CS486 Assignment 1

Glenn Hartmann, Sylvain Loranger, Filbert Ma

gjhartma, slorange, fmma

20266893, 20281054, 20259693

May 26, 2011

Total pages: 22 (not including title page or table of contents)

Contents

Listings

1 Source Code

We implemented 4 different algorithms to experiment broadly with a few different approaches to solving this problem. The first way was the given greedy algorithm from the assignment specifications. The next algorithm we used, which was our first to be outside the given algorithm, was a Genetic Algorithm (the most creative, but least successful). Next was a "Total Cost" heuristic approach, and last was an "Opportunity Cost" heuristic. Descriptions of all 4 algorithms can be found below in Section 4 (page 20).

Listing 1: Graph.java

```
import java.io.File;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;

public class Graph
{
   private String inputFileName = null;
   public static Node[] nodes;
   public static Edge[] edges;
   ArrayList<Edge> path = new ArrayList<Edge>();

   public Graph(String[] args, String defaultTest)
   {
      if(args.length >= 1){
         inputFileName = args[0];
      }
      else{
         inputFileName = defaultTest;
      }
   }

   public int makeGraph()
   {
      Scanner input = null;

      try{
         input = new Scanner(new File(inputFileName));
      }
      catch(Exception e){
         try{
            input = new Scanner(new File("src/"+inputFileName));
         }
         catch(Exception e2){
            return -1;
         }
      }

      ArrayList<Node> nodesTemp = new ArrayList<Node>();
```

```
        ArrayList<Edge> edgesTemp = new ArrayList<Edge>();

        //create all nodes
        int i = 0;
        while(input.hasNextLine())
        {
            Node n = new Node(input.nextLine(), i);
            nodesTemp.add(n);
            i++;
        }

        //create all edges
        for(i = 0; i < nodesTemp.size(); i++){
            Node n1 = nodesTemp.get(i);
            for(int j = 0; j < nodesTemp.size(); j++){
                Node n2 = nodesTemp.get(j);
                int value = n1.value(n2);
                if(value > 0){
                    Edge e = new Edge(n1, n2, value);
                    n1.addToNext(e);
                    edgesTemp.add(e);
                }
            }
        }
        nodes = new Node[nodesTemp.size()];
        edges = new Edge[edgesTemp.size()];
        nodesTemp.toArray(nodes);
        edgesTemp.toArray(edges);

        Genetic.setNodes(nodes);

        return 0;
    }

    //sort the edges using their comparison function
    //use the edges in decreasing order
    public Edge[] getPath(){
        Arrays.sort(edges);
        for(int i = edges.length-1; i >= 0; i--){
            Edge e = edges[i];
            Node from = e.getFromNode();
            Node to = e.getToNode();
            if(!from.isFromUsed() && !to.isToUsed() && from.getStartNode() != to)
                addEdgeToPath(e);
        }

        Edge[] rtn = new Edge[path.size()];
        path.toArray(rtn);
        return rtn;
    }

    //helper for getPath()
    private void addEdgeToPath(Edge e){
        Node from = e.getFromNode();
```

```
      Node to = e.getToNode();
      from.useFrom();
      to.useTo();
      //for all nodes that have startNode set as the to node, change it to the from node
      for(int j = 0; j < nodes.length; j++){
         if(nodes[j].getStartNode() == to){
            nodes[j].setStartNode(from.getStartNode());
         }
      }
      path.add(e);
   }

   //used after getPath to order it nicely and to show 0 edges
   public static Edge[] organizePath(Edge[] path){
      Edge[] organized = new Edge[nodes.length-1];
      int count = 0;
      Node to = null;

      for(int i = 0; i < path.length; i++){
         Edge e = path[i];
         Node from = e.getFromNode();
         if(!from.isToUsed()){//if the from node hasnt been used as a to node, then its a start node
            if(to != null){//create 0 node between paths
               organized[count] = new Edge(to,from);
               count++;
            }
            to = e.getToNode();
            organized[count] = e;
            count++;
            while(to.isFromUsed()){//while the to node hasnt been used as a from node, we havent reached the end of the path
               for(int j = 0; j < path.length; j++){//find next one
                  Edge e2 = path[j];
                  if(e2.getFromNode() == to){
                     from = to;
                     to = e2.getToNode();
                     organized[count] = e2;
                     count++;
                     break;
                  }
               }
            }
         }
      }
      for(int i = 0; i < nodes.length; i++){
         if(!nodes[i].isToUsed() && !nodes[i].isFromUsed()){
            if(to == null){
               nodes[nodes.length-1].useTo();
               to = nodes[nodes.length-1];
            }
            organized[count] = new Edge(to,nodes[i]);
            count++;
```

```
            to = nodes[i];
         }
      }
      //System.out.println(count + " " + (nodes.length-1));
      return organized;
   }

   //print out the wordsnake
   public static String pathString(Edge[] edges){
      String s = "";
      if(edges.length > 0)
         s = edges[0].getFromNode().getWord();
      for(int i = 0; i < edges.length; i++){
         s += edges[i].getToNode().getWord().substring(edges[i].getValue());
      }
      return s;
   }

   public void printNodes()
   {
      for(int i = 0; i < nodes.length; i++){
         System.out.println(nodes[i].getWord());
      }
   }

   public int size()
   {
      return nodes.length;
   }
}
```

