# Table of Contents

Design and Implementation of a Threaded Search Engine for Tour Recommendation Systems ........................................ 1  
*Junghoon Lee, Gyung-Leen Park, Jin-hee Ko, In-Hye Shin, and Mkiyung Kang*

Analysis of the Position Effect to the Vehicle Speed and Stop Probability ................................................................. 8  
*Junghoon Lee, Gyung-Leen Park, Hye-Jin Kim, Min-Jae Kang, Cheol Min Kim, and Jinhwan Kim*

Approach to Privacy-Preserve Data in Two-Tiered Wireless Sensor Network Based on Linear System and Histogram .......... 17  
*Van H. Dang, Sven Wohlgemuth, Hiroshi Yoshiura, Thuc D. Nguyen, and Isao Echizen*

Fine-Grained Access Control for Electronic Health Record Systems ................................................................. 31  
*Pham Thi Bach Hue, Sven Wohlgemuth, Isao Echizen, Dong Thi Bich Thuy, and Nguyen Dinh Thuc*

On Indiscernibility in Assessments .................................................. 39  
*Sylvia Encheva*

Flexible Ubiquitous Learning Management System Adapted to Learning Context ......................................................... 48  
*Ji-Seong Jeong, Miuye Kim, Chan Park, Jae-Soo Yoo, and Kwan-Hee Yoo*

Effects of Knowledge Sharing and Social Presence on the Intention to Continuously Use Social Networking Sites: The Case of Twitter in Korea ........................................................................ 60  
*Bong-Won Park and Kun Chang Lee*

Exploring Individual Creativity from Network Structure Perspective: Comparison of Task Force Team and R&D Team ............................................. 70  
*Kun Chang Lee, Seong Wook Chae, and Young Wook Seo*

Webpage Segments Classification with Incremental Knowledge Acquisition ......................................................................... 79  
*Wei Guo, Yang Sok Kim, and Byeong Ho Kang*

Multi Agent System Based Path Optimization Service for Mobile Robot ........................................................................... 88  
*Huyn Kim and TaeChoong Chung*
Design and Implementation of a Threaded Search Engine for Tour Recommendation Systems*

Junghoon Lee¹, Gyung-Leen Park², Jin-hee Ko¹, In-Hye Shin¹, and Mikyung Kang²,***

¹ Dept. of Computer Science and Statistics, Jeju National University, 690-756, Jeju Do, Republic of Korea
{jhlee, jhko, ihshin76}@jejunu.ac.kr
² University of Southern California - Information Sciences Institute, VA22203, USA
glpark@jejunu.ac.kr, mkkang@isi.edu

Abstract. This paper implements a threaded scan engine for the $O(n!)$ search space and measures its performance, aiming at providing a responsive tour recommendation and scheduling service. As a preliminary step of integrating POI ontology, mobile object database, and personalization profile for the development of new vehicular telematics services, this implementation can give a useful guideline to design a challenging and computation-intensive vehicular telematics service. The implemented engine allocates the subtree to the respective threads and makes them run concurrently exploiting the primitives provided by the operating system and the underlying multiprocessor architecture. It also makes it easy to add a variety of constraints, for example, the search tree is pruned if the cost of partial allocation already exceeds the current best. The performance measurement result shows that the service can run even in the low-power telematics device when the number of destinations does not exceed 15, with an appropriate constraint processing.

1 Introduction

With the development of vehicular telematics networks, many new services can be provided to drivers according to their vehicle types. For example, a rent-a-car driver can retrieve a bunch of real-time information on the current traffic condition as well as the tourist attraction he wants to visit. In addition, a taxi driver can pick up a passenger according to the dispatch system [1]. Such telematics services will evolve along with the development of new vehicular communication technologies and the performance upgrade of telematics devices. With the ever-growing communication speed and the sufficient computing power, more

* This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency). (NIPA-2010-(C1090-1011-0009)).
*** Corresponding author.
sophisticated services can be developed and provided to the users, even though they need to process a great volume of data and run a complex algorithm.

