RelView – An OBDD-Based Computer Algebra System for Relations

Rudolf Berghammer and Frank Neumann

Institut für Informatik und Praktische Mathematik,
Christian-Albrechts-Universität Kiel,
Olshausenstraße 40, D-24098 Kiel

Abstract. We present an OBDD-based Computer Algebra system for relational algebra, called RelView. After a short introduction to the OBDD-implementation of relations and the system, we exhibit its application by presenting two typical examples.

1 Introduction

Since many years relational algebra has widely been used by mathematicians and computer scientists as a convenient means for problem solving; see e.g., [12,3]. One main reason for the use of relational algebra is that it has a fixed and surprisingly small set of operations all of which can efficiently and with reasonable effort be implemented on finite carrier sets using, e.g., Boolean arrays, linked lists, or ordered binary decision diagrams (OBDDs). Thus, a Computer Algebra system for relational algebra can be implemented with reasonable effort, too. In this paper, we want to give an impression of such a system, called RelView, which has been developed at Kiel University since 1993 and is available free of charge, see http://www.informatik.uni-kiel.de/~progsys/relview.html.

RelView can be used to solve many different tasks while working with relational algebra, relation-based discrete structures, and relational programs. E.g., it can support relation-algebraic reasoning. In this field typical applications are the search for counter-examples or the detection of new properties. For these activities, "playing" and "experimenting" with relation-algebraic expressions is essential and this is one of the purpose RelView has been designed for. In particular, the interactive nature of the system allows to add, change and remove relations and their representations, and makes it possible to invoke computations at every time within a working session. A further application domain is programming. This not only concerns the implementation of relational algorithms using the system's programming language, but also typical tasks appearing in (formal) program development like specification testing, detection of loop invariants and other important properties necessary for correctness proofs, rapid prototyping, and improving efficiency. Third, the advantages of the system when using it in teaching and for visualization respectively animation should be mentioned. We have recognized that visualizing advanced concepts of relational algebra (like relational domain constructions) in RelView is very helpful. Furthermore, since

RelView allows computations not only to be executed fully-automatically but also in a stepwise fashion, it is a very good means to demonstrate how a certain algorithm works. Finally, it should be mentioned that we found it very attractive to use RelView for producing good examples in teaching, which frequently have been proven to be the key of fully understanding a concept. Of course, we are not able to show all these aspects of RelView in this paper.

2 Relation-Algebraic Preliminaries

We write $R : X \leftrightarrow Y$ if R is a relation with domain X and range Y, i.e., a subset of $X \times Y$. If the sets X and Y of R's *type* $X \leftrightarrow Y$ are finite, we may consider R as a Boolean matrix. Since this Boolean matrix interpretation is well suited for many purposes and also used as one possibility of RelView to depict relations, in the following we often use matrix terminology and matrix notation. Especially, we speak about the rows and columns of R and write R_{xy} instead of $(x, y) \in R$. We assume the fundamentals of relational algebra to be known, viz. R^{T} (*transposition*), $\overline{R}$ (*complement*), $R \cup S$ (*union*), $R \cap S$ (*intersection*), RS (*composition*), $R \subseteq S$ (*inclusion*), and the special relations O (*empty relation*), L (*universal relation*), and I (*identity relation*). A relation R is *univalent* if $R^{\mathsf{T}}R \subseteq \mathsf{I}$, *total* if $R\mathsf{L} = \mathsf{L}$, *injective* if R^{T} is univalent, *reflexive* if $\mathsf{I} \subseteq R$, *antisymmetric* if $R \cap R^{\mathsf{T}} \subseteq \mathsf{I}$, *transitive* if $RR \subseteq R$, *symmetric* if $R = R^{\mathsf{T}}$, and *irreflexive* if $R \subseteq \overline{\mathsf{I}}$. A univalent and total relation is a *mapping*, a reflexive, antisymmetric, and transitive relation is a *partial order*, and a reflexive and transitive relation is a *quasi order*.

By $\mathrm{syq}(R, S) := \overline{R^{\mathsf{T}}\overline{S}} \cap \overline{\overline{R}^{\mathsf{T}} S}$ the *symmetric quotient* $\mathrm{syq}(R, S) : Y \leftrightarrow Z$ of two relations $R : X \leftrightarrow Y$ and $S : X \leftrightarrow Z$ is defined. Many properties of this construct can be found in [12]. In this paper we only need the equivalence

$$\mathrm{syq}(R, S)_{yz} \iff \forall\, x : R_{xy} \leftrightarrow S_{xz}\,. \tag{1}$$

Relational algebra provides some possibilities for modeling sets. The first modeling which we will apply in this paper uses *vectors*. These are relations v with $v = v\mathsf{L}$. For v being of type $X \leftrightarrow Y$ this condition means: Whatever set Z and universal relation $\mathsf{L} : Y \leftrightarrow Z$ we choose, an element $x \in X$ is in relationship $(v\mathsf{L})_{xz}$ either to none element $z \in Z$ or to every element $z \in Z$. Since for a vector the range is irrelevant, in the following we consider mostly vectors $v : X \leftrightarrow \mathbf{1}$ with a specific singleton set $\mathbf{1} = \{\perp\}$ as range and omit in such cases the second subscript, i.e., write v_x instead of $v_{x\perp}$. Such a vector can be considered as a Boolean matrix with exactly one column, i.e., as a Boolean column vector, and *represents* the subset $\{x \in X \mid v_x\}$ of its domain X.

