Computing All-Pairs Shortest Paths by Leveraging Low Treewidth

Léon Planken and **Mathijs de Weerdt**
Delft University of Technology
{l.r.planken,m.m.deweerdt}@tudelft.nl

Roman van der Krogt
Cork Constraint Computation Centre
roman@4c.ucc.ie

Abstract

Considering directed graphs on n vertices and m edges with real (possibly negative) weights, we present two new, efficient algorithms for computing all-pairs shortest paths (APSP). These algorithms make use of directed path consistency (DPC) along a vertex ordering d. The algorithms run in $\mathcal{O}\left(n^2 w_d\right)$ time, where w_d is the graph width induced by this vertex ordering. For graphs of constant treewidth, this yields $\mathcal{O}\left(n^2\right)$ time, which is optimal. On chordal graphs, the algorithms run in $\mathcal{O}\left(nm\right)$ time. We show empirically that also in many general cases, both constructed and from realistic benchmarks, the algorithms often outperform Johnson's algorithm, which represents the current state of the art with a run time of $\mathcal{O}\left(nm + n^2 \log n\right)$. These algorithms can be used for temporal and spatial reasoning, e.g. for the Simple Temporal Problem (STP), which underlines its relevance to the planning and scheduling community.

1 Introduction

Finding shortest paths is an important and fundamental problem in communication and transportation networks, circuit design, graph analysis—e.g. for computing the betweenness (Girvan and Newman 2002)—and is a subproblem 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) (Dechter, Meiri, and Pearl 1991), 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 job-shop scheduling problems. The shortest path computations in these applications can account for a significant part of the total run time. These topics have received substantial interest in the planning and scheduling community (Satish Kumar 2005; Bresina et al. 2005; Shah and Williams 2008).

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 (Dechter et al.) is by computing all-pairs shortest paths (APSP) on its STN, e.g. with the Floyd–Warshall algorithm. The state of the art for computing APSP is Johnson's algorithm, which runs in $\mathcal{O}\left(n^2 \log n + nm\right)$ time using a Fibonacci heap (Fredman and Tarjan 1987).

Recently there has been specific interest in STNs stemming from hierarchical task networks (HTNs) (Castillo, Fdez-Olivares, and González 2006; Bui and Yorke-Smith 2010). These graphs have the "sibling-restricted" property: each task, represented by a pair of vertices, is connected only to its sibling tasks, its parent or its children. In these graphs the number of children of a task is restricted by a constant *branching factor*, and therefore the resulting STNs also have a tree-like structure.

In this paper we present two new algorithms for APSP in Section 3. One algorithm is based on a point-to-point shortest path algorithm by Chleq (1995), and another is similar to Planken et al.'s (2008) algorithm for enforcing partial path consistency (P^{3}C). These algorithms advance the state of the art in computing APSP. In graphs of constant treewidth, such as the sibling-restricted STNs based on HTNs, the run time is bounded by $\mathcal{O}\left(n^2\right)$, which is optimal since the output is $\Theta(n^2)$. In addition to these STNs, examples of such graphs are outerplanar graphs, graphs of bounded bandwidth, graphs of bounded cutwidth, and series-parallel graphs.

When the algorithms are applied to chordal graphs, they have a run time of $\mathcal{O}\left(nm\right)$, which is a strict improvement over the state of the art (Chaudhuri and Zaroliagis 2000, with a run time of $\mathcal{O}\left(nmw_d^2\right)$; w_d is defined below). Chordal graphs are an important subset of general sparse graphs: interval graphs, trees, k-trees, directed path, and split graphs are all special cases of chordal graphs (Golumbic 2004). Moreover, any graph can be made chordal using a so-called triangulation algorithm. Such an algorithm operates by eliminating vertices one by one, connecting the neighbours of each eliminated vertex and thereby inducing cliques in the graph. The *induced width* w_d of the elimination ordering d is defined to be equal to the cardinality of the largest set of neighbours thus connected. The upper bound of the run time of both proposed algorithms on these general graphs, then, is $\mathcal{O}\left(n^2 w_d\right)$; this is better than the bound on Johnson's algorithm if $w_d \in \mathcal{O}\left(\log n\right)$.

Finding an elimination ordering of minimum induced width is an NP-hard problem. This minimum induced width is the tree-likeness property of the graph mentioned above,

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Algorithm 1: DPC (Dechter, Meiri, and Pearl 1991)

Input: A weighted directed graph $G = \langle V, E \rangle$ and a vertex ordering $d : V \to \{1, \ldots, n\}$
Output: CONSISTENT if DPC could be enforced on G; INCONSISTENT if a negative cycle was found

```
1 for k ← n to 1 do
2     forall i < j < k such that {i,k}, {j,k} ∈ E do
3         w_{i→j} ← min{w_{i→j}, w_{i→k} + w_{k→j}}
4         w_{j→i} ← min{w_{j→i}, w_{j→k} + w_{k→i}}
5         E ← E ∪ {{i,j}}
6         if w_{i→j} + w_{j→i} < 0 then
7             return INCONSISTENT
8 return CONSISTENT
```

i.e. the *treewidth*, denoted w^*. In contrast, the induced width is not a direct measure of the input (graph), so the bound of $\mathcal{O}\left(n^2 w_d\right)$ is not quite proper. Therefore, we experimentally establish the computational efficiency of the proposed algorithms in Section 4 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. However, we start by introducing the concepts of induced width, treewidth, chordal graphs and triangulation in more detail.

