A new location-aware calendar-based application for dynamic minimum path trip planning

Georgios-Andreas Liagouras[a], Amir A. Sayegh[b] and Polychronis Koutsakis[a]
[a] *Department of Electronic and Computer Engineering, Technical University of Crete, Greece*
[b]*TELUS Canada*
Email: liagouras.georgios@gmail.com, amirsayegh@gmail.com, polk@telecom.tuc.gr

***Abstract*— The convergence of several trends, including the proliferation of mobile, cloud technologies, social media, and socio-economic trends such as bring your own device (BYOD), have led to not only the democratization of computing but also to information overload. This creates an opportunity for pattern recognition and 'Big Data' technologies to support mobile 'context-aware' computing where technology understands human intentions, and effectively 'disappears'. In this short paper we explore, through the development of an Android-based test application, one of the capabilities of this computing paradigm, which to the best of our knowledge has not been explored. Namely, we explore the complexity of dynamic calendar based minimum path computing.**

Keywords: context-aware computing; dynamic calendar; minimum path; time-saving application.

I. INTRODUCTION AND MOTIVATION

Since the beginning of the new millennium, cell phones have emerged as an important part of our daily life, changing the way we live and work. These significant technological advances have led to the inclusion of new and elaborate features in cell phones, turning them into a mini computer, a mailing system, a text messenger, a video camera, even a game console.

A recent major breakthrough is that the development of mobile applications, which was once a prerogative of cell phone manufacturers, has been made available to developers for further improvements and innovation. This move marked the beginning of a new era for cell phones, the era of smartphones. Mobile applications with access to smartphones' hardware such as 3G/Wi-Fi, GPS, camera, offer services that have a great impact on the way we perform our daily tasks. The idea that devices can both sense and react to their environment according to specific rules, in order to increase this impact, is defined in computer science as *context awareness*. Given that almost all modern smartphones have GPS, accelerometers, gyroscopes and altimeters, many new possibilities arise, such as location-aware advertisements, information and services (transportation, entertainment, etc.). Recent examples of context-aware applications include ‘Apple Siri’,‘Google Now’, ‘Blackberry Hub’, ‘OnTime’, ‘Sherpa (Osito)’, ‘Tempo’, etc.

For the interested reader, the survey presented in [9] is one of the most cited surveys of context aware computing systems. The taxonomy considers several aspects of the mobile applications space, including context awareness, highlighting functional, architectural, technological, and implementation issues. In [10], the authors survey existing research and approaches towards realization of opportunistic user context recognition with mobile phones. They introduce the typical architecture of a mobile-centric user context recognition system as a sequential process of sensing, preprocessing, and context recognition. In [11], the authors present an overview of the current state-of-the-art work in predictive mobile computing. They present a survey of phenomena that mobile phones can infer and predict, along with a description of machine learning techniques used for such predictions. In [12], the authors present a location-aware call handling assistant as a dynamic solution. The assistant runs on mobile devices and enables users to manage calls based on their current context (in particular, their location and activity, the date and time, and the caller and caller’s group). This system exploits Bluetooth technology for location determination and for user modelling. In [13], the authors present their work on the Zonezz platform. The platform identifies meaningful locations such as 'home' or 'work'. It provides an easy to understand context modeling and fully runs on a mobile device without the need for a central service. Other applications can use this platform to create context-awareness. They conclude the paper by showing the benefits of this platform with the help of a context-aware calendar tool.

Motivated by this general concept, we built the first part of our proposed Business-Life Synchronization (BLS) application, the goal of which is to make users’ life easier by helping them take care of everyday tasks in the minimum possible time. To achieve this, we proposed the concept of a location- aware calendar, that the user would utilize as his daily agenda. This calendar will dynamically indicate to the user the optimal route to visit all of his destinations in the minimum time, thus saving time and money. We also added extra functionalities by changing the route according to users' needs, showing the address of every location on the map and associating specific locations with messages which will appear when the users arrive at those locations. BLS extracts from the user's default calendar the locations that the user needs to visit within the day. The crux of our contribution is that the route gets updated as the calendar entries evolve in real-time. Practically, our biggest challenge is that although smartphone compute power has evolved significantly, we must balance run-time delay, phone energy consumption, and network delay over the mobile network. From a mathematical modeling perspective, the "network" consisting of the locations and the paths that the user will follow forms an undirected weighted graph, in which the locations are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's length. We attempt to find –in near real-time- the optimal route from the perspective of minimum delay in order to visit all locations starting from our current one, which is the only constant in the studied problem. This problem is similar in nature with the Traveling Salesman Problem, but the major difference is that the final destination is not the initial location. To solve the problem in real-time, we use three algorithms to find the optimal route: an exhaustive algorithm that computes all the possible permutations in order to serve as upper bound on complexity, the heuristic Tabu Search algorithm and one method of Simulated Annealing-based neighborhood search. We then compare the algorithms in terms of their discovered route duration and the computational load they incur. Additionally, we implement user-centric features such as early-warning notifications, route modification, location-aware messages, and view toggling.

