Matching Class Diagrams: With Estimated Costs Towards the Exact Solution?

Sabrina Uhrig
Bayreuth University
Applied Computer Science I
95440 Bayreuth, Germany
sabrina.uhrig@uni-bayreuth.de

ABSTRACT

It is widely known that the general matching problem on graphs is a non-polynomial optimization problem. Thus all differencing algorithms we know of use heuristics to identify corresponding elements (e.g.[2],[6]) apart from those that rely on unique identifiers (e.g.[5],[3]). We wonder if an exact algorithm can be designed which computes a minimal cost matching between the elements of the class diagrams and which, although it shows a non-polynomial worst case behaviour, delivers its solution much faster in most cases. In this position paper we describe our ongoing work, the idea of an algorithm which works with estimated transformation costs in order to reduce the computation costs. The algorithm has not been implemented yet; it has only been manually tested on a few examples.

Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.7 [**Distribution, Maintenance, and Enhancement**]: Software EngineeringVersion Control

General Terms

Algorithms

Keywords

Differencing, UML Diagram

1. INTRODUCTION

Differences between two versions of a class diagram are needed to compare and merge class diagrams in team development processes. As there is no generally agreed definition upon the term difference it is avoided in this paper. What we would like to compute in fact is the cheapest directed delta, which is a set of cost-weighted edit operations that transform one class diagram into another. Having found such a delta, we can derive a distance and the corresponding elements.

CVSM'08, May 17, 2008, Leipzig, Germany.

As it is not easy to identify corresponding elements of different versions in such a way that a minimal delta is computed, several diff and merge tools rely on unique identifiers: When an element in a diagram is created, it is assigned a new unique identifier. When the diagram is copied, the identifiers of its elements are retained. To a great extent, the calculation of differences is for free when unique identifiers are present. Thus unique identifiers simplify algorithms and make them more efficient. However, unique identifiers make differencing and merging dependent on the history of changes. In rare cases, it might even happen that two versions are considered to have an empty intersection even though they are isomorphic. This situation occurs when both versions have been created with the same contents independently by different users. Thus we decided that our algorithm shall not rely on unique identifiers [1].

2. THE ALGORITHM

We present the idea of an exact algorithm for the computation of a cost minimal directed delta between two class diagrams. The algorithm is currently limited to the underlying simplified data model of class diagrams with a small set of feasible edit operations (move operations are not included). The focus is on finding the corresponding classes. In order to reduce the computation costs, the algorithm uses an estimated cost function to handle the dependencies between the classes which are connected by edges. It is important that the estimated costs are a lower bound for the real costs. The algorithm finds an optimal matching according to the estimated transformation costs in polynomial time and we are interested in the circumstances under which the optimality condition holds for the real costs, too.

The section is structured as follows: To specify the optimization problem, we describe the data model the algorithm operates on in 2.1, the edit operations which are feasible in 2.2 and their weighting by the cost function in 2.3. To reduce the computation costs we use estimated cost values as described in 2.4. A network graph is constructed in 2.5 that is needed to formulate and solve the optimization problem which is defined in 2.6. In section 2.7 we formulate a sufficient optimality condition. After summarizing the steps of the algorithm in 2.8 we show a simple example in section 2.9.

2.1 The underlying data model

The algorithm has been designed to operate on the simplified model of class diagrams shown in figure 1. A class diagram consists of classes, which can be connected pairwise

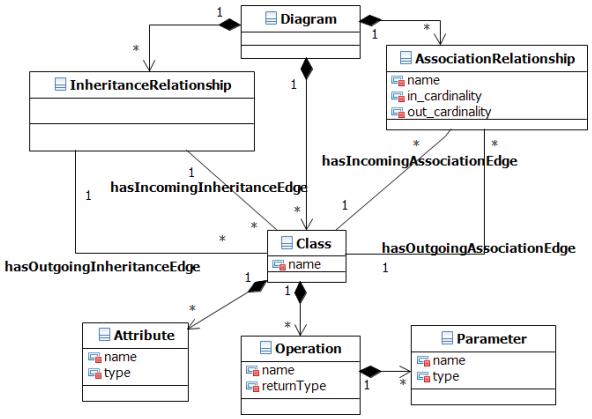

Figure 1: The underlying data model (modeled in RSA).

by association or inheritance edges. Furthermore a class can have attributes, described by name and type, and operations consisting of a name, a return type and optional ordered parameters described by name and type. Association edges have a name and cardinalities for both anchor points.

