Reduce your consumption, ecological footprint, costs, and energy dependency.

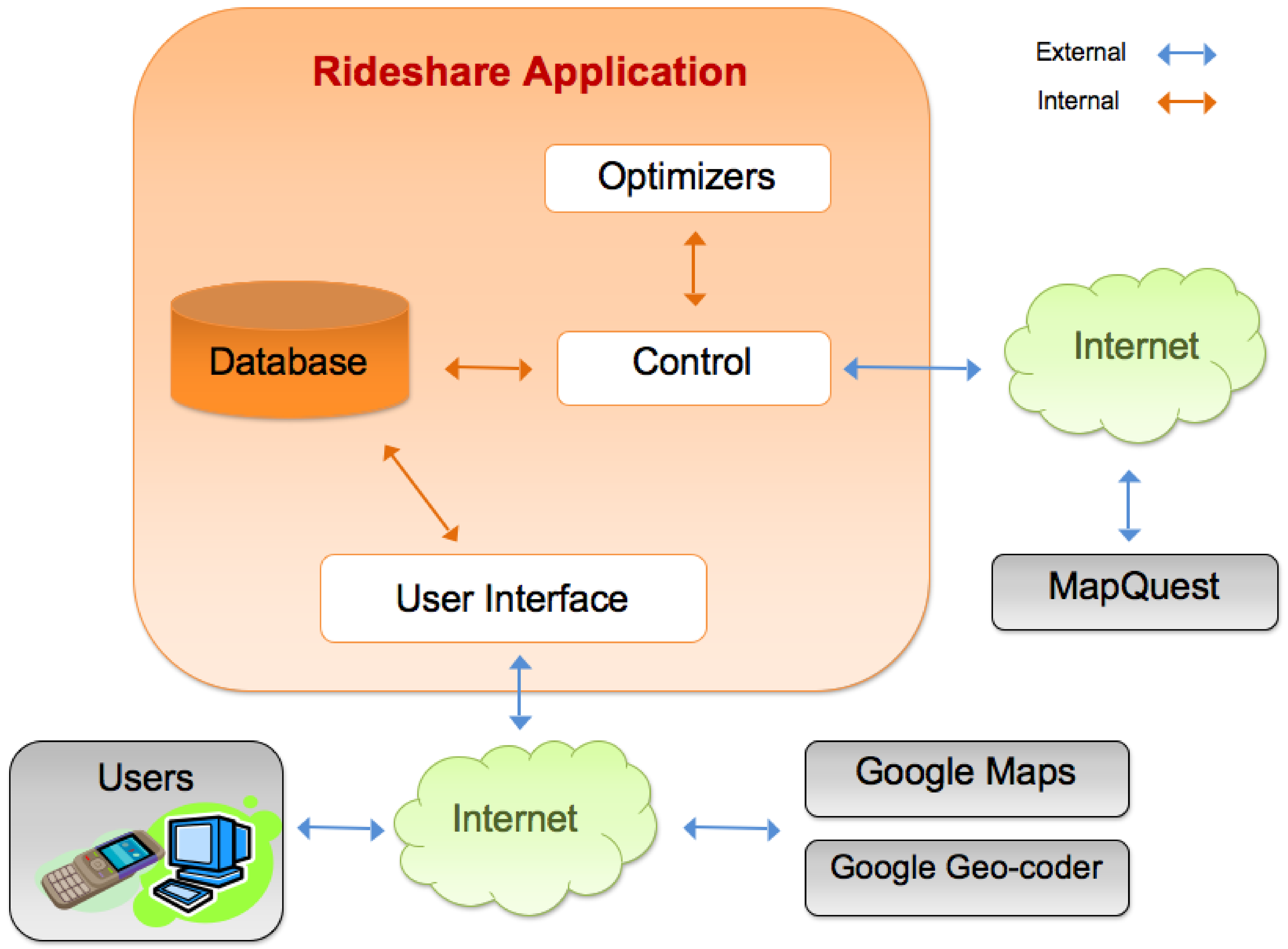
Share and cooperate.
Cooperation has a way of benefiting everyone involved.
ride with your friends, befriend your riders.

Hundreds of millions of trips are taken every day from places we all live to places we all go.

