Contents

ANNALES MATHEMATICAE ET INFORMATICAE 48. (2018)

ANNALES MATHEMATICAE ET INFORMATICAE

TOMUS 48. (2018)

EGER 1774
ESZTERHÁZY KÁROLY UNIVERSITY

HUNGARIA, EGER

ANNALES MATHEMATICAE ET INFORMATICAE

International journal for mathematics and computer science

Referred by
Zentralblatt für Mathematik
and
Mathematical Reviews

The journal of the Institute of Mathematics and Informatics of Eszterházy Károly University of Applied Sciences is open for scientific publications in mathematics and computer science, where the field of number theory, group theory, constructive and computer aided geometry as well as theoretical and practical aspects of programming languages receive particular emphasis. Methodological papers are also welcome. Papers submitted to the journal should be written in English. Only new and unpublished material can be accepted.

Authors are kindly asked to write the final form of their manuscript in LaTeX. If you have any problems or questions, please write an e-mail to the managing editor Miklós Hoffmann: `hofi@ektf.hu`

The volumes are available at `http://ami.uni-eszterhazy.hu`

ANNALES MATHEMATICAE ET INFORMATICAE

VOLUME 48. (2018)

INSTITUTE OF MATHEMATICS AND INFORMATICS
ESZTERHÁZY KÁROLY UNIVERSITY
HUNGARY, EGER

Selected papers of the 10^{th} International Conference on Applied Informatics

HU ISSN 1787-5021 (Print)
HU ISSN 1787-6117 (Online)

A kiadásért felelős az
Eszterházy Károly Egyetem rektora
Megjelent a Líceum Kiadó gondozásában
Kiadóvezető: Nagy Andor
Felelős szerkesztő: Zimányi Árpád
Műszaki szerkesztő: Tómács Tibor
Megjelent: 2018. október Példányszám: 30

Készítette az
Eszterházy Károly Egyetem nyomdája
Felelős vezető: Kérészy László

Annales Mathematicae et Informaticae
***48** (2018) pp. 5–13*
http://ami.uni-eszterhazy.hu

Interactive Rendering Framework for Distance Function Representations

Csaba Bálint, Gábor Valasek

Eötvös Loránd University
csabix.balint@gmail.com
valasek@inf.elte.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Sphere tracing, introduced by Hart in [5], is an efficient method to find ray-surface intersections, provided the surface is represented by a signed distance function (SDF) or a lower estimate of it.

This paper presents an interactive rendering framework for visualising exact and estimate SDF representations. We demonstrate the performance of the system by visualising 3D fractals and its modularity by rendering algebraic and meta surfaces. In addition, we discuss SDF estimation of algebraic surfaces.

Keywords: Computer Graphics, Signed Distance Functions, Real-time Rendering

MSC: 65D18, 68U05

1. Introduction

Rendering surfaces represented by signed distance functions (SDF) has not been in the spotlight of computer graphics research. Even though fractals have been a focus of much interest on on-line forums, literature on rendering a more general representation of surfaces, namely direct visualisation of SDFs, is scarce; the latest advancement in the field is the contribution of Keinert et. al. in [6] (2014).

A general SDF rendering engine has a far greater flexibility than incremental image synthesis based systems; even ray-tracers of practice are limited to a fixed set of surface approximations. In an SDF based rendering engine, CSG[1]-models, 3D

fractals, algebraic surfaces, and meta-surfaces can all be rendered directly without any pre-processing. This means that the surfaces appear in a considerably higher quality than any pre-processed polygon approximation.

However, the main disadvantage of using SDFs is the lower rendering speed compared to incremental image synthesis based rendering engines. Additionally, traditional ray-tracers and game engines both use the same set of primitives (usually polygons) which does not include SDFs. This paper focuses on the representation and rendering of SDFs, with emphasis on the case of algebraic surfaces.

Previous work The algorithm for rendering SDFs known as sphere-tracing was first investigated by Hart in [5] (1994). It is an iterative ray-tracing algorithm, illustrated in Figure 1. This algorithm has been commonly used for the past two decades for rendering SDFs, most notably fractals [3, 4, 7, 9].

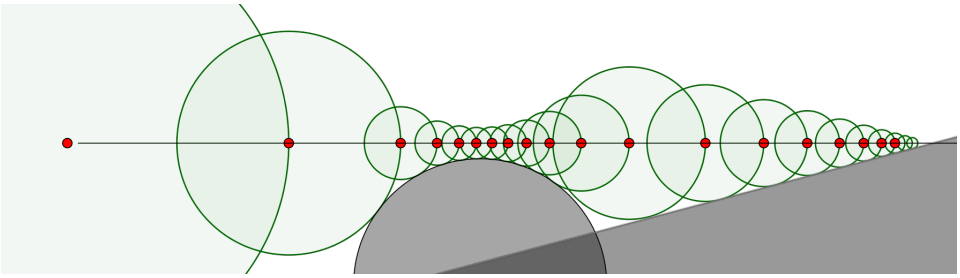


Figure 1: Illustration of the sphere-tracing algorithm in 2D.
At every point (red dot) along the ray, the distance to the surface is estimated, in this case, to the union of a half-plane and a circle. This distance defines a sphere (green circles) in which there are no intersections between the ray and the surface. Thus, sphere-tracing travels this distance along the ray to get the next estimate of the intersection point.

Following a short overview of singed distance functions, we introduce an algorithm for algebraic surface visualization. Approximating the surface normal is common problem, for witch a novel method is presented in Section 5.

2. Signed Distance Functions

In this section, definitions and notations are introduced for future reference. Definition 2.2 is from Hart's original work in [5].

Definition 2.1 (Distance to set)**.** Let (X, d) be a metric space, $x \in X$, and $A \subset X$. Then let $d(x, A) := \inf_{a \in A} d(x, a)$ (where $\inf \emptyset := +\infty$).

Definition 2.2 ((Signed) Distance Function)**.** The $f : \mathbb{R}^n \to \mathbb{R}$ function is an exact *(singed) distance function*, or (S)DF, if for any $\boldsymbol{p} \in \mathbb{R}^n$:

[1] CSG, Constructive Solid Geometry: A tree-like representation of the scene using primitive objects as leaves, set operations as nodes, and transformations as edges. [2]

$$f(\boldsymbol{p}) = d\left(\boldsymbol{p}, f^{-1}(0)\right) \quad \left(\text{or} \quad f(\boldsymbol{p}) = \begin{cases} d(\boldsymbol{p}, \text{bound}(D)) & \text{if} \quad \boldsymbol{p} \in D \\ -d(\boldsymbol{p}, \text{bound}(D)) & \text{if} \quad \boldsymbol{p} \notin D \end{cases} \right). \tag{2.1}$$

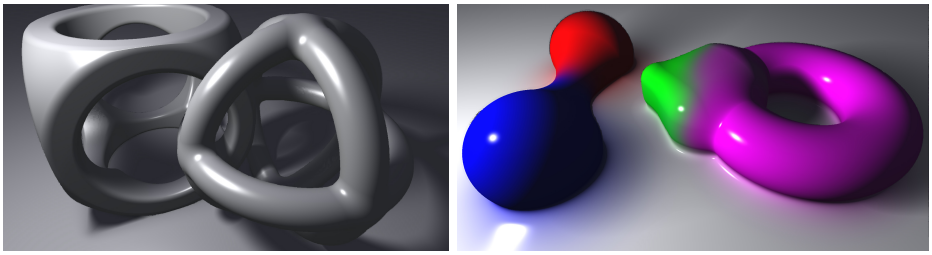
Where $\text{bound}(D) = \overline{D} \setminus \text{int}(D)$ denotes the boundary of a set.

Definition 2.3 (Distance Function Estimate)**.** The $f : \mathbb{R}^n \to \mathbb{R}$ function is a (signed) distance function estimation if and only if there exists a $q : \mathbb{R}^n \to [1, K)$ bounded ($K \in \mathbb{R}$) function, such that $f \cdot q$ is a (singed) distance function.

Remark 2.4. Besides the SDF being an upper bound to the estimate, Definition 2.3 provides a lower bound for the estimate, so sphere-tracing algorithms still converge.

The following theorem by Hart [5] describes how SDFs representing objects can be combined to create more complex geometries using CSG-like constructions. Figure 2a shows an application of the polynomial soft-min/max versions of set operations to various geometries.

Theorem 2.5 (Set operations)**.** *Let $f, g \in \mathbb{R}^n \to \mathbb{R}$ be (S)DF. Then* ***(i)*** $\{f \equiv 0\} \cup \{g \equiv 0\} = \{\min(f, g) \equiv 0\}$*,* ***(ii)*** $\{f \equiv 0\} \cap \{g \equiv 0\} = \{\max(f, g) \equiv 0\}$*,* ***(iii)*** $\{f \equiv 0\} \setminus \{g \equiv 0\} = \{\max(f, -g) \equiv 0\}$*. Additionally, the* $\min(f, g)$ *and* $\max(f, g)$ *are (singed) distance function estimates.*



(a) Soft-min/max using 3 tori, 3 cylinders, 2 spheres, 1 cube, and 1 plane (b) Meta-surface of 2 spheres, 1 cube, 1 torus, and 1 plane

Figure 2: Demonstration of the CSG model capabilities using our rendering engine

Moreover, by using different blending functions between primitive geometries one can achieve the look of different phenomena, like water [8]. Figure 2b shows meta surfaces rendered in our system.

3. Algebraic surface estimation

Let us now consider the problem of estimating SDFs to algebraic surfaces of the form $f(x, y, z) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} a_{ijk} x^i y^j z^k \quad (a_{ijk} \in \mathbb{R})$. To construct an SDF from this, we have to use the following theorem [5]

Theorem 3.1. *If $f \in \mathbb{R}^n \to \mathbb{R}$ is a (S)DF, it is Lipschitz continuous, and* $\operatorname{Lip} f = 1$.

Therefore, for any Lipschitz continuous f function $\frac{f}{\operatorname{Lip} f}$ is a signed distance function estimate. Although algebraic surfaces are not Lipschitz continuous over $\mathbb{R}^3$, they become Lipschitz over any finite bounded subset of space. In this case, if the distance from a given point to the surface is r, the estimation would be $f(\boldsymbol{p})/\operatorname{Lip} f|_{S_r(\boldsymbol{p})} = f(\boldsymbol{p})/\operatorname{Lip}_{S_r(\boldsymbol{p})} f$, where $S_r(\boldsymbol{p})$ is the sphere with centre $\boldsymbol{p}$ and radius $r > 0$. This provides the following fixpoint-iteration

$$F(r, \boldsymbol{p}) = \frac{f(\boldsymbol{p})}{\operatorname{Lip}_{S_r(\boldsymbol{p})} f} = r. \tag{3.1}$$

Iterating F on its first argument, r, results in an estimation of the distance function.

