МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
Национальный исследовательский ядерный университет «МИФИ»

ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ

Направление подготовки: 01.03.02 Прикладная математика и информатика

ПРОЕКТНАЯ ПРАКТИКА

Пояснительная записка к проекту

«Моделирование передачи данных в оптических сетях и повышение эффективности существующих сетей»

Выполнили:
Соболь К.А., Б19-511

Научный руководитель:
Кочанов Марк Борисович, должность

Москва, 3 июня 2020 года

1. Оглавление:

2. Введение:

- В современном мире большая часть высокоскоростного интернета передается посредством волоконно-оптической связи. Это способ передачи информации, использующий в качестве носителя информационного сигнала электромагнитное излучение оптического (ближнего инфракрасного) диапазона, а в качестве направляющих систем — волоконно-оптические кабели. Благодаря высокой несущей частоте и широким возможностям мультиплексирования пропускная способность волоконно-оптических линий многократно превышает пропускную способность всех других систем связи и может измеряться терабитами в секунду. Малое затухание света в оптическом волокне позволяет применять волоконно-оптическую связь на значительных расстояниях без использования усилителей. Волоконно-оптическая связь свободна от электромагнитных помех и труднодоступна для несанкционированного использования: незаметно перехватить сигнал, передаваемый по оптическому кабелю, технически крайне сложно.
- В основе волоконно-оптической связи лежит явление полного внутреннего отражения электромагнитных волн на границе раздела диэлектриков с разными показателями преломления. Оптическое волокно состоит из двух элементов — сердцевины, являющейся непосредственным световодом, и оболочки. Показатель преломления сердцевины несколько больше показателя преломления оболочки, благодаря чему луч света, испытывая многократные переотражения на границе сердцевина-оболочка и распространяется в сердцевине, не покидая её.
- Частота информационного сигнала в оптических сетях находится в пределах ближнего инфракрасного (100 – 400 ТГц) диапазона, т.к. именно в этих пределах достигается наименьшее затухание.
  В силу сложной квадратурной модуляции скорость передачи информации зависит от длины диапазона частот, по которым происходит передача сигнала.
- Понятно, что систему оптических сетей можно представить в виде графа, где вершины – являются некими отправителями и принимающими информационный сигнал, а дуги между ними – это кабель сети с определенными частотами и длинной. На этом этапе возникает проблема передачи данных в оптических сетях. А именно необходимо разработать такой алгоритм, благодаря которому вся эта сложная система будет взаимодействовать между собой.
  Как уже было сказано, сигнал должен передаваться по определенным частотам(срезам), которые еще и должны быть смежные. Т.е. от одной вершины(отправителя) к другой вершине(получателя) срезы должны остаться неизменными на всем пути, а также покрывать объем запроса, который необходимо обработать, а также быть наиболее оптимальными с точки зрения длины пути.

3. Аналитическая часть:

В статье https://arxiv.org/pdf/1904.06994.pdf представлен адаптированный алгоритм Дейкстры для оптических сетей. А также сравнение его эффективности с другими существующими алгоритмами для данной задачи. Сравнение эффективности алгоритм происходит не на реальных условиях, а на специально подобранных значениях входных данных. К сожалению для проверки по этим данным в статье использовали суперкомпьютер и проверить эффективность самому довольно затруднительно.

В оптических сетях принято делить весь диапазон частот света на каналы с диапазоном примерно в 25 – 50 ГГц. Но в условиях эластичных оптических сетей мы имеем дело с динамическим трафиком, когда соединение не длятся долго. Кроме того, учитывая растущие оптические сети, постоянно растущую потребность в полосе пропускания и гибкости соединения, которая еще больше возрастает из-за требований беспроводных сетей пятого поколения (5G).

В статье было предложено разделить каналы на так называемые срезы шириной по 6,25 ГГц, тем самым повысив эластичность сети (т.к. для обработки небольшого запроса будет выделятся небольшой срез, в отличие от целого канала)

За основу алгоритма в статье взят алгоритм Дейскстры. Только вот сравнение вершим происходит по меткам, которые содержать в себе длину пути и диапазон доступных срезов.