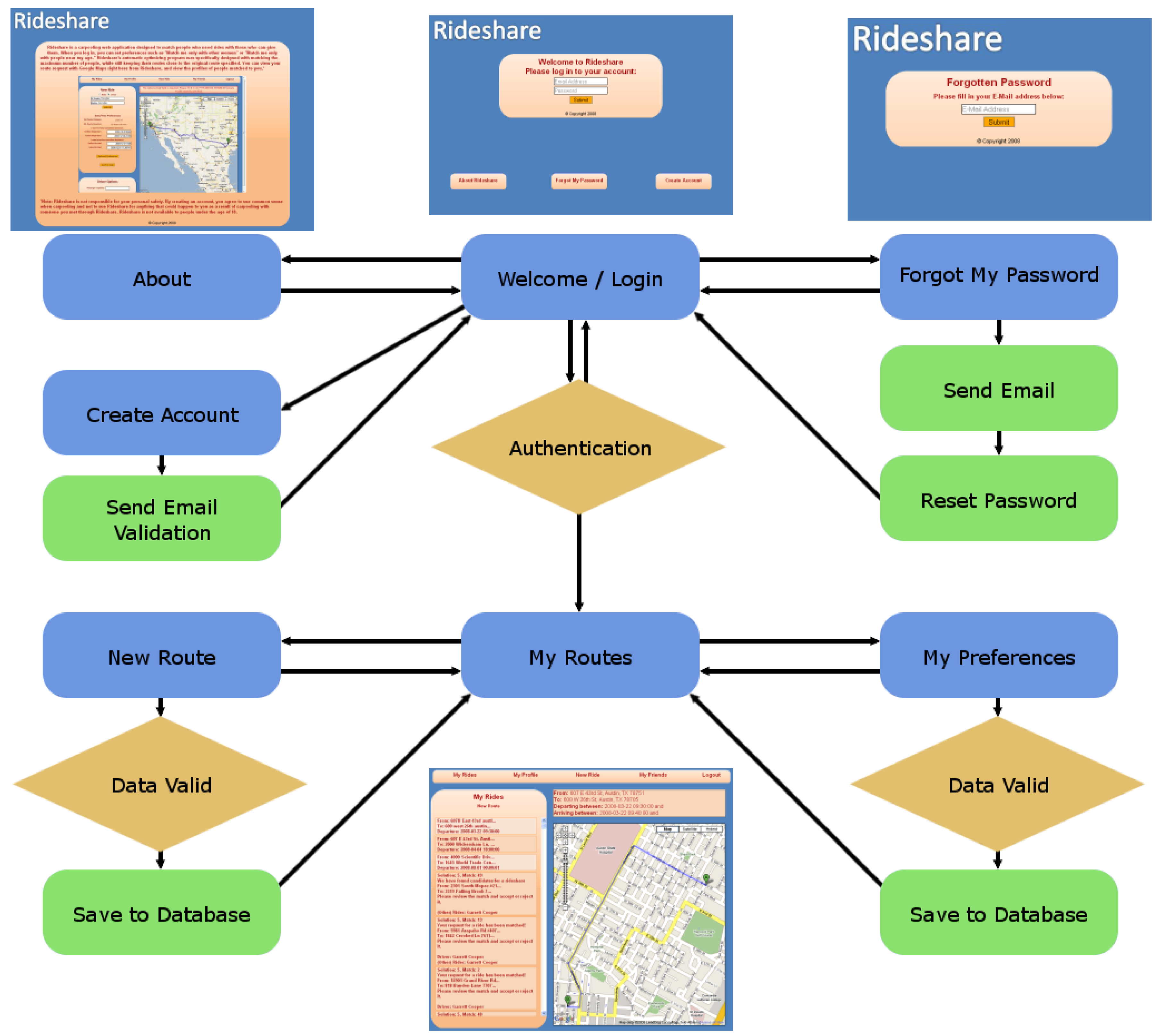
The energy costs and global consequences of overconsumption don't seem to be falling.

Add your trip to the lot, and find someone going your way.
Enter your ride locations, schedule, and preferences.
If we find matching requests, we will inform you by email and rss feed. It's that simple.

In addition to providing you with results, we generate matches in a way that allows more people to be matched. This means that we improve the likelihood that you get results, and reduce everyone's ecological footprint even further. :)


## Rideshare Sitemap




## Why This Problem is Difficult

The problem of finding the best possible overall driver and rider matching (i.e. global best solution) is just a version of a well-known problem called the Vehicle Routing Problem (VRP), which is stated as:

> "The Vehicle Routing Problem or VRP is a combinatorial optimization and nonlinear programming problem seeking to service a number of customers with a fleet of vehicles. Proposed by Dantzig and Ramser in 1959, VRP is an important problem in the fields of transportation, distribution and logistics. Often the context is that of delivering goods located at a central depot to customers who have placed orders for such goods. Implicit is the goal of minimizing the cost of distributing the goods. **Many methods have been developed for searching for good solutions to the problem, but for all but the smallest problems, finding global minimum for the cost function is computationally complex.**"
>
> –Wikipedia entry for Vehicle Routing Problem

It is classified as an NP-Hard problem, meaning that no known polynomial-time algorithms for solving it exist. If we approached solving it in the naïve sense, i.e. try every possible solution, reaching a solution would take an amount of time that exceeds the history of the known universe.

To top it off, Rideshare includes scheduling and preference matching, and allows for arbitrarily many depots.

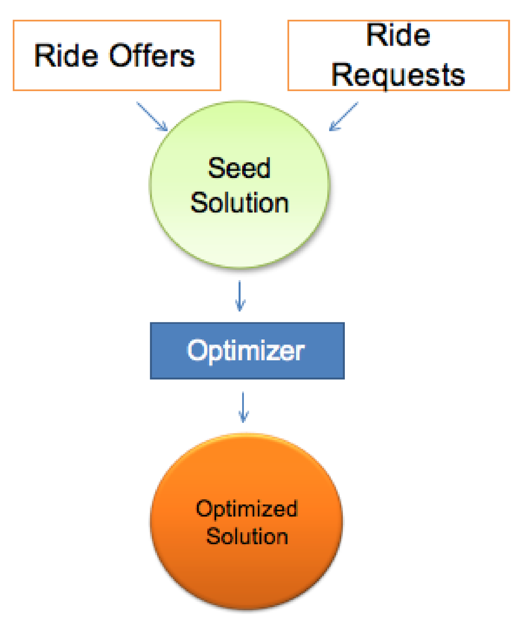
Our approach involved using three different algorithms which were developed by many of our wise predecessors, and adapted by us.

## How We Find an Optimal Solution

In order to match users, we utilize three different Optimizers.

An Optimizer takes in a set of drivers and riders, and creates a "seed solution."

A Solution sets up Rideshares which initially contain only the drivers.




The Optimizer applies an algorithm to find an optimal result. It compares different solutions using a scoring function, and generates the highest scoring solution it can, returning it as the optimized solution.

The score of a given solution is:

Number of Riders Matched + Number of Drivers Matched + ∑(drivers' seed distances)/∑(rideshares' final distances)

The whole part of the score represents the primary criteria for comparing solutions. It consists of the total number of users matched.