2 Preliminaries

Both algorithms for all-pairs shortest paths presented in this paper rely on the fact that the graph has already been made directionally path-consistent along a certain vertex ordering. In this section, we therefore briefly discuss the algorithm for directed path consistency (DPC) and how to find a vertex ordering required for DPC.

Directed Path Consistency

Dechter, Meiri, and Pearl (1991) proposed DPC, included here as Algorithm 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 $G = \langle V, E \rangle$ and a vertex ordering d, which is a bijection between V and the natural numbers $\{1, \ldots, n\}$. In this paper, we simply represent the ith vertex in such an ordering as the natural number i. The weight on the arc from i to j is represented as $w_{i \to j}$; further, our shorthand for the existence of an arc between these vertices, in either direction, is $\{i, j\} \in E$. Finally, we denote by G_k the graph induced on vertices $\{1, \ldots, k\}$, so $G_n = G$.

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. Moreover, assuming $i < j$ and given a path π between such a pair of neighbours that except for its endpoints lies outside G_i, a defining property of DPC is that it ensures that $w_{i \to j}$ is no higher than the total weight of this path. This implies in particular that after running DPC, $w_{1 \to 2}$ and $w_{2 \to 1}$ are labelled by the shortest paths between vertices 1 and 2.

The run time of DPC depends on a measure w_d called the *induced width* relative to the ordering d of the vertices. This induced width is exactly the highest number of neighbours $j < k$ encountered during the DPC algorithm. It is not a property of the graph that forms DPC's input per se; rather, it is dependent on both the graph and the vertex ordering used. With a careful implementation, DPC's time bound is $\mathcal{O}\left(n w_d^2\right)$ if this ordering is known beforehand.

The edges added by DPC are called *fill edges* and make the graph *chordal* (sometimes also called triangulated). Indeed, DPC differs from a triangulation procedure only by its manipulation of the arc weights. In a chordal graph, every cycle of length four or more has an edge joining two vertices that are not adjacent in the cycle. By definition of the induced width, the number of edges in such a chordal graph, denoted by $m_c \geq m$, is $\mathcal{O}\left(n w_d\right)$.

Finding a Vertex Ordering

In principle, DPC can use any vertex ordering to make the graph both chordal and directionally path-consistent. However, since the vertex ordering defines the induced width, it directly influences the run time and the number of edges m_c in the resulting graph. As mentioned above, finding an ordering of minimum induced width and determining the treewidth is an NP-hard problem in general. Still, the class of constant-treewidth graphs can be optimally triangulated in $\mathcal{O}(n)$ time (Bodlaender 1993); and if G is already chordal, we can find a *perfect elimination ordering* (resulting in no fill edges) in $\mathcal{O}(m)$ time, using *maximal cardinality search* (MCS) (Tarjan and Yannakakis 1984).

For general graphs, various heuristics exist that often produce good results. We mention here the minimum degree heuristic (Rose 1972), which in each iteration chooses a vertex of lowest degree. Since the ordering produced by this heuristic is not fully known before DPC starts but depends on the fill edges added, an adjacency-list-based implementation will require another $\mathcal{O}(\log n)$ factor in DPC's time bound. However, for our purposes in this article, we can afford the comfort of maintaining an adjacency matrix, which yields bounds of $\mathcal{O}\left(n^2 + n w_d^2\right)$ time and $\mathcal{O}\left(n^2\right)$ space.

3 All-Pairs Shortest Paths

Even though, to the best of our knowledge, a DPC-based APSP algorithm has not yet been proposed, algorithms for computing single-source shortest paths (SSSP) based on DPC can be obtained from known results in a relatively straightforward manner. Chleq (1995) proposed a point-to-point shortest path algorithm that with a trivial adaptation computes SSSP; Planken, de Weerdt, and Yorke-Smith (2010) implicitly also compute SSSP as part of their IPPC algorithm. These algorithms run in $\mathcal{O}(m_c)$ time and thus can simply be run once for each vertex to yield an APSP algorithm with $\mathcal{O}(n m_c) \subseteq \mathcal{O}\left(n^2 w_d\right)$ time complexity. Below, we first show how to adapt Chleq's algorithm to compute APSP; then, we present a new, efficient algorithm named Snowball that relates to Planken et al.'s (2008) $\mathrm{P}^3\mathrm{C}$.

Chleq's Approach

Chleq's point-to-point shortest path algorithm, simply called "Min-path" and reproduced here as Algorithm 2, runs in

Algorithm 2: Min-path (Chleq 1995)

Input: Weighted directed DPC graph $G = \langle V, E \rangle$; (source, destination) pair (s, t)
Output: Distance from s to t

```
1 ∀i ∈ V : D[i] ← ∞
2 D[s] ← 0
3 for k ← s to 1 do
4     forall j < k such that {j, k} ∈ E do
5         D[j] ← min{D[j], D[k] + w_{k→j}}
6 for k ← 1 to t do
7     forall j > k such that {j, k} ∈ E do
8         D[j] ← min{D[j], D[k] + w_{k→j}}
9 return D[t]
```

Algorithm 3: Chleq-APSP

Input: Weighted directed DPC graph $G = \langle V, E \rangle$
Output: Distance matrix D

```
1 for i ← 1 to n do
2     D[i][*] ← Min-paths(G, i)
3 return D
```

$\mathcal{O}(m_c)$ time because each edge is considered exactly twice. This algorithm can be used for SSSP as well within the same time bounds by setting $t = n$ and returning the entire array D instead of just $D[t]$. Algorithm 3 then calls this SSSP algorithm (Min-paths) n times to compute APSP.

