Essay: An Evolutionary Multi-Agent System for Database Query Optimization

Slava Shekh

University of Adelaide

Abstract. Query optimization is a critical step in query processing, providing significant performance improvements to the operation of database systems. For over 30 years, query optimization has been the subject of research and a wide range of optimization methods have been applied to this problem. Gonçalves [1] proposes a novel approach to query optimization using an evolutionary multi-agent system, in which autonomous agents explore the search space and evolve based on feedback from the environment. This essay discusses the system developed by Gonçalves [1], compares it to existing approaches in the literature, and provides an overall critique of the work.

1 Introduction

Databases and database management systems (DBMS) play a crucial role in almost all modern computer systems, providing the means to define, store and manage information [2]. Queries provide a mechanism for retrieving specific information from these databases. One of the most important parts of processing a query is considered to be the query optimization step, because it has a direct impact on database performance [3]. Query optimization involves ordering the database operations in the query in order to minimize the overall execution time.

The join operation, which involves combining two or more database relations, is the most time consuming database operation [4]. Hence, the effectiveness of query optimization is largely influenced by the order in which the relations in a query are joined. The significance of the join operation has lead to a combinatorial optimization problem called the join ordering problem, which involves ordering the join operations in a query to minimize the overall execution time. For example, consider a join operation on the relations R_1, R_2, R_3 and R_4. Figure 1 shows two possible ways in which these four relations can be joined. Both join orders will produce the same result, but one may take significantly longer to evaluate [5]. The join ordering problem has been shown to be NP-complete [6], meaning that there is no known way of solving the problem efficiently (in polynomial time).

This essay discusses and critiques the work of Gonçalves [1] involving the development of an evolutionary multi-agent system for the join ordering problem. In a multi-agent system, many intelligent agents interact competitively or cooperatively in an environment, often to solve a particular problem [7]. On the

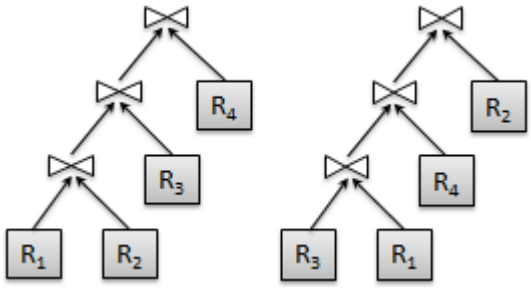

Fig. 1. Two possible join orders for a query containing four relations.

other hand, evolutionary algorithms attempt to solve problems by evolving a population of individuals representing different solutions in an effort to find a high quality solution. An evolutionary multi-agent system combines ideas from both of these approaches by allowing the agents or the system itself to evolve and adapt over time [8].

1.1 Motivation

Join ordering is a critical part of query optimization and has a direct impact on database performance. Ioannidis [9] demonstrates that the execution time of a query can vary by orders of magnitude, depending on the order of the join operations. The same query can take seconds, hours or days due to the varying I/O costs of storing temporary relations for different join orders [9]. Research into the join ordering problem helps query optimization algorithms minimize the occurrence of costly join operations.

Although efficient join orderings are desirable to find, the state space of the problem grows rapidly. Given N relations, there are $\frac{(2(N-1))!}{(N-1)!}$ different join orders [10]. Although a query involving 3 relations only has 12 possible join orders, a query with 10 relations has over 17 billion, and for larger queries the state space quickly becomes intractable. Therefore, exhaustive methods that fully explore the state space are only feasible for small values of N. As the number of relations grows, it becomes increasingly difficult to find a good join order in a limited amount of time. This further motivates the need for research into join order optimization, particularly in finding effective non-exhaustive optimization methods.

1.2 Relevant Work

The join ordering problem has been the subject of research for over 30 years, starting with the work by Selinger [11] using dynamic programming to target the problem exhaustively. This was followed by algorithms using a method called Nested-Loop-Join, which had a better time complexity than the dynamic programming approach, but didn't guarantee an optimal solution [6]. Since those

initial methods, a variety of optimization techniques have been applied to the problem, including simulated annealing [12], iterative improvement [13], genetic algorithms [14] and tabu search [15]. More recently, multi-agent systems have also been investigated as an option for join order optimization [4].