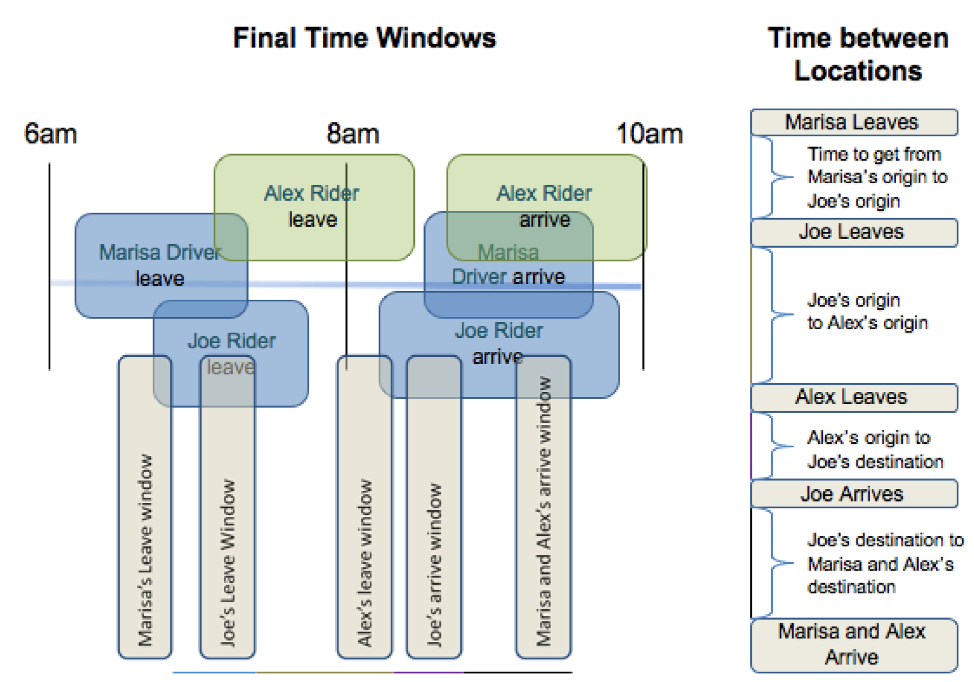
Our secondary criteria for a good solution is the fraction. It gives us an idea of the average driver deviation from their original route. A smaller fraction means shorter deviations.

## Compatibility - Can I Add This Rider?

How to take a Rideshare and rider and ascertain whether they are compatible. This might sound like a simple enough task, but it involves much more than meets the eye.

Of course, checking for age and gender preferences is simple enough. But schedule and route compatibility are a different story. It turns out that the first can be done if we apply a clever observation about the Driver's worst case leave window, and the second is yet another instance of our good friend the Vehicle Routing Problem.

Suppose we have Marisa Driver and Joe Rider in a match, and we would like to see if adding a Alex Rider would create an incompatible ride schedule or route length.




To do this we need to find out the best way of ordering the riders, so as to minimize route length. Since the capacity of a vehicle is fairly small, we find the best route by simplytrying all combinations.

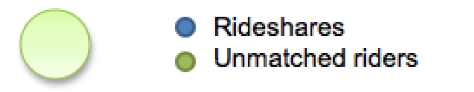
Once we have an optimal route for this group, we verify that the itinerary doesn't conflict with anyone's schedule. We do this by narrowing the leave and arrive windows until we reach a worst case leave window for the driver.

We then add to this leave window the time necessary to get from point to point, each time verifying that the users' time windows do not conflict with the arrival window for that point.

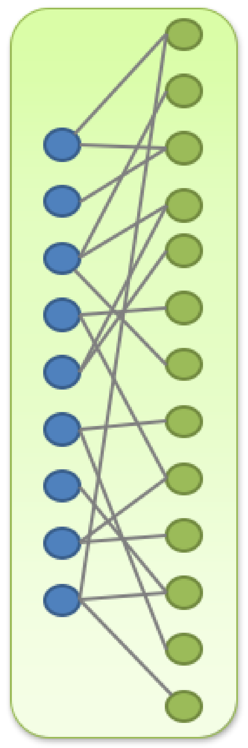
# Algorithms

## Bipartite Matching Optimizer

1. Get seed solution



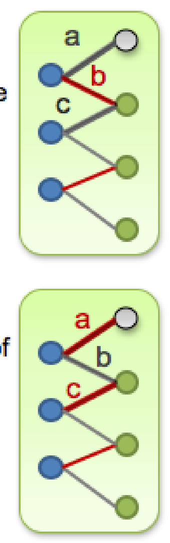

2. Connect compatible riders and rideshares with unmatched edges. (Read "Compatibility – Can I Add This Rider?" to see how we do this)



3. For a random unmatched rider, check if there is an augmenting path leading from that rider.

An augmenting path is a path that begins and ends with unmatched edges.

Such a path allows us to add one edge to the matching, since the number of unmatched edges is one greater than the number of matched edges.

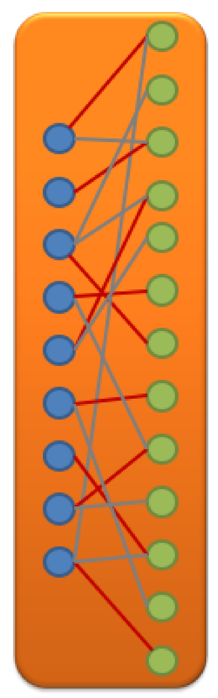



The path **a->b->c** is an augmenting path. It allows us to remove **b** and add **a** and **c** to the matching.

