HU ISSN 1785-1270

PRODUCTION SYSTEMS AND INFORMATION ENGINEERING

A Publication of the University of Miskolc

VOLUME 8 (2019)

MISKOLCI EGYETEM
UNIVERSITY OF MISKOLC

MISKOLC UNIVERSITY PRESS

HU ISSN 1785-1270

PRODUCTION SYSTEMS AND INFORMATION ENGINEERING

A Publication of the University of Miskolc

VOLUME 8 (2019)

MISKOLCI EGYETEM
UNIVERSITY OF MISKOLC

MISKOLC UNIVERSITY PRESS

Secretariat of the Vice-Rector for
Scientific and International Affairs
University of Miskolc
Responsible for publication: Prof. Dr. Tamás Kékesi
Responsible for layout and editing: Dr. Attila Károly Varga
Published by the Miskolc University Press
under the leadership of Attila Szendi
Responsible for duplication: Erzsébet Pásztor
Number of copies printed: 70
Put to the Press on December 18, 2019
Number of permission: TNRT 2016-100-ME
HU ISSN 1785-1270

EDITORAL BOARD


Production Systems and Information Engineering
Volume 8 (2019), pp. 5–18
https://doi.org/10.32968/psaie.2019.001

SIMPLIFIED VOXEL BASED VISUALIZATION

Péter Mileff
University of Miskolc, Hungary
Department of Information Technology
mileff@iit.uni-miskolc.hu

Judit Dudra
Bay Zoltán Nonprofit Ltd., Hungary
Department of Structural Integrity
judit.dudra@bay-zoltan.hu

[Accepted December 2017]

Abstract. Voxel-based technology has already been present in the early years of computer visualization. However its usage was almost entirely confined to the field of medical image synthesis. Due to the continuous development of the CPU and graphics hardware, its role has increased again nowadays. It came to the fore especially in the field of computer games because of its specific properties. This publication presents a simplified visualization model and optimization techniques, which are able to visualize smaller (perhaps animated) voxel sets in real-time with certain compromises. The objective of the solution is not to reach the level of photorealistic image synthesis, but a provide a well applicable and simple method which can be applied for computer games and other graphical applications

Keywords: voxel based visualization, software rendering, optimization

1. Introduction

Almost the entire area of the computer visualization is dominated today by GPU rasterized polygon models. Although the voxel-based approach has been available from the very beginning, but the early slow hardware were not yet ready in performance for an approach based on atomic model structure. They were strongly limited in memory and storage, so it is no wonder that the filling based (scanline, half space approaches) polygon image synthesis has become dominant.

In the early ages, the process of rasterization was performed entirely and exclusively by the CPU, the graphics accelerator hardware appeared only later. However, the technology has been continuously evolved over the years, mainly due to the computer games. Today it seems that voxel based technology is a major trend in the area of photorealistic visualization. Besides the growing of hardware performance, the expectation about visualization have also increased. A modern computer game has millions of polygons on the screen at the same time. All of the voxelized versions of these would consume a lot of memory and CPU/GPU time. Today's GPUs are able to render a smaller scene with various real-time effects (e.g. lighting, depth of field, shadows etc), but in cases where many models are on the screen, the available GPU memory is probably insufficient. This led to the unfolding techniques of real-time background data streaming into GPU memory.

To summarize, we can state that modern computer graphics stands before the introduction of the effective voxel technology. Companies, the greatest players in the computer game industry are constantly looking for voxel based complementary solutions to achieve a more realistic view. These techniques, algorithms are complicated, require a high level of knowledge from different areas. This article therefore does not focus on photorealistic visualization, rather displaying smaller voxel sets in a simple way. To answer the question how voxel sets can we visualize without a deep mathematical knowledge beside acceptable quality compromises.

2. Related works

The voxel-based, alternative rendering technology has already appeared in the early years of the computer visualization. Because early hardware did not allow to use displaying models which are able to provide professional, high images quality, thus voxel-based techniques became less known. The main area of its application was the medical diagnostic image synthesis, but some interesting computer games were born applying ray base solutions at the rendering. Although the CPUs were not yet fast enough, but owing to the so-called Wave Surfing algorithm, low-end hardware were able to visualize three dimension models (mainly terrains) in real time. Good examples are: Comanche - 1992, Delta Force - 1998, Armored Fist - 1994, Blade Runner - 1997, Hexplore - 1998 etc. The early consoles (Amiga, Nintendo, Gameboy) used similar technology with the help of software rasterization.

After the continuous spread of the GPU based rendering, which supported the polygon based approach, the software based (voxel) solutions fell more and more into the background. Although there have always been software products (Outcast - 1999, Motocross Stunt Racer - 2002, Red Alert series),

which used the voxel approach, the polygon-based technologies come to the fore.

In recent years, voxels have become to a frequented area again. Due to the evolution of the GPU, mainly the approaches using some kind of ray-based technology (ray-tracing, cone tracing etc.) are showing progress. Solutions applying tree structures for faster rasterization are also popular. In publication [3], one of the leader of the GPU market, the NVidia examines the possibilities of using voxel representations as a generic way for expressing complex and feature-rich geometry on current and future GPUs. Their benchmarks show that the octal tree based voxel representation is competitive with triangle-based representations in terms of ray casting performance. Szymon and Marc [4] describe a system for representing and progressively displaying complex meshes that combines a multiresolution hierarchy based on bounding spheres with a point based rendering system. In [11] sparse voxel database structure (GVDB) was investigated based on the voxel database topology. GVDB introduces an indexed memory pooling design for dynamic topology, and a novel hierarchical traversal for efficient raytracing on the GPU. They method can give large performance improvements over CPU methods.

Global illumination is a modern trend to make scene lighting more realistic. The voxel representation is becoming more important in this area. The leading company, Unreal Technologies presented a voxelisation based model for global illumination, applying Deferred rendering [6]. The importance of this technique lies in the fact that the technology is ready for the modern computer games. Paper [7] and [8] also approach the problem of modern lighting using voxel-based technology.

The voxel-based image synthesis is constantly evolving, it will be seen on the screen of every average computer within a few years.

3. Voxel based visualization

Voxel based visualization is not a new area in the field of digital image synthesis. The idea behind the name is that the model builds the 3D model from voxels instead of describing it by today's popular a polygon mesh. It can be difficult to define the term voxel. The literature often calls it as three-dimensional pixel, but can also be called as atom. A typical voxel representation contains the position, size and color information. In addition, renderers using different technologies can store other information (e.g. normal vector), which are used to achieve a more realistic visualization used (e.g. Ambient Occlusion). The voxel-based representation has several advantages over polygon-based storage model. Since the voxel set includes all necessary information of the model

for displaying, texture mapping and mip mapping are not needed. The color stored in voxels determines exactly the "look" of the model. Another advantage of the representation is that we can define a very detailed model structure, built upon atomic units. If observe the current trends in the computer game industry, we can declare that the voxel technology can be a very possible future of the computer graphics. This conclusion comes from the fact, that the global objective of real time computer visualization is to reach more realistic, physically correct results on the screen. Therefore the number of applied triangles are constantly increasing and their size is getting smaller. This is a process which tends to an atomic level, which is the world of voxels. Nowadays, screen space solutions (Normal Mapping, Occlusion Mapping, Parallax Mapping etc) are dominated against complex triangle based model structures, because using current screen space algorithms to add detail to a model are cheaper for the GPU, than working with millions of triangles.

The main downside of the voxel technology is hidden in the large data set. Even less detailed model structure has a relatively large set of voxels. Big data set of models requires large amount of main and GPU memory, which is fairly limited in case of GPU based visualization. Various, so-called streaming technology must be developed to keep the current possible visible parts of the virtual world in the GPU memory, however this technology needs a very complex computer and visualization knowledge. Further additional problem of the voxel set is the following: because the voxel based world comprises a large number of voxels, the graphics pipeline should handle large data sets during the various transformations. Because every CPU or GPU cycle counts, this property has a significant impact on performance [1]. Today's graphics hardware does not directly support the voxel based visualization. There are some trends emerged, mainly based on ray-based solutions, but there is no officially supported direction/pipeline from the GPU manufacturers and graphics APIs (OpenGL, DirectX) like in case of polygon-based solutions. using polygons it is enough to specify a set of vertices, textures, and other properties of the object, the GPU can visualize the data almost directly. To visualize a voxel set with the GPU, the programmer usually should develop a custom technology using (sometimes very complex) shaders performing some kind of simplified ray-based technology.

Today, the NVidia company provides a ray tracing engine called Optix, which is able to perform ray tracing on their GPUs. But since it is not officially supported by the common graphics APIs, its application possibilities are limited. The computer game industry will not use this technology until it is not part of the the graphical APIs (OpenGL, DirectX) and uniformly supported on all modern GPU.

The representation of the voxel set is independent of the displaying process. In practice, many volume rendering algorithms have been developed. This paper reviews the most important approaches and also presents a software based simplified solution.

3.1. Cube based approach

This approach is the easiest, we might say the naive approach to display voxel sets. During the visualization, all elements of the voxel set is represented by a three-dimensional cube on the screen. The size of the cube is usually predetermined. Although it is a simple technique, it is fairly popular in today computer games (e.g. Minecraft, FEZ, Stonehearth, Voxatron etc). There are several reasons for this: the visualization model is simple, easy to understand and the results are cubic having retro impressions.

Although the model seems to be simple, because basically every cube is specified by a specific color, in practice, displaying larger models (millions of cubes) cause many serious problems to the GPU due the high number of polygons. As typical example, the shadow creations process can be mentioned. If we use the popular shadow mapping technique, the models need to be renderer in multiple passes. Handling point lights are even more complex, because cube-map is required to store the 6 depth textures in case of only one light. The another approach, shadow volume has also problem with large data sets: this algorithm builds a coherent polygon mesh in real time from the world vertices which are visible from the light perspective. In order to achieve a reasonable frame rate, a number of additional optimization process (e.g. Space partitioning, Occlusion Culling, Ordering objects by z coordinate etc) should be introduced, which makes this technique also complex at the professional level.

3.2. Ray based solutions

An another popular rasterization form of a voxel-based world are the ray-based approaches. A simplified variant (Wave surfing algorithm-2D raycasting) of this technique has already been developed and used in early computer games (Comanche, Wolfeinstein, Delta Force, Armored Fist, Outcast etc) and medical diagnostic procedures. The main idea behind these approaches is that the rasterization problems and the hidden surface determination are solved independently by pixel to pixel. The algorithm casts rays through the render target (e.g. the screen) pixels to the virtual space, then it recursively examines their traversal, collision points and features. The big advantage of the procedure lies in its simplicity. The algorithm can solve numerous visualization problem, which are able to be achieved with current “forward” and “deferred”

Figure 1. Example of a cube based voxel visualization (Voxatron game)

rendering techniques, but using only a variety of additional techniques that require deep technological and mathematical knowledge (e.g. Cascaded Shadow mapping, Ambient Occlusion etc).

Raytracing and its variants (e.g. path tracing) is the primary starting point for today's global illumination rendering solutions. Today, many attempts unfolds, where the target of the experiment is to develop real time global illumination using voxel based representations. Example: Epic Games-Sparse Voxel Octree Global Illumination, ID Software - Sparse Voxel Octree, NVidia - Efficient Sparse Voxel Octrees [3].

Performing global illumination like ray tracing on a voxel set is a computationally intensive, slow process. Therefore it is essential to use accelerator structures, which are usually based on some kind of space partitioning tree (e.g. KD-tree, BSP tree etc.). Using these structures, the visualization process is the following: the casted rays are hit with the tree levels in order to speed up the voxel searching process, which color should be drawn on the screen. The main disadvantage of ray based solutions is that it is not supported by official graphics APIs(OpenGL, DirectX). Although the graphics pipeline is programmable via shaders, the GPUs are not designed to support the ray-based algorithms by directly by hardware. Today's fast GPUs are able to achieve some ray based rendering in real time. However these algorithms

are currently applied only in graphical demos, not in commercial projects (like games).

4. Simplified voxel rendering

The briefly described above techniques are not able to serve all demands. However, there are cases when we want to display smaller voxel sets with acceptable performance and with certain visual compromises (reduced shading/shadows etc.). Good examples are the two-dimensional computer games, where although the projection is two-dimensional, some simple pickable items, models are three-dimensional, they rotate or perform simple animation. Because the display does not want to rent a retro characteristic to the screen, using larger cubes can not be a solution. The rendered voxel set should reflect a two-dimensional characteristic on the screen, but stored in a 3D voxel model format. Figure 2 shows the nature of the approach.

Figure 2. Red Alert, 2D game with 3D voxel units

None of the above solutions are not fully capable of performing these type of rendering tasks. In case of the cube based approach, very small cubes (or possibly balls) could be used in order to achieve high quality game objects. Almost every cube would represent one pixel on the screen. In this case, an

unnecessary amount of vertex set is created, which would impose a serious load on the GPU.

The question may arise: why do we need a voxel set to represent a game object, why not implement polygons? In these cases, the polygon set itself would also be dense. But an another important cause is that the developers would like to use the characteristic of the voxel representation, that the object is built from atomic parts and therefore it is destructive.

In the following a simplified voxel rendering solution is presented, which is capable to perform the above-defined tasks efficiently.

4.1. The square based approach

To describe the approach of the simplified voxel renderer, let start from the logic of the displaying process. A voxel is a three-dimensional unit, its rasterisation generally requires a two-dimensional mathematical projection, where the corresponding pixels should be determined in function of the voxel color and size. However the question arises, that if the voxel is mapped to the two-dimensional space anyway, why bother the three-dimensional extent? Can the extent of the voxel be modeled in two dimensions only?

We can apply a simple square based mapping to solve the problem. This technique allows to visualize small voxel sets well in real time under certain compromises. The following figure shows this simple mapping:

Figure 3. The image shows a magnified mapping of 4 voxels. Two voxels are next to each other and two voxels are located behind

So, during the displaying process, voxels are represented as painted "tiles" (square/rectangle) with predetermined size. The color of the tile is the color determined by the voxel. The difference in x and y directions between the two rows is a natural feature of three-dimensional mapping. In the above example, the first row is located ahead in the space and the model isn't positioned in the origo.

The following pseudo code describes the mathematical model which maps a voxel into the two dimensional space.

```
var per_z = 1.0f / voxel.z;
var square_size = SIZE_FACTOR;

 x_s1 = + ((voxel.y - square_size) * per_z) + half_screen_width;
 y_s1 = - ((voxel.x - square_size) * per_z) + half_screen_height;
 x_s2 = + ((voxel.y + square_size) * per_z) + half_screen_width;
 y_s2 = - ((voxel.x + square_size) * per_z) + half_screen_height;
```

The result can be seen on Figure 4:

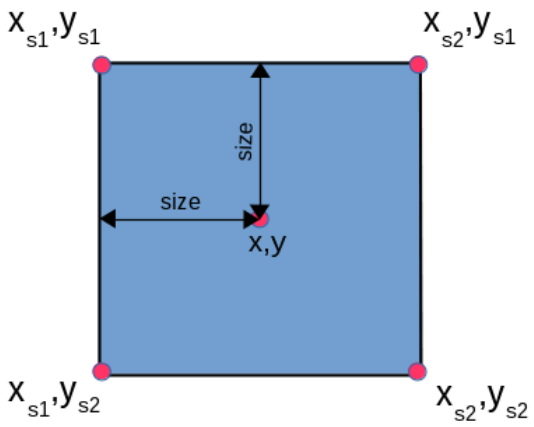

Figure 4. The shape of a screen mapped voxel

The (x, y, z) in the above formulas represent the spatial coordinates of a voxel, the `square_size` is an empirical parameter of the two-dimensional mapping, the size of the square and the x_{s1}, y_{s1}, x_{s2}, y_{s2} coordinates are the corners of the 2D square. `square_size` parameter can be configured according to the rendering requirements. Usually this parameter is constant. Its behavior is the following: if the voxel position is far from the view plane, the model will be smaller on the screen. Therefore we can adjust this parameter to a smaller number, because of the distance of the mapped voxel centers will be also closer and their shape will overlap each other. If the model is closer to the new view plane, the distance between the voxels will be larger. For this reason, the size parameter should be a bigger number, otherwise there will be gaps in the mapped model on the screen as the following image shows:

Figure 5. Gap between projected voxels

4.1.1. Rasterizing the voxels

The easiest way to draw a voxel is to use a software based rendering approach. Software rendering uses the power of CPU to create the final voxel graphics. During the rendering process each voxel is drawn into a framebuffer which is located in the main memory, then the buffer is passed to the GPU and will be displayed on the screen. The solution is not incompatible with the GPU-based visualization: the real objective in this case is to integrate the two rendering techniques. During or after the hardware rendered models, the software framebuffer can also be drawn.

The following code describes the drawing of a voxel:

```
for i = x_scr to x_scr2
  for j = y_scr to y_scr2
    index = i * frame_buffer_width + j;
    if (j > frame_buffer_width || i > frame_buffer_height || j < 0 ||
        i < 0)
      continue;
    if (voxel.z < zBuffer[index] )
       z_buffer[index] = voxel.z;
       frameBuffer[index] = voxel.color;
    end
  end
end
```

It is clearly visible, that the solution also requires a z-buffer implementation. The reason for this is that, due to the spatial location of the voxel set, the mapped squares can overlap each other in certain areas. The following image summarizes the result of this technique:

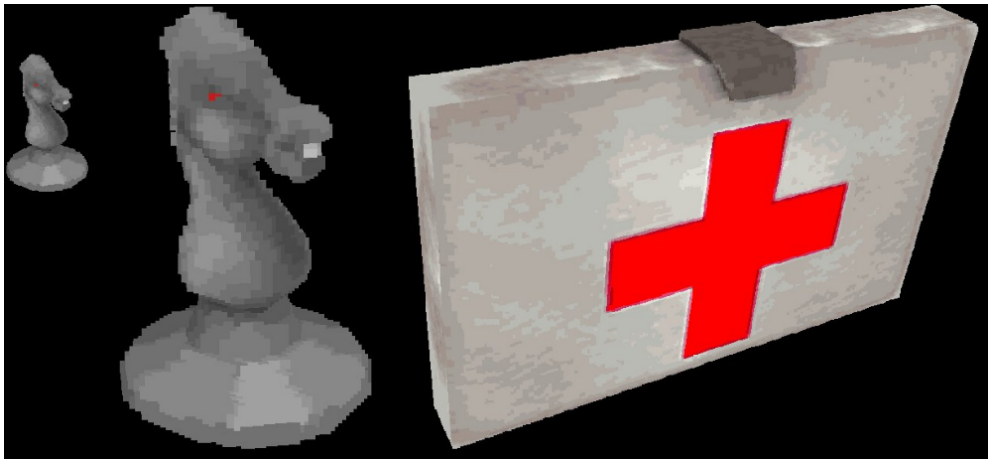

Figure 6. Voxel models in case of different z values

Figure 6 shows two models, where the first model appears twice with different z values. It can be seen that viewing the horse figure from a distance we can obtain satisfactory result, but if from closer the characteristics of the rectangular representation appear. The level of angularity can be eliminated by the tuning of the voxel density and its size. However the rendering time may increase significantly, as the rendering of the second models shows. The medkit model consist of 1.548.288 voxels. Obviously the visual quality is much better, but while 355 FPS was measured at the horse model (49.152 voxels), the performance was dropped to 63 FPS (without optimization) in case of rendering the medkit.

For this reason, this approach is mainly intended for rendering small models in real-time locating not too close to the camera.

5. Accelerating the rendering process

The procedure - although it does not contain any complex mathematical formula - is quite computationally intensive because of the double iteration loop. For every voxel an area of the framebuffer should be colored. The situation is much worse when the voxels are located in the way, that they need to be draw from back to front. In this case, due to the ordering of the voxels, the z buffer cannot reject the non-visible pixels and a lot of pixels will be continuously overwritten. Therefore this rendering technique requires some additional extension, to reduce the number of necessary iterations.

One of the most important optimization is to skip the non-visible voxels of the model. This requires to build a suitable representation for the model, which is capable to determine the outer voxels. During the rasterization the load of the renderer can be reduced significantly.

A further acceleration approach can be the following: if the position of the model changes along the z direction, after a certain z distance it makes no sense to draw squares into the framebuffer. Because of the distance, the boundary points of the mapped voxel are slipped to each other, therefore it is enough to assign a simple pixel to a voxel. With this extension, the performance can be also improved.