Usually, we have to calculate the distance in a certain direction, for example along a ray. Let $\boldsymbol{s}(t) := \boldsymbol{p} + t \cdot \boldsymbol{v}$. We must calculate the Lipschitz constant of the following on a given $S_r(\boldsymbol{p})$ set.

$$f(\boldsymbol{s}(t)) = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} a_{ijk} (p_x + tv_x)^i (p_y + tv_y)^j (p_z + tv_z)^k \tag{3.2}$$

The substitution method for calculating $\operatorname{Lip}_{[-r,r]} f \circ \boldsymbol{s}$ of (3.2) is treating this expression as a $f \circ \boldsymbol{s} \in \mathbb{R}[t, p_x, p_y, p_z, v_x, v_y, v_z]$ seven variable polynomial. Multiplying out, then ordering the terms, we get $N \leq n_i + n_j + n_k + 1$ number of monomials in t. Let $P_n(\boldsymbol{p}, \boldsymbol{v}) \cdot t^n$ denote the nth monomial.

Therefore, $\operatorname{Lip}_{t \in [-r,r]} (P_n(\boldsymbol{p}, \boldsymbol{v}) \cdot t^n) \leq n \cdot r^{n-1} |P_n(\boldsymbol{p}, \boldsymbol{v})|$ is the estimate of the Lipschitz constant of the nth monomial[2], where r is from (3.1), and the sum of these is the upper-estimate of the Lipschitz constant of f.

The problem with this approach is that in practice, we have to be able to make symbolic calculations within the engine and generate GPU code based on the algebraic surface given.

4. Taylor-series method

Our method is based on the fact that a Taylor expansion of a polynomial is itself. To calculate $P_n(\boldsymbol{p}, \boldsymbol{v})$ first we note that $P_n = \frac{1}{n!}(f \circ s)^{(n)}(0)$. Now, let us find an efficient way to compute the nth derivative of $f \circ \boldsymbol{s}$. Let

$$g_{ijk}(t) := (p_x + tv_x)^i (p_y + tv_y)^j (p_z + tv_z)^k \quad (t \in [-r, r]) , \tag{4.1}$$

so $P_n = \sum_{i=0}^{n_i} \sum_{j=0}^{n_j} \sum_{k=0}^{n_k} a_{ijk} \frac{g_{ijk}^{(n)}(0)}{n!}$. Let $h_{ijk}(t) := \frac{iv_x}{p_x + tv_x} + \frac{jv_y}{p_y + tv_y} + \frac{kv_z}{p_z + tv_z}$.

Note that $g'_{ijk} = g_{ijk} \cdot h_{ijk}$, so $g_{ijk}^{(n+1)} = \sum_{m=0}^{n} \binom{n}{m} g_{ijk}^{(m)} h_{ijk}^{(n-m)}$, where

[2] On estimating Lipschitz constants: [1]

$$h_{ijk}^{(n)}(t) = (-1)^n \cdot n! \left[i \left(\frac{p_x}{v_x} + t \right)^{-n-1} + j \left(\frac{p_y}{v_y} + t \right)^{-n-1} + k \left(\frac{p_z}{v_z} + t \right)^{-n-1} \right]. \quad (4.2)$$

Thus, $h_{ijk}^{(n)}$, $g_{ijk}^{(n)}$, and P_n can all be computed, and so is the following approximation:

$$\underset{[-r,r]}{\text{Lip}}\,(f \circ \boldsymbol{s}) = \underset{t\in[-r,r]}{\text{Lip}} \left(\sum_{n=0}^{n_i+n_j+n_k} P_n \cdot t^n \right) \leq \sum_{n=1}^{N} n \cdot r^{n-1} |P_n| \,. \quad (4.3)$$

Finally, repeating the (3.1) iteration gives us the distance estimate.

Algorithm 1: Calculating P_n

In : $\boldsymbol{p}, \boldsymbol{v}$ and f in sparce-form: $f(x,y,z) = \sum_{l=0}^{L} A_l \cdot x^{I_l} y^{J_l} z^{K_l}$ where $A \in \mathbb{R}^L$; $I, J, K \in \mathbb{N}^L$

Out : $P \in \mathbb{R}^N$ is for P_n coefficients.

Temp : $G, H \in \mathbb{R}^N$ for g_{ijk} and h_{ijk}.

$P := (f(\boldsymbol{p}), 0, 0, \ldots, 0)$;
for $l = 0 \,..\, L-1$ **do**
  $G := \left(p_x^{I_l} \cdot p_y^{J_l} \cdot p_z^{K_l}, 0, 0, \ldots, 0 \right)$;
  for $n = 1 \,..\, N-1$ **do**
    $H_{n-1} := -(-1)^n (n-1)! \cdot \left(I_l \frac{v_x^n}{p_x^n} + J_l \frac{v_y^n}{p_y^n} + K_l \frac{v_z^n}{p_z^n} \right)$;
    for $m = 0 \,..\, n-1$ **do**
      $G_n := G_n + \binom{n-1}{m} \cdot G_m H_{n-m-1}$;
    $P_n := P_n + \frac{1}{n!} A_l \cdot G_n$;

Algorithm 2: Fix-point iteration

In : The ray $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^3$, and $P \in \mathbb{R}^N$ coefficient vector from Algorithm 1 is given. For better convergence, a $\lambda \in (0, 1]$ relaxation constant, and $r_0 > 0$ starting radius, eg. from linear approximation, is also given.

Out : $r > 0$ distance that can be travelled along the $\boldsymbol{s}(t) = \boldsymbol{p} + t \cdot \boldsymbol{v}$ $(t > 0)$ ray.

Temp : The Lipschitz constants will be calculated in $Lip > 0$ variable.

$r := r_0$;
for $i = 0 \,..\, iters$ **do**
  $Lip := 0$;
  for $n = 1 \,..\, N$ **do**
    $Lip := Lip + n \cdot r^{n-1} \,|P_n|$;
  $r := r \cdot (1 - \lambda) + \frac{f(\boldsymbol{p})}{Lip} \cdot \lambda$;

Figure 3: Novel algorithms for algebraic surface visualization.
First, using equations (4.1)–(4.3), the P_n coefficients of the Taylor expansion of $f \circ \boldsymbol{s}$ are calculated. Second, the fix-point iteration in (3.1) is used to find the right step size for the sphere-tracing algorithm.

Implementing this approach is easier as it does not require symbolic expressions and complex code generation, see the algorithms on Figure 3. Figure 4 summarises our results. The algorithm can be stopped at any derivative thus achieving quadratic complexity in the number of derivatives and linear in the number of terms.

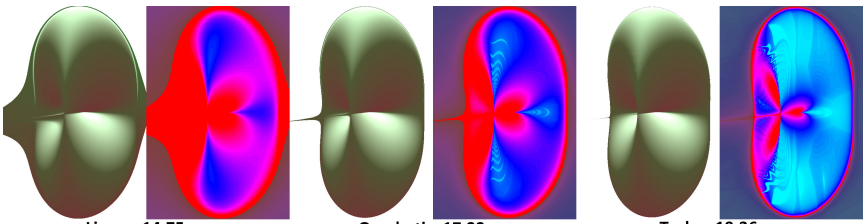


Figure 4: Comparison of SDF estimations with capped amount of steps along a ray.
The algebraic surface $f_{K_1,K_2}(x,y,z) = (x^2+y^2+z^2)(K_1x^2+K_2y^2) - 2z(x^2+y^2) = 0$ has a singular line that makes it hard to visualize from this angle. The Taylor method converges closer to the surface in less steps in about the same time as the traditional linear and quadratic SDF approximations. The light-blue means it only takes one step, and in the red region it takes 70.

5. Normal estimation

The surface normal at $\boldsymbol{p} \in \{f \equiv 0\}$ is defined as the unit vector $\mathbf{norm}(\boldsymbol{p}) = \frac{\nabla f(\boldsymbol{p})}{\|\nabla f(\boldsymbol{p})\|_2}$, for any surface defined by the differentiable implicit function $f \in \mathbb{R}^3 \to \mathbb{R}$.

In this section, we focus on calculating the normal numerically. The one-sided (forward or backward) difference method gives an error of $\mathcal{O}(\epsilon)$ for the derivative. A more accurate method is the symmetric difference:

$$\nabla f(x,y,z) = \frac{1}{2\epsilon} \cdot \begin{bmatrix} f(x+\epsilon,y,z) - f(x-\epsilon,y,z) \\ f(x,y+\epsilon,z) - f(x,y-\epsilon,z) \\ f(x,y,z+\epsilon) - f(x,y,z-\epsilon) \end{bmatrix} + \mathcal{O}\left(\epsilon^2\right) . \tag{5.1}$$

The idea of our approach is to take the following vectors (or stencil)

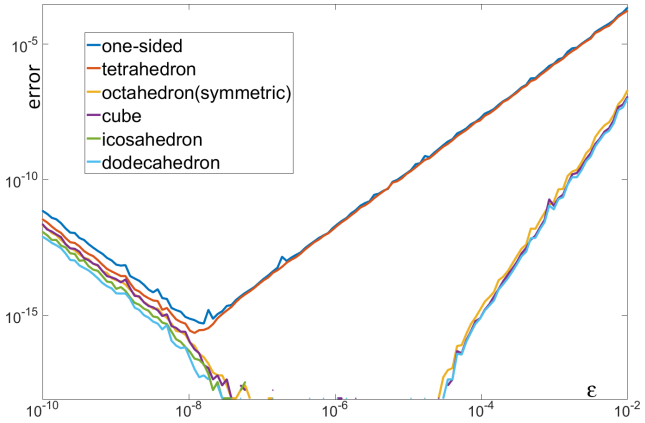
$$\boldsymbol{v}_1 := [+1,\ 0,\ 0]^\top,\ \boldsymbol{v}_3 := [0,+1,\ 0]^\top,\ \boldsymbol{v}_5 := [0,\ 0,+1]^\top,$$

$$\boldsymbol{v}_2 := [-1,\ 0,\ 0]^\top,\ \boldsymbol{v}_4 := [0,-1,\ 0]^\top,\ \boldsymbol{v}_6 := [0,\ 0,-1]^\top,$$

then (5.1) is equivalent to $\nabla f(\boldsymbol{p}) = \frac{1}{2\epsilon} \sum_{i=1}^{6} f(\boldsymbol{p} + \epsilon \cdot \boldsymbol{v}_i) \cdot \boldsymbol{v}_i$, thus we define

$$\mathbf{norm}(\boldsymbol{p}) = \frac{1}{Z} \sum_{i=1}^{6} f(\boldsymbol{p} + \epsilon \cdot \boldsymbol{v}_i) \cdot \boldsymbol{v}_i \ . \tag{5.2}$$

where $Z \in \mathbb{R}$ is the normalising constant. This means that the samples of the function are taken at the vertices of an octahedron.