For the development of our application we used the Android platform. Android is a Linux-based operating system for mobile devices such as smartphones and tablet computers, developed by Google in conjunction with the Open Handset Alliance[1]. Google releases the Android code as open-source, under the Apache License. Additionally, Android has a large community of developers writing applications ("apps") that extend the functionality of devices. Developers write primarily in a customized version of Java, and apps can be downloaded from online stores such as Google Play (formerly Android Market), the app store run by Google, or third-party sites. The key issue for BLS, besides the proper programming of the application (in Android 2.3 Gingerbread [2])[1], is to use an algorithm that will incur the minimum possible computational complexity while finding the optimal route (in terms of minimum delay) for the user to follow.

In our work, we perform extensive evaluations to demonstrate the advantages and weaknesses of the studied algorithms and examine how their performance affects the application's efficiency.

This paper is organized as follows. In section II we present the basic model and some additional features supported by BLS. In section III we explain the problem of finding the optimal route and present the algorithms used to solve it. In section IV we evaluate the performance of the algorithms in terms of computational load and accuracy of results and compare them against each other.

II. PROGRAMMING CALENDAR FUNCTION IN BLS

The application consists of components that are loosely bound to each other and each one is specified by an Activity. An Activity is an application component that provides a screen with which users can interact. Each Activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows [3]. Inside the Activities we implement our algorithms and the rest of the application's features. Our application is characterized by two Activities, the Calendar Activity and the Maps Activity.

A. Calendar Activity

This Activity is used for the application's launch. It contains methods that are used to populate the initial screen with image objects, button assignments that make the transitions to the user's calendar or the Maps Activity, and algorithms that display pop-up messages based on the network status and the calendar events.

More specifically, the user is notified for the following:

[1] Please note that the challenges described in this paper still apply with recent versions of Android.

1) A pop-up message is displayed for the network status when the user touches the viewMap button and there is no active network connection (Figure 2). It should be noted that, below the selected date, which appears in English, the name of the month appears by default in the language associated with the country where the user is located (in this case, Greece). For the same reason, in Figures 3-6 the street and city names appear again in Greek, while the rest of the message appears in English (the whole message can appear in Greek as well, but this choice was obviously not made here in order to make the paper easy to understand).

2)A pop-up message is displayed with the number of events scheduled on a user selected date (Figure 1).

3)A pop-up message is displayed that reminds the user to select a date before using any of the BLS features such as inserting an event, modifying an event, viewing events or calculating the optimal route.

4) A pop-up message is displayed that asks for user verification in case of application exit.

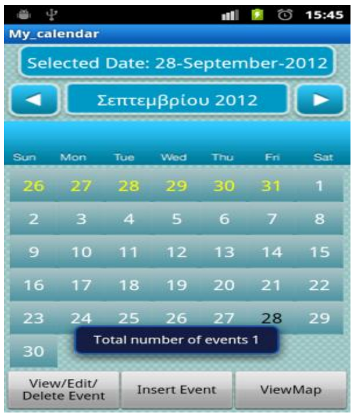


Figure 1. Total number of events notification

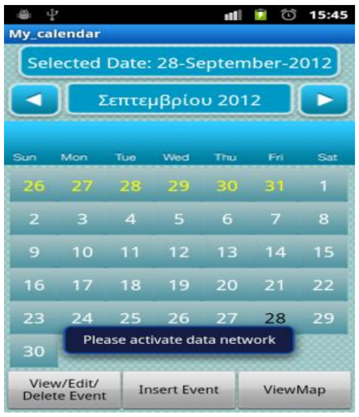
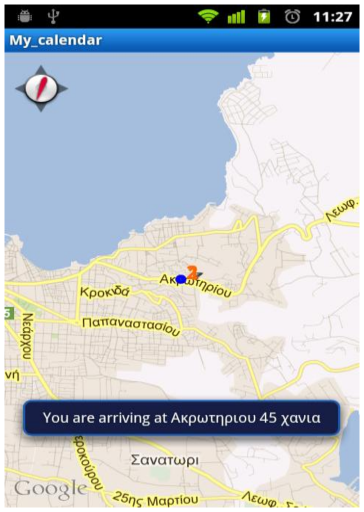
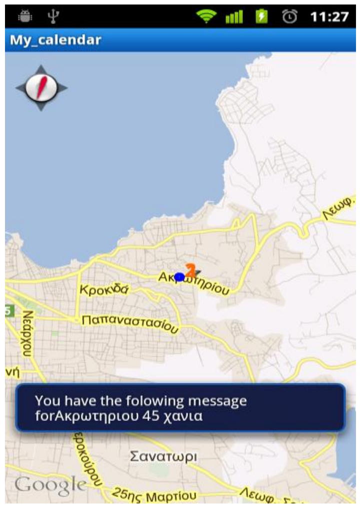