4. Exchange all unmatched and matched edges on the augmenting path.

5. Repeat 3-4 until all unmatched riders have been tried.

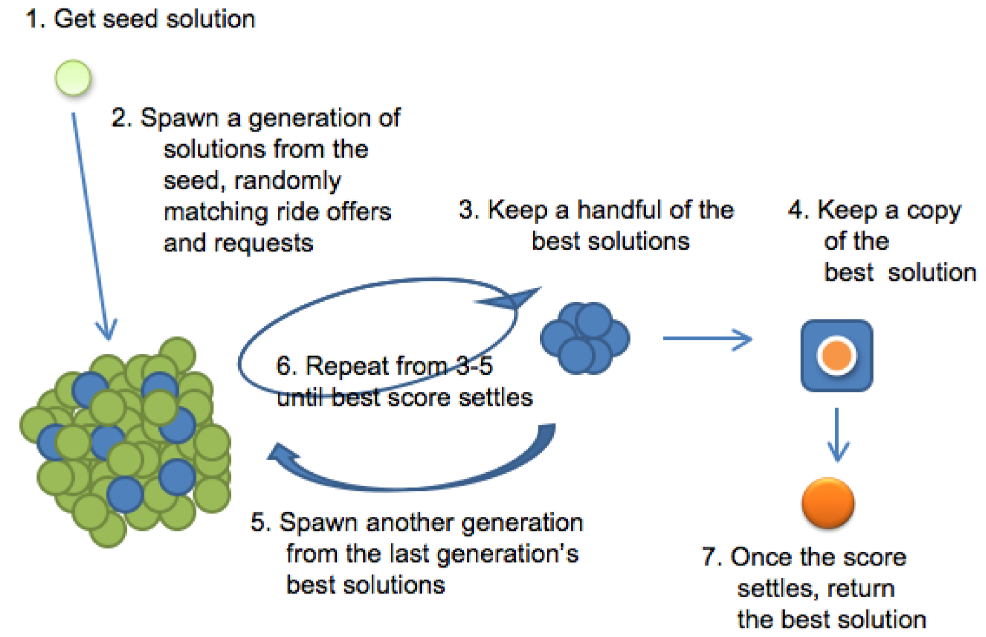
This produces a maximal matching:



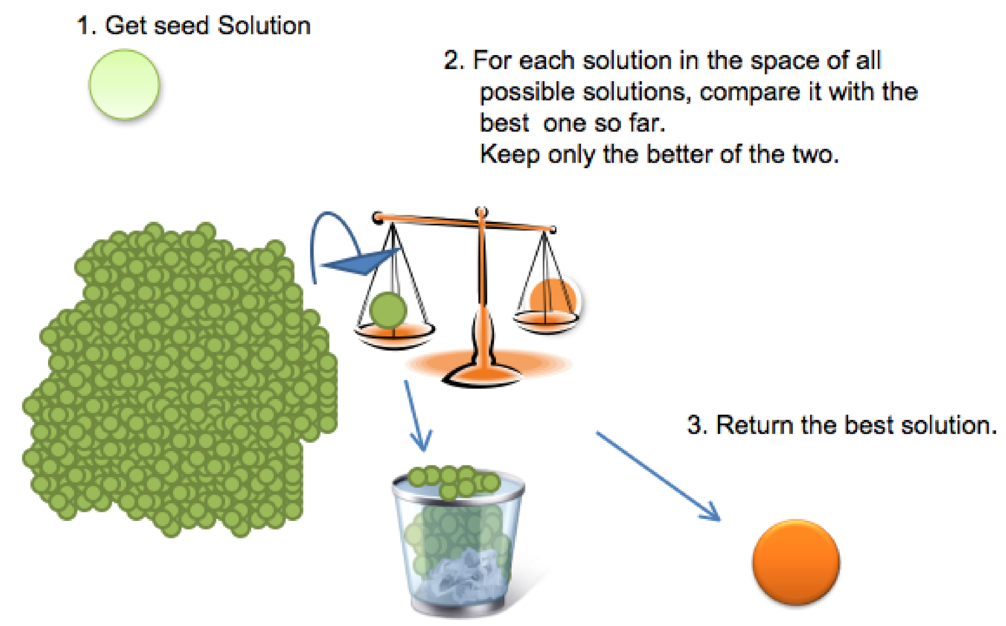
6. Add all matched riders to their rideshares.

