Rideshare Archival Record

Yoni Ben-Meshulam
Garrett Cooper
Derrick Huhn
Patrick Lowry

Outline

- The Carpooling problem
- How Rideshare solves it
  - Design choices
- What is involved?
  - Optimizer
  - User Interface
  - Controller
- How successful is it at solving the problem?
  - Testing
  - Evaluation
- Project management - Trac

Problem Statement

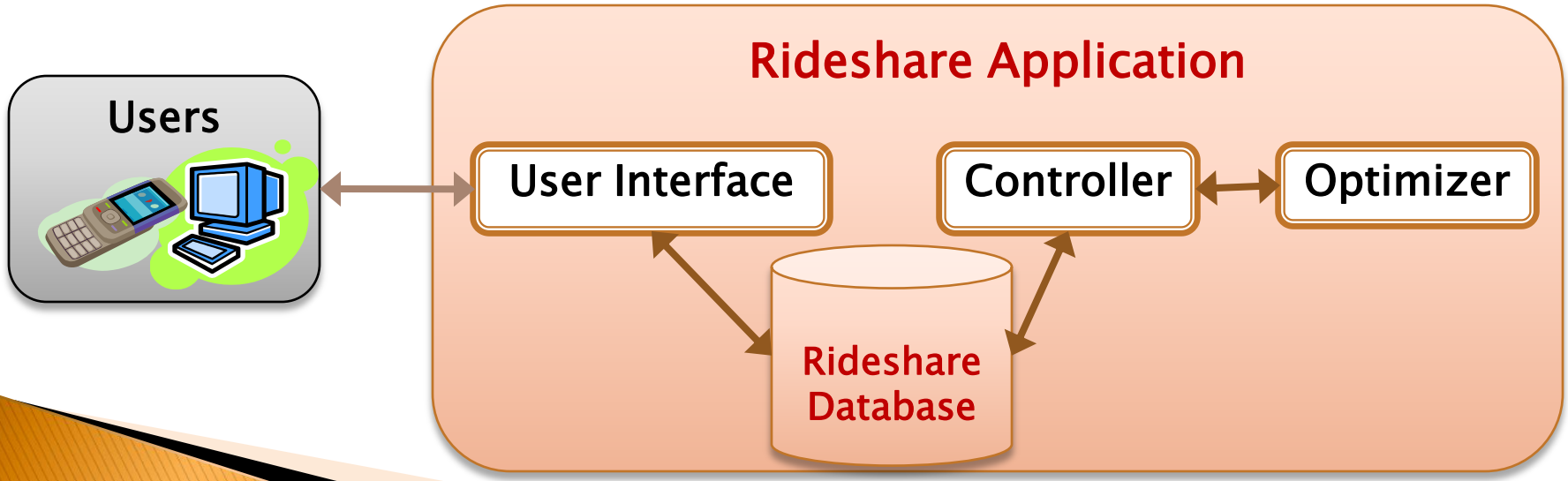
- Carpooling Problem = Vehicle Routing Problem
- But Really:
  - Capacitated Multi-Depot Vehicle Routing Problem with Time-Windows
  - Other Preferences (age, gender, rating)
  - NP-Hard
- Realistic solution times
- Globally optimized
- Reduce consumption, ecological footprint, cost, and energy dependency.

Choices

- User Interface
  - Maps Integration
    - Google Maps, MapQuest, Yahoo! Maps
- Optimization
  - Controller Library
    - Maps Integration
    - Database Interaction
    - Build System
  - Optimizers
    - Brute Force, Genetic, Bipartite Graph Matching

Architecture

- Separated user interface from optimizers
  - Communicate information through database
  - Allows data to be gathered while optimizers are running



Optimizer Challenges

- Framework to build on
  - Usable by all optimizers
  - Assessing Compatibility between a Rideshare and an unmatched rider
    - Simple preference checks
    - Scheduling
    - Deviation
  - Optimizing the order of points in a route
  - Common Scoring Function

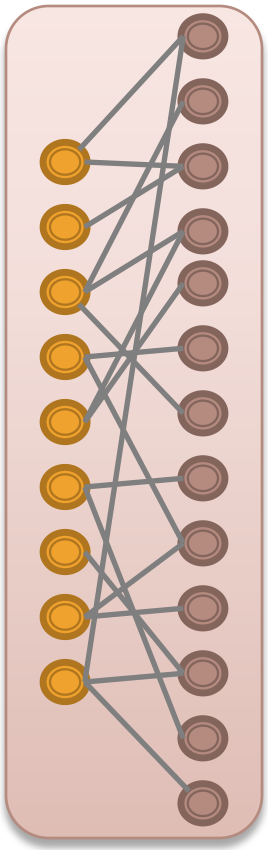
Scoring Function

$$Score = \text{Users Matched} + \frac{1}{\text{Distance}}$$

- Allows us to compare different solutions
- Users Matched (Whole Number)
  - The number of matched users (drivers and riders)
  - Bias toward spreading riders
- Distance (Fraction)
  - Total distance all drivers will take (whether they are matched or not)
  - Minimizes route deviation

Bipartite Optimizer

1. Get seed solution



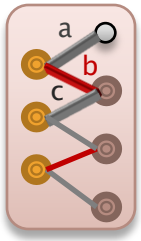
2. Connect compatible riders and rideshares with unmatched edges.

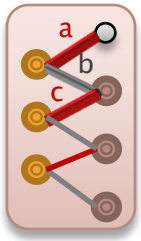
Rideshares
Unmatched riders

Unmatched edges
Matched edges

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

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.

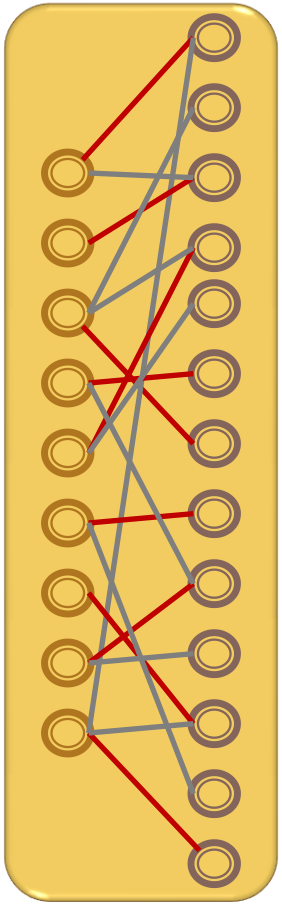
4. Exchange all unmatched and matched edges on the augmenting path.

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

This produces a maximal matching:

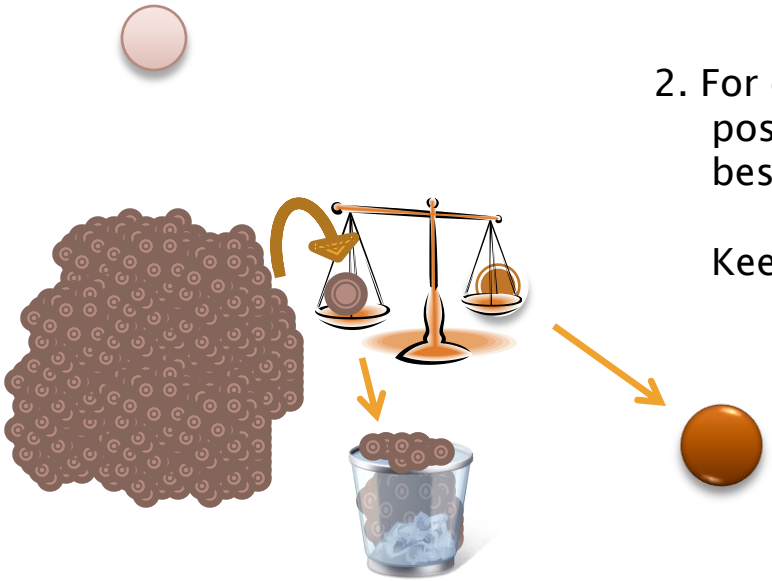
6. Add all matched riders to their rideshares.