Tour planning is considered to be one of the most promising telematics services, especially for rent-a-car drivers on the tourist place [2]. Its main function is to decide the tour schedule based on personal preference, current traffic condition, tourist attraction information, and so on, as shown in Figure 1. In most tourist places, the information on each tourist attraction is usually organized already in the database, or sometimes in ontology [3]. Such information covers the details on each tour point including how to reach, how much time it takes on average, activity to do, and so on. Hence, the amount of data and their processing grows significantly. Meanwhile, Jeju province in the Republic of Korea has diverse kinds of unique tourist attractions, such as beaches, volcanoes, cliffs, mountains, subsidiary islands and so on, while hosting a lot of tour activities including ocean sport, golf, hiking, and the like within a relatively small area. Since so many diverse tour plans are possible, the tourist needs the assistance from an intelligent tour planning system [4].

![Tour planning system diagram](image)

Fig. 1. Tour planning system

Generally, the tour recommender system consists of two steps [5]. First, the system selects the candidate POIs sorted by their ranks or scores determined by the system-specific criteria and algorithms. The second step filters candidate POIs to generate a corresponding tour schedule, considering calculated ranks, given time constraint, user location, and current traffic condition as shown in Figure 2. Particularly, for a tourist who accesses and retrieves information via a telematics device inside rent-a-car, it is necessary to minimize the response time to say nothing of the number of user-system interactions. Computing an optimal route for multiple POIs is an instance of TSP (Traveling Salesman Problem) which is known to be NP-hard, and the number of candidate POIs is extremely critical to the response time.

Obviously, the efficient heuristic and the multithreaded program are two best strategies to speed up the computation time for a complex problem. TSP is one of the most classical computation-intensive problems for which a lot of famous heuristic methods have been developed [1]. Most of them, including
Lin-Kernighan algorithm, achieved great improvement in computation speed and accuracy [6]. However, existing schemes mainly consider just the distance between the destinations, so it is quite difficult to integrate additional constraints a driver may specify and submit in tour planning. For example, lunch time must be put between 12:00 and 13:00 on a specific restaurant type, or the museum must be visited prior to beach area. Without an efficient heuristic, the time complexity of \( O(n!) \) is very sensitive to the number of POIs that are to be scheduled. Moreover, those services need to process a great amount of data including personal preference, detailed POI attributes, and the navigation-scale road network [7].

Even though the execution time grows too much to be used for larger \( n \), a strong constraint can prune the unnecessary branch in the search space tree. Moreover, along with a well-defined constraint, the multithreaded execution can further improve the computation time without sacrificing the accuracy of the final solution. For the sake of verifying how fast the multithreaded program can response to the user request, actual implementation and application to the real data is important. In this regard, this paper is to build a multithreaded computing framework for the tour planning service and measure its performance, aiming at providing convenient tour schedule to vehicular telematics users. The framework includes threaded search engine and constraint processing primitives.

This paper is organized as follows: After issuing the problem in Section 1, Section 2 designs and implements the threaded search engine for the tour schedule. Performance measurement results are demonstrated to reveal the practicality of our implementation in Section 3. Finally, Section 4 summarizes and concludes this paper with a brief introduction of future work.

2 System Design and Implementation

The overview of our framework is described in Figure 2. The first step is to process the road network represented by the ESRI shape file format. Based on this, a route between every pair of POIs can be calculated. In integrating the up-to-date traffic condition, a commercial or civilian service is available in most cities including Jeju. This information is mapped to the road network. Second,
POI details have been organized in travel ontology by another member of our research team [3]. Third, POIs are classified by their characteristics first and necessary fields are defined, while each field is associated with the appropriate attribute [8]. Additionally, how to specify the user preference and constraint was researched in [9]. Here, user preference is defined by geographical terrain preference, time preference, and activity preference, based on general attributes such as demographic information, tour history, and tour motivation.

As a preliminary step to integrate all these components, this paper implements the multithreaded search space scan engine which assigns subtrees in the search space to each thread. The multiprocessor architecture is now commonly available in PCs, notebook computers, and even in vehicular telematics devices. In this architecture and the Windows-based operating system, each thread is allocated to each processor and executed. The tour schedule service can run either on the in-vehicle telematics device or on the remote high capacity server and then transmitted to the driver via the appropriate wireless communication channel. Our implementation runs on the machine equipped with Intel Core2 Duo CPU working at 2.4 GHz clock, 3.0 GB memory, and Windows Vista operating system, while the measurement of actual execution time can give us a useful guideline, for example, where the service is placed, how much constraint can be combined, up to how many destinations the service can provide the reasonable service level, and the like.

