Integrated Environment Based on Anytime Solution Search Algorithms and A Non-Relational Database for Real-Time Intelligent Systems

A. P. Eremeev[a], I. A. Poliushkin[a] and N. A. Paniavin[a]

[a]National Research University "MPEI", Moscow, Russia

Abstract

In this paper we describe how to apply an integrated environment based on anytime solution search algorithms and a non-relational (NoSQL) database to intelligent real-time systems. As an example, we consider the problem of routing between two points on a map. The choice of anytime algorithms of search of the decision is caused by their orientation for real-time searching. The choice of the NoSQL database is caused by greater efficiency when working with the information presented in the form of graphs compared to traditional databases. It is shown that anytime algorithms also allow to speed up the process of obtaining a solution, which is very relevant with rather strict routing time constraints. Integration of the anytime algorithms and NoSQL databases are particularly useful when solving problems with search spaces represented as a graph.

Keywords: Anytime Algorithms, NoSQL Databases, Heuristic Search, Routing.

1. Introduction

As an example of the problem of finding a real-time solution using graphs and anytime search algorithms based on heuristic functions actively used in intelligent real-time systems (IRTS) [Eremeev and Mitrofanov 2011], we consider routing between two points on a map. Due to presence of the natural heuristic function which is the shortest distance between two points, known heuristic search algorithms are widely used to solve this problem, in particular, based on modifications of the classical algorithm A* [Russell tt al 2006]. However, due to the known NP-complexity of such algorithms, the route search time on the map may be sufficiently high with a large graph dimension and this is practically unacceptable for the IRTS. Therefore, it is advisable to use algorithms from the class of anytime algorithms to find a solution in IRTS.

Anytime algorithms are algorithms, the quality of the found results monotonically improves with an increase in the operation time of the algorithm. The first solution is usually obtained in a relatively short period, after which the solution gradually improves, approaching the optimal [Thayer tt al 2010].

Russian Advances in Artificial Intelligence: selected contributions to the Russian Conference on Artificial intelligence (RCAI 2020), October 10-16, 2020, Moscow, Russia

eremeev@appmat.ru (A.P. Eremeev); poliushkinia@yandex.ru (I.A. Poliushkin); panyawin2010@yandex.ru (N.A. Paniavin)

It is proposed to use a non-relational database (DB) instead of a classical relational DB based on SQL to store the terrain map as a graph. The primary purpose of non-relational models of data representation and corresponding DBs is an extension of features of relational models and databases for those applications where they are not flexible enough. This fact is especially actual in the presence of poorly defined initial information. It is typical for IRTS, which is intended to support decision-making by operational and dispatching personnel, to manage a complex technical/technological object or transport object in problematic (abnormal, emergency) situations [Bashlikov 2011]. The advantages of NoSQL DBs and technologies are also high performance and ease of handling specific non-strictly fixed data structures.

In the context of the problem of routing and applying of DB for storage of graph structures (graph DB) based on NoSQL, we should also note the advantage of non-relational data models such as relationship storage level. In a relational DB, relationships are stored at the level of generalized tables, which is convenient when handling a large number of fixed-structure records of the same type. In opposite, the graph database stores relationships at the level of individual records, since each record must be processed at the time of the query individually to determine the structure of the stored data [Neo4j GSG]. Further, the links between the elements (nodes) in the graph database are objects of the database layer, so that no time spends for calculating these links when executing the query, as in the relational database. One more important advantage of a graph DB based on NoSQL over relational is that time of processing of query in the relational data model depends on the number of records while in graph DBs, it practically is not. This is important for IRTS, and when processing big data sets. Concerning IRTS of the decision support type, we should also note the perspective of NoSQL-based implementation of the temporal database supporting dynamic data processing and non-monotonic output (solution search) using temporal logic, in particular, Allen interval logic [Eremeev and Mitrofanov 2011, Eremeev tt al 2011].

The paper considers the possibilities of using integrated environments based on anytime heuristic search algorithms and non-relational (NoSQL) databases in the IRTS. As an example we choose the problem of routing between two points on the ma. The search space is represented as a graph.