The Snowball Algorithm

In this section, we present an algorithm that computes APSP (or full path-consistency), dubbed Snowball, that has the same worst-case time bounds but is more efficient about it.

The idea behind the algorithm is that we grow a clique of computed distances, one vertex at a time, starting with the trivial clique consisting of just vertex 1. 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.

The name of our algorithm derives from its "snowball effect": the clique of computed distances grows quadratically during the course of its operation. Let us now formally prove the algorithm's soundness and complexity.

Algorithm 4: Snowball

Input: Weighted directed DPC graph $G = \langle V, A \rangle$
Output: Distance matrix D

```
1 ∀i, j ∈ V : D[i][j] ← ∞
2 ∀i ∈ V : D[i][i] ← 0
3 for k ← 1 to n do
4     forall j < k such that {j, k} ∈ E do
5         forall i ∈ {1, ..., k − 1} do
6             D[i][k] ← min{D[i][k], D[i][j] + w_{j→k}}
7             D[k][i] ← min{D[k][i], w_{k→j} + D[j][i]}
8 return D
```

Theorem 1. *The Snowball algorithm computes all-pairs shortest paths in* $\mathcal{O}(nm_c) \subseteq \mathcal{O}(n^2 w_d)$ *time.*

Proof. The proof is by induction. After enforcing DPC, $w_{1\to 2}$ and $w_{2\to 1}$ are labelled by the shortest path between vertices 1 and 2. For $k = 2$ and $i = j = 1$, the algorithm then sets $D[1][2]$ and $D[2][1]$ to the correct values.

Now, assume that $D[i][j]$ is set correctly for all vertices $i, j < k$. Let $\pi : i = v_0 \to v_1 \to \cdots \to v_{\ell-1} \to v_\ell = k$ be a shortest path from i to k, and let $h_{\max} = \max\{h \mid v_h \in \pi\}$. By DPC, if $h_{\max} > k$, there exists a path of the same weight where a shortcut $v_{h_{\max}-1} \to v_{h_{\max}+1}$ is taken. This argument can be repeated to conclude that there must exist a shortest path π' from i to k that lies completely in G_k and, except for the last arc, in G_{k-1}. Thus, by the induction hypothesis and the observation that the algorithm considers all arcs from the subgraph G_{k-1} to k, $D[i][k]$ is set to the correct value. An analogous argument holds for $D[k][i]$.

With regard to the algorithm's time complexity, note that the two outermost loops together result in each of the m_c edges in the chordal graph being visited exactly once. The inner loop always has fewer than n iterations, yielding a run time of $\mathcal{O}(nm_c)$ time. From the observation above that $m_c \leq nw_d$, we can also state a looser time bound of $\mathcal{O}(n^2 w_d)$. □

We now briefly discuss the consequences for two special cases: graphs of constant treewidth and chordal graphs. For chordal graphs, we can just substitute m for m_c; further, as described above, a perfect elimination ordering exists and can be found in $\mathcal{O}(m)$ time. This gives the total run time of $\mathcal{O}(nm)$. We also stated above that for a graph of constant treewidth w^*, a vertex ordering with $w_d = w^*$ can be found in $\mathcal{O}(n)$ time. Then, omitting the constant factor w_d, the algorithm runs in $\mathcal{O}(n^2)$ time. This also follows from the algorithm's pseudocode by noting that every vertex k has a constant number (at most w^*) of neighbours $j < k$.

We note here the similarity between Snowball and the P^{3}C algorithm by Planken, de Weerdt, and van der Krogt (2008). Like P^{3}C, Snowball operates by enforcing DPC, followed by a single backward sweep along the elimination ordering. P^{3}C then computes shortest paths only for the arcs present in the chordal graph, in $\mathcal{O}(nw_d^2)$ time.

4 Experiments

We evaluate the two algorithms together with efficient implementations of Floyd–Warshall and Johnson's algorithm with a Fibonacci heap[1] across six different benchmark sets.[2]

The properties of the test cases are summarised in Table 1. This table lists the number of test cases, the range of the

[1] For Johnson's algorithm we used the corrected Fibonacci heap implementation by Fiedler (2008), since the widely used pseudocode of Cormen et al. (2001) contains mistakes.

[2] Available at
http://dx.doi.org/10.4121/uuid:698db457-499f-48a1-bb26-5a54070b4dbe

Table 1: Properties of the benchmark sets

type	#cases	n	m	w_d
Chordal				
– Figure 1	400	200	985–19,900	5–199
– Figure 2	130	214–3,125	22,788–637,009	211
Scale-free				
– Figure 3	190	150	296–2,240	14–103
– Figure 4	426	100–200	460–891	38–58
Diamonds	504	51–379	49–379	2
New York	180	108–4,882	113–8,108	2–51
Job-shop	600	5–241	8–3,840	3–62
HTN	121	500–625	748–1,515	2–144

Table 2: The total induced width, triangulation, and total run time of Snowball over all experiments on general graphs show that the minimum degree heuristic is the best choice.

heuristic	w_d	time (s)	Snowball (s)
min-fill	26127	1410	1539
min-degree	26673	141	270
static min-fill	32085	168	365
static min-degree	32363	156	356
MCS	33127	189	438
random	47966	285	675

number of vertices n, edges m, and the induced width w_d produced by the minimum degree heuristic. More details on the different sets can be found below. Each STP instance is ensured to be consistent by modifying the constraint weights while leaving its structure intact.