We will also use injective mappings for modeling subsets. Given an injective mapping $\imath : Y \leftrightarrow X$, we may consider Y as a subset of X by identifying it with its image under $\imath$. If Y is actually a subset of X and $\imath$ is the identity mapping from Y to X, then the vector $\imath^{\mathsf{T}}\mathsf{L} : X \leftrightarrow \mathbf{1}$ represents Y as subset of X in the sense above. Clearly, also the transition in the other direction is possible, i.e., the generation of an injective mapping

$$\mathrm{inj}(v) : Y \leftrightarrow X \qquad \mathrm{inj}(v)_{yx} :\Longleftrightarrow y = x \tag{2}$$

from a given vector $v : X \leftrightarrow \mathbf{1}$ representing the subset Y of X. We call $\mathrm{inj}(v)$ the *injective mapping generated* by the vector v.

As a third possibility to model subsets of a given set X we will use the set-theoretic *membership relation* defined by (2^X is the power set of X)

$$\mathsf{M} : X \leftrightarrow 2^X \qquad \mathsf{M}_{xY} :\Longleftrightarrow x \in Y\,. \tag{3}$$

Using matrix terminology, a combination of injective mappings and membership relations leads to a *column-wise representation* of sets of subsets. More specifically, if the vector $v : 2^X \leftrightarrow \mathbf{1}$ represents a subset $\mathfrak{S}$ of 2^X in the sense above, then for all $x \in X$ and $Y \in \mathfrak{S}$ we get the equivalence of $x \in Y$ and $(\mathsf{M}\,\mathrm{inj}(v)^{\mathsf{T}})_{xY}$ due to (2) and (3). This means that the elements of $\mathfrak{S}$ are represented precisely by the columns of the relation $C := \mathsf{M}\,\mathrm{inj}(v)^{\mathsf{T}} : X \leftrightarrow \mathfrak{S}$. A further consequence of this fact is that $\overline{C^{\mathsf{T}}\overline{C}} : \mathfrak{S} \leftrightarrow \mathfrak{S}$ is the relation-algebraic specification of set inclusion on $\mathfrak{S}$, that is for all $Y, Z \in \mathfrak{S}$ we have that $(\overline{C^{\mathsf{T}}\overline{C}})_{YZ}$ iff $Y \subseteq Z$.

Using some well-known correspondences between certain logical constructions and relation-algebraic operations (see e.g., [12]) it can easily be shown that if $R : X \leftrightarrow X$ is a quasi order and $v : X \leftrightarrow \mathbf{1}$ represents a subset Y of X, then the *set of greatest elements* of Y with respect to R is represented by the vector

$$max(R, v) := v \cap \overline{\overline{R}^{\mathsf{T}} v} : X \leftrightarrow \mathbf{1}\,. \tag{4}$$

In Section 5 we will apply (4) to compute maximum cliques. This means that X is the power set 2^V of the set V of vertices of the input graph g, the first argument of the relational function *max* of (4) is the *size-comparison relation*

$$\mathsf{S} : 2^V \leftrightarrow 2^V \qquad \mathsf{S}_{AB} :\Longleftrightarrow |A| \le |B|\,, \tag{5}$$

and the second argument of *max* is a vector representing the set of cliques of g.

3 Implementation of Relations Using OBDDs

Assuming the reader to be familiar with the basic facts of OBDDs (as e.g., presented in [5,13]), we sketch in the following how to implement relations with their help. For a complete description we refer to the Ph.D. theses [10,11].

OBDDs are an efficient data structure to implement very large Boolean functions. Our implementation of relation uses this fact. We will illustrate it by a small example. Assume sets $X := \{a, b, c, d\}$ and $Y := \{r, s\}$ and the relation

$$R : X \leftrightarrow Y \qquad R := \{\,(a, r), (c, r), (c, s)\,\}.$$

By means of the canonical binary encodings $c_X : X \to \mathbb{B}^2$ and $c_Y : Y \to \mathbb{B}$ of X and Y, specified by $c_X(a) = 0, 0, c_X(b) = 0, 1, c_X(c) = 1, 0, c_X(d) = 1, 1$ respectively $c_Y(r) = 0, c_Y(s) = 1$, we can define a Boolean function $f_R : \mathbb{B}^3 \to \mathbb{B}$