2. NoSQL solutions for representation of graphs

Graph databases primarily created to solve problems with closely related source data by different relationships in areas where relational DBMS are not effective enough. Graph nodes use as entity stores, and edges use to store relationships between them [Neo4j CM]. There are no restrictions on the number and types of relationships a node can have. Graphs are well suited for the storage and processing of semantic networks. As an example, we can mention social networks or a map of routes around the city can.

Various DBMS may be used to implement a graph database based on the NoSQL concept. Let's look at some of them in context applications in IRTS. For example, the rather well-known non-relational Neo4j graph database created by Neo Technologies using Java and Scala languages. This graph DBMS uses its data format adapted for storing graph information. For comparison with classic relational DBMS, this approach allows for additional optimization in

the case of an elaborate presentation structure. The number of nodes is limited to 234.

The graph DBMS Neo4j provides a simple, flexible, yet powerful data model easily structured according to the requirements of the task. DBMS is optimal for heavy related entities (in a relational model, records link by many-to-many relationships) that can be easily handled by platform tools. Neo4j is also suitable for linked and semi-structured data. Neo4j provides a declarative Cypher query language for the visual representation of graphics using ASCII-art syntax. Commands in this language are similar to SQL commands, so they are effortless to master. Unlike the query language in relational DBMS, the query structure does not require expensive merge operations to retrieve data. The query language allows you to retrieve information about a neighbor or link (edge) without using joins or indexes.

In Neo4j, there are two types of caching: file and object. The file cache designed to handle data on the hard disk to increase the speed of reading and writing to the hard disk. The object cache stores the graph objects themselves (vertices, edges, and their properties) in a unique format, which allows increasing performance when bypassing the graph. A significant difference between Neo4j and relational DBMS is that there is no need to set primary or external key constraints for data. There are also no restrictions on relationships. You can specify any relationship between the two nodes. The nodes themselves can contain an arbitrary set of data. The flexibility of the Neo4j makes it more attractive to work with network data (social networks, transport routes, etc.).

Neo4j provides fast solutions on large volumes of processed information but requires more resources from the system. At the same time, a new task appears before the programmer. Choosing between fast resolution and limited disk space. At the moment, Neo4j is one of the most common graph databases supporting the NoSQL concept.

OrientDB, in turn, combines the capabilities of the document-oriented and graph-based DBMS, which stores data in table-based models. DBMS has three data schemes that allow for the storage of poorly structured data (schema-less), strictly structured (schema-full), mixed data (schema-hybrid). This set of diagrams provides the flexibility of the model itself. Each table is considered a separate class. Each class has some reserved names, such as a primary key. There is also a set of required fields, such as parent node ID, flag, type, and others (in Neo4j they are registered by default). OrientDB also does not use expensive merging operations. Instead, fixed pointers between records used. DBMS positions itself as a multimodel system provides not just as a graph model system [OrientDB].

The new stage in the evolution of non-relational graph DBMS is TigerGraph [TigerGraph]. It provides a cloud-based solution and is capable of parallel processing and analysis in real-time. Additional development is the ability to detect impractical and cumbersome connections in terms of information processing. Language of inquiries SQL-like. The most common degree of data compression under resource constraints is x10. This compression accelerates query performance and reduces memory usage and cache.

In this work, the NoSQL DBMS Neo4j-Desktop is chosen for the investigation of anytime algorithms for finding solutions to the routing problem on the model graph. In subsequent research we are planning computer modeling using the OrientDB and TigerGraph databases to obtain comparative characteristics of anytime search algorithms and temporal database implementations in terms of their application for the IRTS.

3. Anytime algorithms

Several anytime algorithms are known for informed search on the graph. For example, Anytime Window A*, Anytime Restaring A*, Anytime Continuing A*, Anytime Repairing A*, and others [Thayer tt al 2010]. In this work, algorithms obtained from classical algorithms using restarting, continuing, and repairing methods are analyzed both in terms of their use to solve the given routing problem and to find a solution in IRTS as a whole.

These methods are based on a heuristic function. This function may or may not be admissible. Let us remind that a heuristic function called admissible unless it overestimates the cost of the path from any vertex to the target, otherwise called non-admissible. A non-admissible heuristic function can be derived from an admissible one by multiplication on some number $w > 1$ [Hansen tt al 2007].