Listing 2: Genetic.java

```
import java.util.Random;

public class Genetic
{
   private Node startGene;    //one gene for the start node
   private Edge[] edgeGenes;  //one gene for each edge in the path
   private int mateLower;     //used by generation to determine which genetics
       get to mate
   private int mateUpper;     //used by generation to determine which genetics
       get to mate
   private static Random rand = Wordsnake.r;
   private static Node[] allNodes = null;
   private int score = -1;    //score of the path

   public static final double MUTATION_RATE = 0.00; // TODO experiment with this
        value

   //creating the first generation of genetics
   public Genetic(){
      int num = allNodes.length;
      edgeGenes = new Edge[num-1];
      startGene = allNodes[rand.nextInt(allNodes.length)];
```

```java
      boolean[] available = new boolean[num];
      for(int i = 0; i < num; i++){
         available[i] = true;
      }
      Node prevNode = startGene;
      available[startGene.getIndex()] = false;

      //for each nodes, pick a node to go to
      for(int x = 0; x < num-1; x++){
         boolean random = true;
         int numNext = prevNode.getNumNext();
         if(numNext > 0){
            Node nextNode = prevNode.getNext(rand.nextInt(numNext)).getToNode();
            if(available[nextNode.getIndex()]){
               edgeGenes[x] = new Edge(prevNode, nextNode);
               available[nextNode.getIndex()] = false;
               random = false;
               prevNode = nextNode;
            }
         }
         if(random){//the nodes has no available nodes to go to so we pick one randomly and add a zero edge
            int i = -1;
            int c = 0;
            for(i = 0; i < num; i++) //count the number of available nodes left
               if(available[i])
                  c++;
            i = -1;
            int j = rand.nextInt(c);
            while(j > -1){ //get the j'th number from the available array
               i++;
               if(available[i])
                  j--;
            }
            edgeGenes[x] = new Edge(prevNode, allNodes[i]);
            available[i] = false;
            prevNode = allNodes[i];
         }
      }
   }

   //used in mate
   public Genetic(Node start, Edge[] edges)
   {
      startGene = start;
      edgeGenes = edges;
      mateLower = -1;
      mateUpper = -1;
   }

   //called by graph
   public static void setNodes(Node[] nodes)
   {
      allNodes = nodes;
```

```
    }

    //used in generation to get the best genetic and decide on which genetics get
        to mate
    public int getScore(){
        if(score == -1){
            score = 0;
            int numEdges = edgeGenes.length;
            for(int x = 0; x < numEdges; x++){
                score += Math.pow(edgeGenes[x].getValue(),2);
            }
        }
        return score;
    }

    public void setMateChanceLower(int i)
    {
        mateLower = i;
    }

    public void setMateChanceUpper(int i)
    {
        mateUpper = i;
    }

    public int getMateChanceLower()
    {
        return mateLower;
    }

    public int getMateChanceUpper()
    {
        return mateUpper;
    }

    //O(n)
    public Edge getNext(Node n){
        for(int i = 0; i < edgeGenes.length; i++){
            if(edgeGenes[i].getFromNode() == n){
                return edgeGenes[i];
            }
        }
        return null;
    }

    //These are used only by mate and its helper function but since we can't pass
        some of these by reference we need to declare outside
    private Node prevNode;
    private boolean[] available = new boolean[allNodes.length];
    private Node childStart;
    private Edge[] childEdges = new Edge[allNodes.length-1];
    private int childNumber;

    //O(n^2)
```

```
   //small chance to mutate at each step
   //starts by picking one of the parents startnodes
   //for each edge it picks randomly between one of the parents to nodes
   //if it mutates or if the parent's is unavailable, itll pick an available edge randomly
   public Genetic mate(Genetic g2)
   {
      Genetic g1 = this;
      int len = g1.getSize();
      if(len != g2.getSize()){
         System.out.println("attempted to mate Genetics with different sizes");
         return null; // these Genetics aren't compatible and shouldn't be mated
      }
      if(len != allNodes.length)
         System.out.println("found a Genetic with an incorrect number of edges");

      childNumber = 0;
      for(int i = 0; i < len; i++){
         available[i] = true;
      }

      //start gene
      double r = rand.nextDouble();
      if(r > MUTATION_RATE / 5){ //use one of the parents
         if(rand.nextBoolean())
            childStart = g1.startGene;
         else
            childStart = g2.startGene;
      }
      else //use a random one
         childStart = allNodes[rand.nextInt(len)];
      available[childStart.getIndex()] = false;

      prevNode = childStart;
      //edge genes
      for(int i = 0; i < len-1; i++){
         double mutationRate = MUTATION_RATE;
         if(g1.getNext(prevNode) != null)
            mutationRate /= g1.getNext(prevNode).getValue();
         if(g2.getNext(prevNode) != null)
            mutationRate /= g2.getNext(prevNode).getValue();
         r = rand.nextDouble();
         boolean mutate = r < mutationRate;
         if(!mutate){ //use one of the parents
            if(rand.nextBoolean()){
               if(!useParentsNextEdge(g1)){//try the first parent
                  if(!useParentsNextEdge(g2)){//try the second parent if the first fails
                     mutate = true;//both parents failed so we have to mutate
                  }
               }
            }
            else{
```

```
                if(!useParentsNextEdge(g2)){//try the second parent
                    if(!useParentsNextEdge(g1)){//try the first parent if the
                        second fails
                        mutate = true;//both parents failed so we have to mutate
                    }
                }
            }
        }
        if(mutate){
            //make new edge from lastNode to a random one in available
            int c = 0;
            for(int k = 0; k < len; k++) //count the number of available nodes
                left
                if(available[k])
                    c++;

            int j = rand.nextInt(c);
            int k = -1;
            while(j > -1){ //get the j'th number from the available array
                k++;
                if(available[k])
                    j--;
            }
            Node n = allNodes[k];
            childEdges[childNumber] = new Edge(prevNode, n);
            available[k] = false;
            prevNode = n;
            childNumber++;
        }
    }

    return new Genetic(childStart, childEdges);
}

//helper function for mate()
//O(n)
private boolean useParentsNextEdge(Genetic parent){
    Edge nextEdge = parent.getNext(prevNode);//O(n)
    if(nextEdge == null)
        return false;
    Node nextNode = nextEdge.getToNode();
    boolean avail = available[nextNode.getIndex()];
    if(avail){
        childEdges[childNumber] = nextEdge;
        childNumber++;
        available[nextNode.getIndex()] = false;
        prevNode = nextNode;
    }
    return avail;
}

public int getSize(){
    return edgeGenes.length+1;
}
```

```

   public String toString(){
      StringBuilder s = new StringBuilder();
      for(int i = 0; i < edgeGenes.length; i++){
         s.append(edgeGenes[i]);
         s.append("\n");
      }
      return s.toString();
   }
}
```

