Fall 2010

Jeppe Rishede Thomsen
Department of Computing
Hong Kong Polytechnic University

Abstract

All the papers either solve the problem of a more efficient cache in a specific domain, or use the network domain, which are both relevant, but not really useful when looking at shortest path caching.

The papers show some interesting ways to use cache, but ultimately their approaches are very domain or query specific so their approaces to caching and cache replacement/invalidation can not be applied directly.

1. Introduction

2. Problem

2.1. Definitions and problem setting

We assume a setting where owners of mobile, positioning enabled, devices want route planning assistance. We assume users prefer online route planning services over offline solutions. we expect users to use network enabled capable of determining and visualizing users location and route. Users want fast response times from online services, comparable to using an offline application [**?**] Using a cache reduces the computational burden [**?**] on an online service, providing faster end-user response time [**?**] by both freeing up computational resources to calculate new routes, as well as being able to immediately provide the shortest path result from the cache. We assume a scenario using only server side caching.

2.2. model

We use a uniform random model(URM) together with a simple 'map',the graph in figure 1, enable us to clearly argue about the advantages expected when using server side caching of shortest-path queries. By using the simple map (fig.1) and a URM together with we can reason about the probabilities that a specific query will occur. Figure 2 illustrates the number of queries possible for a map with 2,3, and 5 locations for figure 2A, 2B, and 2C respectively. Each line underneeth each of the simple graphs represents two possible queries (A->B, B<-A). The number of shortest-path queries possible on a tree-graph with n vertices is $n * (n - 1)$. The probability of seeing any one query, q_i is then $P(q_i|n) = (n * n - 1)^{-1}$

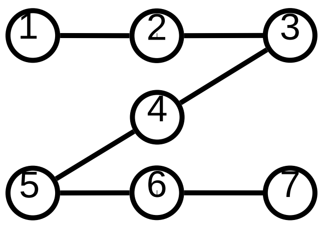

Figure 1. Simple map

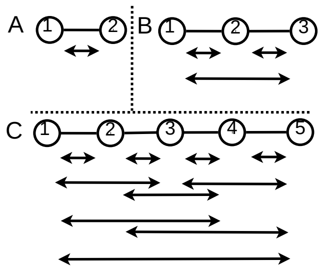

Figure 2. Number of queries possible

2.3. methods

For the sake of simplicity the methods presented in this section will all be based on figure 1 and 3. Figure 1 shows a simple graph which we will use as our map and figure 3 shows the simple scenario in which a user (fig. 3A) issues a route-planning query from 1 to 4 (fig. 3) to an online route-planning server with a build in cache (fig. 3B).

2.3.1. Baseline. The strait forward baseline solution is illustrated in figure 3B. The idea is a server side shortest path cache which will store each query result in the

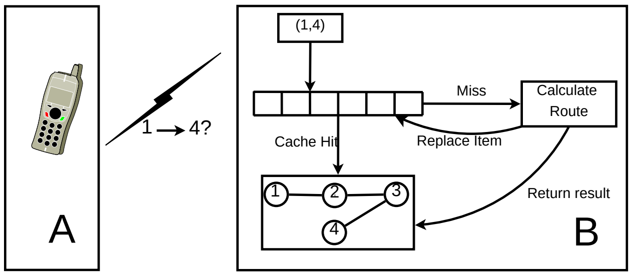

Figure 3. simple graph

cache and only consider exact query matches as cache hits, and to only use a simple cache policy such as LRU or FIFO. The advantage of this solution is clear: it is simple and easily implemented. This simplicity is however obviously also it's main disadvantage, as it is too simple and very inefficient in terms of the utility the cache provides. Using items in the cache only when there is an exact match makes it exceedingly unlikely to get a cache hit due to the nature of route planning (many people share parts of routes, but few the same start and end points) and the sheer number of start-/end-point combinations possible.

2.3.2. ImpBaseline. One way to possible increase the utility of a naive cache as proposed in 2.3.1 would be to exploit the optimal substructure property [1] of the cache items. There is a significant increase in cache hits to be expected by utilizing the optimal substructure of shortest path cache items since it is unlikely many people will plan a route from/to the same place, but it is very likely that some sub-parts will be shared among users, and some users' full path laying within a longer path already calculated. The idea is illustrated in figure 4 where the baseline method would be able to answer query Q1 from the cache, but not Q2. It is this specific disadvange which ImpBaseline addresses and ImpBaseline can therefor answer both Q1 and Q2 from the cache since the result of Q2 now exist as a solution to a subpath of cache item C3. The disadvantage of doing this is the aditional computational resources required to examine the substructure of each cached shortest path search result. It is currently not known if doing this is worth the efford, compaired to just calculating the route, possibly multiple times.

2.3.3. OSC - optimal substructure cache. OSC is more advanced than the two previous proposed solutions and therefor the scenario has been updated in figure 5. To further improve upon ImpBaseline we will again utilize the optimal substructure, making it possible to have much fewer items in cache and still

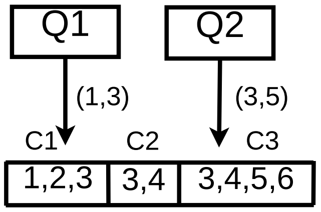

Figure 4. Queries

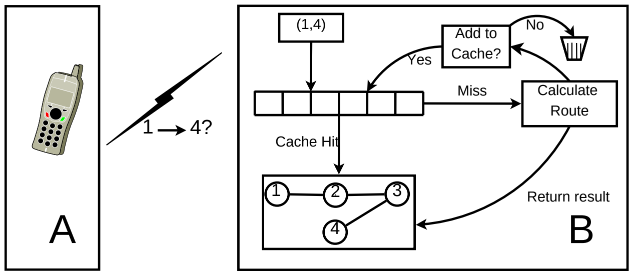

Figure 5. Advanced graph

retain a high cache hit percentage [**?**]. Results with sub-paths shared by many users and longer, rather than short, paths are prefered to increase the utility of the cached shortest-path results.

By adding a more intuitive cache replacement policy which takes in to consideration both the usage of each cache item, as well as the coverage of previously often seen queries it is likely that the utility of the cache would be much higher. This addition is shown with the addition of the "‘add to cache"’ box in figure 5B, added to show a huristic[1] will be used instead of a very simple method like LRU.

1. the actual huristic will ofcause only be defined later

References

[1] T. H. Cormen, C. E. Leiserson, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.