The restarting method is that the solution obtained by the A* algorithm (or another informed search algorithm on the graph) performs not once, but several. The first search performs using a non-admissible heuristic function without making any changes to the search process. Once a solution is found, its cost is remembered and used in subsequent search repeats as follows: vertices for which the estimate using a proper heuristic function is not less than this value not considered. This step is justified by the fact that moving to such vertices can lead to a goal only along a path that is not shorter than the previous one, which follows directly from the definition of a proper heuristic function. Excluding such vertices from consideration allows the algorithm to find a shorter path to the target. If the free vertex list becomes empty on the next repeat, it means that the algorithm is no longer able to improve its decision. At this point, the algorithm ends.

The continuing method is generally similar to the restarting method. The difference is that between the repeats of the algorithm, the lists of closed and open vertices not cleared. That is, the algorithm continues to find a solution from the same state as it was at the time of the previous solution.

The repairing method adds two modifications to the continuing method. The first modification is an introduction of an additional list of vertices. This list contains the vertices to which the search found a new path instead of their rediscovering. Once the next solution found, these vertices transferred to the open list. This fact makes it possible to speed up the search for another solution, avoiding multiple opening of vertices in graphs in which there are many close alternative paths. The second modification is that the parameters of the algorithm change with the improvement of the path. For example, reducing the weight of the heuristic function.

The described modifications can be applied to the method of continuing both together and separately.

Classical search algorithms use the heuristic function to estimate the cost of a path passing through vertex n:

$$f(n) = g(n) + h(n) \tag{1}$$

where $g(n)$ – the cost of the path traveled from the start vertex and $h(n)$ – the admissible estimation of the value of the path remaining to the target.

A non-admissible estimate may be as follows:

$$f(n) = g(n) + w * h(n) \quad (2)$$

where $w > 1$ – weight of the heuristic function $h(n)$.

All three methods can be applied to the A* algorithm with this pair of heuristic functions.

It is proven that if the heuristic function $h(n)$ is admissible, the path found using the weighted heuristic function is no more than w times longer than the optimal one. This fact makes it possible to estimate the worst drop in the quality of the first path proposed by the algorithm relative to the use of the algorithm, which straightway finds the optimal solution.

Note that this estimate is not fair for the repairing A* algorithm because modification made to the first solution search process.

We have chosen anytime A* algorithms with restarting, continuing, and repairing to solve the problem.

4. Research of anytime algorithms in the NoSQL database environment

As noted earlier, the most widely used and convenient environment for working with network data, the DBMS NoSQL Neo4j-Desktop is the environment for implementing anytime solution search algorithms.The basic Neo4j configuration provides more than 450 algorithms for graph processing. It is also possible to complement the built-in libraries with unmanaged server extensions, but their execution requires more precise control by the programmer [Neo4j JDR]. In more detail, each of the search algorithms can be analyzed by calling the *apoc.help*() command.

The test bench has the following configurations:

- HDD 1 Tb 5400 rpm
- Intel Core I5 2,3GHz(Turbo 2,8GHz)
- RAM 16 Gb ddr3
- nVidia GeForce 940M - 2048 Mb
- OS Windows 10 x64
- Neo4j v.4.0.3

The constructed test set consists of 100.000 nodes and 1.000.000 edges; the total size is 1704 MB; data import time is 27 seconds, peak load on RAM 10.9/15.6 GB.

As part of the study, a dependence of the quality of the obtained solution on the time (in milliseconds) of the search algorithm was obtained. The solution quality is evaluated as the ratio of the length of the currently found route to the length of the optimal route.

Figure 1 shows the time dependency graphics of the path quality found by the Restarting A* algorithm for the different weights of the heuristic function.

The x-axis of this graphic is an amount of passed time and the y-axis is the quality if the solution found by that time.