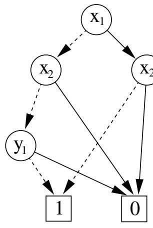


Fig. 1. OBDD for the function f_R

such that $f_R(x_1, x_2, y_1) = 1$ iff $c_X^{-1}(x_1, x_2)$ and $c_Y^{-1}(y_1)$ are defined and related via R. Then each satisfying assignment of the function f_R corresponds to a pair of the relation R and we obtain f_R in disjunctive normal form as

$$f_R(x_1, x_2, y_1) = (\overline{x_1} \wedge \overline{x_2} \wedge \overline{y_1}) \vee (x_1 \wedge \overline{x_2} \wedge \overline{y_1}) \vee (x_1 \wedge \overline{x_2} \wedge y_1) .$$

Here, for instance, the first clause $\overline{x_1} \wedge \overline{x_2} \wedge \overline{y_1}$ expresses the fact that the two elements $c_X^{-1}(0, 0) = a$ and $c_Y^{-1}(0) = r$ are related via the relation R. If we use the fixed variable ordering $x_1 < x_2 < y_1$, then we get the OBDD for the function f_R shown in Figure 1, where a continuous edge means that the value of its source is 1 and a dotted edge means that this value is 0.

The implementation of relations in the shown way leads to the problem that one OBDD can implement several relations. For instance, it is easy to check that the Boolean function obtained from the relation

$$S : Y \leftrightarrow X \qquad\qquad S := \{ (r, a), (s, a), (s, b) \}$$

coincides with f_R. This means that R and S are implemented by the same OBDD. However, the problem can be solved by additionally storing the sizes of the carrier sets since R is the only relation of type $X \leftrightarrow Y$ which leads to f_R.

As general technique, using the two canonical binary encodings $c_X : X \to \mathbb{B}^m$ and $c_Y : Y \to \mathbb{B}^n$ (where $m = \lceil \log |X| \rceil$ and $n = \lceil \log |Y| \rceil$) a relation $R : X \leftrightarrow Y$ is implemented by the two sizes $|X|$ and $|Y|$ and the OBDD of the Boolean function $f_R : \mathbb{B}^{m+n} \to \mathbb{B}$, such that $f_R(x_1, \ldots, x_m, y_1, \ldots, y_n) = 1$ iff the two decodings $c_X^{-1}(x_1, \ldots, x_m)$ and $c_Y^{-1}(y_1, \ldots, y_n)$ are defined and related via the relation R and the variable ordering is $x_1 < \ldots < x_m < y_1 < \ldots < y_n$.

Based on this implementation of relations, in the course of the Ph.D. theses [10,11] many relation-algebraic operations (including, of course, those of Section 2) have been implemented as operations on OBDDs. Due to lack of space we can not go into detail. However, it should be emphasized that the specific variable ordering with the variables encoding the elements of the domain followed by those encoding the elements of the range allows a very efficient implementation of membership relations and size-comparison relations.

In the case of the membership relation $\mathsf{M} : X \leftrightarrow 2^X$ of (3) we obtain a Boolean function $f_{\mathsf{M}} : \mathbb{B}^{m+n} \to \mathbb{B}$ with $m \leq \log |X| + 1$ variables for encoding the elements of the domain X and $n = |X|$ variables for encoding the elements of the range 2^X. It is defined by

$$f_{\mathsf{M}}(x_1, \ldots x_m, y_1, \ldots, y_n) = \begin{cases} y_{c+1} & : \quad c \leq n-1 \\ 0 & : \quad \text{otherwise,} \end{cases}$$

where $c := c_X^{-1}(x_1, \ldots, x_m)$ is the decoding of the bitstring $x_1, \ldots, x_m$.