Listing 3: Generation.java

```
public class Generation {
   private Genetic[] genetics;
   public static final int k = 350;
   private int totalScore;
   Genetic bestGeneticEver;

   public Generation(){
      totalScore = 0;
      genetics = new Genetic[k];
      for(int x = 0; x < k; x++)
      {
         genetics[x] = new Genetic();
      }
   }

   private Generation(Genetic[] gen, Genetic best){
      totalScore = 0;
      // k = gen.size();
      if(k != gen.length)
      {
         //TODO fail horribly
      }
      genetics = gen;

      bestGeneticEver = best;
   }

   public Generation nextGeneration(){
      //get total score and set changes to mate
      for(int i = 0; i < k; i++){
         Genetic g = genetics[i];
         g.setMateChanceLower(totalScore+1);
         totalScore += (Math.pow(g.getScore(), 2));
         g.setMateChanceUpper(totalScore);
      }
      //get k children
      Genetic[] children = new Genetic[k];
      for(int i = 0; i < k; i++){
         children[i] = newChild();
      }
      return new Generation(children, getBestGenetic());
   }
```

```

    private Genetic newChild(){
        int r1 = 0;

        if(totalScore != 0)
        {
            r1 = Wordsnake.r.nextInt(totalScore) + 1;
        }

        Genetic p1 = null;
        for(int i = 0; i < k; i++){
            Genetic g = genetics[i];
            if(r1 >= g.getMateChanceLower() && r1 <= g.getMateChanceUpper()){
                p1 = g;
            }
        }
        if(p1 == null)
            System.out.println("error1 in Generation.java " + totalScore + " " + r1
                );
        Genetic p2 = null;
        while(p2 == null){
            int r2 = 0;
            if(totalScore != 0)
            {
                r2 = Wordsnake.r.nextInt(totalScore);
            }
            for(int i = 0; i < k; i++){
                Genetic g = genetics[i];
                if(r2 >= g.getMateChanceLower() && r2 <= g.getMateChanceUpper()){
                    if(g != p1){
                        p2 = g;
                        break;
                    }
                }
            }
        }

        return p1.mate(p2);
    }

    public int getAverageScore(){
        int total = 0;
        for(int i = 0; i < k; i++){
            Genetic g = genetics[i];
            total += g.getScore();
        }
        return total / k;
    }

    public Genetic getBestGenetic(){
        int highestScore;
        if(bestGeneticEver == null)
            highestScore = 0;
        else
```

```
            highestScore = bestGeneticEver.getScore();
        for(int i = 0; i < k; i++){
            Genetic g = genetics[i];
            if(highestScore < g.getScore()){
                highestScore = g.getScore();
                bestGeneticEver = g;
            }
        }
        return bestGeneticEver;
    }
}
```

