Truly Adaptive Optimization: The Basic Ideas

Giovanni Maria Sacco

Dipartimento di Informatica, Università di Torino, Corso Svizzera 185,
10149 Torino, Italy
sacco@di.unito.it

Abstract. A new approach to query optimization, truly adaptive optimization (TAO), is presented. TAO is a general optimization strategy and is composed of three elements:

1. a fast solution space search algorithm, derived from A*, which uses an informed heuristic lookahead;
2. a relaxation technique which allows to specify a tolerance on the quality of the resulting query execution plan;
3. a paradigm to prove the suboptimality of search subspaces. Non-procedural pruning rules can be used to describe specific problem knowledge, and can be easily added to the optimizer, as the specific problem becomes better understood.

The main contribution over previous research is the use of relaxation techniques and that TAO provides a unifying framework for query optimization problems, which models a complexity continuum going from fast heuristic searches to exponential optimal searches while guaranteeing a selected plan quality. In addition, problem knowledge can be exploited to speed the search up. As a preliminary example, the method is applied to query optimization for databases distributed over a broadcast network. Simulation results are reported.

1 Introduction

Non-procedural query interfaces for relational database systems provide a high degree of data independence and greatly simplify user interaction with the database. The selection of efficient strategies to solve user queries is delegated to a system component called the query optimizer. The query optimizer selects an execution plan on the basis of the estimated costs of different plans that can be used to solve the query.

Research on query optimization [2] has polarized on two different, unreconciled extremes. On the one hand, a number of works uses the exhaustive exploration of the solution space to select the plan with the minimum estimated cost among all the possible plans that can be used to solve the query. As an example, a breadth-first exhaustive search is used in the optimizer of System R [14], a centralized relational database system, and in R* [17], its distributed counterpart. More recent approaches are iterative dynamic programming [9], and the blackboard approach [8].

The main objection to this approach is that query optimization is known to be NP-hard [3], so that the enumeration of all the possible plans requires significant resources for complex queries on several relations. Another important and often overlooked

S. Bressan, J. Küng, and R. Wagner (Eds.): DEXA 2006, LNCS 4080, pp. 751 – 760, 2006.

objection is that estimates of execution costs are known to suffer from significant errors. Pushing the optimization process to a point where discrimination among plans falls inside the estimation error obviously produces no meaningful benefits.

On the other hand, many algorithms use a priori heuristics to derive efficient execution plans. Heuristic strategies require a relatively low amount of resources, depending on the refinement of the strategy, but can be seriously suboptimal. Known problems with heuristics are: a) the use of a priori strategies, usually tailored on intuitive "average" cases; b) the difficulty in optimizing weighted combinations of objective functions; c) no indication on plan quality, in terms of how far is the selected solution from the theoretical optimum; and d) "one size fits all": heuristics do not account for the relative importance of queries. Thus, the selected plan for a given query is always the same, regardless of its frequency of execution.

Alternative approaches to heuristic optimization are randomized algorithms [7], that have constant space overhead but whose running time is usually unpredictable because they are non-deterministic. Typically, randomized algorithms are slower than heuristics and dynamic programming for simple queries and faster than both for very large queries. The best known randomized algorithm for query optimization is 2PO and is a combination of applying iterative improvement and simulated annealing [7]. In many situations, 2PO produces good plans. However, there are situations in which 2PO produces plans that are orders of magnitude more expensive than an optimal plan.

The goal of our research is to provide a single, general and efficient optimization framework that can bridge the gap between exhaustive methods and heuristic solutions by allowing a tolerance on the quality of evaluation plans to be set. Such a tolerance is used to model a complexity continuum going from fast heuristic searches (infinite tolerance) to exponential optimal searches (no tolerance). Thus, as in heuristic strategies, we can speed up the search by providing approximate, suboptimal solutions. Differently from heuristics, however, we are able to guarantee that the selected solution does not exceed the admissible tolerance. In addition, the framework can exploit problem knowledge in order to speed the search up: problem knowledge is at the basis of heuristics, but is usually not exploited by exhaustive strategies.

The optimization framework proposed here is general and can be applied to any optimization problem. In order to provide examples and initial performance measures, it is applied to a simple distributed query optimization problem. The sample problem is not interesting per se, but because it is NP-hard without unnecessary complexities, and it allows the easy gathering of preliminary measures on our approach.

