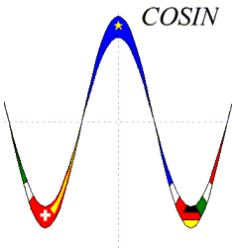

COSIN
IST–2001–33555

COevolution and **S**elf-organization
In dynamical **N**etworks

Algorithms to find paths and connections from local information

Deliverable Number:	D22
Delivery Date:	March 2004
Classification:	public
Contact Authors:	Marco Gaertler and Dorothea Wagner
Document Version:	final
Contract Start Date:	01-03-2002
Duration:	36 months
Project Coordinator:	Guido Caldarelli
Partners:	INFM *Italy*
	Università "La Sapienza" *Italy*
	Universitat de Barcelona *Spain*
	Université de Lausanne *Switzerland*
	Ecole Normal Superieure de Paris *France*
	Universität Karlsruhe *Germany*

Projects founded by the
European Commission under the
Information Society Technologies
Programme (1998-2002)

Abstract

Geographical information plays an important role in the physical representation of the Net as well as for algorithmic problems like finding paths and connections. For query intensive applications like web searching one can benefit from geographical information. Users in such a scenario continuously enter their requests and the main goal is to reduce the (average) response time for answering a query. The algorithmic core problem that underlies the above applications is a special case of the single source shortest path problem and the method of choice is Dijkstra's algorithm. The first phase of this deliverable was devoted to studying the algorithmic issues shortest paths computation in large networks. Techniques from route planning systems, spatial databases and web searching were considered with respect to their general applicability.

Geographical information plays an important role in the physical representation of the Net as well as for algorithmic problems like finding paths and connections. For query intensive applications like web searching one can benefit from geographical information. Users in such a scenario continuously enter their requests and the main goal is to reduce the (average) response time for answering a query. The algorithmic core problem that underlies the above applications is a special case of the single source shortest path problem and the method of choice is Dijkstra's algorithm [5]. The first phase of this deliverable was devoted to studying the algorithmic issues shortest paths computation in large networks. Techniques from route planning systems, spatial databases and web searching were considered with respect to their general applicability.

The application of shortest path computations in travel networks is widely covered in the literature; see e.g., [2, 10]. One of the important features in our scenario is the fact that the network does not change for a certain period of time while there are many queries for shortest paths. This justifies a heavy preprocessing of the network to speed up the queries. Although pre-computing and storing the shortest paths for all pairs of nodes would give us "constant-time" shortest-path queries, the quadratic space requirement for traffic networks with more than 10^5 nodes makes it prohibitive. The most commonly used approach for answering shortest path queries concerns variants of Dijkstra's algorithm [10], targeting at reducing its *search-space* (number of nodes visited by the algorithm).

In [9], we explore the possibility to reduce the search space of Dijkstra's algorithm by using precomputed information that can be stored in $O(n + m)$ space. Our main contribution is that we use the given layout of the graph to extract geometric information to answer the on-line queries fast. In fact, this paper shows that storing partial results reduces the number of nodes visited by Dijkstra's algorithm to only 10%. We use a very fundamental observation on shortest paths. In general, an edge that is not the first edge on a shortest path to the target can be safely ignored in any shortest path computation to this target. More precisely, we apply the following concept. In the preprocessing, for each edge e a set of nodes $S(e)$ is computed which are the nodes that can be reached by a shortest path starting with e. While running Dijkstra's algorithm, those edges e for which the target is not in $S(e)$ are ignored.

As storing all sets $S(e)$ would need $O(mn)$ space, we relax the condition by storing a geometric object for each edge that contains *at least* the nodes in $S(e)$. The shortest path queries are then answered by Dijkstra's algorithm restricted to those edges for which the target node is inside their associated geometric object. Note that this method does in fact still lead to a correct result, but may increase the number of visited nodes to more than the strict minimum (i.e., the number of nodes in the shortest path). In order to generate the geometric objects, a layout is used. For the application of travel information systems, such a layout is given by the geographic locations of the nodes. It is however not required that the edge lengths are derived from the layout. In fact, for some of our experimental data this is not even the case.

We present an extensive experimental study comparing the impact of different geometric objects using real-world test data from traffic networks, a typical field of application for the considered scenario. It turns out that a significant improvement can be achieved. Actually,

in some cases the speed-up is even a factor of about twenty.

