Appendix 1 – Code for Misra – Gries

Main:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MisraGries
{
    class Program
    {
        static void Main(string[] args)
        {
            MGAlgorithm algorithm = new MGAlgorithm (20000);

            algorithm.Process2();

            var v = algorithm.Cache.OrderByDescending(pr => pr.Value).First();

            Console.WriteLine(string.Format("Triple found: {0} {1} {2}",
                v.Key.Values.ElementAt(0),
                v.Key.Values.ElementAt(1),
                v.Key.Values.ElementAt(2)));

            Console.ReadLine();
        }
    }
}
```

Computation:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MisraGries
{
    class MGAlgorithm
    {
        private int cacheSize;
        private Dictionary<Triple, int> cache;

        public Dictionary<Triple, int> Cache { get { return cache; } }

        public DataProvider dt = new DataProvider();

        private int tripleCount = 0;
        private int percentCount = 0;
        private int[] percentages = new int[] { 1, 10, 25, 50, 100 };
        private const int totalTriples = 1925588228;

        public MGAlgorithm(int cacheSize)
        {
            this.cacheSize = cacheSize;
            cache = new Dictionary<Triple, int>(cacheSize);
        }

        public void Process(IEnumerable<int> actors)
        {
            if (actors.Count() < 3) { return; }

            foreach (Triple tr in GetTriples(actors))
            {
                Process(tr);
            }
        }

        public void Process2()
        {
            Stopwatch st = new Stopwatch();
            st.Start();
            Console.WriteLine("Started");

            List<Role> roles = new List<Role>();
            int prevMov = -1;

            foreach (var v in dt.GetRoles())
            {
                if (prevMov != v.MovieId)
                {
                    prevMov = v.MovieId;

                    if (0 < roles.Count)
                    {
                        //Debug.WriteLine("Processing movie " + ++movieCount);

                        foreach (var tr in GetTriples(roles))
                        {
                            Process(tr);
                            tripleCount++;
                            double nr = totalTriples * (percentages[percentCount] / 100.0);
                            if (nr <= tripleCount)
                            {
                                Console.WriteLine(
                                    string.Format("{0} % processed in {1} seconds", percentages[percentCount], st.
    ElapsedMilliseconds / 1000));
                                percentCount++;
                            }
                        }
                        roles.Clear();
                    }
                }
                roles.Add(v);
            }
        }

        private void Process(Triple triple)
        {
            if (cache.ContainsKey(triple))
            {
                cache[triple]++;
            }
            else
            {
                cache.Add(triple, 1);

                if (cacheSize <= cache.Keys.Count)
                {
                    for (int i = 0; i < cache.Keys.Count; i++)
                    {
                        var key = cache.Keys.ElementAt(i);
                        cache[key]--;
                        if (cache[key] <= 0)
                        {
                            cache.Remove(key);
                        }
                    }
                }
            }
        }

        private IEnumerable<Triple> GetTriples(IEnumerable<int> actors)
        {
            for (int i = 0; i < actors.Count(); i++)
            {
                for (int j = i + 1; j < actors.Count(); j++)
                {
                    for (int k = j + 1; k < actors.Count(); k++)
                    {
                        yield return new Triple(actors.ElementAt(i), actors.ElementAt(j), actors.ElementAt(k));
                    }
                }
            }
        }

        private IEnumerable<Triple> GetTriples(IEnumerable<Role> roles)
        {
            for (int i = 0; i < roles.Count(); i++)
            {
                for (int j = i + 1; j < roles.Count(); j++)
                {
                    for (int k = j + 1; k < roles.Count(); k++)
                    {
                        yield return new Triple(
                            roles.ElementAt(i).ActorId,
                            roles.ElementAt(j).ActorId,
                            roles.ElementAt(k).ActorId);
                    }
                }
            }
        }
    }
}
```

Appendix 2 – Code for our algorithm

Main:

```
using Entities;
using System;
using System.Diagnostics;
using TriangleProblem.Utils;
using TriangleProblem.Utils.Computation;
using Utils.Input;

namespace TriangleProblem
{
   public class Program
   {
      const String FILE_PATH = "../../../../imdb/segments/all.csv";

      static void Main(string[] args)
      {
         FileParser fileParser = new FileParser(FILE_PATH);
         Stopwatch stopwatch = new Stopwatch();
         stopwatch.Start();
         Graph graph = fileParser.Parse();
         stopwatch.Stop();

         TimeSpan time = TimeSpan.FromMilliseconds(stopwatch.ElapsedMilliseconds);
         Console.WriteLine("Graph builded in (seconds): " + time.TotalSeconds);

         GraphManager manager = new GraphManager(graph);

         stopwatch.Reset();
         stopwatch.Start();
         Result result = manager.FindTreeActorsThatPlayedInMostMovies();
         stopwatch.Stop();

         foreach (Actor actor in result.Actors)
         {
            Console.WriteLine(actor.Id);
         }

         time = TimeSpan.FromMilliseconds(stopwatch.ElapsedMilliseconds);

         Console.WriteLine("total: "+ result.TotalMovieCount);
         Console.WriteLine("Time elapsed (seconds): " + time.TotalSeconds);
         Console.WriteLine("DONE");
         Console.ReadLine();
      }
   }
}
```