2 Background

Adaptive optimization [1] was the first method to apply results from Artificial Intelligence, namely the A* search algorithm [10], to query optimization and to reconcile these two antithetic approaches into a "middle-of-the-way" approach, called adaptive optimization. In adaptive optimization, the optimization problem is solved through an efficient search of the solution space, which is significantly faster than previous enumerative methods. In addition, the portion of the solution space to be searched can be controlled by a parametric pruning criterion based on the cost of the solution space node to which it is applied.

The A* method [10], a well-known graph search technique, is used in adaptive optimization. A* expands at each stage the node n (not yet expanded) having the smallest total expected cost f*(n) given by:

$$f^*(n) = g(n) + h^*(n) \quad (1)$$

where g(n) is the total cost incurred in producing the database state represented by n, and h*(n) is the estimated cost to reach a state at which the query is solved, from state n. It can be proved [10], that if h*(n)<=h(n) for all n's, where h(n) is the actual future cost to solve the query, A* is guaranteed to reach the optimum. This constraint is called the *admissibility constraint*.

In adaptive optimization, the user (or more likely, the system administrator) can choose h*(n)=0, for all n's, which satisfies the admissibility constraint but results in an inefficient breadth-first search. Alternatively, the user can use the cost of a heuristic solution from n (such as the Initial Feasible Solution in point-to-point networks [4]) as h*(n). This estimate produces a more focused search but does not generally satisfy the admissibility constraint, so that the optimum is not guaranteed. Finally, the amount of resources spent in optimization can be reduced by specifying a parametric pruning criterion, which operates on a list of successors of node n ordered by increasing total cost. Let mtc designate the minimum estimated total cost node. The user can specify that each successor node s such that f*(s)/f*(mtc)>1+T is to be pruned. That is nodes whose total cost is beyond a specified deviation from the minimum are pruned. In addition, the user can request that only the N lowest cost successors of n be retained.

This type of search allows implementing a number of strategies, from greedy searches (if only one successor is retained for each expanded node) to exhaustive searches (if no successor is pruned). The term adaptive is used to indicate that heuristic pruning only depends on the cost of active nodes, rather than on a priori heuristics. In addition, this strategy allows the user to specify tight pruning on one-shot queries (for which very efficient plans are relatively unimportant), and loose pruning or no pruning at all, for often repeated queries (for which even small savings are important in a large scale economy).

Although adaptive optimization goes a long way towards low-cost high-quality query optimization, it does have a number of problems: a) it is quite inefficient for complex queries, so that tight pruning criteria must generally be used; b) it does not give an indication of the quality of the plan unless when used to reach an optimal solution (i.e. with h*(n)=0 and no pruning); and c) it is quite difficult to understand the implications, in terms of space and time, of selected values of N and T.

3 Truly Adaptive Optimization

Truly adaptive optimization (TAO) improves over adaptive optimization in two main ways. First, it concentrates on approximate solutions, since there is little interest in reaching the theoretical optimum because of errors in cost estimates. Differently from adaptive optimization that relaxes the search in order to reduce effort, but is unable to give indications on the quality of the resulting plan, here we want to be able to guarantee that the resulting plan is within a specified tolerance from the optimal solution. At the same time, we want to reduce the search effort as the tolerance on plan quality increases.

In order to meet these goals, algorithm A* with no parametric pruning is used as a basis. Admissible h*(n) estimates are obtained by *completely reduced plans.* A completely reduced plan for n relations is a plan in which all relations are considered in their completely reduced form (i.e. containing the minimum number of tuples and attributes required to compute the result) and are processed by the least expensive actions.

When a node n is selected for expansion, a feasible plan for n, FP(n), is computed by *heuristic lookahead*. FP(n) is the result of a heuristic solution from n, obtained by expanding depth-first the successors of n with minimum f* cost. Initially, the feasible plan from the initial node I, FP(I), is computed. This feasible plan is updated whenever a better feasible plan is computed, so that the cost of FP(I) gives at each stage the minimum cost of a feasible solution, HC(I). At the same time, the estimated cost of the node n, f*(n) to be expanded gives an underestimate of the cost of an optimal solution from I that includes n. Since the node n to expanded by A* is the node with the minimum estimated cost, the search process can be stopped when

$$HC(I)/f^*(n) <= 1+T \tag{2}$$

where T ($T \geq 0$) is the tolerance specified (*relaxed search*).

