Relational Implementation of Simple Parallel Evolutionary Algorithms

Britta Kehden, Frank Neumann, Rudolf Berghammer

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Univ. zu Kiel
Olshausenstraße 40, 24098 Kiel, Germany

Abstract. Randomized search heuristics, among them evolutionary algorithms, are applied to problems whose structure is not well understood, as well as to hard problems in combinatorial optimization to get near-optimal solutions. We present a new approach implementing simple parallel evolutionary algorithms by relational methods. Populations are represented as relations which are implicitly encoded by (reduced, ordered) binary decision diagrams. Thereby, the creation and evaluation is done in parallel, which increases efficiency considerably.

1 Introduction

In the past, randomized search heuristics (see [7] for an overview), among them *evolutionary algorithms* (abbreviated as EAs), became quite popular. Initially developed in the 1960s and 1970s, they have found many applications as simple, robust, and efficient solvers in wide areas of real world problems whose structure is not well understood. But they also have been used in the case of many hard problems in combinatorial optimization to get solutions, which do not differ too much from optimal solutions, in reasonable (for instance, polynomial) time.

Also since some time, relational algebra [10] has been successfully used for algorithm development, especially in the case of discrete structures. Relational programs are implementations of algorithms that are mainly based on a datatype for binary relations. This approach has some benefits. Such programs can be specified and developed in a rigid way, which helps to prove their correctness very formally. Since relational algebra has a fixed and surprisingly small set of base constants and operations, it also opens the possibility of computer-support. Especially, it offers to execute programs with the aid of a relation-algebraic manipulation and programming system like the Kiel RelView tool [1, 3]. This does not only allow to increase their trustworthiness. It can also be used to get a feeling of the programs' behaviour, for instance, concerning runtimes or the quality of the computed solutions if they implement search heuristics or approximation algorithms. Furthermore, since relational programs frequently are quite simple, it enables to play and experiment with them. In case of evolutionary algorithms this may help to adjust free parameters in the right way.

The aim of this paper is to show how to develop relational versions of randomized search heuristics, how to implement them as RelView-programs, and how RelView's specific representation of relations positively influences the runtimes. We concentrate on covering problems and consider minimum vertex covers on undirected graphs and minimum set covers on hypergraphs.

For the first problem in [2] a heuristics is presented. It combines a well-known approximation algorithm (in [5] attributed to Gavril and Yannakakis) with a generic program for computing a minimal subset fullfilling a certain property. Such an approach does not have the chance to escape from local optima. However, this can be different in the case of evolutionary algorithms since they, even when only based on mutation, sample a much larger neighbourhood. Of course, one has to pay for such an approach, normally with a higher runtime.

One can reduce the runtime of evolutionary algorithms if the individuals are created and evaluated in parallel. In this case one speaks of *parallel evolutionary algorithms*, or PEAs for short. Doing parallel random search, it is important how to create and evaluate individuals. Often this is done by a distributed system. The disadvantage of such an approach is that the system can get rather huge and a lot of resources will be necessary. To avoid this drawback, we will use an approach based on one single system. The key idea is to describe the process of creating and evaluating individuals by relation-algebraic expressions, which immediately can be formulated in RelView and then executed using this tool. And here is the place where the specific representation of relations in RelView comes into the play. The system represents them as (reduced, ordered) binary decision diagrams; see [4, 8, 9]. This implicitely means that the creation and evaluation of individuals is performed in parallel.

After having motivated this line of research, we present in Section 2 the algorithm to be implemented using relational methods. In Section 3, we introduce the basics of relational algebra and describe the RelView system. Section 4 presents an implementation of the parallel evolutionary algorithm in RelView. Results of our numerous tests with the tool are reported in Section 5. In Section 6, we finish with some conluding remarks.