На рис. 1 представлен псевдокод алгоритма.

Algorithm 1
In: $G = (V = \{v_i\}, E = \{e_i\}), W(e_i), S(e_i), l, d = (s, t, n)$
Out: $p = (e_1, ..., e_i, ..., e_l)$, $\Sigma = \{\sigma_i\}$

$L_s = \{(0, e_\emptyset, \Omega)\}$
push $(0, e_\emptyset)$ to Q
while Q is not empty **do**
  $q = (c, e) = \text{pop}(Q)$
  $v = e.target$
  if $v == t$ **then**
    break the while loop
  end if
  $SSSC = \{l.SSC : l \in L_v \text{ and } l.c == c \text{ and } l.e == e\}$
  for all $S \in SSSC$ **do**
    for all $e' \in$ outgoing edges of v **do**
      $S' = S \cap S(e')$
      $c' = c + W(e')$
      if $c' \leq m$ and S' can support d **then**
        $v' = e'.target$
        $l' = (c', e', S')$
        if $\nexists l \in L_{v'} : l \leq l'$ **then**
          $L_{v'} = L_{v'} \setminus \{l : l \in L_{v'} \text{ and } l' \leq l\}$
          $L_{v'} = L_{v'} \cup \{l'\}$
          push (c', e') to Q
        end if
      end if
    end for
  end for
end while
return $(p, \Sigma) = trace(L, s, t)$

Рис. 1 псевдокод алгоритма предложенного в статье

4. Теоретическая часть:

Адаптированный алгоритм Дейкстры для оптических сетей. Показательное распределение и распределение Пуассона, триангуляция Делоне и граф Габриэля.

5. Практическая часть:

При реализации алгоритма был сделан класс вершин и класс ребер. Также пригодилась бинарная куча для взятия минимального элемента.

Также в процессе написания проекта я сделал генератор случайных чисел на основе плотности вероятности (а именно показательно распределения и распределения Пуассона). Для проверки эффективности были еще реализованы генераторы триангуляции Делоне и графа Габриэля.

А также множество мелких функций.

6. Экспериментальная часть:

Проверка алгоритма:

- Граф строится на 100 вершинах
- Граф – это граф Габриэля
- Каждое ребро содержит 400 слотов
- Каждый день по {10, 12.5, 15, 17.5, 20, 25, 30, 35, 40, 45, 50, 55, 60, 70, 80, 90, 100, 150, 200, 300, 400, 500, 600, 700, 800, 900, 1000} запросов распределенные при помощи показательного распределения
- Количество занимаемых слотов – 10 по распределению Пуассона
- Вершины, на которых строится запрос выбираются случайно
- Существование запроса: 10 часов распределенные при помощи показательного распределения

Probability – вероятность установления соединения (т.е. пути в согласно условиям)

Time – время установления соединения

Length – длина установленного соединения

Slice – количество занимаемых слотов

Utilization – заполненность графа по отношению к занимаемым слотам

Сравнение моих результатов с результатами в статье:

Общий результат представлен в виде строки – Name, Time, Utilization, Success, Length, Range, Way.

На основе которых строятся графики:

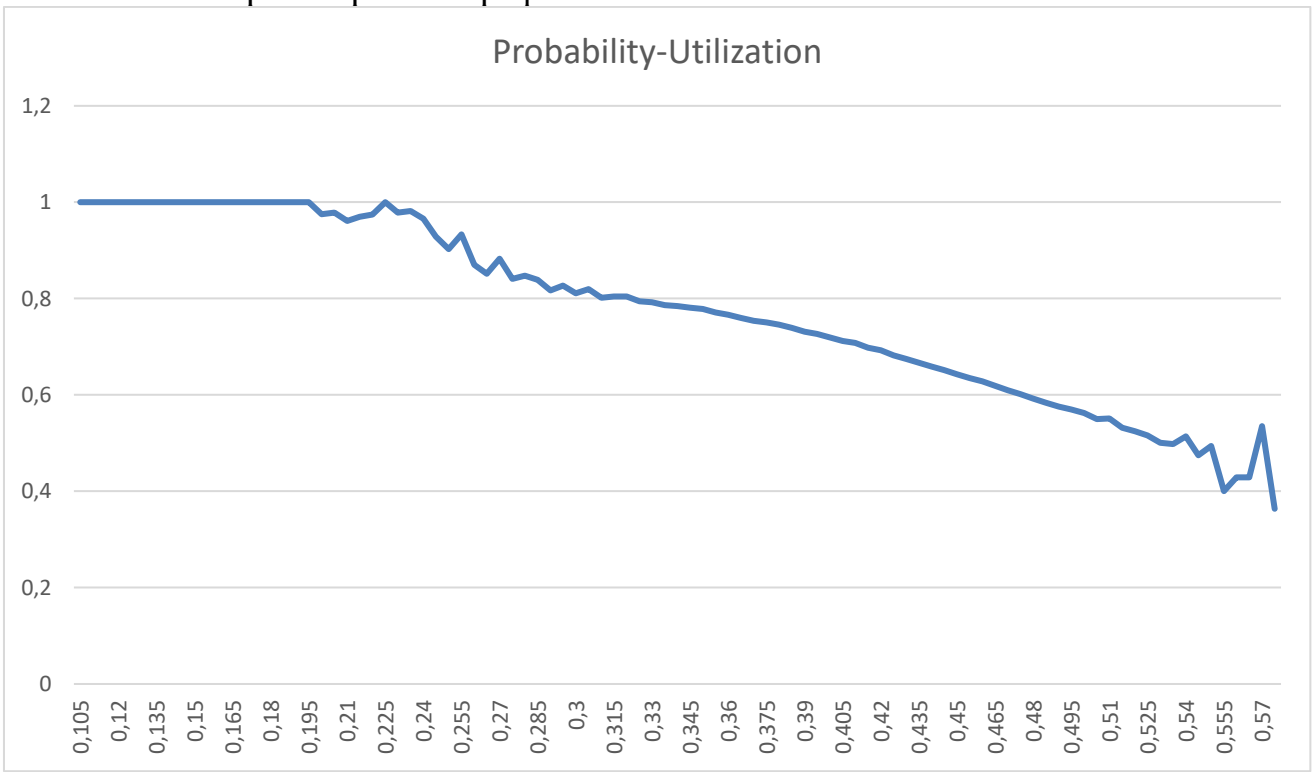

Рис. 2 Probability-Utilization (мой результат)