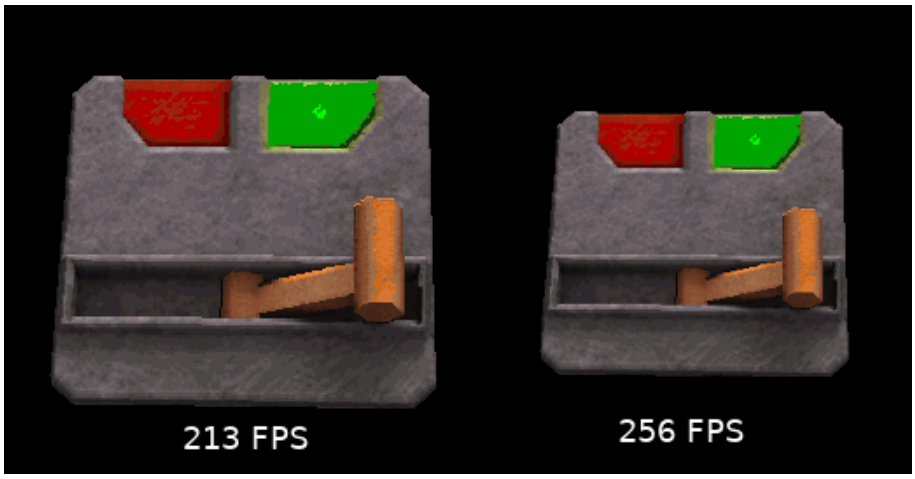

Figure 7. Significant jump in performance: left image is rendered as true squares, right image is rendered as points with different z distance. Number of voxels: 1.548.288

The redundant rasterization due to the formerly mentioned "unfortunately" order of rendering can also be eliminated. Two solutions arise: On the one hand, we can sort voxels along the z direction, then the drawing should began with the nearest. This minimizes the number of overdraws because the z buffer will reject the pixels. The second option is to determine the visible voxels based on the angle of the camera and the model and display only the visible parts.

Implementation of the solution requires substantial changes to the basic structure, where the properties of the voxels are stored separately. An enclosing data structure should be developed that clearly identifies the adjacent voxels. This structure and the approach has the advantage that in this case during the transformations, the calculations shouldn't be performed on each voxel

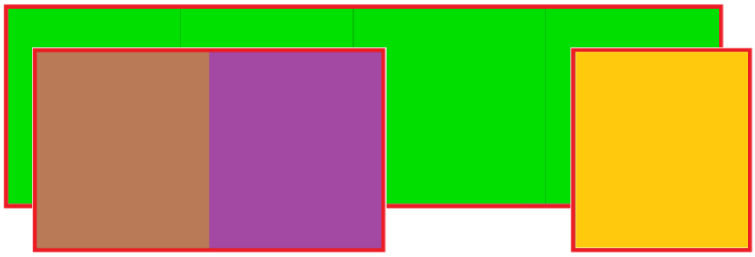

Figure 8. Sample coherent voxel subsets. During the calculations voxels can be handled in one unit, which increases the performance.

separately, but it is enough to be done on the coherent subset (Figure 7.). During the rasterization, the renderer can use these precalculated structure level parameters for drawing and calculating the position of the individual voxels.

6. Conclusion

The field of computer visualization is dominated by the polygon model representations. The graphics hardware vendors based their GPUs on this approach. The voxel technology has been present from the beginning, but it only began to emerge in recent years due to the performance of computers. The method presented in this paper attempts to fill a gap at the field of popular voxel rendering algorithms, which allows fast rendering for smaller voxel sets under certain visual compromises. The solution cannot compete with the performance of the polygon-based visualization, however due to the beneficial properties of voxelization it is well applicable in certain areas. A good example of this is, when the graphics engine combines the voxel-based solutions to the polygon-based visualization opening the way to a modern mixed rendering technology.

Acknowledgements

This research was supported by the European Union and the Hungarian State, co-financed by the European Regional Development Fund in the framework of the GINOP-2.3.4-15-2016-00004 project, aimed to promote the cooperation between the higher education and the industry.

REFERENCES

[1] P. Mileff and J. Dudra: Efficient 2D Software Rendering. Production Systems and Information Engineering, Volume 6, 2012. pp. 99-110.

[2] AKENINE-MÖLLER T. and HAINES, E.: Real-Time Rendering. A. K. Peters. 3nd Edition, 2008. *https://doi.org/10.1201/b10644*

[3] SAMULI L. and TERO K.: Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation. NVIDIA Technical Report NVR-2010-001, February 2010.

[4] RUSINKIEWICZ S. and LEVOY M.: A Multiresolution Point Rendering System for Large Meshes. SIGGRAPH '00 Proceedings of the 27th annual conference on Computer Graphics and Interactive techniques, 2000.

[5] CRASSIN C., NEYRET F., LEFEBVRE S. and EISEMANN E.: GigaVoxels: ray-guided streaming for efficient and detailed voxel rendering. I3D '09 Proceedings of the 2009 symposium on Interactive 3D Graphics and Games, pp 15-22, 2009. *https://doi.org/10.1145/1507149.1507152*

[6] MITTRING M. and HAINES, E.: The Technology Behind the Unreal Engine 4 Elemental Demo. The 39th International Conference and Exhibition on Computer Graphics and Interactive Techniques, 2012.

[7] CRASSIN C., NEYRET F., SAINZ M., EISEMANN E. and GREEN S.: Interactive Indirect Illumination Using Voxel Cone Tracing. Computer Graphics Forum, Volume 30, Issue 7, pages 1921–1930, September 2011. *https://doi.org/10.1111/j.1467-8659.2011.02063.x*

[8] THIEDEMANN S., GROSCH T. and MÜLLER S.: Voxel-based global illumination. I3D '11 Symposium on Interactive 3D Graphics and Games, pp 103-110, 2011. *https://doi.org/10.1145/1944745.1944763*

[9] VEGA-HIGUERA F., HASTREITER P., FAHLBUSCH R. and GREINER G.: High performance volume splatting for visualization of neurovascular data. Visualization, 2005. VIS 05. IEEE, pp 271 - 278, 2005. *https://doi.org/10.1109/vis.2005.47*

[10] SAKAMOTO, N., NONAKA, J., KOYAMADA, K. and TANAKA, S.: Particle-based volume rendering. Visualization, APVIS '07. 6th International Asia-Pacific Symposium on, pp 129 - 132, 2007. *https://doi.org/10.1109/apvis.2007.329287*

[11] RAMA KARL, H.: GVDB: raytracing sparse voxel database structures on the GPU. Proceeding HPG '16 Proceedings of High Performance Graphics, Pages 109-117, 2016


Production Systems and Information Engineering
Volume 8 (2019), pp. 19–37
https://doi.org/10.32968/psaie.2019.002

IMPLEMENTING THE STATE-SPACE REPRESENTATION BY THE SUPER OPERATOR DESIGN PATTERN

Gábor Kusper
Eszterházy Károly University, Hungary
Mathematics and Informatics Institute
kusper.gabor@uni-eszterhazy.hu

Csaba Sass
Eszterházy Károly University, Hungary
Mathematics and Informatics Institute
sass.csaba@uni-eszterhazy.hu

Szabolcs Márien
InnovITech Kft., Hungary
szabolcs.marien@innovitech.hu

[Received October 2017 and accepted December 2017]

Abstract. While implementing state-space representation technique of artificial intelligence we found that the mathematical design suggests having a separate state and operator classes. But this breaks the encapsulation principle of OOP. To solve this we introduce a new structural design pattern, the super operator design pattern. A super operator is a proxy to operators, it has a parameter which decides which operator to call, it returns what the operator returns. In this way one class can represent the state-space representation which contains the operators and the states, so we can preserve encapsulation. On the other hand we break separation of concerns and single responsibility principles, since we do not separate states and operators. The set of methods behind the super operator should have a similar contract, because the super operator can ensure only the common part of the contracts of the methods. If this common part is empty, then this design pattern cannot be used. The new design pattern is applicable if iteration over a set of methods is important; a set of methods have similar contract, for the client is all them same which one is called; the understandability is more important than reusability, e.g., educational purposes; forcing encapsulation over separation of concerns is essential.

Keywords: design patterns, state-space representation, super operator design pattern

1. Introduction

Artificial Intelligence (AI) as a topic is neither new and nor does Object Oriented Programing (OOP). This article is about a new way of implementation of the state-space representation technique of AI by using OOP.

There are some earlier articles where they are entangled to use the benefits of both sides, like the book [11]. Here the mentioned paradigm is mostly used for better modularity, clear hierarchical structure and reusability.

An article closer to our interest is [5] which is a PhD dissertation by Márk Kósa. The focus in his paper is how to teach graph searching algorithms using object oriented methods in Java.

As instructors of Computer Science BSc we had the possibility to teach design patterns and artificial intelligence at the same time. In our case artificial intelligence course covers well-known graph searching algorithms like backtrack and depth-first search, where the graph is generated from a state-space representation. We learned the design patterns and design principles can help us to implement the state-space representation in such a way, that our students can easily understand it and create their own state-space representation based on our new design pattern, called Super Operator.

Stata-space representation [6] is a technique to represent a problem, it is given by the sequence $\langle A, s, G, O\rangle$, where A is the set of possible states, s is the starting state, G is the set of goal states, and O is the set of operators. An operator can construct a state out of a state, i.e., it is a function from A to A. An operator has also a pre and a post condition. The pre-condition is a predicate which decides whether the input of the operator is valid or not, the post-condition decides whether the output of the operator is a valid or not.

In case of the well-known 3-missionaries and 3-cannibals problem [7] (There are 3 missionaries and 3 cannibals on the left side of a river, there is also a 2-sit boat on the left side. The problem is that cannibals are hungry, and are going to eat missionaries if there are more cannibals than missionaries on any sides. We have to move all of them to the right side in safe!) one possible state-space representation is the following: $A = \{a|IsState(a)\}$. $IsState(\langle ml, cl, boat, mr, cr\rangle) :\Leftrightarrow ml \in \{0,1,2,3\} \wedge cl \in \{0,1,2,3\} \wedge boat \in \{left, right\} \wedge mr \in \{0,1,2,3\} \wedge cr \in \{0,1,2,3\} \wedge ml + mr = 3 \wedge cl + cr = 3$, where ml means missionaries on left, cl means cannibals on left, mr means missionaries on right, and cr means cannibals on right. $s = \langle 3, 3, left, 0, 0\rangle$. $G = \{\langle 0, 0, right, 3, 3\rangle\}$. $O = \{move01, move02, move10, move20, move11\}$.

From these 5 operators we give only one formally, the other ones can be constructed out of this one using the intuition that $moveXY$ means that we move X missionaries and Y cannibals by the boat to the other side.

$move01(\langle ml, cl, boat, mr, cr\rangle) = \langle ml, cl-1, right, mr, cr+1\rangle, if\ boat = left$; $\langle ml, cl+1, left, mr, cr-1\rangle, otherwise$.

The operators of the state-space representation are called by a graph searching algorithm, because the graph is not ready when we start the algorithm, instead, some part of the graph is generated by the graph searching algorithm on thy fly. For example: backtrack generates a path, depth-first search generates a spanning tree of the graph. This means that we have to define an interface which is used by the graph-searching algorithms to call the operators. The two main possibilities are the following:

- We follow the mathematical description and we define separate classes like:
  - State, which represents a state,
  - Operator, which represents an operator,
  - StateSpaceRepr, which represents a state-space representation.
- We use the Super Operator Design Pattern and we define only one class:
  - State, which contains the operators and the states and represents state-space representation.

The first solution seems to be better since we know that separation of concerns [9, 10, 3] and single responsibility principle (for short: SRP) [8] are very strong OOP design principles [8, 2]. Separation of concerns states that if two or more concerns can be separated than it is a good idea to separate them, since the resulting source code will have better reusability. SRP state that each class should have only one responsibility, or in other words, each class should have only one kind of reason to change.

The problem is that in the case of the first solution we separate two concerns, state and operation, which belong very tightly together and hence we have to develop these two classes in parallel: the Operator has to know State and the State must be designed in such way that Operation should be easy to implement. Furthermore, if the State is changed then most probably we have to change the Operator and the other way around. This means that we have an implementation dependency, which is not easy to overcome.

These problems come from the fact that this solution breaks the encapsulation [1] OOP principles. It states that a class consists of a data-structure and methods (by other words: operators) on data structure. The data-structure is implemented by fields of the class. The values of the fields give the inner-state of instances of the class. The inner-state must not be modified from outside the class, only the methods of the class may modify the inner-state.

While we have only a few OOP principles [1], like abstraction, inheritance, encapsulation, and polymorphism, we have several OOP design principles [8, 2],

like separation of concerns, single responsibility principle (SRP), open-closed principle (OCP), Hollywood principle (HP), Liskov substitutional principle (LSP), interface segregation principle (ISP), law of Demeter etc.

So, by the first solution we break a very basic principle, encapsulation, while we preserve some high level principles, separation of concerns, and SRP.

The second solution is introduced in this article, and has the name Super Operator Design Principle. In this solution we have only one class, the State class. The operators are methods of this class. The problem is that the number of operators and their functionality may vary wildly, although their contract is the same: they create a new state from an old one if their pre and post-condition is true. An idea would be to define a list of operators, but in case of this solution there is no Operator class. The other idea is to define a "super" operator which can call "basic operators". By "basic operators" we mean operators of the state-space representation.

In this case we need two methods:

- NumberOfOperators(), which returns the number of basic operators.
- SuperOperator(int i), which calls the i-th. basic operator and just returns what it returns.

In this way there is a well-defined interface for basic operators and we can implement the concerns of state and operation in the same class. So we preserve the encapsulation principle, but break the separation of concerns and the SRP. In the following sections we discuss more deeply these two solutions and their main variants.

2. Implementation of State-Space Representation Following the Definition

In this chapter we give the implementation of state-space representation which follows its definition. We use 4 classes:

- State, which represent a state,
- Operator, which represent an operator,
- StateSpaceRepr, which represent a state-space representation, and
- Node, which represent a vertex of the search graph generated out of the state-space representation.

We give here the source code of the first 3 classes (the whole can be found here: `http://pastebin.com/QvYzDG4R`).

```
abstract class State {
    public abstract bool IsState();
```

```
    public abstract bool IsGoalState();
}
abstract class Operator {
    public State Operate(State a){
        if (!PreCondition(a)) return null;
        State b = operate(a);
        if (!PostCondition(b)) return null;
        return b;
    }
    public abstract bool PreCondition(State a);
    protected abstract State operate(State a);
    public abstract bool PostCondition(State a);
    public bool IsApplicableOn(State a) {
        if (!PreCondition(a)) return false;
        State b = operate(a);
        if (!PostCondition(b)) return false;
        return true;
    }
}
class StateSpaceRepr {
    State startState;
    List<Operator> operators;
    public StateSpaceRepr(State s, List<Operator> op) {
        this.startState = s; this.operators = op;
    }
    public bool IsState(State state) { return
       state.IsState(); }
    public State GetStartState() { return startState; }
    public bool IsGoalState(State state) { return
       state.IsGoalState(); }
    public List<Operator> GetOperators() { return operators; }
}
```

We can see that class State and Operator are abstract ones. If we have a problem, like the 3-missionaries and 3-cannibals problem, then we can create child classes by implementing the abstract methods. The State class contains two abstract methods:

- bool IsState(), which decides whether the instance is a state or not, and

- bool IsGoalState(), which decides whether the instance is a goal state or not.

One can argue that these two predicate could be implemented also in the StateSpaceRepr. In this way the State class will be empty and the IsState(State a) and IsGoalState(State a) will be abstract in the StateSpaceRepr. This solution is possible but we think that the presented solution fits better OOP principles, since IsState() and IsGoalState() are methods on a state, so they should be placed in the State class.

The State class is meant to be immutable. This means that if we run an operator on it, then it will be not changed, so we do not have to make a copy of before an operator call, so we do not implement the Cloneable interface. Our implementation of 3-missionaries and 3-cannibals problem (`http://pastebin.com/QvYzDG4R`) follows this convention.

The Operator class contains 3 abstract methods:

- bool PreCondiction(State a), which is the pre-condition of the operator,
- bool PostCondition(State a), which is the post-condition of the operator, and
- State operate(State a), which is the operator itself.

It is interesting that we have an Operate and an operate method at the same time. The Operate method is public but not polymorphic. It follows the template method design pattern and fix the order of PreCondition, operate, and PostCondition. It is also an example for inversion of control, since it calls and abstract method, operate, which will be implemented by the child class, so not the child calls the parent, but the parent calls the child. The operate method is abstract, i.e., polymorphic, and should be overridden by the child to implement the operator, as you can see here: `http://pastebin.com/QvYzDG4R`.

The StateSpaceRepr class contains the following methods:

- IsState(State state), it calls "state.IsState();",
- GetStartState(), gives back the initial (by other word: starting) state,
- IsGoalState(State state), it calls "state.IsGoalState();",
- GetOperators(), returns the list of operators.

This class just follows the mathematical description of state-space representation, which is $\langle A, s, G, O \rangle$, A is the set of states, s is the starting state, G is the set of goal states, and O is the set of operators. The only difference, that instead of set of states we have a predicate, the IsState() method, which can decide whether a State instance is really a state or not. The same applies to set of goal states.

The only problem with this implementation is that it reveals a part of its inner state, since it gives back the reference to operators by GetOperators() method. The rest of the code can change the list of operators through this reference, which can result in a hard-to-detect bug. So this implementation assumes that the rest of the code is trustworthy.

Now we give the implementation of the child classes of State and Operator in case of 3-missionaries and 3-cannibals problem:

```
class MCState : State { // Missionaries And Cannibals State
    int ml, cl, boat, mr, cr;
    public int MissionariesOnLeft { get { return ml; } }
    public int CannibalsOnLeft { get { return cl; } }
    public int Boat { get { return boat; } }
    public int MissionariesOnRight { get { return mr; } }
    public int CannibalsOnRight { get { return cr; } }
    public MCState(int ml, int cl, int boat, int mr, int cr) {
        this.ml = ml; this.cl = cl; this.boat = boat;
        this.mr = mr; this.cr = cr;
    }
    // this creates the start state
    public MCState() {
        ml = 3; cl = 3; boat = 1; // 1: left, -1: right
        mr = 0; cr = 0;
    }
    public override bool IsState() {
        return ml + mr == 3 && cl + cr == 3 &&
               ml >= 0 && ml <= 3 && cl >= 0 && cl <= 3 &&
               mr >= 0 && ml <= 3 && cr >= 0 && cr <= 3 &&
               (boat == 1 || boat == -1) &&
               (ml >= cl || ml == 0) && (mr >= cr || mr == 0);
    }
    public override bool IsGoalState() {
        return mr == 3 && cr == 3 && boat == -1;
    }
    public override bool Equals(object obj) {
        MCState other = (MCState) obj;
        return this.ml == other.ml && this.cl == other.cl &&
               this.boat == other.boat &&
               this.mr == other.mr && this.cr == other.cr;
```

```
    }
}
class MoveMC : Operator {
    int mToMove, cToMove;
    public MoveMC(int mToMove, int cToMove) {
        this.mToMove = mToMove;
        this.cToMove = cToMove;
    }
    public override bool PreCondition(State a) {
        MCState state = (MCState)a;
        if (state.Boat == 1) {
            return state.MissionariesOnLeft >= mToMove &&
                   state.CannibalsOnLeft >= cToMove;
        }
        else {
            return state.MissionariesOnRight >= mToMove &&
                   state.CannibalsOnRight >= cToMove;
        }
    }
    protected override State operate(State a) {
        MCState state = (MCState)a;
        int ml, cl, boat, mr, cr;
        ml = state.MissionariesOnLeft;
        cl = state.CannibalsOnLeft;
        boat = state.Boat;
        mr = state.MissionariesOnRight;
        cr = state.CannibalsOnRight;
        if (boat == 1) {
            ml -= mToMove; cl -= cToMove; boat = -1;
            mr += mToMove; cr += cToMove;
        }
        else {
            ml += mToMove; cl += cToMove; boat = +1;
            mr -= mToMove; cr -= cToMove;
        }
        return new MCState(ml, cl, boat, mr, cr);
    }
    public override bool PostCondition(State a) {
```

```
        return a.IsState();
    }
    public override string ToString() {
        return mToMove + ", " + cToMove;
    }
}
```

The problem with this implementation is that the MCState class has to reveal its inner-state by properties (we have a getter property for each attribute in MCState class), otherwise we could not write the MoveMC operator.

One can argue that in OOP it is quite natural that we have a getter for each attribute. This is true. The real problem is that MCState and MoveMC classes are tightly coupled, i.e., there is a strong implementation dependency between them. If we change the representation of a state, for example we use boolean attribute instead of integer to store where is the boat, then we have to change the operator, and if we change the operator, for example to have less basic operator, then most probably we have to change the state.

This strong implementation dependency comes from the fact that the MCState represents the data-structure, and MoveMC represents the methods on that data-structure.

This means that we break the encapsulation OOP principle, which states that in OOP the data-structure and its method is one entity which is called class. On the other hand we have high-level OOP design principles like separation of concerns and SRP. Separation of concerns states that if two or more concerns can be separated than it is a good idea to separate them, since the resulting source code will have better reusability. SRP state that each class should have only one responsibility, or in other words, a class should have only one reason to change.

We can see that state and operator can be separated, and hence, by separation of concerns, it is good to separate them. SRP is questionable here. MCState has only one responsibility, it represent a 3-missionaries and 3-cannibals state. MoveMC has also only one responsibility, it represent an operator of 3-missionaries and 3-cannibals problem. The question is that they change on the same reason or not? If yes, then we should not separate them, if not, then we should separate them because of SRP.