2 The (1+λ)-EA and Covering Problems

We start with the minimum vertex cover problem. This classical optimization task has the following description. Given an undirected graph $G = (V, E)$ with vertex set $V = \{v_1, \dots, v_n\}$ and edge set $E = \{e_1, \dots, e_m\}$, the task is to find a subset of $V' \subseteq V$ with a smallest number of vertices such that each edge of G is incident to at least one vertex of V'. Computing minimum vertex covers is NP-hard and can be approximated within a worst case ratio of $2 - \frac{2 \ln \ln n}{\ln n}(1 - o(1))$ as shown in [6]. This is up to now the best known approximability result, which is achievable in polynomial time. Nevertheless, on most instances heuristics like evolutionary algorithms have a good chance to achieve better solutions than approximation algorithms. To garantee the best possible performance ratio for an evolutionary algorithm, too, one can integrate such a solution into the initial population. In addition, such a starting point ensures that one always has a vertex cover as the current solution.

In the following, we consider a variant of the well-known $(1+\lambda)$ evolutionary algorithm. It is based on one single individual x and produces independently λ children from x. The selection procedure selects an individual from the children that is not inferior to x. If there is not such an individual in the offspring population, x is chosen for the next generation. Here is our base algorithm:

Algorithm ($(1+\lambda)$-EA)

```
1. Choose $x \in \{0,1\}^k$
2. Create a population $P$ of size $\lambda$ from $x$. Each individual $y$ of $P$ is created by
   flipping each bit of $x$ with probability $1/k$.
3. Let $Q$ be the set of individuals of $P$ that are not inferior to $x$. If $Q$ is not
   emtpy, then replace $x$ by one individual of $Q$. Otherwise, do nothing.
4. Repeat Steps 2 and 3.
```

In the rest of this paper, we assume the loop to be stopped if the individual x has not been improved by an individual of P in t iterations. Creating the λ children in parallel leads to a variant, which is the perhaps simplest parallel evolutionary algorithm that can be considered.

To approximate a minimum vertex cover of $G = (V, E)$ with the aid of $(1+\lambda)$-EA, we take $k := n$ and represent subsets V' of V as bitstrings $x \in \{0,1\}^k$ such that $x_i = 1$ if and only if $v_i \in V'$. The algorithm also can be applied to approximate a solution of the NP-hard minimum hyperedge cover problem for a hypergraph $H = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and hyperedge set $E = \{h_1, \ldots, h_m\}$, where hyperedges are non-empty subsets of V. The problem asks for finding a subset $E' \subseteq E$ with a smallest number of hyperedges such that each vertex from V is contained in at least one hyperedge of E'. As we are searching for a subset of the hyperedges in the hypergraph, we use the above algorithm working on bitstrings of length $k := m$ and, consequently, flip each bit with probability $1/m$ in the mutation step.

3 Relational Preliminaries

In this section, we introduce some basics of relational algebra and consider vectors and points, which form specific classes of relations and subsequently are used for representing individuals. We also have a short look at RelView.

3.1 Relational Algebra

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 and of cardinality m and n, respectively, we may consider R as a Boolean matrix with m rows and n columns. Since this Boolean matrix interpretation is well suited for many purposes, in the following we often use matrix terminology and matrix notation. Especially, we speak of the rows, columns and entries of R and write R_{ij} instead of $(i, j) \in R$. We assume the reader to be familiar with the basic operations on relations, viz. R^{T} (*transposition*), $\overline{R}$ (*negation*), $R \cup S$ (*union*), $R \cap S$ (*intersection*), $R \cdot S$ (*composition*), $R \subseteq S$ (*inclusion*), and the special relations O (*empty relation*), L (*universal relation*), and I (*identity relation*).

If X and Y are finite, then $|R|$ denotes the cardinality of $R : X \leftrightarrow Y$. In Boolean matrix terminology, $|R|$ equals the number of true-entries.

For modeling individuals, in this paper we will use (column) *vectors*, which are relations x with $x = x \cdot \mathsf{L}$. For $x : X \leftrightarrow Y$ this condition means: whatever set Z and universal relation $\mathsf{L} : Y \leftrightarrow Z$ we choose, an element $i \in X$ is either in relationship $(x \cdot \mathsf{L})_{ij}$ to no element $j \in Z$ or to all elements $j \in Z$. As for a vector, therefore, the range is irrelevant, we consider in the following mostly vectors $x : X \leftrightarrow \mathbf{1}$ with a specific singleton set $\mathbf{1} = \{\perp\}$ as range and omit in such cases the second subscript, i.e., write x_i instead of $x_{i\perp}$. Such a vector can be considered as a bitstring if X equals an intervall $[1..k] := \{i \in \mathbb{N} : 1 \leq i \leq k\}$. Then x_i corresponds to the fact that the i-th component of the bitstring is 1. A non-empty vector x is said to be a *point* if it is injective, which relation-algebraically can be specified by $x \cdot x^{\mathsf{T}} \subseteq \mathsf{I}$. In the bitstring model a point $x : [1..k] \leftrightarrow \mathbf{1}$ is a bitstring in which exactly one component is 1.