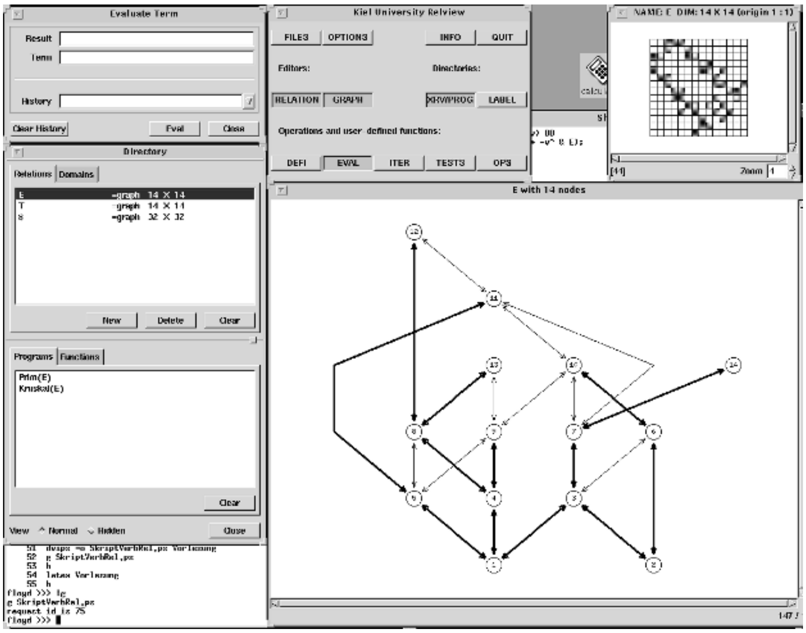
Given a variable ordering such that the variables $x_1, \ldots, x_m$ are tested before the variables $y_1, \ldots, y_n$, the number of nodes of the OBDD of f_{M} with respect to this variable ordering is bounded by $2^m - 1 + n + 2$, that is, by $3|X| + 1$. The argument is that the x-variables are tested in a binary decision tree with at most $2^m - 1$ inner nodes and at most 2^m leaves. (The case that $|X|$ is a power of 2 leads to a complete binary decision tree with $2^m - 1$ inner nodes and 2^m leaves.) Each leaf is replaced by a test of the corresponding y-variables if the number of the binary representation is not greater than $n - 1$. All other leaves are deleted and the incoming edges are directed to the sink 0. There may be reduction rules applicable to this OBDD but this only reduces the number of nodes. The result follows by adding 2 nodes for the sinks.

In the case that $|X|$ is a power of 2, the Boolean function f_{M} of the membership relation $\mathsf{M} : X \leftrightarrow 2^X$ is the well-known *direct storage access function* DSA_n (see e.g., [13]). Here m x-variables address $n = 2^m$ y-variables which decide about the output. For DSA_n the fraction of variable orderings leading to a non-polynomial OBDD size converges to 1, and this also holds for the function f_{M}. A lot of RelView programs work with membership relations, especially if one deals with hard problems or uses the system for supporting the engineering and validation of relational specifications. Therefore, the choice of our specific variable ordering is justified. The large number of bad variable orderings also suggests not to change the variable ordering during the computation process when working with this relation. This has been confirmed by experimental studies showing that the use of different variable orderings and reordering techniques leads in most cases to a much higher computation time in comparison to a computation with our fixed variable ordering.

For the size-comparision relation $\mathsf{S} : 2^X \leftrightarrow 2^X$ of (5) an OBDD-implementation has been developed in [11] which exactly uses $2 + |X|(|X| + 1)$ OBDD-nodes for the proposed variable ordering.

4 The Computer Algebra System RelView

RelView (see [1,2]) is a totally interactive and completely graphic-oriented "specific purpose" Computer Algebra system for dealing with relational algebra. In it all data are represented as (of course, finite) relations. Especially when applied for prototyping or to solve hard problems via enumeration, the system often works on very large objects since, e.g., a membership relation appears during a computation. Therefore, it uses a very efficient internal implementation of relations via OBDDs following the approach sketched in Section 3. Externally, relations are visualized in two different ways; see the snapshot of Figure 2. For relations of type $X \leftrightarrow X$ RelView offers a representation as directed graphs, including sophisticated algorithms for drawing them nicely. Alternatively, arbitrary relations may be depicted as Boolean matrices. This second representation



```
     51  dvips -o SkriptVerbRel.ps Vorlesung
     52  g SkriptVerbRel.ps
     53  h
     54  latex Vorlesung
     55  h
floyd >>> !g
g SkriptVerbRel.ps
request id is 75
floyd >>> █
```

Fig. 2. The screen of RelView

is very useful for visually editing and also for discovering various structural properties that are not evident from a representation of relations as directed graphs. Although the detailed appearance of the windows of RelView depends on specific conceptions of the users, typically the main windows (viz. evaluation window, menu window, directory window, window of the relation editor, window of the graph editor) look as in the screen snapshot of Figure 2.

The main purpose of RelView is the evaluation of relation-algebraic expressions which are constructed from the relations of the workspace using many predefined operations (like `-`, `^`, `&`, `|`, and `*` for complement, transposition, intersection, union, and composition) and tests (like `empty` for testing emptiness) on them, user-defined relational functions, and user-defined relational programs. A relational function is of the form F`(`$X_1, \ldots, X_n$`) =` t, where F is the function name, the X_i, $1 \leq i \leq n$, are the formal parameters (standing for relations), and t is a relation-algebraic expression over the relations of the workspace that can additionally contain the formal parameters X_i. A relational program essentially is a while-program based on the datatype of relations. It starts with a head line containing the program's name and a list of formal parameters. Then the declaration part follows. The third part is its body, a sequence of statements which are separated by semicolons and terminated by the return-clause.

5 A Graph-Theoretic Application: Maximum Cliques

Throughout this section, we fix an undirected graph $g = (V, E)$ and assume that it is represented by the symmetric and irreflexive *adjacency relation* $R : V \leftrightarrow V$.