2.2 The edit operations

The set of feasible edit operations on this model is restricted to the basic operations create, delete and update on the different elements. The type of an element cannot be changed, e.g. an attribute cannot become an operation. Move operations are not supported, so if a class is matched onto another class all attributes of the first class have to be matched onto the attributes of the second class. The same applies for the operations, association and inheritance edges. For instance the feasible edit operations on association edges are: updating a cardinality value, changing the association name, deletion or creation of the edge. Due to the fact that inheritance edges have no changeable qualities, they only can be created or deleted. Moving an inheritance or association edge is no feasible edit operation in this model. A complete list of the feasible edit operations can be found in table 1.

2.3 The cost function

Let A and B be class diagrams according to the model described in section 2.1 and $a_i \in A, i = 0, \dots n$ and $b_j \in B, j = 0, \dots m$ their classes. The cost function $c(a_i, b_j)$ denotes the minimal editing costs of transforming the class a_i into b_j with respect to the feasible edit operations which have been defined in section 2.2. Deletion and creation are inverse operations and therefore assigned with equal costs, i.e. $c(a_1, \epsilon) = c(\epsilon, a_1)$. The editing distance between two identical elements being zero, the cost function fulfills $c(a_i, a_i) = 0$.
Let t_{a_i} denote the subtree rooted at a_i in the composition structure of the data model of Diagram A as shown in figure 1. In order to compute $c(a_i, b_j)$, $c(t_{a_i}, t_{b_j})$ has to be determined. So the transformation of class a_i into class b_j causes:

- the costs of name change plus
- the minimal costs for transforming the attributes of a_i into the attributes of b_j plus
- the minimal costs for transforming the operations of a_i into the operations of b_j plus
- the minimal costs for transforming the association relationship edges of a_i into the association relationship edges of b_j plus
- the minimal costs for transforming the inheritance edges of a_i into the inheritance edges of b_j.

Thus we need also cost functions for the operations on the other types according to the composition structure. For instance the transformation of an operation a_i into an operation of b_j causes:

- the costs of method name change plus
- the costs for updating the return type plus
- the costs for updating the ordered parameters.

Due to the fact that the method parameters are ordered, no further optimization problem has to be solved. The costs assigned to all feasible edit operations can be taken from table 1. Here we explain only the decisions that led to these cost values.

Every change operation, which operates on one element only, is assigned to one cost unit. Deleting an element is more expensive than changing all its qualities and thus is assigned with the cost of changing all its qualities +1. It doesn't matter how similar names are. If they are different the editing costs for transforming one name into another are 1. The costs of deleting an inheritance edge is an exception: the edge has no additional qualities but it seems to be more important and thus it costs 2 to delete or create it.

$c(A, B)$ is then defined as the minimal editing costs to transform the subtree rooted at the root element of Diagram A into the subtree rooted at the root element of Diagram B (The transformation costs of the document roots are zero). Therefore the editing distance between two diagrams is symmetric and the cost function too, i.e. $c(A, B) = c(B, A)$. The editing distance between two identical diagrams being zero, the cost function fulfills $c(A, A) = 0$.

2.4 Splitting the cost function in a sum of certain and estimated costs

If we consider matching one class with another, we can exactly compute the transformation costs for the name, the attributes and the operations, because of the restriction that these elements cannot be moved and have to be matched to the attributes and operations contained in the matched class. So we can compute the transformation costs for each class pair ($O(n^2)$) independently of how the other classes are matched. This is our exact part of the cost function $c_{certain}$.