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



Brute Force Optimizer

1. Get seed Solution



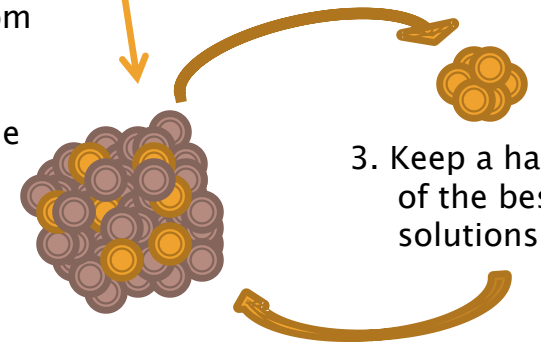
2. For each solution in the space of all possible solutions, compare it with the best one so far.

   Keep only the better of the two.

Genetic Optimizer

1. Get seed solution

2. Spawn a generation of solutions from the seed, randomly matching ride offers and requests



3. Keep a handful of the best solutions

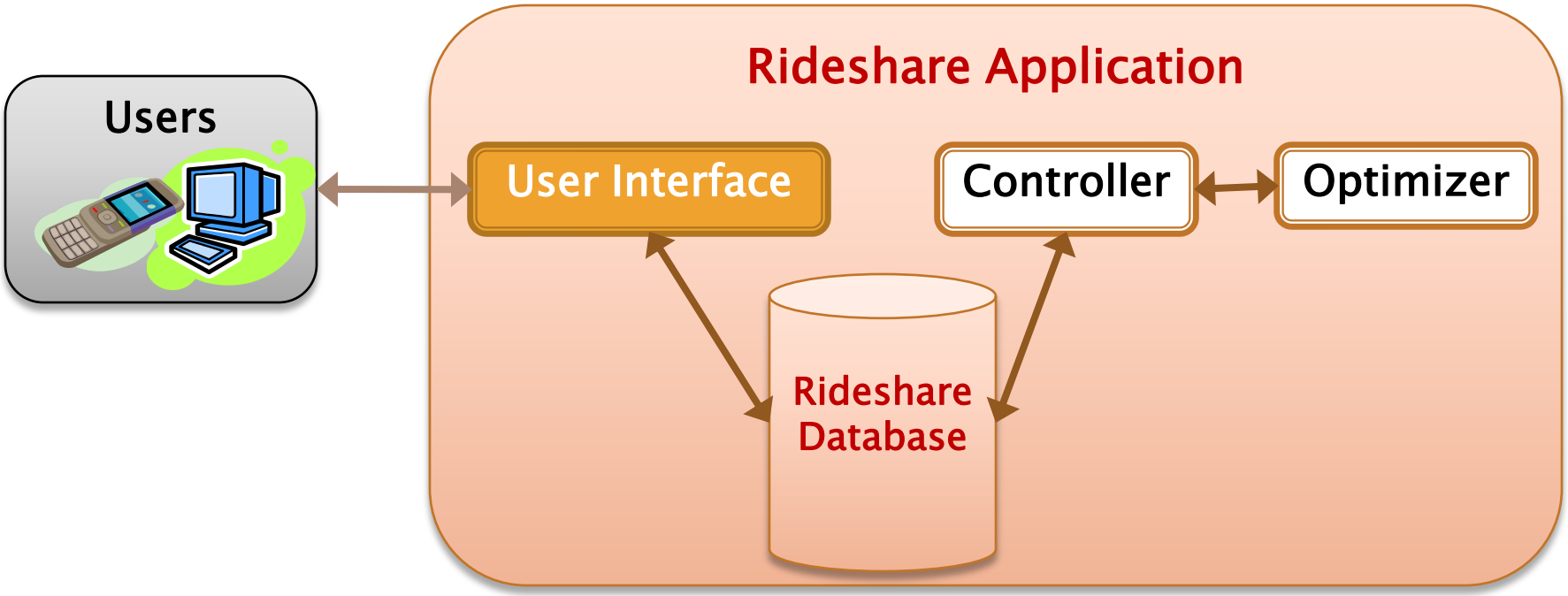
4. Keep a copy of the best solution

5. Spawn another generation from the last generation's best solutions

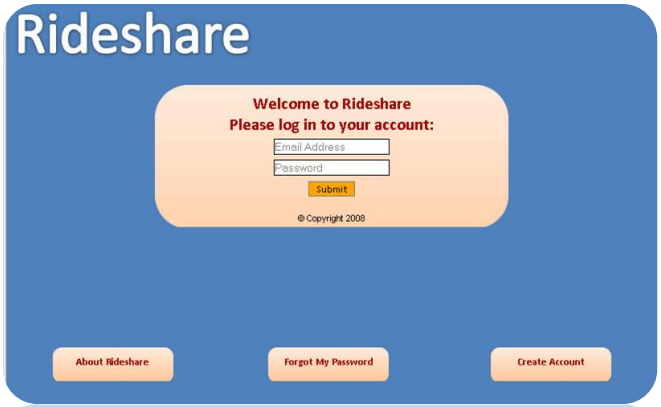
(repeat until best score settles)

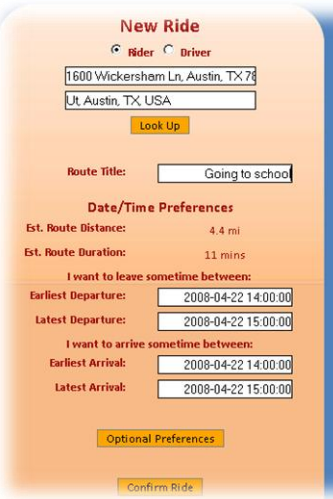
6. Once the score settles, return the best solution

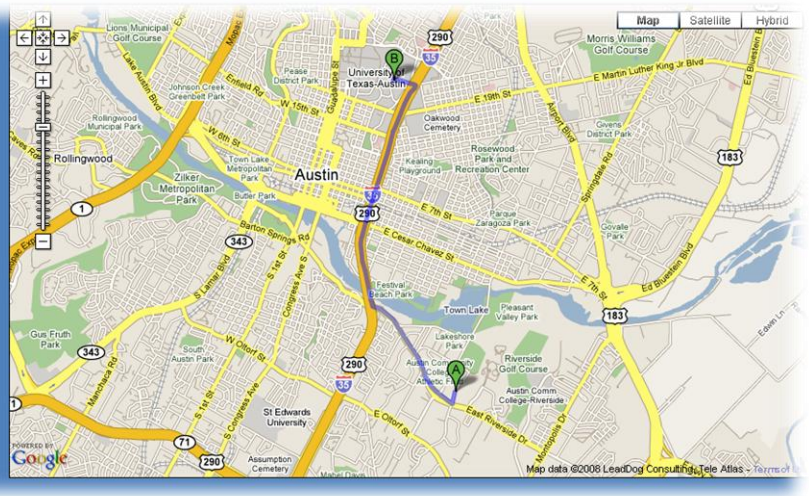
Architecture (User Interface)