We present a reason which applies only to the operator but not for the state, but we admit, that almost any other reason forces the changes both the state and the operator. The reason which applies only to the operator is that we would like to speed it up, for example by getting rid of the "if (boat == 1)" statement. The new operator looks like this:

```
protected override State operate(State a) {
    MCState state = (MCState)a;
    int ml, cl, boat, mr, cr;
    ml = state.MissionariesOnLeft;
    cl = state.CannibalsOnLeft;
    boat = state.Boat;
    mr = state.MissionariesOnRight;
    cr = state.CannibalsOnRight;
    ml -= mToMove * boat;
    cl -= cToMove * boat;
    mr += mToMove * boat;
    cr += cToMove * boat;
    boat = boat * -1;
    return new MCState(ml, cl, boat, mr, cr);
}
```

This means that the condition of SRP holds, so we should also do separation because of it. But there is still one more question. Do we really get better reusability? Can we reuse MoveMC without MCState? In fact not, we cannot reuse it, because of this line:

```
MCState state = (MCState)a;
```

The first step of the operator is to convert the input State into MCState, so to use MoveMC we have to use MCState.

On the other way we could reuse MCState without MoveMC, but it is not easy to find any scenario. A possible scenario is if we create another problem, which uses the same representation. For example, there is a 3-sit boat instead of the 2-sit one.

It seems that it does not pay off to break encapsulation in order to preserve separation of concepts and SRP.

3. Implementation of State-Space Representation using the Super Operator Design Pattern

If we use the super operator design pattern, then we can encapsulate the state and the operator in one class, although we do not know in advance how many basic operators there will be. The solution is the SuperOperator(int i) method, which can call the i.-th basic operator and returns what the basic operator returns.

In this way we need only one class, namely State, instead of State, Operator, and StateSpaceRepr from the previous section. Of course we still need the Node class which will represent the search graph generated out of the state-space representation, but now any information comes from the State class.

Here we give the source code of State (the whole implementation of state-space representation using super operator design pattern is here: `http://pastebin.com/6DNpQjVr`):

```
abstract class State : ICloneable {
    public abstract bool IsState();
    public abstract bool IsGoalState();
    public abstract int GetNumberOfOperators();
    public abstract bool SuperOperator(int i);
    public abstract object Clone();
}
```

One can see that this State class has 3 more methods compared to the State class from the previous section. The new methods are:

- GetNumberOfOperators(), which returns the number of basic operators,
- SuperOperator(int i), which calls the i.-th basic operator and returns what it returns,
- Clone(), which clones the state and which uses deep copy.

The first two methods are used to move the "interface for operators" from StateSpaceRepr to the State class. This means that these two methods are used by rest of the code to call the operators. The method call

- "state.SuperOperator(0);" calls the first basic operator,
- "state.SuperOperator(1);" calls the second basic operator,
- ...
- "state.SuperOperator(state.GetNumberOfOperators()−1);" calls the last one.

The idea is that at the time of the development of the graph searching algorithms, like backtrack, we do not know the concrete problem, we do not know the operators to call, we only know that we have to call (in the worst case) all of them. So we need an interface that is known at the time of developing backtrack and it can be used to iterate over the operators. This interface is the above two methods from class State:

- GetNumberOfOperators(),
- SuperOperator(int i).

The third method, Clone(), is used because we prefer mutable state representation, where the operators can work on the attributes of the state. We know from backtrack that sometimes we have to go back to an earlier state, so before we apply an operator to a state, we have to clone t. This is done in the Node class, which is not presented in this article, but can be found here: `http://pastebin.com/6DNpQjVr`.

If one prefers to have immutable state representation then we need a kind of copy constructor instead of the Clone() method. We have implemented this variant, which can be found here: `http://pastebin.com/gZ9rF4Qz`.

The immutable variant moves the task "copy the attributes" into the operator, which seems to be less natural to our students, so we prefer the mutable variant presented in this article, although it is more dangerous, because sometimes our student uses shallow copy instead of deep copy in the Clone() method. Usually they learn in the course of debugging what is the difference Of course we should also use deep copy if we choice the immutable variant but in this case deep copy take place inside the operator.

Here we give the source code of MCState (the whole implementation is here: `http://pastebin.com/6DNpQjVr`):

```
class MCState : State {
    int ml, cl, boat, mr, cr;
    public MCState() {
        ml = 3; cl = 3; boat = 1; // 1: left, -1: right
        mr = 0; cr = 0;
    }
    public override object Clone() { return
       MemberwiseClone(); }
    public override bool IsState() {
        return ((ml >= cl) || (ml == 0)) &&
               ((mr >= cr) || (mr == 0));
    }
    public override bool IsGoalState() {
        return mr == 3 && cr == 3 && boat == -1;
    }
    public override int GetNumberOfOperators() { return 5; }
    public override bool SuperOperator(int i) {
        switch (i) {
            case 0: return operate(0, 1);
            case 1: return operate(0, 2);
```

```
            case 2: return operate(1, 1);
            case 3: return operate(1, 0);
            case 4: return operate(2, 0);
            default: return false;
        }
    }
    private bool operate(int mToMove, int cToMove) {
        if (!preCondiction(mToMove, cToMove)) return false;
        if (boat == 1) {
            ml -= mToMove; cl -= cToMove; boat = -1;
            mr += mToMove; cr += cToMove;
        }
        else {
            ml += mToMove; cl += cToMove; boat = 1;
            mr -= mToMove; cr -= cToMove;
        }
        if (IsState()) return true;
        return false;
    }
    private bool preCondiction(int mToMove, int cToMove) {
        if (mToMove + cToMove > 2 || mToMove + cToMove < 0 ||
            mToMove < 0 || cToMove < 0) return false;
        if (boat == 1)
            return ml >= mToMove && cl >= cToMove;
        else
            return mr >= mToMove && cr >= cToMove;
    }
    public override string ToString() {
        return ml + "," + cl + "," + boat + "," + mr + "," +
           cr;
    }
    public override bool Equals(Object a) {
        MCState aa = (MCState)a;
        return aa.ml == ml && aa.cl == cl && aa.boat == boat;
    }
}
```

Please note, that the SuperOperator(int i) contains a "big-fat" switch statement which calls the basic operators depending on the value of parameter i.

Usually it is enough to write only a few operator with some parameter. In the above example there is only one operator with two parameters: operate(int mToMove, int cToMove). Although we have only one operator, there are 5 basic operators, because different parameters give different basic operators of the state-space representation. The 5 basic operators are:

- operate(0, 1), which moves no missionaries and 1 cannibal to the other side of the river,
- operate(0, 2), which moves no missionaries and 2 cannibals to the other side of the river,
- operate(1, 1), which moves 1 missionary and 1 cannibal to the other side of the river,
- operate(1, 0), which moves 1 missionary and no cannibals to the other side of the river,
- operate(2, 0), which moves 2 missionaries and no cannibals to the other side of the river.

The switch in the SuperOperator(int i) method calls them depending on the value of i. The order of the basic operator is all the same, but it might be that some order results in better running time.

Please note also, that GetNumberOfOperators() return 5, so there must be 5 cases in the switch of SuperOperator(int i) from 0 to 4, i.e., we can see the SuperOperator(int i) method as a mapping between the range 0 ... GetNumberOfOperators()-1 and the basic operators. We recall the methods GetNumberOfOperators() and SuperOperator(int i) in order to make it easier to the reader to check the above statements:

```
public override int GetNumberOfOperators() { return 5; }
public override bool SuperOperator(int i) {
    switch (i) {
        case 0: return operate(0, 1);
        case 1: return operate(0, 2);
        case 2: return operate(1, 1);
        case 3: return operate(1, 0);
        case 4: return operate(2, 0);
        default: return false;
    }
}
```

These two methods together are the heart of the Super Operator Design Pattern, which is defined more formally in the next section. It is important that the basic operators have the same goal: they create a new state out of the old

one. In other words they contract is very similar, although not the same. For example operate(0, 1) moves 1 cannibal, and operate(0, 2) moves 2 ones, but both of them result in a new state if their pre- and post-condition holds, in this case they return true. This is their common contract.

The super operator is suitable only if the client requires the common contract. If the client would like to move 2 cannibals, then the super operator design pattern is not good, because the client do not know which super operator parameter value maps to the operator which moves 2 cannibals.

The graph searching algorithms are not interested how the basic operators work, for them it is enough that they create a new state if they are applicable. So the super operator design pattern is suitable for them.

The graph searching algorithms must call all the basic operators in the worst case, but this is also possible by the super operator design pattern. It is important to note that the operate(int mToMove, int cToMove) method is private, so it is hidden from the rest of the code. It is hidden by the super operator. Since super operator hides other methods, we can also call it "proxy for methods", or for short "method proxy".

The super operator design pattern preserves the encapsulation OOP principle, since it keeps together the data-structure (the attributes of State) and its methods (the basic operators) in one class, which is the State class; but it breaks separation of concerns and SRP, since the State class fulfills the responsibilities of the states, operators, and the state-space representation.

Note that, that the implementation of Clone() is very easy in this example. We have to call only a C# specific method, called MemberwiseClone(), which creates a shallow copy of the instance. We know that Clone() has to create a deep copy, but since each attribute in MCState is a value-type one, there is no difference between shallow and deep copy. After this case study we give the Super Operator Design Pattern more formally in the next section.

4. Super Operator Design Pattern

In this section we define the Super Operator Design Pattern by the structure used in the GOF book [GOF1995], which is widely accepted by programmers.

First of all we have to discuss whether super operator is a creational, a structural or a behavioral design pattern.

Creational design patterns are used instead of the new keyword. The above case study, especial the immutable variant, creates new State instances, but this is not the intent of the pattern but this is the common contract of the methods behind the super operator. So this is not a creational pattern.

Behavioral patterns focus how objects communicate. This pattern gives conditions only for one class, so this is not a behavioral pattern.

Structural patterns focus on relationships of entities. This pattern describes how a set of methods and a special method, called super operator, are related: the super operator is a mapping between its parameter and between these set of methods, its parameter decides which method to call, it returns what this method returns. So this is a structural design pattern.

Intent. Give a common interface, the super operator, for a set of basic methods which have similar contract, the super operator hides them and ensures the least common contract of them, the parameter of super operator decides which one is called, so the client can iterate over the methods.

Also Known As. Proxy for methods, method proxy.

Motivation. While implementing state-space representation technique of artificial intelligence we found that the mathematical design suggests having a separate state and operator classes. But this breaks the encapsulation principle of OOP. To solve this we might introduce a so called super operator which serves as a proxy to operators. It has a parameter which decides which operator to call, it returns what the operator returns.

In this way state class can contain the operators and the super operator and so we can preserve encapsulation. This means that to create a new state-space representation we need only implement a child class of state. In this way the new state-space representation will be concise and easy to understand. On the other hand we break separation of concerns and SRP, since we do not separate states and operators. Super operator design pattern favors understandability over reusability.

The super operator hides the methods (the operators), and with the help of "get number of operators" makes it possible for the client to iterate over them. In case of state-space representation, the backtrack algorithm can iterate over operators. The set of methods behind the super operator should have a similar contract, because the super operator can ensure only the common part of the contracts of the methods. If this common part is empty, for example, if one of the methods gives back integer and another one gives back string, then this design pattern cannot be used.

Applicability. Super operator design pattern is used when:

- Iteration over a set of methods is important.
- A set of methods have similar contract, for the client is all them same which one is called.
- The understandability is more important than reusability, e.g., educational purposes.

- Forcing encapsulation over separation of concerns is essential.

Structure. There are at least two classes:

- The server class which contains the set of methods hidden by super operator.
- The client class which calls the super operator.

We assume that the client want to call (in the worst case) all of the hidden methods, it wants to call the best one first, or it is irrelevant in which order the methods are called. In this case the server has to have at least two methods:

- SuperOperator(int i, Object p1, Object p2, ..., Object pn),
- GetNumberOfMethods().

The SuperOperator calls the i.-th method with parameters p1, p2, ..., pn. It might call the i.-th method with fewer or ever more parameters. It returns what the i.-th methods gives back. The client calls the SuperOperator from a loop like this one:

```
for(int i = 0; i < server.GetNumberOfMethods(); i++)
{
    Object result = server.SuperOperator(i, p1, p2, ..., pn);
    ...
}
```

The Server class must be either immutable, or must be cloned before call of SuperOperator, or the last call of SuperOperator must be withdrew before the next call of SuperOperator. For the first two cases we have a sample implementation:

- `http://pastebin.com/gZ9rF4Qz`, where the server class is immutable,
- `http://pastebin.com/6DNpQjVr`, where the server class is cloneable.

The server has to maintain a heuristic if the client wants to call the best method at first, the second best afterwards. In this case the heuristic must decide which method is the best.

Consequences. If we use the super operator design pattern then we do not have to implement an Operator class, which results in less modular but more understandable solution.

Related Patterns. There are two structural patterns similar to super operator:

- Proxy, which hides a critical object in a transparent way, and
- Facade, which hides a complex module by a simple interface.

Proxy is similar to super operator, both of them hide some parts of the source code from the rest of the code, but proxy hides a critical object, most probably only one critical object, super operator hides a set of similar methods, most probably not only one method.

Facade is similar to super operator, both of them define a simple interface to hide some part of the source code from the rest of the code, but while façade hides several objects which work together as a complex module, super operator hides only a set of similar methods.

5. Future work

We would like to study open source projects, like the ones hosted by Apache Software Foundation, whether the super operator design pattern is used by some of them or not. If yes, we should understand those variants and integrate them into the description of this pattern. If not, then we must consult software architectures and analyze their opinions about this new design pattern.

REFERENCES

[1] Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston: *Object-Oriented Analysis and Design with Applications.* 3rd Edition, book, 2007.

[2] John Dooley: *Object-Oriented Design Principles, in Software Development and Professional Practice, book chapter.* Apress, pages 115-136, 2011.

[3] Erik Ernst: *Separation of concerns.* Proceedings of the AOSD 2003 Workshop on Software-Engineering Properties of Languages for Aspect Technologies (SPLAT), Boston, USA, 2003.

[4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, ISBN 0-201-63361-2, 1995.

[5] Márk Kósa: *Korszerű információtechnológiai módszerek bevezetése a mesterséges intelligencia oktatásába.* PhD dissertation, University of Debrecen, 2009.

[6] Gergely Kovásznai, Gábor Kusper: *Artificial Intelligence and its Teaching.* Lecture notes, Eszterházy Károly College, 2012.

[7] Vladimir Lifschitz: *Missionaries and cannibals in the causal calculator.* Proceedings the KR2000 conference, pages 85-96., 2000.

[8] Robert C. Martin: *The single responsibility principle*, in The Principles, Patterns, and Practices of Agile Software Development, pages 149-154, 2002.

[9] D.L. Parnas: *On the Criteria To Be Used in Decomposing Systems Into Modules.* Communications of the ACM, 15, 12, 1053-1058, 1972.

[10] D.L. Parnas: *Software Engineering or Methods for the Multi-Person Construction of Multi-Version Programs*. Lecture Notes in Computer Science, Programming Methodology, 1974.

[11] Bernard P. Zeigler: *Object-Oriented Simulation with Hierarchical*, Modular Models: Intelligent Agents and Endomorphic System, ISBN: 978-0-12-778452-6, 2014.

Production Systems and Information Engineering
Volume 8 (2019), pp. 39–49
https://doi.org/10.32968/psaie.2019.003

COST ANALYSIS OF THE PREFIX TREE DATA STRUCTURE

Edit Csizmás
Pallasz Athéné University, Kecskemét, Hungary
Department of Informatics
csizmas.edit@gamf.kefo.hu

László Kovács
University of Miskolc, Hungary
Institute of Information Science
kovacs@iit.uni-miskolc.hu

[Accepted July 2017]

Abstract. The data tree is a very widely used data structure in many application areas. The tree structure provides an efficient storage and data manipulation for different data lists or data sets. The prefix tree is a special tree structure to store ordered lists of data elements. In the prefix tree, the lists with common prefix part share the same path. As the prefix tree can be involved in many data manipulation algorithms, the cost estimation of the tree is an important component in the cost function of the whole data manipulation algorithm. This paper provides a cost analysis related to the tree size for a random input set of object lists. The analysis includes both analytical and simulation methods and the main result presented in this paper is an approximation function based on the gamma-distribution.

Keywords: prefix tree, cost function, gamma-distribution

1. Introduction

The ordered list is a good implementation structure for the set representation. The ordering of the elements enables a more efficient implementation of the set operations. For example, the set intersection can be implemented with a sorted merge algorithm. This kind of implementation is used among others in the query engine of the relational databases during the execution of an inner join operation. Thus, we have an object set with a total ordering on it:

$$\langle Oo_i, \leq \rangle$$

The size of O is denoted with M. A set of objects can be represented with an ordered list:

$$l = o_1, ..., o_n \text{ where } o_{i-1} \leq o_i$$

A set of lists, $L = \langle l_j \rangle$ can be represented with a special tree, prefix tree [1], T, where every list l is represented with a path starting from the root element. The prefix tree constructed from the lists on O is denoted by T_O. The paths with similar prefix part share the same segment in the tree. In the tree every node is assigned to an element $o \in O$, except the root, which is empty node. In the tree, every node represents an object sequence related to the nodes along the path from the root. As some lists may be included in other lists as sub-lists, we need a special flag in the nodes to denote the end symbol of the lists. A given list $o_1, ..., o_n$ is contained in T, if there exists a path where the i-th element of the path is assigned to o_i and at the node related to on has end-node flag set to 1. In Fig. 1, a sample prefix tree is shown which contains the following lists: (1,2),(2,3),(3).

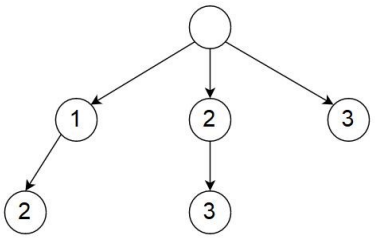

Figure 1. Sample prefix tree

The size of the tree T is equal to the number of nodes in the tree. The prefix tree T_O is a complete tree, if it contains all possible lists on O. The size of the complete prefix tree for the O with M elements is 2^M. This formula can be deducted from the fact that if a new element is added to O, then a node child node assigned to the new element must be appended to every node of the tree. Thus in every step, the size of the tree is doubled (shown in Fig 2.).

In general, the cost of data manipulation operations depend on the size of the data structure. Thus, the size of the prefix tree is an important factor in cost estimation of the data manipulation algorithms. As the size of the data tree depends on the set of input list, the total cost is also a function of the input data. Considering the data set, there are two crucial factors in data generation:

- number of lists

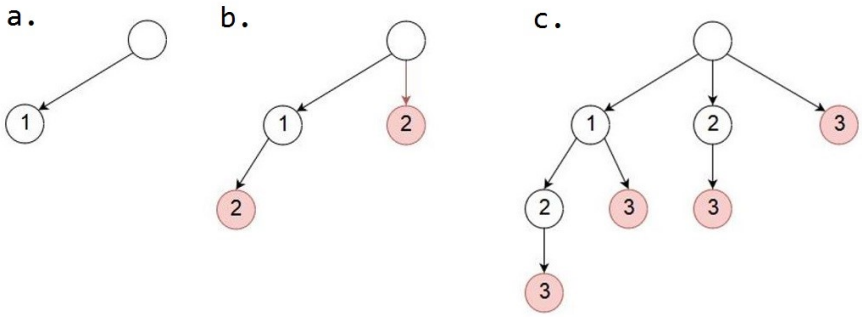

Figure 2. Construction of complete prefix trees

- distribution of the elements in the lists.

The enumeration of tree structures is an intensively investigated topic in the literature. Regarding the general complexity analysis of tree, we can find many contributions [4], [5], [6]. On the other hand, there are no enumeration studies on the prefix tree structure as a specific tree in the literature. The goal of our contribution is to provide an initial analysis on the size complexity of the prefix tree structures.

In our investigation, as an initial step, an independency assumption is applied, that means that the probability of the existence of element o_i in the list, is independent from the probability of any other element o_j. The main parameters of the input data generation are the followings:

- M: the number of elements in O,
- K: the number of lists in the input data set,
- p_i: the probability of the element o_i in the input lists,
- $p_O = (p_1, ..., p_M)$: the probability vector for O.

2. Cost Analysis of the prefix tree size

Two basic approaches are analyzed in our work. The first is based on application of exact probabilistic formulas to determine the size of the generated prefix trees. This method provides precise values but the formulas become too complex when $M > 2$. The second method uses simulation to generate random lists to build up prefix trees. The size of the generated prefix trees are investigated using statistical methods. Based on the experiences, this approach is more suitable to approximate the size of the tree for larger data sets too.

2.1. Enumeration formula

For the case, $M = 2$, the analytical, exact formulas can be constructed on a straightforward way. The formula for the tree size is built up from the following components.

- $P_0 = (1 - p_a)(1 - p_b)$, probability that none of the two elements occurs in the list,
- $P_a = p_a(1 - p_b)$, probability that only element a occurs in the list,
- $P_b = (1 - p_a)p_b$, probability that only element b occurs in the list,
- $P_{ab} = p_a \cdot p_b$, probability that both elements a and b occur in the list.