(a) Error in relation to ϵ. Line breaks when the angle between the analytic and the numeric estimation is zero.

	mean	median
one-sided	119	562
tetrah.	113	843
octah.	*1.0*	*1.0*
cube	0.6	0.7
icosah.	0.5	0.1
dodecah.	0.5	0.1

	std	max
one-sided	382	306
tetrah.	265	313
octah.	*1.0*	*1.0*
cube	0.5	0.6
icosah.	0.4	0.3
dodecah.	0.4	0.3

(b) Relative error to symmetric difference for $\epsilon = 0.01$

Figure 5: Error of normal estimators measured in cosine distance[3].
Our tetrahedron kernel performs slightly better than the one-sided approach and results in a marginally lower mean error with lower variance. Cube, icosahedron and dodecahedron kernels also slightly outperformed the symmetric difference, but they also take considerably more samples.

According to our measurements, an optimal stencil vector set would consist of equal length vectors that fill the space evenly, so the best kernels in these cases

[3] $\text{cosine\_distance}(a,b) = 1 - \text{cosine\_similarity}(a,b) = 1 - \cos(\theta) = 1 - \frac{a \cdot b}{\|a\|_2 \cdot \|b\|_2}$

consisted of vertices of platonic solids. Taking every second vertex of a cube gives us the fastest kernel, the tetrahedron:

$$\boldsymbol{v}_1 := [+1, +1, +1]^\top, \boldsymbol{v}_2 := [+1, -1, -1]^\top, \boldsymbol{v}_3 := [-1, +1, -1]^\top, \boldsymbol{v}_4 := [-1, -1, +1]^\top .$$

This is as fast as the first-order divided difference, but it gave empirically better results as shown on Figure 5. Other platonic solids were also investigated.

Potentially, even higher accuracy can be achieved by sampling surface of the unit sphere with a sequence of low discrepancy, like a Halton sequence. However, this is usually not needed, because the length of the SDF estimate's gradient is usually close to one. Moreover, the normal is needed for calculating lighting effects, and small errors are not visible.

The implementation supports multiple normal calculation algorithms. The tetrahedron kernel proved to be faster than the first-order divided difference one. Symmetric or octahedron kernels introduced barely visible differences in quality along hard edges.

6. Implementation

Figure 6: Mandelbulb fractal displayed with our rendering engine.
Maximum quality was reached after 156.2ms render time. Shadows were at maximum quality after 436.2ms. GPU utilisation was 97-99%. GPU: NVidia 640M (480 GFLOPS).

The rendering engine supports operating in a progressive mode, which means when the camera is not moving, the image quality continues to increase. Therefore, the engine is optimised for static scenes. The C++ and OpenGL implementation is highly efficient achieving near 100% GPU utilisation and provides several features.

Firstly, swapping algorithms between passes was a free operation due to the OpenGL subroutines running on the GPU. This and the algorithms inter-compa-

tibility can be used for a short statistical training to determine the best schedule of algorithms for a given scene.

Secondly, a CSG model creator was also implemented. The user can either write the program computing the SDF directly or build the CSG tree from primitives and other program codes both using a built-in graphical interface.

Finally, the shader programs, including subroutines, were generated on-the-fly, thus the code for the scene geometry is embedded into the code running on the GPU. This greatly reduced both the distance function evaluation times and memory consumption.

7. Summary

This paper presented a direct signed distance function visualisation framework and its application to rendering algebraic surfaces.

We proposed a local signed distance function estimation method to such surfaces and investigated the precision of various surface normal estimation heuristics. We benchmarked the performance of the system by rendering complex scenes incorporating CSG elements, meta-surfaces, and the Mandelbulb fractal.

The framework proved to be highly efficient. In addition, it is highly modular, and outperformed current fractal-viewers [3, 4, 7, 9] in both quality and speed.

References

[1] Kenneth Eriksson, Donald Estep, and Claes Johnson. *Applied Mathematics: Body and Soul*. Number 978-3-540-00890-3. Springer-Verlag Berlin Heidelberg, 1 edition, 2004. Volume 1: Derivatives and Geometry in IR3.

[2] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design (3rd Ed.): A Practical Guide*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[3] Paul Geisler, Daniel White, Paul Nylander, Tom Lowe, David Makin, Buddhi, Joy Leys, Knighty, and Jan Kadlec. Online fractal viewer: FractalLab. http://hirnsohle.de/test/fractalLab/, 2010.

[4] Matthew Haggett. Mandelbulb 3D (MB3D) fractal rendering software. `http://mandelbulb.com/2014/mandelbulb-3d-mb3d-fractal-rendering-software/`, 2014.

[5] John C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1994.

[6] Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. Enhanced Sphere Tracing. In Andrea Giachetti, editor, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2014.

[7] Krzysztof Marczak. Mandelbuilder - 3D fractal explorer. `http://www.mandelbulber.com/`, 2010.

[8] László Szécsi and Dávid Illés. Real-time metaball ray casting with fragment lists. In Carlos Andújar and Enrico Puppo, editors, *Eurographics (Short Papers)*, pages 93–96. Eurographics Association, 2012.

[9] Íñigo Quílez. Mandelbulb.
`http://www.iquilezles.org/www/articles/mandelbulb/mandelbulb.htm`,
2009. (Mandelbulb in real time).

Annales Mathematicae et Informaticae
***48** (2018) pp. 15–22*
http://ami.uni-eszterhazy.hu

Scale-free property of the weights in a random graph model*

István Fazekas, Attila Perecsényi

Faculty of Informatics, University of Debrecen
fazekas.istvan@inf.unideb.hu
perecsenyi.attila@inf.unideb.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

A new modification of the N interaction model [5], which based on the 3-interactions model of Backhausz-Móri [1]. This is a growing model, what evolves by weights. In every step N verticies will interact by form a star graph. We can choose vertices uniformly or according to their weights (preferential attachment). Our aim is to show asymptotic power-law distributions of the weights. The proofs are based on discrete time martingale methods. Numerical result is also presented.

Keywords: random graph, network, scale-free, power-law

MSC: 05C80, 60G42.

1. Introduction

Barabási and Albert [2] gave an explanation for the frequently observed phenomenon that many real-life networks are scale free, i.e., they have power-law degree distribution. To describe real-life networks such as the WWW, social and biological networks, they introduced a random graph model. They defined an evolving graph using the preferential attachment rule, what leads to scale-free graphs. Preferential attachment rule in a random graph model means, that when a new vertex is born, then the probability that the new vertex will be connected to an old vertex is proportional to the degree of the old vertex.

*Attila Perecsényi was supported through the New National Excellence Program of the Ministry of Human Capacities

In [4] a new network evolution model was introduced. In this paper, we shall study the same model. Consider an increasing sequence of weighted undirected graphs. The evolution of the graphs is based on creations of N-star subgraphs. Throughout the paper we call a graph N-star graph if N vertices form a star, that is it has one central vertex, what is connected with $N-1$ peripheral vertices. We start at time 0, and the initial graph is an N-star graph. This graph and all of its $(N-1)$-star subgraphs and all vertices have initial weights 1. Now we increase the size of the graph as follows. At each step N vertices interact with each other. It means that, we draw all non-existing edges between the peripheral vertices and the center vertex, so that, the vertices will form an N-star graph and the weights are increased by 1. The non-existing elements of the graph have weight 0.

We have two options in every step. On the one hand, with probability p, we add a new vertex, and it interacts with $N-1$ old vertices. On the other hand, with probability $1-p$, we do not add any new vertex, but N old vertices interact. Here $0 < p \leq 1$ is fixed.

When a new vertex is born, we have two possibilities again. With probability r, we choose an $(N-1)$-star graph according to to their weights (i.e. preferential attachment), and the new vertex is connected to its central vertex. Here preferential attachment means that the probability that we choose an $(N-1)$-star subgraph is proportional to its weight. With probability $1-r$, we choose $N-1$ old vertices uniformly at random and they will form an N-star graph with the new vertex, so that, the new vertex will be the center. Here uniform choice means that all subsets of vertices with cardinality $N-1$, have the same chance. Here $0 \leq r \leq 1$ is fixed.

In the other case, when we do not add any new vertex, we have two opportunities again. On the one hand, with probability q, we choose an old N-star graph according to their weights (i.e. preferential attachment). That is the chance of an N-star subgraph is proportional to its weight. Then we increase the weights inside the N-star subgraph chosen. On the other hand, with probability $1-q$, we choose uniformly N old vertices, and they form an N-star graph, so that, we choose the center out of the chosen N vertices uniformly. Here $0 \leq q \leq 1$ is fixed.

In [4] power law distribution of the weights of the vertices was shown. In this paper Theorem 2.1 shows that the weights of the N-stars have power law distribution. In the proof we use the Doob-Meyer decomposition and the method of [3].

2. Power law distribution of the weights of N-stars

Let $S(n,w)$ denote the number of N-stars with weight w, and let S_n denote the number of all N-stars after n steps. Furthermore, V_n denotes the number of vertices after n steps.

Theorem 2.1. *Let $0 < p < 1$ and $0 < q$. For all $w = 1, 2, \ldots$ we have*

$$\frac{S(n,w)}{S_n} \to s_w \tag{2.1}$$

almost surely as $n \to \infty$, *where* s_w, $w = 1, 2, \dots$ *are positive numbers satisfying the recurrence relation*

$$s_1 = \frac{1}{h+1}, \quad s_w = \frac{h(w-1)}{hw+1} s_{w-1}, \quad \text{if } w > 1, \tag{2.2}$$

where $h = (1-p)q$. *Moreover,*

$$s_w \sim C w^{-\left(1+\frac{1}{h}\right)} \tag{2.3}$$

as $w \to \infty$, *with* $C = \frac{1}{h}\Gamma\left(1+\frac{1}{h}\right)$.

