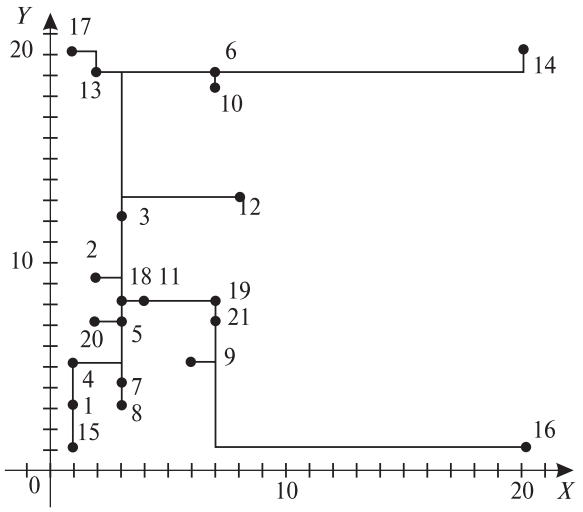
В результате запуска алгоритма 15 раз из 20 получались деревья, изображенные на рисунках 13(а), 13(б) и 13(в), длиной 80 единиц при разных последовательностях исходных точек. Остальные 5 раз были получены различные последовательности и деревья большей длины.



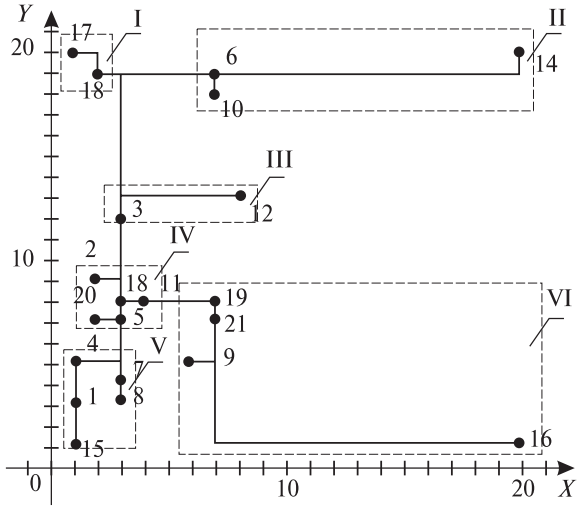
а



б



в



г

Рис. 13. Деревья для 21 точки, полученные при разном порядке данных

Результаты работы алгоритмов для 21 исходной точки приведены в табл. 3.

Таблица 3

Исходные точки

№	1	2	3	4	5	6	7	8	9	10	11	12	13
X	1	2	3	1	3	7	3	3	6	7	4	8	2
Y	3	9	12	5	7	19	4	3	5	18	8	13	19

№	14	15	16	17	18	19	20	21
X	20	1	20	1	3	7	2	7
Y	20	1	1	20	8	8	7	7

Порядок точек и результаты можно представить в виде табл. 4.

Если рассматривать только последовательности, в которых были получены самые короткие деревья, то во всех последовательностях можно выделить повторяющиеся области. Так, любая последовательность состоит из этих областей, соединенных между собой в разном порядке. В табл. 4 для наглядности каждая вторая область выделена жирным шрифтом.

В данном случае области выделялись аналитическим путем. Выделялись области, которые присутствовали во всех последовательностях и были неразрывны. Точки в областях менялись местами,

но не пересекались с другими областями. С большим количеством точек для выделения областей можно использовать любой алгоритм кластеризации по одному признаку.

Таблица 4

Результат работы программы

№	Порядок точек	Доля от общего числа запусков	Длина дерева
1	14, 6, 10, 2, 18, 20, 5, 11, 21, 16, 9, 19, 8, 4, 15, 1, 7, 12, 3, 13, 17	35%	80 (рис. 13(*б*))
2	17, 13, 12, 3, 7, 15, 1, 4, 8, 6, 14, 10, 19, 9, 16, 21, 18, 2, 20, 11, 5	25%	80 (рис. 13(*б*), рис. 13(*в*))
3	7, 8, 4, 15, 1, 5, 20, 11, 2, 18, 9, 16, 21, 19, 3, 12, 13, 17, 6, 10, 14	15%	80 (рис. 13(*б*), рис. 13(*г*))
4	Другие порядки	25%	82-86

На основании анализа табл. 4 можно выделить области, как показано на рис. 13(*г*).

