

VILNius gediminas technical university

faculty of fundamental sciences

department of information technologies

Elmira Sultanova

Using Data-Oriented Design (DOD) for Algorithm Optimization

Final Bachelor Thesis

Information Technologies Study Programme, State code 6121BX033

Finance specialization

Study Field of Informatics

Vilnius, 2025

**Table of Contents**

[1. Introduction 3](#_Toc188479119)

[1.1. Relevance 3](#_Toc188479120)

[1.2. Objectives and tasks 4](#_Toc188479121)

[1.3. Methodology Overview 4](#_Toc188479122)

[1.4. Practical Uses 4](#_Toc188479123)

[2. Research and Literature Review 5](#_Toc188479124)

[2.1. Data oriented Design 5](#_Toc188479125)

[2.1.1. Advantages of Data Oriented Design 5](#_Toc188479126)

[2.1.2. Drawback of Data Oriented Design 5](#_Toc188479127)

[2.2. Memory Access and Bandwidth Optimizations 6](#_Toc188479128)

[2.2.1. Loop Tiling (Blocking) 6](#_Toc188479129)

[2.2.2. Prefetching 7](#_Toc188479130)

[2.3. Single Instruction Multiple Data (SIMD) 7](#_Toc188479131)

[2.3.2. Vectorization 8](#_Toc188479132)

[2.3.3. Advantages of Vector processing 8](#_Toc188479133)

[2.3.4. Disadvantage of Vector processing 8](#_Toc188479134)

[2.4. Relevant Literature 9](#_Toc188479135)

1. Introduction

As systems, users, and data grow, there is a need to optimize algorithms to increase efficiency and speed. Pathfinding algorithms are no exception, as they are crucial in fields such as game development, navigation systems, robotics, and artificial intelligence where efficiency is essential (Patras et al., 2021; Rafiq et al., 2020). One of the most important aspects to consider when optimizing an algorithm is data management. Data Oriented Design (DOD) addresses this issue by prioritizing data while programming, by allowing to effectively handle memory access, cache, and data (Dang, 2023).

A\* (A star) is one of the most used pathfinding algorithms. It is faster and more effective than other algorithms because it uses a heuristic function to 'predict' the path (GeeksforGeeks, 2024). Despite its effectiveness as maps become larger, it requires a large amount of processing power and memory (Foead et al., 2021). This research paper will focus on optimizing the A\* algorithm to display the use of DOD in algorithm optimization.

* 1. Relevance

In modern systems, efficient use of computational resources is very important as it has a direct influence on performance, energy usage, and scalability. Optimizing algorithms may help to reduce energy consumption, improve performance, speed, and enable scalability. However, it is difficult to apply optimization as it might compromise the accuracy of the algorithm. Additionally, the increased robustness of the algorithm can lead to a larger use of computational resources (Penev et al., 2024; Sacthesw, 2024). Henceforth, applying DOD might help to balance performance with the usage of memory by prioritizing the data handling and management.

The A\* algorithm, its use, and different techniques of optimization have been widely researched. Existing research focuses on improving the pathfinding, eliminating the number of nodes, smoothing the path, and improving heuristics by adding new factors (L. Liu et al., 2022; Tang et al., 2021; Sun & Li, 2016). These approaches are largely focused on introducing new and more effective functions, however, there is lack of focus on handling data and managing memory. Although DOD has been researched in different contexts, such as multi-threaded video games and optimizing deep networks, its uses for algorithm optimization remain unexplored (Shah et al., 2020; Wingqvist et al., 2022).

* 1. Objectives and tasks

The aim of this research is to analyse the DOD approaches in optimizing algorithms by applying these principles to the A\* algorithm and measuring its performance.

1. Implementing a map and A\* algorithm in C# with SFML (Simple and Fast Multimedia Library): Create a visual map with obstacles and the classical A\* algorithm that can navigate through the map
2. Identifying and applying DOD in C#: Analyzing the ways of data reconstruction and access based on DOD
3. Applying the identified principles to the A\* algorithm: Creating variants of the A\* algorithm with modifications according to DOD
4. Comparing the performance of different A\* algorithms: using BenchmarkDotNet to compare the performance
5. Analysing the results: Comparing how each version of the A\* algorithm performed and evaluating the use of DOD in optimizing algorithms
   1. Methodology Overview

The methodology consists of implementation and benchmarking. Firstly, a classic A\* algorithm is implemented with visuals to demonstrate the path that was found. Second, techniques for Memory Access and Bandwidth Optimizations Temporal such as tiling, memory prefetching and Computational Optimization methods such as SIMD using vectorization are used to optimize the A\* algorithm, focusing of data handling, memory access, time and cache efficiency. BenchmarkDotNet will be used to evaluate different variants of the algorithm. The results will be compared to evaluate the effectiveness of DOD in optimizing algorithms.