3.2 The RelView System

RelView is a tool for calculating with relations and relational programming. It has been developed at Kiel University since 1993, in particular in the course of the Ph.D. theses [8, 9] and a series of Diploma theses. As already mentioned, the tool uses a representation of relations based on binary decision diagrams (BDDs). The latter have been shown to be very compact representations of Boolean functions (see e.g., [11]). Exactly this property is used to implement a relation $R : X \leftrightarrow Y$ via a Boolean function f_R such that R_{ij} if and only if $f_R(x, y) = 1$, where x and y are the binary representation of i and j, respectively. In doing so, the relational constants and operations can be implemented using the constant BDDs True and False and standard operations on BDDs.

The main purpose of RelView is the evaluation of relation-algebraic expressions. These are constructed from the relations of the system's workspace using pre-defined operations and tests, user-defined relational functions, and user-defined relational programs.

Relational functions are of the form $F(R_1, \ldots, R_n) = exp$, where F is the function name, the R_i, $1 \leq i \leq n$, are the formal parameters (standing for relations), and exp is a relation-algebraic expression over the relations of the system's workspace that can additionally contain the formal parameters. A relational program is much like a function procedure in the programming languages Pascal or Modula 2, except that it only uses relations as data type. It starts with the headline, i.e., the name of the program and the list of formal parameters. Then the declaration part follows, which consists of the declarations of local relational domains (direct products and sums), local relational functions, and local variables. The third part is the body, a sequence of statements, which are separated by semicolons and terminated by the `RETURN`-clause.

The choice of a point contained in a non-empty vector is fundamental in relational programming. Therefore, RelView possesses a corresponding pre-defined operations `point`. It is deterministic. The implementation uses that the system only deals with relations on finite carrier sets, which are lineareIy ordered by an internal enumeration. But RelView allows to generate random relations, too, using the pre-defined operation `random`. If it is applied to two relations $R : X \leftrightarrow Y$ and $S : U \leftrightarrow V$, then a relation $T : X \leftrightarrow Y$ is generated uniformly at random with the additional property that for all $i \in X$ and $j \in Y$ the probability of T_{ij} being true is $|S|/(|U|\,|V|)$. Especially, for S as point $p : U \leftrightarrow \mathbf{1}$ we obtain a random relation of the same type as R and $1/|U|$ as probability of a pair to be contained in it due to $|p| = 1$.

4 Relational Implementation of PEAs

Now, we demonstrate how the instances of $(1+\lambda)$-EA for minimum vertex cover and minimum hyperedge cover, respectively, can be implemented using relational methods and RelView. We start with the development of the basic modules, i.e., the parallel creation of the children of a given individual and the parallel test whether the new individuals are feasible solutions. Based on these modules and their RelView-implementations, we obtain the final RelView-programs as specific instantiations of a general program.

4.1 Relational Creation of Childrens

We assume that K denotes the intervall $[1..k]$ of the natural numbers as introduced in Section 3.1. Individuals of $(1+\lambda)$-EA are bitstrings from $\{0,1\}^k$, which we represent relation-algebraically by vectors of type $K \leftrightarrow \mathbf{1}$ as shown in Section 3.1, too. In the following, we treat the construction of a population of λ children from a given individual (current solution).