But the costs for the transformation of inheritance and association relationship edges matching two classes cannot be determined exactly without knowing how the classes at the other side have been matched. This is the reason why this problem has non-polynomial complexity. There are $n!$ combinations for matching n to n classes, for which we have to determine the costs. To avoid this effort, we use estimated costs as second part of our cost function $c_{estimated}$ which states the costs we have at least.
To deduce the estimated costs for the transformation of the

Table 1: Feasible edit operations and their costs (c.).

edit operation	operates on elements of type	costs
name update	class	1
	attribute	
	operation	
	parameter of an operation	
	association edge	
type update	attribute	
	parameter of an operation	
	return type of an operation	
value update	cardinality of an association	
create/delete	class	1 + c. of name update + c. of deleting/creating all attributes, operations, association edges and inheritance edges
	operation	1 + c. of name update + c. of return type update + c. of deleting/creating all parameters
	attribute	1 + c. of name update + c. of type update = 3
	parameter	1 + c. of name update + c. of type update = 3
	inheritance edge	2
	association edge	1 + c. of name update + 2* (c. of cardinality update) = 4

inheritance relationship edges when matching two classes, we simply count the number of incoming and outgoing edges for both classes and compute the number of edges that have to be deleted/created:

Inheritance edges: Class a has two incoming inheritance edges and class b has one incoming and one outgoing inheritance edge. So at least one incoming edge of class a and the outgoing edge of class b have to be deleted (costs: 4). We do not know whether the incoming edge of a and one of the incoming edges of b can be matched until the classes on the other side of the edges have been matched, but certainly the real costs in the final matching are ≥ 4.

Deducing the estimated costs for the transformation of the association relationships is more complex:

Association edges: If class a has n association edges and class b m association edges we have to solve the matching problem to find the minimal cost matching for transforming $p{=}min(m,n)$ association edges into $q{=}max(m,n)$ association edges while deleting the $d{=}|n-m|$ unmatched edges. This can be solved in polynomial time.

Due to the fact that association and inheritance edges connect two classes, we cannot attribute the full costs to the class we inspect as this would result in counting the costs twice. Thus these costs have to be halved. Due to the sort of estimation we use we can be sure that the sum of the certain costs and the estimated costs is a lower bound to the real costs of the matching:

THEOREM 1.

$$c_{real}(a_i, b_j) \geq c_{certain}(a_i, b_j) + c_{estimated}(a_i, b_j)$$

$$\forall a_i \in A, b_j \in B$$

We define $c_{lb} := c_{certain}(a_i, b_j) + c_{estimated}(a_i, b_j)$.

2.5 Construction of the network graph

First, a complete bipartite graph is built, containing nodes for all classes a_i, i=0,...n of Diagram A on one side and all classes b_j, j=0,...m of Diagram B on the other side; w.l.o.g. Diagram A has as many classes as Diagram B or even more classes ($n \geq m$). The edges are directed, i.e. arrows from the elements of Diagram A to Diagram B. In order to allow for the delete operations a node ϵ with directed edges from all classes of Diagram A to ϵ is added. A source node r has to be added and connected with directed edges to all classes of A and a sink node s to which all classes of B and ϵ are connected with directed edges. At last add directed edges from ϵ to every class of B. In sum our network graph G $=(V, E)$ has a set of nodes $V =\{$r,s,ϵ, a_1,...a_n, b_1,...$b_m\}$ and the above described set of edges E: (a_i,b_j) $\forall i,j$, (r,a_i) $\forall i$, (b_j,s) $\forall j$, (a_i,ϵ) $\forall i$,(ϵ,b_j) $\forall j$ and the edge (ϵ,s).

Each directed edge is assigned with a minimum capacity (λ), a maximum capacity(κ) and costs (c). These are (0,1,0) for all directed edges which start at r. The directed edge $\epsilon \rightarrow s$ is assigned with (0,$n-m$,0). The edges $a_i \rightarrow b_j$ are assigned with (0,1,$c(a_i, b_j)$), $c(a_i, b_j)$ denotes the estimated cost function value as stated in 2.3. The edges $a_i \rightarrow \epsilon$ have the labels (0,1,$c(a_i, \epsilon)$), the edges $b_j \rightarrow s$ are labeled with (0,1,0) and the edges $\epsilon \rightarrow b_j$ are assigned with (0,1,c(ϵ, b_j)). Figure 5 shows the flow graph for the example in section 2.9.

If we send n units through r into the network graph they have to be transported to s according to the given minimal and maximal capacities on each edge. So every a_i has to be matched onto one b_j or onto ϵ, because one unit is transported into a_i. Also every b_j is matched exactly once, because only one unit can be transported from b_j to s. If this unit comes from an a_i that means class a_i is matched with class b_j. Else this unit comes from the ϵ node which means that no class of Diagram A is matched with b_j, so b_j has to be created.