It can be verified that

$$P_0 + P_a + P_b + P_{ab} = (1 - p_a)(1 - p_b) + p_a(1 - p_b) + (1 - p_a)p_b + p_a p_b = 1$$

Considering the size of the resulted prefix tree, the probabilities of the different size values can be calculated. The probability of the size i is denoted by P_i and the value can be calculated on the following way:

$$P_1 = P_0^K$$

$$P_2 = \sum_{i=1}^{K} \binom{K}{i} P_a^i P_0^{K-i} + \sum_{i=1}^{K} \binom{K}{i} P_b^i P_0^{K-i}$$

$$P_3 = \sum_{i=1,j=1}^{K} \binom{K}{i}\binom{K-i}{j} P_a^i P_b^j P_0^{K-i-j} + \sum_{i=1,j=0}^{K} \binom{K}{i}\binom{K-i}{j} P_a^j P_{ab}^i P_0^{K-i-j}$$

$$P_4 = \sum_{i=1,j=1,l=0}^{K} \binom{K}{i}\binom{K-i}{j}\binom{K-i-j}{l} P_a^l P_{ab}^i P_b^j P_0^{K-i-j-l}$$

The average value of the tree size is calculated with:

$$E[Ntree] = P_1 + 2 \cdot P_2 + 3 \cdot P_3 + 4 \cdot P_4$$

The exact formulas were calculated in a Matlab application for different element probability values. For the sake of simplicity, both elements have the same probability: $p_a = p_b$. The resulted size-function is shown in Figure 3.

As for an arbitrary M, the number of tags in the formula increases to 2^M, it is not possible to use this approach for analysis of larger problems. Thus the application of the exact enumeration formulas is very limited.

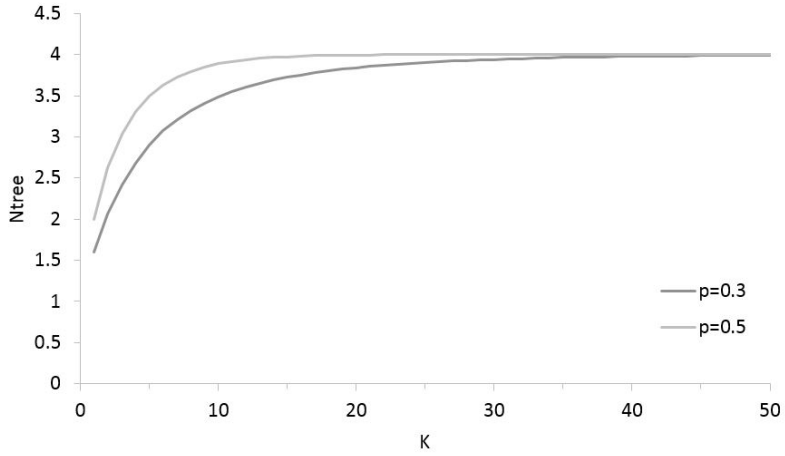

Figure 3. The average size for different element probabilities (M=2)

2.2. Cost Analysis with Simulation

In the simulation, prefix trees are built up from random lists. The generation function to construct the random lists is based on the Monte Carlo method and it depends on three parameters:

- M: the number of elements in O,
- K: the number of lists in the input data set,
- p: the common probability of the elements to appear in the list.

The goal of the simulation is to determine the dependency between the tree size and the input parameters. In our implementation, the lists are stored in a binary matrix form, where the columns correspond to the elements and the rows denote the generated lists. The matrix element $m(i, j)$ is equal to 1 if the i-th list contains the j-th element, otherwise the value is equal to 0.

The size of the trees are calculated with the following algorithms:

- ordering of the input lists by the length of the list,
- eliminating the lists contained in other lists,
- tree construction from the reduced set of lists.

In the simulation, N denotes the number of performed tests. The average of the test results is used to construct the size function. The empirical cost function yielded by the tests for $M = 2$, is shown in Fig. 4.

As it is expected, the simulated cost function for larger N values is more smoother than the functions for small N values. In the investigated parameter

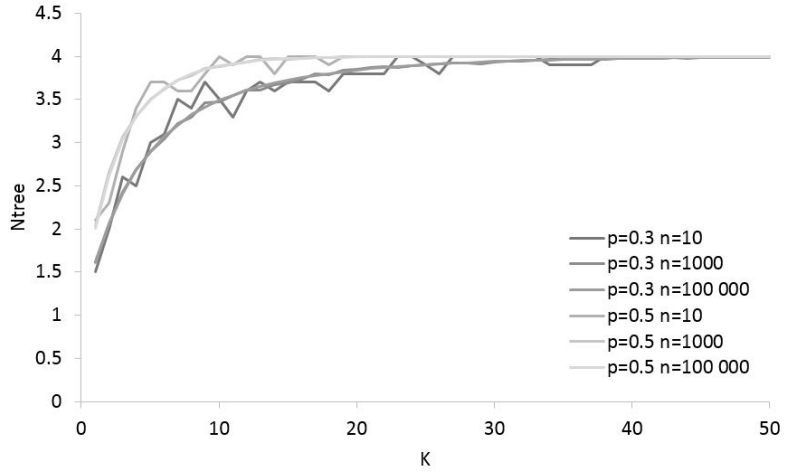

Figure 4. The experimental cost function in dependency from K. M = 2; p = 0,3 and 0,5; N = 10; 1 000; 100 000

ranges, the simulated cost functions are very similar to the exact calculated cost functions. The corresponding error values, calculated as the difference between the simulated and calculated cost values, are given in Table 1.

Table 1. The error of the simulated cost function

	N = 10	N = 1000	N = 100000
p = 0.3	0.6981146	0.098185	0.010077
p = 0.5	0.549177	0.064796	0.007265

For larger M and K values, as the run time of the simulation became significantly high, only lower number of tests were executed. The parameter range investigated in the test series are given in Table 2.

Table 2. The investigated parameter ranges in the simulations

	min	max	step
M	5	50	5
K	200	12000	200
p	0.3	0.5	0.2

In Fig. 5 and Fig. 6, the tree size is shown in dependency of K for different M values. It can be seen that for a higher K value, the prefix tree will be complete and the cost remain constant. The shape of the cost function in Fig

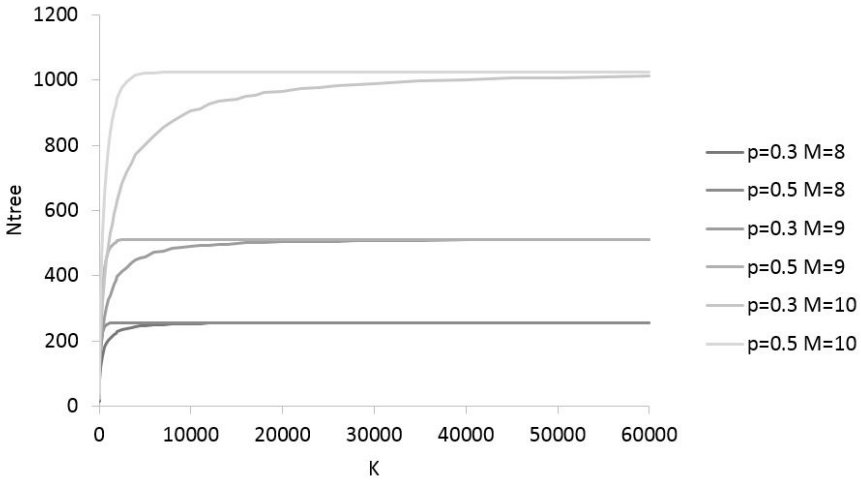

Figure 5. Simulated cost function for large K values

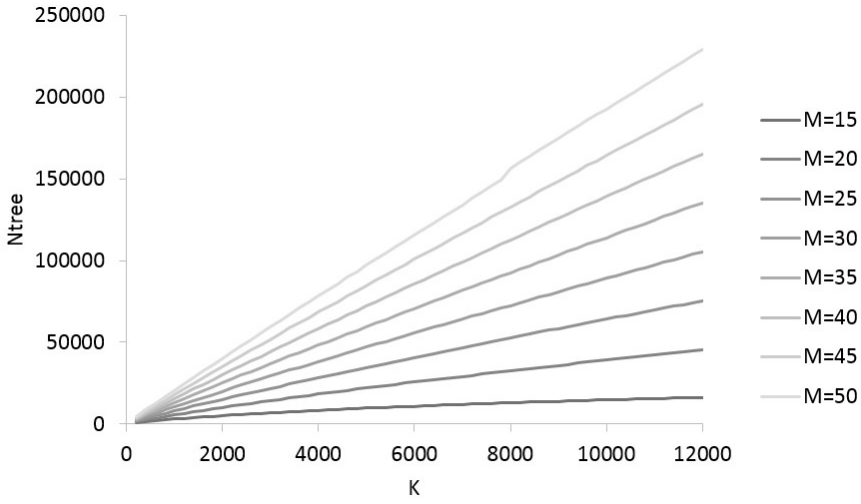

Figure 6. Simulated cost function for small K values

6 shows that the investigated parameter range is still far from the saturation threshold.

We have tested the value of the saturation threshold in the dependency of the M value. The result is shown in Fig. 7. Our experience is that the K

saturation value is an exponential function of M and the increase is larger for $p = 0.3$ than for $p = 0.5$.

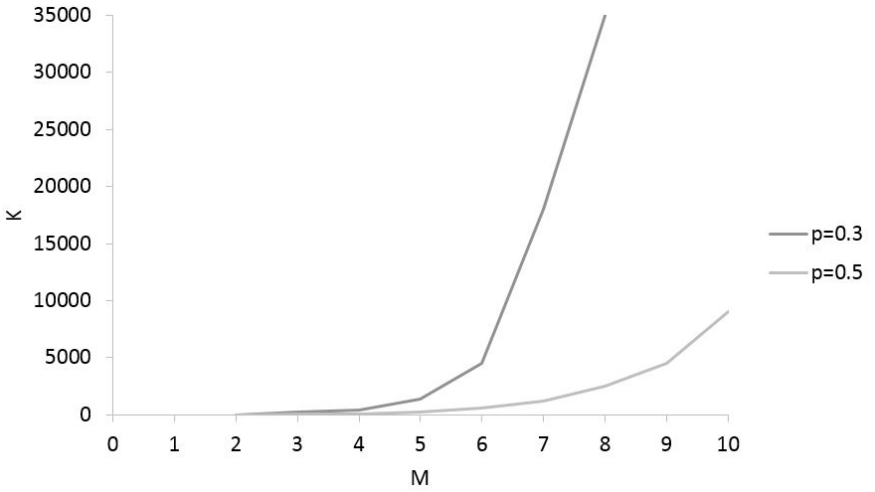

Figure 7. The saturation value (K) in dependency of M, p=0,3; p= 0,5

3. Approximation of the cost function

In order to find an appropriate analytical approximation function, we have first normalized the experimental functions. The normalization on the axis y means that the cost value is transformed into the [0,1] interval. The same normalization can be performed along the axis x too. With the application of the normalization, we can transform the shape of the cost function into a form independent from the current value of M. Thus the resulted graphs can be used to describe the cost function for every M value. Fig. 8 and Fig. 9 show the normalized form of the cost function given with a red color.

The analysis of the normalized cost functions shows that the have a common shape which is very similar to the gamma distribution. The gamma distribution is a continuous probability distribution with two parameters [2]. The two parameters are the shape parameter (k) and the scale parameter (θ). The probability density function of the gamma distribution is given with the following formula:

$$f(x, k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)}$$

where

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$$

The corresponding distribution function is given with

$$P(k, \theta x) = \frac{1}{\Gamma(a)} \int_0^{\theta x} z^{k-1} e^{-z} dz$$

The approximation algorithm was implemented in Matlab using the system function gamcdf [3].

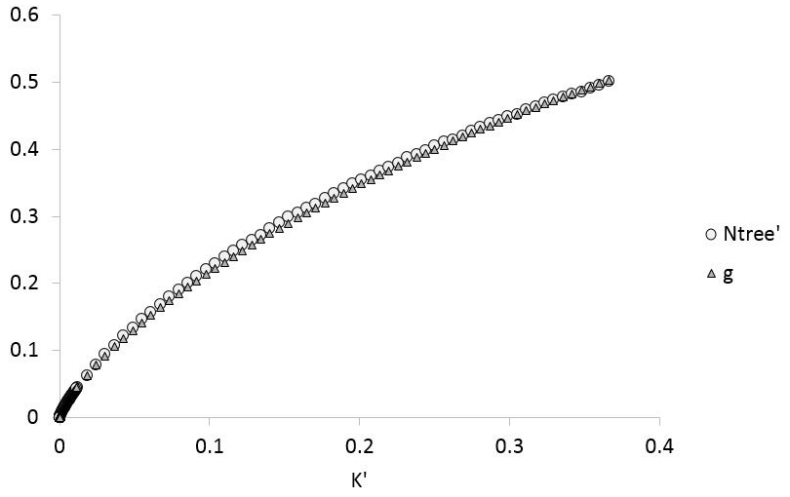

Figure 8. The normalized cost function in dependency of normalized K values, M = 15-50 p = 0.5, k = 0.75; theta = 0.8

Using the regression method for (k, θ) to determine the best matching $f(x, k, \theta)$ gamma distribution, we could achieve a very good approximation of the experimental cost functions. In Fig 8. and Fig 9, the approximation gamma distribution functions are given in color blue.

4. Conclusion

In the performed analysis, we have tested the size function of the prefix tree in dependency from the size and value distribution of the input data on an analytical and experimental way. Test results show that simulation-based experiments provide a good approximation of the analytical cost functions. After normalization of the experimental cost functions, we have found that

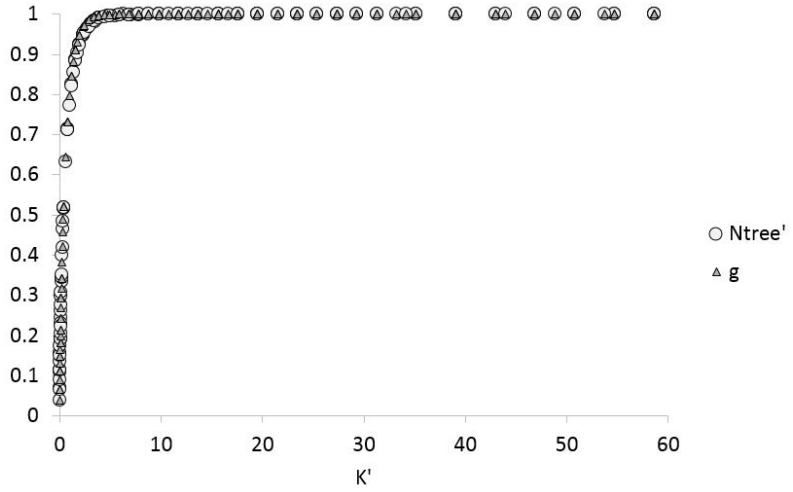

Figure 9. The normalized cost function and the gamma approximation in dependency of normalized K values, M =8; 9; 10; p = 0.5; k = 0.75; theta = 0.8;

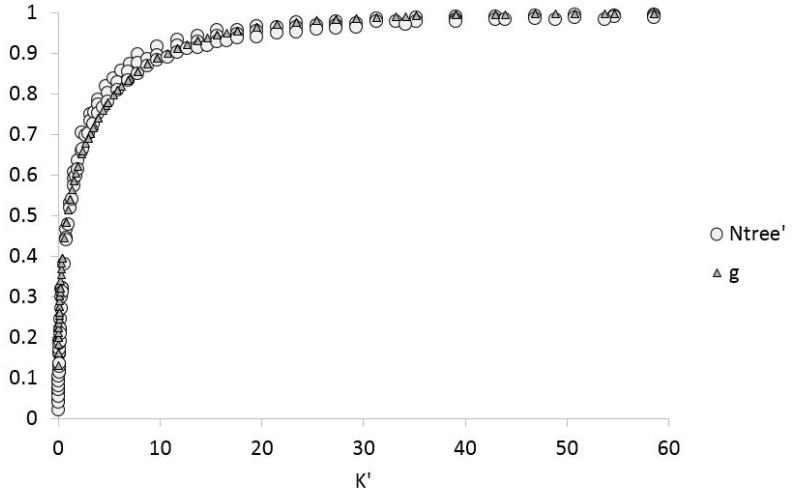

Figure 10. The normalized cost function and the gamma approximation in dependency of normalized K values, M = 8; 9; 10; p = 0.3; k = 0.3; theta = 12

the normalized cost functions can be efficiently approximated with a gamma distribution. In the next phase of the research project, the goal of the analysis

will be the development of an efficient method for calculation of the scale and shape parameters of the best matching gamma distribution.

REFERENCES

[1] M. Hamedanian, M. Nadimi and M. Naderi: *An Efficient Prefix Tree for Incremental Frequent Pattern Mining*, International Journal of Information and Communication Technology Research, Vol 3 (No 2), 2013, pp. 49-56

[2] L. Kenneth: *Numerical analysis for statisticians*, Springer Science and Business Media, 2010.

[3] MATLAB Documentation. [Online]. Available: https://www.mathworks.com/help/stats/gamcdf.html. [date: 19-12-2016].

[4] A. Cayley : *A theorem on trees*, Quart. J. Maths. Vol. 23 (1889), pp. 376-378

[5] C. Chauve, S. Dulucq and O. Guibert: *Enumeration of some labelled trees*, Private Communication

[6] T.C. Cheng: *On computing distinguishing numbers of trees and forests*, The Electronic Journal of Combinatorics Vol. 13 (2006) *https://doi.org/10.37236/1037*


Production Systems and Information Engineering
Volume 8 (2019), pp. 51–67
https://doi.org/10.32968/psaie.2019.004

APPLYING INTRUSION DETECTION ALGORITHMS ON THE KDD-99 DATASET

Mohammad Almseidin
University of Miskolc, Hungary
Department of Information Technology
alsaudi@iit.uni-miskolc.hu

Maen Alzubi
University of Miskolc, Hungary
Department of Information Technology
alzubi@iit.uni-miskolc.hu

Mouhammd Alkasassbeh
Mutah University, Jordan
Information Technology Department
mouhammd.alkasassbeh@mutah.edu.jo

Szilveszter Kovacs
University of Miskolc, Hungary
Department of Information Technology
szkovacs@iit.uni-miskolc.hu

Abstract. Practical task of information reliability and security is the effective intrusion detection and prevention. Open systems are vulnerable. Having in detail information about system structures, more and more sophisticated network intrusion methods could be easily developed and quickly tested. Intruders are always keeping update information about the current technology and generate new intrusion methods. There are several defense solutions against intrusions. The most common solution is Intrusion Detection System (IDS). For giving a short overview of some IDS methods, this paper applies the commonly available KDD-99 dataset for compare and discuss the IDS performance in case of different intrusion types. In this paper, the IDS performance of the J48, Random Forest, Random Tree, Decision Table, Multi-layer Perceptron (MLP) and Naive Bayes Classifier compared based on the average accuracy rate, precision, false positive and false negative performance in case of DOS, R2L, U2R, and PROBE attacks. Moreover, the focus would be on false alarm values. During the tests, the random forest algorithm produced the highest average of accuracy rate 93.77%, while the Random tree algorithm had the lowest rate 90.57%. The lowest value of false negative was produced by the decision table algorithm.

Keywords: KDD-99 dataset, Intrusion Detection, The Denial of service attack, Data mining Algorithms

1. Introduction

Information technology had a rapid development in the last two decades. Computer networks are widely used by industry, business and in various fields of the human life. Maintaining the reliability of networks became an essential task of the IT administrators. On the other hand, the rapid development also produces several challenges and the question of network reliability became a very difficult task. There are many types of attacks threatening the availability, the integrity and the confidentiality of computer networks. The Denial of Service attack (DoS) considered as one of the most common harmful attacks.

The aim of DoS attacks is to temporarily deny services for the end users. In the most common case, it consumes the network resources and overloads the system with undesired requests. For this reason the DoS acts as a large umbrella of naming for all types of attacks which aim to consume computer and network resources. In 2000 Yahoo was the first victim of a DoS attack, which was also the date, when the DoS recorded its first public attack [1]. Nowadays web services and social websites are the main target of DOS attacks [2].

From another vulnerability perspective, the remote to local (R2L) attacks are another common types of attacks which are designed to gain local access permissions remotely in case if some network resources (e.g. servers) are protected by allowing access only for local users. There are several types of R2L attacks e.g. SPY and PHF. These types of attacks aim to prepare illegal remote access to the network resources [3].

Related to the illegal access to the network and computer resources, the type of User to Root (U2R) attacks aim to switch the attacker access permission from normal user to the root user, who has full access rights to the computers and network resources [4]. The main challenge is that attackers are always keeping up-to-date their tools and techniques for exploiting any kind of vulnerabilities appearing to be known. Hence, it is very difficult to detect all types of attacks based on single fixed solutions. For that Intrusion Detection System (IDS) became an essential part of network security. It is designed to monitor the network traffic and generate alerts when any attacks appear. IDS can be implemented to monitor network traffic of a specific device (host IDS) or to monitor all the network traffics (network IDS) which is the most common type used.