That is, we suppose for all $x, y \in V$ that R_{xy} iff $\{x, y\}$ is an edge from E. A *clique* of g is a subset C of V in which every pair x, y of distinct vertices is connected by an edge, i.e., $\{x, y\} \in E$. The clique C is called *maximum* if its cardinality is maximum. The maximum clique problem is NP-hard. Moreover, it is even one of the hardest problems with respect to polynomial-time approximability.

We want to compute maximum cliques of g using RelView. First we develop a vector of type $2^V \leftrightarrow \mathbf{1}$, which represents the set of all cliques of g. Let X be an arbitrary set of vertices. We decide whether X is a clique, using the predicate-logic specification of X to be a clique, the definition (3) of the membership relation $\mathsf{M} : X \leftrightarrow 2^X$, and some simple laws of predicate logic respectively well-known correspondences between logical and relation-algebraic constructions:

$$\begin{aligned}
X \text{ is a clique of } g &\iff \forall\, x, y : x \in X \land y \in X \land y \neq x \rightarrow R_{yx} \\
&\iff \forall\, x : x \in X \rightarrow (\forall\, y : y \in X \rightarrow (y = x \lor R_{yx})) \\
&\iff \forall\, x : \mathsf{M}_{xX} \rightarrow (\forall\, y : \mathsf{M}_{yX} \rightarrow (\mathsf{I} \cup R)_{yx}) \\
&\iff \forall\, x : \mathsf{M}_{xX} \rightarrow (\neg\exists\, y : \mathsf{M}_{yX} \land \overline{\mathsf{I} \cup R}_{yx}) \\
&\iff \forall\, x : \mathsf{M}_{xX} \rightarrow \neg(\mathsf{M}^{\mathsf{T}}\overline{\mathsf{I} \cup R})_{Xx} \\
&\iff \neg\exists\, x : \mathsf{M}^{\mathsf{T}}_{Xx} \land (\mathsf{M}^{\mathsf{T}}\overline{\mathsf{I} \cup R})_{Xx} \\
&\iff \neg\exists\, x : (\mathsf{M}^{\mathsf{T}} \cap \mathsf{M}^{\mathsf{T}}\overline{\mathsf{I} \cup R})_{Xx} \land \mathsf{L}_x \qquad\qquad \mathsf{L} : V \leftrightarrow \mathbf{1} \\
&\iff \overline{(\mathsf{M}^{\mathsf{T}} \cap \mathsf{M}^{\mathsf{T}}\overline{\mathsf{I} \cup R})\mathsf{L}}_X
\end{aligned}$$

Next, we remove the subscript X from the last expression of the derivation following the vector-representation of sets introduced in Section 2. To improve efficiency, after that we apply symmetry of R in combination with some well-known relation-algebraic laws to transpose only a "row vector" instead of a relation of type $V \leftrightarrow 2^V$, yielding

$$\mathit{cliques}(R) := \overline{\mathsf{L}^{\mathsf{T}}(\mathsf{M} \cap \overline{\mathsf{I} \cup R}\mathsf{M})}^{\mathsf{T}} : 2^V \leftrightarrow \mathbf{1} \tag{6}$$

as relation-algebraic specification of the vector representing the set of all cliques of g. (Using an OBDD-implementation of relations, transposition of a relation with domain or range $\mathbf{1}$ is trivial. It only means to exchange domain and range, the OBDD remains unchanged. See [11] for details.)

Now, let $\mathsf{S} : 2^V \leftrightarrow 2^V$ be the size-comparison relation on 2^V as introduced by (5). Then an application of the relational function *max* of (4) to S and the vector $\mathit{cliques}(R)$ immediately yields the following vector-representation of the set $\mathfrak{M}$ of all maximum cliques of g:

$$\mathit{maxcliques}(R) := \mathit{max}(\mathsf{S}, \mathit{cliques}(R)) : 2^V \leftrightarrow \mathbf{1}\,. \tag{7}$$

The column-wise representation of $\mathfrak{M}$ by $\mathsf{M}\,\mathrm{inj}(\mathit{maxcliques}(R))^{\mathsf{T}} : V \leftrightarrow \mathfrak{M}$ now is a direct consequence of the technique explained in Section 2.

Each of the above relation-algebraic specifications can be easily translated into the programming language of RelView. Especially, (4), (6), and (7) read as RelView-code as follows:

```
max(R,v) = -v & -(-R^ * v).
cliques(R)
  DECL M
  BEG  M = epsi(O(R))
       RETURN -(L1n(R) * (M & -refl(R) * M))^
  END.
maxcliques(R) = max(cardrel(O(R)),cliques(R)).
```

The RelView tool automatically allows to generate uniform random relations, also of a specific kind and density; see [11] for details. We have applied this feature to test the efficiency of our approach, where we used a Sun-Fire 880 workstation running Solaris 9 at 750 MHz. Figure 3 shows some experimental results for randomly generated symmetric and irreflexive adjacency relations. The number of vertices N of the random graph is listed at the x-axis and the time needed to execute `maxcliques` is listed at the y-axis. The four curves have been obtained by varying N from 50 to 500 by steps of 50 vertices and the probability of a pair $\{x, y\}$ to be an edge of a graph from 10% to 25% by steps of 5%. We performed at least 20 experiments for each N and each density and computed in all cases the arithmetic mean of the execution times.