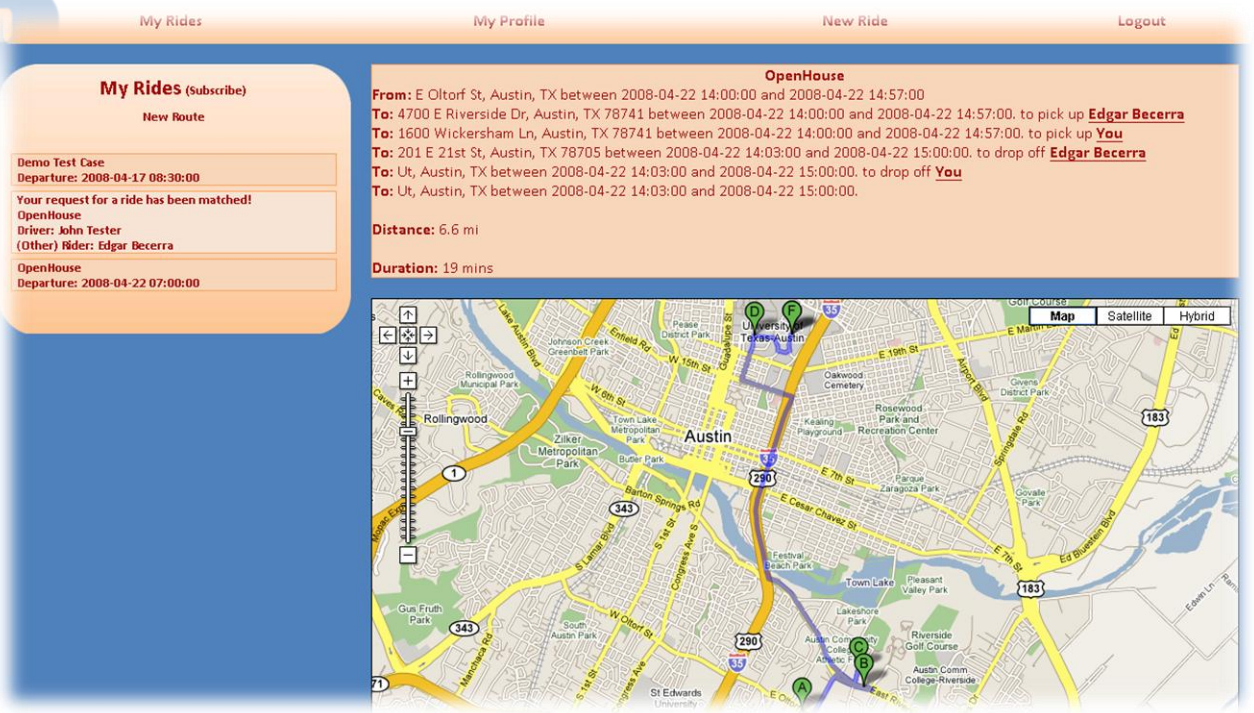


Screen shots





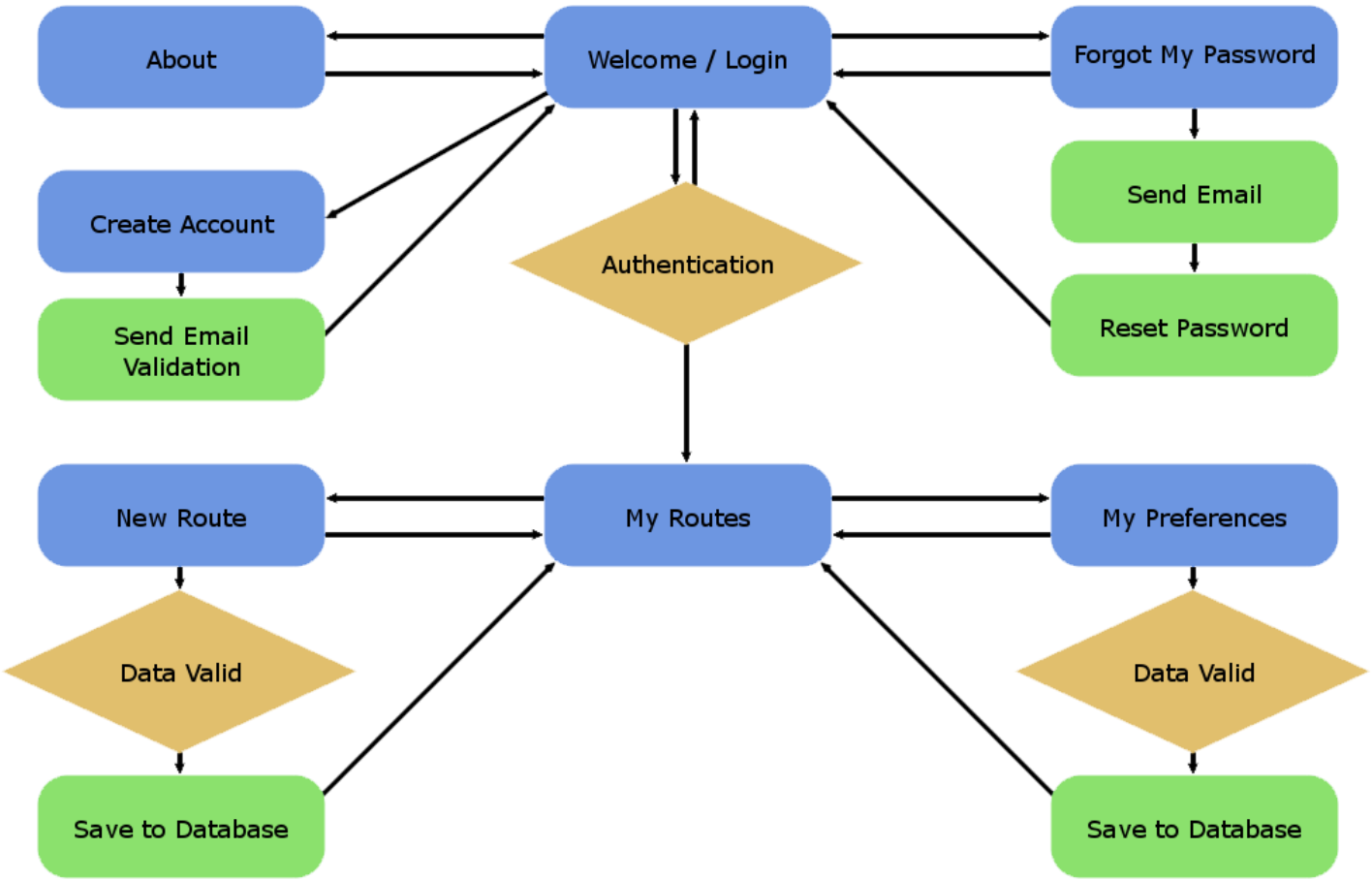




Front-End Challenges

- Multiple Users
- Extensible
  - Cascading StyleSheets (CSS)
- Google Maps
  - Offload complex address translation
  - AJAX Problems
- Input Data Handling
  - Analyze and Interpret Input Data
- RSS Feeds