7. Repeat 2-6, until no more augmenting paths exist.

## Genetic Optimizer




## Brute Force Optimizer

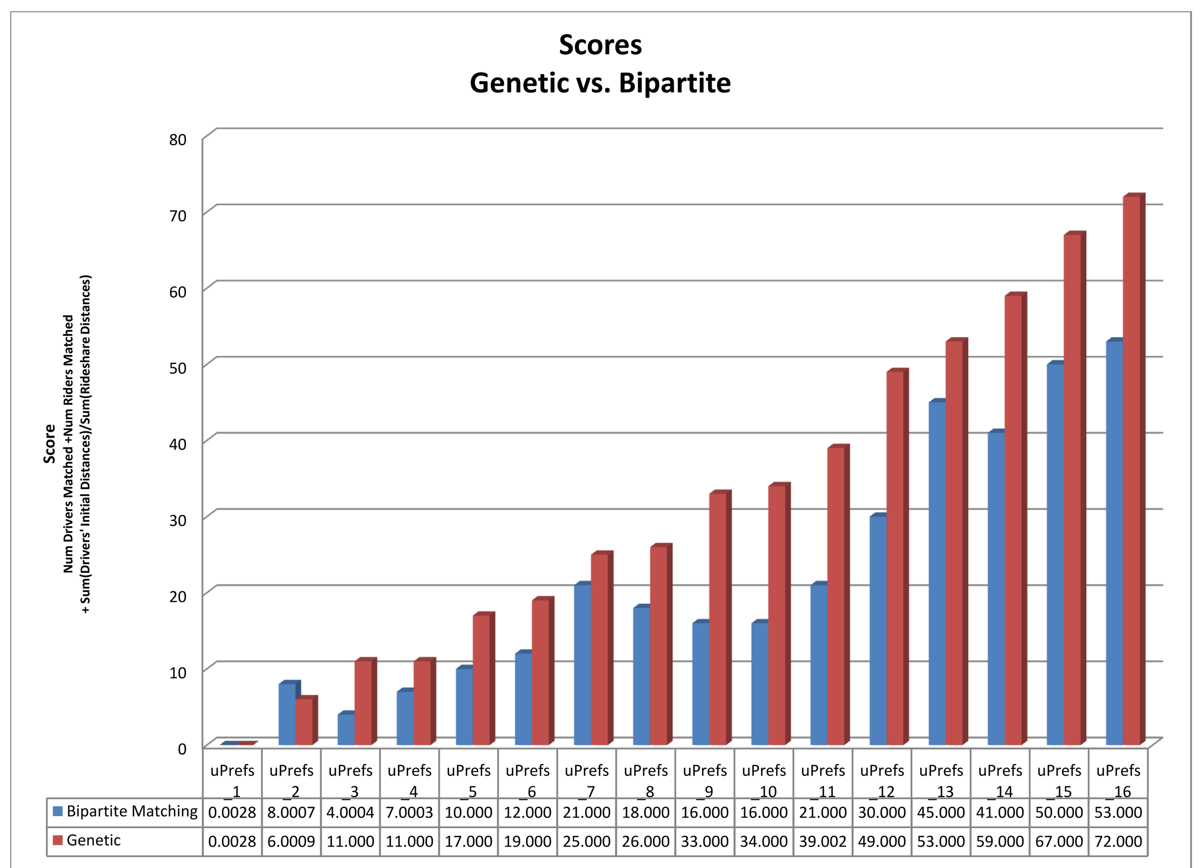



# Pseudo-code For the Algorithms

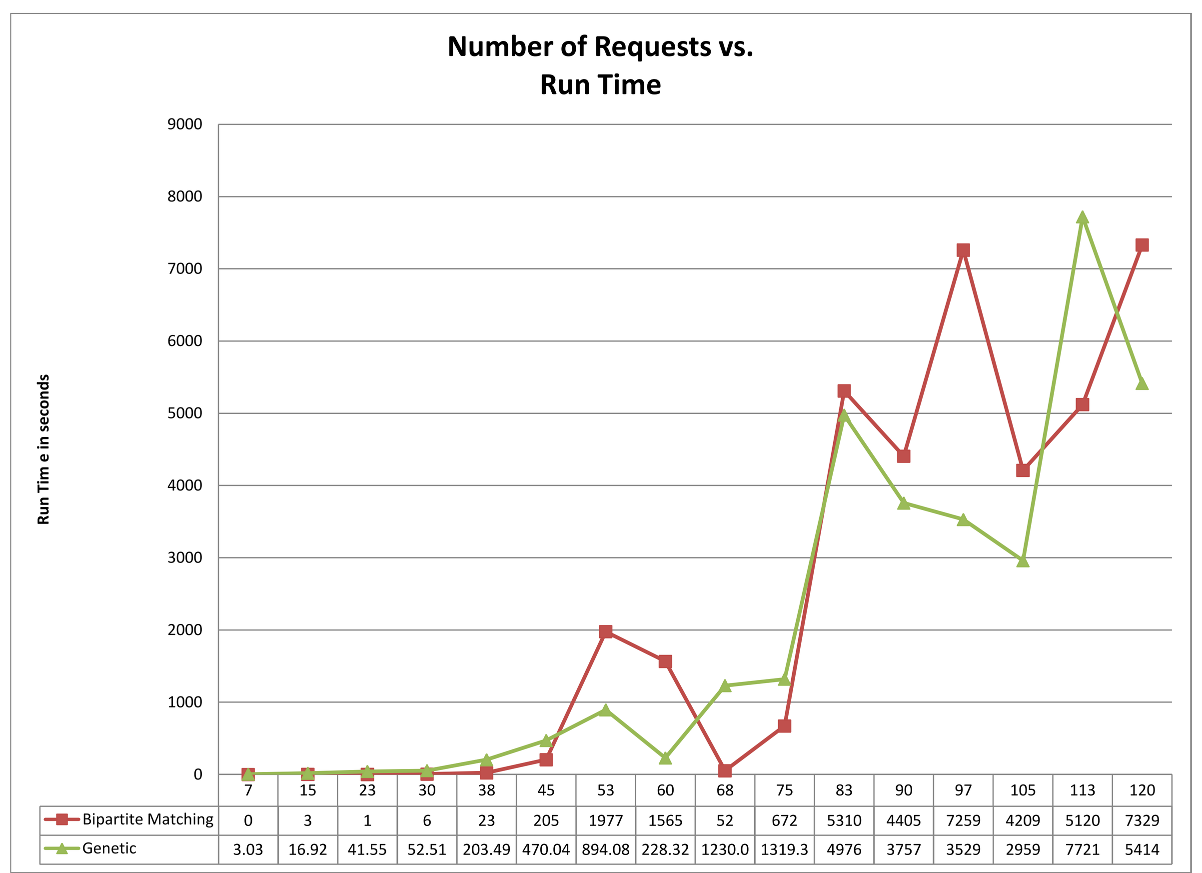
```
BipartiteMatchingOptimizer( Solution S )
    Rideshares = S.Drivers()
    last_score = 0.0
    while( score(S) > last_score )
            last_score = score(S)
            CompatibleEdges = unmatchedEdges(Rideshares, S.Riders)
            while( S.hasAugmentingPath() )
                    augmentingPath = GetAugmentingPath(S)
                    for each edge in augmentingPath do
                            if( edge = matched ) edge = unmatched
                            if( edge = unmatched ) edge = matched
            end while
            for each edge in Edges.matchedEdges() do
                    add edge ->rider to edge->rideshare
                    remove edge->rider from S.Riders
    end while
    return S

GetAugmentingPath( Solution  S )
    for each rider in UnmatchedRiders(S) do
            traverse unmatched edges from rider to rideshares
            traverse matched edges from rideshares to riders
            return augmentingPath if an unmatched rideshare is reached
    return false
```

```
GeneticOptimize( Solution S )
    Solutions  Population, Best
    Population.GenerateNextPopulation(S, NULL)
    time = amount of time to run
    time.countDown()
    while(time not expired)
            for each solution in Population do
            score(solution)
            Best = getBest(Population)
            GenerateNextPopulation(Population, Best)
    end while

GenerateNextPopulation( Solutions Population, Best )
    if( Best = NULL)
            for each solution in Population
            solution = S
    else
            for each solution in Population
            randomly assign riders to rideshares in solution
```