All algorithms were implemented in Java.[3] The experiments were run using Java 1.6 in server mode, on Intel Xeon E5430 CPUs. All times we report are the measured CPU times. For Chleq and Snowball, this includes the time that was spent triangulating the graphs. Each instance was run ten times, which was averaged to obtain the run time for that particular instance. The only exception to this is the New York set: because of the long run times on the larger of these graphs, we only averaged over two runs for each instance.

Triangulation As discussed in Section 2, finding an optimal vertex ordering (with minimum induced width) is NP-hard, but several efficient triangulation heuristics for this problem exist. We ran our experiments with six different heuristics: the minimum fill and minimum degree heuristics, static variants of both (taking into account only the original graph), an MCS ordering on the original graph, and a random ordering. All of these, except minimum fill, have time complexities within the bound on the run time of Chleq and Snowball. We found that the minimum degree heuristic gave on average an induced width only 2% higher than that found by minimum fill, but with drastically lower run time. All other heuristics, though themselves faster, yielded an induced width at least 20% higher, resulting in a longer total triangulation time and at least a 30% longer total run time of Snowball (see the summary of the results over all benchmarks given in Table 2). This confirms earlier work on finding good heuristics (Kjærulff 1990). In the experimental results included below we therefore only show the results based on the minimum degree heuristic.

[3]Our implementations are available in binary form at
`http://dx.doi.org/10.4121/uuid:177969ea-0ca3-4090-a363-1fe37cd35017`

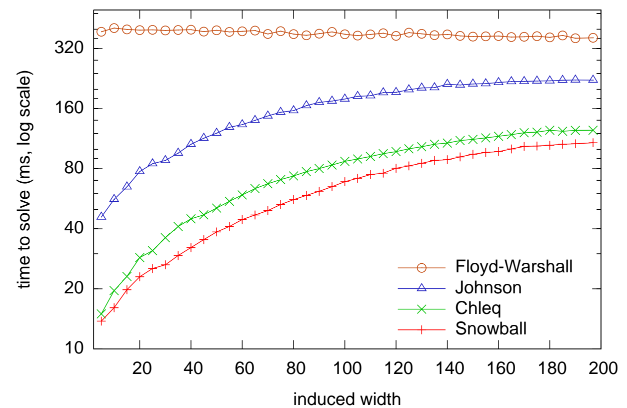

Figure 1: Run times on generated chordal graphs of a fixed size 200 and varying treewidth.

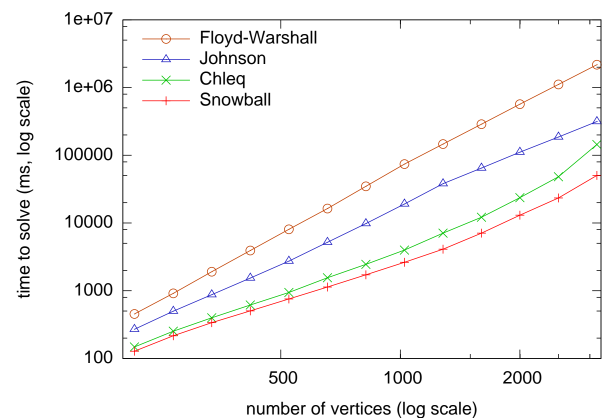

Figure 2: Run times on generated chordal graphs of a fixed treewidth of 211.

Chordal Graphs To evaluate the performance of the new algorithms on chordal graphs, we construct chordal graphs of a fixed size of 200 with a treewidth ranging from five up to the number of vertices, thus yielding a complete graph at the high end (see Figure 1). Overall, and especially for sparse graphs, the run time of the new algorithms is well below that of Johnson. Figure 2 shows the run times on chordal graphs of a constant treewidth and increasing size. Here, the two new algorithms outperform Johnson by nearly an order of magnitude (a factor 9.3 for Snowball around $n = 1300$), confirming the expectations based on the theoretical upper bounds.

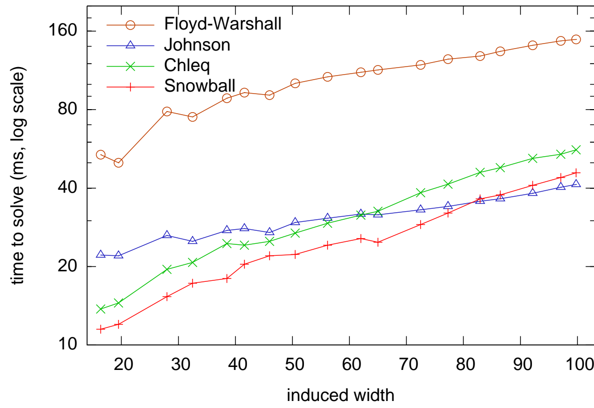

Figure 3: Run times on the scale-free benchmarks for graphs of 150 vertices and varying induced width.

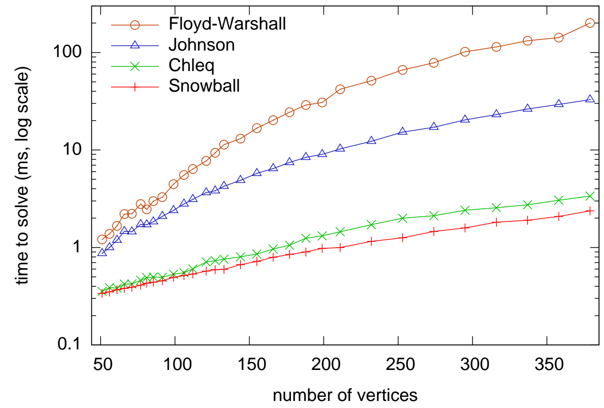

Figure 5: Run times on the diamonds benchmarks for graphs of varying size.