Front-End Map



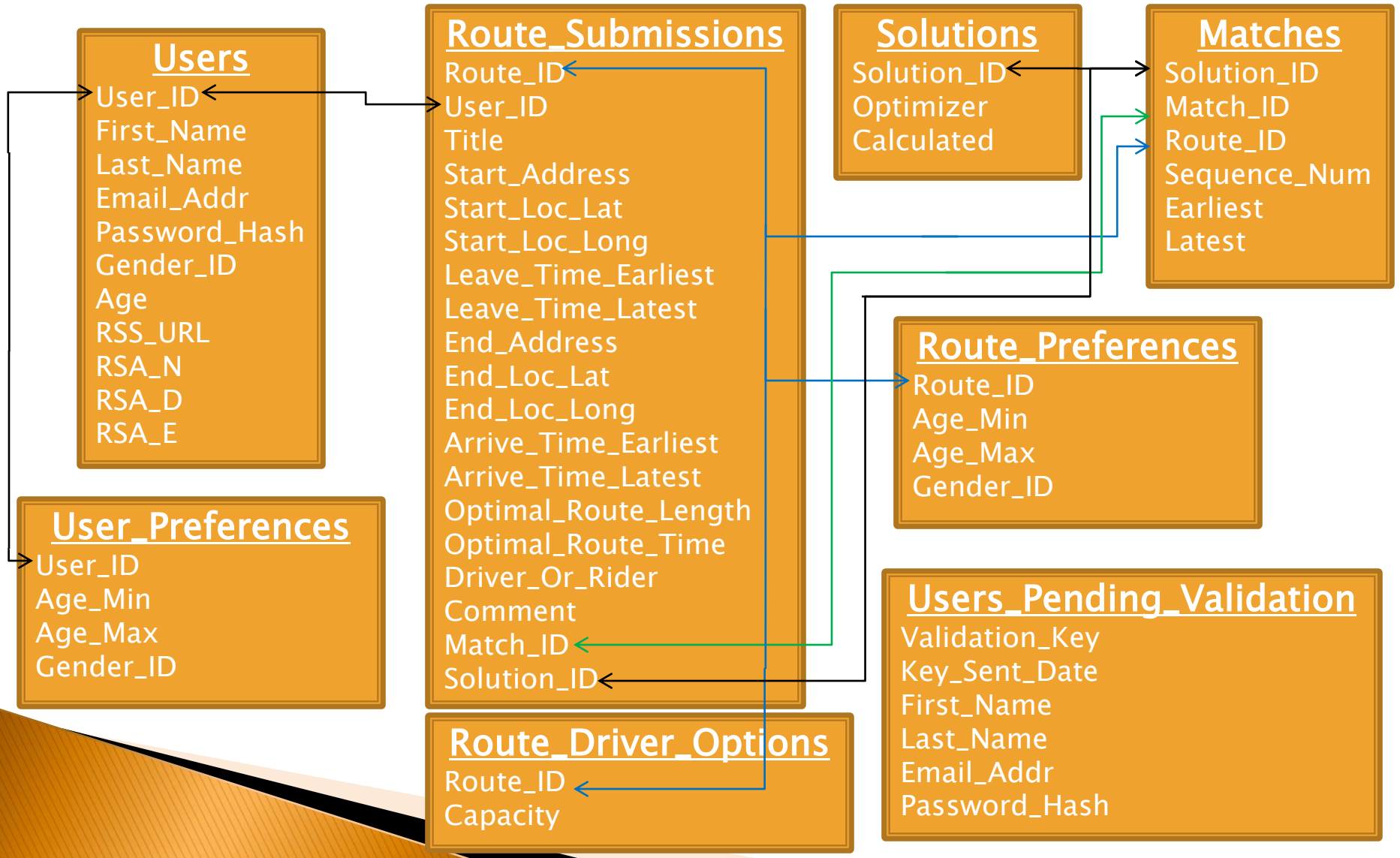
Database Challenges

- Data Formatting and Structure
  - Had to be usable for both the front and back-ends
    - DATETIME, and ENUM data types
- Data Duplication
  - How to avoid it?
    - More tables
- How to represent complex data structures?
  - Matches in solutions

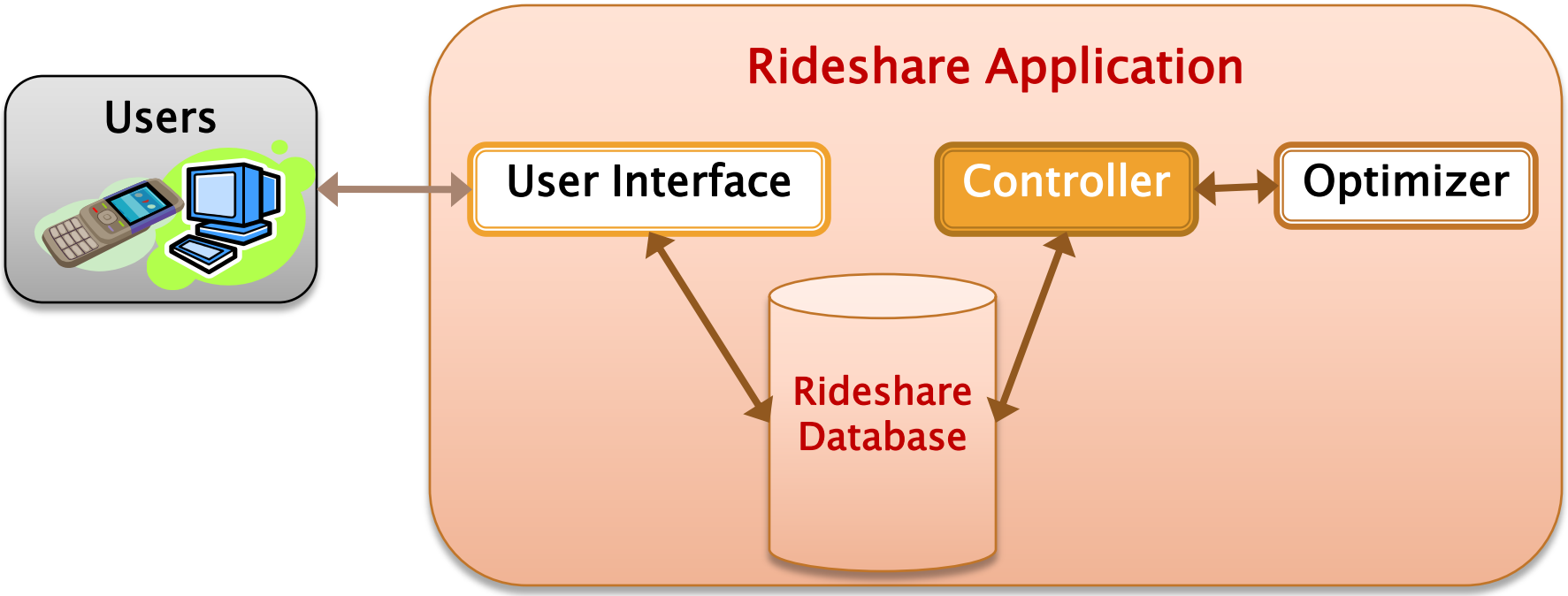
Database Implementation

- Relational Database
  - Normalization Trade-offs
    - Improves level of relation
      - Able to represent more complex data types
      - Less data duplication
    - More overhead
      - More tables
      - More IDs and joins (links) between tables

Database Structure



Architecture (Controller)



Controller Challenges

- Database communication
  - MySQL++ Library
- Limitations of Libraries (MapQuest API)
  - RouteMatrix only supported 25 Locations
    - Solution: RouteMatrixLocal – Local route calculations
- Data structures
  - STL: Vectors -> Maps
  - Represent Driver and Rider Submissions, Matches, Solutions, etc.
- Remote Development