15:45
My_calendar
Selected Date: 28-September-2012
Σεπτεμβρίου 2012
Sun Mon Tue Wed Thu Fri Sat
Please activate data network
View/Edit/ Delete Event
Insert Event
ViewMap

Figure 2. Network notification



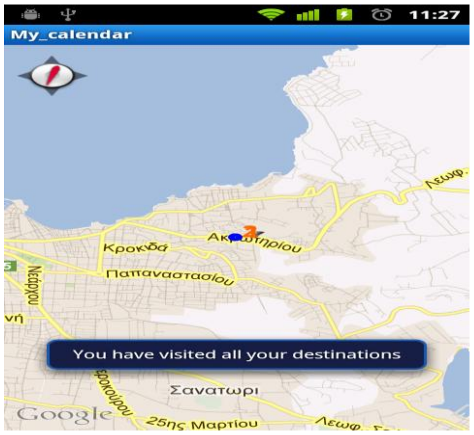
You are arriving at Ακρωτηριου 45 χανια

Figure 3. Arrival notification



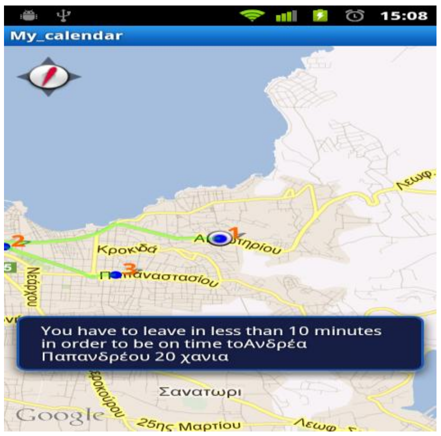
You have the folowing message forΑκρωτηριου 45 χανια

Figure 4. Message notification



You have visited all your destinations

Figure 5. Number of remaining events notification



You have to leave in less than 10 minutes in order to be on time toΑνδρέα Παπανδρέου 20 χανια

Figure 6. Urgency notification

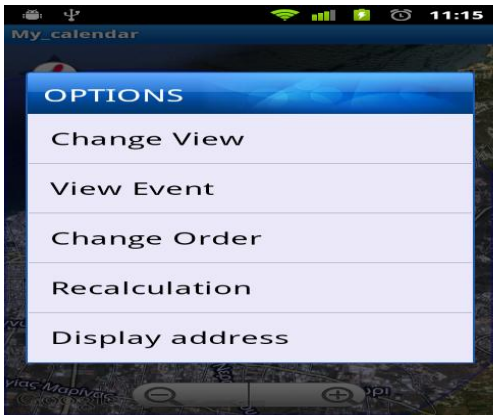
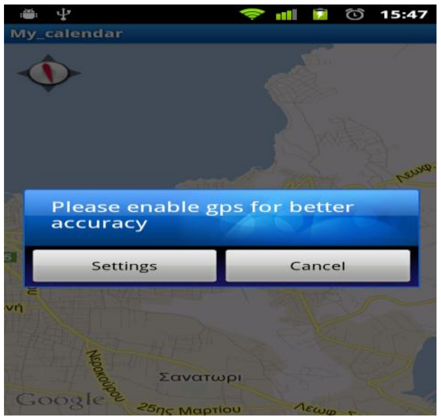


Figure 7. Menu



Please enable gps for better accuracy

Figure 8. GPS notification

B. Maps Activity

This Activity is launched when the user clicks the viewMap Button. It displays the map that contains the optimal route. Moreover in this activity there are algorithms that:

1) Calculate the optimal route.
2) Track the user's current position every 15 seconds.
3) Offer the user the option to change the order of events (Figure 7).
4) Toggle between satellite and street view (Figure 7).
5) Recalculate the optimal route if an error occurs or if the user wishes so (Figure 7).
6) Offer the user the option to view messages that correspond to specific locations on the map (Figure 7).
7) Display the address of selected locations (Figure 7).
8) Check the GPS status and alert the user if it is turned off (Figure 8).
9) Based on the user's scheduled appointments, his current location and the other locations' distance from it, BLS notifies the user 10, 5 and 0 minutes before his needed time of departure, respectively, in order to ensure that the user will not be late (Figure 6).
10) Inform the user about his arrival at a destination point, when he is at a radius of 120 metres around this point (Figure 3).
11) Inform the user about his route's end when he has visited all the locations (Figure 5).
12) For every location visited, display the message that corresponds to that event (Figure 4).

III. EXHAUSTIVE AND HEURISTIC ALGORITHMS

A. Problem modeling

The "network" consisting of the locations and the paths that the user will follow forms an undirected weighted graph. The following are the key elements of our model:

a. Locations. The locations are the graph's vertices and can be extracted from the events of the default calendar application.

b. Paths. These are the graph's edges and on the map are represented by the route that connects the locations.

c. Distance. This is the path’s distance, between two locations.

d. Current location. The current location is the starting point of our algorithms and is essential for their functionality.