Listing 4: Edge.java

```
public class Edge implements Comparable<Edge>
{
    private int value;
    private Node fromNode;
    private Node toNode;
    private int opportunityCost = -1;
    private int totalCost = -1;

    public Edge(Node from, Node to, int val)
    {
        fromNode = from;
        toNode = to;
        value = val;
    }

    public Edge(Node from, Node to)
    {
        fromNode = from;
        toNode = to;
        value = from.value(to);
    }


    public Node getFromNode()
    {
        return fromNode;
    }

    public Node getToNode()
    {
        return toNode;
    }

    public int getValue()
    {
        return value;
    }

    public void setTo(Node to)
    {
        toNode = to;
```

```
    }

    public void setFrom(Node from)
    {
        fromNode = from;
    }

    public void setValue(int val)
    {
        value = val;
    }

    //returns the next best alternative for the from and to node
    public int getOpportunityCost(){
        if(opportunityCost == -1){
            Edge[] edges = Graph.edges;
            int fromCost = 0, toCost = 0;
            for(int i = 0; i < edges.length; i++){
                Edge e = edges[i];
                if(this != e && fromNode == e.fromNode){
                    if(fromCost < e.value){
                        fromCost = e.value;
                    }
                }
                if(this != e && toNode == e.toNode){
                    if(toCost < e.value){
                        toCost = e.value;
                    }
                }
            }
            opportunityCost = Math.max(fromCost,toCost);
        }
        return opportunityCost;
    }

    //returns the total value of each node the to and from node could have been
        used for
    public int getTotalCost(){
        if(totalCost == -1){
            Edge[] edges = Graph.edges;
            int cost = 0;
            for(int i = 0; i < edges.length; i++){
                Edge e = edges[i];
                if(this != e && fromNode == e.fromNode){
                    cost += e.value;
                }
                if(this != e && toNode == e.toNode){
                    cost += e.value;
                }
            }
            totalCost = cost;
        }
        return totalCost;
    }
```

```java

    //used by sort to make a path
    //implementation varies with algorithm
    public int compareTo(Edge e) {

        if(Wordsnake.solution == 1){
            return this.value - this.getTotalCost() - (e.value - e.getTotalCost());
        }
        else if(Wordsnake.solution == 2){
            return this.value - this.getOpportunityCost() - (e.value - e.
                getOpportunityCost());
        }
        else{
            return this.value - e.value;
        }
    }

    public String toString(){
        if(Wordsnake.solution == 1 && value > 0){
            return fromNode.getWord() + " --> " + toNode.getWord() + " --- " +
                value + " " + totalCost;
        }
        else if(Wordsnake.solution == 2 && value > 0){
            return fromNode.getWord() + " --> " + toNode.getWord() + " --- " +
                value + " " + opportunityCost;
        }
        else{
            return fromNode.getWord() + " --> " + toNode.getWord() + " --- " +
                value;
        }
    }
}
```

