Abstract: Nontraditional applications of database systems require the efficient evaluation of recursive queries. The transitive closure of a binary relation has been identified as an important and frequently occurring special case. Traditional algorithms for computing the transitive closure, as developed in the field of algorithmic graph theory, hold both the operand relation and the result relation within directly addressable main memory. The newly anticipated applications, however, deal with very large relations that do not fit into main memory and therefore must be blockwise paged to and from secondary storage. Thus we have to design algorithms and optimization methods for computing the transitive closure of very large relations. We survey and compare various such algorithms and methods in a unifying manner. In particular we identify eight basic strategies to generate and to refine transitive closure algorithms: algebraic manipulation, implementation of the join operator, reusage of newly generated tuples, enforcement of some ordering of tuples, blocking of adjacency lists, tuning and preprocessing, taking advantage of topological order, and selection of an access structure for adjacency lists. The analysis demonstrates the great variety of options on the different description levels and how they are compatible. Based on experiments some specific algorithms are recommended.

*) Research was done as a part of the ESPRIT-Project 311 "Advanced Data and Knowledge Management System", supported by the European Community under a subcontract with Nixdorf Computer Corporation.
1 Introduction

Recently it has become obvious that the facilities of today's database systems are too restricted in many cases. Non-traditional application areas as PROLOG, expert systems, and knowledge bases require the efficient evaluation of recursive queries ([Ag 87], [BiRaSt 87], [Da 87], [JaAgNe 87], [Ro 86], [VaBo 86]). In this context a special recursive query, the transitive closure of a binary relation, has attracted a lot of attention ([AgJa 87], [Io 86], [Lu 87], [LuMiRi 87], [VaBo 86]). It is easy to define, and many general recursive queries can be expressed in a language containing the operators of the relational algebra plus an operator for the transitive closure.

On the one side evaluation procedures have been developed on the level of the relational algebra. These include e.g. the NAIV, SEMINAIV, and LOGARITHMIC methods. On the other side there exist promising algorithms for computing the transitive closure in the field of graph theory. Because they have been developed as main memory algorithms and work over boolean matrices a reformulation is necessary before utilizing them in the context of database systems. This group contains e.g. the algorithms of Warshall, Warren, Bloniarz/Fischer/Meyer, and Goralcikova/Koubek ([Wars 62], [Warr 75], [BIFiMe 76], [GoKo 79]). In the course of reformulating these methods the level of the relational algebra turns out to be too high. Instead the algorithms must be tailored to the data structures which store the relations.

Transitive closure algorithms can be categorized into iterative methods (depending on the length of the underlying graph) and direct methods ([AgJa 87]). The algorithms of the first group are all iterative, whereas those of the second group are all direct.

The main goals of our paper are:
- The description of all algorithms on the same level.
- The categorization of the algorithms with respect to some basic generation strategies. We start from the most simple algorithm and develop all the other algorithms by applying the generation strategies.
- The simplification of the comparison of algorithms developed in different contexts. Especially the fact is revealed that in general direct algorithms are superior to iterative ones.
- The summary of further optimization ideas collected from different fields.

We have identified the following set of basic generation strategies.

GEN 1) Selection of a method on the algebraic level (for computing the transitive closure).