Therefore, we need to solve the problem of finding the shortest route between locations i and j, with specific stops in-between the start and end locations. This is a minimization problem of starting at a specified vertex and ending after having visited each vertex exactly once. It is an NP-hard problem, very similar to the travelling salesman problem (TSP) [4], as also mentioned in the Introduction, with the major difference that we do not return to the starting location when we have visited all our destinations. However, in our case there are additional limitations, because of the type of the application. Besides finding the optimal route, it is also of major importance to keep the *network delay* (requests to Google Maps servers for geographical data) and the *run-time delay* (running the iterations) within reasonable values, as a large delay would discourage mobile users from utilizing BLS and would also result in increased energy consumption at the smartphone.

B. Algorithms' Descriptions

To solve the problem in real-time, we have used and evaluated three algorithms: an exhaustive algorithm that computes almost all of the possible permutations, as well as the heuristic algorithm Tabu Search and one method of Simulated Annealing-based neighborhood search. All of our results have been derived for 95% t-confidence intervals (constructed in the usual way [8]).

B.1 Permutation Algorithm

B.1.1. Description

The permutation algorithm guarantees a global optimum solution therefore it was the obvious choice from the beginning in order to use it as a base for our comparisons (effectively acting as the computational delay bound). The algorithm reads the addresses from the calendar application and computes all the possible permutations. For each permutation we find the total route duration and then we choose the permutation with the shortest duration. The terms "addresses" and "locations" will be used interchangeably in the rest of the paper.

B.1.2. Complexity

As expected by its exhaustive nature, the permutation algorithm incurs a very high computational load. For n addresses there are (n-1)! different permutations (since the initial address is constant), which constitute the run-time delay "burden" of the system. There are also (n-1)*(n-1)! server requests (network delay) because for each permutation we need (n-1) requests to find the distances among all locations. In order to improve the algorithm's response time we make two simple improvements:

a) we keep in memory all the route times that have already been computed at the server in previous iterations so that we don't have to resend the same request repeatedly, and

b) we assume that the route time between two locations is the same regardless of the direction the user is headed. This assumption is reasonably accurate, although there are cases where the route from location A to location B is non-negligibly different from the respective route from B to A.

These improvements help to significantly decrease the complexity associated with the network delay and allow us to use a n^2 array, as the complexity becomes equal to $(n^2-n)/2$ (number of routes to send requests for, minus the elements of the diagonal which are equal to zero (distance to self), divided by two because we do not consider the user's direction as explained above).

The pseudo-code for the permutation algorithm is presented below, in Figure 9.

```
Input: Locations' names from the Android calendar app
Output: Optimal Route

For i=0 to n do
Retract location[i]
End for
For  i=0 to n
  For j=0 to n
          If  Distance_Matrix[i][j] exists
          time_to_destination=time_to_destination+ Distance_Matrix[i][j]
          else
          time_to_destination=time_to_destination+ Distance[i][j]
          Distance_Matrix[i][j]=Distance[i][j]
           End if
  End for
End for
 Find Minimum(time_to_destination)
Draw Route(Minimum(time_to_destination))
```

Figure 9. Pseudo-Code for the Permutation Algorithm

B.2 Tabu Search Algorithm

B.2.1. Description

Tabu Search [5] is an improvement over basic local search that attempts to overcome local search problems by not being stuck in a local minimum. This is accomplished by allowing the acceptance of non-improving moves, in order to not be stuck in a locally optimum solution, and in the hope of finding the global best. Tabu Search also allows us to escape from sub-optimal solutions by the use of a tabu list. A tabu list is a list of possible moves that could be performed on a solution. These moves could be swap operations (as in TSP) or subtractions and additions in case of dealing with numerical optimization problems. If a move is accepted (new best solution is found), its move is made tabu for a certain number of iterations, i.e. we cannot perform the same move for a certain number of iterations. When a move is made tabu, it is added to the tabu list with a certain value called the Tabu Tenure (tabu length). With each iteration, the tabu tenure is decremented. Only when the tabu tenure of a certain move is 0, can the move be performed and accepted.

To allow a tabu move, we need to apply aspiration criteria [7]. Aspiration criteria allow a tabu move to be selected based on certain constraints. Tabu Search performs the following steps:

```
1. Create an initial solution (in our case the initial solution is the order in which the user inserted the events), and call it the current solution.
2. Find the best neighbor of the current solution by applying certain moves.
3.   If the best neighbor is reached by performing a non-tabu move,
          accept this as the new current solution,
      else
          find another neighbor (best non-tabu neighbor).
4. If the maximum number of iterations is reached (or any other stopping condition)
          go to step 5, else go to step 2.
5. The globally best solution is the best solution found throughout the iterations.
```

B.2.2. Complexity

The complexity of the Tabu Search algorithm is significantly smaller than that of the permutation algorithm. The reason is that it searches for local optimums. The impact of the network delay on this algorithm is the same (complexity equal to $(n^2-n)/2$) but there is a big improvement in terms of the run-time delay, for which the computational complexity is equal to $k*n^2$,

where k is a variable defined by the BLS programmer (it has been chosen to be equal to 30 in our system because in almost all cases studied the use of larger values did not render any significant improvements), $k*n$ is the number of iterations and n is the number of accesses in the tabulist and the distance matrix in each iteration.