2.6 The optimization problem

To get a cost minimal matching we have to transport n units from r to s with minimal costs. This can be formulated as optimization problem as follows:

OPTIMIZATION PROBLEM 1.

$$minimize \sum_{(i,j)\in E} c_{ij}\phi_{ij}$$

$$s.t. \sum_{i\rightarrow j} \phi_{ij} - \sum_{k\rightarrow i} \phi_{ki} = \begin{cases} n & \text{if i = r} \\ -n & \text{if i = s} \\ 0 & \forall\ i \in V\backslash\{r,s\} \end{cases} \quad (1)$$

$$0 \leq \phi_{ij} \leq \kappa_{ij}, \ (i,j) \in E \tag{2}$$

ϕ_{ij} denotes the number of units which are transported on the arrow $i \rightarrow j$ and c_{ij} its transport costs per unit. Condition (1) makes sure that Kirchhoff's Current Law is true: the sum of units flowing into a node is equal to the sum of units flowing away from that node for every node in the network except for the source and the sink node. Condition (2) makes sure that the given maximal capacities of the edges are not exceeded. More details on such an optimization problem can be found in [4].

The Busacker-Gowen Algorithm, which can solve this problem in polynomial time, is also described in [4]. It starts with a feasible flow of value 0 on G and increases the flow value in n steps, one unit per step. In every step a shortest augmenting path has to be determined on the flow network $\widehat{G}$ that consists of the directed edges (i,j) of G with $\phi_{ij} < \kappa_{ij}$ and the back edges (k,l) with $\phi_{lk} > 0$.

2.7 The optimality condition

If we transport a flow of an value of n from r to s with minimal costs on the c_{lb}-weighted edges, we get an optimal solution for the matching problem with costs c^*_{lb}.

THEOREM 2.

$$c^*_{lb} \leq c(i)_{lb}$$

$$\forall \{i | i \text{ is a matching}\}$$

Due to the fact that all the classes have been matched, we can now recompute the costs $c_{estimated}$ for each matched pair of classes in order to get the exact costs c_{real}. And therefore we can formulate a sufficient optimality condition:

OPTIMALITY CONDITION 1.

$$\text{If } c_{\text{real}} = c^*_{\text{lb}}$$

then c^*_{lb} *is optimal.*

2.8 The steps

In this section we summarize the steps of the algorithm outlined so far.

1. Construction of the network graph G containing the bipartite graph induced by the classes of the two diagrams as described in section 2.5
2. Computation of $c_{lb}(a_i,b_j)$, $c_{lb}(a_i,\epsilon)$ and $c_{lb}(\epsilon,b_j)$ $\forall$ classes $a_i \in A$ and $b_j \in B$ with estimated costs for inheritance relationship edges and association relationship edges as described in sections 2.3 and 2.4
3. Solution of the optimization problem described in section 2.6: a flow of n units from r to s with minimal costs c^*_{lb} on the c_{lb}-weighted edges of G has to be found.
4. Determination of the exact costs of the found matches c_{real}
5. If $c^*_{lb} = c_{real}$ a minimal exact matching has been found.

2.9 An example

In this section we have a look at a simple example: We compute the corresponding classes of Diagram A and Diagram B as shown in figures 2 and 3. The internally used

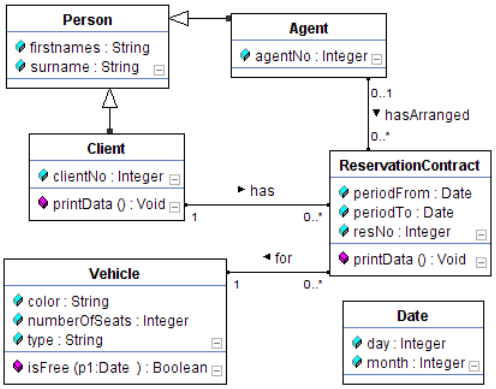

Figure 2: Class diagram A (modeled in Fujaba).