Because of lack of memory, in the case of 25% density we have not been able to deal with $N > 300$ vertices. If, however, we restricted us to more sparse graphs, larger numbers of vertices could be treated successfully. For example, $N = 700$ and a density of 10% led to approximately 12 minutes execution time and, for the same density, $N = 1000$ led to roughly 100 minutes.

Using the same environment, we have also tested `maxcliques` on some DIMACS benchmaks (see [7]) with up to 500 vertices. The results are shown in Table 1, where the fourth column contains the sizes of the maximum cliques and the sixth column shows the computation times in seconds. Apart from the size of a maximum clique and the computation time it is very interesting to see the number of maximum cliques for the considered benchmarks. For instance, in the case of MANN_a9 there are exactly 9540 maximum cliques.

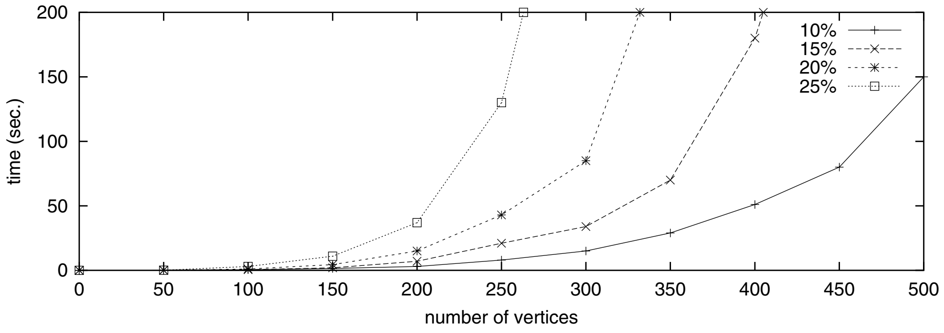


Fig. 3. Computational results on randomly generated instances

Table 1. Computational results on DIMACS Benchmarks

Problem	Vertices	Edges	Clique Size	# Max. Cliques	Time
c-fat200-1	200	1534	12	14	0.77
c-fat200-2	200	3235	24	1	0.68
c-fat500-1	500	4459	14	19	8.96
c-fat500-2	500	9139	26	19	7.92
johnson8-2-4	28	210	4	105	0.02
hamming6-4	64	704	4	240	0.81
MANN_a9	45	918	16	9540	0.30

Of course, in view of efficiency our approach cannot compete with special programs and tools for the *exact solution* of the problems we dealt with (although the complexities usually are the same). Compare, for example, our computation times with the times given in [8]. Indeed, RelView is able to compute all maximum cliques within a reasonable time in the case of sparse graphs. As mentioned before in the consideration of random instances, however, it has its difficulties if density increases. But it should be emphasized that the system yields *all solutions*. In some applications this may be helpful. For example, the enumeration of maximum independent sets can be reduced to the enumeration of maximum cliques. Using the *edge adjacency relation* construction (see [12]), thus, we are able to enumerate all maximum respectively all perfect matchings of graphs. The latter can be used to compute permanents of 0/1-matrices, since this number equals the number of perfect matchings of a specific bipartite graph.

6 A Lattice-Theoretic Application: Cut Completion

Assume (X, R) to be a partially ordered set, that is, $R : X \leftrightarrow X$ to be a partial order on X. For a set $Y \in 2^X$ let $Y^{\downarrow}$ denote the set of its lower bounds and