Listing 5: Node.java

```java
import java.util.ArrayList;

public class Node
{
    private String word;
    private boolean fromUsed = false;   //whether this node has been used as a
        from node
    private boolean toUsed = false;     //whether this node has been used as a
        from node
    private Node startNode = this;      //the starting node on this node's path
    private int index;                  //the Nodes number in the Nodes array (
        Genetic uses this)

    private ArrayList<Edge> next = new ArrayList<Edge>(); //used by genetic

    public Node(String str, int i)
    {
        word = str;
        index = i;
    }
```

```java

    public String getWord()
    {
        return word;
    }

    public void useFrom(){
        fromUsed = true;
    }

    public void useTo(){
        toUsed = true;
    }

    public boolean isFromUsed(){
        return fromUsed;
    }

    public boolean isToUsed(){
        return toUsed;
    }

    public Node getStartNode(){
        return startNode;
    }

    public void setStartNode(Node n){
        startNode = n;
    }

    public void addToNext(Edge e){
        next.add(e);
    }

    public int value(Node n){
        if(this == n)
        {
            // don't match words with themselves
            return -1;
        }
        int i = 0;
        String w1 = this.word;
        String w2 = n.word;
        int len1 = w1.length();
        int len2 = w2.length();
        if(len2 < len1)
        {
            i = len1 - len2;
        }
        while(i < len1 && !w1.substring(i).equals(w2.substring(0, len1 - i))){
            i++;
        }
        return len1 - i;
    }
```

```

    public Edge getNext(int i)
    {
        return next.get(i);
    }

    public int getNumNext()
    {
        return next.size();
    }

    public int getIndex(){
        return index;
    }
}
```

