Clarke-Wright vehicle routing algorithm Implementation Report

Sam Serrels
40082367@napier.ac.uk
Edinburgh Napier University
Algorithms and Data Structures (SET09117)

1 Introduction

[Clarke and Wright 1964] [Lysgaard 1997]

Problem summary including any limitations of your solution. Large and complex video games tend to use 3rd party physics solutions, this vastly cuts down on the project development man-hours, and the maintenance thereafter. Third party physics solutions have the benefit of being battle tested out in the wild beforehand, so internal reliability is usually a given. A further benefit is that being developed solely for the purpose of being a "a good physics engine" by people who are usually experts in the field, large optimisations are already implemented. The problems arise in the implementation, the coupling of a physics engine and the existing codebase. While they are usually well coded, they are not tailor made to each game.

2 Method

Description of the experiment that you conducted. Trying to regain performance from an external physics engine can be a hard task, diving into the source code requires expert knowledge of the inner-workings of the whole system. A common path is to shape the design of the game code to conform better to the demands of the physics engine and hope that the internal optimisations will be sufficient. Often enough, they are not.

3 Results

Tables and charts that show the performance of your solution (how long does it take to run your algorithm). Demonstration that your solution is valid. Demonstration of the quality of your solution (what is the cost).

Optimising for Physics Engines Trying to regain performance from an external physics engine can be a hard task, diving into the source code requires expert knowledge of the inner-workings of the whole system. A common path is to shape the design of the game code to conform better to the demands of the physics engine and hope that the internal optimisations will be sufficient. Often enough, they are not.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis

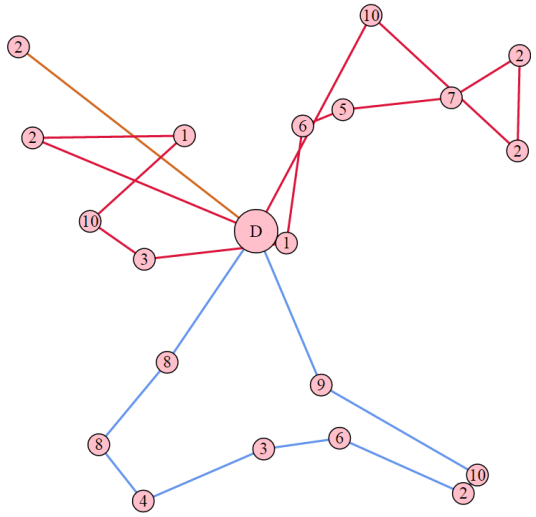

Figure 1: ***Parallel Algorithm with 20 Customers*** -

vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi

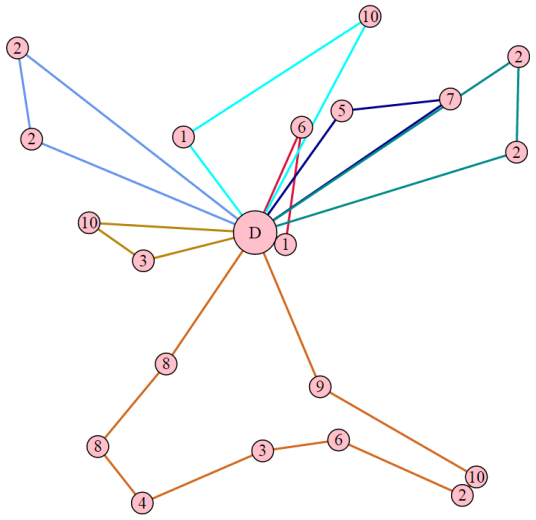

Figure 2: ***Sequential Algorithm with 20 Customers*** -

fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

4 Conclusions

Optimising for Physics Engines Summary of your results. Reflection on your performance on this assessment.

5 Appendix

References

CLARKE, G., AND WRIGHT, J. 1964. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research 12*, 4, 568–581.

LYSGAARD, J. 1997. Clarke and wright's savings algorithm http://pure.au.dk/portal-asb-student/files/36025757/bilag_e_savingsnote.pdf. *Department of Management Science and Logistics, The Aarhus School of Business*.