Let $x : K \leftrightarrow \mathbf{1}$ be the vector-representation of the current solution. We model the population of the λ children of x by means of a relation P such that each column of P, considered as a vector of type $K \leftrightarrow \mathbf{1}$, represents a single child of x. To create P, we first constitute a relation A consisting of λ columns equal to x. Defining L as intervall $[1..\lambda]$, this easily can be obtained as follows:

$$A : K \leftrightarrow L \qquad A = x \cdot \mathsf{L}, \text{ where } \mathsf{L} : \mathbf{1} \leftrightarrow L\,. \tag{1}$$

After that, we flip each entry in A with probability $1/k$ following the procedure of Section 3.2. Let $F : K \leftrightarrow L$ denote the relation where each entry has probability $1/k$ of being true. Then column l of P (i.e., the l-th child of x) is obtained by putting its i-th component, $1 \leq i \leq k$, to the entry A_{il} if F_{il} is false, and to the negation of the entry A_{il} in case of F_{il} being true. This precisely means that P equals the symmetric difference of A and F, in terms of relational algebra:

$$P : K \leftrightarrow L \qquad P = (A \cap \overline{F}) \cup (\overline{A} \cap F)\,. \tag{2}$$

Translated into the language of RelView, the relation P of (2) is computed if the following relational program `children` is applied to the relation A of (1).

```
children(A)
  DECL p, F
  BEG  p = point(Ln1(A));
       F = random(A,p)
       RETURN (A & -F) | (-A & F)
  END.
```

In this code, `Ln1` computes an universal vector the domain of which equals the domain of the argument and the range of which is $\mathbf{1}$. The symbols `-`, `|`, and `&` denote in RelView negation, union, and intersection, respectively.

4.2 Testing Individuals to be Vertex Covers

Having solved the task of creating λ children of x in parallel in form of a relation $P : K \leftrightarrow L$, we now tackle the problem of testing feasibility. In this subsection, we concentrate on vertex covers. Hyperedge covers are considered in Section 4.4.

It is well-known how to test a single vector to represent a vertex cover of a graph; see e.g., [10]. However, essential for a fast relational version of $(1 + \lambda)$-EA is to decide in parallel which columns / individuals of $P : K \leftrightarrow L$ represent a vertex cover of the graph G of Section 2. We solve this task by developing a vector of type $L \leftrightarrow \mathbf{1}$ such that its l-th entry is true if and only if the l-th column of P represents a vertex cover of G. Using first-order logic, the l-th entry of the vector has to be true if and only if the following formula holds:

$$\forall\, i : (\exists\, j : R_{ij} \wedge \neg P_{jl}) \rightarrow P_{il}\,. \tag{3}$$

Here i, j range over K and $R : K \leftrightarrow K$ is the *adjacency relation* of G, i.e., for all $i, j \in K$ the vertices v_i and v_j are connected via an edge if and only if R_{ij}.

Starting with (3), we can calculate as given below. In doing so, we only use simple logical and relation-algebraic laws in combination with well-known correspondences between logical and relation-algebraic constructions.

$$\begin{aligned}
\forall\, i : (\exists\, j : R_{ij} \wedge \neg P_{jl}) \rightarrow P_{il} &\iff \forall\, i : (\exists\, j : R_{ij} \wedge \overline{P}_{jl}) \rightarrow P_{il} \\
&\iff \forall\, i : (R \cdot \overline{P})_{il} \rightarrow P_{il} \\
&\iff \forall\, i : \neg(R \cdot \overline{P})_{il} \vee P_{il} \\
&\iff \forall\, i : (\overline{R \cdot \overline{P}} \cup P)_{il} \\
&\iff \neg\exists\, i : \neg(\overline{R \cdot \overline{P}} \cup P)_{il} \\
&\iff \neg\exists\, i : (R \cdot \overline{P} \cap \overline{P})_{il} \\
&\iff \neg\exists\, i : (R \cdot \overline{P} \cap \overline{P})^{\mathsf{T}}_{li} \wedge \mathsf{L}_i \\
&\iff \overline{(R \cdot \overline{P} \cap \overline{P})^{\mathsf{T}} \cdot \mathsf{L}}_l \\
&\iff \overline{\mathsf{L} \cdot (R \cdot \overline{P} \cap \overline{P})}^{\mathsf{T}}_l\,.
\end{aligned}$$

From the last expression, we get $\overline{\mathsf{L} \cdot (R \cdot \overline{P} \cap \overline{P})}^{\mathsf{T}} : L \leftrightarrow \mathbf{1}$ as relation-algebraic specification of the vector we are interested in, where $\mathsf{L} : \mathbf{1} \leftrightarrow K$ is a "row vector". An immediate consequence is the following RelView-function, which computes this vector from R and P:

```
IsVertexCover(R,P) = -(L1n(R) * (R * -P & -P))^.
```

4.3 The Relational (1+λ)-EA for Minimum Vertex Covers

In what follows, modules are independent of the type of minimization problem which we consider and can be adapted to maximization problems very fastly, too. For simplicity we assume that our algorithm should approximate a minimum vertex cover. Minimum edge covers are considered in the next subsection.