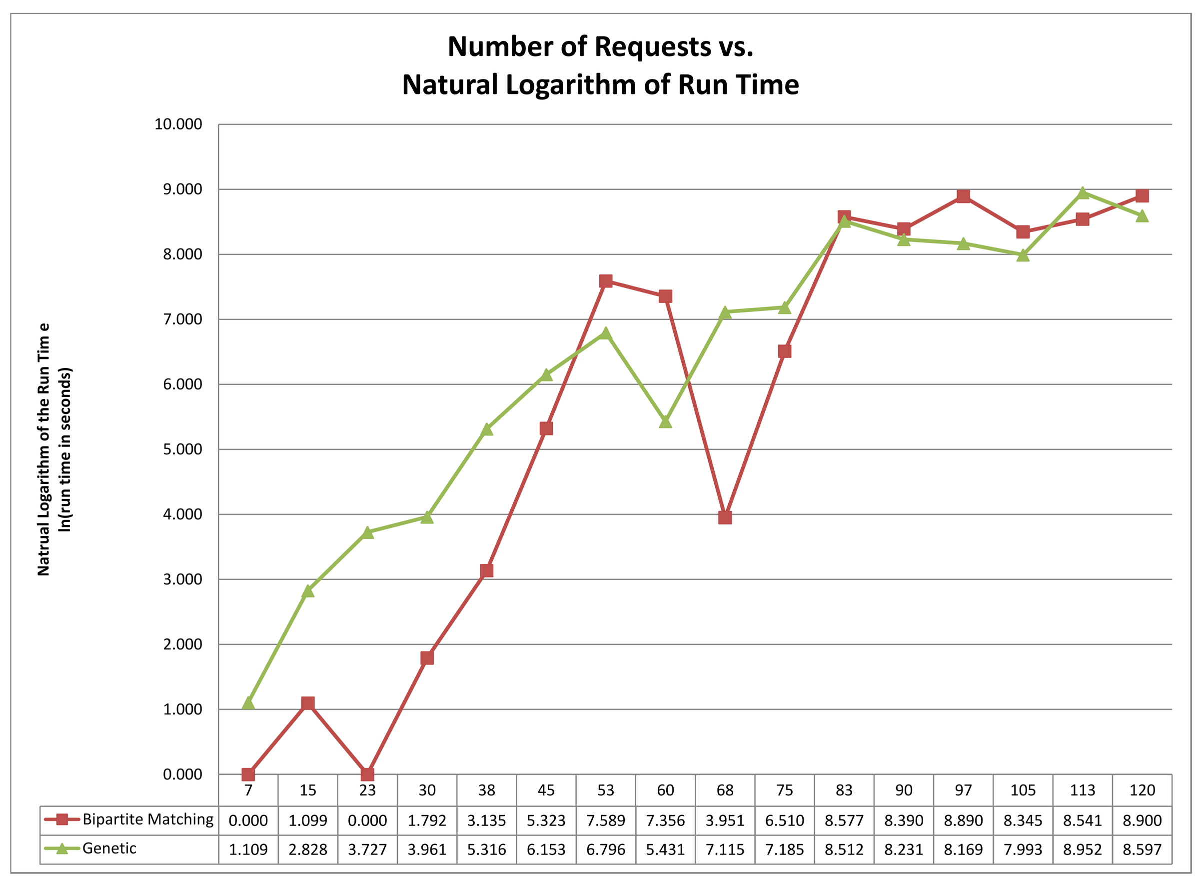
```
BruteForceOptimizer( Solution S )
    best_solution = S
    best_score = score(S)
    for each solution in solution_space(S)
do
            score = score(solution)
            If(score > best_score)
            best_solution = solution
            best_score = score
    return best_solution

Score( Solution S )
    score = S.numRidersMatched() +  S.numDriversMatched()
    original_route_lengths  = Sum( drivers' seed route lengths )
    total_route_length = Sum( final route lengths )
    deviation = original_route_lengths/total_route_length
    score += deviation
    return score
```