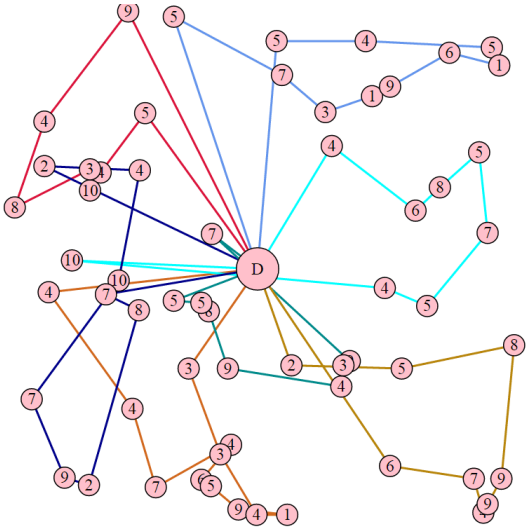

Figure 3: ***Parallel Algorithm with 50 Customers*** -

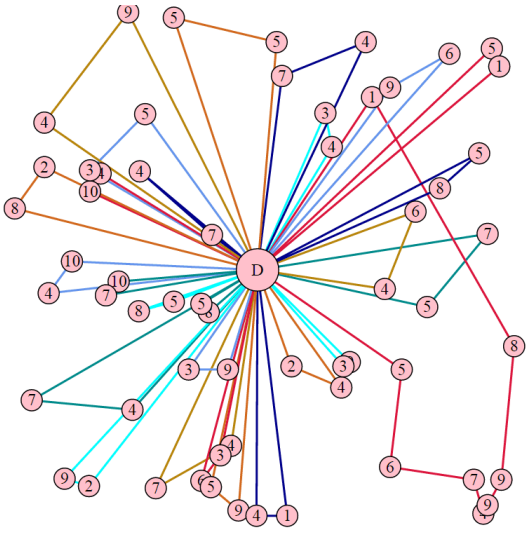

Figure 4: ***Sequential Algorithm with 50 Customers*** -

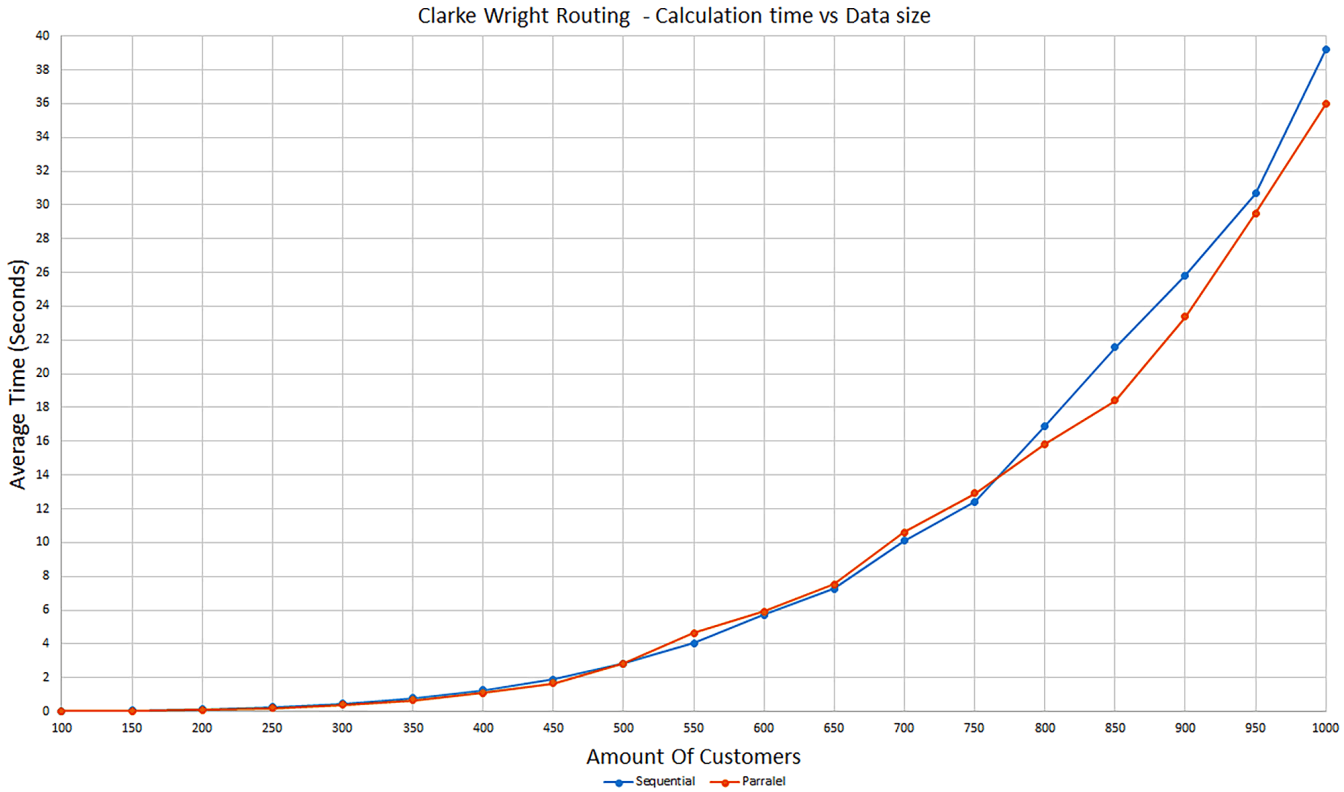

Figure 5: ***Clark Wright implementation results*** *- Average time to process each algorithm with an increasing amount of customers*

Number of Customers	Dumb solution cost	Sequential Cost	Parralel Cost	Sequential Routes	Parralel Routes	Sequential Time (Seconds)	Parralel Time (Seconds)
10	3781	1955	2027	2	2	0.000909157	0.001013164
20	7931	3749	3019	7	3	0.001404089	0.001417152
30	10302	5576	4717	11	5	0.000896818	0.000497884
40	15775	7271	4562	15	5	0.000736218	0.000602604
50	20073	9746	5753	21	7	0.001405751	0.00112024
60	23194	11918	6183	27	7	0.002299655	0.001860301
70	25925	12783	7293	31	8	0.003710384	0.002752865
80	28757	14238	7813	35	10	0.005782794	0.004625466
90	34580	17543	9202	41	11	0.011383535	0.008565139
100	38704	19375	9854	46	13	0.012820712	0.011544076
150	58323	29787	14632	71	18	0.042564373	0.035033029
200	76929	38920	17809	96	23	0.112553256	0.087635926
250	94218	47177	19586	119	28	0.236658078	0.19809618
300	116481	58912	22549	145	32	0.458011761	0.3729932
350	132529	67566	25634	171	37	0.795852936	0.655195371
400	153609	78127	31412	196	46	1.242856913	1.10756995
450	174708	88128	32206	220	51	1.909523115	1.674798974
500	188740	96343	35368	246	58	2.837620963	2.831905538
550	208973	105802	37923	270	63	4.026630142	4.657466973
600	228962	116788	43113	296	71	5.720755786	5.927701178
650	246924	126412	42089	322	72	7.251799771	7.534708523
700	266675	135277	44760	345	79	10.09592432	10.61137241
750	283723	143957	48740	370	82	12.4131069	12.93193556
800	311675	158686	51313	396	88	16.92304974	15.81973182
850	329742	168032	55694	422	97	21.56937135	18.40950886
900	347699	177212	59699	446	106	25.80634481	23.36173086
950	361993	184769	59529	472	105	30.69257549	29.51002702
1000	390783	199258	61701	497	109	39.20094816	35.98109894

Table 1: *Results of all tests carried out on both versions of the Clarke Wright algorithm*