* 1. Practical Uses

The methodologies and findings in this research have many practical uses in computer science and engineering. In game development, the A\* algorithm is used to create non-playable characters (NPC), optimizing the algorithm can increase the performance of video games with many NPCs (Candra et al., 2021). Furthermore, in robotics, the A\* algorithm is used to help autonomous robots move, the enhanced algorithm will enable them to navigate through dynamic environments more efficiently and allow further advancements in automation (Ducho et al., 2014). Furthermore, the findings of this research can be applied to improve navigation applications in which the A \* algorithm is used to find paths through the city (Veisi et al., 2023). Moreover, optimizations focusing on the data can be very useful in modern datasets that keep growing and complexity for systems where performance and efficient use of computational power are important.

1. Research and Literature Review
   1. Data oriented Design

Data oriented design is a way of programming that mainly focuses on how data in handled by viewing it as the main part of any application or system. It enables performance optimization, efficient organization of data and make scalability easier. The main principles of DOD are the following:

* An application should be made around data, access patterns and relationships in regards to data
* The way data flows through the system should be made clear, focusing on how data moves between components
* Elements should have the least dependence on each other to enable easier parallelization
* Design should focus on speed, improving cache locality and minimal bottlenecks

(Dang, 2023b)

* + 1. Advantages of Data Oriented Design
* Simpler Parallelization – since data-oriented design encourages simpler and less dependent components it is possible to apply threading with minimal amount of synchronisation
* Efficient use of cache – in DOD the same code is ran many times and data is processed in blocks this means the instruction cache is used very efficiently
* Easier to test – as the flow of data is managed and parts of the program are loosely dependent writing unit tests for different components of the program becomes simpler (Noel, 2009)
  + 1. Drawback of Data Oriented Design

The drawback of DOD is that it is not intuitive, and many developers are not user to implementing it. This means that it maybe difficult to read the code and difficult to apply it with existing object-oriented design.

(Noel, 2009)

* 1. Memory Access and Bandwidth Optimizations

Memory access is the method of how the data is stored and retrieved from computer’s memory which is executed by memory controllers. Memory bandwidth is the rate in which the data can be written and read form the device memory. Both are connected as efficient memory access leads to faster bandwidth. The following are methods that can be used to improve memory access and bandwidth.

* + 1. Loop Tiling (Blocking)

Tiling is the process of dividing the threads into smaller parts to improve speed. In loop tiling the loop nest is divided into different iterations which means the data in is processed a piece at a time. This is applied so that the data segments can fit into cache blocks. Since it is very fast for the system to access cache memory with the right tile size the memory can be greatly improved. (*Loop Tiling | Code Guidelines for Correctness, Modernization, and Optimization*, 2025)

For example, the method shown can be used to transpose a matrix:

 public bool Transpose(double[,] input, double[,] output, int size)

    {

        for (int i = 0; i < size; i++)

        {

            for (int j = 0; j < size; j++)

            {

                output[i, j] = input[j, i];

            }

        }

        return output[0, size - 1] != 0;

    }

* + - * 1. Method to tranpose a matrix

This method causes no problems when using a small matrix, as the data can be stored in cache. However, if bigger matrix were to be used it were to go beyond the cache size the data would be stored in the main memory. It takes more time for the CPU to access the main memory. In the following code, tiling is used making sure that the matrix is processed in smaller pieces.

 public bool Transpose(double[,] input, double[,] output, int size)

    {

         const int TileSize = 16;

        for (int ii = 0; ii < size; ii += TileSize)

        {

            for (int jj = 0; jj < size; jj += TileSize)

            {

                for (int i = ii; i < Math.Min(ii + TileSize, size); i++)

                {

                    for (int j = jj; j < Math.Min(jj + TileSize, size); j++)

                    {

                        output[i, j] = input[j, i];

                    }

                }

            }

        }

        return output[0, size - 1] != 0;