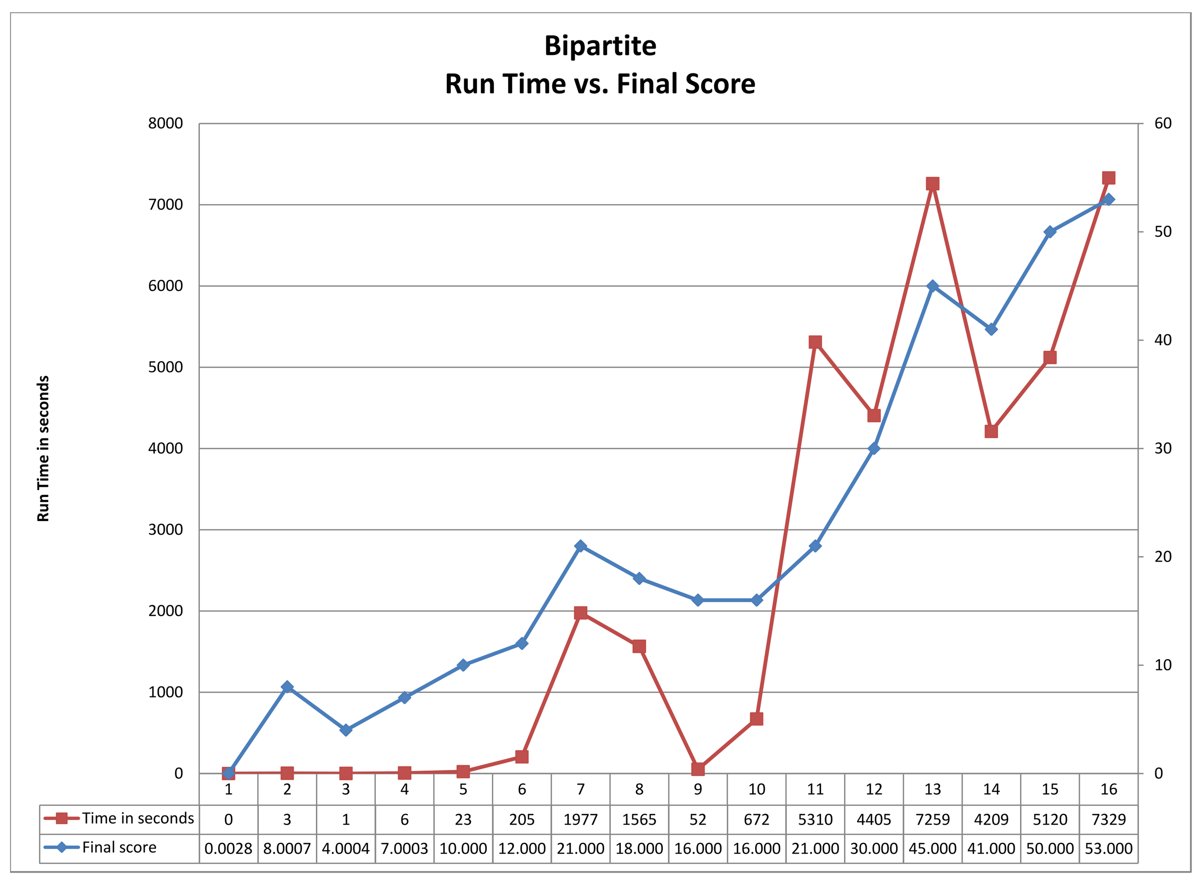

| | uPrefs_1 | uPrefs_2 | uPrefs_3 | uPrefs_4 | uPrefs_5 | uPrefs_6 | uPrefs_7 | uPrefs_8 | uPrefs_9 | uPrefs_10 | uPrefs_11 | uPrefs_12 | uPrefs_13 | uPrefs_14 | uPrefs_15 | uPrefs_16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bipartite Matching | 0.0028 | 8.0007 | 4.0004 | 7.0003 | 10.000 | 12.000 | 21.000 | 18.000 | 16.000 | 16.000 | 21.000 | 30.000 | 45.000 | 41.000 | 50.000 | 53.000 |
| Genetic | 0.0028 | 6.0009 | 11.000 | 11.000 | 17.000 | 19.000 | 25.000 | 26.000 | 33.000 | 34.000 | 39.002 | 49.000 | 53.000 | 59.000 | 67.000 | 72.000 |