Due to the large number of optimization methods that have been applied to this problem, several comparative studies have been conducted [13, 16, 17]. These studies have shown that exhaustive approaches are more suited to less complex problems, and that no single optimization method is ideal for every situation. For example, Steinbrunn [17] compared dynamic programming, iterative improvement, simulated annealing and genetic algorithms, amongst others. The results showed that iterative improvement and genetic algorithms performed better when there was limited execution time, while simulated annealing started yielding higher quality solutions than other techniques when time was not a limiting factor [17]. Section 2.1 provides a more detailed comparison of some of these methods to the approach used by Gonçalves [1].

Although the recent focus has been primarily on non-exhaustive methods due to the rapidly growing state space of the problem, many modern DBMS such as PostgreSQL and H2 use a hybrid approach that combines the best of both exhaustive and non-exhaustive methods [1]. For example, a DBMS may use exhaustive search for small queries that have less than a certain number of relations. For any larger queries, a non-exhaustive method may be applied to get a good, albeit non-optimal join order.

2 Multi-Agent Query Optimizer

The main contribution of Gonçalves [1] is the development of an evolutionary multi-agent system for the join ordering problem, called Multi-Agent Query Optimizer (MAQO). Although evolutionary multi-agent systems have been successfully applied in other research areas, such as prediction systems [8], as far as the authors know evolutionary multi-agent systems have not been applied to join ordering [1]. This suggests that their overall approach is novel.

Given a query to optimize, MAQO initialises a number of agents that run in separate execution threads. The query is encoded into an ordered list of nodes, where each node represents a relation in the query, and the order of the nodes represents the join order. Figure 2 shows an example of how a query involving the joining of four relations is encoded.

One agent starts with the input query encoded in the order it was given, while other agents start with a randomized ordering of the relations in the query. This means that the agents will initially be randomly distributed across the state space. Each agent explores the state space, competing with others to find the best join order they can. Agents start with an initial number of life points equal to the number of relations in the query being optimized. At each iteration of its execution thread, an agent loses one life point. When an agent runs out of life, it dies, halting its thread. To prolong their existence, agents can request life

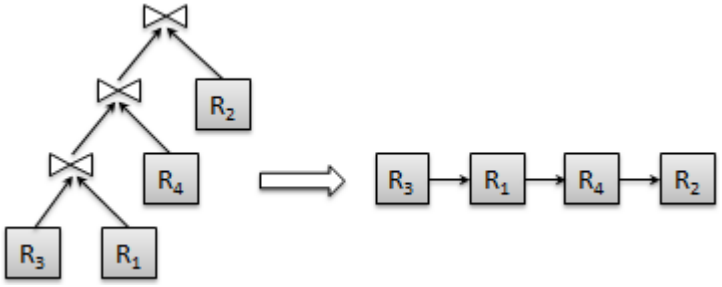

Fig. 2. Encoding a query containing four relations.

points from other agents. When all agents die, the algorithm outputs the best join order found across all agents and then terminates [1].

With the exclusion of life points, the behaviour described so far is not unusual for a traditional multi-agent system [7]. The evolutionary aspect of MAQO, and arguably one of the most novel aspects of the system, is the concept of agent profiles. A profile defines the actions that are available to an agent at a given point in time. At each cycle of its execution thread, an agent can be in one of five profiles: REPRODUCTION, MUTANT, SEMI-GREEDY, RANDOM-DESCENT or RESOURCE [1].

REPRODUCTION and MUTANT allow an agent to produce new solutions using crossover and mutation operators, while SEMI-GREEDY gives the agent the ability to apply a greedy heuristic search with some random variation to create new solutions. Once a good solution is found, it can be refined using a random-descent local search provided by the RANDOM-DESCENT profile. Finally, after executing for a number of cycles an agent may run low on life points, and the RESOURCE profile gives it the ability to request life points from other agents [1].

When an agent in the RESOURCE profile requests life points, it is more likely to receive them from agents that are in worse parts of the state space [1]. The general idea is that initially agents are dispersed across the search space, and over time, agents located in worse parts of the search space will die off, because fewer and fewer agents will provide them with life points. Conversely, agents that are located in the better parts of the search space will tend to live longer and have the opportunity to search for even better solutions. Although this is a novel concept, particularly in its application to the query optimization domain, this approach does have some general similarities to techniques like simulated annealing [12], such as the progression of the search from initially exploratory to becoming more and more focused over time.