Controller Implementation

- Executable for invoking libController's features
- Components
  - libController
    - Optimizer Logic
    - Database Access
    - Data structures for optimizer
  - Algorithm Visualizer
  - Notifications

libController

- 8,000+ Lines of Code
- 45 C++ Files
  - Optimizer Logic – 13 Files
  - Database Access – 4 Files
  - Data structures – 17 Files
  - Utility – 11 Files
- Build System
- MySQL++ and MapQuest Advantage API

Class List

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

Clients:
- RouteClient
- DBAccess

Routes:
- Route
- RouteSubmission
- RiderRouteSubmission
- DriverRouteSubmission

RiderRouteSubmission

DriverRouteSubmission
- size_t capacity

Preferences:
- UserPreference
- RoutePreference

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

Algorithm Visualizer

- Visual display of intermediate solutions generated by optimizers
- XHTML/JavaScript
- Animation utilizes MapQuest's image API
- Metaprogramming
  - C++ generates JavaScript that is executed in browser

Ride Share - Mozilla Firefox

file:///I:/home/dhuhn/release/visualizer/visualizer.html

Ride Share

Solution Information

Solution: 4

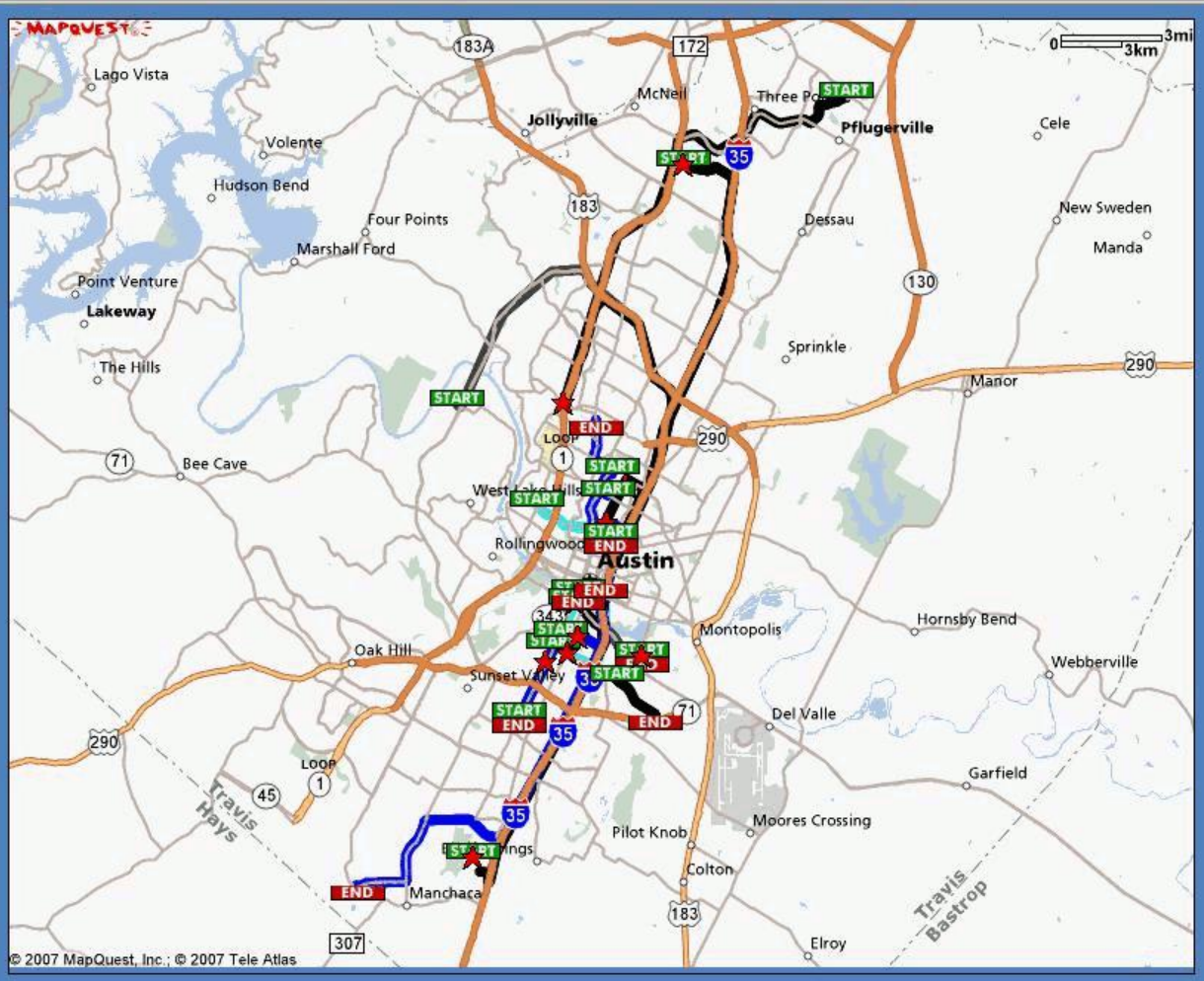
Score: 21.0098

Matches: 18

Riders: 12

Match 1 with Driver: **John Tester**
Rider #1 **Rainbow Rider**
Rider #2 **Dave Driver**
Match 2 with Driver: **Ronald Rider**
Match 3 with Driver: **Trina Tester**
Rider #1 **Ramses Rider**
Match 4 with Driver: **Ronald Rider**
Rider #1 **Rainbow Rider**
Match 5 with Driver: **John Tester**
Rider #1 **Ronald Rider**
Match 6 with Driver: **David Tester**
Match 7 with Driver: **Dave Driver**
Rider #1 **Trina Tester**
Match 8 with Driver: **Delilah Driver**
Match 9 with Driver: **John Tester**
Rider #1 **Delilah Driver**
Rider #2 **Dave Driver**
Match 10 with Driver: **John Tester**
Match 11 with Driver: **Garrett Cooper**
Match 12 with Driver: **John Tester**
Rider #1 **Dave Driver**
Rider #2 **Derrick Huhn**
Match 13 with Driver: **John Tester**
Match 14 with Driver: **Devin Vitone**
Rider #1 **Constantine Caramanis**
Match 15 with Driver: **Garrett Cooper**
Match 16 with Driver: **Garrett Cooper**
Match 17 with Driver: **Garrett Cooper**
Rider #1 **Patrick Lowry**
Match 18 with Driver: **David Tester**

Start! Stop! < >



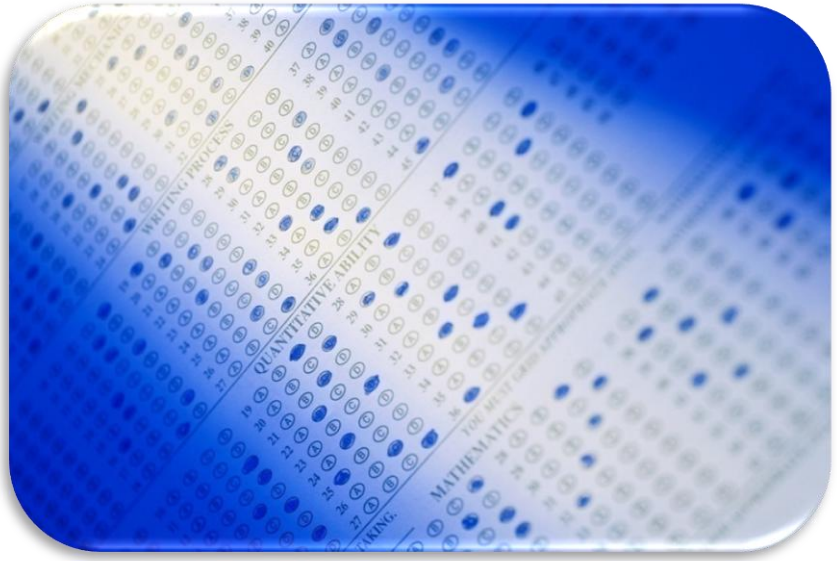
Notifications (E-mail and RSS Feed)