Listing 6: Wordsnake.java

```
import java.io.BufferedWriter;
import java.util.Random;

public class Wordsnake
{
    public static int solution = 2;
    public static String testName = "test1";
    //0 -> greedy
    //1 -> total cost
    //2 -> opportunity cost
    //3 -> genetic algorithm

    public static Random r = null;
    public static int NUMBER_OF_GENERATIONS = 3000;
    public static BufferedWriter out;

    public static void main(String[] args)
    {
        String inputFile = testName+".txt";
        if(args.length >= 1){
            inputFile = args[0];
        }

       //for genetic algorithm
        if(solution == 3){
            long seed = System.currentTimeMillis();
            if(args.length >= 2){
                seed = Integer.parseInt(args[1]);
            }
            System.out.println("Using seed " + seed);
            r = new Random(seed);
        }

        Graph newGraph = new Graph(args, inputFile);
        int bufReader = newGraph.makeGraph();
        if(bufReader == -1)
        {
```

```
            System.out.println("Could not open the file");
            return;
        }

       //for heuristic algorithms
        if(solution < 3){

            Edge[] path = newGraph.getPath();
            path = Graph.organizePath(path);

            int score = 0;
            for(int i = 0; i < path.length; i++){
                score += Math.pow(path[i].getValue(), 2);
                System.out.println("" + path[i]);
            }
            System.out.println(""+Graph.pathString(path));
            System.out.println("Maximal number = " + score);
        }
        else{//genetic algorithm

            //parameters that we can play with:
            //Genetic::mutationRate
            //Generation::k (number of Genetics/Generation)
            //Generation's killing off Genetics algorithm
            //WordSnakes NUMBER_OF_GENERATIONS

            Generation g = new Generation();
            for(int i = 0; i < NUMBER_OF_GENERATIONS; i++){
                g = g.nextGeneration();
            //only output every 10 generations
                if(i%10 == 9)
                    System.out.println("At generation " + (i+1) + " best score is " + g.getBestGenetic().getScore()
                        + " and the average score is " + g.getAverageScore());
            }
            System.out.println("Maximal number = " + g.getBestGenetic().getScore());
            System.out.println(g.getBestGenetic().toString());
        }
    }
}
```

2 words.txt Testcase Results

Table 1: Results for the words.txt testcase

	Best Score	Best Wordsnake
Greedy Algorithm	355	incrediblemishapenultimaterriblendingratessilaterrestrialsuddenudenseayessentialastingerunderdevelopediatrickyeternallytenseemergeriatricertainvent
Genetic Algorithm	339	emergeriatrickysuddenudenselastingerunderdevelopediatricertaincrediblendingratessilaterrestrialseeminventerriblemshapenultimatenseayessentiallyeternalas
Total Cost Heuristic	332	ratessilaterriblendingrateincrediblemishapenultimateternalastingerunderdevelopediatrickyetenseemergeriatricertainventerrestrialsuddenudenseallyessential
Opportunity Cost Heuristic	356	yessentialastingerunderdevelopediatrickyeternallyincrediblemishapenultimatenseemergeriatricertainventerriblendingratessilaterrestrialsuddenudensea

3 Additional Testcases and Results

Our first testcase tests our algorithms on a set of words with many conflicts to ensure that our techniques and heuristics can find a good answer even with local competition.

Listing 7: many_conflicts_test.txt

```
abcde
bcdef
bcdeg
cdefg
defgabc
defgmno
mnopqr
pqrstu
fgmnop
abcdlmno
stuabc
studef
stumno
stupqr
nopcde
```

Table 2: Results for the many_conflicts_test.txt testcase

	Best Score	Best Wordsnake
Greedy Algorithm	138	bcdefgmnopqrstupqrstumnopcdestudefgabcdegstuabcdlmno
Genetic Algorithm	138	studefgabcdegbcdefgmnopqrstupqestuabcdlmnostumnopcde
Total Cost Heuristic	124	stupqrstumnopcdefgabcdegstuabcdlmnopqrstudefgmnopbcdef
Opportunity Cost Heuristic	138	stupqrstumnopcdebcdefgmnopqrstuabcdlmnostudefgabcdeg

The next testcase verifies that our program correctly handles input with very large overlap between sets of words.