The pseudo-code for the Tabu Search algorithm is presented below, in Figure 10.

```
Input: Locations' names from the Android calendar app
Output: Optimal Route

For i=0 to n do
Retract location[i]
End for

For  i=0 to n
   For j=i to n

 Distance_Matrix[i][j] =Distance[i][j]
Distance_Matrix[j][i] =Distance[j][i]

   End for
End for

For i=0 to n iterations
currSolution =getBestNeighbor(tabuList, Distance_Matrix, currSolution)

currCost=getCost(currSolution)

            if (currCost<bestCost)
            bestCost=CurrCost
            bestSol=CurrSolution
            End if
End For
Draw_Route(bestSol)
```

Figure 10. Pseudo-Code for the Tabu Search Algorithm

B.3 Simulated Annealing (SA) Algorithm

B.3.1 Description

This algorithm is a Simulated Annealing-based neighborhood search. We implement SA using pairwise interchange [6], and we make certain important additions to the general logic of the algorithm, as explained below.

Following the same approach as before, we create a distance matrix where we store the time distances between locations. Next we create an initial solution based on the array's data by sorting the distances from the initial location in ascending order. Then we apply the pairwise interchange method based on the initial solution in two ways. First we apply the pairwise interchange on the initial solution, similarly to the logic of the algorithm in [6]. Then, we add an equal number of solutions to the pool of solutions found via the first approach, by randomly producing exactly one more solution for each solution of that pool. This is done by using each solution of the pool as a seed in order to perform one more pairwise interchange at random. Among all the produced solutions, the solution with the minimum delay is the one chosen as the best.

B.3.2 Complexity

The impact of the network delay remains the same (complexity equal to $(n^2-n)/2$). The run-time delay is a combination of the sorting algorithms and pairwise interchanging. We use the default sorting algorithm in Java, which is currently Timsort, a hybrid algorithm of merge sort and insertion sort. Its complexity is O(nlogn). The complexity of the two methods of pairwise interchange is n(n-1)+2 because each one generates n(n-1)/2 solutions plus the initial solution that is used twice.

The pseudo-code for our Simulated Annealing algorithm is presented below, in Figure 11.

```
Input: Locations' names from the Android calendar app
Output: Optimal Route

For i=0 to n do

Retract location[i]

End for

For  i=0 to n
   For j=i to n

Distance_Matrix[i][j] =Distance[i][j]
Distance_Matrix[j][i] =Distance[j][i]

   End for
End for

InitialSol=Sort(Distance_Matrix[0])

For i=1 to n
            For j=i to n
PairwiseInterchange(InitialSol)
tempTime=calculate_time(Distance_Matrix, InitialSol);
            End for
End for
Find_best( tempTime)
Draw_Route
```

Figure 11. Pseudo-Code for the Simulated Annealing Algorithm.

IV. EVALUATION

We used two different methods to evaluate the performance of each algorithm. Both methods receive as input n addresses ($1<n\leq 10$). While $n<10$, new addresses can be added in order to reflect dynamic changes in a user's schedule. We do not consider higher values of n, mainly because we assume that it is unlikely that a user will need to visit more than 10 locations within a day, but also because the permutation algorithm leads to unacceptably high delays when the number of locations increases, therefore it would not make sense to include it in further comparisons. We comment on this problem of the permutation algorithm below.

For each method we perform 10 executions for every input of n addresses and we compare the algorithms in terms of the average durations of the "optimal" routes that they generate and in terms of their average total execution time (i.e., network delay+run-time delay).

For the first method, in every run we test a different order of the same n addresses whereas in the second method, in every run we test n different addresses. Due to the great number of permutations that the permutation algorithm runs and due to the smartphone's hardware limitations, the algorithm can not be tested for $n>7$. Please note that for a large n (which is impractical, since it implies a user has more than 7 places to go to, and even in that case he/she can use the app for the most time-pressuring 7 and then use it again) we would be able to extend the processing through the use of a cloud-based server.