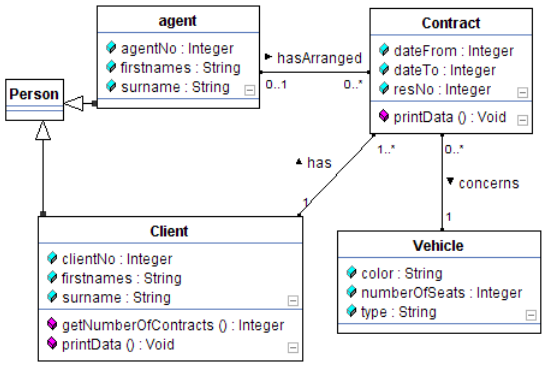

Figure 3: Class diagram B (modeled in Fujaba).

identifiers in this example can be taken from table 4. For a class $a_i \in$ Diagram A, there are possibilities of matches with class $b_j, j = 1, \ldots, 5$ and the deletion of a_i.
The values of the cost function for this example can easily be computed. See table 2 for the possible matches of class a_4. Let us explain the entries in the row (a_4,b_1): If a_4 has to be transformed into b_1, the class name has to be changed (costs: 1), the attribute `clientNo` has to be deleted (costs: 3), the operation `printData()` has to be deleted (costs: 3). a_4 has an outgoing inheritance relationship edge, b_1 has two incoming inheritance relationship edges. Therefore at least three inheritance relationship edges have to be deleted/created (costs: 3∗2:2=3). a_4 has an association relationship edge, b_1 has no association relationship edge. Thus the association edge has to be deleted/created (costs:4:2=2). All in all: $c_{real}(a_4,b_1) \geq 12$, c_{lb}=12. After computation of the costs $c_{lb}(a_i,b_j)$ $\forall$ i,j, $c_{lb}(a_i,\epsilon)$, $\forall$ i and $c_{lb}(\epsilon,b_j)$, $\forall$ j (see table 2) and the construction of the network graph G (see figure 5) the Busacker-Gowen Algorithm finds the minimal cost matching $\{(a_1,b_1), (a_2,b_2), (a_3,b_3), (a_4,b_4), (a_5,b_5), (a_6,\epsilon)\}$ with summary costs of c^*_{lb}=42. As $c_{lb}* = c_{real}$ we have found an optimal exact matching.

If class a_6 in Diagram A had a further attribute `year` of type `Integer` the Busacker-Gowen Algorithm would deliver the minimal cost matching $\{(a_1,b_1), (a_2,\epsilon), (a_3,b_3), (a_4,b_4), (a_5,b_5), (a_6,b_2)\}$ with summary costs of c^*_{lb}=44. As $c_{lb}* < c_{real} = 51$ we have not found an optimal exact matching in this case.

Figure 4: Class identifiers in example 2.9.

identifier	diagram: class name
a_1	Diagram A: Person
a_2	Diagram A: Agent
a_3	Diagram A: ReservationContract
a_4	Diagram A: Client
a_5	Diagram A: Vehicle
a_6	Diagram A: Date
b_1	Diagram B: Person
b_2	Diagram B: agent
b_3	Diagram B: Contract
b_4	Diagram B: Client
b_5	Diagram B: Vehicle

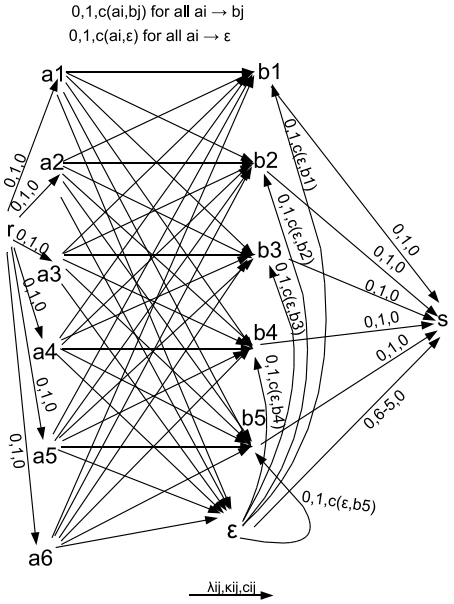

Figure 5: The extended bipartite graph.

3. FUTURE RESEARCH

Once the algorithm has been implemented, we can apply it to test sets of diagrams to evaluate the percentage of cases where an exact solution can be found. In order to increase this number the defined cost function or its estimation may have to be modified. Furthermore the underlying data model can be extended and refined. Finally, the result of the exact algorithm may be compared with that of heuristic algorithms. The question remains, however, whether the user considers the exact matching with respect to a given cost function in fact as more accurate as the matching computed using heuristics.

