Computing All-Pairs Shortest Paths by Leveraging Low Treewidth

Léon Planken \textsuperscript{a} \hspace{1cm} Mathijs de Weerdt \textsuperscript{a} \hspace{1cm} Roman van der Krogt \textsuperscript{b}

\textsuperscript{a} Faculty EEMCS, Delft University of Technology, The Netherlands
\textsuperscript{b} Cork Constraint Computation Centre, University College Cork, Ireland

Finding shortest paths is an important and fundamental problem in communication and transportation networks, circuit design, graph analysis and is a sub-problem of many combinatorial problems, such as those that can be represented as a network flow problem. In particular, in the context of planning and scheduling, finding shortest paths is important to solve the Simple Temporal Problem (STP) \cite{1}, which in turn appears as a sub-problem to the NP-hard Temporal Constraint Satisfaction Problem (TCSP) and Disjunctive Temporal Problem (DTP) classes, powerful enough to model e.g. job-shop scheduling problems. The shortest path computations in these applications can account for a significant part of the total run time. Instances of the STP, called Simple Temporal Networks (STNs), have a natural representation as directed graphs with real edge weights. The canonical way of solving an STP instance \cite{1} is by computing all-pairs shortest paths (APSP) on its STN, using e.g. the Floyd–Warshall algorithm. The state of the art for computing APSP is Johnson’s algorithm, which runs in $O(n^2 \log n + nm)$ time using a Fibonacci heap.

In this paper we present a new algorithm for APSP, similar to our P\textsuperscript{3}C algorithm for enforcing partial path consistency \cite{2}. This algorithm advances the state of the art in computing APSP. In graphs of constant treewidth, the run time is bounded by $O(n^3)$, which is optimal since the size of the output is $\Theta(n^3)$. When the algorithm is applied to chordal graphs, it has a run time of $O(nm)$, which is a strict improvement over the state of the art (with a run time of $O(nmw_d^2)$, where $w_d$ is the induced width of a vertex ordering $d$). The upper bound of the run time of our algorithm on general graphs is $O(n^2w_d)$; this is better than the bound on Johnson’s algorithm if $w_d \in O(\log n)$.

Our algorithm for all-pairs shortest paths relies on the fact that the graph has already been made directionally path-consistent along a certain vertex ordering. Directed path consistency (DPC), included here as Algorithm 1, was proposed in \cite{1} for checking whether an STP instance is consistent (i.e. the graph contains no negative cycles). This algorithm takes as input a weighted directed graph $G = (V,E)$ and a vertex ordering $d$, which is a bijection between $V$ and the natural numbers $\{1, \ldots, n\}$. In our paper, we simply represent the $i$th vertex in such an ordering as the natural number $i$. The weight on the arc from $i$ to $j$ is represented as $w_{i \rightarrow j}$; further, our shorthand for the existence of an arc between these vertices, in either direction, is $\{i,j\} \in E$. In iteration $k$, the algorithm adds edges (in line 5) between all pairs of lower-numbered neighbours $i,j$ of $k$, thus triangulating the graph (i.e. making it chordal; we denote the number of edges in this chordal graph by $m_c \geq m$). Moreover, assuming $i < j$ and given a path $\pi$ between such a pair of neighbours that except for its endpoints lies outside $G$, a defining property of DPC is that it ensures that $w_{i \rightarrow j}$ is no higher than the total weight of this path. This implies in particular that after running DPC, $w_{1 \rightarrow 2}$ and $w_{2 \rightarrow 1}$ are labelled by the shortest paths between vertices 1 and 2.

The idea behind our algorithm, dubbed “Snowball”, is that we grow a clique of computed distances, one vertex at a time, starting with the clique consisting of just vertices 1 and 2. When adding vertex $k$ to the clique, we compute the distance to (from) each vertex $i < k$. We are then ensured by DPC that there exists a shortest path to (from) $i$ that has an edge $\{k,j\}$ for some $j < k$ as its first (last) edge. This means that the algorithm only needs to look “down” at lower-numbered vertices. We then prove the following:

\textbf{Theorem 1.} \textit{The Snowball algorithm computes all-pairs shortest paths in $O(nm_c) \leq O(n^2w_d)$ time.}
We experimentally established the computational efficiency of our algorithm on a wide range of graphs, varying from random scale-free networks, parts of the road network of New York City, to STNs generated from HTNs, and job-shop scheduling problems. For the New York City benchmark (Figure 1a), we observe that Floyd–Warshall is very slow with its $\Theta(n^3)$ run time (and produces the expected straight line), and that Johnson and Snowball are each significantly faster than their predecessor. For graphs around 1000 vertices, Snowball is over an order of magnitude faster than Johnson. Figure 1b depicts results on STNs based on Hierarchical Task Networks (HTNs). In these graphs, constraints may usually occur only between parent tasks and their children, and between sibling tasks, but we consider an extension that includes landmark variables that mimic synchronisation between tasks in different parts of the network, and thereby cause some deviation from the tree-like HTN structure. Here also, the results indicate that for the majority of STNs stemming from HTNs, Snowball is more efficient than Johnson.

References