Conceptually there are two types of IDS, Anomaly based IDS and Misuse based IDS. Anomaly based IDS implemented to detect attacks based on the recorded normal network behavior. It compares the current real time traffics with the previously recorded normal traffics. This type of IDS is widely used

because it has the ability to detect the new (previously unknown) type of intrusions, too. On the other hand, conceptually it registers the largest values of false positive alarms too, for the situations, which is normal, but not recorded among the "normal network behavior" samples (e.g. there is an uncommonly large number of normal packets considered to be attacking traffic).

Misuse intrusion detection systems are implemented to detect attacks based on a repository of attack signatures. Conceptually it has no false positive alarms but a new type of attack (which signature is missing from the repository) can succeed to pass-through as a normal traffic.

According to [5], attacks detection considered as a classification problem because the target is to clarify whether the packet either a normal or an attack packet. Therefore, an IDS can be built based on the methodology of machine learning algorithms.

To compare the IDS performance of different machine learning algorithms, in this paper, the following algorithms have been studied: J48, Random Forest, Random Tree, Decision Table, Multi-layer Perceptron (MLP) and Naive Bayes Classifier. For the model formation and evaluation the publicly available KDD-99 benchmark dataset was applied. The studied attack types were DOS, R2L, U2R, and PROBE.

The rest of the paper is organized as follows: section (2) summarizes the work related to the IDS application of the KDD-99 dataset and briefly introduces the applied machine learning algorithms. The preprocessing steps of the KDD-99 dataset are discussed in section (3). Section (4) gives a brief overview of the selected data mining algorithms that are used in the experiments. In section (5) some details of the IDS model forming is presented briefly. Section (6) introduces the applied metrics used for evaluating the performance of the IDS methods and discusses the experiments and the achieved results. Finally, section (7) concludes the paper.

2. IDS and the KDD-99 dataset

The commonly available KDD-99 is the data set used at The Third International Knowledge Discovery and Data Mining Tools Competition [6] for the task of building a network intrusion detector. The competition was held in conjunction with KDD-99 The Fifth International Conference on Knowledge Discovery and Data Mining in 1999. Although KDD-99 dataset is rather old, it is still widely used in academic research for testing and comparing IDS performance [7]. Because of its unceasing popularity, for comparing and discussing the IDS performance in case of different intrusion types in this paper also the KDD-99 dataset is chosen. In [2], for a classifier selection model, the authors

made a deep survey of IDS and the KDD-99 dataset. They extracted 49,596 instances of KDD-99 dataset to implement several machine learning algorithms e.g. Naive Bayes and MLP. Authors succeeded to propose two models for detecting intrusions types of the KDD-99 dataset. In [8] the authors applied a MATLAB implementation of Support Vector Machine (SVM) algorithm for IDS. They used the KDD-99 dataset as an IDS benchmark data. They claimed that the SVM algorithm needs long training time and hence the usability of SVM is limited. In [9] the authors preprocessed the KDD-99 dataset, symbolized and normalized the attributes to the [–1, 1] range. Then a feed forward neural network was applied in two experiments. The authors concluded that the neural network is not efficient for detecting R2L and U2R attacks but it has acceptable accuracy rate in detection DOS and PROBE attacks. In [10] the authors are implemented Fuzzy ARTMAP, Radial-basis Function, Back propagation (BP) and Perceptron-back propagation-hybrid (PBH) IDS. The four algorithms evaluated and tested on the KDD-99 dataset, in which the BP and PBH algorithms achieved the highest accuracy rate. Another research direction focuses on attributes selection algorithms in order to reduce the cost of the computation time. In [11] authors are focusing on selecting the most significant attributes to design IDS that have a high accuracy rate with low computation time. They implemented the IDS based on extended classifier and neural network to reduce false positive alarm as much as possible. In [12] the information gain algorithm was implemented to be an effective attributes selection method for improving the DoS intrusion detection. The genetic algorithm (GA) was also implemented to enhance detection of different intrusion types. In [3] the author proposed a methodology to derive the maximum detection rate with the minimum false positive rate. The GA was applied to generate a number of effective rules to detect intrusions. They achieved 97% accuracy on the KDD-99 dataset. In [13] the Naive Bayes algorithm was applied to detect all intrusions types of the KDD-99 dataset. The authors concluded that the detection rate is unacceptable if they apply only a single IDS algorithm. Some IDS research is focusing on a specific type of attack. In [14] a new Distributed Denial of Service (DDoS) dataset is collected from the samples of http ood, smurf, SiDDoS and udp ood attacks data. The DDoS dataset then tested with different IDS algorithms. For detecting the DDoS intrusions, the MLP algorithm achieved the highest accuracy rate (98.36%). Another example for applying the KDD-99 dataset for evaluating different IDS methods can be found in [15], where the performance of 20 different classifiers were compared on different attack categories. Regarding to the implemented experiments the Multivariate Adaptive Regression Splines (MARS) algorithm getting a higher accuracy rate. Furthermore, the fuzzy logic obtained an accepted accuracy rate compared with other implemented algorithms. Moreover,

the lowest accuracy rate recorded by Partial Decision Tree (PART) algorithm. Additionally, The acceptable IDS should perform with an accepted average accuracy rate and lowest possible false negative value.

The KDD-99 dataset still provides a reasonable benchmark environment for testing and evaluating various machine learning algorithms. It is also important to note, that a single machine learning algorithm could not provide an acceptable detection rate. One solution for this problem is the application of different IDS algorithms for detecting various type of attack threats. In the followings seven types of Machine Learning and Data mining Algorithms (J48, Random Forest, Random Tree, Decision Table, MLP, Naive Bayes, and Bayes Network) will be implemented, tested, compared and evaluated based on KDD-99 dataset. Our interest is directed to the most important performance parameters, like false negative and false positive attack detections. We would like to select the most promising IDS methods which could achieve an acceptable accuracy rate with the minimum false negative detections.

3. Preprocessing the KDD-99 dataset

The KDD-99 dataset can serve as a good sample for several intrusion behaviors, and good benchmark for testing and evaluating intrusion detection algorithms. The KDD-99 dataset first published by the MIT Lincoln labs at the University of California in 1999 and still available in UCI Machine Learning Archive [16]. It includes 4,898,431 instances with 41 attributes.

The first step of the IDS model generation is the preprocessing of the dataset. For this reason in our case the KDD-99 dataset was first imported to an SQL server 2008, then various statistical measurements values e.g. distribution of instances records, attacks types and occurrence ratios were calculated. Fig. 1 presents the main preprocessing steps of the KDD-99 dataset.

Statistical measurements provide a deep understanding of this dataset in order to extract impartial experiments. Table 1 illustrates the distribution of the attacks types within KDD-99 dataset. There are 21 type of attacks, which can be categorized into four groups with different number of instances and occurrences. 79% of the instances are related to DOS attacks, 19% are belong to normal packets and 2% can be categorized as other attacks types. Based on these values the KDD-99 appears to be an unbalanced dataset. The packets have 41 attributes.

These attributes are basic information which can be collected during the TCP/IP connection [4]. Table 2 illustrates these fundamental TCP/IP attributes. One important contribution of the KDD-99 dataset, that it also contains 32 expert suggested attributes which can help the understanding of

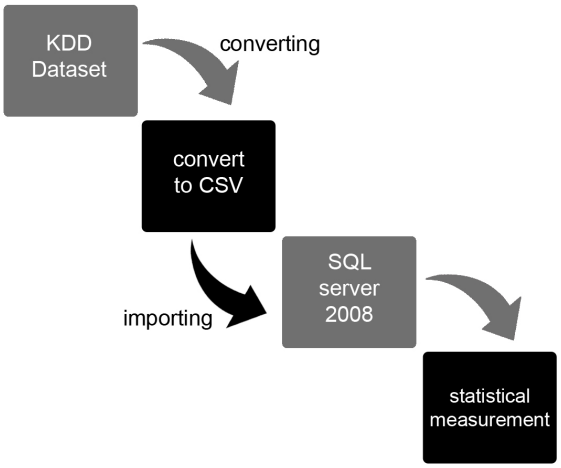

Figure 1. Preprocessing steps of the KDD-99 dataset

the behavior of an attack type. I.e. the most significant attributes of the four attack groups (DOS, R2L, U2R and PROBE) are also included.

4. The applied Data Mining algorithms

This section provides a brief overview of the machine learning algorithms applied for the IDS classification tasks in the rest of the paper. Machine learning algorithms can be categorized as supervised and unsupervised algorithms [17]. Supervised algorithms learn for predicting the object class from pre-labeled (classified) objects. The unsupervised algorithm finds the natural grouping of objects given as unlabeled data. In our IDS study supervised learning algorithms will be applied, as the imported KDD-99 dataset includes predefined classes.

J48 Classifier: This classifier is designed to improve the implementation of the C.4.5 algorithm, which is introduced by Ross Quilan [18] in 1993. The output of this classifier is in the form of decision binary trees, but with more stability between computation time and accuracy then the original C.4.5 [19]. The decision about the expected output is the leaf node of the decision tree structure.

Decision Table Classifier: The main idea of this classifier is to build a lookup table for identifying the predicted output class. There are several algorithms e.g. breadth first search, genetic algorithm and cross validation can be implemented to generate an efficient decision table [20]. The lookup table includes

Table 1. The distribution of the attack types within the KDD-99 Dataset

Categories of Attack	Attack name	Number of instances
DOS	SMURF	2,807,886
	NEPTUNE	1,072,017
	Back	2,203
	POD	264
	Teardrop	979
U2R	Buffer overflow	30
	Load Module	9
	PERL	3
	Rootkit	10
R2L	FTP Write	8
	Guess Passwd	53
	IMAP	12
	MulitHop	7
	PHF	4
	SPY	2
	Warez client	1,020
	Warez Master	20
PROBE	IPSWEEP	12,481
	NMAP	2,316
	PORTSWEEP	10,413
	SATAN	15,892

Table 2. The fundamental attributes of a TCP/IP connection

Attributes	Type
Total duration of connections in second	continuous
Total number of bytes from sender to receiver.	continuous
Total number of bytes from receiver to sender	continuous
Total number of wrong fragments	continuous
Total number of urgent packets	continuous
Protocol type	discrete
Type of service	discrete
The status of the connection (normal or error)	discrete
Label (1) if the connection established from to the same host. Otherwise label (0)	discrete

a set of conditions and the expected classes. These are the rules of the decision table classifier, which are predicting the classes for the incoming inputs [21].

The rules of the decision table can also be fuzzyfied, this case the Decision Table Classifier can also handle uncertainties of the inputs and classes.

Multi-layer Perceptron (MLP) Classifier: MLP is one of the most common algorithms that proved its effectiveness to deal with several application areas e.g. time series classification and regression problems [22]. During the implementation the testing phase can be short, but the training phase typically needs a long time. MLP algorithm can be implemented with various transfer functions e.g. Sigmoid, Linear and Hyperbolic. During the implementation the number of outputs, or expected classes is straightforward, but the number of the hidden layer neurons should be correctly defined for having an effective MLP classifier. At the beginning, every node within the neural network had its randomly weight and bias values, the large weight values in the input layer present the most effective attributes within a dataset, and on the contrary, the small weight values present the least effective attributes within a dataset.

Naive Bayes Classifier: This classifier refers to the group of probabilistic algorithms. It implements Bayes theorem for classication problems. The first step of Naive Bayes classifier is to determine the total number of classes (outputs) and calculate the conditional probability for each dataset classes. After that, the conditional probability is calculated for each attribute. The standard formula of Naive Bayes can be found e.g. in [10]. Furthermore, it has the ability to work with discreet and continuous attributes too. On the contrary of MLP classifier Naive Bayes can be implemented within a short period of time [13]. The Naive Bayes Classifier can be represented as a Bayesian Network (BN) or a Belief Network. BN presents independent conditional probabilities based on understanding framework. In general BN is an acyclic graph between expected class (output) and a number of attributes [23].

Random Tree Classifier: It is one of the classification tree algorithms. The random tree classifier is a finite group of decision trees. The number of trees must be fixed in advance. Each individual tree represents a single decision tree. Each individual tree has randomly selected attributes from the dataset. The entire dataset is predicted from several decision trees outputs and choose the winner expected class based on total numbers of votes [24].

Random Forest Classifier: It is one of the ensemble learning algorithms. The main goal of this algorithm is to enhance trees algorithms based on the concept of the forest. Random forest algorithms [25] have an acceptable accuracy rate. It can be implemented to be able to handle noise in the dataset. It is averaging multiple decision trees, trained on different parts of the same dataset. The number of trees must be fixed in advance. Each individual tree within a forest predicts the expected output. Then the expected output selected by a voting technique [25].

5. Generating the IDS Models

There are 21 types of attacks appearing in the KDD-99 dataset. These attacks are categorized into four groups (DOS, R2L, U2R, and PROBE). Each attack types has different number of instances and occurrences in dataset.

After the preprocessing of the KDD-99 dataset, 148,753 instances of records have been extracted to an SQL server. This labeled data serves as a training set for the further IDS model generation. The attack categories and types with the number of instances are presented on Table 3. Based on the analysis of KDD-99 dataset the occurrence distribution of the different attack types was recorded. 79% of the extracted data present DOS attacks, 19% is related to the instaces of normal traffic and 2% is related to other types of intrusions (U2R, R2U and PROBE).

Table 3. The Training Model Dataset.

Categories of Attack	**Attack name**	**Number of instances**
DOS	SMURF	85,983
	NEPTUNE	32,827
	Back	70
	POD	10
	Teardrop	30
U2R	Buffer overflow	10
	Load Module	2
	PERL	1
	Rootkit	5
R2L	FTP Write	2
	Guess Passwd	10
	IMAP	4
	MulitHop	2
	PHF	1
	SPY	1
	Warez client	31
	Warez Master	7
PROBE	IPSWEEP	382
	NMAP	70
	PORTSWEEP	318
	SATAN	487
Normal		28,500

In this paper, the experiments were performed on an Ubuntu 13.10 platform, Intel R, Core(TM) i5-4210U CPU @ 1.70GHz (4CPUs), 6 GB RAM. The

applied machine learning tool was the Waikato Environment for Knowledge Analysis (WEKA) [26]. It is an open source tool written in JAVA and available for free. It provides all the classifiers referred in this paper. These are the J48, Random forest, Random Tree, Decision Table, Multilayer Perceptron (MLP), Naive Bayes and Bayes Network. Based on the preprocessed 148,753 instances according to the labeled attack categories (see attack types and categories on Table 3) all the seven studied classifiers were created. For the creation of the classifiers all the labeled data were processed as training data. The cross validation of the classifiers were omitted, as our goal was to compare the best available performance of the different type of classifiers based on an existing data (KDD-99 records), not on an unknown data set. Then the classifiers were saved for a comprehensive study introduced in the followings.

6. Performance of the IDS implementation

After the IDS model generation, the next step is the comparative study of the models. In order to implement a fair testing phase fully randomized 60,000 instances have been extracted from the preprocessed database. The extracted testing data included all the 21 attack types of the KDD-99 dataset and labeled according to the attack categories introduced on Table 3. There are several metrics that can be used for evaluating the efficiency of the IDS model. In this paper, the confusion matrices were generated for each classification algorithms. Furthermore, the following performance metrics [14] were computed:

- **True Positive (TP)**: This value represents the correct classification of the attack packets as attacks.
- **True Negative (TN)**: This value represents the correct classification of the normal packets to be normal traffic.
- **False Negative (FN)**: This value represents an incorrect classification, where the attack packet classified as normal packet. A large FN value presents a serious problem of confidentiality and availability of network resources, because the attacker succeeded to pass through the IDS.
- **False Positive (FP)**: This value represents incorrect classification, where the normal packet classified as an attack. The increasing of FP value increases the computation time, but it is considered as less harmful than the increased FN value.
- **Precision**: Is one of the primary performance indicators. It presents the total number of records that are correctly classified as attack divided by a total number of records classified as attack. The precision can be calculated as follows:

$$P = \frac{TP}{(TP + FP)} \tag{6.1}$$

In addition, the number of both correctly and incorrectly classified instances are recorded with respect to the time taken for proposed training model.

During the testing phase, the following classification parameters were applied:

- **J48 tree classifier**: confidence factor = 0.25; numFolds = 3; seed = 1; unpruned = False, collapse tree = true and sub tree rising = true.
- **Random forest classifier**: number of trees = 100 and seed = 1.
- **Random tree classifier**: min variance = 0.001 and seed = 1.
- **Decision Table classifier**: Best First Search (BFS) and cross value = 1.
- **MLP classifier**: search learning rate = 0.3, momentum = 0.2, validation threshold = 20.

Table 4 presents the TP rate and the Precision values of the studied classification algorithms during the experiments. It can be concluded that the random forest classifier achieved the highest 93.1% TP rate, and the random tree classifier achieved the lowest 90.6% TP rate. I.e. the random tree classifier has the lowest correct attacks classification value. The decision table classifier reached the lowest 94.4% precision value. This indicates that the decision table classifier suffers from an increasing false positive value. Therefore, there is a large number of normal packets classified as attack packets.

Table 4. The True Positive Rate and the Precision

Classification Algorithms	**TP Rate**	**Precision**
J48	0.931	0.989
Random forest	0.938	0.991
Random tree	0.906	0.992
Decision table	0.924	0.944
MLP	0.919	0.978
Naive Bayes	0.912	0.988
Bayes Network	0.907	0.992

In general, the TP rate and precision values are important performance parameters for a common intrusion detection system, but from another perspective the most serious performance parameters are the FP rate and the FN rate. The goal of this study is to decrease both of these parameters, as much as possible, especially the FN parameters. The FP and FN performance

parameters of the IDS tests are summarized on Table 5. It can be concluded, that the random tree classifier achieved the highest 0.093 FN rate. Hence there is a large number of attacks classified as normal packet. On the contrary with the decision table classifier which is achieved the lowest 0.002 FN rate. In the same time, the decision table classifier reached the highest 0.073 FP rate. It means that there is a large number of normal packet classified as attack packets.

Table 5. The False Positive Rate and the False Negative Rate

Classification Algorithms	**FP Rate**	**FN Rate**
J48	0.005	0.063
Random forest	0.001	0.061
Random tree	0.001	0.093
Decision table	0.073	0.002
MLP	0.014	0.066
Naive Bayes	0.002	0.085
Bayes Network	0.001	0.092

Table 6 presents the Root Mean Square Error (RMSE) and area under the Receiver Operating Characteristic (ROC). RMSE presents the difference between the actual and the desired outputs based on the confusion matrix. The model with lower value of RMSE indicates better output prediction efficiency, on the contrary large value of RMSE indicates lower prediction efficiency. The ROC value is calculated based on the true positive and the false positive values. The large value of ROC indicates that the model has better intrusion detection ability, while the lower value present the weakness of the model.

Table 6. The Root Mean Square Error and the Area under the Receiver Operating Characteristic

Classification Algorithms	**ROC Area**	**Root Mean Squared Error**
J48	0.969	0.0763
Random forest	0.996	0.0682
Random tree	0.953	0.0763
Decision table	0.984	0.0903
MLP	0.990	0.0813
Naive Bayes	0.969	0.0872
Bayes Network	0.997	0.0870

According to the results on Table 7, the Bayes network classifier achieved the highest 0.997 ROC value, while the random tree classifier achieved the lowest 0.953 value. Furthermore, the random forest classifier had the lowest 0.0682 RMSE value, while the decision table presented the highest 0.0903 value. After the classification of 60,000 instances of the KDD-99 dataset, the total number of incorrectly classified records for each selected classifier, and the average accuracy rate is presented on the Table 7. The average accuracy rate is calculated according to the following formula:

$$AverageAccuracyRate = \frac{TP + TN}{TP + FN + FP + TN} \tag{6.2}$$

Table 7. Average Accuracy Rate

Classification Algorithms	**Correctly classified Instances**	**incorrectly classified Instances**	**Accuracy Rate**
J48	55,865	4,135	93.10%
Random Forest	56,265	3,735	93.77%
Random tree	54,345	5,655	90.57%
Decision table	55,464	4,536	92.44%
MLP	55,141	4,859	91.90%
Naive Bayes	54,741	5,259	91.23%
Bayes Network	54,439	5,561	90.73%

It is important to mention, that it could take a long time to build the IDS model. Based on the experiments, the building the random tree classifier model is the fastest, while training the MLP classifier was taken about 176 minutes. In our experiments it was the longest model generation time. From the results of the tests, we can conclude the followings:

- **The Random forest** achieved the highest 93.77 accuracy rate with the smallest RMSE value and false positive rate.
- **The Random tree** classifier reached the lowest 90.73 average accuracy rate with smallest ROC value.
- **Regarding to** the average accuracy rate there is no big difference between the MLP classifier and the Naive Bayes classifier.
- **All classification** algorithms present acceptable precision rates for detecting normal packets.
- **Bayes network** classifier recorded the highest value for detecting correctly the normal packets.

- **There are no** big differences between the MLP and the J48 algorithms based on FN parameters.
- **Despite** that the decision table classifier did not reached the highest accuracy rate, but it had the lowest FN rate. The model generation time was also acceptable.
- **All of the tested** classification algorithms had acceptable model generation time, except the MLP.
- **It can be concluded** that the rule based algorithms (decision table) are presented an acceptable accuracy rate with the lowest FN rate, which is increasing the confidentiality and the availability of the network resources.