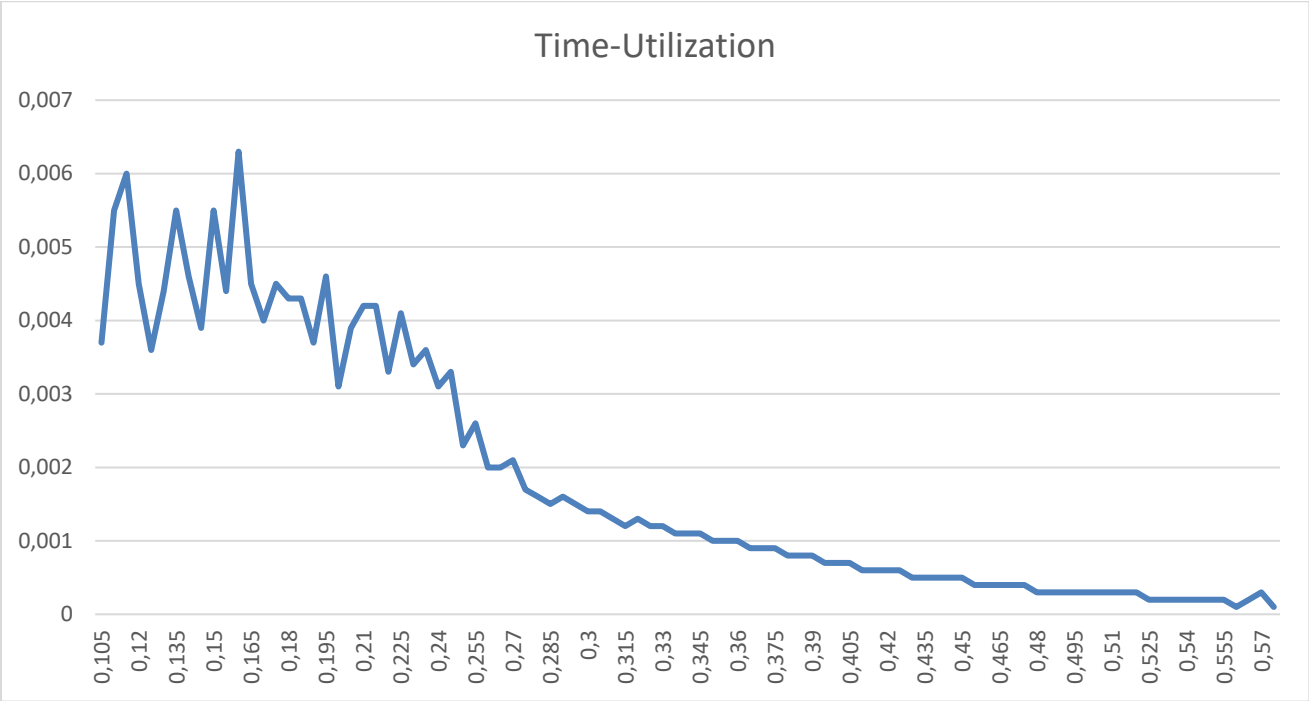

Рис. 3 Time-Utilization (мой результат)