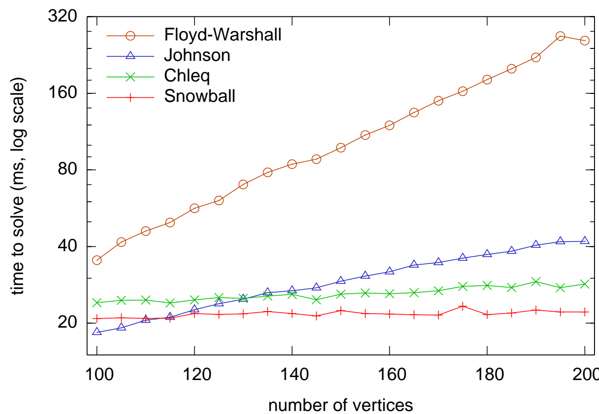

Figure 4: Run times on the scale-free benchmarks for graphs of induced widths 38 to 58 and varying sizes.

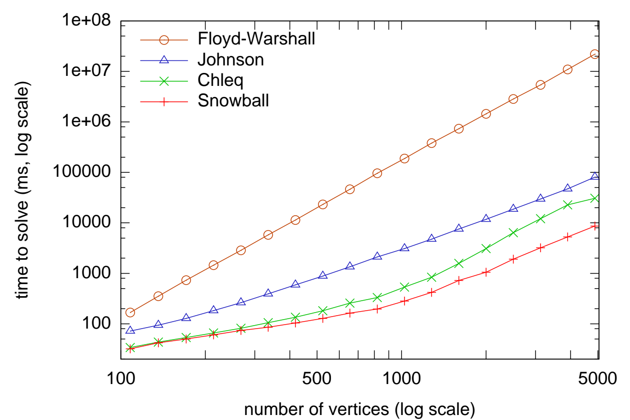

Figure 6: Run times on the New York benchmarks for subgraphs of varying sizes.

General Graphs

For general graphs, we expect from the theoretical analysis that Chleq and Snowball are faster than Johnson when the induced width is low, and that Johnson is faster on sparse graphs of a large induced width. The main question is at which induced width this changeover occurs.

Scale-Free Graphs To see at which induced width Johnson is faster, we compare the run times on graphs of size 150 and an induced width from 14 to 103 (more than half the number of vertices). The number of edges varied from 296 to 2240 along these runs. The results of this experiment can be found in Figure 3. Here we see that up to an induced width about 80, Snowball is the most efficient, and for higher induced widths Johnson is fastest. A consistent observation but from a different angle can be made from Figure 4, where the induced width is between 38 and 58, the number of edges is between 460 and 891 and the size of the graph is varied from 100 to 200. Here we see that for small graphs up to 110 vertices, Johnson's algorithm is the fastest, but after that Snowball overtakes it (this holds for all results up to a very sparse graph of almost 700 vertices).

STNs from Diamonds The "diamond" graphs used in the this benchmark set, defined by Strichman, Seshia, and Bryant (2002), feature two parallel paths of equal lengths starting from a single node, and ending in another single node. From the latter node, two paths start again, to converge on a third node. This pattern is repeated a number of times (depending on the size of the graph). The final node is then connected to the very first one. Their main property is that they are very sparse. We looked at 504 graphs, ranging in size from 51 to 379 vertices, with an induced width of 2. The induced width is clearly extremely small, which translates into Chleq and Snowball being considerably faster than Johnson, as evidenced by Figure 5.

Selections from New York Road Network More interesting than the artificially constructed graphs are graphs based on real networks, for which shortest path calculations are relevant. The first of this series is based on the road network of New York City, which we obtained from the

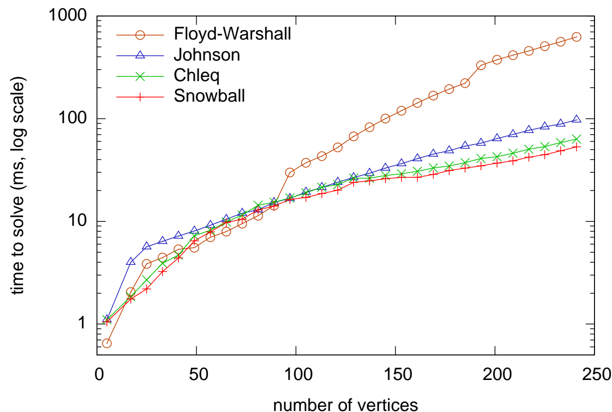

Figure 7: Run times on the job-shop benchmarks for graphs of varying size.

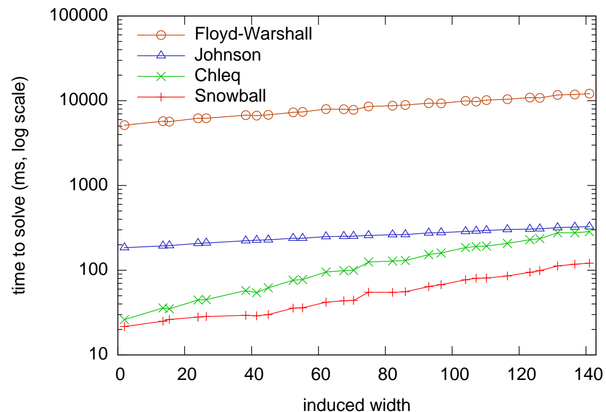

Figure 8: Run times on the HTN benchmarks for graphs from 500 to 625 vertices and varying induced width.