The results of the first method indicate the expected excellence of the permutation algorithm in terms of the optimal route's duration. Figure 12 shows that the permutation algorithm almost always finds the global optimum for $n<8$ (this is the reason why only two algorithms are mainly shown in the results, i.e., that the permutation algorithm's route has zero difference from the optimal route and therefore no such difference exists; the same explanation stands for Figures 14 and 16, below). This accuracy however comes with the price of the very high complexity of the algorithm, causing its execution time to be by far the worst in comparison to the other two algorithms, as shown in Figure 13. As we can see from the results for 6 addresses, the permutation algorithm requires approximately triple execution time in comparison to Tabu Search and Simulated Annealing, and for seven addresses this gap grows to over a minute. Therefore its use for over 6 addresses is not recommended because such high delays are most likely not tolerable by the average mobile user.

The other two algorithms display similar behavior both in terms of execution time and route's duration. The results indicate that both algorithms are significantly faster in execution time than the permutation algorithm. Regarding the route's duration, both algorithms present excellent results for small inputs (n<4), as shown in Figure 12. For larger values of n the performance of the Tabu Search algorithm deteriorates fast; its "optimal" routes are ≥40% longer in comparison to the actual optimal ones when n>5, and more than 50% longer than the actual optimal ones for n≥10. The Simulated Annealing algorithm shows a slower performance deterioration but it also leads to more than 50% larger route durations, for n=10.

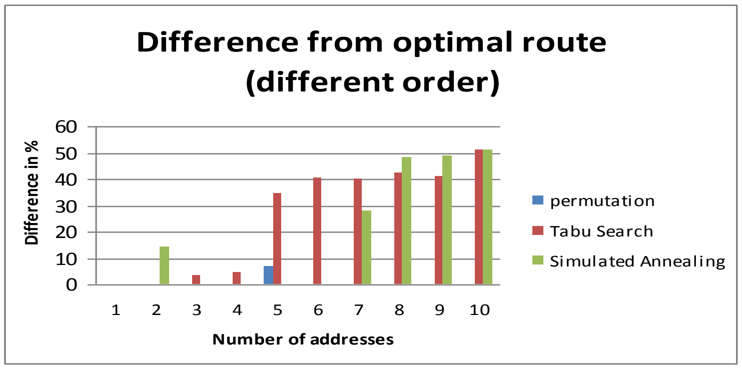


Figure 12. Comparison of the three algorithms in terms of their efficiency in finding the shortest route.

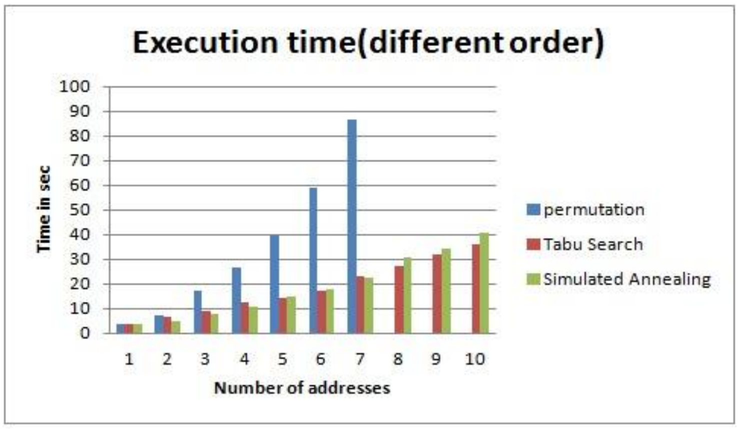


Figure 13. Comparison of the three algorithms in terms of their execution time.

In conclusion, the accuracy of the permutation algorithm can not compensate for the tradeoff of exceeding delays. The other two algorithms, which attempt to find local minima and extract the best route among them, offer much lower execution times due to their lower complexity at the cost of being unable to find the global optimum. Additionally, it needs to be emphasized that the ordering of the addresses affects the Tabu Search algorithm because in every run new local searches must be made. Our proposed variant of the Simulated Annealing algorithm offers better results than Tabu Search for $n<8$ and comparable results for larger values of n. The reason is the sorting at the beginning of the algorithm and the added solutions to the basic ones from the pairwise interchange.

The results of the second method were almost identical to the ones extracted from the first method. The permutation algorithm is still the most accurate and the one with the largest execution time, whereas the other two algorithms are significantly faster (Figure 15). Both algorithms find "optimal" solutions that lead to route durations longer than the truly optimal ones by more than 40%, for $n>5$ (Tabu Search) and $n>7$ (for our Simulated Annealing algorithm, which outperforms Tabu Search, as shown in Figure 14). It should be noted that the app can also work by adding time constraints for the visit to each address, associated with the visit to each address. However, this addition does not alter the nature of our results in terms of the comparison of the three algorithms, which is the focus of this paper.

Finally, we studied the case where the addresses entered by the user are within a short range (largest distance between addresses was 500m). Although the performance of the Tabu Search algorithm was not affected by this change, our Simulated Annealing algorithm displayed significant improvements: the difference between its computed “optimal” routes and the actual optimal ones dropped on average to under 10% (Figure 16). The proposed sorting at the beginning of the SA algorithm contributes to this improvement.