- E-mail
  - Users need to be notified when they have been matched in a Rideshare
  - C++ Generates Perl which invokes SendMail
  - Use libController to retrieve users' emails

- RSS Feed
  - Users can subscribe to an RSS feed that is dynamically generated using PHP
  - Real-time notification for active users

Testing and Optimizer Evaluation

- Unit Tests
- uShip Data
- Corner Cases
- Optimizer Characterization and Evaluation



Unit Tests

- Over 100 individual unit tests
  - Each feature/function tested individually
  - Hand-tailored test cases
- Used for stabilization before each commit

uShip Data

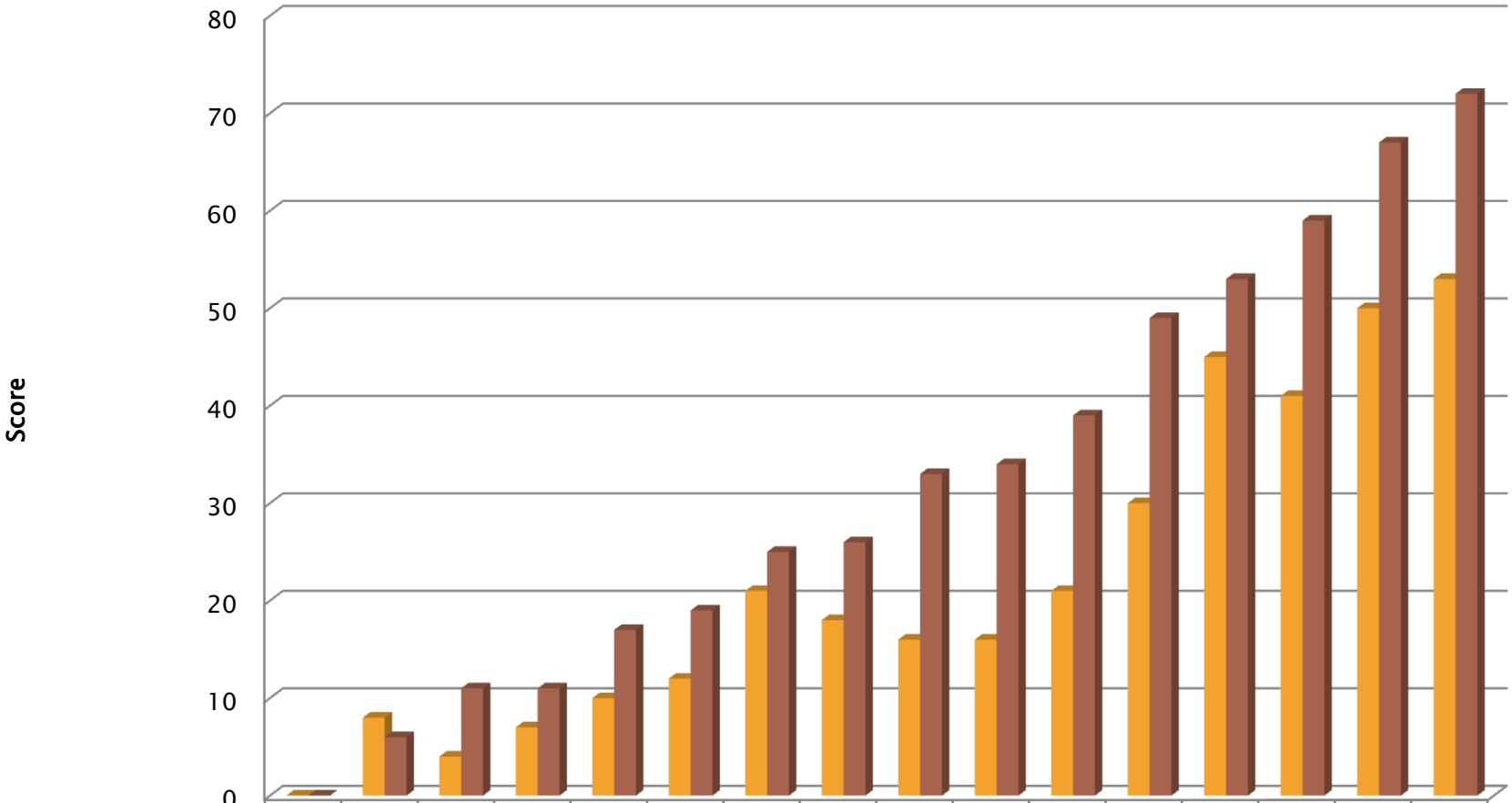
- Raw uShip Data
- 2100 Working Locations in Texas
- uShip Data with Preferences
  - 20 Non-inclusive test cases
  - Size ranges from 7 to 120, with regular increments

data and screenshot thanks to uShip.com

Corner Cases

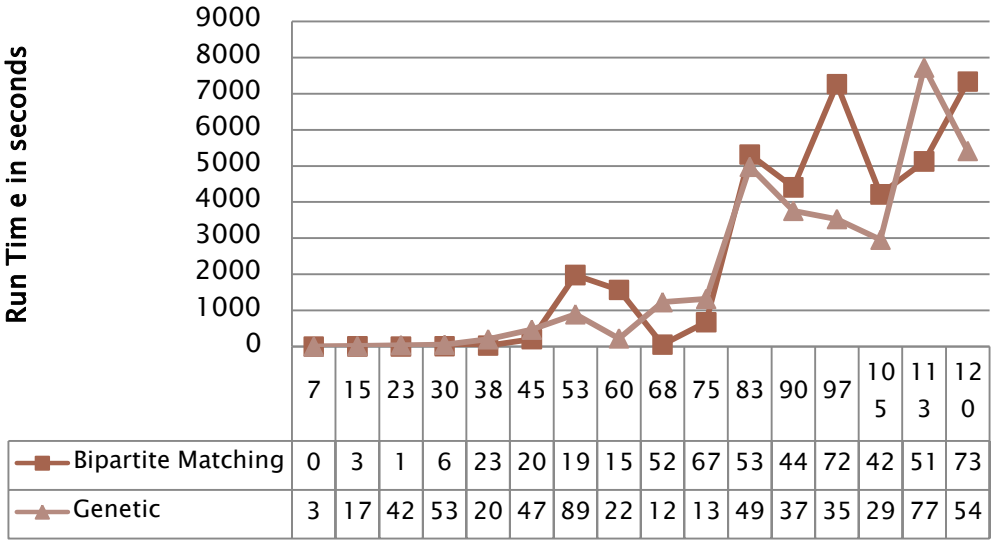
- Designed to test specific functions of the optimizers
  - Capacity
  - Going Out of the Way
    - Moving backwards to pick up a rider
    - Picking up the closer rider
  - Time Windows
  - Maximum Route Deviation
  - Maximizing Riders
  - Single-solution test case for Bipartite