The second contribution concerns the dynamic version of the above mentioned scenario; namely, the case where the graph may dynamically change over time as streets may be blocked, built, or destroyed, and trains may be added or canceled. In this work, we present new algorithms that dynamically maintain geometric containers when the weight of an edge is increased or decreased (note that these cases cover also edge deletions and insertions). We also report on an experimental study with real-world railway data. Our experiments show that the new algorithms are 2-3 times faster than the naive approach of recomputing the geometric containers from scratch.

Our dynamic algorithms are perhaps the first results towards an efficient algorithm for the dynamic single source shortest path problem without using the output complexity model under which algorithms for the dynamic single source shortest path problem are usually analyzed. We would also like to mention that existing approaches for the dynamic all-pairs shortest paths problem (see e.g. [11] for a recent overview) are not applicable to maintain geometric containers, because of their inherent quadratic space requirements.

The last contribution concerns methodological issues regarding our implementations, which have been carried out in `C++`. Implementing and supporting that many variations of Dijkstra's algorithm in `C++` is a tedious task if it is not planned carefully. We employ several techniques to maintain a common code base that is at the same time small, flexible and efficient. We use a blend of the design pattern *template method* [6], parameterized inheritance [3], and template meta-programming [1, 7].

Adding functionality to graph algorithms can be achieved by the design pattern *template method* [6] or an extension of the design pattern *visitor*, the approach of the BOOST graph library [4]. Our work deviates from the latter and is closer to the former, which it actually enhances to grasp parts of *aspect-oriented programming*.

Of course, many techniques are known to speed-up Dijkstra's algorithm heuristically, while optimality of the solution can still be guaranteed. In most studies, such techniques are considered individually. The focus of [8] is the *combination* of speed-up techniques for Dijkstra's algorithm. All possible combinations of four known techniques, namely *goal-directed search*, *bi-directed search*, *multi-level approach*, and *shortest-path bounding boxes*, are considered. It is illustrated how these can be implemented. In an extensive experimental study the performance of different combinations is compared and it is analyzed how the techniques harmonize when applied jointly. Several real-world graphs from road maps and public transport and two types of generated random graphs are taken into account.

References

[1] A. Alexandrescu; *Modern C++ design: generic programming and design patterns applied*; Addison-Wesley; 2001

[2] C. Barrett, K. Bisset, R. Jacob, G. Konjevod, M. Marathe; Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of

the TRANSIMS router; *Proc. 10th European Symposium on Algorithms (ESA 2002)*, eds. R. Möhring, R. Raman; LNCS 2461; 126–138; Springer; 2002

[3] G. Bracha, W. Cook; Mixin-based inheritance; *Proc. Conference on Object-Oriented Programming: Systems, Languages, and Applications / Proc. European Conference on Object-Oriented Programming*, ed. N. Meyrowitz; 303–311; ACM Press; 1990

[4] J. Siek, L.-Q. Lee, A. Lumsdaine; *The Boost Graph Library: User Guide and Reference Manual*; Addison-Wesley; 2002

[5] E. W. Dijkstra; A note on two problems in connexion with graphs; *Numerische Mathematik*; 1:269; 1959

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides; *Design Patterns: Elements od Reusable Object-Oriented Software*; Addison-Wesley Professional Computing Series; Addison-Wesley; 1995

[7] U. Eisenecker, K. Czarnecki; *Generative Programming in C++*; chap. 10, 397–501; Addison-Wesley; 2000

[8] M. Holzer, F. Schulz, and T. Willhalm; *Combining Speed-Up Techniques for Shortest Path Computations*, Preprint 2004

[9] D. Wagner, T. Willhalm and C. D. Zaroliagis; *Geometric Shortest Path Containers*; Universität Karlsruhe, Fakultät für Informatik 2004-5, 2004

[10] F. B. Zahn, C. E. Noon; A comparison between label-setting and label-correcting algorithms for computing one-to-one shortest paths; *Journal of Geographic Information and Decision Analysis*; 4(2); 2000

[11] C. Zaroliagis; Implementations and experimental studies of dynamic graph algorithms; *Experimental Algorithmics*, eds. R. Fleischer, B. Moret, E. M. Schmidt; LNCS 2547; 229–278; Springer; 2002