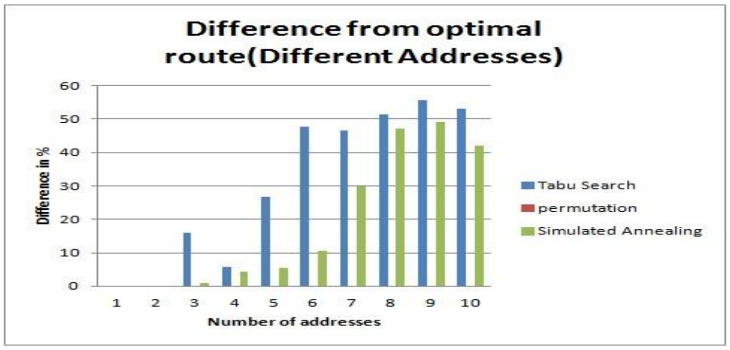


Figure 14. Comparison of the three algorithms in terms of their efficiency in finding the shortest route.

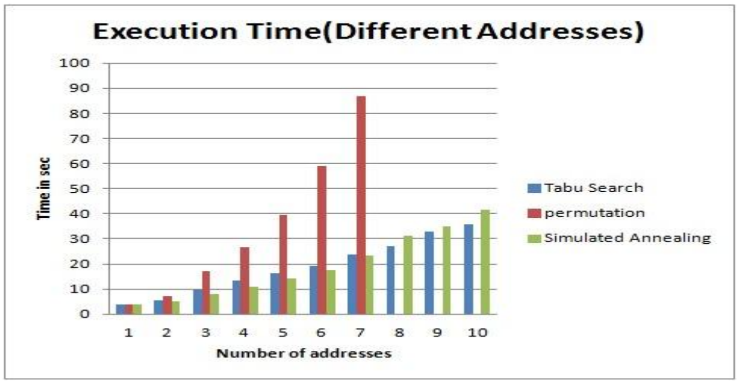


Figure 15. Comparison of the three algorithms in terms of their execution time.

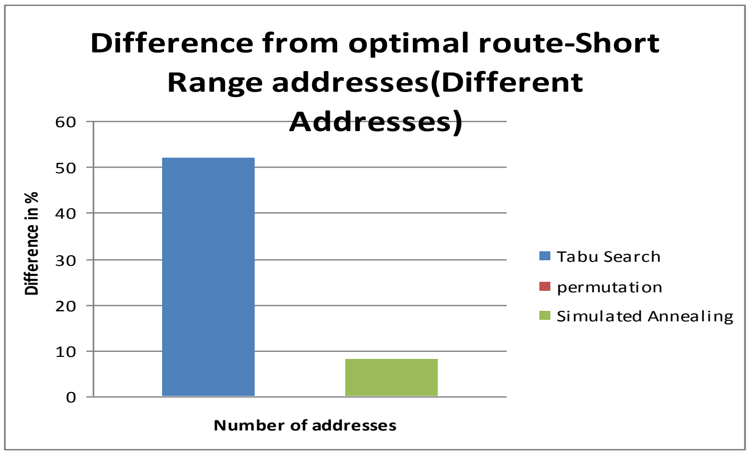
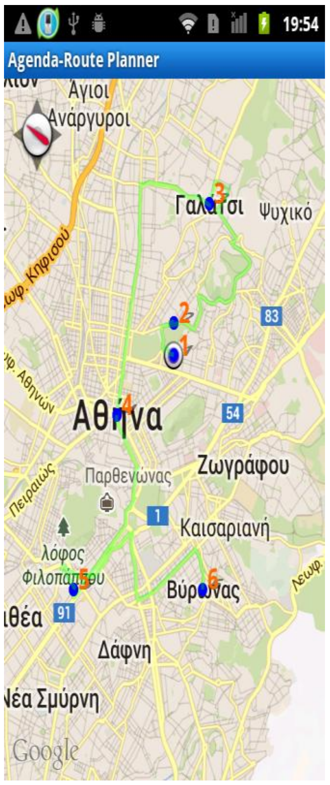
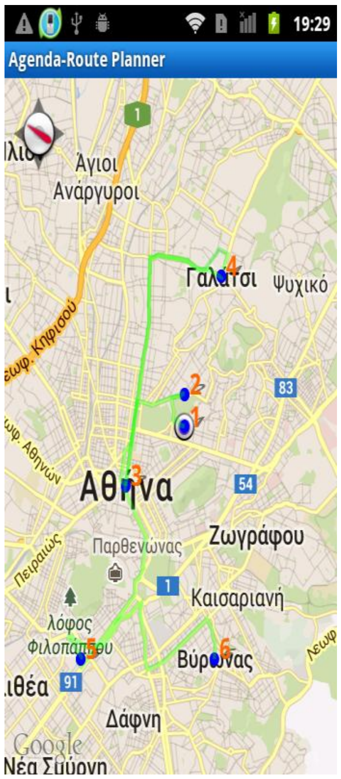


Figure 16. Comparison of the three algorithms in the case of short-range addresses.