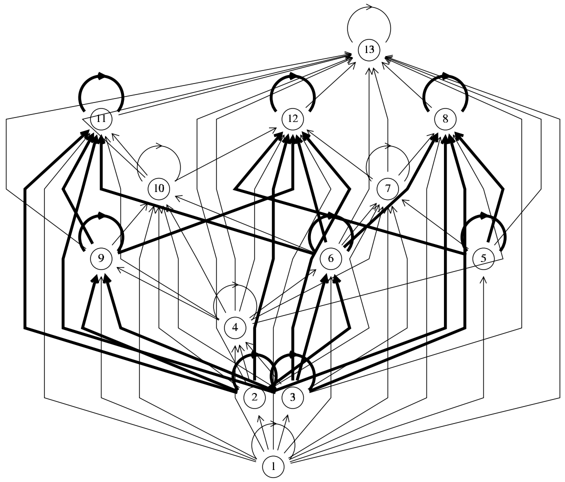


Fig. 4. Visualization of a cut completion

$Y^{\uparrow}$ denote the set of its upper bounds with respect to R. If $Y = Y^{\uparrow\downarrow}$, then this set is called a *Dedekind cut* of (X, R). Obviously, for each element $x \in X$ the set $[x] := \{y \in X \mid R_{yx}\}$ is a Dedekind cut of (X, R), called the *principal cut* generated by x. Now, let $\mathfrak{C}$ denote the set of all Dedekind cuts of (X, R) and $\mathfrak{P}$ be the subset of all principal cuts. Then $(\mathfrak{C}, \subseteq)$ is a complete lattice. It is called the *cut completion* (or Dedekind-MacNeille completion) of (X, R), since it contains $(\mathfrak{P}, \subseteq)$ as sub-order and the latter is order-isomorphic to (X, R) via the injective function $\sigma : X \to \mathfrak{C}$, mapping x to the principal cut generated by x. The lattice $(\mathfrak{C}, \subseteq)$ is the smallest complete lattice which embeds (X, R) as a sub-order; for more details on cut completion see [6].

Again we start with the vector-representation of the decisive set $\mathfrak{C}$. Assume $Y \in 2^X$. If we formalize lower and upper bounds using predicate logic and apply definition (3) of the membership relation $\mathsf{M} : X \leftrightarrow 2^X$, then we have

$$\begin{aligned} Y = Y^{\uparrow\downarrow} &\iff \forall\, x : x \in Y \leftrightarrow x \in Y^{\uparrow\downarrow} \\ &\iff \forall\, x : x \in Y \leftrightarrow (\forall\, y : y \in Y^{\uparrow} \to R_{xy}) \\ &\iff \forall\, x : x \in Y \leftrightarrow (\forall\, y : (\forall\, z : z \in Y \to R_{zy}) \to R_{xy}) \\ &\iff \forall\, x : \mathsf{M}_{x,Y} \leftrightarrow (\forall\, y : (\forall\, z : \mathsf{M}_{zY} \to R_{zy}) \to R_{xy})\,. \end{aligned}$$

Now, we apply the same strategy as in the case of cliques in combination with property (1) to replace in the last formula all logical constructions by relation-algebraic ones. Doing so, we arrive after some steps at

$$\mathit{cuts}(R) := (\mathrm{syq}(\mathsf{M}, \overline{\overline{R}\,\overline{\overline{R}^{\mathsf{T}}\mathsf{M}}}) \cap \mathsf{I})\mathsf{L} : 2^X \leftrightarrow \mathbf{1} \tag{8}$$

as relation-algebraic specification of the vector representing the set $\mathfrak{C}$ of all Dedekind cuts of (X, R), where $\mathsf{I} : 2^X \leftrightarrow 2^X$ and $\mathsf{L} : 2^X \leftrightarrow \mathbf{1}$. Using (8), the column-wise representation of the set $\mathfrak{C}$ by

$$\mathit{cutslist}(R) := \mathsf{M}\,\mathrm{inj}(\mathit{cuts}(R))^{\mathsf{T}} : X \leftrightarrow \mathfrak{C} \tag{9}$$

is an immediate consequence of the remark of Section 2. The same holds for the inclusion on the set of cuts, i.e., the partial order of the cut lattice $(\mathfrak{C}, \subseteq)$. Here we have the relation-algebraic specification

$$\mathit{cutord}(R) := \overline{\mathit{cutslist}(R)^{\mathsf{T}}\,\overline{\mathit{cutslist}(R)}} : \mathfrak{C} \leftrightarrow \mathfrak{C}\,. \tag{10}$$

Also the embedding of (X, R) into its cut completion $(\mathfrak{C}, \subseteq)$ can be formulated quite easily using relational algebra. Given $x \in X$ and $Y \in \mathfrak{C}$, we obtain

$$[x] = Y \iff \forall\, y : R_{yx} \leftrightarrow y \in Y \iff \mathrm{syq}(R, \mathit{cutlist}(R))_{xY}\,,$$

where the first step uses the definition of principal cuts and the second step applies (1) and the equivalence of $y \in Y$ and $\mathit{cutlist}(R)_{yY}$. This leads to

$$\mathit{sigma}(R) := \mathrm{syq}(R, \mathit{cutset}(R)) : X \leftrightarrow \mathfrak{C} \tag{11}$$

as relation-algebraic version of the embedding function $\sigma : X \to \mathfrak{C}$.