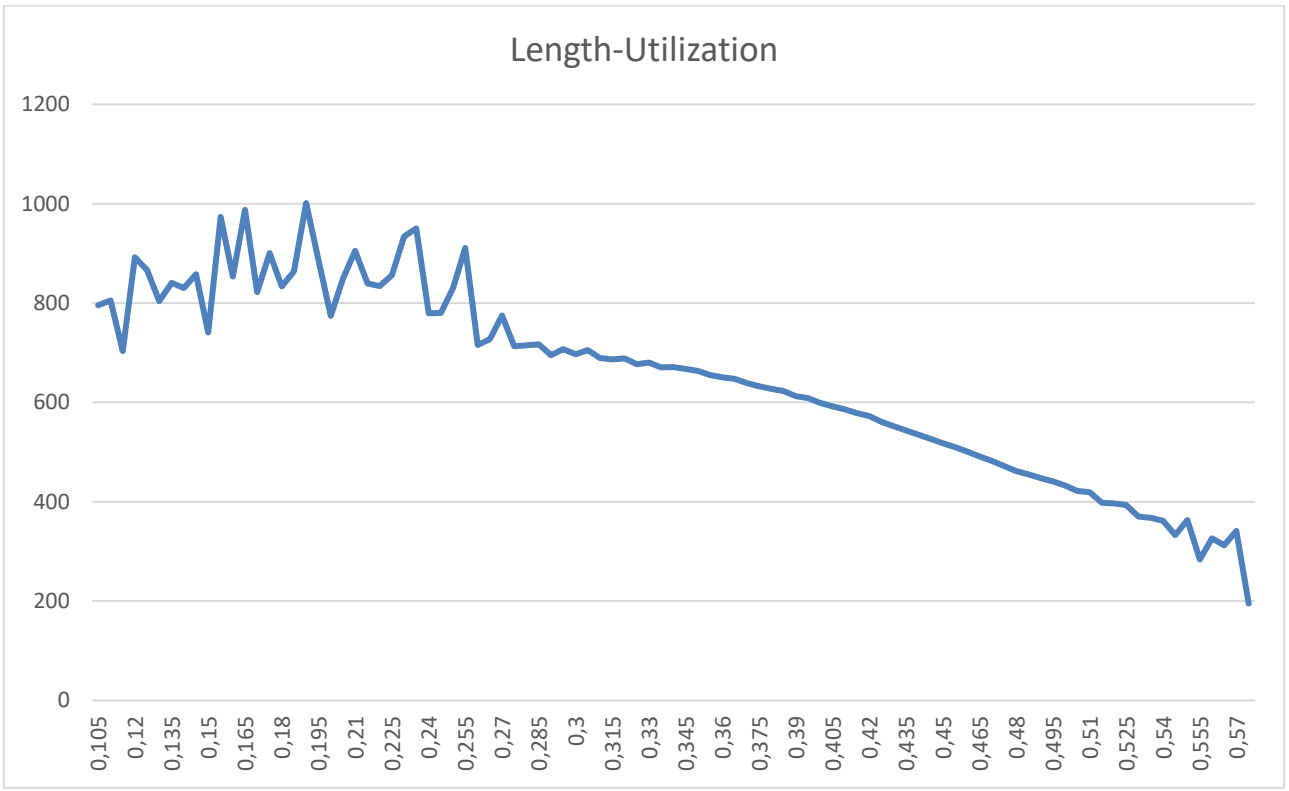

Рис. 4 Length-Utilization (мой результат)

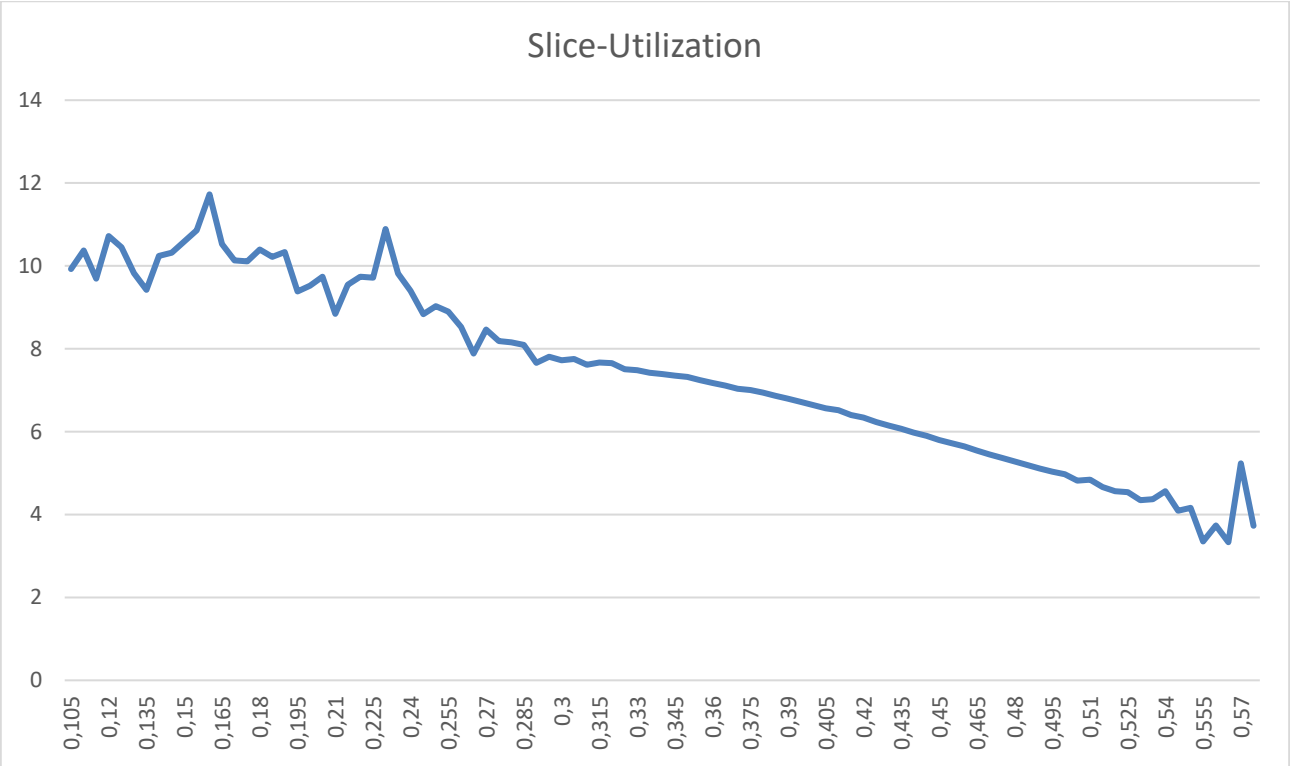

Рис. 5 Slice-Utilization (мой результат)