In this way, the optimization process can be seen as a refinement of the best heuristic solution so far obtained. Such a refinement is only carried out as required by the specified tolerance and stops when the heuristic solution is good enough. Thus, this relaxation models a complexity continuum going from going from fast heuristic searches (infinite tolerance) to exponential optimal searches (no tolerance). As in heuristic strategies, we can speed up the search by providing approximate, suboptimal solutions. Differently from heuristics, however, we are able to guarantee that the selected solution does not exceed the admissible tolerance.

Relaxation plays a central role in TAO because it is the only mechanism used to produce approximate solutions.. First of all, the higher the tolerance is, the lower the search effort generally is. In the limit case of an infinite tolerance, the first heuristic solution computed by TAO is acceptable, and the search is immediately terminated. In this case, the complexity of a TAO search equals the complexity of a greedy heuristic strategy: the minimum complexity for a meaningful optimization strategy. As the tolerance decreases, more and more solutions will be tested. TAO can be then practically applicable even to very complex problems by specifying relatively large tolerances in order to decrease the amount of resources needed for the search. Significant tolerances are required anyway because of compound errors in result estimates.

The second improvement over adaptive optimization is a uniform way of exploiting problem knowledge. Problem knowledge is the basis of heuristic strategies. Optimal solution space search strategies do not account for it, except, as in A*, to improve estimates of future costs and consequently the penetrance of the search. Estimates of future costs are an important part of TAO and admissible estimates were discussed above. In addition, a paradigm to accommodate non-procedural problem knowledge in order to speed up the search is used.

Whereas heuristics use problem knowledge to specify which portions of the solution space are considered, the mechanism used in TAO, *pruning rules*, specifies which portions of the solution space can be proved suboptimal and can therefore be

pruned without losing optimal solutions. Pruning rules are based on the following *harmless pruning paradigm*: given two nodes n and n', n' can be harmlessly pruned iff, assuming that an optimal solution is reachable from n', it can be proved that an optimal solution is also reachable from n. Pruning a node n means removing it and its successors. Storage is released so that there is a decrease in the workspace requirements. A decrease in the time complexity of the search occurs if a pruned node n would have eventually been considered for expansion.

As we will show in the following, there are a number of general pruning rules that are applicable to any optimization problem, while other rules can be derived for specific problems. Among general rules, the duplicate state rule is especially important for query optimization. This rule states that given two nodes n and n' that represent the same database state, only the node with the minimum total estimated cost needs to be retained. Due to join commutativity, the same database state can be produced by a combinatorial number of plans. Only the lowest cost plan to reach a database state needs to be preserved. This action obviously preserves the optimum, and is especially important to reduce the search effort.

In summary, truly adaptive optimization is composed of three main elements:

1. a fast search strategy based on the A* search algorithm
2. a mechanism to specify an acceptable tolerance on the quality of resulting plans
3. a mechanism to specify pruning rules: rules that account for general and problem specific knowledge and can be used to prune portions of the solution space, while retaining the optimum solution.

TAO is a general optimization framework and can be applied to any optimization problem. However, for expediency, we will refer specifically to the application of TAO to distributed query optimization in the following discussion.

4 Pruning Rules

A number of pruning rules can be applied to any optimization problem, and consequently to problems different from query optimization. In addition, rules may be defined at different levels of abstraction for query optimization in general, for distributed query optimization, and for specific problems (specific rules for distributed query optimization in point-to-point tree networks are discussed in [13]). A few general pruning rules are discussed in the following.

Higher cost rule: Any node n such that $f^*(n) \geq HC(I)$ can be pruned.

Proof: By the admissibility of h*: $f^*(n) \leq f(n)$. Consequently, $HC(I) \leq f(n)$, and if an optimal solution is reachable from n, it is also reachable using the current heuristic solution from HC(I). Therefore, nodes having an expected cost higher than the cost of the current heuristic solution can be safely pruned.