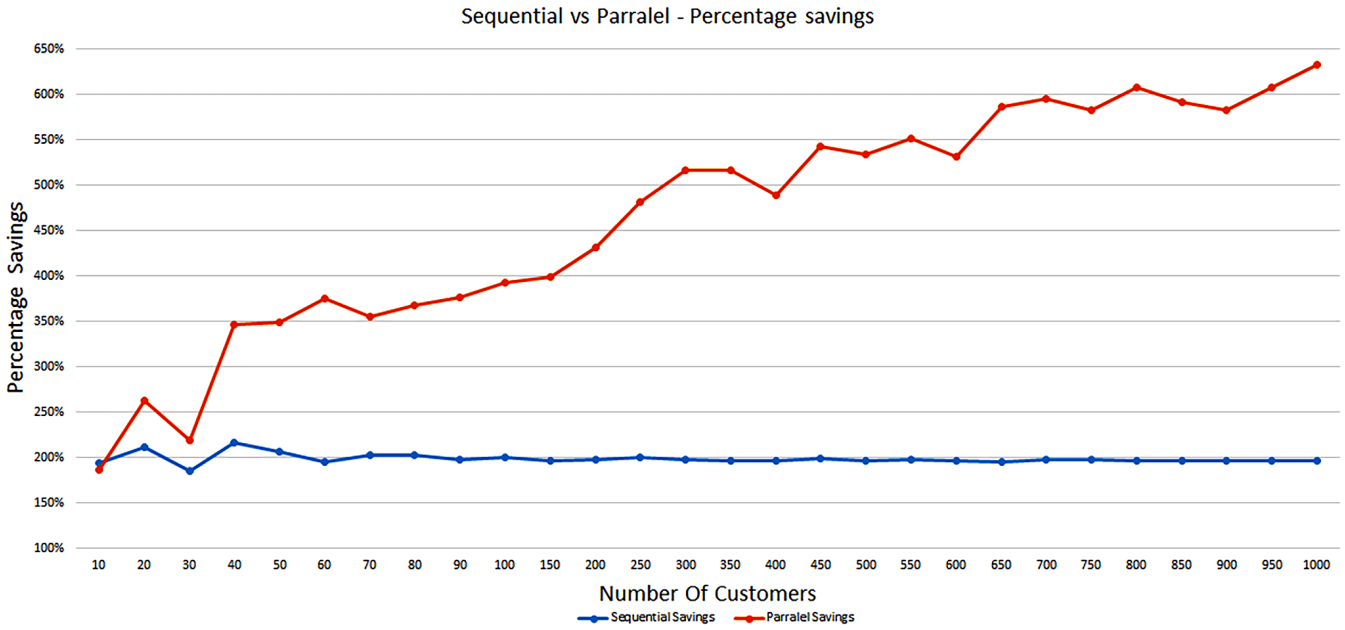

Figure 6: ***Clark Wright Percentage Savings*** *- The Percentage distance cost saved by each algorithm, based from sending a single truck to each customer*

