Chapter 6

Conclusions

6.1 Summary of the main results

We presented two new transitive closure algorithms that are based on Tarjan's strong component algorithm. The first new algorithm, called CR_TC, constructs the successor sets during the strong component detection. The second new algorithm, called STACK_TC, constructs a successor set of a strong component immediately after detecting the component. Both algorithms construct the successor sets from strong components, not from vertices. An improvement to previous transitive closure algorithms that are based on strong component detection is that the new algorithms scan the input graph only once without generating a partial successor set for each vertex. CR_TC has a smaller worst-case execution time than the previous algorithms that construct the successor sets during strong component detection [36, 40, 64]. STACK_TC has a smaller worst-case execution time than any previous transitive closure algorithm that is based on strong component detection except Munro's algorithm [91], which has a smaller worst-case execution time when the inputs are dense. However, Munro's algorithm is impractical, since it has high constant costs, and its worst-case execution time with sparse inputs is large.

We showed how unnecessary stack operations can be avoided in Tarjan's strong component algorithm.

We presented two new transitive closure representations. The first representation is based on intervals of integers. Each component is represented as its reverse topological number, and a successor set is represented as a collection of intervals of consecutive component numbers. A successor set is represented very compactly compared to traditional set representations such as lists or trees. The representation generalizes the method for compressing the transitive closure of an acyclic graph presented by Agrawal et al. [4]. The new representation allows cyclic inputs, and it can be computed during a single depth-first traversal, whereas the representation by Agrawal et al. requires many traversals of the input graph.

We presented another new representation that is based on the chain decomposition of an acyclic input graph by Simon [112]. This new representation also allows cyclic inputs and it can be computed during a single depth-first traversal whereas Simon's representation requires several passes over the input graph.

We showed how the new representations help avoiding redundancy caused by multiple paths between pairs of vertices in a graph.

We described simulation experiments in which we studied the average-case behavior of

transitive closure algorithms and representations. In the first experiment, we studied the average size of representations. The interval representation required a much smaller memory space than traditional set representations. The average size of the interval representation was linear to the number of vertices in random graph model $G(n, p, l)$. It was also linear in model $G(n, p)$ except when the expected outdegree was slightly greater than one. In this case, the average size was about $0.55n \log n$. The chain representation did not save any space compared to a list representation containing strong components.

In the second experiment, we studied the average complexity of constructing the successor sets with different algorithms and representations. STACK_TC required the least number of accesses of basic elements of the representation both in the list representation and in the interval representation. With the interval representation, the average number of accesses in CR_TC and in STACK_TC was linear to the number of vertices except in random graph model $G(n, p)$ when the expected outdegree was slightly greater than one. In this case, the average number of accesses was quadratic to the number of vertices. Since a basic element access takes a constant time, and since the other operations in CR_TC and in STACK_TC always take a time linear to the size of the graph, this indicates that the new algorithms are usually linear to the size of the input when the interval representation is used.

In the third experiment, we compared the execution times of Schmitz's algorithm [105], algorithm CR_TC, and two versions of algorithm STACK_TC. STACK_TC was the fastest algorithm. It was about two to three times faster than Schmitz's algorithm when the expected outdegree was greater than three. CR_TC was slower than the other algorithms when the expected outdegree was below three, but became faster than Schmitz's algorithm in greater expected outdegrees.

These results satisfy rather well two of our initial goals that we listed in the introduction. Algorithm STACK_TC is more efficient than the previous algorithms. The interval representation is usually linear to the number of vertices, and the execution times of STACK_TC and CR_TC with interval representation are usually linear to the size of the input graph.

6.2 Further research

The third of our goals was that the methods that we develop can be used efficiently both in a main memory environment and in a secondary memory environment. However, in this thesis, we only studied the behavior in a main memory environment. We shall study further the behavior of our new transitive closure algorithms and representations in a secondary memory environment. Hirvisalo [50] adapted algorithm STACK_TC to a disk environment and compared it with algorithm BTC by Ioannidis et al. [64]. The preliminary results indicate that STACK_TC with the interval representation requires less disk I/O than BTC. It seems that since the interval representation is very compact, the main cost in a disk environment is related to the traversal of the input graph and not to the successor set creation. Thus, we must design methods for speeding up graph traversal in a disk environment.

We shall study other problems that are related to the full transitive closure problem, e.g., partial transitive closure problems and generalized transitive closure problems. Some techniques that we used in the transitive closure problem can also be used in these other problems, but new approaches are also required. For instance, the interval representation cannot be used

to store path information compactly.

We also want to develop better methods for algorithm performance study by simulation. Hirvisalo [50] developed a set of tools for analyzing the disk behavior of transitive closure and other related algorithms. These tools will be developed further.

In the simulation experiments, we used two models of random graphs $G(n,p)$ and $G(n,p,l)$ to generate the inputs. We shall try to find other useful models of random graphs, and study the behavior of transitive closure algorithms and representations in these models. Some new models of random graphs could, for instance, mimic the inputs occuring in some specific application areas of transitive closure computation.

Algorithm performance study by simulation has its pros and cons. A weakness is that it requires much time; another weakness is that the results are valid only in those parts of the space of possible inputs that the simulations cover. Thus, we shall also try to do mathematical average case analysis when it seems to be possible.