Listing 8: maximal_overlap_test.txt

```
abcde
```

```
bcdef
cdefg
fghij
ghijk
hijkl
ijklm
jklmn
klmno
lmnop
mnopq
nopqr
opqrs
pqrst
qrstu
rstuv
stuvw
tuvwx
uvwxy
vwxyz
```

Table 3: Results for the maximal_overlap_test.txt testcase

	Best Score	Best Wordsnake
Greedy Algorithm	292	abcdefghijklmnopqrstuvwxyz
Genetic Algorithm	292	abcdefghijklmnopqrstuvwxyz
Total Cost Heuristic	292	abcdefghijklmnopqrstuvwxyz
Opportunity Cost Heuristic	292	abcdefghijklmnopqrstuvwxyz

overlap_front_back is just a simple test to make sure the code correctly handles combining from the front and to the back.

Listing 9: overlap_front_back_test.txt

```
nestea
nest
tea
```

Table 4: Results for the overlap_front_back_test.txt testcase

	Best Score	Best Wordsnake
Greedy Algorithm	25	nestea
Genetic Algorithm	25	nestea
Total Cost Heuristic	25	nestea
Opportunity Cost Heuristic	25	nestea

The no_overlap testcase makes sure that our program correctly returns a score of 0 when there are no overlapping words.

Listing 10: no_overlap_test.txt

```
anna
```

```
bob
crytec
drizzled
effe
fourf
gang
hanth
ioi
jouliaj
krak
lol
mom
noun
onto
pop
quaq
rawr
sages
tut
uou
vorteav
wow
xanx
yoyoy
zaz
```

Table 5: Results for the no_overlap_test.txt testcase

	Best Score	Best Wordsnake
Greedy Algorithm	0	zazannabobcrytecdrizzledeffefourfganghanthioijouliajkraklolmomnounontopopquaqrawrsagestutuouvorteavwowxanxyoyoy
Genetic Algorithm	0	bobfourfzazyoyoyjouliajpopontonounmomkrakannacrytecvorteavxanxdrizzledouotuteffeganghanthioiquaqrawrsageslolwow
Total Cost Heuristic	0	zazannabobcrytecdrizzledeffefourfganghanthioijouliajkraklolmomnounontopopquaqrawrsagestutuouvorteavwowxanxyoyoy
Opportunity Cost Heuristic	0	zazannabobcrytecdrizzledeffefourfganghanthioijouliajkraklolmomnounontopopquaqrawrsagestutuouvorteavwowxanxyoyoy

The overlap_itself testcase makes sure that words that begin and end the same do not mistakenly try to gain score by overlapping themselves.

Listing 11: overlap_itself_test.txt

```
haha
reenter
terminator
ending
inglip
lipase
necromancer
certainly
lyric
iceman
```

Table 6: Results for the overlap_itself_test.txt testcase

	Best Score	Best Wordsnake
Greedy Algorithm	44	necromancertainlyricemanendinglipasereenterminatorhaha
Genetic Algorithm	44	endinglipasereenterminatornecromancertainlyricemanhaha
Total Cost Heuristic	37	endinglipasecertainlyicemanecromancereenterminatorhaha
Opportunity Cost Heuristic	41	endinglipasereenterminatoricemanecromancertainlyrichaha

“same_words” tests the behaviour of multiple copies of a word given in the input word list.

Listing 12: same_words_test.txt

```
insert
insert
insert
insert
insert
insert
insert
insert
insert
```

Table 7: Results for the same_words_test.txt testcase

	Best Score	Best Wordsnake
Greedy Algorithm	288	insert
Genetic Algorithm	288	insert
Total Cost Heuristic	288	insert
Opportunity Cost Heuristic	288	insert

4 Algorithm Descriptions

4.1 Greedy

This is the exact simplistic algorithm given to us on the assignment description. This was our starting point for the other 3 algorithms below, and we included it only as a reference to benchmark our other algorithms’ performance.