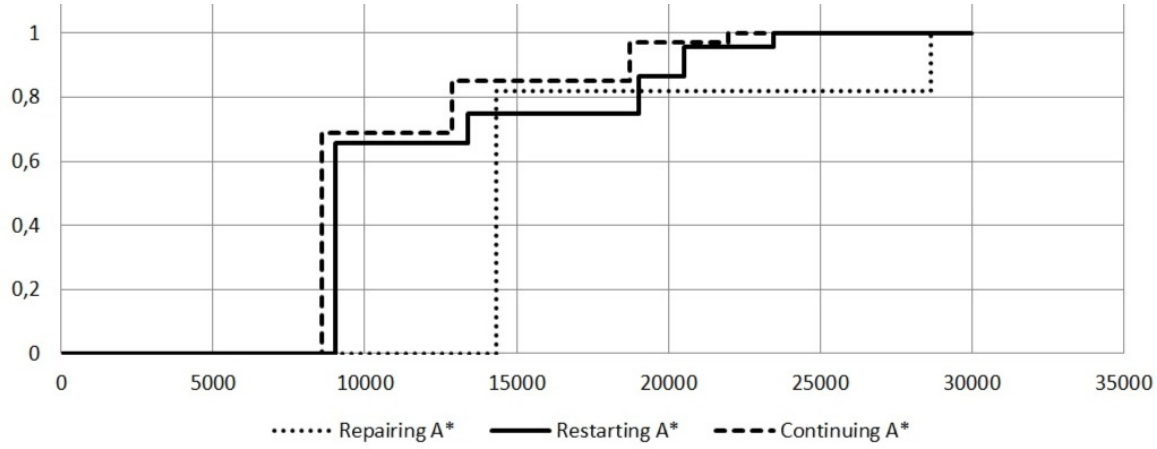

Figure 1: Time dependency (ms) of quality of the path found by Restarting A* algorithm for different weights of heuristic function

As can be seen from the graph, the best result is reached with weight $w = 5$. It provides the fastest finding (more than 3 times faster) of the first route, with this route having relatively high quality. The simulation showed that the Repairing A* finds the optimal route in 24 seconds, which is 8 seconds faster than the search by the classic A* algorithm.

Let us compare the route search process with different anytime versions of A* algorithms (with repairing, with restarting, with continuing) with weight $w = 5$. The path quality as a function of search time graphics are presented in Figure 2.

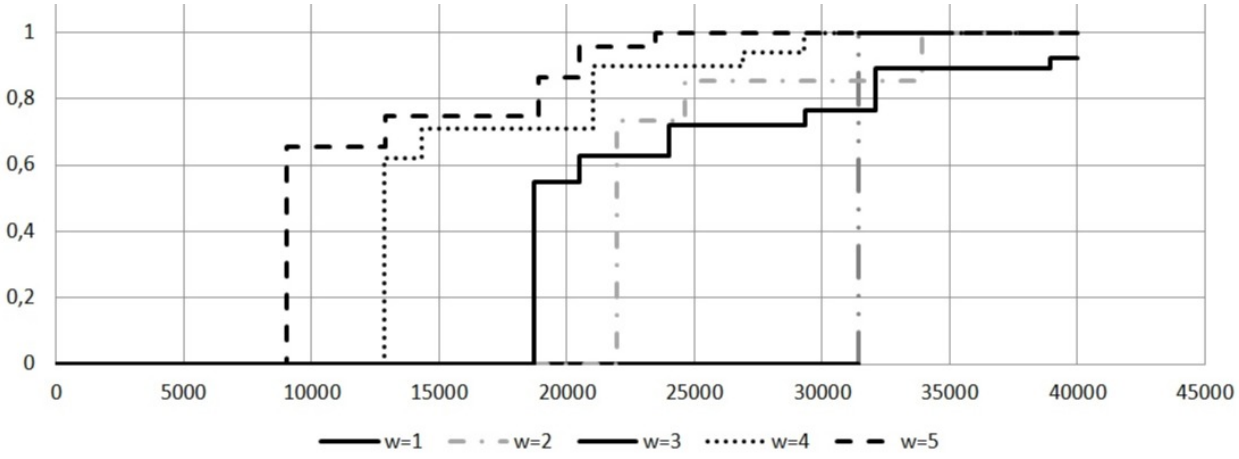

Figure 2: Comparison of solution search for different anytime versions of A*

From this graph, we can conclude that using the Continuing A* algorithm version works better than Restarting one, so it can be recommended as the most preferred solution for routing

problems for search spaces represented as graphs.