    }

* + - * 1. Method to transpose a matrix optimized using tiling

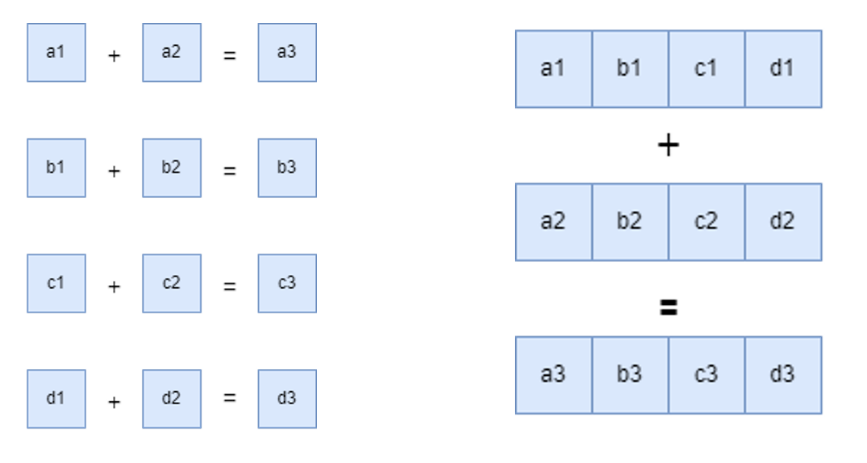
As shown the matrix is now processed in smaller sizes defined by the ‘TileSize’. The data that is in tiles can be stored in cache memory and therefore processed faster.

* + 1. Prefetching

Prefetching is the process of fetching the data before it can be used by the system. It is used to improve the number of cache hits which is when the data or the instruction that is needed is already available in cache. To apply prefetching spatial locality and algorithmic locality are considered. Spatial locality highlights that the data that will be used next are near the data that is used now. Furthermore, Algorithmic locality points out that the data that is going to be used by an algorithm should be. Applying these principles and feting the data before it is needed can lead to less access latency during execution and improved bandwidth, as the data can be fetched when the bandwidth usage is lower. (*What Is Cache Prefetching*, n.d.)

* 1. Single Instruction Multiple Data (SIMD)

SIMD operation in a method that allows multiple data to be processed with a single instruction, unlike sequential approach where each data is processed requires it’s own instruction. For example instead of executing an add operation separately for each data. (Barry & Crowley, 2012) A single add operation can be used for all the data as shown in Figure 3.



* + - * 1. SIMD visualized
    1. Vectorization

There are multiple ways to apply SIMD, modern compilers identify loops that can be converted and operated on a single set of instructions. However, this intrinsic process is not always sucessful, henceforth vectorization process can be guided within the code. (Barry & Crowley, 2012) This can be accomplished by using the Vector which is a dynamic array that stores elements of the same data structure. CPU has vector processors. Vector processors are designed to handle multiple data at a time resulting in significant increase in performance. (GeeksforGeeks, 2024b)

* + 1. Advantages of Vector processing

There are many advantages to vector processing to achieve parallelism:

* Productively moves information – vector processors can transport data effectively and limit memory bottlenecks
* Less time and computational resources spent – it takes time and resources to get, unravel and execute instructions
* Saving energy – processing multiple instructions at a time uses less energy compared to singular instructions (GeeksforGeeks, 2024b)
  + 1. Disadvantage of Vector processing

The drawback of this method is that it is difficult to implement in practice and may make the code more complex and difficult to understand.

* 1. Relevant Literature

Evaluating the performance of object-oriented and data-oriented design with multi-threading in game development

DOI: 10.1109/gem56474.2022.10017610

This article analyses the difference between how Object-Oriented Design (OOD) and Data Oriented Design (DOD) perform in game development. The study had a basic game which applied OOD and DOD principles. After the performance of the implementations were measured. The researchers kept track of time, CPU and memory used. The study then applied multithreading to both implementations and measured the performance. The study concluded that in both single threaded and multithreaded versions the DOD approach was more efficient due to better CPU memory utilization.

Research on Path-Planning Algorithm Integrating Optimization A-Star Algorithm and Artificial Potential Field Method

DOI: 10.3390/electronics11223660

The article focuses on optimizing A\* algorithm by combining it with artificial potential field method and least squares method. The aim was to enhance path finding in cleaning robots.