4.2 Genetic

Our original idea was to make use of a Genetic Algorithm (see `https://secure.wikimedia.org/wikipedia/en/wiki/Genetic_algorithm`). What this means is that we devised a way to encode each “solution” uniformly, created a “generation” of randomly generated solutions (“individuals”), then tested the “fitness” of each individual. The most fit individuals have a better chance to mate (by combining their representation of a solution with another high-scoring individual of the same generation). The newly created individuals from mating form the second generation. The idea is that, given large enough population diversity, and enough generations, that the individuals will tend more and more toward better solutions. This approach is much less time-consuming than a brute-force approach, but more time-consuming than a simple heuristic. We believed that this could achieve better end results than a heuristic might, so we decided to attempt this solution despite the greater runtime requirements.

In our specific implementation, we decided to encode an "individual" as a starting node, followed by $n-1$ edges to make a path through the original graph.

In practise, our genetic algorithm often got stuck on local maxima and stopped progressing after a certain point. Although there is some degree of random chance in this algorithm, we could not seem to get answers as high as heuristic approaches no matter how many times we ran our algorithm. The traditional solution to the problem of local maxima in a genetic algorithm is to introduce "genetic mutation". This is the process of randomizing a certain solution with very low probability to ensure greater diversity within a generation, and thereby have a greater chance of getting out of a local maximum. Although this did solve our problem of getting stuck in local maxima, it seemed to severely impede overall progress, so we were still unable to get good solutions with a reasonable number of individuals and generations.

We believe that this approach could have worked if we had continued to experiment with different probabilities for mating and mutation, however overall it seems to be less efficient at runtime than a simpler heuristic anyway.

4.3 Total Cost

Because our genetic algorithm did not produce results as favourable as we had hoped, we also tried some more traditional heuristic approaches. The first of which is a fairly simple approach. At every node x, we picked the next node y_i to be the one which maximizes the function $f(x, y_i) = v(x, y_i) - g(x, y_i)$ where x is the current node, y_i is the potential next node, $v(x, y_i)$ is the value of combining node x with node y_i (ie, the overlap between nodes x and y_i), and $g(x, y_i)$ is the sum of all values forgone to merge node x with node y_i. In other words, if node x has n outward edges (pointing to nodes $y_0, y_1, \ldots, y_{n-1}$),

$$g(x, y_i) = \sum_{j=0}^{i-1} y_j + \sum_{j=i+1}^{n-1} y_j$$

This heuristic, despite its simplicity, ended up getting values almost as high as our Genetic Algorithm, and at much higher runtime speeds.

4.4 Opportunity Cost

Our second heuristic algorithm improves on our first. Instead of subtracting the total cost of all nodes forgone, as in the Total Cost heuristic, we borrowed a concept from economics: Opportunity Cost. We decided that subtracting all the forgone nodes was an inaccurate heuristic since we really could only ever pick one at a time anyway. So instead, we calculate the Opportunity Cost — the value of the single highest node forgone.

Formally, for a node x with n outward edges (pointing to nodes $y_0, y_1, \ldots, y_{n-1}$), we pick the y_i to maximize the value $f(x, y_i) = v(x, y_i) - g(x, y_i)$, where $v(x, y_i)$ is the value of the overlap between nodes x and y_i, and

$$g(x, y_i) = \max\left(\max_{j=0}^{i-1}(y_j), \max_{j=i+1}^{n-1}(y_j)\right)$$

As expected, this heuristic did prove to outperform the simpler Total Cost approach above.

5 Acknowledgements

- We based our main graph algorithm on the one included with the assignment handouts. We added heuristics onto this and used it as a starting point for our genetic algorithm.
- We researched techniques and concepts relating to Genetic Algorithms (but did not take any specific code) from the following sources:
  - `https://secure.wikimedia.org/wikipedia/en/wiki/Genetic_algorithm`
  - `http://www.rennard.org/alife/english/gavintrgb.html`
- We received no other help on this assignment