7. Conclusions and Future Works

The KDD-99 dataset was applied for measuring the performance of seven classification algorithms (Random tree, Random forest, Naive Bayes, MLP, Decision table and J48) in IDS performance. The KDD-99 dataset includes instances from 21 types of attacks from four attack groups (DOS, R2L, U2R, and PROBE). Each attack types has different number of instances and occurrences in dataset. In our tests first the IDS models for the seven studied classification algorithms were generated. Then their IDS performances were tested based on randomly chosen KDD-99 data.

According to our experiments, from 60,000 randomly chosen testing records, the random forest algorithm achieved the highest 93.77% accuracy value. During the same test it has 3,735 incorrectly classified records. The random tree algorithm achieved the lowest 90.57% accuracy value with 5,655 incorrectly classified records. Regarding to the root mean squared error values, also the random forest algorithm achieved the lowest 0.0682 value, while the decision table algorithm had the highest 0.0903 value. The Naive Bayes algorithm needed the shortest model generation time, while the MLP algorithm reached the longest 176 minute training time.

All the seven studied classification algorithms achieved acceptable precision for detecting normal packets. The decision table algorithm had the lowest 0.002 false negative value, which means that it can detect various intrusion types of the KDD-99 dataset successfully. The effectiveness of any IDS always suffers from false negative values. The acceptable IDS should perform with the lowest possible false negative value. Consequently, as a part of the future work, we would like to modify the rule based decision table algorithm to a fuzzy rule based system to generate an IDS model which can achieve an acceptable accuracy rate with the lowest possible false negative classification.

8. Acknowledgement

The described study was carried out as part of the EFOP-3.6.1-16-00011 "Younger and Renewing University – Innovative Knowledge City – institutional development of the University of Miskolc aiming at intelligent specialization" project implemented in the framework of the Szechenyi 2020 program. The realization of this project is supported by the European Union, co-financed by the European Social Fund.

REFERENCES

[1] G. C. Kessler, "Defenses against distributed denial of service attacks," *SANS Institute*, vol. 2002, 2000.

[2] H. A. Nguyen and D. Choi, "Application of data mining to network intrusion detection classifier selection model," in *Challenges for Next Generation Network Operations and Service Management: 11th Asia-Pacific Network Operations and Management Symposium, APNOMS 2008, Beijing, China, October 22-24, 2008. Proceedings*, vol. 5297. Springer Science & Business Media, 2008, p. 399.

[3] S. Paliwal and R. Gupta, "Denial-of-service, probing & remote to user (r2l) attack detection using genetic algorithm," *International Journal of Computer Applications*, vol. 60, no. 19, pp. 57–62, 2012.

[4] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 2009, pp. 1–6. [Online]. Available: https://doi.org/10.1109/cisda.2009.5356528

[5] P. Amudha, S. Karthik, and S. Sivakumari, "Classification techniques for intrusion detection-an overview," *International Journal of Computer Applications*, vol. 76, no. 16, 2013. [Online]. Available: https://doi.org/10.5120/13334-0928

[6] "Kdd cup 1999 data," 1999, accessed on 01.07.2017. [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[7] A. Ozgur and H. Erdem, "A review of kdd99 dataset usage in intrusion detection and machine learning between 2010 and 2015," *PeerJ Preprints*, vol. 4, no. e1954v1, 2016. [Online]. Available: https://doi.org/10.7287/peerj.preprints.1954v1

[8] M. K. Lahre, M. T. Dhar, D. Suresh, K. Kashyap, and P. Agrawal, "Analyze different approaches for ids using kdd 99 data set," *International Journal on Recent and Innovation Trends in Computing and Communication*, vol. 1, no. 8, pp. 645–651, 2013.

[9] F. Haddadi, S. Khanchi, M. Shetabi, and V. Derhami, "Intrusion detection and attack classification using feed-forward neural network," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*. IEEE, 2010, pp. 262–266. [Online]. Available: https://doi.org/10.1109/iccnt.2010.28

[10] Z. Zhang, J. Li, C. Manikopoulos, J. Jorgenson, and J. Ucles, "Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification," in *Proc. IEEE Workshop on Information Assurance and Security*, 2001, pp. 85–90.

[11] W. Alsharafat, "Applying artificial neural network and extended classifier system for network intrusion detection." *International Arab Journal of Information Technology (IAJIT)*, vol. 10, no. 3, 2013.

[12] N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria, "Decision tree analysis on j48 algorithm for data mining," *Proceedings of International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 6, 2013.

[13] C. Fleizach and S. Fukushima, "A naive bayes classifier on 1998 kdd cup," 1998.

[14] M. Alkasassbeh, G. Al-Naymat, A. B. Hassanat, and M. Almseidin, "Detecting distributed denial of service attacks using data mining techniques," *International Journal of Advanced Computer Science & Applications*, vol. 1, no. 7, pp. 436–445. [Online]. Available: https://doi.org/10.14569/ijacsa.2016.070159

[15] S. O. Al-mamory and F. S. Jassim, "Evaluation of different data mining algorithms with kdd cup 99 data set," *Journal of Babylon University/Pure and Applied Sciences*, vol. 21, no. 8, pp. 2663–2681, 2013.

[16] S. D. Bay, "The uci kdd archive [http://kdd. ics. uci. edu]. irvine, ca: University of california," *Department of Information and Computer Science*, vol. 404, p. 405, 1999.

[17] M. Al-Kasassbeh, "Network intrusion detection with wiener filter-based agent," *World Appl. Sci. J*, vol. 13, no. 11, pp. 2372–2384, 2011.

[18] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[19] M. S. Bhullar and A. Kaur, "Use of data mining in education sector," in *Proceedings of the World Congress on Engineering and Computer Science*, vol. 1, 2012, pp. 24–26.

[20] R. Kohavi and D. Sommerfield, "Targeting business users with decision table classifiers." in *KDD*, 1998, pp. 249–253.

[21] P. Aditi and G. Hitesh, "A new approach of intrusion detection system using clustering, classification and decision table," 2013.

[22] S. K. Pal and S. Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on neural networks*, vol. 3, no. 5, pp. 683–697, 1992. [Online]. Available: https://doi.org/10.1109/72.159058

[23] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[24] A. Cutler and G. Zhao, "Pert-perfect random tree ensembles," *Computing Science and Statistics*, vol. 33, pp. 490–497, 2001.

[25] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009. [Online]. Available: https://doi.org/10.1145/1656274.1656278


Production Systems and Information Engineering
Volume 8 (2019), pp. 69–87
https://doi.org/10.32968/psaie.2019.005

C/C++ APPLICATIONS ON THE WEB

Martin Szabó
University of Miskolc, Hungary
Department of Information Engineering
sz.martin91@gmail.com

Károly Nehéz
University of Miskolc, Hungary
Department of Information Engineering
aitnehez@uni-miskolc.hu

[September 2017 and accepted November 2017]

Abstract. The JavaScript technology, especially in the latest five years, has been evolving very rapidly. In the world of WWW, of course, other technologies are also available for developers, but JavaScript is one of the best ways to develop applications for both traditional web and the latest mobile environments. The modern web application frameworks unfortunately do not support direct integration of C/C++ technologies. This problem can be solved by various utility software, e.g. the Emscripten compiler, which translates C/C++ codes into JavaScript leveraging LLVM technology as a transitional layer. Thus C/C++ programmers are not omitted from the world of Web and mobile development and they can reuse existing mature codebases, especially in the field of computer graphics. This paper describes theory, practical use and inherent potential deals of Emscripten technology. A native C++ OpenGL application will be used as a real-world example, demonstrating efficiency and flexibility of this technique. Performance evaluation of the JavaScript code compared to the native C/C++ application will also be presented. To promote better understanding, our source code is also available upon request.

Keywords: C/C++, JavaScript, LLVM, Emscripten, OpenGL

1. Introduction

Evolution of mobile and web technology development more and more excludes C/C++ programmers from the web client and mobile software market. The reason is that users prefer thin-client applications which can be launched on

mobile devices and in desktop browsers. Applications are written in JavaScript easier to be carried over between different browsers/platforms, easy to maintain and can be developed faster. These advantages can reduce the cost of development. Also, there are some different optimization techniques to increase performance, and reduce resource requirement of applications at the same time. In contrast, high-level C/C++ programming language is harder to port between different platforms, and require a recompilation in the most cases. But, in case of complex and resource-intensive tasks (i.e. game engine development), it's still worth choosing C/C++ technologies.

Based on the current studies [1] programmers prefer more C/C++, than JavaScript. In additional, a lot of complex systems have been written in C/C++. These applications are constantly under maintenance and upgrades, so they are still doing their job well (so they are still able to do their job well). So the question is, why cannot we use these great systems via web or mobile devices? The traditional idea is that, we should rewrite source code of the whole application. But this problem can be solved with a new approach, via Emscripten technology.

Using Emscripten, applications written in C/C++ programming language, can be translated to JavaScript. During compilation process, the code translated to LLVM bit code, so it can be optimized. The result will be a less resource-intensive, but outstanding performance JavaScript-based software. Unlike the original code, the applications have been created this way, will be able to run on mobile devices and in browsers. So the recompiling code automatically provides the benefits of JS technology.

This article can be divided into four parts. The first part contains description of the essential theoretical background of Emscripten framework, including a solid theoretical description of JavaScript, and its advantages and disadvantages will be presented. As an integral part of Emscripten, we present the structure and the most important functions of the LLVM compiler. The second part of the paper presents the Emscripten itself, and contain a guide required to use the technology. The third section shows how to use the framework in practice, via a simple graphical application. It includes the constraints of the technology, and other tasks required to achieve a proper final result. The last part describes our experiences gained through the testing of the framework, and contains some benchmark data, we got from the C/C++, and the translated JavaScript application.

2. Background

A basic knowledge of C/C++ language is required to use Emscripten technology. The usage of the compiler can be problematic without understand the code appears at the frontend of the framework (even if we have a lot of source files (even in the case of having a lot of source files). There will be more information about this issue in Section 4. Furthermore, the JavaScript code generated by the Emscripten does not require additional development work, it is necessary to briefly introduces the script language, including its advantages and disadvantages.

JavaScript [2] is an object-oriented, prototype-based scripting language that is mainly used for Web sites, either as an integral part of an html file or in a separated JS file. It started to spread in the 1990s, and still widely used as the part of the websites, with HTML, and CSS. Other programming languages also can be run on the web, such as Java, and Flash, but in these cases, the extensions are not integrated into the browser applications, but must be manually installed. Furthermore these are not platform independent tools, for example usually cannot be run on iOS devices. Because of these issues JavaScript becomes the most widely used programming language to create the client side of web applications. The advantages of the language are the following. software created by JavaScript can be run on desktop, and mobile browsers too. It runs on client side, the development is relatively easy, quick, and does not require recompilation. Disadvantages include the fact, that it is browser dependency and slow speed. Although it standardized, but different engines (layout engines) can interpret it in different ways, which leading to operational inconsistencies. Compared to C/C++ applications, it significantly provides less performance, but with optimization procedures of LLVM compiler (which is the part of the Emscripten technology) a majority of the lost performance can be recovered.

Before creating the JavaScript code, Emscripten compiles the C/C++ sources into LLVM bit code. This procedure can give a quasi-optimal code as a final result, which is already ready to run in browsers. LLVM compiler is a very important, and useful part of Emscripten, so this paper deals with its theoretical background as well.

We have collected and investigated other technologies designed to run a specific programming language on the Web. For example Google Web Toolkit [3] which translates Java to JavaScript, Pyjamas (Python → JavaScript), SCM2JS [4] (Scheme → JavaScript), and AFAX [5] (F# → JavaScript) frameworks are serve this purpose too. But, in this paper we are focusing C/C++, and these tools are not suitable for our goals, so we will not deal with them.

2.1. Low Level Virtual Machine (LLVM)

Emscripten framework uses LLVM to optimize the output of the JavaScript code. The LLVM compiler has been designed to optimize any programming languages in compile-time, link-time, run-time, and in idle time. A robust open source application, that competes with GCC in terms of compilation speed and the performance of the generated code. As a consequence, it has been widely used in both academia and industry [7].

The code representation describes the code as an abstract RISC-like (reduced) instruction set, but also provides higher level information for more efficient analysis. Including type information, explicit control flow graphs, and an explicit dataflow representations. It supports language independent instruction set and type systems. The instructions stored in SSA (Static Single Assignment) form, which helps to simplify analysis of dependencies between variables [8].

The LLVM code representation has the following novel properties [7]:

1. Low-level, language-independent type-system for implement data types and operations from high level languages, eliminating the primitives at all stages of optimization. The type system includes type information used by sophisticated (but language-independent) techniques, such as algorithms for pointer analysis, dependence analysis, and data transformation. In other words: simple, language-independent type system, that exposes the primitives generally used to implement high level programming language features.
2. Instruction set for performing type conversions and low-level address arithmetic without losing the type information.
3. Two, low-level exception-handling instructions for effectively implementing high-level language specific exceptions with simple methods.

The LLVM compiler is source-language-independent due to low-level instruction set, and memory model. The technology slightly offer more than the standard assembly languages, and the type system does not prevent to add type information to the code. It also does not have specific runtime requirements [7].

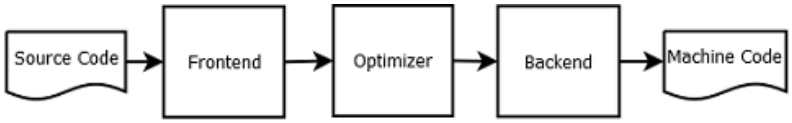

Figure 1. Simplified model of LLVM

The simplified usage of the compiler framework can be described as follows: The source code at the frontend (written in any programming language) translated to machine code. Then different optimization methods (depends on the user choice) optimize the source. Finally the result will be runnable code at the backend, written in an appropriate programming language for the target architecture (Figure 1).

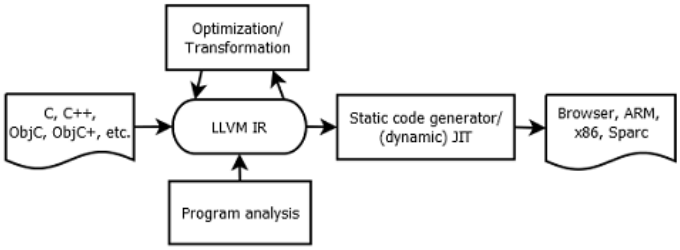

Figure 2. Detailed model of LLVM

The LLVM-based compiler allows to compile the high level programming language resources to LLVM IR language (Figure 2). The IR (Intermediate Representation) is assembly-like low-level programming language that allows optimization, code compilation, and static analysis. The resulting code runs on a variety of systems, including for example the X86/64, and the ARM architectures. The technology allows static translation (as usual in the case of the GCC), but also supports dynamic JIT (Just-In-Time) translation. The project created mainly for C/C++ front-end, but has the ability to compile any other programming language too, for example Haskell, Scheme, Scala, Objective C etc. [6]

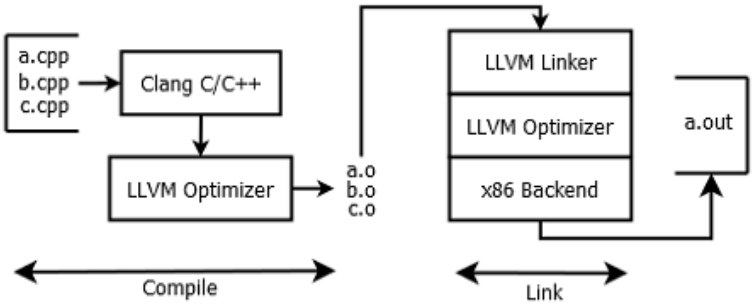

Figure 3. LLVM LTO

Most of the modern compiler environments support optimization in linking time (LTO)(Figure 3). The disadvantage of LTO is that, it can deal with only one unit simultaneously (like a C file, with its headers) and cannot optimize a larger context. LLVM compilers, (like Clang) support this type of optimization with the *–flto*, and *–o4* commands. These settings indicate to the compiler, that it needs to produce .o file from the LLVM byte code instead of a native object file, and that, the code generation have to be in the linking phase, as shown in Figure 3. The linker detects, that the generated .o files are not native object files, but contain LLVM byte code, therefor it reads every byte code file into the memory, and after linking executes optimization processes on the entire code. As the optimizer unit oversees the entire structure now, it can perform its tasks more effectively [9].

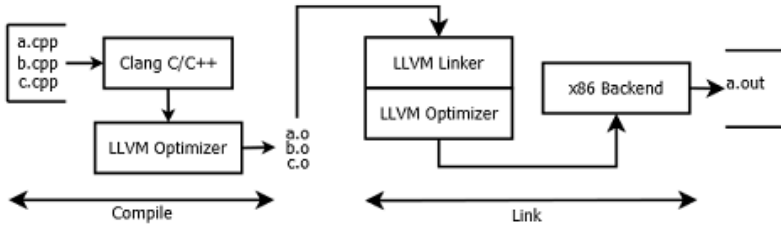

Figure 4. LLVM ITO

Installation time optimization (ITO) can take into account special features of the target architecture as well (eg. MMX, SSE, AVX). The advantage of this method is that, the optimizer unit will recognize the device specifications in all cases, so the architectural benefits are exploitable(Figure 4).

3. Emscripten Conception

Applications, written in C / C++ language are generally "make life easier for the computer" so it can achieve higher performance. In contrast, applications written in JavaScript make life easier for programmers, but some loss in performance occurs. If high performance is essential for a given kind of software (for example, video-game engines, distributed systems etc.) the C/C++ language is inevitable. Of course, it is not expected to make the same application for multiple platforms, by the reason of the short deadlines, generated by rapidly developing technology. This problem can be solved by the Emscripten technology that converts the C/C++ source code for the web and/or the mobile version of the same product, with minimal modification (adds Emscripten specific methods etc., see Chapter 4). Build-in LLVM compiler also eliminates

the above-mentioned performance loss using various optimization methods. The behavior of the resulting software is same as the original application, but its availability (due to the web environment) and its performance (due to the optimization) is greater (Figure 5).

Figure 5. Emscripten compiling conception [11]

Emscripten framework is written in JavaScript language, and it can be downloaded from the following link under MIT open source license: http://www.emscripen.org. The compiler is designed to fix parts of the original high-level structure of the code, that were lost during the compilation to low level LLVM byte code, when creating the JavaScript code. In the following we describe the LLVM – JavaScript compilation process, and present the Relooper algorithm, which is able to create high level loop structure from the low level branching data.

There are two available methods to use this technology: the first option is the pre-translation (static) mode, the second is the runtime translation (dynamic) mode. Then we translate the resulting code to JavaScript with the help of Emscripten. The latter option is useful when the run time of the language is written in a programming language, that has, but the language itself has no frontend. For example, currently Python does not have a frontend, but it is possible to translate CPython (Python standard C implementation) to JavaScript, and run Python code. Frontends for various languages exist, including C/C++, and also various new and emerging languages (e.g., Rust) [11].

3.1. Optimization process

Many (mostly big, but sometimes smaller) application's code, during the software development processes become too complex, so the errors almost inevitable, and the final performance can be far below what is expected. The most common problem is that the developers do not invest enough emphasis on optimization, and do not clarify various dependencies like dynamic libraries, etc., because these limitations have not been appeared on the platform, which the application is designed. If we would like to run the application on another hardware, it might not be able to ensure proper performance, and/or it will be difficult to port to other platform.

Emscripten itself does not provide optimization automatically without the appropriate command line settings. With default settings, the framework generates the JavaScript code without any optimization (i.e. it contains more than necessary variable declaration). For example it stores every variable in an array, and controls the flow of execution using a switch-in-a-loop, instead of normal JavaScript loops and ifs. Before compiling the C/C++ code to JavaScript, the Emscripten technology optimize it with the help of the built-in LLVM, furthermore the Closure Compiler give some more optimization for the .js code afterwards. These procedures remove unneeded variables, eliminate dead code and inline functions [11]. There are two important optimization methods, as follows:

1. **Variable nativization:** converts every variable that are on the stack to native JavaScript variables. The Emscripten tries to implement this process to the maximum number of variables during the optimization processes. Except those variables, which were used outside of a function, or were passed to another method.
2. **Relooping:** recreate the high level loop and of structures from the low level LLVM assembly code.

3.2. Relooper algorithm

The Relooper is the most complex module of Emscripten. It generates high-level JavaScript flow structures (loops, ifs), Emscripten code blocks from set of labeled fragments of code [11]. The JS engines are designed to run the codes as fast as possible, so the structure of the code have to be manufactured to comply with this restriction. To present the steps of the Relooper algorithm we have to define the three different blocks of Emscripten. These are shown below [11]:

1. **Simple:** contains one internal label, and a reference to the Next block, which the internal label branches to.
2. **Loop:** it represents a basic loop, and contains two internal sub-blocks (the Inner block, that appears inside the loop, and the Next block, that appear outside the loop).
3. **Multiple:** it represents branches joining each other, and also contains two internal sub-block, the Handled, and the Next blocks.

The steps of the Relooper algorithm are as follows [11]:

1. Receive a set of labels and entry point. The goal is to make a block from the labels.
2. Tracking one of the possible execution path, it calculates for every label which other labels it can reach.