Whenever a node n with minimum f* is selected by heuristic lookahead, the heuristic can keep track of the minimum f*(n') among the siblings n' of n. We denote this quantity by F*(n); it provides the total expected cost of the cheapest alternative to the selected node. When HC(n) becomes available (i.e. when the heuristic backs up from

its recursive descent), the following rule can be applied to immediately detect when the siblings of a heuristically generated node are suboptimal, and consequently to prune them before expansion:

Alternate cost rule: Let n' be the son of n for which the heuristic solution was computed. If $g^*(n) + HC(n) \leq F^*(n')$, then n need not be expanded (and no siblings of n' are consequently generated).

Proof: By contradiction. Let an optimal solution be reachable from a sibling of n', n", but not from n'. By the admissibility of h*: f(n") ≥ F*(n'). Therefore, $g(n) + HC(n) \leq f(n")$, against the hypothesis.

In most searches, a given database state can be represented by a set N of nodes. This is mainly due to join commutativity: all the sequences derived by permutation from a set of joins produce the same database state. Only the cheapest node for a given database state needs to be preserved. The following rule applies:

Duplicate state rule: Given two nodes, n and n', both representing the same database state, n can be pruned if $f^*(n) \geq f^*(n')$.

Proof: Since n and n' correspond to the same state, $h^*(n)=h^*(n')$. Consequently, $f^*(n) \geq f^*(n')$ implies that if an optimal solution is reachable from n it is also reachable from n'.

Hashing techniques can be used to efficiently detect state duplication. If a heuristic solution from n has already been computed, and $f^*(n) = f^*(n')$, then n is to be retained, and n' pruned, in order to avoid the recomputation of the heuristic. If $f^*(n) > f^*(n')$, the heuristic solution from n is absorbed by the new node n'. The duplicate state rule significantly reduces the solution space to be searched.

Assume that an action α is being applied, and that it has a minimum cost, i.e. equal to the cost in the completely reduced plan. This means that

1. action α must be performed to find a solution (otherwise its cost in the completely reduced plan would be 0), and
2. that no other action can decrease its cost.

Action α is thus a locally optimal move and should be immediately performed. Consequently, all the siblings of the node generated by α (i.e. alternate actions) need not be considered.

Process ASAP rule: Let $n_1, ..., n_i, \ldots, n_k$ be the set N of all the immediate successors of node n. If a node n_j exists, such that $f^*(n.) / f(n) = 1$, then all the siblings of n_j can be pruned.

Proof: If the condition is met, the action, which generates nj from n, has the same cost in the current plan and in the completely reduced plan, i.e. the minimum possible cost. Remember that only necessary actions have a non-null cost in the completely reduced plan. Consequently, no decrease in the cost of the action is obtained by postponing it, and the action is a local optimal move. This rule is especially useful to avoid exploring all the permutations of actions in a set of completely reduced relations.

5 Experimental Data

Two different distributed query optimization problems were studied [12] in order to investigate the most important properties of TAO. These problems are defined on databases distributed over a broadcast network [11]. Here we studied two different formulations: 1. only relation transmissions are considered; and 2. both relation and attribute transmissions for semijoins are considered.

The experiments were conducted on samples of random queries generated by a query synthesizer. For each random query, a random initial database state was generated by the synthesizer. Each sample consisted of 3500 queries, subdivided into 7 sets from 3- to 9-relation queries.

The first experiment was designed to investigate whether TAO searches offer a significant speedup over A* searches, and to compare plan quality and search effort of TAO vs. a heuristic strategy. These experiments were conducted on optimization experiment 1. The reference heuristic is strategy B-1 by Sacco [11], which iteratively broadcasts the smallest relation not yet transmitted. In order to characterize the search requirements, we used the total number of nodes created t, which gives a measure of time complexity. Simulated strategies are:

1. a uninformed A* search (h*(n)=0, for all n's)
2. an informed A* search with the elimination of duplicate states (h*(n) is estimated by completely reduced plans)
3. a TAO search with generally applicable rules but no relaxation
4. the B-1 heuristic algorithm

T's for the first three strategies are reported in figure 1. From these results, it can be concluded that uninformed A* searches are not practical except for problems on few variables. A case for considering informed searches and the elimination of duplicate states is provided by strategy (2), whose space and time complexity appears practical even for problems with many variables. TAO searches produce significant improvements over strategy (2) both in time and in space. In particular, for 9-relation queries, 4.9 plans were inspected and 3 plans stored, on the average. This compares favourably with heuristic B-1, which stores approximately 2 plans.