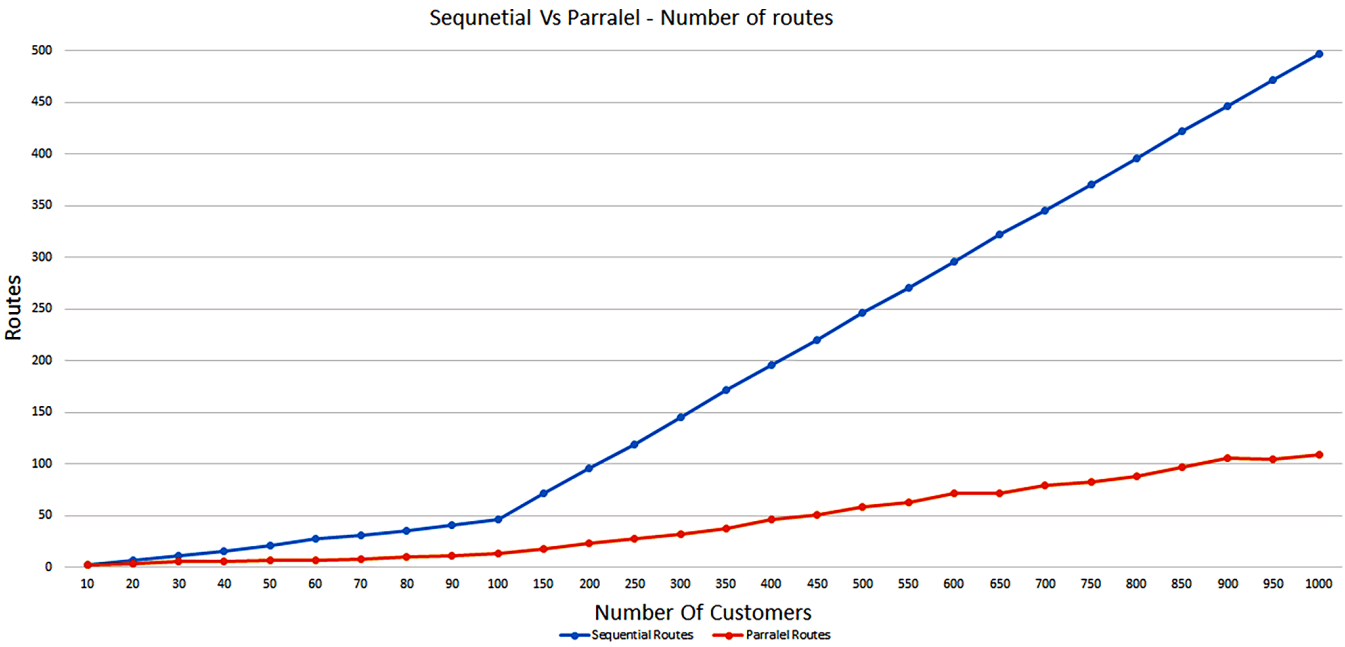

Figure 7: ***Clark Wright Routes*** *- Number of routes required by each algorithm. X-axis is not a uniform scale.*

6 Code

6.1 ClarkeWright.java

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;

class Route implements Comparable<Route>
{
   private int _capacity;
   private int _weight;
   private double _cost;
   private double _savings;
   public ArrayList<Customer> customers;

   private void calculateSavings(){
      double originalCost = 0;
      double newCost = 0;
      double tempcost =0;
      Customer prev = null;

      //Foreach customer in the route:
      for(Customer c:customers){
         // Distance from Depot
         tempcost = Math.sqrt((c.x*c.x)+(c.y*c.y));
         originalCost += (2.0*tempcost);

         if(prev != null){
            // Distance from previous customer to this customer
            double x = (prev.x - c.x);
            double y = (prev.y - c.y);
            newCost += Math.sqrt((x*x)+(y*y));
         }else{
            //If this is the first customer in the route, no change
            newCost += tempcost;
         }
         prev = c;
      }
      newCost += tempcost;
      _cost = newCost;
      _savings = originalCost - newCost;
   }

   public Route(int capacity){
      _capacity = capacity;
      customers = new ArrayList<Customer>();
      _weight =0;
      _cost= 0;
      _savings =0;
   }

   public void addCustomer(Customer c, boolean order){
      //Add customer to the start or end of the route?
      if(order){
         customers.add(0,c);
      }else{
         customers.add(c);
      }

      if(c.c > _capacity){
         System.out.println("Customer order too large");
      }

      _weight += c.c;

      if(_weight > _capacity){
         System.out.println("Route Overloaded");
      }

      calculateSavings();
   }

   public double getSavings(){
      return _savings;
   }
   public double getCost(){
      return _cost;
   }
   public int getWeight(){
      return _weight;
   }
   public int compareTo(Route r) {
      return Double.compare(r.getSavings(), this._savings);
   }

}

//
//##Sequential solver##
//

public class ClarkeWright
{
   public static int truckCapacity = 0;

   public static ArrayList<List<Customer>> solve(ArrayList<Customer> customers){
      ArrayList<List<Customer>> solution = new ArrayList<List<Customer>>();
      HashSet<Customer> abandoned = new HashSet<Customer>();

      //calculate the savings of all the pairs
      ArrayList<Route> pairs = new ArrayList<Route>();

      for(int i=0; i<customers.size(); i++){
         for(int j=i+1; j<customers.size(); j++){
            Route r = new Route(truckCapacity);
            r.addCustomer(customers.get(i),false);
            r.addCustomer(customers.get(j),false);
            pairs.add(r);
         }
      }
      //order pairs by savings
      Collections.sort(pairs);

      HashSet<Route> routes = new HashSet<Route>();
      routes.add(pairs.get(0));
      pairs.remove(0);