3. If there is only one entry point, and cannot return back, then creates a *Simple block*, where the entry point is the internal label, and the *Next block* contain every other label. The entry points of the *Next block* are those points, which the internal label can branch.
4. If it can return all of the entry points, creates a *Loop block*, whose *Inner block* contains all of the labels, which can reach one of the entry points, and whose *Next block* contains every other labels.
5. If there are more than one entry point, then it creates a *Multiple block*. Find all labels that cannot be reached by other entry points for all entry point.
6. If it cannot create a *Multiple block*, then it prepares a *Loop block* as described above.

4. How to Use Emscripten

The installer available at the following link: http://www.emscripten.org.
A lot of tutorials are available on the same link too. Source codes can be found in the Emscripten's sub-folder after the installation. The framework is available on Windows, Linux, and iOS as a command-line environment. The official website provides detailed descriptions about the technology, but in this paper we present a more complex example via a C/C++ graphical application. The manual does not complete, is shows only the functions we utilized. The demo application is available at the following link: http://goo.gl/YmV2EA. (Figure 6)shows the the demo application.

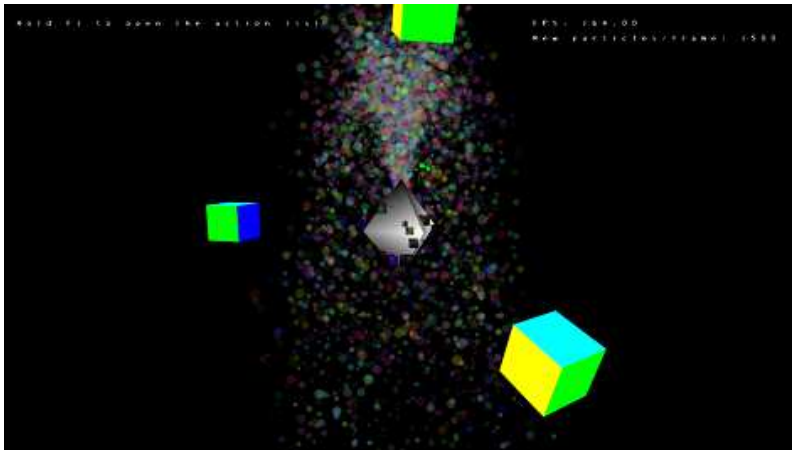

Figure 6. Demo application

4.1. Coding process

Usually if the source code does not contain graphical elements, then it does not require interventions before use. However, most of the code of graphical applications have to be prepared for different environments (i.e. if the code contains infinite loop [gameloop]). This is the job of application developers. The code transformation process will work, but the JavaScript code will not run properly in the browsers. The JavaScript code can run only in one thread, so these solutions cannot be used without modifying the code. To solve this program the developers of the Emscripten technology creates the *emscripten_set_main_loop (function, int fps, int simulate_infinite_loop)* method. The *function* parameter indicates the primary method have to be done by *while* loop; *fps* means frames per second (0: unlimited); and the last parameter represents the infinite loop simulation (0: run only once, 1: infinite loop). In our application we used this method as follows:

```
#ifdef EMSCRIPTEN
#include <emscripten.h>
#endif

void loop(){
/* draw models, translations, rotations */
/* draw particles */
}
int main( void ){
/* initializing GLEW */
/* Creating matrices, textures, particle system, models (coordinates, colors) */
#ifdef __EMSCRIPTEN__
  emscripten_set_main_loop(loop, 0, 1);
  #else
  do{
  loop();
  }
  while( glfwGetKey(window, GLFW_KEY_ESCAPE ) != GLFW_PRESS &&
```

glfwWindowShouldClose(window) == 0);

#endif

}

In order to use the Emscripten specific methods in our code, we have to include *emscripten.h* header file. With the use of *#ifdef* , we can compile the source code and run the application as a C/C++ and Emscripten versions too without performing any changes in the code. So, if we would like to run it as a C/C++ application, Emscripten specific parts will not compile, on the other hand, if we would like to transform the source code to JavaScript, then these parts will compile too.

Developers of the Emscripen offer the SDL window manager library to use. In contrast we used GLFW3, and our experiences show that this library is fully compatible with the Emscripten as well. However, the necessary header files are not included in the framework. The solution to this problem is simple, we only need to copy the appropriate files into the Emscipten's *include* folder. But the SDL tools do not require any action from the developers.

To move and to rotate three-dimensional shapes, it is required to perform matrix transform operations. The GLM libraries contain the necessary methods for this. This header files are not part of the Emscripten framework, but easy to install them, as described above.

The application has been made in OpenGL 3.3 environment. To draw the models, we originally used GLSL 3.3 (OpenGL Shading Language) language, which is a high-level language based on the C programming language syntax. Most of the sample programs we found on the internet use this version as well. However, we realized that the Emscripten only supports the WebGL compatible GLSL ES 1.0, and 2.0 shader languages, and GLSL 3.3 code cannot be automatically converted to one of these. We can solve this problem with manual code modifications. The following table shows the differences between the two mentioned shader language implementation through the vertex and the fragment shader we used in our application.

Table 1: Differences between the vertex shaders

GLSL 3.3	GLSL ES 1.0

`#version 330 core`
`layout(location = 0) in vec3`
`vertexPosition modelspace;`
`layout(location = 1) in vec3`
`vertexColor;`
`out vec3 fragmentColor;`
`uniform mat4 MVP;`
`void main() {`
`gl_Position = MVP *`
`vec4(vertexPosition_modelspace, 1);`
`fragmentColor = vertexColor;`
}
	`attribute vec3`
	`vertexPosition modelspace;`
	`attribute vec3 vertexColor;`
	
	`varying vec3 fragmentColor;`
	`uniform mat4 MVP;`
	`void main() {`
	`gl_Position = MVP *`
	`vec4(vertexPosition_modelspace, 1);`
	`fragmentColor = vertexColor;`
	}

Table 2: Differences between the fragment shader

GLSL 3.3	GLSL ES 1.0

`#version 330 core`
`in vec3 fragmentColor;`
`out vec3 color;`
`void main() {`
`color = fragmentColor;`
 	
}
	`precision mediump float;`
	`varying vec3 fragmentColor;`
	`/* varying vec3 color; */`
	`void main() {`
	`gl_FragColor =`
	`vec4(fragmentColor, 1.0);`
	}

Table 3: Summary of the differences between the two versions

GLSL 3.3		**GLSL ES 1.0**	
vertex shader	**fragment shader**	**vertex shader**	**fragment shader**
#version xxx core	#version xxx core	–	precision mediump float
layout(location = x) in	in	attribute	varying
out	out	varying	varying
	out *variable's name*		gl_FragColor
	texture		texture2D

There were no further issues has been reported during the translation of the sample application. However, we tried to compile a complex video-game engine, but a new issue appeared. The software contained some old, OpenGL 1.x specific methods that Emscripten was unable to translate correctly. This problem should be solved by the *–s LEGACY_GL_EMULATION = 1* argument, but based on our experiences, some functions are not useable yet. Note that: the framework could not translate the following OpenGL methods:

1. `glListBase`
2. `gluBuild2DMipmaps`
3. `glDeleteList`
4. `glDrawPixels`
5. `glLightf`
6. `glVertex2D`
7. `glGenLists`
8. `glEndList`
9. `glCallLists`
10. `glNewList`

4.2. Compiling process

The compiler framework can only work in a command-line environment. For large-scale projects, it is subservient to create a Makefile, because of the numerous source files and other auxiliary files, like models, shader, and textures. This prevents (using different technologies, changing name, or number of the source files etc.) you from typing the – usually long – commands into the command prompt again.

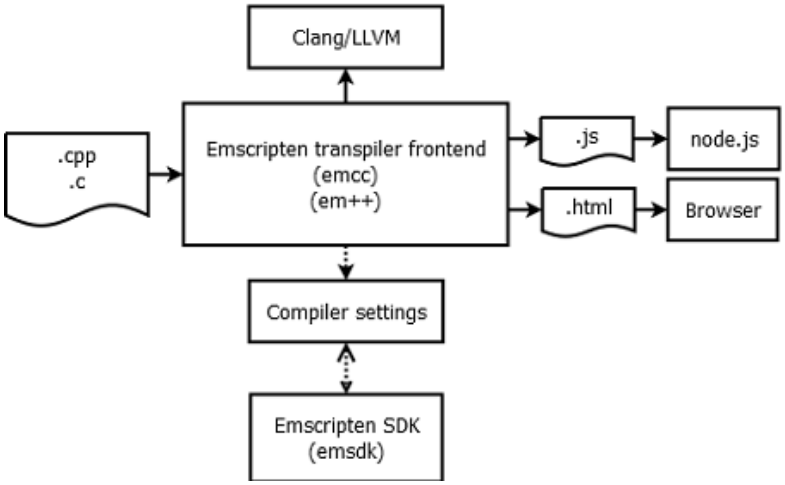

Figure 7. Detailed model of Emscripten

The compiler input is the C/C++ files and the output depends on the extensions specified (.js, .html). It can be .bc (or .o), .js, and .html. The first represents pure LLVM byte code, the second represents JS code, and the last one represents JS code embedded into html template. Tracking and setup the runtime parameters are easier using the html template. The code obtained in this manner will be immediately executable in desktop and mobile browsers (Figure 7). The LLVM byte code cannot run on its own.

The command which is required to compile the application is divided into five parts. The first part is the emcc (in case of C source), or the em++ (in case of C++ source) command. This is followed by the list of source files, then the switches required for the used technologies. If other files are needed to run the application properly (models, textures, shader etc.) Emscripten has to preload these, so we need to mark them as well. In this case, a .data file will be generated (besides .js and .html files), which will contain these information. The last part of the command will indicate the output file.

emcc main.c shader.c –s USE_SDL = 2 –preload-file texture.jpg –o index.html

Switches and other necessary commands we used in our application are shown in the following table with explanation:

Table 4: Command line switches for the different technologies [10]

-s USE_SDL = 2	allow SDL2
-s USE_SDL_IMAGE = 2	allow SDL_image
-s USE_GLFW = 3	allow GLFW3
-s FULL_ES2 = 1	allow GLSL ES2
-s LEGACY_GL_EMULATION=1	force to handle OpenGL 1.x methods
-s DEMANGLE_SUPPORT = 1	convert class and function names
-std=c++11	handle c++11 specific opportunities (i.e. lambda) kezelése
–preload-file dir/fájl_neve	load files (without file name the whole directory will be loaded)

Each switch performs different operations depending on what is required. (Table 5) We analyzed the results given by all the available switches. Our experiences will be summarized later.

Table 5: Switches for optimization [10]

-O0	No optimization
-O1	Simple optimization. Fast, but not really increase the performance. Eliminates run-time exception handling assertions. Relooping.
-O2	Same as the –O1 switch, but extended with JS optimization. It is useful in preparing the final version of the application. The translation is slow, however, this method provides the smallest and the fastest JS code.
-O3	Same as the –O2 switch, but provides more JS code optimization.
-Os	Same as the –O2 switch, but provides ,,extra" optimization, that reduces the size of the code, and increases the speed of the application. Only for byte code optimization.
-Oz	Same as the –Os switch, but takes further code size reduction. Only for byte code optimization.
-s DISABLE_EXCEPTION _CATCHING = 0	Enable exception handling
-s AGGRESSIVE_VARIABLE _ELIMINATION = 1	Eliminate unnecessary variables
-s ALLOW_MEMORY _GROWTH = 1	Unlimited memory usage

As explained above, it is always useful to create a *makefile* for our projects – especially if it contains a large number of files – to facilitate the translation process, and the practical testing. With this method it is enough editing the *makefile*, instead of typing all of the commands again and again. Of course, the resulting *makefile* is not compatible with the Emscripten system, so we need to perform some changes. The converted, Emscripten compatible *makefile* contains the following code:Emscripten makefileW9067CC = emcc
SOURCES := $(wildcard *.cpp)
LDFLAGS = -O2 –llvm-opts 2
OUTPUT = glcore.html
F = –preload-file
all: $(SOURCES) $(OUTPUT)
$(OUTPUT): $(SOURCES)
$(CC) $(SOURCES) –bind -s USE_GLFW = 3 -s FULL_ES3 = 1 -std = c++11

```
$(F) ColorFragmentShader.fragmentshader $(F) TransformVertexShader.vertexshader
$(F) TextVertexShader.fragmentshader $(F) TextVertexShader.vertexshader
$(F) Particle.vertexshader $(F) Particle.fragmentshader $(F) particle.DDS $(F)
font.DDS $(LDFLAGS) -o $(OUTPUT)
clean:
rm $(OUTPUT)
rm $(OUTPUT).mapW9067
```

5. Performance Analysis

Based on our experiences there are some cases, when the use of this technology is simple, and the C/C++ source code does not require any code modification. However, there are – mainly – graphical applications, where we have to modify the source code of the original C/C++ application. Fortunately, these changes are quite simple, only infinite loops ([*gameloops*]) should be changed as described above.

Another problem is that, although a lot of files needed to develop a graphical software are built into the compiler, the service is not complete, some of the methods we have to use, are not part of the package. Mostly, these are the functions used in the older OpenGL 1.1 technology. The personal solution to this problem is not an easy task, it requires programming work within the Emscripten framework. According to our research, the most appropriate solution is the Regal graphics library package [14]. Furthermore there are some other tools, which are unavoidable in many cases, but the framework does not include them. For example the GLM library package for matrix operations, or the GLFW3 windows manager technology (instead of SDL/SDL2). However, in these cases, we only need to copy the necessary header files into to include folder of the Emscripten, as described above.

The files (models etc.), used by the framework, need to be preloaded, and this must be done by the developer. It is not a problem, but it may cause a minor inconvenience. However, there are many other suitable solutions, for example setting the root folder, wrapping the files etc.

We compiled and launched our graphical application with every possible optimization parameter. Our benchmark analysis includes CPU usage, memory usage, rendering speed (FPS), size of the resulting code, and speed of the translation process. The study was extended to the original native C/C++ application as well. The measurements were made with 1,500 new particles/frame. The results are summarized in the following table.

Table 6: Benchmarks

optimization level	**CPU usage [%]**	**memory usage [MB]**	**speed [FPS]**	**size of the JS code [KB]**	**translation speed [sec]**
original C/C++	~ 23.7	~ 84.9	~ 53	429	0.33
O0 (none)	~ 12.1	~ 35	~ 115	2686	5.07
O1 (js)	~ 9.4	~ 31	~ 142	2169	4.81
O2 (js)	~ 8.2	~ 26	~ 162	581	6.62
O3 (js)	~ 7.7	~ 19	~ 162	553	6.79
Os (LLVM)	~ 8.1	~ 27	~ 162	546	6.80
Oz (LLVM)	~ 8.0	~ 28	~ 160	549	6.83

6. Summary

With the help of the Emscripten technology, application codes written in almost any high level languages (such as C/C++, Python etc.) can be transformed into JavaScript code, thus the original applications can be used on the web, or on even mobile devices. This technology can optimize source codes on many ways during the translation process. This feature is available due to the LLVM (Low Level Virtual Machine) compiler framework. Emscripten is accessible through a command-line environment and if the user would like to use various technologies (SDL, GLFW, OpenGL 1.x methods etc.), load files (models, textures, fonts etc.) or select optimization level and several different other options are available.

A C/C++ graphical application has been implemented for the purpose of doing benchmark tests. The JS code can run in a browser, in WebGL environment. The required HTML file is created by the Emscripten too. We used OpenGL 3.3, GLSL, GLFW3, and GLM libraries for the graphical visualization. Our measurements 2include CPU utilization, and memory usage, FPS (frames per second) values, size of the codes and translation speed. The tests were performed with all the available optimization techniques. The source code of sample application is available here: http://goo.gl/YmV2EA.

This paper contains the theoretical foundations for the technology acquisition, including JavaScript, LLVM and Emscripten. We have presented practical usage of the workspace and our experiences through our sample application.

Acknowledgements

The described article was carried out as part of the EFOP-3.6.1-16-2016-00011 Younger and Renewing University Innovative Knowledge City institutional development of the University of Miskolc aiming at intelligent specialisation project implemented in the framework of the Szechenyi 2020 program. The realization of this project is supported by the European Union, co-financed by the European Social Fund.

REFERENCES

[1] TIOBE Index for February 2016, `http://www.tiobe.com/tiobe_index?page=index`

[2] D. Flanagan. *JavaScript: The Definitive Guide.* O'Reilly Media, 2006

[3] C. Prabhakar. *Google Web Toolkit: GWT Java Ajax Programming.* Packt Publishing, 2007

[4] F. Loitsch and M. Serrano. Hop *Client-Side Compilation. InTrends in Functional Programming,* SetonHall University, Intellect Bristol, 2008

[5] T. Petek and D. Syme. *AFAX: Rich client/server web applications in F#.* Draft. Retrieved April 2011

[6] `http://www.llvm.org`

[7] C. Lattner and V. Adve. *LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation,* 2004

[8] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. *Efficiently computing static single assignment form and the control dependence graph.* October 1991

[9] Chris Lattner. *LLVM,* 2004.

[10] Official homepage of Emscripten: `http://www.emscripten.org`

[11] Alon Zakai. *Emscripten: An LLVM-toJava-script Compiler,* 2013

[12] Joel Galenson, Cindy Rubio-González, Sarah Chasins, Liang Gong. *Research.js: Evaluating Research Tool Usability on he Web,* University of California, Berkeley, 2014

[13] Malek Musleh, Vijay S. Pai. *Architectural Characterization of Client-side JavaScript Workloads & Analysis of Software Optimizations,* Purdue University, 2015

[14] Source code of Regal tool: `https://github.com/p3/regal`


Production Systems and Information Engineering
Volume 8 (2019), pp. 89–100
https://doi.org/10.32968/psaie.2019.006

CLUSTERING-BASED OPTIMISATION OF MULTIPLE TRAVELING SALESMAN PROBLEM

Anita Agárdi
University of Miskolc, Hungary
Institute of Information Science
agardianita@iit.uni-miskolc.hu

László Kovács
University of Miskolc, Hungary
Institute of Information Science
kovacs@iit.uni-miskolc.hu

[Accepted December 2017]

Abstract. The TSP is the problem to find the shortest path in a graph visiting every nodes exactly once and returning to the start node. The TSP is shown to be an NP-hard problem. To provide an acceptable solution for real life problems, the TSP are usually solved with some heuristic optimization algorithms. The paper proposes a clustering-based .problem decomposition algorithm to form the global route with merging of best local routes. Based on our test results, The proposed method can improve the efficiency of the standard heuristic methods for the TSP and MTSP problems.

Keywords: Traveling Salesman Problem, Clustering, Heuristic optimization

1. Introduction

In base Traveling Salesman Problem (TSP), a sales agent should visit all cities exactly once, except the a depo city which is the source and terminal of the route. The cities are represented with the vertices of a directed graph. The edges are weighted and the goal is to find a route visiting all cities and having a minimal total weight. The problem to build an optimal route corresponds to find an optimal permutation of the vertices creating a minimal length route. The TSP algorithm can be applied to solve many different problems, like

- planning of an optimal transportation route for both delivery and collection tasks [2],

- heuristic solving of the knapsack optimization problem [3],
- planning of optimal production sequence [1],
- printing press scheduling problem [4].

Based on the literature, the following variants of the TSP problem can be highlighted:

- base Traveling Salesman Problem (TSP): single depot position and single salesman,
- Multiple Traveling Salesman Problem (MTSP): single depot position and many salesmen,
- Multi Depot Multiple Traveling Salesman Problem (MDMTSP): many depots and many salesmen; every salesman is assigned to a single given depot.

The TSP is a NP-hard problem [10], that means it is at least as hard as the hardest problems in NP. The cost of a brute-force algorithm is in $O(N!)$, thus some heuristic should be used to find an approximation of the global optimum.

From the family of popular evolutionary algorithms, the following methods are used most widely to find the god approximation of the optimal route:

- Genetic algorithm [5]: the set of parameter vectors is optimized using selection, crossover and mutation operators.
- Particle swarm optimization [6]: more agents are generated which move randomly but the optimum found by them is reinforced by other members of the colony.
- Ant Colony Optimization [7]: more agents are generated which collaborate one another and with their environment.
- Tabu search [8]: the local search phase is extended with prohibitions (tabu) rules to avoid unnecessary position tests.

Considering the standard heuristic algorithms, we can highlight the following methods:

- Nearest neighbor algorithm [9]: it select the nearest unvisited node as the next station of the route.
- Pairwise exchange of edges [12]: two disjoint edges are removed from the route and two new edges are involved into the route to reduce the total cost.

In many real applications the nodes corresponds to points in the space, where the weights correspond to the distances between these positions. In this paper, we are focusing on the problem where the nodes represent points in the Euclidean space. We assume that there exists always a connection between any

node pair within the graph. Considering the TSP with nodes in Euclidean-space, we can observe that the optimal route usually connects neighboring nodes, i.e. the adjacent nodes are near to each others.

In this paper, we introduce a two-phase route planning method with clustering of the cities. In the first phase, the nodes are partitioned into disjoint clusters. For every cluster, a local optimum route is generated. In the second phase, these local routes are merged into a global route. The global route is then refined using some standard heuristic methods.

