Fall 2010

February 8, 2011

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. Motivation

Shortest Path (SP) calculation is much slower than retrieval from cache (**TODO:** ***figure out how by how much for each SP algorithm***)

Definition 2.1: A **SP Cache element** is a list $(v_s, v_1, ..., v_t)$ of vertices where v_s/v_T are the start/end vertices of the list. v_s is connected with v_t by a shortest path defined by the vertices between v_s and v_T in the list.

The cache is a set $\{SP_0, ..., SP_n\}$ of SP cache elements.

Definition 2.2: Cache replacement: Given two otherwise equal SP results, i.e. having the same length and having the same **value** (by some scoring mechanism defined for cache replacement) the SP result already in the cache will stay in the cache.

TODO: ***what other SSSP algorithms might be useful in comparing speed difference between cache and SP calculation?***

TODO: ***Argue that since storing map data is cheap (little space required) but calculating SP is very expensive, then using the same, or more, space for cache than used for map data makes sense.***

i	item freq	item length	C(l)
1	11	10	100
2	10	15	200
2*	2	15	2000

2.1. SP Results - facts

- number of times each node in query has been relevant to:
  - a cache hit
  - a query result
- number of times source/target -nodes has been used in a query
- sub-paths which are popular in query results (connected nodes which all "'score high"')

2.2. Meeting

C(l) calculates the number of vertices visited when finding a shortest path with Dijkstra

TODO: ***modify Dijkstra implementation to return this number***

item length could be considered a factor instead of a score (replacing + with * when adding op the score with item frequency)

TODO: ***write argument why to use + or * when calculating score.***

Item 1 has higher frequency than item 2, but the saving of putting item 2 in cache is much higher.

item one cost 11*100=1100 and item 2 10*200=2000.

assuming cache hit cost 1, then adding item one would save = 1100-(100+10*1)=990, and item two would save 2000-(200+9*1=1791. the equation is: $(total\_cost\_without\_cache) - (cost\_of\_one\_Dijkstra\_run + (total\_runs - 1 * cost\_of\_cache\_hit))$

need to focus on arguing about archiving lowest possible run time, do not connect to strongly with one graph, argue independently of map/graph that cost model and methods will be good. **TODO:**

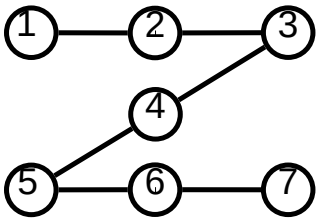

Figure 1. Simple map

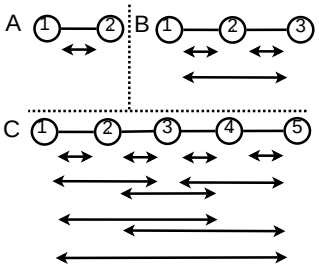

Figure 2. Number of queries possible

3. Problem

3.1. 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.

3.2. model

TODO: ***add more examples on advantages/disadvantages of*** $\{LRU, ImpBaseline, OSC\}$
TODO: ***Fig.4- add about text: space per edge***
TODO: ***Fig.4- add about text: probability cache item is useful*** 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 underneath 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 $P(q_i|n) = (n * n - 1)^{-1}$

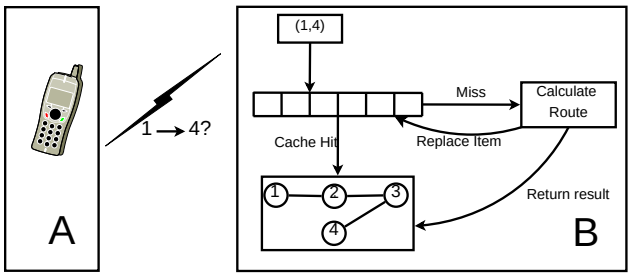

Figure 3. simple graph

3.3. 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).

3.3.1. 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 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.

3.3.2. Improved Baseline. One way to possible increase the utility of a naive cache as proposed in 3.3.1 would be to exploit the *optimal substructure property* [**?**] of the cache items. There is a significant increase

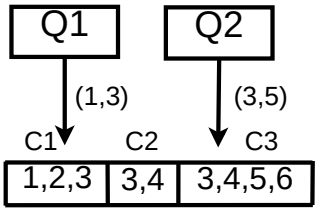

Figure 4. Queries

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 disadvantage 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. Doing this adds a need for additional computational resources **TODO:** ***how many resources?*** required to examine the substructure of each cached shortest path search result. It is currently not known if doing this is worth the effort, compared to just calculating the route, possibly multiple times.

3.3.3. 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 Improved Baseline we will again utilize the optimal substructure, making it possible to have much fewer items in cache and still retain a high cache hit percentage [?]. Results with sub-paths shared by many users and longer, rather than short, paths are preferred 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 heuristic[1] will be used instead of a very simple method like LRU.

4. Related work reference

reference support for related work section.