DIMACS challenge website.[4] This network is very large (with 264,346 vertices and 733,846 edges) so we obtained several subgraphs of varying sizes using a simple breadth-first search from a random starting location. The results of all algorithms on these subgraphs can be found in Figure 6. Here we observe the same ranking of the algorithms as on the chordal graphs of a fixed treewidth and for diamonds: Floyd–Warshall is very slow with its $\Theta(n^3)$ run time (and produces the expected straight line), then for Johnson, Chleq, and Snowball, each is significantly faster than its predecessor. This can be explained by considering the induced width of these graphs. Even for the largest graphs the induced width is around 30, which is considerably smaller than the size of the graph.

Note that these graphs are planar and tailored algorithms, mentioned below in Section 5, exist for this class. We feel, however, that our results are still of interest.

STNs from Job-Shop Scheduling Each of the 600 instances in the job-shop set is generated from a job-shop instance randomly drawn from the job-shop problems in the SMT-LIB (Ranise and Tinelli 2003). The most striking observation for the experiments on these STNs is that the difference between Johnson and the two new algorithms is not significant for the problem sizes in the benchmark set. The run times for Floyd–Warshall are better for some instances with up to 100 variables, while for larger graphs the other algorithms are significantly faster.

STNs from HTNs Finally, we consider a benchmark set whose instances imitate so-called sibling-restricted STNs originating from Hierarchical Task Networks. This set is therefore particularly interesting from a planning point of view. In these graphs, constraints may occur only between parent tasks and their children, and between sibling tasks (Bui and Yorke-Smith 2010). We consider an extension that includes *landmark variables* (Castillo, Fdez-Olivares, and González 2002) that mimic synchronisation between tasks in different parts of the network, and thereby cause some deviation from the tree-like HTN structure. We generate HTNs using the following parameters: (i) the number of tasks in the initial HTN tree (fixed at 250; note that tasks have a start and end point), (ii) the branching factor, determining the number of children for each node (between 4 and 7), (iii) the depth of the HTN tree (between 3 and 7), (iv) the ratio of landmark time points to the number of tasks in the HTN (randomly drawn from $[0, 0.5]$), and (v) the probability of constraints between siblings (when possible; randomly drawn from $[0, 0.5]$).

These settings result in graphs of between 500 and 625 vertices, with induced widths varying between 2 and 144. Figure 8 shows the results of these experiments as a function of the induced widths of the graphs. We can see that only for the larger induced widths, Johnson and Chleq come close. These large induced widths are only found for high landmark ratios of 0.5. The results indicate that for the majority of STNs stemming from HTNs, Snowball is more efficient than Johnson.

Induced Width on General Graphs

On general graphs, the run time of the proposed algorithms depends on the induced width of the ordering produced in the triangulation. In this experiment we analyze the induced width for all problem instances in the previous experiments and compare it to a lower bound $x \leq w^*$ on the treewidth. Neither for the used triangulation heuristic (minimum degree) nor for the lower bound we are aware of any theoretical guarantee on the quality with respect to the treewidth, but we can calculate the ratio w_d/x to get an upper bound on the ratio w_d/w^*.

These results can be found in Figure 9. The curves represent functions $w_d(x) = cx^k$ that express the relation of the induced width to the lower bound x. We find for New York $k = 4.55$, for HTN $k = 2.52$, for scale-free $k = 1.24$, and for job shop $k = 1.05$ with very small constants $c < 0.05$ for New York and HTN and c around 1 for the other two. Since the x in these functions is (just) a lower bound on the treewidth w^*, we tentatively conclude that in most settings

[4] http://www.dis.uniroma1.it/~challenge9/

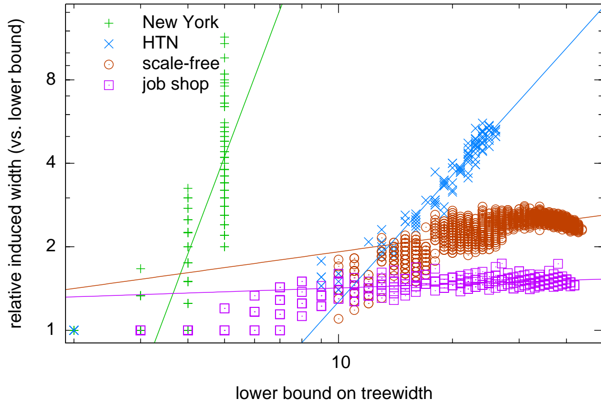

Figure 9: An upper bound on the induced width can be determined experimentally by comparing it to a lower bound on the treewidth.

$w_d(x)$ is $\mathcal{O}\left(w^{*2.5}\right)$ and the run time of the algorithms thus is $\mathcal{O}\left(n^2 w^{*2.5}\right)$.

An alternative to a triangulation heuristic would be to use an approximation algorithm with a bound on the induced width that can be theoretically determined. For example, Bouchitté et al. (2004) give a $\mathcal{O}(\log w^*)$ approximation of the treewidth w^*. Using such an approximation would give an upper bound on the run time of Snowball of $\mathcal{O}\left(n^2 w^* \log w^*\right)$. However, the run time of obtaining this approximate induced width is $\mathcal{O}\left(n^3 \log^4 n w^{*5} \log w^*\right)$ and has a high constant as well, so their work is—for now—mainly of theoretical value.