Using $\overline{\mathsf{L} \cdot (R \cdot \overline{P} \cap \overline{P})}^{\mathsf{T}}$, we can easily obtain a child of x that represents a vertex cover. But the algorithm has to select an individual of the new population, which is not inferior to x. Therefore, we introduce two additional vectors of type $L \leftrightarrow \mathbf{1}$. These will allow to get a smaller set of candidates for the next generation.

The first vector, we call it c, is motivated by the fact that children being a superset of x should not be selected for the next generation. Hence, for all $l \in L$ we define c_l to be true if and only if the following formula holds:

$$(\forall\, i : (\exists\, j : R_{ij} \wedge \neg P_{jl}) \rightarrow P_{il}) \wedge (\neg \forall\, i : A_{il} \rightarrow P_{il})\,. \tag{4}$$

In Boolean matrix notation this means that c_l is true if and only if the l-th column of P represents a vertex cover of G and does not contain the l-th column of A, i.e., the vector x. A translation of the second part of (4) into $((A \cap \overline{P})^{\mathsf{T}} \cdot \mathsf{L})_l$ is similar to the treatment of the first part in Section 4.2. Therefore, we renounce the details and show only the final relation-algebraic form of c, where $\mathsf{L} : \mathbf{1} \leftrightarrow K$:

$$c : L \leftrightarrow \mathbf{1} \qquad\qquad c = \overline{\mathsf{L} \cdot (R \cdot \overline{P} \cap \overline{P})}^{\mathsf{T}} \cap (\mathsf{L} \cdot (A \cap \overline{P}))^{\mathsf{T}}\,. \tag{5}$$

The motivation for the second vector, called d, is that for getting a fast algorithm, it is very useful to identify in addition those children of x, which are described by the vector c and are contained in x. In a component-wise description, this means for all $l \in L$ that d_l is true if and only if

$$c_l \wedge (\forall\, i : P_{il} \rightarrow A_{il}) \tag{6}$$

holds. Note that the l-th column of P is contained strictly in x if d_l is true. Again we renounce the details of a formal derivation of a relation-algebraic form of d from specification (6) and present only the final result (using $\mathsf{L} : \mathbf{1} \leftrightarrow K$):

$$d : L \leftrightarrow \mathbf{1} \qquad\qquad d = c \cap \overline{\mathsf{L} \cdot (\overline{A} \cap P)}^{\mathsf{T}}\,. \tag{7}$$

After these preparations, we are in a position to present a parallel implementation of $(1+\lambda)$-EA for the minimum vertex cover problem in RelView. The following program, called `PEA`, has four input relations: the adjacency relation $R : K \leftrightarrow K$ of G, a vector $s : K \leftrightarrow \mathbf{1}$ representing an initial vertex cover, a vector $lambda : L \leftrightarrow \mathbf{1}$ with $\lambda = |L|$ as size of the generated populations, and a vector $term$ the cardinality $|term|$ of which defines the maximal number t of non-improving iterations of the loop until the algorithm terminates.

```
PEA(R,s,lambda,term)
  DECL A, P, c, d, p, x, y, z
  BEG  x = s; z = term;
       WHILE -empty(z) DO
         z = z & -point(z); A = x * L(lambda)^; P = children(A);
         c = IsVertexCover(R,P) & (L1n(R) * (A & -P))^;
         d = c & -( L1n(R) * (-A & P))^;
         IF -empty(d) THEN
           p = point(d); x = P * p; z = term
         ELSE
           IF -empty(c) THEN
             p = point(c); y = P * p; c = c & -p;
             WHILE cardlt(x,y) & -empty(c) DO
               p = point(c); y = P * p; c = c & -p OD;
             IF cardlt(y,x) THEN x = y; z = term FI;
             IF cardeq(y,x) THEN x = y FI FI FI OD
       RETURN x
  END.
```