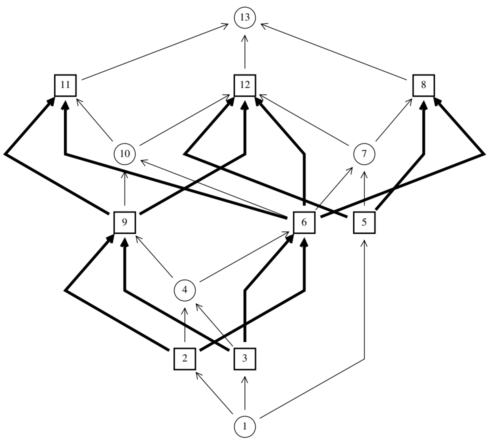


Fig. 5. Visulization of the *Hasse-diagram*

It is trivial to translate (8) until (11) into RelView-code. This allows to compute and visualize cut completions with the tool. Figure 4 graphically depicts the relation $C := \mathit{cutord}(R) : \mathfrak{C} \leftrightarrow \mathfrak{C}$ of the cut completion of an ordered set (X, R). The embedding into $(\mathfrak{C}, C)$ is visualized by boldface arrows, where the latter has been obtained by two steps. First, the sub-relation $S := \mathit{sigma}(R)^{\mathsf{T}}\, R\, \mathit{sigma}(R)$ of C has been computed. Then, in the graph-representation of C the arrows corresponding to elements of S have been marked using a specific command.

Layouts of partial orders as given in Figure 4 are neither economic nor easy to comprehend since they contain many superfluous arrows. Therefore, it is customary to depict only the Hasse-diagram as shown in Figure 5. To obtain this picture, we have computed the Hasse-diagrams H_C and H_S of C and S, respectively, via calls of a small RelView-program. Then we have marked the arrows corresponding to elements of H_S in the graph-representation of the union $H_C \cup H_S$. And, finally, we have emphasized the set of vertices represented by the vector $\mathit{sigma}(R)^{\mathsf{T}}\, \mathsf{L} : \mathfrak{C} \leftrightarrow \mathbf{1}$ by drawing these vertices as squares.

7 Conclusion

We have presented the OBDD-based specific purpose Computer Algebra system RelView for relational algebra, and have exhibited its use by two examples. The novelty and real attraction of our approach is the combination of OBDDs, relational algebra, visualization, and animation in an efficient and flexible software system. RelView uses only standard procedures for OBDD manipulation, which are available in any OBDD-package. Hence, any improvement in the OBDD area leads to a greater efficiency of RelView computations.

Experience has taught us that relational algebra is a powerful tool for dealing with many problems on discrete structures. As the examples of Section 5 and 6 show, it is often possible to "calculate" concise algorithms from formal specifications, so that correctness is established by construction. Algorithms can be executed and their results visualized with RelView. This allows to check them against the specifications and to detect errors. Due to the shortness and clearness of relation-algebraic expressions respectively RelView programs, the user can easily play and experiment with them.

RelView has been combined with other tools, e.g., SniffAlyzer for viewing and analyzing software architectures facts (see [11]). We have isolated the core functionality of RelView from the entire system and collected in a C-library, called Kure. With this library at hand, relational algebra can efficiently and easily be integrated into many other software systems. The PetRA tool for the analysis of Petri nets is a first application of this approach; see [9] for details.

References

1. R. Behnke et al., RelView – A system for calculation with relations and relational programming, in: Proc. 1. Conf. on Fundamental Approaches to Software Engineering, LNCS 1382, Springer (1998), 318-321.
2. R. Berghammer, T. Hoffmann, Modelling sequences within the RelView system, J. of Univ. Comp. Sci. 7 (2001), 107-123.
3. R. Berghammer, B. Möller, G. Struth (eds.), Proc. 7th Int. Workshop *Relational Methods in Computer Science*, LNCS 3051, Springer (2004).
4. C. Brink, W. Kahl, G. Schmidt (eds.), Relational methods in computer science, Advances in Computer Science, Springer (1997).
5. R.E. Bryant, Symbolic Boolean manipulation with ordered binary decision diagrams, ACM Com. Surv. 24 (1992), 293-318.
6. B.A. Davey, H.A. Priestley, Introduction to lattices and orders, Cambridge Univ. Press (1991).
7. DIMACS implementation challenges (second challenge, 1992-1993), Available via URL http://dimacs.rutgers.edu/Challenges/.
8. T. Fable, Simple and fast: Improving a branch-and-bound algorithm for maximum clique, in: Proc. 10. Europ. Symp. on Alg., LNCS 2461, Springer (2002), 485-498.
9. A. Fronk, Using relational algebra for the analysis of Petri nets in a CASE tool based approach, Proc. 2nd IEEE Int. Conf. on Software Engineering and Formal Methods, IEEE Press (2004), 396-405.
10. B. Leoniuk, ROBDD-based implementation of relational algebra with applications (in German), Diss., Univ. Kiel (2001).
11. U. Milanese, On the implementation of a ROBDD-based tool for the manipulation and visualization of relations (in German), Diss., Univ. Kiel (2003).
12. G. Schmidt, T. Ströhlein, Relations and graphs. Discrete mathematics for computer scientists, EATCS Monographs on Theoret. Comp. Sci., Springer (1993).
13. I. Wegener, Branching programs and binary decision diagrams: Theory and applications, SIAM Monographs on Discr. Math. and Appl., SIAM (2000).