The different cities are clustered into disjoint groups where a group contains similar nodes. The similarity of two nodes is measured with the distance value between the corresponding positions in the Euclidean space. To merge the local routes into a global route, the optimum route of the clusters and the optimum connection edges must be determined within a separate optimization module. In Multiple Traveling Salesman Problem, we can use the clustering method to determine the set of nodes assigned to the same salesman. Every salesman must visit the cities of the same cluster.

The efficiency of the proposed two-phase TSP method is tested with the following heuristic TSP optimization methods: Hill-climbing algorithm; Nearest neighbor algorithm and Nearest Insertion algorithm.

2. Formal description of the TSP

The formal model of TSP can be given with the following linear programming description:

- N: number of nodes in the graph.
- u_i : the position of the i-th city in the solution path.
- D: distance matrix; d_{ij} is the weight of the edge from the i-th node to the j-th node; the distance values are non-negative values.
- X: adjacency matrix of the Hamilton cycle; $x_{ij} = 1$ if there is a directed edge in the path from the i-th node to the j-th node; otherwise $x_{ij} = 0$.
- the objective function of the path optimization is: $\sum_{i=1}^{N} \sum_{j=1}^{N} d_{ij} x_{ij} \rightarrow min$.
- every node has only one incoming edge: $\forall j (i \neq j) : \sum_{i=1}^{N} x_{ij} = 1$.
- every node has only one outgoing edge: $\forall i (i \neq j) : \sum_{j=1}^{N} x_{ij} = 1$.
- there is only a single tour covering all cities: $u_i - u_j + N x_{ij} \leq N - 1$.

In the case of multi-salesman traveling (MTSP) problem more than one cycles should be generated (each salesman has a separate cycle) and each node is visited only by one salesman. For MTSP, the formal model should be extended with the following elements:

- M : number of salesmen.
- constraints on the depo node (start and stop): $\sum_{i=1}^{N} x_{i0} = M, \sum_{j=1}^{N} x_{0j} = M$.

Due to the high complexity of TSP, there exists no algorithm for global exact optimization with polynomial cost. For example, the Held-Karp algorithm solves the problem in $O(n^2 2^n)$ complexity. In order to provide an acceptable solution for real life problems, the TSP are usually solved with some heuristic optimization problem.

In the existing TSP algorithms, we use some heuristics to find an optimum route. In our investigation, we use the following methods: hill climbing algorithm; nearest neighbor algorithm and nearest insertion algorithm.

Having a context parameter set position p_0 , the hill climbing method evaluates some neighboring positions and selects the position with the best improvement factor. This selected position will the next context position. The algorithm terminates if no neighboring position with better fitness value is found. In order to avoid weak local optimum positions, some random relocation mechanism is added to the base algorithm. In this case, several initial positions are generated and processed. The 2-opt and 3-opt switch operations are the most widely used methods to generate the neighboring positions. The 2-opt method select two random edges from the context route and replace them with two new matching routes. For example, having a route $A \to a \to B \to b$ with two selected edges, a and b, the neighbor route is given with $A \to b \to B' \to a$, where B' denotes the inverse traversing of B. The 3-opt switch selects 3 edges and replaces them with 3 new edges to get a neighboring route.

In the case of Nearest Neighbor algorithm [13] uses a simple and sometimes efficient method to generate the optimum route. The algorithm starts with a context node selected randomly. In the next step, the method evaluates all free nodes and selects the node nearest to the context node. This node will be the next element of the route. The algorithm will then process this node as a new context node. This greedy algorithm terminates if all the nodes are merged into the route. The MTSP variant of the Nearest Neighbor method partitions the nodes into disjoint groups belonging to the different agents. The simplest way to perform a partitioning is to set a size threshold nl on the single routes. This constraint means that the greedy algorithm presented before terminates if the nodes in the route equals to nl. After completing a local route, the method starts to build up the local route for the next agent. Another specialty of this algorithm is that every local route starts with the depo node.

Also the Nearest Insertion method builds up the route on an incremental way. It starts with an node selected randomly. Having a context route, the algorithm evaluates all free nodes and calculates the minimal length increase related to inserting the selected node. Unlike the Nearest Neighbor method, the selected node can be inserted into any positions within the context route. The free node with the best minimal length increase is inserted into the route. In the case of MTSP, the method initially starts the generation of local route for ever agent.

3. Clustering methods

The clustering produces a partitioning of the elements where similar objects are assigned to the same group while unsimilar objects should be mapped to different groups. The input data can be given on two different ways. One option is to use attribute vectors to describe the objects where the similarity is calculated from the attribute values. Another option is to define the similarity matrix of the objects directly. The elements with high similarity should be assigned to the same cluster.

There are three main types of clustering methods:

- partitioning methods,
- hierarchical methods,
- density-based methods.

In the case of partitioning methods, in every iteration, a new partitioning is generated. Having an initial partitioning, the quality is improved by re-assignment of the items to different clusters. The re-assignment iteration is terminated if the improvement value is getting below a threshold. The K-means algorithm is a widely used partitioning-based clustering method.

In the hierarchical clustering methods, we take some extremum partitioning (for example every item is a separate cluster). In every iteration step, the current partitioning will be refined or it will be more coarse. The HAC [14] a known example of this algorithm group.

In the case of density-based clustering, a item-density value is calculated for every item position (or for the whole item domain). The clusters are defined as maximal connected dense areas. Regarding the quality, the following properties should be met by the selected partitioning algorithm [15]:

- scalability,
- supporting arbitrary cluster shape,
- detection of outlayer (noise) items,
- efficiency for high dimensional item domains.

The K-means clustering partitions the items into a fixed number of clusters. The number of the clusters is given as an input parameter set by the users. Initially, the cluster centers are set randomly. In every iteration step:

- the items are assigned to the nearest center,
- a new center point is calculated for every cluster,
- if the old and new centers are within a threshold distance, the iteration terminates.

The new cluster center is calculated with

$$x_c = \frac{1}{N_c} \sum_{i_c} x_{i_c}$$

The algorithm optimizes the sum of squared errors value:

$$\sum_{c} \sum_{i_c} d(x_c, x_{i_c})^2$$

There are two main approaches in the hierarchical clustering methods. The first approach uses an agglomerative, bottom-up construction concept, while the second approach is based on a divisive, top-down algorithm. In the case of agglomerative method, every item is a separate cluster initially. In the iterative loops, the two clusters having the largest similarity are merged in to a new single cluster. The HAC method is the most widely used hierarchical clustering method [14], where the clusters are merged on a greedy way. Two different termination conditions can be given: minimal number of the clusters or the minimal similarity for merging. The following methods are main methods to measure the similarity (or distance) of two cluster.

- single linkage method: the distance of two clusters is equal to the distance of the two nearest points

$$d(c_1, c_2) = min_{x \in c_1, y \in c_2} d(x, y)$$

- complete linkage method: the distance of two clusters is equal to the distance of the two farthest points:

$$d(c_1, c_2) = max_{x \in c_1, y \in c_2} d(x, y)$$

- centroid linkage method: the distance of two clusters is equal to the distance of the two center points:

$$d(c_1, c_2) = d(x_{c_1}, x_{c_2})$$

- average linkage method: the distance of two clusters is equal to the average distance related to any possible point pairs:

$$d(c_1, c_2) = \frac{1}{|c_1||c_2|} sum_{x \in c_1, y \in c_2} d(x, y)$$

4. Clustering extended heuristic methods

We propose the extension of the standard heuristics methods with a clustering phase in order to decompose the problem domain into a set of smaller domains. This decomposition can be considered as the application of the 'divide and conquer' approach in solving of the TSP optimization problems. To analyze the efficiency of clustering-based decomposition phase, we have developed the following algorithms.

Clustered hill-climbing for TSP:

1. Clustering of the nodes into disjoint groups (K: no specific constraint).
2. Calculate the cluster centers.
3. Hill-climbing TSP on the set of clusters to determine the cluster-level route.
4. Determining the input/output ports of the clusters.
5. Hill-climbing TSP for every cluster to determine the inner-level routes.
6. Merging the inner-level routes into a global route based on the cluster-level route.

Clustered hill-climbing for MTSP:

1. Clustering of the nodes into disjoint groups (K : number of the agents).
2. Extending every cluster with the common depot element.
3. (Clustered) Hill-climbing TSP for every cluster to determine the local routes for every agent.

Clustered hill-climbing for MDMTSP:

1. Clustering of the nodes into disjoint groups (K : number of the agents).
2. Extending every cluster with the nearest depot element taken also the capacity constraints into account.
3. (Clustered) Hill-climbing TSP for every cluster to determine the local routes for every agent.

Clustered nearest neighbor for TSP:

1. Clustering of the nodes into disjoint groups (K: no specific constraint).
2. Calculate the cluster centers.
3. Nearest neighbor TSP on the set of clusters to determine the cluster-level route.
4. Determining the input/output ports of the clusters.
5. Nearest neighbor TSP for every cluster to determine the inner-level routes.
6. Merging the inner-level routes into a global route based on the cluster-level route.

Clustered nearest neighbor for MTSP:

1. Clustering of the nodes into disjoint groups (K : number of the agents).
2. Extending every cluster with the common depot element.
3. (Clustered) nearest neighbor TSP for every cluster to determine the local routes for every agent.

Clustered nearest neighbor for MDMTSP:

1. Clustering of the nodes into disjoint groups (K : number of the agents).
2. Extending every cluster with the nearest depot element taken also the capacity constraints into account.
3. (Clustered) nearest neighbor TSP for every cluster to determine the local routes for every agent.

Clustered nearest insertion for TSP:

1. Clustering of the nodes into disjoint groups (K: no specific constraint).
2. Calculate the cluster centers.
3. Nearest insertion TSP on the set of clusters to determine the cluster-level route.
4. Determining the input/output ports of the clusters.
5. Nearest insertion TSP for every cluster to determine the inner-level routes.
6. Merging the inner-level routes into a global route based on the cluster-level route.

Clustered nearest insertion for MTSP:

1. Clustering of the nodes into disjoint groups (K : number of the agents).
2. Extending every cluster with the common depot element. (Clustered) nearest insertion TSP for every cluster to determine the local routes for every agent.

Clustered nearest insertion for MDMTSP:

1. Clustering of the nodes into disjoint groups (K : number of the agents).
2. Extending every cluster with the nearest depot element taken also the capacity constraints into account.
3. (Clustered) nearest insertion TSP for every cluster to determine the local routes for every agent.

In the case of multi-depot TSP variants, the depots may be fixed in advance or it can be located to any nodes. In our algorithm, we used the cluster-center depo location mechanism.

The implemented test framework provides the following functions:

- generation of test data,
- test data file level input/output,
- execution of the mentioned standard heuristic methods,
- solving TSP, MTSP and MDMTSP problems,
- adding cluster-based optimization,
- representation of the results in graph formats.

5. Test results

The evaluation tests were executed for small and medium sized problem domains with the following parameters:

Table 1. Test parameters

	N: number of nodes	A: number of agents	D: number of depots
small domain	10–100	1–10	1–10
medium domain	100–1000	1–100	1–100

In the tests, two main quantities were analyzed:

- length of the generated route (L),
- execution time (T) in ms.

The results for (N:1000, A:10, D:1) are summarized in the Table 2. The column CL denotes whether the algorithm includes a clustering phase or not.

Table 3 shows the summarized results for medium size input domain (N:1000, A:100, D:1).

The analysis of the test results show, that

- the clustering provides significantly improvement in fitness efficiency for the complex MTSP and MDMtSP problems,
- the clustering can improve the fitness efficiency of the standard heuristic methods for the TSP problem, too,
- the fitness efficiency of the proposed low cost clustering optimization module is comparable with the fitness efficiency of the complex compound optimization algorithms requiring a higher execution cost,
- the clustering phase requires additional 20%–80% computation costs.

Based on the performed experiences, the proposed cluster optimization phase is a promising approach to enhance the efficiency of the complex TSP problems. In the next phase of our investigation, we will focus on the development of an improved TSP-adjusted clustering method.

Table 2. Test parameters for small number of agents

method	CL	T tsp	T mtsp	T mdmtsp	L tsp	L mtsp	L mdmtsp
nearest neighbor	N	125	83	125	2,787	3,603	3,234
nearest neighbor	Y	105	125	152	3,151	2,978	3,053
nearest insertion	N	31	43	105	3,062	4,294	3,795
nearest insertion	Y	36	56	123	3,255	3,784	3,068
hill climbing	N	16,124	24,014	24,012	3,538	10,051	10,421
hill climbing	Y	18,154	17,297	18,354	2,626	3,099	2,673
nearest neighbor + hill climbing	N	16,154	25,341	25,887	2,530	3,070	2,896
nearest neighbor + hill climbing	Y	17,895	15,781	15,443	2,611	2,978	2,605

6. Conclusion

For TSP problems with high number of nodes, the multi layered optimization is superior to the single level optimization. In the proposed multi layered optimiza tion, the node set is partitioned into clusters or into hierarchy of clusters. For each cluster, a separate local TSP optimization is performed and then the local routes are merged into a global route. Based on the test experiments the proposed method is superior to the single level optimization method for both the TSP and MTSP problems

Acknowledgments. The described article/presentation/study was carried out with the support of the EFOP-3.6.1-16-00011 Younger and Renewing University InnovativeKnowledge City institutional development of the University of Miskolc aiming at intelligent specialisation project implemented in the framework of the Szechenyi 2020 program. The realization of this project is supported by the European Union, co-financed by the European Social Fund.

Table 3. Test parameters for medium number of agents

method	CL	T tsp	T mtsp	T mdmtsp	L tsp	L mtsp	L mdmtsp
nearest neighbor	N	125	62	62	2,784	10,122	3,889
nearest neighbor	Y	17	12	93	2,612	9,620	2,976
nearest insertion	N	31	31	26	3,062	13,425	7,182
nearest insertion	Y	181	125	62	3,405	9,994	2,784
hill climbing	N	16,124	24,452	24,002	3,538	22,974	14,762
hill climbing	Y	30,045	33,012	31,034	2,643	9,355	2,597
nearest neighbor + hill climbing	N	16,164	24,514	24,881	2,530	8,905	3,081
nearest neighbor + hill climbing	Y	33,012	32,012	30,875	2,625	9,322	2,602

REFERENCES

[1] Jeong, EY., Oh, S.C., Yeo, YK. et al. Application of traveling salesman problem (TSP) for decision of optimal production sequence *Korean J. Chem. Eng.* ,(1997) Vol 14, pp 416 -421. *https://doi.org/10.1007/bf02707062*

[2] Berbeglia, G., Cordeau, J. F., Gribkovskaia, I., Laporte, G. Static pickup and delivery problems: a classification scheme and survey. *Top*, Vol 15(1), (2007), pp. 1-31. *https://doi.org/10.1007/s11750-007-0009-0*

[3] Laporte, G., Martello, S. The selective travelling salesman problem. *Discrete applied mathematics*, Vol 26(2-3), (1990), pp. 193-207. *https://doi.org/10.1016/0166-218x(90)90100-q*

[4] Gorenstein, S. Printing press scheduling for multi-edition periodicals. *Management Science*, Vol 16(6) (1970), pp. B-373. *https://doi.org/10.1287/mnsc.16.6.b373*

[5] Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D. Genetic algorithms for the traveling salesman problem. *In Proceedings of the first International Conference on Genetic Algorithms and their Applications* (1985), pp. 160-168.

[6] Wang, K. P., Huang, L., Zhou, C. G., Pang, W. Particle swarm optimization for traveling salesman problem. *In Machine Learning and Cybernetics*, (2003) Vol. 3, pp. 1583-1585. *https://doi.org/10.1109/icmlc.2003.1259748*

[7] Dorigo, M., Gambardella, L. M. Ant colonies for the travelling salesman problem. *biosystems*, Vol 43(2), (1997), pp. 73-81. *https://doi.org/10.1016/s0303-2647(97)01708-5*

[8] Malek, M., Guruswamy, M., Pandya, M., Owens, H. Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem, *Annals of Operations Research*, Vol 21(1), (1989) pp. 59-84.*https://doi.org/10.1007/bf02022093*

[9] Lin, S., Kernighan, B. W. An effective heuristic algorithm for the traveling-salesman problem, *Operations research*, Vol 21(2), (1973) pp. 498-516 *https://doi.org/10.1287/opre.21.2.498*

[10] Held, M.; Karp, R. M., A Dynamic Programming Approach to Sequencing Problems, *Journal of the Society for Industrial and Applied Mathematics* Vol. 10 (1962), 196–210. *https://doi.org/10.1145/800029.808532*

[11] Sumanta Basu, Swarm Optimization Algorithm for the Traveling Salesman Problem, *EvoCOP 2006: Evolutionary Computation in Combinatorial Optimization* (2006), 99–110.

[12] C. Nilsson., Heuristics for the traveling salesman problem., *Tech. Report, Linköping University, Sweden* (2003),

[13] Arthur E. Carter , Cliff T. Ragsdale, A new approach to solving the multiple traveling salesperson problem using genetic algorithms, *European Journal of Operational Research 175* (2006), 246–257. *https://doi.org/10.1016/j.ejor.2005.04.027*

[14] Maria-Florina,B, Liang , Y., Gupta,P. , Robust Hierarchical Clustering, *Journal of Machine Learning Research* , Vol 15, (2014), pp. 4011-4051.

[15] S. Saitta, B. Raphael, I.F.C. Smith, A Bounded Index for Cluster Validity, *Machine Learning and Data Mining in Pattern Recognition*, Vol 4571 (2007), pp. 174-187.

Notes for Contributors

related to the papers to be submitted
for Production Systems and Information Engineering

Aims and scope

The aim of the journal is to publish high quality research papers connected with both production systems and information engineering at the same time. Special emphasis is given to articles on the theoretical models and methods, as well as practical applications of discrete production processes including new (or partially new) software tools. Using a new term proposed in special literature in the nineties, the main profile of this journal is Production Information Engineering.

Frequency of the journal

One volume a year (80–200 pages per volume) is planned.

Submission of manuscript

Submission of a manuscript implies that the paper has not been published, nor is being considered for publication elsewhere. All papers should be written in English. Two copies of the manuscript should be submitted on pages of A4 size.

Each manuscript should be provided with an English Abstract of about 50–100 words, reporting concisely on the objective and the results of the paper. The English Abstract is to be followed by the keywords.

References should be grouped at the end of the paper in numerical order of appearance. Author's name(s) and initials, paper titles, journal name, volume, issue, year and page numbers should be given for all journals referenced.

For the purpose of refereeing, papers should initially be submitted in hard-copy (twofold) to the secretaries. The eventual supply of an accepted-for-publication paper in its final camera-ready form will ensure more rapid publication. As regards the detailed format requirements, please consider the next homepage: http://ait.iit.uni-miskolc.hu/~psaie.

A Short History of the Publications of the University of Miskolc

The University of Miskolc (Hungary) is an important centre of research in Central Europe. Its parent university was founded by the Empress Maria Teresia in Selmecbánya (today Banska Stiavnica, Slovakia) in 1735. After the First World War the legal predecessor of the University of Miskolc moved to Sopron (Hungary) where, in 1929, it started the series of university publications with the title Publications of the Mining and Metallurgical Division of the Hungarian Academy of Mining and Forestry Engineering (Volumes I–VI). From 1934 to 1947 the Institution bad the name Faculty of Mining, Metallurgical and Forestry Engineering of the József Nádor University of Technology and Economics Sciences at Sopron. Accordingly, the publications were given the title Publications of the Mining and Metallurgical Engineering Division (Volumes VII–XVI). For the last volume before 1950 due to a further change in the name of the Institution Technical University, Faculties of Mining, Metallurgical and Forestry Engineering, Publications of the Mining and Metallurgical Divisions was the title. For some years after 1950 the Publications were temporarily suspended.

After the foundation of the Faculty of Mechanical Engineering in Miskolc in 1949 and the movement of the Sopron Mining and Metallurgical Faculties to Miskolc the Publications restarted with the general title Publications of the Technical University of Heavy industry in 1955. Four new series Series A (Mining), Series B (Metallurgy), Series C (Machinery) and Series D (Natural Sciences) were founded in 1976. These came out both in foreign languages (English, German and Russian) and in Hungarian.

After the foundation of the Faculty of Mechanical Engineering in Miskolc in 1949 and the movement of the Sopron Mining and Metallurgical Faculties to Miskolc the Publications restarted with the general title Publications of the Technical University of Heavy industry in 1955. Four new series Series A (Mining), Series B (Metallurgy), Series C (Machinery) and Series D (Natural Sciences) were founded in 1976. These came out both in foreign languages (English, German and Russian) and in Hungarian. In 1990, right after the foundation of some new faculties, the university was renamed the University of Miskolc. At the same time the structure of the Publications was reorganized so that it could follow the faculty structure. Accordingly, three new series were established: Series E (Legal Sciences), Series F (Economic Sciences), and Series G (Humanities and Social Sciences). The seven series are constituted by some periodicals and publications, which come out with various frequencies.

PRODUCTION SYSTEMS AND INFORMATION ENGINEERING

Volume 8 (2019)

CONTENTS