Using x as present solution, each iteration of the main loop of `PEA` works as follows. First, a new population is produced in form of a relation P using the specifications (1) and (2) and the relational program `children` of Section 4.1. After that, the vectors c and d are computed as described in (5) and (7), respectively. If $d \neq \mathsf{O}$, i.e., there is a real subset of the present solution x in P that is a vertex cover, one of these individuals is selected with the aid of a point and the present solution is changed accordingly. As this signifies an improvement, the number of the longest sequence of non-improving iterations of the loop is set to t. The latter is modeled by the assignment $z = \mathit{term}$. If, however, no such individual exists in P, then the individuals of P that are vertex covers and no supersets of x are checked one after another by means of an inner loop. If an individual y is discovered that is smaller than x, then y is taken as new present solution and z is changed to *term* since an improvement has been achieved. An individual y is also taken as new present solution if $|x| = |y|$. Since this case, however, does not mean an improvement, z remains unchanged.

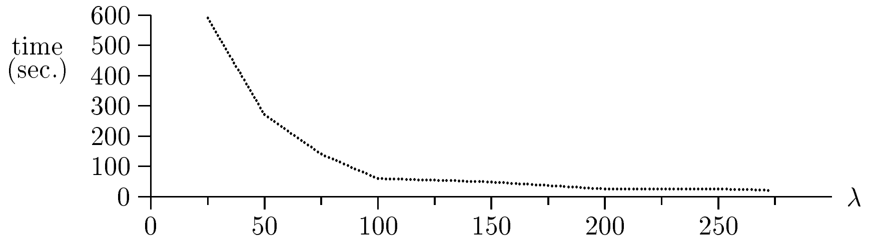
4.4 The Case of Minimum Hyperedge Covers

Now, we consider the hypergraph $H = (V, E)$ of Section 2. That is, we take $k := m$. Furthermore, we assume $R : N \leftrightarrow K$ to be the *incidence relation* of H. This means that N equals the intervall $[1..n]$ and that for all $i \in N$ and $j \in K$ vertex v_i is contained in hyperedge h_j precisely if R_{ij}.

Based on the relations $P : K \leftrightarrow L$ of (2) and R, we can derive $\overline{\mathsf{L} \cdot \overline{R \cdot P}}^{\mathsf{T}} :$ $L \leftrightarrow \mathbf{1}$ as vector the l-th component of which is true if and only if the l-th column of P is a hyperedge cover of H. A formulation as RelView-function is obvious and its use in `PEA` (instead of `IsVertexCover`) yields a parallel implementation of $(1 + \lambda)$-EA for the minimum hyperedge cover problem in RelView.