Table 1 reports the observations on plan quality for the B-1 heuristic and the first successful TAO heuristic lookahead. The heuristic used in TAO lookahead appears consistently better than B-1, with an average deviation from the optimum not exceeding 4% and a maximum observed deviation of 110%. In addition, over 63% of the plans generated by the first TAO heuristic solution are optimal. These results provide a good case for the use of *informed heuristic strategies*. However, the quality of heuristic plans depends on the specific optimization problem: in fact, the heuristic lookahead for problem 2 produced average deviations from optimum ranging from 6% to 21%, and a maximum observed deviation of 5921%. Therefore, heuristic strategies can be acceptable for ad-hoc queries, but their worst case can be quite bad for heavy, highly repeated queries, for which the slightly higher cost of a TAO search appears well justified.

The second experiment addresses relaxed searches. Since TAO searches are very efficient for problem 1, we used the more complex problem 2. Results are aggregated by number of possible actions at the start node, that more closely models actual query

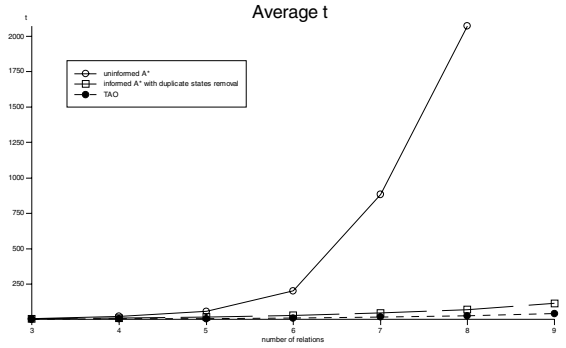

Fig. 1. Average t for problem 1

Table 1. Comparison of costs of heuristic solutions

#rel	B-1		TAO heuristic lookahead	
	Mean deviation	Max deviation	Mean deviation	Max deviation
3	0.03	0.92	0.01	0.67
4	0.05	0.92	0.02	0.94
5	0.07	0.84	0.03	0.73
6	0.08	1.35	0.03	0.53
7	0.07	1.43	0.04	0.93
8	0.10	1.47	0.04	1.10
9	0.09	1.28	0.03	1.02

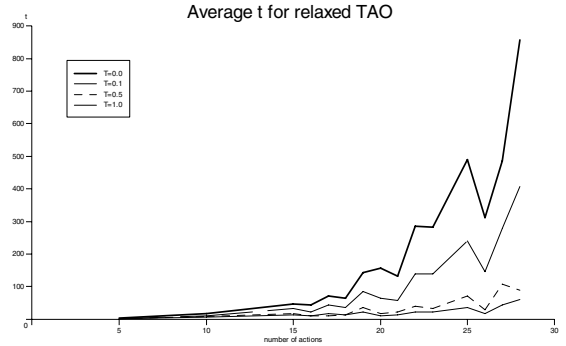

Fig. 2. Average t for optimal and relaxed TAO searches (problem 2)

complexity than the number of relations in the query. The results reported are relative to a 10%, a 50% and a 100% tolerance. Because of estimation errors, we believe that at least a 100% tolerance would be used in practice. Results are reported in figure 2, and show that even minimal tolerances produce significant benefits. Although relaxed

searches guarantee that no plan deviates from the optimum more than the allowed tolerance, actual average deviations are very low and range from 1% (10% tolerance) to 15% (100% tolerance). Consequently, even large tolerance produce very good plans, on the average.

6 Conclusions

Our primary goal in the present research was to bridge the gap between exhaustive and heuristic optimization, and produce a single, general and coherent framework that can be easily adapted to any optimization problem and can be used to produce solutions whose cost is within a predefined deviation from an optimal solution. With respect to previous research, the "one size fit all" approach is finally overcome, and the required quality of execution plans can account for their frequency, criticity, and expected estimation errors. In practice, as we indicated previously, we expect relaxed searches to be the norm, with admissible deviations as large as 100% from optimal costs. The experiments conducted show that TAO searches are indeed close contenders of heuristic searches, as far as search effort and resources are concerned. Even moderate tolerances produce a significant search speedup, while selected plans are, on the average, much better than the admissible deviation. Analytic measurements for pruning rules are not reported here, but the results reported in [13] show that they can produce a significant, additional decrease in the overall complexity of the search.