The researchers optimized the A\* algorithm and compared it to the traditional A\* star, ant colony and RRT algorithm. Path planning time, path smoothness and response time was measured. The research concluded that the optimized A\* algorithm performed 60% better than the traditional and 65.2% better than the bidirectional A\* algorithm. Additionally, the paths were smoother and more continuous.

References

Barry, P., & Crowley, P. (2012). Digital signal processing using General-Purpose processors. In *Elsevier eBooks* (pp. 317–346). https://doi.org/10.1016/b978-0-12-391490-3.00011-4

Candra, A., Budiman, M. A., & Pohan, R. I. (2021). Application of A-Star algorithm on Pathfinding game. *Journal of Physics Conference Series*, *1898*(1), 012047. https://doi.org/10.1088/1742-6596/1898/1/012047

Dang, T. (2023a, October 3). *Revolutionize your code: the magic of Data-oriented Design (DOD) programming*. Orient Software. https://www.orientsoftware.com/blog/dod-programming/

Dang, T. (2023b, October 3). *Revolutionize your code: the magic of Data-oriented Design (DOD) programming*. Orient Software. https://www.orientsoftware.com/blog/dod-programming/

Duchoň, F., Babinec, A., Kajan, M., Beňo, P., Florek, M., Fico, T., & Jurišica, L. (2014). Path Planning with Modified a Star Algorithm for a Mobile Robot. *Procedia Engineering*, *96*, 59–69. https://doi.org/10.1016/j.proeng.2014.12.098

Foead, D., Ghifari, A., Kusuma, M. B., Hanafiah, N., & Gunawan, E. (2021). A Systematic Literature review of A\* pathfinding. *Procedia Computer Science*, *179*, 507–514. https://doi.org/10.1016/j.procs.2021.01.034

GeeksforGeeks. (2024a, July 30). *A\* search algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/a-search-algorithm/

GeeksforGeeks. (2024b, October 12). *Vector processor classification*. GeeksforGeeks. https://www.geeksforgeeks.org/vector-processor-classification/

Liu, L., Wang, B., & Xu, H. (2022). Research on Path-Planning algorithm integrating optimization A-Star Algorithm and Artificial Potential Field method. *Electronics*, *11*(22), 3660. https://doi.org/10.3390/electronics11223660

*Loop tiling | Code Guidelines for Correctness, Modernization, and Optimization*. (2025, January 16). https://open-catalog.codee.com/Glossary/Loop-tiling/

Noel. (2009, December 4). *Data-Oriented Design (Or Why You Might Be Shooting Yourself in The Foot With OOP) – Games from Within*. https://gamesfromwithin.com/data-oriented-design

Penev, K., Gegov, A., Isiaq, O., & Jafari, R. (2024). Energy efficiency evaluation of artificial intelligence algorithms. *Electronics*, *13*(19), 3836. https://doi.org/10.3390/electronics13193836

Sacthesw, F. (2024, August 24). *Impact of computational resources on algorithm selection*. Future Machine Learning. https://futuremachinelearning.org/impact-of-computational-resources-on-algorithm-selection/

Shah, A., Wu, C., Mohan, J., Chidambaram, V., & Krähenbühl, P. (2020). Memory optimization for deep networks. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2010.14501

Sun, D., & Li. (2016). Evaluation function optimization of A-Star algorithm in optimal path selection. *Revista Tecnica De La Facultad De Ingenieria Universidad Del Zulia*. https://doi.org/10.21311/001.39.4.14

Tang, G., Tang, C., Claramunt, C., Hu, X., & Zhou, P. (2021). Geometric A-Star Algorithm: An improved A-Star algorithm for AGV path planning in a port environment. *IEEE Access*, *9*, 59196–59210. https://doi.org/10.1109/access.2021.3070054

Veisi, O., Du, D., Moradi, M. A., Guasselli, F. C., Athanasoulias, S., Syed, H. A., Müller, C., & Stevens, G. (2023). *Designing SafeMap Based on City Infrastructure and Empirical Approach: Modified A-Star Algorithm for Earthquake Navigation Application*. https://doi.org/10.1145/3615900.3628788

*What is Cache Prefetching*. (n.d.). https://www.ioriver.io/terms/cache-prefetching

Wingqvist, D., Wickstrom, F., & Memeti, S. (2022). *Evaluating the performance of object-oriented and data-oriented design with multi-threading in game development*. https://doi.org/10.1109/gem56474.2022.10017610