In MAQO, each agent profile defines a unique behaviour and the combination of profiles across agents defines the overall behaviour of the system. Throughout the execution of the system, agents can change profiles, potentially multiple times [1]. This gives the agents the ability to adapt to the environment created by the particular problem that is being solved. For example, a query may be found

to have a very randomly distributed state space, requiring agents to be highly exploratory to find good solutions. In this case, a large number of the agents could switch to MUTANT profiles to give them this exploratory capability. If this is found to be ineffective, they could try changing to SEMI-GREEDY or REPRODUCTION profiles, and so on.

2.1 Experimental Results

When considering existing query optimization algorithms in the literature [17], the approach used in MAQO has considerable novelty. The competitiveness between agents, particularly their finite and rapidly decreasing life, angles the system towards rapid optimization. On the other hand, the evolutionary aspect gives the system adaptability in a largely unknown environment. The combination of these two properties results in an algorithm that can quickly and effectively navigate the search space, in a way that has not been previously explored for the join ordering problem.

Evidence of the algorithm's effectiveness can be seen in the experimental results, in which MAQO was compared to the H2 Query Optimizer (H2QO). H2QO is the official query planner of the H2 DBMS and uses a hybrid of exhaustive, greedy and genetic algorithms for the optimization [18]. For the experiments, each algorithm was run with 160 random queries, and this was repeated 3 times to ensure that results were consistent. Each algorithm was therefore run a total of 480 times [1].

Overall it was found that MAQO found lower cost query plans (implying more efficient join orders) than H2QO in 99 percent of cases [1]. This was found to be consistent across various experimental settings, including variable number of relations in the input query, and different types of query structures or shapes [1]. These results show a clear superiority of the query optimization in MAQO over H2QO, and demonstrate the potential effectiveness of an evolutionary multi-agent system for the join ordering problem. However, since MAQO was only compared to one other query optimization algorithm, further experimentation and comparison to state-of-the-art algorithms is necessary to validate these initial findings.

2.2 Other Query Optimizers

A wide range of optimization techniques have been applied to the join ordering problem, as introduced in Section 1.2. None of these methods use evolutionary multi-agent systems for the problem, but some of them share common elements with the MAQO [1]. The most relevant of these works is the application of genetic algorithms to the join ordering problem [14] and the use of multi-agent systems [4]. Each of these systems has been tested using a different experimental setting so direct comparison between algorithms is difficult. However, approaches can still be compared from an algorithmic point of view.

A genetic algorithm was first applied to the join ordering problem by Bennett [14]. The algorithm is initialised with a population of individuals that are

encoded similarly to MAQO, where each individual represents a particular join order. Individuals broadcast their fitness to a limited number of individuals in their local neighbourhood. Crossover and mutation are then applied, which produces offspring that replace their parents in the subsequent generation.

Experiments showed that the genetic algorithm was effective compared to the best known methods at the time [14]. In comparison, MAQO can support similar evolutionary behaviour by having a number of agents use the REPRODUCTION profile. This allows the agents to mimic the behaviour of the genetic algorithm, but also provides the flexibility to change profiles during execution to adapt to different types of queries. For example, the algorithm may start with a lot of REPRODUCTION agents, but if this proves to be ineffective, the agents could switch to MUTANT profiles. The genetic algorithm uses fixed crossover and mutation rates once execution is started [14], and therefore lacks the type of adaptability provided by MAQO.

The other research with strong relevance to MAQO is an approach that uses a multi-agent system for query optimization [4]. Given a query, a Query Distributor Agent subdivides the query between a number of Local Optimizer Agents. Each Local Optimizer Agent runs a local genetic algorithm on its part of the query to find an efficient join order. The Local Optimizer Agents then provide their optimized sub-queries to the Global Optimizer Agent that combines them in a way that tries to maximize the efficiency of the overall join order. This approach was shown to be effective in comparison to dynamic programming [4].