5. Software implementation

The obtained results form the basis of a software package focused on functioning both as part of IRTS or autonomously. The architecture of the complex is shown in Figure 3.

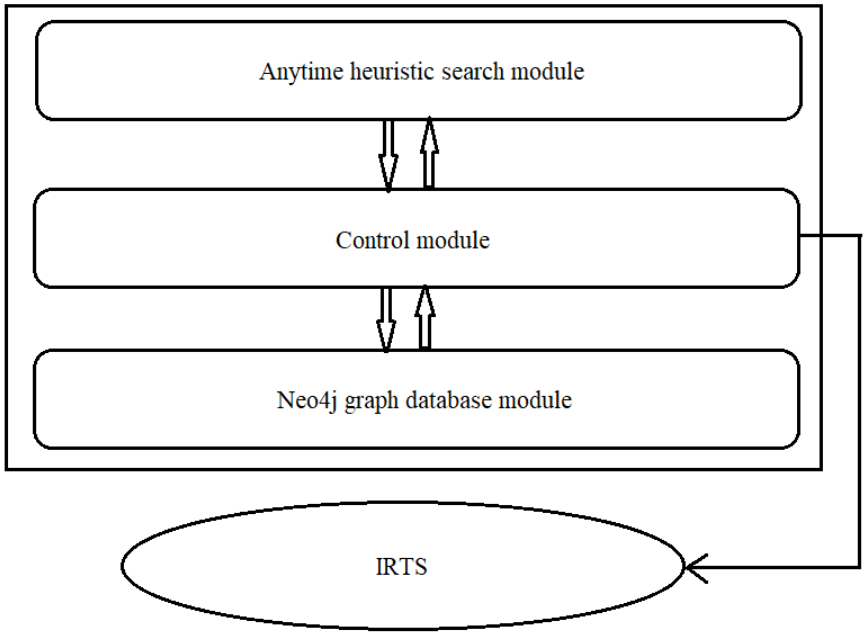

Figure 3: Program complex architecture

Anytime heuristic search module provides a software implementation of the considered anytime heuristic search algorithms. The control module implements the interaction of the modules and the interface with the IRTS. NoSQL (Neo4j) graph database module is a software implementation of NoSQL DB based on the Neo4j-Desktop NoSQL DBMS discussed above or in the future using the OrientDB or TigerGraph DB. The state space graph is stored in this database.

Loading and updating of states, transitions between them and communication with anytime algorithms for finding a solution is carried out using the control module. Upon receipt of a request for a solution from the IRTS, the control module activates the execution of the corresponding anytime algorithm from Anytime Heuristic Search module, and also provides an interface for the algorithm to retrieve information from the Neo4j graph database module in response to requests to the state space graph. When a solution is found, the control module sends a message to the IRTS and the best solution currently found. Depending on the response of the IRTS, the search process can be stopped (if the solution which found suits) or continued

in order to improve the solution. When finding the best possible solution for the given time constraints (up to the optimal one), the control module informs the IRTS about this and stops the search.

We should note that Anytime heuristic search module is a library of anytime heuristic search algorithms implemented in C++. Using this programming language allows you to search faster than using implementation of a search in the Java language. The UML class diagram of the library of anytime heuristic search algorithms is shown in Figure 4. All algorithms are inherited from the SearchBase base class, which provides a unified interface for calling the search algorithm. Due to the use of inheritance and the mechanism of virtual methods (polymorphism), various search algorithms can be called without modifying the code of the calling function.

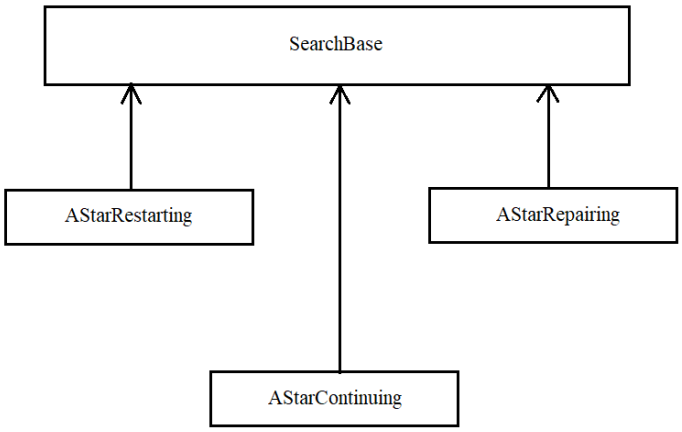

Figure 4: Class diagram of the Anytime heuristic search module

We should note that the implemented library is an independent module and can be used in IRTS independently of NoSQL database in solving problems for which storing of the search space is not required.