5 Related Work

For dense, directed graphs with real weights, the state-of-the-art APSP algorithms run in $\mathcal{O}\left(n^3/\log n\right)$ time (Chan 2008; Han 2008). These represent a serious improvement over the $\mathcal{O}\left(n^3\right)$ bound on Floyd–Warshall, but do not profit from the fact that in most graphs that occur in practice, the number of edges m is significantly lower than n^2.

This profit is exactly what algorithms for sparse graphs aim to achieve. Recently, an improvement was published over Johnson's $\mathcal{O}\left(nm + n^2 \log n\right)$ algorithm: an algorithm for sparse directed graphs running in $\mathcal{O}\left(nm + n^2 \log\log n\right)$ time (Pettie 2004). With an efficient implementation, this algorithm is thus faster than Johnson (in worst cases, for large graphs) when $m \in o(n \log n)$. The upper bound of $\mathcal{O}\left(n^2 w_d\right)$ on the run time of Snowball is smaller than this established upper bound when the induced width is small (i.e. when $w_d \in \mathcal{O}(\log\log n)$), and, of course, for chordal graphs and graphs of constant treewidth.

We are familiar with one earlier work to obtain shortest paths by leveraging low treewidth. Chaudhuri and Zaroliagis (2000) present an algorithm for answering single-source shortest path (SSSP) queries with preprocessing $\mathcal{O}\left(w_d^3 n \log n\right)$ and query time $\mathcal{O}\left(w_d^3\right)$. A direct extension of their results to APSP would imply a run time of $\mathcal{O}\left(n^2 w_d^3\right)$ on general graphs and $\mathcal{O}\left(nmw_d^2\right)$ on chordal graphs. Our result of computing APSP on general graphs in $\mathcal{O}\left(n^2 w_d\right)$ and in $\mathcal{O}(nm)$ on chordal graphs is thus a strict improvement.

Other restrictions on the set of graphs for which shortest paths are computed lead to other algorithms, sometimes with tighter bounds. For example, for unweighted chordal graphs, APSP lengths can be determined in $\mathcal{O}\left(n^2\right)$ time, but only after for each vertex edges are added to all neighbours of its neighbours (denoted by G^2) (Han, Sekharan, and Sridhar 1997). Considering only planar graphs, recent work shows that APSP be found in $\mathcal{O}\left(n^2 \log^2 n\right)$ (Klein, Mozes, and Weimann 2010).

In the context of planning and scheduling, a number of similar APSP problems need to be computed sequentially, potentially allowing for a more efficient approach using dynamic algorithms. Even and Gazit (1985) provide a method where addition of a single edge can require $\mathcal{O}\left(n^2\right)$ steps, and deletion $\mathcal{O}\left(n^4/m\right)$ *on average*. Demetrescu and Italiano (2006) and Thorup (2004) later give an alternative approach with an amortized run time of $\mathcal{O}\left(n^2(\log n + \log^2 \frac{n+m}{n})\right)$.

6 Conclusions and Future Work

In this paper we give two algorithms for all-pairs shortest paths with a run time bounded by (i) $\mathcal{O}\left(n^2\right)$ for all graphs of constant treewidth, matching earlier results that also gave $\mathcal{O}\left(n^2\right)$ (Chaudhuri and Zaroliagis 2000); (ii) $\mathcal{O}(nm)$ on chordal graphs, improving over the earlier $\mathcal{O}\left(nmw_d^2\right)$; and (iii) $\mathcal{O}\left(n^2 w_d\right)$ on general graphs, showing again an improvement over previously known tightest bound of $\mathcal{O}\left(n^2 w_d^3\right)$. In these bounds, w_d is the induced width of the ordering used; experimentally we have tentatively determined this to be bounded by the treewidth to the power 2.5. In addition, we provided extensive experiments showing that Snowball is faster than Chleq, and both consistently outperform Johnson and Floyd–Warshall in most settings (e.g. for STNs stemming from HTNs of size around 600 when the induced width is below 140).

The similarity to $\mathrm{P}^3\mathrm{C}$ suggests an improvement to Snowball for a $\mathcal{O}\left(nw_d^2 + n^2 s\right)$ run time, where s is the size of the largest minimum separator in the chordal graph. This improvement is of importance when separators are small while the treewidth may not be. HTN-based sibling-restricted STNs, for instance, always have separator size $s = 2$. Letting the branching factor rise as high as $\mathcal{O}(\sqrt{n})$, the improved Snowball algorithm would still have an optimal $\mathcal{O}\left(n^2\right)$ time complexity. We plan to implement this improvement in the near future.

In other future work, we would like to also experimentally compare our algorithms to the recent algorithms by Pettie (2004) and the algorithms for graphs of constant treewidth by Chaudhuri and Zaroliagis (2000). In addition, we are interested in more efficient triangulation heuristics, or triangulation heuristics with a guaranteed quality, to be able to give a guaranteed theoretical bound on general graphs. Another direction, especially interesting in the context of planning and scheduling, is to use the ideas presented here to design

a faster algorithm for dynamic APSP, for example, building upon work by Demetrescu and Italiano (2006) and Planken, de Weerdt, and Yorke-Smith (2010).

Acknowledgments

Roman van der Krogt is supported by Science Foundation Ireland under Grant number 08/RFP/CMS1711.

References

Bodlaender, H. L. 1993. A Linear Time Algorithm for Finding Tree-decompositions of Small Treewidth. In *Proc. of the 25th ACM Symposium on the Theory of Computing*, 226–234.