To begin with, this paper makes a program which calculates the distance between each of n nodes using the A* algorithm [10]. Classical Dijkstra's algorithm consumes too much time, as the number of pairs gets larger for larger n. Even if A* algorithm cannot always find the optimal solution, its execution time is much short [8]. Moreover, one-to-many version of A* is also available [11]. In the subsequent version, we will implement a thread that works in background to update the cost between the major POIs according to the traffic condition change. For n POIs, it is necessary to visit n! leaf nodes, each of which corresponds to a complete schedule, if we don't have another constraints. Furthermore, each of n top-level subtrees has (n - 1)! leaf nodes. In the dual CPU platform, the tour planner creates two equal-priority threads, each of which processes n/2 subtrees. If no other constraint is given, the execution time of each thread is the same. Otherwise, it is advantageous to create n threads and allocate them one by one each time a thread completes scanning one subtree.

Next, constraint processing can prune unnecessary branch traversal in the search space tree. Basically, after the search procedure reaches a leaf node, that is, when a complete schedule is built, the cost for the schedule is calculated. If the cost is less than the current best, namely the smallest cost, the current best will be replaced by the new one. However, if the cost for an incomplete schedule already exceeds the current best, it is useless to execute the remaining allocation, as adding a new entry will only increase monotonously. Accordingly, our implementation checks the cost of partial allocation before proceeding to the next entry allocation. Moreover, for the real-life tour schedule, such constraints listed in Section 1 can further narrow the search space, reducing the execution
time. It has the same effect as the reduction in the number of destinations, which is critical in $O(n!)$ search space processing.

3 Performance Measurement

This section measures the performance of our system implementation. Figure 3 plots the measurement result for the actual execution time of our implementation. The experiment makes the number of POIs change from 7 to 12. For simplicity and without losing generality, the cost between each node is selected randomly. As shown in this figure, the execution time is insignificant until $n$ reaches 8, where single-threaded version takes 0.01 sec and the dual-threaded version 0.007 sec, respectively. For the number of POIs less than or equal to 10, it takes less than a second. When the number of nodes is 11, it takes 8.8 seconds to make a tour schedule in threaded version, cutting the computation time almost by half. The thread context switch and management overhead prevents further improvement. From here, the program can clearly take advantage of the threaded execution. However, if the number exceeds 12, even the dual-threaded version cannot provide a reasonable execution time without a well-defined constraint processing.

![Figure 3. Execution time measurement](image)

Anyway, this result indicates that the tour planner can work also on the telematics device when $n$ is less than 10. For a single day trip, the number of POIs generally lies in this range. In addition, threaded version can achieve the computation speed-up almost proportional to the number of CPUs when $n$ is larger than or equal to 10, significantly reducing the response time.

Figure 4 plots the effect of constraint processing. For two threads, each thread routine stores its own current best to alleviate the interference between the threads in accessing the shared variable: Even if the single common current best
can further reduce the search space, mutual exclusion on the shared variable takes indiscernible overhead. The execution time is just 0.39 sec, while the non-constrained version takes 102.7 sec as shown in Figure 3 and Figure 4. In our experiment, the execution time is less than 30 sec until the number of POIs is 17, but it goes to 700 sec when the number of POIs is 18. After all, this experiment reveals that an efficient heuristic or strong constraint can significantly improve the execution time. The tour schedule is highly likely to have strong constraints, as the tourist has different requirements according to his tour length, preference, and the like. So, our framework can work efficiently in the real-life environment.

![Figure 4. Effect of constraint processing](image)

4 Concluding Remarks

This paper has designed and implemented a threaded scan engine for the $O(n!)$ search space for the tour recommendation system and measured its performance. The implemented system allocates the subtree to the respective threads and makes them to run concurrently on a multiprocessor architecture. In addition, the constraint processing primitive is added to prune the subtree when the partial allocation in progress has no hope to make a solution better than the current best. The measurement result shows that the service can run even in the telematics device for the single day trip, even without any constraint processing. In our experiment, the execution time is less than 30 sec until the number of POIs is 17 on an average performance personal computer. Based on this implementation, an efficient service framework can be built for the development of prospective vehicular telematics services by integrating existing POI ontology, mobile object database, personalization profile.

At the next step, we are to classify constraints that may be given to our search engine and measure their effect, namely, how much each constraint group can reduce the search space. The constraint may be time-based, sequence-based, similarity-based, group-based, and so on.
References