Proof. First we show that

$$\frac{S(n,w)}{n} \to k_w \tag{2.4}$$

almost surely as $n \to \infty$ for any fixed w. Here k_w, $w = 1, 2, \dots$ are fixed nonnegative numbers.

We compute the conditional expectation of $S(n,w)$ with respect to $\mathcal{F}_{n-1}$ for $w \geq 1$. Let $S(n,0) = 0$ for all n. For $n, w \geq 1$ we have

$$\mathbb{E}\left(S(n,w)|\mathcal{F}_{n-1}\right) = p(n,w-1)S(n-1,w-1) + \left(1-p(n,w)\right)S(n-1,w) +$$

$$+\delta_{1,w}\left[p + (1-p)(1-q)\left(1 - \frac{S_{n-1}}{\binom{V_{n-1}}{N}N}\right)\right], \tag{2.5}$$

where

$$p(n,w) = (1-p)\left[q\frac{w}{n} + (1-q)\frac{1}{\binom{V_{n-1}}{N}N}\right]. \tag{2.6}$$

Let

$$c(n,w) = \prod_{i=1}^{n}\left(1-p(n,w)\right)^{-1}, \quad w \geq 1. \tag{2.7}$$

It is easy to see that the above random variable is $\mathcal{F}_{n-1}$ measurable. Applying the Marcinkiewicz strong law of large numbers for the number of vertices, we have

$$V_n = pn + \mathrm{o}\left(n^{1/2+\varepsilon}\right) \tag{2.8}$$

almost surely, for any $\varepsilon > 0$.

Using (2.8) and the Taylor expansion for $\log(1+x)$ we obtain

$$\log c(n,w) = -\sum_{i=1}^{n}\log\left(1 - h\frac{w}{i} - \frac{(1-p)(1-q)}{\binom{V_{i-1}}{N}N}\right) = hw\sum_{i=1}^{n}\frac{1}{i} + \mathrm{O}(1),$$

where the error term is convergent as $n \to \infty$. It means

$$c(n,w) \sim h_w n^{hw} \tag{2.9}$$

almost surely as $n \to \infty$ and h_w is a positive random variable.

Let us consider the following process:

$$Z(n,w) = c(n,w)S(n,w) \quad \text{for } w \geq 1.$$

Here $\{Z(n,w), \mathcal{F}_n, n = 1,2,\dots\}$ is a nonnegative submartingale for any fixed $w \geq 1$. By the Doob-Meyer decomposition of $Z(n,w)$, we can write

$$Z(n,w) = M(n,w) + A(n,w)$$

where $M(n,w)$ is a martingale and $A(n,w)$ is a predictable increasing process. The general form of $A(n,w)$ is the following:

$$A(n,w) = \mathbb{E}Z(1,w) + \sum_{i=2}^{n} \left[\mathbb{E}\left(Z(i,w)|\mathcal{F}_{i-1}\right) - Z(i-1,w)\right]. \tag{2.10}$$

Now from (2.5) and (2.10), we have

$$A(n,w) = \mathbb{E}Z(1,w) + \sum_{i=2}^{n} c(i,w)\Bigg[p(i,w-1)S(i-1,w-1) + \\ + \delta_{1,w}\left(p + (1-p)(1-q)\left(1 - \frac{S_{i-1}}{\binom{V_{i-1}}{N}N}\right)\right)\Bigg]. \tag{2.11}$$

Let $B(n,w)$ be the sum of the conditional variances of $Z(n,w)$. In the following we give an upper bound for $B(n,w)$:

$$B(n,w) = \sum_{i=2}^{n} \mathbb{D}^2\left(Z(i,w)|\mathcal{F}_{i-1}\right) = \sum_{i=2}^{n} \mathbb{E}\{\left(Z(i,w) - \mathbb{E}\left(Z(i,w)|\mathcal{F}_{i-1}\right)\right)^2 |\mathcal{F}_{i-1}\} = \\ = \sum_{i=2}^{n} c(i,w)^2 \mathbb{E}\{\left(S(i,w) - \mathbb{E}\left(S(i,w)|\mathcal{F}_{i-1}\right)\right)^2 |\mathcal{F}_{i-1}\} \leq \\ \leq \sum_{i=2}^{n} c(i,w)^2 \mathbb{E}\{\left(S(i,w) - S(i-1,w)\right)^2 |\mathcal{F}_{i-1}\} \leq \\ \leq \sum_{i=2}^{n} c(i,w)^2 = \mathrm{O}\left(n^{2hw+1}\right). \tag{2.12}$$

Above we used that $c(n,w)$ is $\mathcal{F}_{i-1}$ measurable, (2.5) and the fact that, at each step only one N-star is involved in interaction.

We use induction on w. Let us consider the case when $w = 1$. From (2.9) and (2.11), we have

$$A(n,1) = \mathbb{E}Z(1,1) + \sum_{i=2}^{n} c(i,1)\left[p + (1-p)(1-q)\left(1 - \frac{S_{i-1}}{\binom{V_{i-1}}{N}N}\right)\right] \sim$$

$$\sim \sum_{i=2}^{n} h_1 n^h \left[p + (1-p)(1-q)\left(1 - \frac{S_{i-1}}{i^N}\right)\right] \sim h_1 \frac{n^{h+1}(1-h)}{h+1} \tag{2.13}$$

as $n \to \infty$. Using (2.12), we have

$$B(n,1) = \mathrm{O}\left(n^{2h+1}\right),$$

so

$$B(n,1)^{\frac{1}{2}} \log B(n,1) = \mathrm{O}\left(A(n,1)\right).$$

The conditions of Proposition VII-2-4 of [6] is fulfilled, so we have

$$Z(n,1) \sim A(n,1) \tag{2.14}$$

almost surely on the event $\{A(n,1) \to \infty\}$ as $n \to \infty$. So from (2.9), (2.13) and (2.14), we obtain

$$\frac{S(n,1)}{n} = \frac{Z(n,1)}{c(n,1)n} \sim \frac{A(n,1)}{c(n,1)n} \sim \frac{h_1 n^{h+1}(1-h)}{h_1 n^h n} = \frac{1-h}{1+h} = k_1 > 0, \tag{2.15}$$

as $n \to \infty$.

Let $w > 1$. Suppose that (2.4) is true for all weight less than w. Now from (2.8), (2.9) and (2.11), using the induction hypothesis, we obtain

$$A(n,w) = \mathbb{E}Z(1,w) + \sum_{i=2}^{n} \left(c(i,w)p(i,w-1)S(i-1,w-1)\right) \sim$$

$$\sim \sum_{i=2}^{n} h_w i^{hw} k_{w-1} i \left[h\frac{w-1}{i} + \frac{(1-p)(1-q)}{i^N}\right] \sim k_{w-1} h_w h(w-1)\frac{n^{wh+1}}{wh+1} \tag{2.16}$$

almost surely as $n \to \infty$. We see that the conditions of Proposition VII-2-4 are true, so we have $Z(n,w) \sim A(n,w)$. Therefore, from (2.9) and (2.16), we have

$$\frac{S(n,w)}{n} = \frac{Z(n,w)}{c(n,w)n} \sim \frac{A(n,w)}{c(n,w)n} \sim \frac{k_{w-1} h_w h(w-1)\frac{n^{wh+1}}{wh+1}}{h_w n^{wh} n} =$$

$$= k_{w-1}\frac{h(w-1)}{wh+1} = k_w. \tag{2.17}$$

Now we show that

$$\frac{S_n}{n} \to B, \tag{2.18}$$

almost surely as $n \to \infty$ where $B = 1 - h$.

First we compute the conditional expectation of S_n with respect to $\mathcal{F}_{n-1}$. We can see that the number of N-stars increases if and only if the number of N-stars of weight 1 increases, so we have

$$\mathbb{E}\{S_n|\mathcal{F}_{n-1}\} = S_{n-1} + p + (1-p)(1-q)\left(1 - \frac{S_{n-1}}{\binom{V_{n-1}}{N} N}\right) = \gamma_{n-1} S_{n-1} + B, \tag{2.19}$$

where

$$\gamma_{n-1} = 1 - (1-p)(1-q)\frac{1}{\binom{V_{n-1}}{N} N}.$$

Let

$$G_n = \prod_{i=1}^{n-1} (\gamma_i)^{-1}, \quad n \geq 1. \tag{2.20}$$

Here G_n is an $\mathcal{F}_{n-1}$ measurable random variable. Furthermore, let

$$Z_n = G_n S_n \quad \text{for } 1 \leq n. \tag{2.21}$$

From (2.19), we obtain

$$\mathbb{E}\{Z_n | \mathcal{F}_{n-1}\} = Z_{n-1} + BG_n. \tag{2.22}$$

We can see that $\{Z_n, \mathcal{F}_n, n = 1, 2, \dots\}$ is a nonnegative submartingale. Applying again the Doob-Meyer decomposition for Z_n, we have

$$Z_n = M_n + A_n,$$

where M_n is a martingale and A_n is a predictable increasing process. From (2.10) and (2.22), we obtain

$$A_n = \mathbb{E}Z_1 + B\sum_{i=2}^{n} G_i. \tag{2.23}$$

By (2.8) and applying the Taylor expansion for $\log(1+x)$, we can give lower and upper bounds for G_i, so we obtain

$$C_1 n < A_n < C_2 n, \tag{2.24}$$

where C_1 and C_2 appropriate positive constants. Let B_n be the sum of the conditional variances of Z_n. In the following we give an upper bound for B_n:

$$B_n = \sum_{i=2}^{n} \mathbb{D}^2(Z_i|\mathcal{F}_{i-1}) = \sum_{i=2}^{n} \mathbb{E}\{(Z_i - \mathbb{E}(Z_i|\mathcal{F}_{i-1}))^2 | \mathcal{F}_{i-1}\} =$$

$$= \sum_{i=2}^{n} G_i^2 \mathbb{E}\{(S_i - \mathbb{E}(S_i|\mathcal{F}_{i-1}))^2 | \mathcal{F}_{i-1}\} \leq \sum_{i=2}^{n} G_i^2 \mathbb{E}\{(S_i - S_{i-1})^2 | \mathcal{F}_{i-1}\} \leq$$

$$\leq \sum_{i=2}^{n} G_i^2 \leq C_3 n, \tag{2.25}$$

where C_3 is a positive constant. Above we used that G_i is $\mathcal{F}_{i-1}$ measurable and the fact that, at each step, at most one N-star can be born. Using (2.25), we have $B_n^{1/2} \log B_n = \mathrm{O}(A_n)$. From (2.24), we can see that $A_n \to \infty$ as $n \to \infty$, so applying Proposition VII-2-4 of [6], we obtain

$$Z_n \sim A_n \tag{2.26}$$

almost surely as $n \to \infty$.