Bouchitté, V.; Kratsch, D.; Müller, H.; and Todinca, I. 2004. On Treewidth Approximations. *Discrete Applied Mathematics* 136(2-3):183–196.

Bresina, J. L.; Jónsson, A. K.; Morris, P. H.; and Rajan, K. 2005. Activity Planning for the Mars Exploration Rovers. In *Proc. of the 15th Int. Conf. on Automated Planning and Scheduling*, 40–49.

Bui, H. H., and Yorke-Smith, N. 2010. Efficient Variable Elimination for Semi-Structured Simple Temporal Networks with Continuous Domains. *Knowledge Engineering Review* 25(3):337–351.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2002. A Temporal Constraint Network Based Temporal Planner. In *Proc. of the 21st Workshop of the UK Planning and Scheduling Special Interest Group*, 99–109.

Castillo, L.; Fdez-Olivares, J.; and González, A. 2006. Efficiently Handling Temporal Knowledge in an HTN planner. In *Proc. of the 16th Int. Conf. on Automated Planning and Scheduling*, 63–72.

Chan, T. 2008. All-Pairs Shortest Paths with Real Weights in $\mathcal{O}\left(n^3/\log n\right)$ Time. *Algorithmica* 50:236–243.

Chaudhuri, S., and Zaroliagis, C. D. 2000. Shortest Paths in Digraphs of Small Treewidth. Part I: Sequential Algorithms. *Algorithmica* 27(3):212–226.

Chleq, N. 1995. Efficient Algorithms for Networks of Quantitative Temporal Constraints. In *Proc. of the 1st Int. Workshop on Constraint Based Reasoning*, 40–45.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms, 2nd edition*. MIT Press.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49(1–3):61–95.

Demetrescu, C., and Italiano, G. F. 2006. Fully Dynamic All-Pairs Shortest Paths with Real Edge Weights. *Journal of Computer and System Sciences* 72(5):813–837.

Even, S., and Gazit, H. 1985. Updating Distances in Dynamic Graphs. *Methods of Operations Research* 49:371–387.

Fiedler, N. 2008. Analysis of Java implementations of Fibonacci Heap. `http://tinyurl.com/fibo-heap`.

Fredman, M., and Tarjan, R. E. 1987. Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms. *Journal of the ACM* 34(3):596–615.

Girvan, M., and Newman, M. E. J. 2002. Community Structure in Social and Biological Networks. *Proc. of the National Academy of Sciences of the USA* 99(12):7821–7826.

Golumbic, M. 2004. *Algorithmic Graph Theory and Perfect Graphs*. Elsevier.

Han, K.; Sekharan, C. N.; and Sridhar, R. 1997. Unified All-Pairs Shortest Path Algorithms in the Chordal Hierarchy. *Discrete Applied Mathematics* 77(1):59–71.

Han, Y. 2008. A Note of an $\mathcal{O}\left(n^3/\log n\right)$-time Algorithm for All-Pairs Shortest Paths. *Information Processing Letters* 105(3):114–116.

Kjærulff, U. 1990. Triangulation of Graphs - Algorithms Giving Small Total State Space. Technical report, Aalborg University.

Klein, P. N.; Mozes, S.; and Weimann, O. 2010. Shortest Paths in Directed Planar Graphs with Negative Lengths: A Linear-space $\mathcal{O}\left(n\log^2 n\right)$-time Algorithm. *ACM Transactions on Algorithms* 6(2):1–18.

Pettie, S. 2004. A New Approach to All-pairs Shortest Paths on Real-weighted Graphs. *Theoretical Computer Science* 312(1):47–74.

Planken, L. R.; de Weerdt, M. M.; and van der Krogt, R. P. J. 2008. P^{3}C: A New Algorithm for the Simple Temporal Problem. In *Proc. of the 18th Int. Conf. on Automated Planning and Scheduling*, 256–263.

Planken, L.; de Weerdt, M.; and Yorke-Smith, N. 2010. Incremental Partial Path Consistency for STP. In *Proc. of the 20th Int. Conf. on Automated Planning and Scheduling*, 129–136.

Ranise, S., and Tinelli, C. 2003. The SMT-LIB Format: An Initial Proposal. In *Proc. of Pragmatics of Decision Procedures in Automated Reasoning*.

Rose, D. J. 1972. A Graph-Theoretic Study of the Numerical Solution of Sparse Positive Definite Systems of Linear Equations. In Read, R., ed., *Graph theory and computing*. Academic Press. 183–217.

Satish Kumar, T. K. 2005. On the Tractability of Restricted Disjunctive Temporal Problems. In *Proc. of the 15th Int. Conf. on Automated Planning and Scheduling*, 110–119.

Shah, J. A., and Williams, B. C. 2008. Fast Dynamic Scheduling of Disjunctive Temporal Constraint Networks through Incremental Compilation. In *Proc. of the 18th Int. Conf. on Automated Planning and Scheduling*, 322–329.

Strichman, O.; Seshia, S. A.; and Bryant, R. E. 2002. Deciding Separation Formulas with SAT. In *Proc. of the 14th Int. Conf. on Computer Aided Verification*, volume 2404 of *LNCS*, 209–222. Springer.

Tarjan, R. E., and Yannakakis, M. 1984. Simple Linear-time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal on Computing* 13(3):566–579.

Thorup, M. 2004. Fully-dynamic All-Pairs Shortest Paths: Faster and Allowing Negative Cycles. In *Algorithm Theory*, volume 3111 of *LNCS*, 384–396. Springer.