4. REFERENCES

[1] S. Förtsch and B. Westfechtel. Differencing and merging of software diagrams - state of the art and challenges. In J. Filipe, B. Shishkow, and M. Helfert, editors, *ICSOFT 2007 - Second International Conference on Software and Data Technologies*. INSTICC, 2007.

[2] U. Kelter, J. Wehren, and J. Niere. A generic difference algorithm for UML models. In P. Liggesmeyer, K. Pohl, and M. Goedicke, editors, *Software Engineering 2005*, LNI 64, pages 105–116. GI, 2005.

Table 2: The values of the cost function.

	name	attr.	op.	inher.	ass.	$C_{certain}$	C_{lb}
$c(a_1, b_1)$		6				6	6
$c(a_1, b_2)$	1	3		3	2	4	9
$c(a_1, b_3)$	1	7	6	2	6	14	22
$c(a_1, b_4)$	1	3	6	3	2	10	15
$c(a_1, b_5)$	1	5	0	2	2	6	10
$c(a_1, \epsilon)$	2	6		2		8	10
$c(a_2, b_1)$	1	3		3	2	4	9
$c(a_2, b_2)$	1	6				7	7
$c(a_2, b_3)$	1	7	3	1	5	11	17
$c(a_2, b_4)$	1	7	6		1,5	14	15,5
$c(a_2, b_5)$	1	7	0	1	1	8	10
$c(a_2, \epsilon)$	2	3		1	2	5	8
$c(a_3, b_1)$	1	9	3	2	6	13	21
$c(a_3, b_2)$	1	5	3	1	5	9	15
$c(a_3, b_3)$	1	4	0		1	5	6
$c(a_3, b_4)$	1	5	3	1	5	9	15
$c(a_3, b_5)$	1	5	3		4,5	9	13,5
$c(a_3, \epsilon)$	2	9	3		6	14	20
$c(a_4, b_1)$	1	3	3	3	2	7	12
$c(a_4, b_2)$	1	7	3		1	11	12
$c(a_4, b_3)$	1	7	0	1	5	8	14
$c(a_4, b_4)$		6	3		0,5	8	8,5
$c(a_4, b_5)$	1	7	3	1	0,5	11	12,5
$c(a_4, \epsilon)$	2	3	3	1	2	8	11
$c(a_5, b_1)$	1	9	6	2	2	16	20
$c(a_5, b_2)$	1	3	6	1	1	9	11
$c(a_5, b_3)$	1	5	5		5,5	11	16,5
$c(a_5, b_4)$	1	3	5	1	1	9	10
$c(a_5, b_5)$			6		0,5	6	6,5
$c(a_5, \epsilon)$	2	9	6		2	17	19
$c(a_6, b_1)$	1	6		2		7	9
$c(a_6, b_2)$	1	6		1	2	7	10
$c(a_6, b_3)$	1	5	3		6	9	15
$c(a_6, b_4)$	1	6	6	1	2	13	16
$c(a_6, b_5)$	1	6			2	7	9
$c(a_6, \epsilon)$	2	6				8	8
$c(b_1, \epsilon)$	2			2		2	4
$c(b_2, \epsilon)$	2	9		1	2	11	14
$c(b_3, \epsilon)$	2	9	3		6	14	20
$c(b_4, \epsilon)$	2	9	6	1	2	17	20
$c(b_5, \epsilon)$	2	9			2	11	13

[3] A. Mehra, J. C. Grundy, and J. G. Hosking. A generic approach to supporting diagram differencing and merging for collaborative design. In D. F. Redmiles, T. Ellman, and A. Zisman, editors, *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, pages 204–213. ACM, 2005.

[4] K. Neumann and M. Morlock. *Operations Research*. Carl Hanser Verlag, 2002.

[5] J. Rho and C. Wu. An efficient version model of software diagrams. In *Asia Pacific Software Engineering Conference*, pages 236–243. IEEE Computer Society Press, 1998.

[6] Z. Xing and E. Stroulia. UMLDiff: an algorithm for object-oriented design differencing. In D. F. Redmiles, T. Ellman, and A. Zisman, editors, *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, pages 54–65. ACM, 2005.

.