Using (2.26) and (2.23), we have

$$\frac{K_n}{n} = \frac{Z_n}{G_n n} \sim \frac{A_n}{G_n n} = \frac{\mathbb{E}Z_1}{G_n n} + B\frac{1}{G_n}\frac{1}{n}\sum_{i=2}^{n} G_i \to B \tag{2.27}$$

almost surely.

Finally, from (2.4) and (2.18), we obtain

$$\frac{S(n,w)}{S_n} = \frac{S(n,w)}{n}\frac{n}{S_n} \to \frac{k_w}{B} = s_w \tag{2.28}$$

almost surely as $n \to \infty$. By using (2.28) for (2.15) and (2.17), we have the recurrence of s_w (cf. (2.2)). Applying several times (2.2), we obtain

$$s_w = s_1 \prod_{i=2}^{w} \frac{h(i-1)}{hi+1} = \frac{1}{h}\frac{(w-1)!}{\prod_{j=1}^{w}\left(j+\frac{1}{h}\right)} = \frac{1}{h}\frac{\Gamma(w)\Gamma\left(1+\frac{1}{h}\right)}{\Gamma\left(w+1+\frac{1}{h}\right)}. \tag{2.29}$$

Since $\sum_{w=1}^{\infty} s_w = 1$, the sequence $s_1, s_2, \ldots$ is a proper discrete probability distribution.

Now applying Stirling's formula for (2.29), we obtain the power law distribution (2.3). □

3. Numerical result

In this section we present a numerical result. The 4-star model was generated with parameters $p = 0.5$, $q = 0.5$ and $r = 0.5$. We simulated $n = 10^5$ steps. To visualize the power law distribution we used log-log scale. Figure 1 shows that the weight distribution of 4-stars is indeed power law distribution.

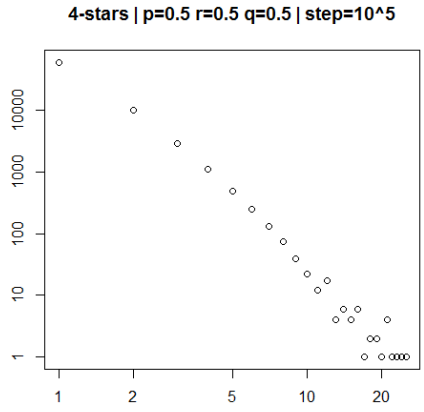


Figure 1: The weight distribution of 4-stars

References

[1] Backhausz Á., Móri T. F., Weights and degrees in a random graph model based on 3-interactions, *Acta Math. Hungar.*, Vol. 143(1) (2014), 23–43.

[2] Barabási A. L., Albert R., Emergence of scaling in random networks, *Science*, Vol. 286 (1999), 509–512.

[3] I. Fazekas, Cs. Noszály, A. Perecsényi, Weights of Cliques in a Random Graph Model Based on Three-Interactions, *Lithuanian Mathematical Journal*, 55 (2015), 207-221.

[4] I. Fazekas, Cs. Noszály, A. Perecsényi, The N-stars network evolution model, *in preparation.*

[5] Fazekas, I., Porvázsnyik, B., Scale-free property for degrees and weights in an N-interactions random graph model, *J. Math. Sci.* Vol. 214 (2016), 69–89.

[6] Neveu, J. Discrete-parameter martingales, *North-Holland, Amsterdam*, 1975.

Annales Mathematicae et Informaticae
***48** (2018) pp. 23–32*
http://ami.uni-eszterhazy.hu

A web-based programming environment for introductory programming courses in higher education

Győző Horváth

Eötvös Loránd University, Faculty of Informatics
gyozo.horvath@inf.elte.com

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Choosing the right programming environment has a great influence on the efficiency of the educational, learning and problem solving processes. While there are many good examples for such environments for the younger generation, which involve block-based programming, gamified learning, appropriate language of the tasks and user interface design, introductory programming courses in higher education rarely take into account the role of the programming environment. In this article we have analyzed a typical problem solving process in an introductory programming course with a special focus on the programming environment. We have found that many distracting factors may make the learning process difficult.

Based on our investigation we introduce a web-based programming environment which takes into account the special needs of newcomers to the programming land. This environment tries to exclude the distracting factors and support the problem solving process in a right way. Beside our methodological considerations, the technical background of supporting traditional programming languages, such as C++, in the web browser is also presented. Finally we make methodological recommendations how this tool can be a part of the teaching and learning process through different types of tasks and learning organizing methods.

Keywords: web, teaching, programming, development environment, higher education

MSC: 97Q60, 97B40, 97U70

1. Introduction

Choosing the right programming environment has a great influence on the efficiency of the educational, learning and problem solving processes. There are many good examples for such environments mainly for the younger generation, taking into account the specific needs of users of these ages. The user interface design is appropriate: nice and colourful for the youngest ones, or comes in different thematic flavours from toys, computer games or films (e.g. [1]) for teenagers. The task descriptions are usually simple and straightforward according to age of the students. The chosen programming language is mainly block-based (like [2]) for the introductory lessons, because it hides the syntactic difficulties behind the blocks, and beginner students only have to deal with the semantic meanings of the blocks, and how to place them one after another or inside to achieve the task. And finally, task solving is often wrapped in a gamified clothes, making the learning process fun and challenging (e.g. [3]).

Introductory programming courses in higher education, however, often considers novice programmer students, as if they have a lot of experience in handling complex processes, but usually this is not the case, no matter how much this would be expected. Treating them as "mature" programmers involves: using code-based programming language from the very beginning, giving them mathematical problems to solve, and using professional or professional-like integrated development environments (IDE), while students need to cope with the more essential mental model of programming. It is hard for the students without any former programming experience to take such big steps in many areas of the programming field. Curriculum should pay attention to gradually introduce newer and newer topics, in order to evenly distribute the cognitive load and thus make the knowledge processing much more effective by students.

Programming, however, is not just coding, it is part of a more general task, problem solving. Problem solving begins with the interpretation of the *task description*, continues with abstracting out and describing the data and their relations contained therein (*specification*), then it provides a solution as a sequence of elementary steps in an abstract language (pseudo-code, *algorithm*), which is finally implemented in the given or chosen programming language (*coding*). Problem solving, however, does not end with this latter step even in a narrower scope, as we have to make sure that the program works correctly by *testing* it, and the detected errors need to be *correct*ed. For smaller programs and tasks, the problem solving may end here. Introductory courses require such programming environments that support both the basic steps and skills of problem solving and the coding phase at the same time.

In this article first we analyse a typical problem solving process in an introductory programming course with a special focus on the programming environment. After this and based on our investigation, we look for better alternatives than using the traditional IDEs, and propose a programming environment which tries to meet the required expectations.

2. Analysing traditional programming environments

In this chapter a typical problem solving process will be analysed, assuming that it takes place in an introductory programming lesson in higher education with students with different preconceptions about programming and problems solving. In order to serve this kind of heterogeneity, the course needs to introduce every concept from the basics to build a systematic knowledge common for everyone. Accordingly, the tasks are relatively small, even at the end of the course there are no tasks that need to be disassembled into several files.

Let us review the problem solving process from the aspect of the tools and development environments. The first step is to get to know the task. This can be done verbally, written on a board, or projected to the wall. The *task description* can be paper-based or digital. Digital material may be published on a general website or in a dedicated task library. Its format can be any of the well-known document formats (HTML, PDF, docx, etc.). Typically, the task description can be accessed in a different software environment from the one where implementation would take place.

The next two steps, *specification and algorithm*, are designed for planning. Planning is traditionally done on paper or on a board. As a new phenomenon, however, it is increasingly common for students to write notes on their digital devices and, on the one hand, they do not have an exercise book or pen, and on the other hand, performing the above two design steps with traditional editors (e.g. text or image editors) is a much larger task than doing it manually. Thus, students are prone to skip this planning step more and more frequently. However, this article does not want to address this issue. The message of this part of problem solving is that planning considerations are implemented in a different environment or tool than coding.

The spectacular and creative part of problem solving is the *implementation phase*, when the plan (pseudo-code) turns into code. This step often occurs in a development environment. These specific environments are chosen because they contain all the whistles and bells needed for convenient development in the chosen programming language, as opposed to a generic code editor, where setting up a basic programming session is a time-consuming task and often the user interface does not support simple usage scenarios.

Integrated Development Environments (IDEs) are prepared with tools for editing, compiling, and running certain types of programs (such as CodeBlocks, Visual Studio, Netbeans). They are convenient to use and the default settings are often sufficient. Their big disadvantage, however, is that they are designed to write much more complex programs than an initial course needs. Usually a single file is enough for solving a simpler task, but IDEs generally think in the concept of projects with many files. Their interface is often very complicated, as they provide many functionalities through menus, toolbars, panels, and settings. They give much more than is needed, and this may distract the attention.

Another thing that can make the usage of any kind of desktop environment

problematic is the installation process. Editors should be installed in classrooms, and they need to be installed at home computers by the student. Desktop applications may have other dependencies, and they need to run on different operating systems. This complexity may lead to errors.

One of the final steps after coding is testing (followed by detecting and correcting errors), consisting of two steps: syntax and semantic checking. *Syntax errors* are revealed during compilation: the error list usually appears in a separate panel, and in better environments the error is indicated in the source code as well.

Semantic testing has two important parts: preparing the test cases and the executing the tests. In worse scenarios test cases are announced only verbally, but in better cases they are written to the board or to the exercise book. Testing is initially made manually: the students enter the input data manually in the command line window and monitor the response of the program. The command line usually appears in a separate window independently of the developer environment. For longer inputs, tests are written to a file, which are redirected to the standard input of the running program. Creating these test files can be done in the development environment or in another program. Redirection is often not possible in IDEs, so testing requires the opening of a command line window, which adds complexity to this step.

Development in a traditional environment is often supplemented with online judging systems (e.g. [4, 5]), which verify the code objectively. These systems usually contain only the task descriptions and the batched, automated verification services, the development is still performed in separate IDEs.

Analysing this process, several problems can be identified in the relation of programming environments and introductory courses, beginner students:

- IDEs are too general: they are general-purpose development environments, and are not intended to support specific, methodologically-based problem solving processes, which would be better for beginner students. They are only focus on the implementation part of this process.
- IDEs can not guide the student, it is left for the teacher or the students themselves.
- Other knowledges are also needed, e.g. using the command line, redirecting, uploading and downloading files.
- Considering the available number of lessons per week, teaching the different tools and the whole toolchain proportionally takes much more time than teaching the essentials of problem solving, compared to their importance.
- If students come from a gamified environment, using professional IDEs is big gap to leap through.