Scores
Genetic vs. Bipartite



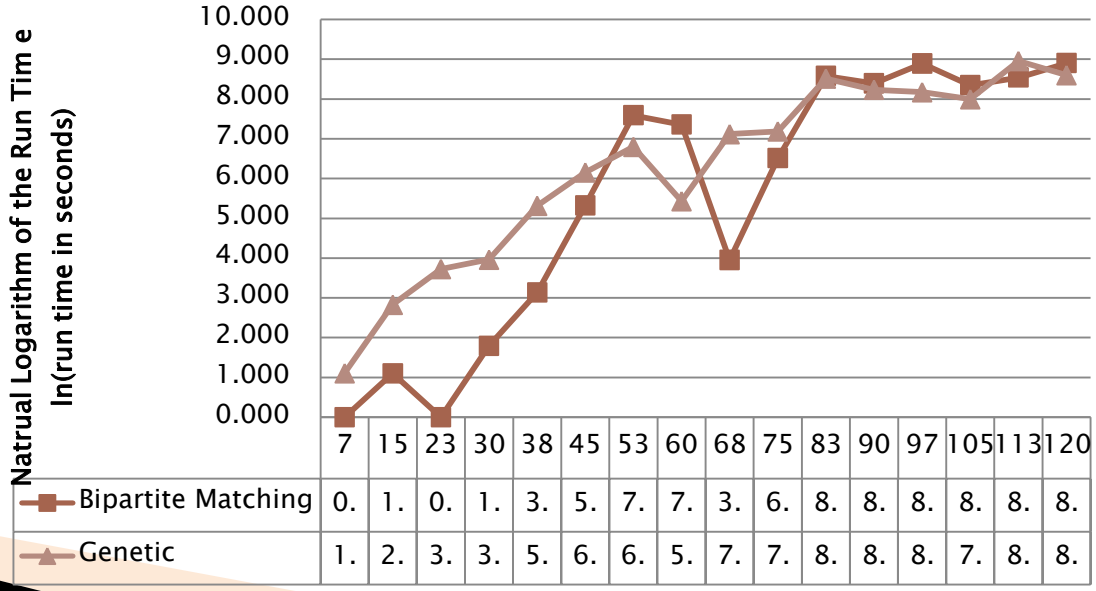
	uPrefs_1	uPrefs_2	uPrefs_3	uPrefs_4	uPrefs_5	uPrefs_6	uPrefs_7	uPrefs_8	uPrefs_9	uPrefs_10	uPrefs_11	uPrefs_12	uPrefs_13	uPrefs_14	uPrefs_15	uPrefs_16
Bipartite Matching	0.00	8.00	4	7	10	12	21	18	16	16	21	30	45	41	50	53
Genetic	0.00	6.00	11	11	17	19	25	26	33	34	39	49	53	59	67	72

Number of Requests vs. Run Time



	7	15	23	30	38	45	53	60	68	75	83	90	97	105	113	120
Bipartite Matching	0	3	1	6	23	20	19	15	52	67	53	44	72	42	51	73
Genetic	3	17	42	53	20	47	89	22	12	13	49	37	35	29	77	54

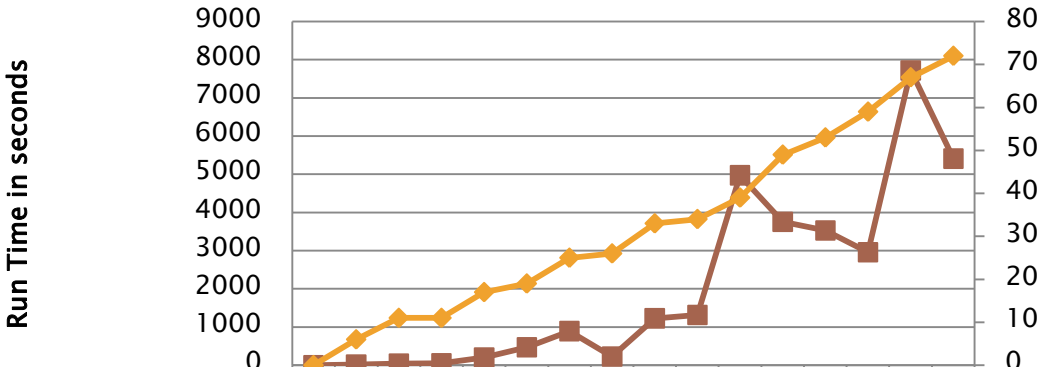
Number of Requests vs. Natural Logarithm of Run Time



	7	15	23	30	38	45	53	60	68	75	83	90	97	105	113	120
Bipartite Matching	0.	1.	0.	1.	3.	5.	7.	7.	3.	6.	8.	8.	8.	8.	8.	8.
Genetic	1.	2.	3.	3.	5.	6.	6.	5.	7.	7.	8.	8.	8.	7.	8.	8.

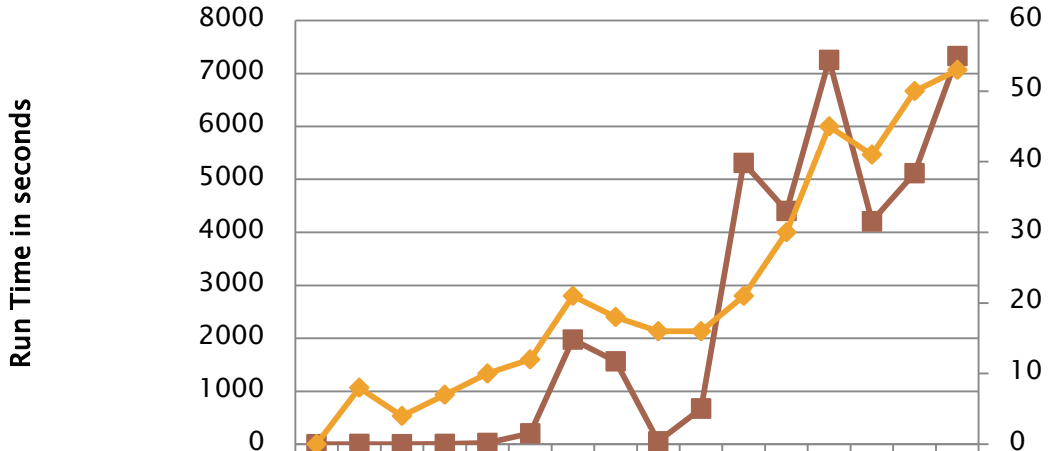
Run Time vs. Final Score

Genetic



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Time in seconds	3	17	42	53	20	47	89	22	12	13	49	37	35	29	77	54
Final Score	0	6	11	11	17	19	25	26	33	34	39	49	53	59	67	72

Bipartite



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Time in seconds	0	3	1	6	23	20	19	15	52	67	53	44	72	42	51	73
Final score	0	8	4	7	10	12	21	18	16	16	21	30	45	41	50	53

Time and Cost Considerations

- Cost
  - Open Source == Free
  - ECE Department provided Development Systems
- Time
  - Coupled tasks that hindered development
  - 15% debugging C++ related problems
  - 5% dedicated to code not used
  - 20% dedicated to user interface
- Improvements
  - 20% of tickets involved fixing bugs
  - More code review sessions!