This paper is primarily based on an unpublished research corpus dating back from the'80s. A number of important papers have appeared since, and new optimization strategies proposed. However, the main motivations of the present research are still valid [18]. The severe impact of estimation errors on plan optimality is now widely recognized and analysed in literature [6]. The fact that heuristic strategies produce plans whose uncontrolled quality may be abysmally low is now acknowledged [15]. There is a growing consensus on the fact that optimization costs must be reduced, or at least controlled. For instance, Ilyas et al. [5] propose a strategy to reduce optimization costs by using an acceptable upper bound on the estimated cost of optimization to stop the optimization process before completion, if required. Although there is a bound on optimization effort, there is no guarantee on the quality of the solution. In relaxed TAO searches, on the contrary, the plan generated is guaranteed within a predefined tolerance from the optimum, but no bound on the time required for optimization can be guaranteed. However, this approach can easily incorporated in TAO, with the significant benefit that an upper bound on the resulting plan quality is available, if the TAO search is prematurely terminated.

Finally, we believe the results reported by Waas and Galindo-Legaria [16] to be especially relevant here. Their work shows that with a relatively small sample from the solution space, it is possible to find plans that are pretty close to the optimum. In fact, the percentage of plans that are within twice the optimum cost is non-trivial. These results confirm our evidence of the high efficiency of TAO, even on queries on many relations.

References

1. Balbo, G., Di Leva, A., Sacco, G. M., Adaptive query optimization in point-to-point networks, in: "Distributed Data Sharing Systems", (F.A. Schreiber, ed.), North-Holland, 1984
2. Chaudhuri, S., An overview of query optimization in relational systems, Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1998, 34 - 43
3. Hevner, A.R., The optimization of query processing in distributed database systems, Ph.D. Thesis, Purdue University, W.Lafayette, IN, 1979
4. Hevner, R.A., Yao, S.B., Query processing in distributed database systems, IEEE Trans. On Software Engineering, SE-5:3, 1979
5. Ilyas, I., et al. Estimating compilation time of a query optimizer, Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 2003, 373 - 384
6. Ioannidis, Y. E., Christodoulakis, S., On the propagation of errors in the size of join results, Proceeding of the ACM SIGMOD Conf., 1991
7. Ioannidis,Y.E., Kang, Y. C. Randomized algorithms for optimizing large join queries. In Proc. of the 1990 ACM SIGMOD Conf., 312–321, 1990
8. Kemper, A., et al. A blackboard architecture for query optimization in object bases. In Proceedings of the Conference on Very Large Data Bases, 1993
9. Kossmann,D., Stocker, K. Iterative dynamic programming: A new class of query optimization algorithms. ACM Transactions on Database Systems 25, 1, 2000
10. Nilsson, N.J., Principles of artificial intelligence, Tioga Publishing Company, Palo Alto, CA, 1980
11. Sacco, G.M., Distributed query evaluation in local area networks, Proceed. IEEE Conf. on Data Engineering, 1984
12. Sacco, G. M., Truly adaptive query optimization , Dept. of Computer Science, Univ. of Torino, Italy, TR 4/8/84, 1984
13. Sacco, G. M., Truly adaptive query optimization in point-to-point tree networks, Dept. of Computer Sciences, University of Torino, Italy, TR 3/21/89, 1989
14. Selinger, P.G., et al, Access path selection in a relational database management system , Proceed. ACM SIGMOD Conference, 1979
15. Steinbrunn, M., Moerkotte, B., Kemper, A., Heuristic and randomized optimization for the join ordering problem**,** The VLDB Journal, 6:3, 1997, 191 - 208
16. Waas, F., Galindo-Legaria, C., Counting, enumerating, and sampling of execution plans in a cost-based query optimizer, Proceedings of the 2000 ACM SIGMOD Conference, 2000, 499 - 509
17. Williams, R. et al.. R*: An Overview of the Architecture. IBM Research, RJ3325, 1981 Reprinted in: M. Stonebraker (ed.), Readings in Database Systems, Morgan Kaufmann Publishers, 1994, 515–536.
18. Winslett, M., David DeWitt speaks out, ACM SIGMOD Record, 31:2, 2002, 50 – 62.