      //start combining pairs into routes
      for(Route ro :routes)
      {
         outerloop: for(int i=0; i<pairs.size(); i++){
            Route r = pairs.get(i);
            Customer c1 = r.customers.get(0);
            Customer c2 = r.customers.get(r.customers.size()-1);
            Customer cr1 = ro.customers.get(0);
            Customer cr2 = ro.customers.get(ro.customers.size()-1);

            boolean edge = false;
            for(int a=0; a<2;a++)
            {
               edge = !edge;
               Customer e1 = (!edge) ? c1 : c2;
               Customer e2 = (edge) ? c1 : c2;
               //do they have any common nodes?
               if(e1 == cr1 ||e1 == cr2){
                  //could we combine these based on weight?
                  if(e2.c + ro.getWeight() <= truckCapacity){
                     //Does route already contain BOTH these nodes?
                     if(!ro.customers.contains(e2)){
                        //no, but is it in another route already?
                        boolean istaken = false;
                        for(Route rr :routes)
                        {
                           if(rr.customers.contains(e2)){
                              istaken = true;
                              break;
                           }
                        }
                        if(!istaken){
                           //No other route have this, add to route
                           if(e1 == cr1){
                              ro.addCustomer(e2, true);
```

```
                        }else{
                            ro.addCustomer(e2, false);
                        }
                    }
                }
                abandoned.remove(e2);
                pairs.remove(r);
                i--;
                continue outerloop;
            }
        }
    }

    //If we reach here, the pair hasn't been added to any routes
    boolean a = false;
    boolean b = false;
    for(Route rr :routes){
        if(rr.customers.contains(c1)){
            a = true;
        }
        if(rr.customers.contains(c2)){
            b = true;
        }
    }
    if(!(a||b)){
        //no routes have any of these customers, make new route
        abandoned.remove(c1);
        abandoned.remove(c2);
        routes.add(r);
    }else{
        //Some routes have some of these customers already
        if(!a){
            abandoned.add(c1);
        }
        if(!b){
            abandoned.add(c2);
        }
    }
    pairs.remove(r);
    i--;
  }

}

//A Customer can be left over due to capacity constraints
outerloop:for(Customer C:abandoned){
  //we could tack this onto the end of a route if it would fit
  for(Route r:routes){
    if(r.getWeight() + C.c < truckCapacity)
    {
      //would this be more efficient than sending a new truck?
      Customer[] cca={r.customers.get(r.customers.size()-1),
      r.customers.get(0)};
      for(Customer cc:cca){
        double X = C.x - cc.x;
        double Y = C.y - cc.y;
        if(Math.sqrt((X*X)+(Y*Y))
        < Math.sqrt((C.x*C.x)+(C.y*C.y)))
        {
          r.addCustomer(C, false);
          break outerloop;
        }
      }
    }
  }

  //Send a new truck, just for this Customer
  ArrayList<Customer> l = new ArrayList<Customer>();
  l.add(C);
  solution.add(l);
}

//output
for(Route r:routes){
  ArrayList<Customer> l = new ArrayList<Customer>();
  l.addAll(r.customers);
  solution.add(l);
}
return solution;
}

//
//##Parallel solver##
//

public static ArrayList<List<Customer>> solveP(ArrayList<Customer> customers){
  ArrayList<List<Customer>> solution = new ArrayList<List<Customer>>();
  HashSet<Customer> abandoned = new HashSet<Customer>();

  //calculate the savings of all the pairs
  ArrayList<Route> pairs = new ArrayList<Route>();

  for(int i=0; i<customers.size(); i++){
    for(int j=i+1; j<customers.size(); j++){
      Route r = new Route(truckCapacity);
      r.addCustomer(customers.get(i),false);
      r.addCustomer(customers.get(j),false);
      pairs.add(r);
    }
  }
  //order pairs by savings
  Collections.sort(pairs);

  HashSet<Route> routes = new HashSet<Route>();
  routes.add(pairs.get(0));
  pairs.remove(0);

  //start combining pairs into routes
  outerloop: for(int j=0; j<pairs.size(); j++){
    Route r = pairs.get(j);
    Customer c1 = r.customers.get(0);
    Customer c2 = r.customers.get(r.customers.size()-1);

    for(Route ro :routes)
    {
      Customer cr1 = ro.customers.get(0);
      Customer cr2 = ro.customers.get(ro.customers.size()-1);
      boolean edge = false;
      for(int a=0; a<2;a++)
      {
        edge = !edge;
        Customer e1 = (!edge) ? c1 : c2;
        Customer e2 = (edge) ? c1 : c2;
        //do they have any common nodes?
        if(e1 == cr1 || e1 == cr2){
          //could we combine these based on weight?
          if(e2.c + ro.getWeight() <= truckCapacity){
            //Does route already contain BOTH these nodes?
            if(!ro.customers.contains(e2)){
              //no, but is it in another route already?
              boolean istaken = false;
              for(Route rr :routes){
                if(rr.customers.contains(e2)){
                  istaken = true;
                  break;
                }
              }
              if(!istaken){
                //No other route have this, add to route.
                if(c1 == cr1){
                  ro.addCustomer(e2, true);
                }else{
                  ro.addCustomer(e2, false);
                }
              }
            }
            abandoned.remove(e2);
            pairs.remove(r);
            j--;
            continue outerloop;
          }
        }
      }
    }