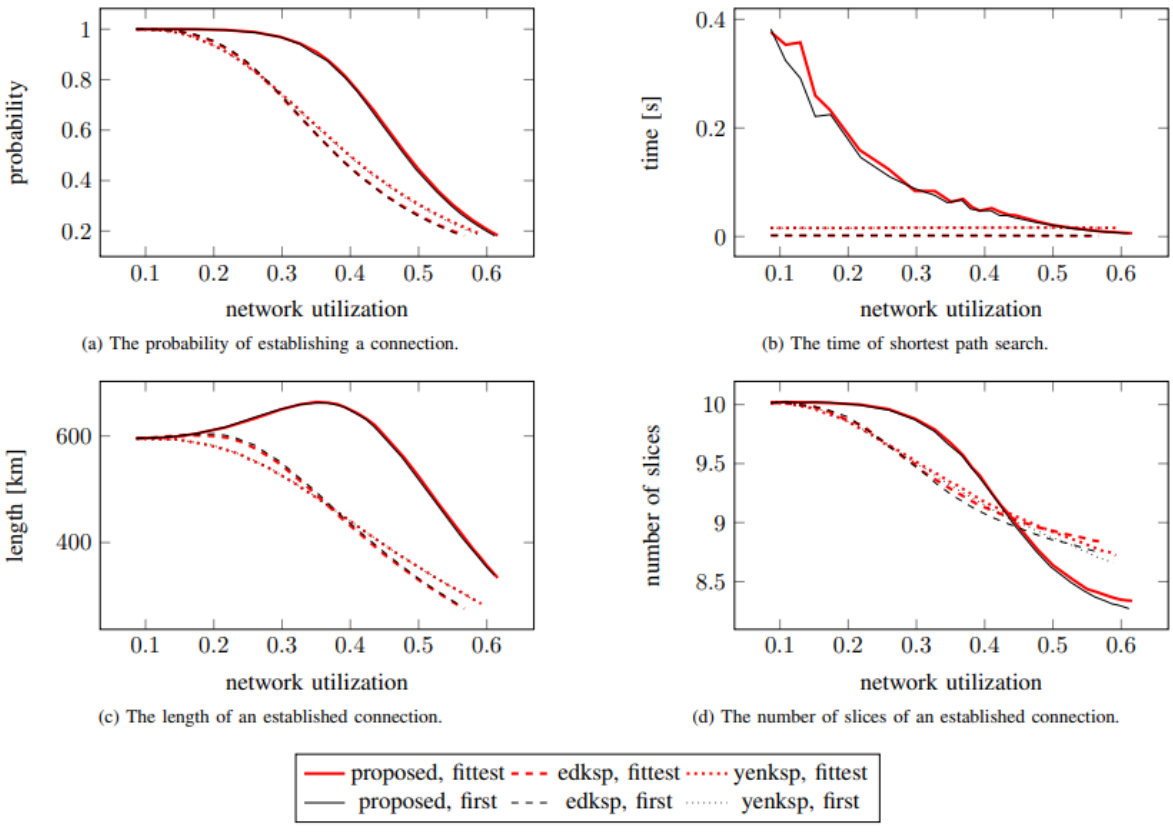

Рис. 6 (результаты из статьи)

Разность результатов возможно вызвана тем, что я тестировал алгоритм на графе Габриэля из 30 вершин, а в статье на графе из 100 вершин.

7. Заключение:

- В процессе разработки проекта было сделано:

✓ Бинарная куча
✓ Поиск Дейкстры
✓ Поиск Йена
✓ Адаптированный поиск Дейкстры для оптической сети
✓ Моделирование работы алгоритмов (наиболее приближенной к реальным условиям):

-Триангуляция Делоне
-Граф Габриэля
-Показательное распределение
-Распределение Пуассона

- Ожидается:

✓ Адаптированный алгоритм Йена для оптической сети (идейно уже готов)
✓ Сравнение алгоритмов (с помощью мощного компьютера)

По отношению к графикам результатов можно заметить, что начиная с загруженности в 0,27 по 0,54 все графики имеют вполне стабильный вид. Понятия не имею чем это вызвано (может быть даже работой моего компьютера при проверке работы алгоритма).
В общем и целом, если смотреть с большим оптимизмом, то мои результаты и результаты из статьи немного похожи.
Возможно если бы проверка осуществлялась на графе из 100 вершин, то графики совпадали бы еще больше.