Looking at the number of supporting programs, the whole workflow is too complex: in order to achieve their goals, students need to focus on eight different sources of information on seven different platforms:

1. Task description (separate window)

2. Design (board or exercise book)
3. Coding (IDE)
4. Compilation (IDE, separate panel)
5. Running (separate console)
6. Writing test files (separate window)
7. Testing with files (separate console)
8. Automated testing (separate web application)

Considering the heterogeneity of these introductory courses, mainly coming from the previous knowledges of students, and the complex workflow that traditional programming environments provide, some demands can be formulated against a programming environment:

- Support beginners: complex processes should be made easier or left out.
- Be specific for introductory courses: it is not needed to be prepared for solving complex tasks. Introductory courses has simple tasks. Support these and support them well.
- Support simple programs: programs consist of one file, they need to read from standard input and write to standard output.
- Support the steps of task solving process: programming environment should lead the students' hand during the process, and should support all of the important steps of task solving process.
- Support methodology: what is important methodologically, it should be supported by the environment, if it is possible.
- Support curriculum: be flexible enough to support different introductory curriculum.
- User-friendly interface: ignore every distracting element from the user interface.
- Monitor students' performance: support monitoring of the progression, and make room for further personalization (e.g. giving different tasks for different students based on their results).
- Stand-alone usage, practice mode: the programming environment should be used with or without the teacher's direction.
- No installation: be platform-independent avoiding errors during installation.

3. Existing alternatives as solutions

Beside the traditional development environments there are other programming platforms that can provide an alternative solution for the problematic aspects of introductory programming courses introduced above. Every alternative environment tries to operate with a simplified user interface, where all the necessary information is available in the same program. It is common in every environment that they are web-based which provides all the features that the web platform can bring: ubiquity, no-installation set-up for the user, easier maintenance. Of course, these environments are different according to their specific needs.

The first group of these programming environments consists of the *online learning platforms*, like CodeAcademy [6] or Khan Academy [7]. They are designed to be used alone without any external guidance; they proceed forward in small steps, introducing small amount of new knowledge at one time. They are using textual descriptions or video tutorials to introduce a topic, and an online editor with automatic tests for practising. It would be hard to use them in a certain curriculum or in a lesson, and they do not support some parts of the task solving procedure, like planning, manually testing and debugging.

The members of the second group are the *online code editor environments*, such as CodingGround [8], Rextester [9], jDoodle [10], or Ideone [11]. They are web-based IDEs, with single or multiple file support, and sometimes with a built-in console (CodingGround). Usually standard input can be specified, and they give back the result of the compilation and the text of the standard output. They try to be general, so they can not support many of the demands that were formulated against an introductory programming environment: there is no place to give or collect task descriptions (or at least in comments), they do not support the methodologically formed steps, they do not support testing. However, these environments are great examples, that the development of command line applications can be achieved in browsers with the help of the web-platform.

A variation of the latter group is the online programming contest platforms or *code training platforms* such as CodeChef [12] or Codewars [13]. They are more than the previous group in a sense that they provide task description, automated tests for checking the solution, and sometimes manual tests can be given also. But they are lack of flexibility, activity monitoring can not be fulfilled, and teachers can not give their own tasks to the system. From this point of view they operate as a combination of an online IDE and judge system.

The last group of alternative environments is consist of those *online editors* which try to focus *on educational problems*. One of those web-based platform is repl.it [14]. It started as an online IDE with a built-in console, but later it was extended with some very useful educational tools, like classroom management, creating assignments, automated tests, monitoring student activities, giving task descriptions. These features are great and comes handy in certain situations, but manual testing is missing among those features, and thus some methodology-based requirements are not fulfilled.

4. The proposed programming environment

Our proposed programming environment tries to solve those issues which come from the scattered nature of the traditional programming environments along with some methodological considerations to help beginner students in learning programming. The first version of our in-browser programming environment [15] eliminated the distracting elements, pulled together the different type of tasks, except for the planning phase, into one user interface, where the task description, the coding area, the input and outputs of manual tests, and automatic tests took place. With these

it kept the attention in one place, it supported the steps of the problem solving process, and code editing had all the features a beginner needed in a comfortable, user-friendly environment. One of the main features of the first version was that it worked in an isolated environment without internet connection (offline). But this latter feature was this environment main drawback: it narrowed the potential programming languages into JavaScript and TypeScript (or, strictly speaking, any language that can be compiled in the browser), it did not give any chance to monitor students activities, only one user test could be given.

Learning from the drawbacks of the first version, the new version of the proposed programming environment was rewritten from ground up around similar user interface design principles, but with very different operations in the background. The user interface of the new environment can be seen in Figure 1. It is divided into two parts. Task description, manual and automatic tests are available in a tabbed panel on the left, while an easy-to-use and feature-rich code editor fills the right side of the browser window with the necessary buttons and informations. The workflow is the following: students can choose among the available tasks in the drop-down menu beside the logo; can read the task description; can make the planning outside of the environment; can implement the solution in the code editor; can make multiple manual tests to determine the correctness of the solution by themselves; can verify the solution with the help of automatic test prepared by the teacher.

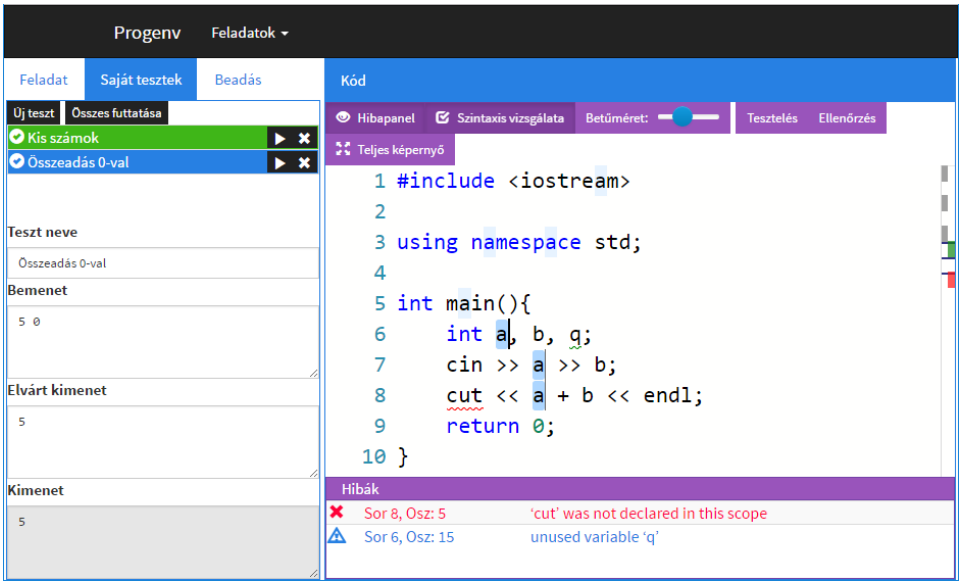


Figure 1: The user interface of the proposed programming environment

The heart of the application, the background mechanism was moved to server

side, where far more opportunities are available. The new architecture can be seen in Figure 2. Every operation that needs some language-specific feature (syntax checking, compilation, testing) sends a request to a REST API (Representational State Transfer Application Programming Interface) on the server side. Due to security reasons compilation and execution needs to be run in a sandboxed environment. The HTTP response contains the results of the requested operation, and this information is displayed on the interface.

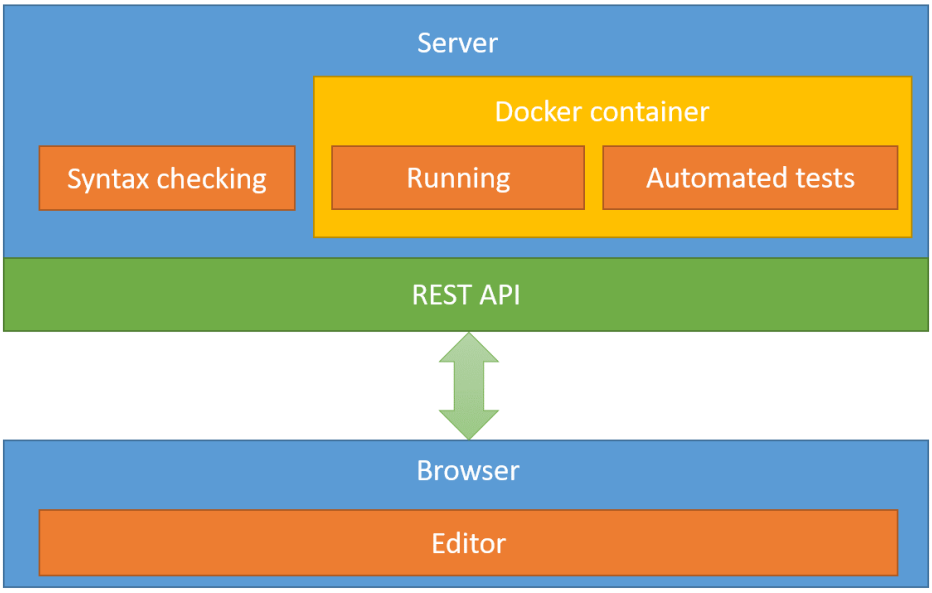


Figure 2: The schematic architecture of the proposed programming environment

This version of the environment follows the concept of single-page applications. The technologies that were used involves React, React-router, MobX, Monaco editor, Flexbox, Markdown on the client side, and node.js, express.js, Docker on the server.

5. Discussion and summary

The main advantage of the proposed programming environment that it keeps the problem solving process in focus (instead of the environment). It gains time both on the students' and the teacher's side. With different tasks and monitoring it opens up possibilities towards personalization and differentiated works. The web platform makes it possible to get rid of the installation process, to learn independently of time and place, and with sufficient automations it can offer off-class learning

opportunities [16, 17].

The following type of tasks can be used in this environment:

- implementing a solution from scratch (or with minimal initial code);
- implementing one or more specific functions, the other part of the code is written already;
- correcting errors in a pre-made, but wrong program; it develops code reading.

Summarizing, the proposed web-based programming environment can help the learning processes of beginner students in an introductory programming course. The environment is comfortable, has a non-distracting user interface, supports methodology, workflow and curriculum, and its flexible, language-agnostic architecture opens up new way towards personalisation and user monitoring. In this form it could support in-class and stand-alone usage as well.