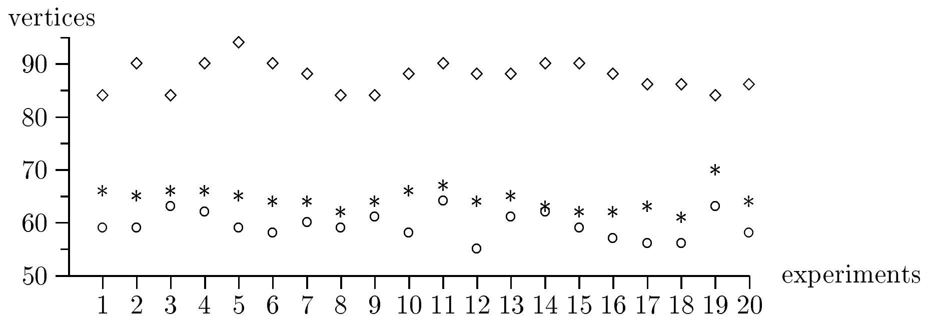
5 Experiments

We have carried out a lot of experiments with random relations of various sizes and densities on a Sun-Fire 880 workstation running Solaris 9 at 750 MHz. In each test we set the product λt to ek^2, because in the case that the current solution x is not inclusion minimal, the probability that a single child of x improves x is at least $\frac{1}{k}(1 - \frac{1}{k})^{k-1} \geq \frac{1}{ek}$. Hence, the expected number of individuals that have to be created to reach an improvement is at most ek. Using Markovs' inequality, the probability that in the run of `PEA` a inclusion minimal solution has not been produced is upper bounded by $1/k$. The figure below shows some experimental results for minimum vertex covers, graphs with $k := n = 100$ vertices and approx. 500 edges, and the set of all vertices (represented by $\mathsf{L} : V \leftrightarrow \mathbf{1}$) as initial vertex cover. In it the number of children λ is listed at the x-axis and the arithmetic mean of the execution times of 20 experiments (in seconds) at the y-axis. The curve has been obtained by varying λ from 25 to en so that, for example, $\lambda := n/2$ implies $t = 544$ (after rounding).



Apart from the execution times the pictures of all other tests look very similar to the above one. We also have experimented with $\lambda \leq en$ being a variable but t being a constant, for instance $t = n$. Especially for a small λ this leads to a much better runtime, however also to a worse result.

To get a feeling for the quality of the results, we have compared `PEA` with Gavril/Yannakakis and an instantiation of the program of [2] for vertex covers. It proved to be superior in each experiment we have performed. The following picture shows the results of 20 tests on graphs with 100 vertices and again approx. 500 edges. The size of the computed result is listed at the y-axis. The upper curve belongs to Gavril/Yannakakis, that in the middle to the instantiation of the generic program, and the lower one to our algorithm.



6 Conclusions

We have presented a new approach to implement parallel evolutionary algorithms based on relational algebra and BDDs. The relational version for the vertex cover problem is derived by formal specification of the different moduls of the PEA. Each population of children is represented as one relation which is implicitly encoded by one single BDD. Therefore, creation and evaluation of the children is done in parallel using relational operations. Our experiments for the vertex cover problem show that this approach reduces the runtime in comparision to sequentiell implementation based on a small population of children significantly. We have also shown that the relational PEA can be adapted to other problems (e.g., the minimum set cover problem in hypergraphs) very fastly.

Our future work will concentrate on applications of our approach to other problem domains. We also want to investigate parallel evolutionary algorithms based on larger parent populations as well as the implementation of other selection procedures and crossover operators.

References

1. Behnke R. et al.: RelView — A system for calculation with relations and relational programming. In: Astesiano E. (ed.): Proc. 1st Conf. *Fundamental Approaches to Software Engineering*, LNCS 1382, Springer, 318-321 (1998).
2. Berghammer R.: A generic program for minimal subsets with applications. In: Leuschel M., editor, Proc. 12th Int. Workshop *Logic-based Program Development and Transformation*, LNCS 2664, Springer, 144-157 (2003).
3. Berghammer R., Hoffmann T., Leoniuk B., Milanese U.: Prototyping and programming with relations. Electronic Notes in Theoretical Computer Science 44 (2003).
4. Berghammer R., Leoniuk B., Milanese U.: Implementation of relational algebra using binary decision diagrams. In: de Swart H. (ed.): Proc. 6th Int. Workshop *Relational Methods in Computer Science*, LNCS 2561, Springer, 241-257 (2002).
5. Cormen T.T., Leiserson C.E., Rivest R.L.: Introduction to algorithms. The MIT Press (1990).
6. Halperin, E.: Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs, Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms, ACM-SIAM (2000).
7. Hromkovic J.: Algorithms for hard problems. Introduction to combinatorial optimization, randomization, approximation, and heuristics. Springer (2001).
8. Leoniuk B.: ROBDD-based implementation of relational algebra with applications (in German). Ph.D. thesis, Inst. für Inf. und Prak. Math., Univ. Kiel (2001).
9. Milanese U.: On the implementation of a ROBDD-based tool for the manipulation and visualization of relations (in German). Ph.D. thesis, Inst. für Inf. und Prak. Math., Univ. Kiel (2003).
10. Schmidt G., Ströhlein T.: Relations and graphs. Springer (1993).
11. Wegener, I.: Branching programs and binary decision diagrams – theory and applications. SIAM Monographs on Discr. Math. and Appl. (2000).