8. Список литературы:

1) Szcześniak, Ireneusz, and Bożena Woźna-Szcześniak. "Adapted and constrained Dijkstra for elastic optical networks." *2016 International Conference on Optical Network Design and Modeling (ONDM)*. IEEE, 2016.
2) Ahujia, R. K., Thomas L. Magnanti, and James B. Orlin. "Network flows: Theory, algorithms and applications." *New Jersey: Rentice-Hall* (1993).
3) Velasco, Luis, et al. "Solving routing and spectrum allocation related optimization problems: From off-line to in-operation flexgrid network planning." *Journal of Lightwave Technology* 32.16 (2014): 2780-2795.
4) Zhang, Guoying, et al. "A survey on OFDM-based elastic core optical networking." *IEEE Communications Surveys & Tutorials* 15.1 (2012): 65-87.
5) Wan, Xin, et al. "Dynamic routing and spectrum assignment in flexible optical path networks." *2011 Optical Fiber Communication Conference and Exposition and the National Fiber Optic Engineers Conference*. IEEE, 2011.
6) Cetinkaya, Egemen K., et al. "On the fitness of geographic graph generators for modelling physical level topologies." *2013 5th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, 2013.

9. Приложения:

A.

Реализованный алгоритм:

```
public void Algo(Vertex start, Vertex end, double max_distance, int min_diaposon)
      {
         Reset_Label();
         bool check;
         BinaryHeap Heap_Priority = new BinaryHeap();
         Vertex pointer_vertex = start;
         var edge_start = new Edge(start, start, All_Slots);
         var pointer_heap = new Edge_Distance(edge_start, 0);
         Heap_Priority.Add_To_Heap(pointer_heap);
         List<int[]> list_range = new List<int[]>();

         start.Labels.Add(new Label(0, edge_start, All_Slots));
         while (Heap_Priority.Edges_Sort.Count != 0)
         {
            pointer_heap = Heap_Priority.Get_Min();
            pointer_vertex = pointer_heap.Edge.To;

            list_range = new List<int[]>();

            foreach (var label in pointer_vertex.Labels)
            {
               if (label.Distance == pointer_heap.Distance && label.Previous_Edge.From == pointer_heap.Edge.From &&
label.Previous_Edge.To == pointer_heap.Edge.To)
               {
                  list_range.Add(label.Slots);
               }
            }

            foreach (var range in list_range)
            {
               foreach (var outedge in pointer_vertex.OutEdge)
               {
                  if (outedge.Slots.Length == 0)
                  {
                     continue;
                  }
                  int[] cross = Cross_Range(range, outedge.Slots);
                  double distance = pointer_heap.Distance + outedge.Weight;

                  if (distance < max_distance && Max_Diaposon(cross).Length >= min_diaposon)
                  {
                     var vertex = outedge.To;
                     var label_add = new Label(distance, outedge, cross);

                     var remove_label = new List<Label>();

                     check = true;
                     foreach (var label_vertex in vertex.Labels)
                     {
                        if (distance >= label_vertex.Distance && Comparison_Mas(cross, Cross_Range(cross, label_vertex.Slots)) ==
true)
                        {
                           check = false;
                           break;
                        }
                        if (distance <= label_vertex.Distance && Comparison_Mas(label_vertex.Slots, Cross_Range(cross,
```

```
label_vertex.Slots)) == true)
                        {
                            remove_label.Add(label_vertex);
                        }
                    }

                    if (check)
                    {
                        foreach (var label_del in remove_label)
                        {
                            vertex.Labels.Remove(label_del);
                        }
                        vertex.Labels.Add(label_add);
                        Heap_Priority.Add_To_Heap(new Edge_Distance(outedge, distance));
                    }
                }
            }
        }
    }
```