There are many ideas for further development. User management is still missing, there should be pages where new tasks can be prepared, where task and user assignment could be achieved. User activity monitoring and personalisation would be great, and it could serve as a potential assignment platform as well. Debugging is still an issue.

References

[1] Code Studio, `https://studio.code.org/` [cited 2017 May 22]

[2] Scratch, `https://scratch.mit.edu/` [cited 2017 May 22]

[3] CodeCombat, `https://codecombat.com/` [cited 2017 May 22]

[4] "Bíró" judging system on the Faculty of Informatics, Eötvös University, `http://biro.inf.elte.hu/` [cited 2017 May 22]

[5] "Mester" judging system in Hungary, `http://mester.inf.elte.hu/` [cited 2017 May 22]

[6] Codecademy, `https://www.codecademy.com/` [cited 2017 May 22]

[7] Khan Academy, `https://www.khanacademy.org/` [cited 2017 May 22]

[8] CodingGround, `https://www.tutorialspoint.com/codingground.htm` [cited 2017 May 22]

[9] Rextester, `http://rextester.com/` [cited 2017 May 22]

[10] jDoodle, `https://www.jdoodle.com/` [cited 2017 May 22]

[11] Ideone, `https://ideone.com/` [cited 2017 May 22]

[12] CodeChef, `https://www.codechef.com/ide` [cited 2017 May 22]

[13] Codewars, `http://codewars.com/` [cited 2017 May 22]

[14] Repl.it, `https://repl.it/` [cited 2017 May 22]

[15] Horváth, Gy., Menyhárt, L. Webböngészőben futó programozási környezet megvalósíthatósági vizsgálata, *INFODIDACT 2016* Paper 3. (2016)

[16] Horváth, Gy., Menyhárt L. Oktatási környezetek vizsgálata a programozás tanításához, *INFODIDACT 2014* Paper 7. (2014)

[17] Horváth, Gy., Menyhárt, L., Zsakó, L. Egy webes játék készítésének programozás-didaktikai szempontjai, *INFODIDACT 2015* Paper 2. (2015)

Annales Mathematicae et Informaticae
***48** (2018) pp. 33–42*
http://ami.uni-eszterhazy.hu

Efficiency Analysis of the Vertex Clustering in Solving the Traveling Salesman Problem

László Kovács, Anita Agárdi, Bálint Debreceni

University of Miskolc, Institute of Information Sciences
kovacs@iit.uni-miskolc.hu

Submitted March 5, 2018 — Accepted September 13, 2018

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. Due to the high complexity of TSP, there exists no algorithm for global exact optimization with polynomial cost. In order to provide an acceptable solution for real life problems, the TSP are usually solved with some heuristic optimization problem. The paper proposes a multi layered optimization model, where the node set is partitioned into clusters or into hierarchy of clusters. Based on the test experiments the proposed method is superior to the single level optimization method for both the TSP and MTSP problems.

Keywords: Traveling Salesman Problem, Clustering

MSC: 90-08

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most intensively investigated optimization problem in graph theory. The TSP is a problem to find the shortest path in a graph visiting every nodes exactly once and returning to the start node [5]. The solution path is a Hamiltonian cycle of the graph. The TSP is a NP-hard problem [1], that means it is at least as hard as the hardest problems in NP.

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 $\mathcal{O}(n^2 2^n)$ complexity. In order to provide an acceptable solution for real life problems, the TSP is usually solved with some heuristic optimization method.

From the family of popular evolutionary algorithms, the following methods are used most widely to find the good approximation of the optimal Hamiltonian cycle:

- Genetic algorithm [6]: the parameter vectors are optimized using the selection, crossover and mutation operators;
- Particle swarm optimization [7]: more agents are generated which move randomly but the optimum found by them is reinforced by other members of the colony;
- Ant Colony Optimization [8]: more agents are generated which collaborate with their environment;
- Tabu search [9]: the local search phase is extended with prohibitions (tabu) rules to avoid unuseful position testings.

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

- Nearest neighbor algorithm [10]: it selects the nearest unvisited node as the next station of the route;

- Pairwise exchange of edges [10]: two disjoint edges are removed from the route and two new edges are involved into the route to reduce the total cost.

In our investigation, we have focused on the most widely used method, the application of Genetic Algorithm.

2. Solving TSP using Genetic Algorithm

In Genetic Algorithm, the chromosomes or individuals are the basic building blocks in search state representation. In the TSP problem, a state of the search space corresponds to a route in the graph, i.e. to a permutation of the nodes. In our implementation, the permutation is given with the sequence of node indexes in order of traversing. Thus the route $(n_1, n_4, n_3, n_5, n_2)$ is given with the sequence $(1, 4, 3, 5, 2)$.

In the case of MTSP problem, the chromosome should represent the description of every cycles. There are four main representation forms for genotypes in the MSTP problem [11]:

- One chromosome technique (a special gene is used to separate the different cycle sections);

- Two chromosome technique: one chromosome is used for the description of the linked cycles, while the second contains the sections for the different agents;

- Multi chromosome technique: each agent has a specific chromosome to describe its route; the different chromosomes should be synchronized;

- Two-part chromosome technique: the chromosomes contain two parts. The first part is the linked list of the separate cycles, while the second part contains the cycle length values for the separate agents. For example, the chromosome $((1, 5, 6, 2, 3, 4)(4, 2))$ describe the following graph routes:

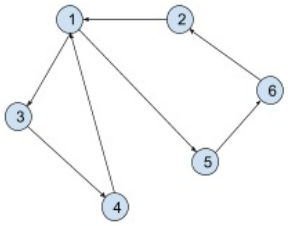


Figure 1

The implemented program applies the two-part chromosome technique to solve the MTSP problem. Regarding the genetic operators, the mutation of the route segment is performed with a swap operation. For example, the $(1, 5, 6, 2, 3, 4)$ route is modified to $(1, 5, 6, 4, 3, 2)$. In the case of two-part chromosome technique, the two segments are altered separately. In the agent segmentation part, the mutation is executed with selection of two positions where one of the values will be increased and the other will be decreased by one. For example, a mutation of $((1, 5, 6, 2, 3, 4)(4, 2))$ can be given with $((1, 3, 6, 2, 5, 4)(3, 3))$. For the crossover operation, the partially matched crossover method was implemented. In this method, some gene pairs are selected for substitution and the corresponding gene values are replaced with their substitution values. In the case of MTSP, only the route cycle part is altered with the crossover operation.

In our analysis, the base TSP and MTSP algorithms using genetic algorithms are used as baseline algorithms to be compared with the algorithm using node clustering.

3. Clustering Methods in Optimization

In the optimal route of the TSP, the edges usually connect the nearest nodes to each others. It would have no sense to make big jumps over and back among the nodes. This kind of behavior induces the heuristic rule of locality: near nodes in the route are near in the route graph too. Based on this heuristic assumption, it seems reasonable to group the near nodes into clusters and to perform a hierarchical optimization. There is a within-cluster optimization to determine the optimal route within the cluster and there is an intra-cluster optimization to determine the optimal path among the clusters. This approach implements the widely used divide and conquer concept, the problem of big complexity will be split into several subproblems of lower complexity.

In the literature, a big variety of clustering methods can be found. Most of the methods belong to the category of distance-based (discriminative) clustering [8] ,where the similarity between the objects are determined first and then the groups of similar objects are constructed. The main benefit of the distance-based methods is the simplicity and the descriptive power of the corresponding algorithm. In the case of model based clustering / generative clustering [7] approaches, a model type

is specified a priori. A set of probabilistic generative models are defined, where each model corresponds to a group. The model for the cluster indexed by i is given with a parameter set λ_i. The method of expectation maximization is used to determine the assignments between the objects and clusters.

From another orthogonal viewpoint, the clustering methods can be categorized as hierarchical and partitional clustering. In the case of hierarchical method [12], a hierarchy of partitions are generated. At the top level, all objects are assigned to a single top-cluster, while the leaf nodes corresponds to the objects as singleton clusters. The most widely used hierarchical clustering method is the HAC [13] method, which uses an agglomerative clustering [14] algorithm, where existing groups are merged with similar objects / groups into new extended groups.

The partitional clustering [14] methods partition the objects into groups based on some optimization criteria. The objects are usually re-assigned from one group to some other group during the learning process. The k-means [15] clustering uses this partitional approach where clusters are represented by the centroids of the cluster members. This k-means method is the dominant method on the field of cluster analysis and also we have implemented this clustering as the baseline method in our investigation.

The input of the k-means method is the object set in a vector space and the initial set of the cluster centroids. The number of the required clusters and the initial positions of the centroids should be given as input parameters. The goal of the algorithm is to optimize the intra-cluster distances, i.e. the within-cluster sum of squares :

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} d^2(x, c_i)$$

In the formula, C_i denotes the i-th cluster with the centroid c_i. For the evaluation of the clustering, we have used the Silhouette measure [16]:

$$s_k = \frac{b_k - a_k}{max(a_k, b_k)}$$

where

a_k: the average distance of x_k to objects in cluster C_k

b_k: the average distance of x_k to objects in nearest cluster different from C_k.

4. Solving TSP using Clustered Vertices

In the literature, there are only few publications on application of clustering in TSP problems. In [2], the k-means clustering technique is implemented in the proposed two-levels optimization system. The work analyzes only the one-salesman problem and does not discuss the method of the intra-cluster optimization level. Only the one traveling salesman problem is investigated also in the work [4], where the HAC clustering method was applied. The publication [3] proposes a solution for the

multi salesmen TSP domain. First the clusters of the nodes are generated and then each cluster is assigned to a single salesman.

In our investigation, which uses the k-means clustering method, we have extended the existing approaches with three novel elements:

- introduction of tree and multi level optimization
- adaption of k-means clustering to the MSTP problems.
- optimal selection of the k value

The three level optimization means that a hierarchy of k-means clustering is built up on the node set. In the single level clustering, the number of the nodes within a cluster was usually too high, thus the applied genetic algorithm could not achieve a good result. In order to reduce the complexity of the input data for within cluster TSP, the generated clusters can be clustered into subclusters in order to use the genetic algorithm with smaller node sets.

The multi level optimization can be used also for solving the MSTP problems. In this case, the top level corresponds to the salesmen, the number of clusters is equal to the number of salesmen. The resulted clusters can be sub-clustered into smaller sets for the genetic algorithm to solve the separate TSP problems.

The desktop test program was developed in Java and it uses the WEKA API to perform the clustering operations.