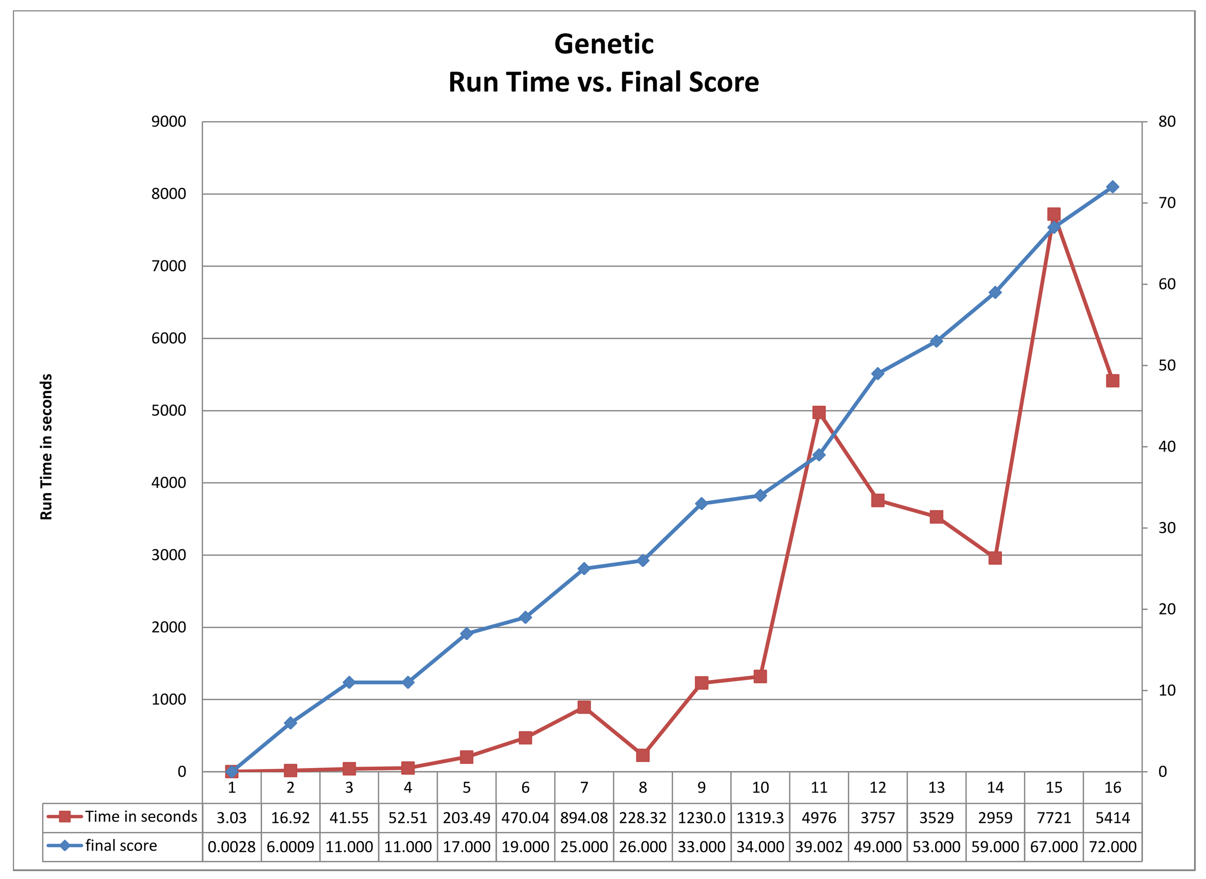

| | 7 | 15 | 23 | 30 | 38 | 45 | 53 | 60 | 68 | 75 | 83 | 90 | 97 | 105 | 113 | 120 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bipartite Matching | 0 | 3 | 1 | 6 | 23 | 205 | 1977 | 1565 | 52 | 672 | 5310 | 4405 | 7259 | 4209 | 5120 | 7329 |
| Genetic | 3.03 | 16.92 | 41.55 | 52.51 | 203.49 | 470.04 | 894.08 | 228.32 | 1230.0 | 1319.3 | 4976 | 3757 | 3529 | 2959 | 7721 | 5414 |




| | 7 | 15 | 23 | 30 | 38 | 45 | 53 | 60 | 68 | 75 | 83 | 90 | 97 | 105 | 113 | 120 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bipartite Matching | 0.000 | 1.099 | 0.000 | 1.792 | 3.135 | 5.323 | 7.589 | 7.356 | 3.951 | 6.510 | 8.577 | 8.390 | 8.890 | 8.345 | 8.541 | 8.900 |
| Genetic | 1.109 | 2.828 | 3.727 | 3.961 | 5.316 | 6.153 | 6.796 | 5.431 | 7.115 | 7.185 | 8.512 | 8.231 | 8.169 | 7.993 | 8.952 | 8.597 |




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 0 | 3 | 1 | 6 | 23 | 205 | 1977 | 1565 | 52 | 672 | 5310 | 4405 | 7259 | 4209 | 5120 | 7329 |
| Final score | 0.0028 | 8.0007 | 4.0004 | 7.0003 | 10.000 | 12.000 | 21.000 | 18.000 | 16.000 | 16.000 | 21.000 | 30.000 | 45.000 | 41.000 | 50.000 | 53.000 |




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time in seconds | 3.03 | 16.92 | 41.55 | 52.51 | 203.49 | 470.04 | 894.08 | 228.32 | 1230.0 | 1319.3 | 4976 | 3757 | 3529 | 2959 | 7721 | 5414 |
| final score | 0.0028 | 6.0009 | 11.000 | 11.000 | 17.000 | 19.000 | 25.000 | 26.000 | 33.000 | 34.000 | 39.002 | 49.000 | 53.000 | 59.000 | 67.000 | 72.000 |

## Implementation

```
Optimizer
    optimize() = 0
    getBestScore() = 0
    getBestSolution() = 0
    double score()
    vector<Solution> solutionHistory
```

```
BruteForceOptimizer
    optimize()
    getBestScore()
    getBestSolution()
    itereate()
```

```
GeneticOptimizer
    optimize()
    getBestScore()
    getBestSolution()
    generatePopulation()
```

```
BipartiteOptimizer
    optimize()
    getBestScore()
    getBestSolution()
    single_optimize()
    iterate()
    getAugmentingPath()
    updateSolution()
    updateEdges()
    initEdges()
    riderIterate()
    matchIterate()
```

```
Optimizers:
    Optimizer
    BruteForceOptimizer
    GeneticOptimizer
    BipartiteOptimizer
```

```
Optimizer Structs:
    CompatibilityMatrix
    RouteMatrix
    RouteMatrixLocal
```

```
Testing:
    TesterUtil
    TestCases
```

```
Utility:
    SolutionUtils
    Utils
    TesterUtils
```

```
Route
    startLoc()
    endLoc()
    getLoc()
    numLocs()
    get_type() = 0
    UserPreference userPref
    RoutePreference routePref
```

```
DB Structs:
    TimeWindow
    UserInfo
    Solution
    Match
    Address
    Location
    PointInfo
    Preference
    Rideshare
```

```
RouteSubmission
    setComment()
    setUserInfo()
    get_type()
    uint32_t routeId
    uint32_t userId
    double optimalDistance
    UserInfo userInfo
```

```
RiderRouteSubmission
```

```
DriverRouteSubmission
    size_t capacity
```

```
Class List
    Address
    BipartiteOptimizer
    BruteForceOptimizer
    CompatibilityMatrix
    DBAccess
    DriverRouteSubmission
    GeneticOptimizer
    Location
    Match
    Optimizer
    PointInfo
    Preference
    RiderRouteSubmission
    Rideshare
    RouteClient
    Route
    RouteInfo
    RouteMatrix
    RouteMatrixLocal
    RouteOptimizer
    RoutePreference
    RouteSubmission
    Solution
    SolutionUtils
    TestCases
    TesterUtil
    TimeWindow
    UserInfo
    UserPreference
    Utils
```