Safety and Ethical Aspects

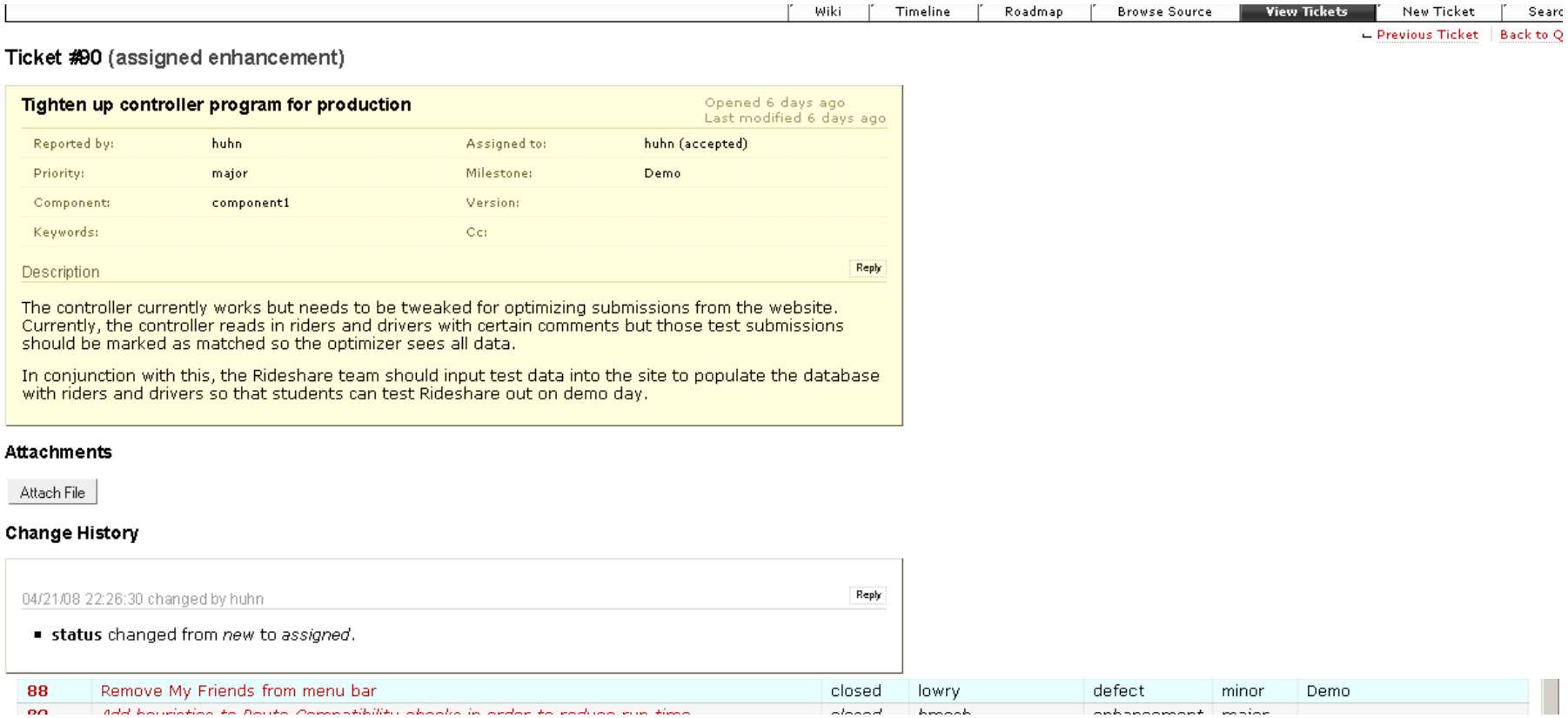
- Personal safety:
  - Encourage users to take advantage of site features to screen matches
  - Minimum age eighteen

- Social Innovation
  - Reduce ecological footprint
  - Save time and energy
  - Promote cooperative, sustainable living

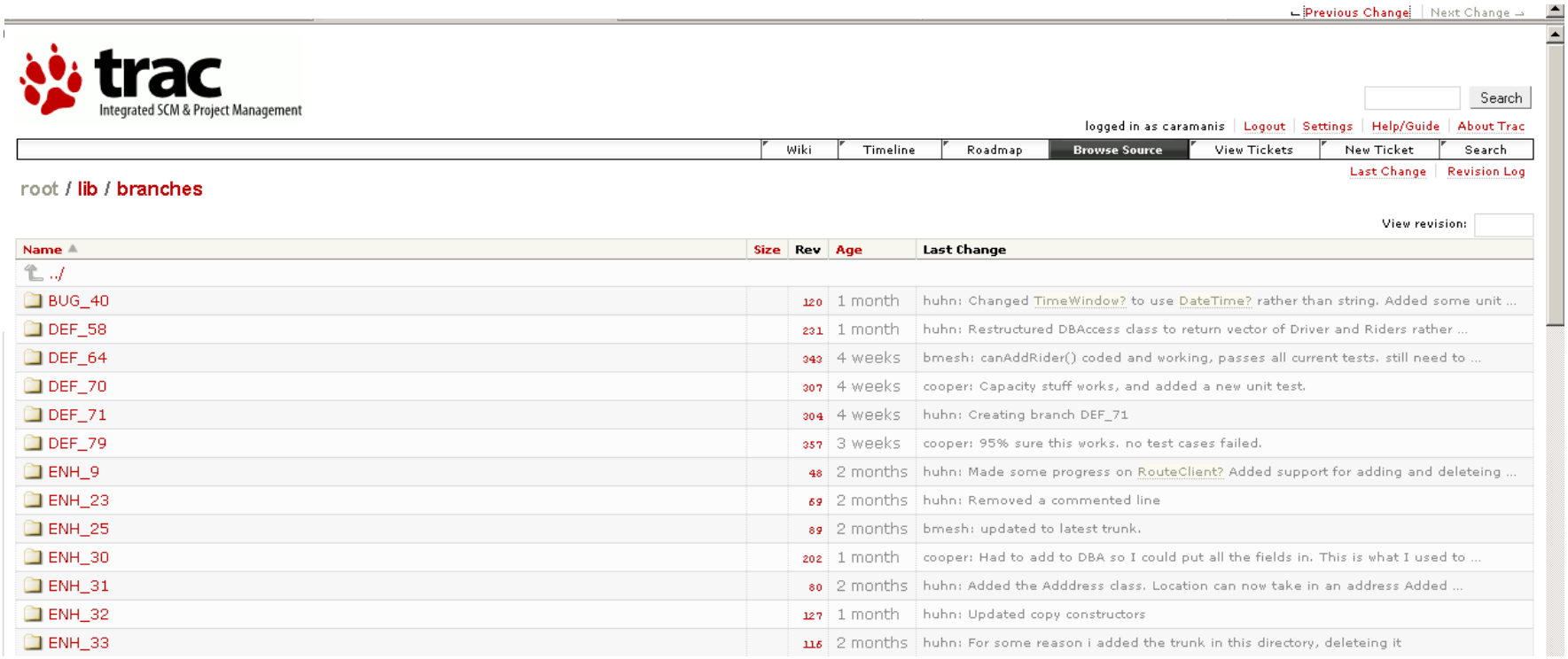
Trac

- Wiki System
- Ticket System
- Subversion Integration
- Timeline/Roadmap

Ticket System



Subversion Integration



Conclusions and Recommendations

- Results
  - Completed all of the tasks for optimization
  - Didn't implement problem reduction (not a necessity)
  - Didn't implement confirmation (time)
  - Successful testing of all functionalities

- Recommendations
  - Develop user interface for social networking, phones
  - Extensible Design