Finally, Figure 17 shows an example of the differences in the shortest route computed by the three algorithms when given the same input of addresses (in this case, our example is from the city of Athens whereas the former examples and results were from the city of Chania, in Crete). It is clear from Figure 17 that the three solutions are different.





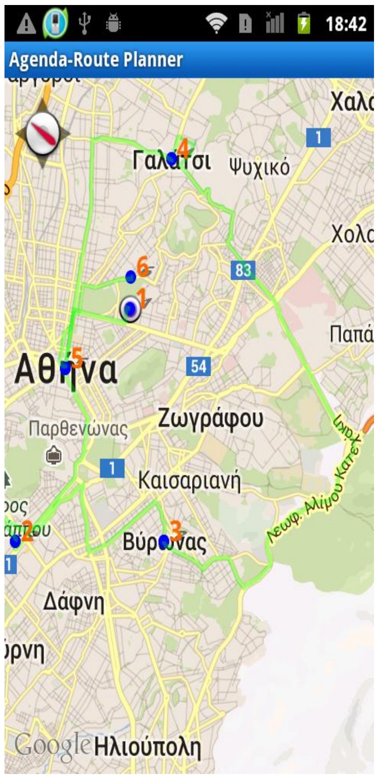


Figure 17. Left to right: Permutation Algorithm, Simulated Annealing, Tabu Search solutions for the shortest route using the same input of addresses.

V. FINAL COMMENTS, CONCLUSION AND FUTURE WORK

In this short paper we presented our effort to create a new mobile application that aims to relieve part of the burden of the complexity of daily life. We evaluated the performance of three algorithms on problems related to finding shortest routes, and we discussed the algorithms' advantages and weaknesses. Our evaluations have shown that an excellent solution in terms of both execution time and accuracy of results is difficult to find, which is why certain compromises have to be made to achieve a balance between computational load and accuracy in results. Our application, BLS, also includes functionalities that further improve user experience.

We have evaluated BLS on a home Wi-Fi network which provided stable speed and good accuracy of acquiring our current position. On a 3G network our experiments have shown that users should expect on average a 50% increase in network delay and a significant offset on the estimate of their current position. Therefore it is strongly advised that GPS should be enabled in such a case, despite being energy-consuming. The above remark, of course, is not valid for phones that support offline map services. These phones are still a minority, but even when/if phones with this capability prevail, the impact in our application will only be the minimization of network delay. Hence, once again this does not alter the nature of our results regarding the computational complexity of the three algorithms used in our work.

Our future work involves the evaluation of the performance of more algorithms such as genetic algorithms and hybrid solutions in order to improve the accuracy of our results in finding the shortest route without compromising, via additional delays, the user's experience. BLS will soon be available in Google Play, with the use of the permutation algorithm for n≤4 and our SA algorithm for n>4.

REFERENCES

1.[Online]: http://www.openhandsetalliance.com/android_overview.html

2.[Online]:http://developer.android.com/about/versions/android-2.3-highlights.html

3.[Online]: http://developer.android.com/guide/components/activities.html#StartingAnActivity

4.M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman Eds, 1979.

5.F. Glover, "Tabu Search - Part I", ORSA Journal on Computing, Vol. 1, No. 2, 1989, pp. 190-206.

6. K. Boloor, R. Chirkova, T. Salo and Y. Viniotis, "Heuristic-based request scheduling subject to a percentile time SLA in a distributed cloud", in Proc. of the IEEE Globecom 2010, Miami, USA.

7.[Online]: http://voidexception.weebly.com/simple-tabu-search-using-java---traveling-sales-man-tsp.html

8. A. M. Law and W. D. Kelton, "Simulation modeling &analysis" (2nd ed.), 1991, McGraw Hill Inc.

9. G. Chen and D. Kotz, "A survey of context-aware mobile computing research", Technical Report, Dartmouth College, 2000, [Online]: http://www.cs.dartmouth.edu/reports/TR2000-381.pdf

10. S. Tabatabaei, A. Gluhak and R. Tafazolli, "A Survey on Smartphone-Based Systems for Opportunistic User Context Recognition", ACM Computing Surveys, Vol. 45, No. 3., 2013.

11. V. Pejovic and M. Musolesi, "Anticipatory Mobile Computing: A Survey of the State of the Art and Research Challenges", University of Birmingham Technical Report CSR-13-02, December 2013.

12. I. Bokharouss, W. Wobcke, C. Yiu-Wa, A. Limaru and A. Wong, "A Location-Aware Mobile Call Handling Assistant," in Proc. of the 21st IEEE International Conference on Advanced Information Networking and Applications Workshops (AINAW), 2007, Vol.2, 2007, pp. 282-289.

13. J. Roth, “Context-aware apps with the Zonezz platform”, in Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld ’11).