  }
```

```
306
307         //If we reach here, the pair hasn't been added to any routes
308         boolean a = false;
309         boolean b = false;
310         for(Route ro :routes){
311            if(ro.customers.contains(c1)){
312               a = true;
313            }
314            if(ro.customers.contains(c2)){
315               b = true;
316            }
317         }
318         if(!(a||b)){
319            //no routes have any of these customers, make new route
320            abandoned.remove(c1);
321            abandoned.remove(c2);
322            routes.add(r);
323         }else{
324            //Some routes have some of these customers already
325            if(!a){
326               abandoned.add(c1);
327            }
328            if(!b){
329               abandoned.add(c2);
330            }
331         }
332         pairs.remove(r);
333         j--;
334
335      }
336
337      //A Customer can be left over due to capacity constraints
338      outerloop:for(Customer C:abandoned){
339         //we could tack this onto the end of a route if it would fit
340         for(Route r:routes){
341            if(r.getWeight() + C.c < truckCapacity)
342            {
343               //would this be more efficient than sending a new truck?
344               Customer[] cca={r.customers.get(r.customers.size()-1),
345               r.customers.get(0)};
346               for(Customer cc:cca){
347                  double X = C.x - cc.x;
348                  double Y = C.y - cc.y;
349                  if(Math.sqrt((X*X)+(Y*Y))
350                  < Math.sqrt((C.x*C.x)+(C.y*C.y)))
351                  {
352                     r.addCustomer(C, false);
353                     break outerloop;
354                  }
355               }
356            }
357         }
358
359         //Send a new truck, just for this Customer
360         ArrayList<Customer> l = new ArrayList<Customer>();
361         l.add(C);
362         solution.add(l);
363      }
364
365      //output
366      for(Route r:routes){
367         ArrayList<Customer> l = new ArrayList<Customer>();
368         l.addAll(r.customers);
369         solution.add(l);
370      }
371      return solution;
372   }
373 }
```

6.2 VRSolution.java

Lines 20 to 28

```
1
2    //Students should implement another solution
3    public void clarkeWrightSolution(boolean b){
4       ClarkeWright cw = new ClarkeWright();
5       cw.truckCapacity = prob.depot.c;
6       if(b){
7          this.soln = cw.solveP(prob.customers);
8       }else{
```

6.3 Experiment.java

```
1 import java.util.*;
2 public class Experiment {
3
4    public static void main(String[] args)throws Exception{
5       String outdir = "output/";
6       String problemdir = "tests/";
7       String [] probs = {
8             "rand00010",
9             "rand00020",
10            "rand00030",
11            "rand00040",
12            "rand00050",
13            "rand00060",
14            "rand00070",
15            "rand00080",
16            "rand00090",
17            "rand00100",
18            "rand00150",
19            "rand00200",
20            "rand00250",
21            "rand00300",
22            "rand00350",
23            "rand00400",
24            "rand00450",
25            "rand00500",
26            "rand00550",
27            "rand00600",
28            "rand00650",
29            "rand00700",
30            "rand00750",
31            "rand00800",
32            "rand00850",
33            "rand00900",
34            "rand00950",
35            "rand01000",
36            "fail00002",
37            "fail00004"
38            };
39       for (String f:probs){
40          VRProblem vrp = new VRProblem(problemdir+f+"prob.csv"↩
          );
41          VRSolution vrs = new VRSolution(vrp);
42          System.out.print(vrp.size()+",");
43          for(int i=0;i<50;i++){
44             long start = System.nanoTime();
45             vrs.clarkeWrightSolution(true);
46             long delta = System.nanoTime()-start;
47             System.out.print(delta+",");
48          }
49          System.out.print("\n");
50       }
51
52    }
53 }
```