Although the approach used by Ghaemi [4] involves both genetic algorithms and a multi-agent system, it is not an evolutionary multi-agent system. The genetic algorithms are only applied in finding efficient join orders, but the agents and the overall system itself does not evolve over the course of execution. Running the system on different queries would result in similar overall behaviour. In contrast, the behaviour of the agents in the MAQO changes and evolves over the execution of the algorithm [1], adapting to the specific query being optimized and changing the general behaviour of the system.

3 Critical Review

Overall, the paper by Gonçalves [1] is well-written and provides thorough background information on the relevant domains, including database query optimization and evolutionary multi-agent systems. The paper also presents an overview of the most notable contributions to these areas from over 30 years of research [1], which is helpful for readers who are new to these areas. Both the contributions of the paper and the novelty of the approach used by Gonçalves [1] are made clear in relation to existing techniques in query optimization.

The main contribution of Gonçalves [1] is the development of MAQO, an evolutionary multi-agent system for the join ordering problem, which is a technique that hasn't been applied to join ordering in the past. The idea of an evolutionary multi-agent system is interesting because it combines two existing fields of artificial intelligence into a new concept that tries to bring together the advantages

of both. From the experimental results of MAQO, evolutionary multi-agent systems prove to be effective for join ordering [1], and it will be interesting to see what other optimization problems they could be applied to in the future.

One of the strengths of the paper is the detailed algorithm description, which provides the reader with a clear understanding of its mechanics. The algorithm consists of a number of agents executing actions associated with a particular profile. The specific actions of each agent profile are thoroughly described in the paper, and in some cases diagrams and examples are used to support the ideas. For example, in the REPRODUCTION profile, agents can use either the ordered crossover or the sequential construction crossover to generate new solutions [1]. The paper describes the functionality of each operator and supports this with visual examples to help convey the idea to the reader.

On the contrary, an area that could've been improved is the explanation of the circumstances under which agents change profiles. The paper describes the idea that there are multiple agent profiles and that an agent can change between them over the course of its execution. With the exception of the RESOURCE profile, which an agent can change to when it reaches a critical amount of life points [1], the triggers for other profile changes are not well explained. For instance, what would trigger a MUTANT agent to change to the REPRODUCTION profile?

3.1 Design Choices

In general, Gonçalves [1] didn't provide sufficient justification of design choices. A number of decisions have been made in developing MAQO, but the reasoning behind some of these decisions remains unclear. One example is the choice of mutation and crossover operators. The authors describe the mutation and crossover operators in detail, giving a clear explanation of their mechanics, but do not comment on the reason for choosing these particular operators or even the possibility of investigating other operators.

Similarly, the paper describes how the initial agent profiles are selected in a particular way, including one MUTANT, one RANDOM-DESCENT, one SEMI-GREEDY and a number of REPRODUCTION agents [1]. Yet, the reasoning for this selection is not discussed in the paper. Using a different combination of initial agent profiles may not have any effect on the algorithm, but without any explanation from the authors, it seems reasonable to speculate that different initial agent profiles could result in a different convergence rate for the algorithm, or even a different end result.

In the area of design choices, a strength of the paper is in the calibration of the algorithm. Two of the input parameters to the algorithm are "number of agents" and "initial life" [1]. Prior to running the experiments, these two parameters were calibrated over five random test queries to determine the best choice of parameter values. Performing this calibration meant that the authors did not have to choose arbitrarily parameter values, and the calibrated values resulted in improved performance of the MAQO algorithm during experimentation [1]. This is an example where the authors made a justified design choice, based on the results of the calibration. A similar approach could have been beneficial in

other parts of their algorithm, such as determining the most suitable mutation and crossover operators.

3.2 Experimentation

An additional strength of the paper is the description of the experimental setup and method. In particular, the paper explained the environment that was used to run the experiments, the method that was used to generate test queries, and statistical methods that were used for analysis [1]. This gives the reader a clear understanding of how the results were produced, and provides the possibility for future work to be directly benchmarked against MAQO.

The experiments themselves compared MAQO to H2QO on a number of randomly generated queries. Since H2QO is the official query planner of the H2 DBMS [18], the comparison was suitable, yet could've been further strengthened by also comparing MAQO to other query optimization algorithms. In particular, MAQO could've been compared to some of the techniques discussed in Section 1.2 or other state-of-the-art methods in the literature. This would provide more definitive evidence on whether MAQO is truly a superior algorithm for query optimization.