Области образуются скоплениями точек, что хорошо видно из I, IV и V областей на рис. 13(*г*). Точка 14 была включена в одну область с точками 10 и 6, так как любые другие соединения образуют более длинное дерево. Так же произошло с точками 16 (область VI), 12 и 3 (область III).

На основании полученных результатов можно сделать вывод, что признаком, по которому выделяются области, является расстояние между точками.

Таким образом, имеет смысл предложить предобработку исходных данных для существующих алгоритмов решения задачи Штейнера на основании выделения областей.

5. Результаты численных экспериментов

Для программной реализации модифицированного генетического алгоритма потребовалось создать 3 класса: класс СУБВ, класс истории эволюции и класс самой эволюции.

Класс эволюции – это основная часть блока вычислений, в которой реализован генетический алгоритм. В нем реализованы операторы кроссинговера, селекции, мутации и вычисления фитнес-функции.

Класс истории эволюции используется для того, чтобы изучить возможности реализованного алгоритма. Он хранит все популяции, что требует немалого количества оперативной памяти.

Двухуровневая структура организации программного продукта позволяет выполнять параллельные действия, поэтому при увеличении количества ядер наблюдается заметное повышение производительности[4]. С целью определения прироста производительности были проведены эксперименты, при которых определялось время создания следующей популяции с заданными параметрами алгоритма: 8 блоков вычислений, 20 исходных точек, 500 популяций для каждого блока, 20% лидеров, 6% вероятность мутации. В системе с установленном процессором Core 2 Quad 2,4 GHz 8MB L2 cache путем отключения ядер были получены результаты, представленные в табл. 5.

Таблица 5

Зависимость производительности от количества ядер

	Среднее время выполнения, сек	Прирост по сравнению с 1-ядром, %
1-ядро	5,12	0
2-ядра	3,43	49
4-ядра	2,98	72

Таким образом, при переходе с одноядерного процессора на двухядерный наблюдается 49-процентное увеличении производительности, а при переходе с 2-ядерного на 4-ядерный еще 15-процентное.

Заключение

Приведенные исследования показывают, что для уменьшения времени вычислений целесообразно выделять области исходных данных для создания начальных популяций при решении задачи Штейнера с помощью генетического алгоритма. С целью повышения эффективности особое внимание при программной реализации необходимо уделить двухуровневой структуре организации параллельных вычислений.

На основании исследований разработан модифицированный генетический алгоритм решения задачи Штейнера как часть программного комплекса маршрутизации пакетов данных в сети дистанционного обучения. Программная реализация выполнена на языке программирования C++ с использованием библиотеки QT 4.2.

Список литературы: 1. *Ольшевский А.И.* Алгоритмы построения маршрута при групповой рассылке сетевых пакетов данных дистанционного обучения на базе ДонГИИИ // Искусственный интеллект. – 2007. – № 4. – С. 483-490. 2. *Курейчик В.М.* Генетические алгоритмы и их применение. Таганрог: ТРТУ, 2002. – 242 с. 3. *Маршалл Берн У., Грэм Рональд Л.* Поиск кратчайших сетей: Пер. с англ. – М.: Спитер, 1993. – 104 с. 4. *Наумов Л.А.* Метод разбиения задач на подзадачи, рекурсия, «разделяй и властвуй» // Динамическое программирование. – 2002, № 3,4. – С. 94-106.

Поступила в редколлегию 20.10.2008

УДК 519.6

Вирішення задачі Штейнера за допомогою генетичного алгоритму / А.І. Ольшевський, М.Я. Починський // Біоніка інтелекту: наук.-техн. журнал – 2008. – № 2 (69). – С. 145-151.

У роботі розглядаються питання вирішення задачі Штейнера за допомогою генетичного алгоритму. Для зменшення часу обчислень пропонується дворівнева структура з передобробкою початкових точок. Описуються запропоновані підходи. Приводяться аналіз особливостей формування початкових даних і деякі результати чисельних експериментів.

Табл. 5. Іл. 13. Бібліогр.: 4 найм.

UDC 519.6

One Solution of Steiner's Problem Using the Genetic Algorithm / A.I. Olshevskiy, M.Ya. Pochinskiy// Bionics of Intelligence: Sci. Mag. – 2008. – № 2 (69). – P. 145-151.

In the article some questions of Shteyner's problem using the genetic algorithm are considered. The two-level structure with preprocessing of initial points is proposed to reduce time of calculations. The suggested approaches are described. The analysis of initial data forming and some results of numerical experiments are presented.

Tab. 5. Fig. 13. Ref.: 4 items.