6. Conclusion

A research of anytime solution search algorithms using an example of a routing problem on a model graph has shown that anytime A* algorithms with restarting and continuing implemented (integrated) in a database-NoSQL environment apply to solve the routing problem on a map.

One possible practical application of the algorithms may be quickly searching for a new route when the vehicle deflects from the route. The quickly found first route allows the vehicle to be led in the right direction until the optimal route is found. Another possible application is to estimate the length of the route quickly. Since the first route found is no more than w times

longer than the shortest route, the length of the shortest route encloses between the length of the first route and the length of the first route divided by w.

Computer modeling has shown that the integration of anytime solution search algorithms with non-relational NoSQL Db is promising for their application in IRTS with sufficiently strict time constraints and noisy input data. In this regard, a significant advantage of anytime search algorithms is also the faster finding of the optimal path (with the necessary time resources allocated) compared to classical search algorithms.

We should notice that non-relational data models and related NoSQL databases, for example, multimodel OrientDB and graph TigerGraph, which can provide parallel processing and data analysis in the RT mode, can be used as the basis for the implementation of the temporal database for the IRTS. The example of such systems is the IDSS RT for monitoring and controlling nuclear power facilities [Bashlikov 2011].

Acknowledgments

The work was supported by RFBR projects 20-07-00498 a, 18-01-00201 a and 18-01-00459 a.

References

[Eremeev and Mitrofanov 2011] A.P. Eremeev, D.A. Mitrofanov. An anytime algorithm of hierarchical reasoning for real-time systems. *Intelligent Systems.* Issue 5. Moscow, Physmathlit. 2011. p 85-110. (in Russian)

[Russell tt al 2006] S. Russell, P. Norvig. *Artificial Intelligence: Modern Approach, 2-nd issue.* Williams, 2006. (in Russian)

[Thayer tt al 2010] J. Thayer, W. Ruml. Anytime Heuristic Search: Frameworks and Algorithms. *Proceedings of the Third Annual Symposium on Combinatorial Search (SOCS-10).* 2010. P. 121-128.

[Bashlikov 2011] A.A. Bashlikov. *Fundamentals of designing intelligent decision support systems in the nuclear power industry: textbook* / A.A. Bashlikov, A.P. Eremeev , INFRA-M, 2018, P. 351 (in Russian)

[Neo4j GSG] The Neo4j Getting Started Guide v4.0. 2020. P. 3-6 https://neo4j.com/docs/pdf/neo4j-getting-started-4.0.pdf, last accessed 2020/05/01.

[Neo4j CM] The Neo4j Cypher Manual v4.0. 2020. P. 2-3 https://neo4j.com/docs/pdf/neo4j-cypher-manual-4.0.pdf, last accessed 2020/05/01.

[OrientDB] OrientDB Manual. P. 5-12 , P. 84-90. https://orientdb.com/docs/last/OrientDB-Manual.pdf, last accessed 2020/05/01.

[TigerGraph] TigerGraph open documentation https://docs.tigergraph.com, last accessed 2020/05/01.

[Hansen tt al 2007] E. A. Hansen, R. Zhou. Anytime Heuristic Search *Journal of Artificial Intelligence Research 28.* 2007. P. 267-297.

[Neo4j JDR] The Neo4j Java Developer Reference v4.0. 2020. P. 17-24 https://neo4j.com/docs/pdf/neo4j-java-reference-4.0.pdf , last accessed 2020/05/1.

[Eremeev tt al 2011] A.A. Eremeev, A.P. Eremeev., A.A. Panteleev. Opportunities for implementing a temporal database for intelligent systems, *Software products and systems*, #2, 2011, pp. 3-7.