Bulding graph:

```
using Entities;
using Entities.DAL;
using Entities.Extensions;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
namespace Utils.Input
{
    public class FileParser : IParser
    {
        private String FilePath { get; set; }

        public FileParser(String filePath)
        {
            FilePath = filePath;
        }

        public Graph Parse()
        {
            String[] deli = {","};
            String[] deliMovieTitle = {","', '","};
            Graph graph = new Graph();
            Roles roles = new Roles();

            using (FileInput input = new FileInput(FilePath))
            {
                String line;

                while ((line = input.ReadLine()) != null)
                {
                    var array = line.Split(deli, StringSplitOptions.RemoveEmptyEntries);
                    int actor1Id = int.Parse(array[0]);
                    int actor2Id = int.Parse(array[1]);
                    int movieId = int.Parse(array[2]);

                    if (!graph.Actors.ContainsKey(actor1Id)) graph.Actors.Add(actor1Id, new Actor() { Id = a
ctor1Id });
                    if (!graph.Actors.ContainsKey(actor2Id)) graph.Actors.Add(actor2Id, new Actor(){ Id=act
or2Id });
                    if (!graph.Movies.ContainsKey(movieId)) graph.Movies.Add(movieId, new Movie() { Id
= movieId });

                    Actor actor1 = graph.Actors[actor1Id];
                    Actor actor2 = graph.Actors[actor2Id];
                    Movie movie = graph.Movies[movieId];
                    Edge edge;

                    if (actor1.Edges.Exists(e => e.StartNode == actor1 && e.EndNode == actor2))
                    {
                        edge = actor1.Edges.FirstOrDefault(e => e.StartNode == actor1 && e.EndNode == acto
r2);

                    }
                    else
                    {
                        edge = new Edge() { StartNode = actor1, EndNode = actor2 };
                        actor1.Edges.Add(edge);
                    }
                    edge.CommonMovies.Add(movie);
                }
            }

            return graph;
        }
    }
}
```

Computations:

```
using Entities;
using System;
using System.Collections.Generic;
using System.Linq;

namespace TriangleProblem.Utils.Computation
{
   public class GraphManager
   {
      private Graph Graph { get; set; }

      public GraphManager(Graph graph)
      {
         Graph = graph;
      }

      public GraphManager()
      {
         RemoveEdges(Graph.Actors[0]);
      }

      public Result FindTreeActorsThatPlayedInMostMovies()
      {
         int movies_count = 0;
         Result result = new Result();

         foreach (Actor actor in new List<Actor>(Graph.Actors.Values))
         {
            for (int i = 0; i < actor.Edges.Count; i++)
            {
               List<Movie> movies1 = new List<Movie>(actor.Edges[i].CommonMovies);
               if (movies_count >= movies1.Count)
                  continue;
               for (int j = i + 1; j < actor.Edges.Count; j++)
               {
                  List<Movie> movies2 = new List<Movie>(actor.Edges[j].CommonMovies);
                  if (movies_count >= movies2.Count)
                     continue;
                  int count = CommonMovieSubsetCount(movies1, movies2);
                     if (movies_count < count)
                     {
                        movies_count = count;
                        result.Actors[0] = actor;
                        if (actor == actor.Edges[i].StartNode)
                        {
                           result.Actors[1] = actor.Edges[i].EndNode;
                        }
                        else
                        {
                           result.Actors[1] = actor.Edges[i].StartNode;
                        }
                        if (actor == actor.Edges[j].StartNode)
                        {
                           result.Actors[2] = actor.Edges[j].EndNode;
                        }
                        else
                        {
```

```
                        result.Actors[2] = actor.Edges[j].StartNode;
                    }
                }
            }

        }
        //RemoveEdges(actor);
    }

    result.TotalMovieCount = movies_count;

    return result;
}

public int CommonMovieSubsetCount(List<Movie> movies1, List<Movie> movies2)
{
    int p1 = 0;
    int p2 = 0;
    int count = 0;
    while (p1 < movies1.Count && p2 < movies2.Count)
    {
        if (movies1[p1].Id == movies2[p2].Id)
        {
            count++;
            if (p1 < movies1.Count)
            {
                p1++;
            }
            if (p2 < movies2.Count)
            {
                p2++;
            }
        }
        else if (movies1[p1].Id < movies2[p2].Id)
        {
            if (p1 < movies1.Count)
            {
                p1++;
            }
        }
        else if (movies2[p2].Id < movies1[p1].Id)
        {
            if (p2 < movies2.Count)
            {
                p2++;
            }
        }
    }

    if (p1 == movies1.Count)
    {
        for (int i = p2; i < movies2.Count; i++)
        {
            if (movies1[p1 - 1].Id == movies2[i].Id)
                count++;
        }
    }

    if (p2 == movies2.Count)
    {
        for (int i = p1; i < movies1.Count; i++)
        {
            if (movies2[p2 - 2].Id == movies1[i].Id)
                count++;
        }
    }
    return count;
}
```