One of the unsolved problems for query optimization algorithms is the inaccuracy of database cost models, which algorithms use for cost estimation during optimization [19]. Gonçalves [1] found that the cost model in H2 DBMS suffers from the same inaccuracies, and therefore chose to base the experimental results on the estimates given by the cost model, rather than actual execution time. Although cost model inaccuracy is a known problem, reporting on theoretical cost rather than actual cost weakens the overall validity of the results. Future work could evaluate MAQO based on actual costs, or attempt to reduce the discrepancy between estimated and actual costs in order make cost models more useful in general.

4 Conclusion

Gonçalves [1] proposes a novel approach to the join ordering problem using evolutionary multi-agent systems. The experimental results show that the system developed by Gonçalves [1] consistently outperforms H2QO, the official query planner in the H2 DBMS. This suggests that evolutionary multi-agent systems are an effective approach for join ordering and database query optimization. However, since the results only compare MAQO to H2QO, further experimentation and comparison to state-of-the-art algorithms is needed to validate the results presented in the paper. This would confirm the effectiveness of MAQO and more broadly, support the notion of using evolutionary multi-agent systems for database query optimization.

References

1. Gonçalves, F.A., Guimarães, F.G., Souza, M.J.: An evolutionary multi-agent system for database query optimization. In: Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference, ACM (2013) 535–542
2. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. Addison Wesley (1990)
3. Hameurlain, A., Morvan, F.: Evolution of query optimization methods. In: Transactions on Large-Scale Data- and Knowledge-Centered Systems. Springer (2009) 211–242
4. Ghaemi, R., Fard, A.M., Tabatabaee, H., Sadeghizadeh, M.: Evolutionary query optimization for heterogeneous distributed database systems. World Academy of Science, Engineering and Technology **43** (2008)
5. Moerkotte, G.: Building query compilers. Available at http://db.informatik.uni-mannheim.de/moerkotte.html.en (2006)
6. Ibaraki, T., Kameda, T.: On the optimal nesting order for computing n-relational joins. ACM Transactions on Database Systems (TODS) **9**(3) (1984) 482–502
7. Wooldridge, M.: An introduction to multiagent systems. John Wiley & Sons (2009)
8. Cetnarowicz, K., Kisiel-Dorohinicki, M., Nawarecki, E.: The application of evolution process in multi-agent world to the prediction system. In: Proceedings of the Second International Conference on Multi-Agent Systems, ICMAS. Volume 96. (1996) 26–32
9. Ioannidis, Y.E.: Query optimization. ACM Computing Surveys (CSUR) **28**(1) (1996) 121–123
10. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database system concepts. Volume 4. McGraw-Hill Hightstown (1997)
11. Selinger, P.G., Astrahan, M.M., Chamberlin, D.D., Lorie, R.A., Price, T.G.: Access path selection in a relational database management system. In: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, ACM (1979) 23–34
12. Ioannidis, Y.E., Wong, E.: Query optimization by simulated annealing. Volume 16. ACM (1987)
13. Swami, A., Gupta, A.: Optimization of large join queries. Volume 17. ACM (1988)
14. Bennett, K.P., Ferris, M.C., Ioannidis, Y.E.: A genetic algorithm for database query optimization. Computer Sciences Department, University of Wisconsin, Center for Parallel Optimization (1991)
15. Matysiak, M.: Efficient optimization of large join queries using tabu search. Information sciences **83**(1) (1995) 77–88
16. Swami, A.: Optimization of large join queries: combining heuristics and combinatorial techniques. ACM SIGMOD Record **18**(2) (1989) 367–376
17. Steinbrunn, M., Moerkotte, G., Kemper, A.: Heuristic and randomized optimization for the join ordering problem. The VLDB Journal: The International Journal on Very Large Data Bases **6**(3) (1997) 191–208
18. Mueller, T.: H2 performance. http://www.h2database.com/html/performance.html (November 2013)
19. Theodoridis, Y., Stefanakis, E., Sellis, T.: Cost models for join queries in spatial databases. In: Proceedings of the 14th International Conference on Data Engineering, IEEE (1998) 476–483