1. the actual heuristic will of cause only be defined later

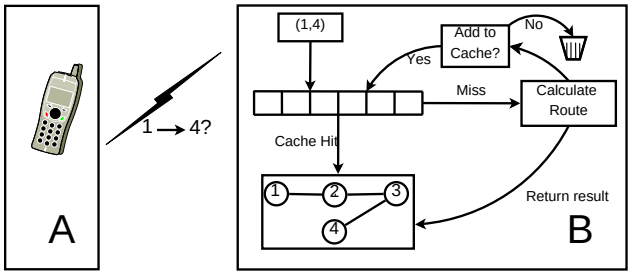

Figure 5. Advanced graph

4.0.4. On effective presentation of graph patterns: a structural representative approach. They develop an approach that combine two focuses when mining patterns in graphs. 1. they introduce a method to relax the tightness of the pattern subgraph pattern matching, so they can have high support for subgraphs which are very similar, but not exact. 2. as many mining approaches return allot (often very similar) patterns, they propose a method to collapse similar patterns so the user is presented with something that is easier to get an overview of and gain an understanding of the data. [**?**]

4.1. Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments

They develop two cache replacement and invalidation techniques for mobile clients communicating with a LBS. They argue that in the setting of spatial data and LBS then it is important to consider more than just the access time when doing cache replacement. They look at the spatial area where an object in the cache is valid as well as the direction the user is moving. They do this besides calculating the probability that this object will be accessed again.

Assumes all POI objects are fixed size and no updates will be made. [**?**]

4.2. Nearest-Neighbor Caching for Content-Match Applications

[**?**]

4.3. Caching Content-based Queries for Robust and Efficient Image Retrieval

They study how to do caching with Content-based Image Retrieval, and they support range and kNN queries. They focus on how to do caching when many

of the queries are similar, but not the same (e.g. picture cropped or color changes) without polluting the cache. Their approach works in metric space and they develop an approximate method to check if the result can be satisfied by the cache. They archive good results, getting few direct cache hits, but still satisfying many queries from similar queries in the cache.

[**?**]

4.4. Caching Complementary Space for Location-Based Services

They develop the notion of Complementary Space(CS) to help better use a cache on a mobile client. CS is different levels for representing the objects on a map within MBRs. At the lowest level they just show the object, and as the levels go up they include more and more objects within MBRs, looking at the trade of in communication up/down link from a mobile client. They always have the entire world represented within the clients cache, at different levels, and offer no solution to how they will handle server updates to the map.

This is very similar to [**?**], although the approach does not formally depended on an R-tree, they still use one and offer no viable alternative, which lessens the difference even more. Their results are better than their competitors, including [?], though it seems that they stop their graphs just before [?] beats them.

4.5. Proactive Caching for Spatial Queries in Mobile Environments

They develop an approach which uses the index of an R-tree to add context to a cache of spatial object on a mobile client. They develop several communication and space saving techniques by representing less important parts of the R-tree in more compact ways, or just not storing the lower nodes/leaves of the tree. They also formally prove the asymptotic bounds of their algorithms.

[?]

4.6. Aggregate Aware Caching for Multi-dimensional Queries

They develop a method to re-use items in a cache for a data werehouse. They take advantage of the levels of aggregation which exist in OLAP query results and come up with a way where they can get aggregated results using full or partial more detailed data from the cache. The use most of the paper on showing and proving that their algorithms have a good running time and admit them selfes that the approach is not very mature, though they still manage to get resonable results. [**?**]

4.7. DynaMat: A Dynamic View Management System for Data Warehouses

abstract: Pre-computation and materialization of views with aggregate functions is a common technique in Data Warehouses. Due to the complex structure of the warehouse and the different profiles of the users who submit queries, there is need for tools that will automate the selection and management of the materialized data. In this paper we present DynaMat, a system that dynamically materializes information at multiple levels of granularity in order to match the demand (workload) but also takes into account the maintenance restrictions for the warehouse, such as down time to update the views and space availability. DynaMat unifies the view selection and the view maintenance problems under a single framework using a novel ąğgoodnessąĺ measure for the materialized views. DynaMat constantly monitors incoming queri and materializes the best set of views subject to the space constraints. During updates, DynaMat reconciles the current materialized view selection and refreshes the most beneficial subset of it within a given maintenance window. We compare DynaMat against a system that is given all queries in advance and the pre-computed optimal static view selection. The comparison is made based on a new metric, the Detailed Cost Savings Ratio introduced for quantifying the benefits of view materialization against incoming queries. These experiments show that DynaMat's dynamic view selection outperforms the optimal static view selection and thus, any sub-optimal static algorithm that has appeared in the literature. [**?**]

4.8. Cache-Oblivious Data Structures and Algorithms for Undirected Breadth-First Search and Shortest Paths

[**?**]

4.9. Cached Shortest-Path Tree: An Approach to Reduce the Influence of Intra-Domain Routing Instability

They assume a network setting and try to reduce the time and computational load it takes when network topology changes, as well as prevent any links from

being unreachable if the topology changes often. The propose a cache with shortest-path trees, arguing that even if the topology changes often, then it is mostly between the same configurations (e.g. a computer/router is turned off/on) meaning that a cache with the most common seen configurations will be able to drastically reduce the amount of computation needed to recalculate routing tables.

[**?**]

4.10. On Designing a Shortest-Path-Based Cache Replacement in a Transcoding Proxy

[**?**]

4.11. Optimizing Graph Algorithms for Improved Cache Performance

[**?**]