5. Test Results

We have performed several tests for both the TSP and MTSP problems with clustering and without clustering. The parameter ranges and of the test runs is given in Table 1 .

	TSP no clustering	MTSP no clustering	TSP with 2-level clustering	TSP with 3-level clustering	MTSP with 2-level clustering
number of nodes	10-1000	10-1000	10-1000	10-1000	10-1000
number of iterations	1000	1000	1000	1000	1000
size of population	100	100	100	100	100

Table 1: The parameter range of the tests

Regarding the efficiency of the different methods, Table 2 summarizes the quality of the optimization (length of the shortest path found by the algorithm). It can be seen that the application of the clustering module improves significantly the efficiency of the optimization process.

number of nodes	TSP no clustering	TSP with 2-level clustering	TSP with 3-level clustering	MTSP no clustering	MTSP with 2-level clustering
50	763	523	550	1328	1043
200	4358	1106	1202	6473	2385
400	10938	1647	1791	16820	4177
1000	36156	5801	3528	41052	6097

Table 2: The length of the local optimum route

The time costs of the different optimization process is shown in Table 3 .

number of nodes	TSP no clustering	TSP with 2-level clustering	TSP with 3-level clustering	MTSP no clustering	MTSP with 2-level clustering
50	5.9472	5.4596	9.2239	3.1367	6.2551
200	50.6509	20.6353	25.492	46.3972	21.1015
400	90.4167	4.384	6.1536	66.3395	4.6927
1000	729.451	12.1869	12.7243	555.1245	11.3891

Table 3: The run time of optimization process (sec)

The Figure 2 shows the generated routes for a sample MSTP problem with a single source and destination node. In Figure 3 and Figure 4, the two main clustering methods are compared from the viewpoint of clustering efficiency. Our experiences show that both methods have the same efficiency, they can provide very similar results. Regarding the optimal route length, the average distance is about 4%. The two clustering methods show a larger difference from the viewpoint of execution costs. Here, the k-means methods provides a faster execution than the HAC method.

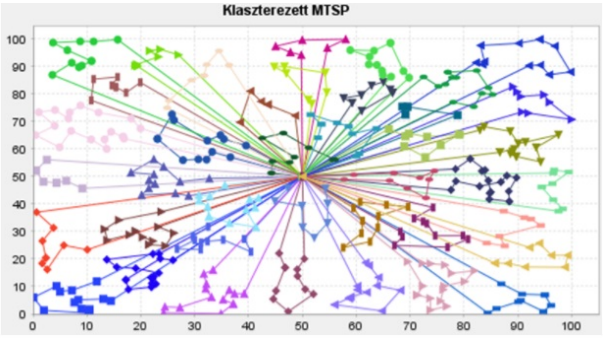


Figure 2: Optimum route of a sample MTSP problem

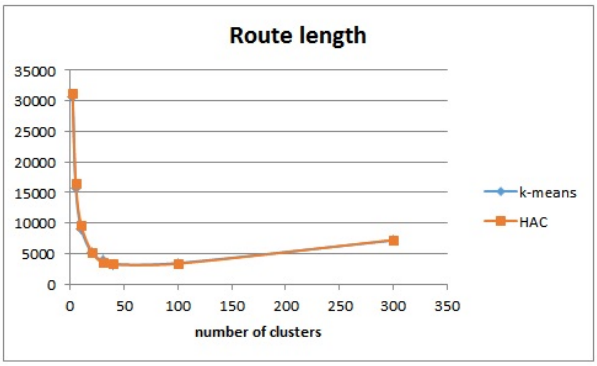


Figure 3: Optimum route with the different clustering methods ($N = 1000$)

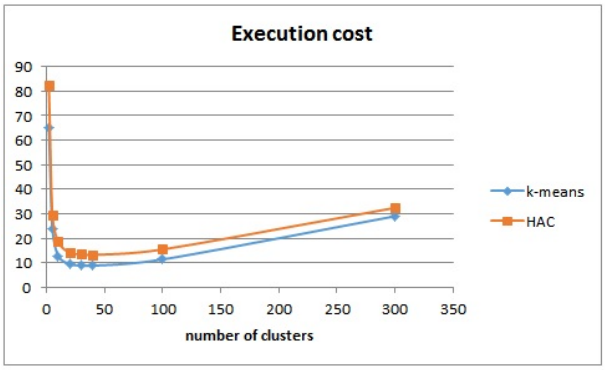


Figure 4: Execution cost of the route optimization ($N = 1000$)

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 optimization, the node set is partitioned into clusters or into hierarchy of clusters. The top cluster covers the whole input domain. For each cluster, a separate within cluster optimization is performed and then the local, cluster-level 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 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 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] 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.

[2] Priyanka Yashwant Chavan, Ashwini Ravindra Gundale, Mrunali Shekhar Thorat, Ashwini Hirendra Jambhulkar, Two-Level Genetic Algorithm for Clustered Traveling Salesman Problem with Application in Large-Scale TSPs, *International Journal of Advance Research in Computer Science and Management Studies*, (2015), 335–340.

[3] R.Nallusamy, K.Duraiswamy, R.Dhanalaksmi, P. Parthiban, Optimization of Non-Linear Multiple Traveling Salesman Problem Using K-Means Clustering, Shrink Wrap Algorithm and Meta-Heuristics, *International Journal of Engineering Science and Technology* (2009), 129–135.

[4] S N S Kalyan Bharadwaj.B, és Krishna Kishore.G, Srinivasa Rao.V, Solving Traveling Salesman Problem Using Hierarchical Clustering and Genetic Algorithm, *(IJCSIT) International Journal of Computer Science and Information Technologies* (2011), 1096–1098.

[5] 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.

[6] 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.

[7] Elizabeth F. G. Goldbarg, Marco C. Goldbarg, Givanaldo R., Swarm Optimization Algorithm for the Traveling Salesman Problem, *EvoCOP 2006: Evolutionary Computation in Combinatorial Optimization* (2006), 99–110.

[8] Pan Junjie1, Wang Dingwei2, An Ant Colony Optimization Algorithm for Multiple Travelling Salesman Problem Proceedings of the First International Conference on Innovative Computing, *Information and Control (ICICIC'06)* (2006).

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

[10] C. Nilsson., Heuristics for the traveling salesman problem., *Tech. Report, Linköping University, Sweden* (2003), `http://www.ida.liu.se/~TDDB19/reports_2003/htsp.pdf`

[11] 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.

[12] Anna Szymkowiak, Jan Larsen, Lars Kai Hansen, Hierarchical Clustering for Datamining, *in Proceedings of KES-2001 Fifth International Conference on Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies* (2001), 261–265.

[13] Maria-Florina Balcan Yingyu Liang y Pramod Gupta, Robust Hierarchical Clustering, *Journal of Machine Learning Research 15* (2014), 4011–4051.

[14] Anitha Elavarasi, J. Akilandeswari , B. Sathiyabhama, A SURVEY ON PARTITION CLUSTERING ALGORITHMS, *International Journal of Enterprise Computing and Business Systems (Online) (Online)* `http://www.ijecbs.com` Vol. 1 (1 January 2011),

[15] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, An Efficient k-Means Clustering Algorithm: Analysis and Implementation, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2002), Vol. 24 881–892.

[16] Saitta S., Raphael B., Smith I.F.C.,A Bounded Index for Cluster Validity., *In: Perner P. (eds) Machine Learning and Data Mining in Pattern Recognition. MLDM 2007. Lecture Notes in Computer Science* (2007), Vol. 4571 174–187.

Annales Mathematicae et Informaticae
***48** (2018) pp. 43–50*
http://ami.uni-eszterhazy.hu

Bypassing Memory Leak in Modern C++ Realm

Dorottya Papp, Norbert Pataki

Dept. of Programming Languages and Compilers,
Fac. of Informatics, Eötvös Loránd University, Budapest
dorottyapapp@yahoo.com,patakino@elte.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Deallocation of dynamically allocated memory belongs to the responsibility of programmers in the C and C++ programming languages. However, compilers do not support the work of the programmers with error or warning diagnostics. Thus the result of this behaviour can be memory leak. Programs' memory consumption may be unreasonably big and even the operating system can be too slow because of the swapping.

We present some different scenarios when memory leak occurs. We show the root cause of the scenarios. This paper presents existing tools for detecting or avoiding memory leak. These tools work in different ways. We analyze the smart pointers of C++11 standard, Valgrind that is a run-time heap profiler, Hans Boehm's garbage collector and the Clang Static Analyzer. We present the pros and cons of the tools. We analyse how difficult it to use these tools, how the efficiency is affected and how these tools can be enhanced for overcome unwanted memory leak. We present our proposals to make the tools more effective.

Keywords: C++, smart pointers, memory leak, garbage collection

MSC: 68N15 Programming languages

1. Introduction

Memory leak can occur in programs written in C/C++ because the deallocation task of dynamically allocated heap memory belongs to the programmers. There is

no compiler support for this task and no background job for automatic deallocation. There are pros and cons of this approach. It can be also a problem that memory management of C and C++ is not the same. Programmers use `malloc` and `free` in C programs, but in C++ `new` and `delete` is used typically. However, C's constructs are available in C++, as well. These constructs do not just syntactically differ but have different semantics, so it can be problem if one mixes them up in the source code. However, `new` and `delete` also have versions for one object and for arrays that should not be mixed [2].

However, C++ is an ever evolving language that has been upgraded with new language constructs and new standard libraries in the last years [10]. Furthermore, a bunch of new subtle tools (e.g. static analysers) become available for better development. In this paper we analyse how the modern tools help us bypassing memory leaks. We deal with Valgrind framework that finds memory leak at runtime, and Clang Static Analyzer that is a modern, continuously improving static analyser. We present the C++11's standard smart pointers and the non-standard, but well-known Boehm garbage collector. We analyse what are pros and cons of these constructs.

This paper is organized as follows. We present the tools that are evaluated in this paper in section 2. We present some of our examples that cause memory leak in section 3. We have a large number of test cases to evaluate the tools. We analyse the tools based on the examples and other aspects in section 4. Finally, this paper concludes in section 5.

2. Tools

In this section we present tools that help us overcome memory leaks. The tools work in different ways. We distinguish these tools if they prevent or detect memory leaks.

2.1. Valgrind and Memcheck

Valgrind is a widely-used framework that is able to execute the code with many special validation features and gaining profiling information [7]. This tool executes the code in special "mocked" environment, so the code is untouched, and compiled, linked as usually [8]. However, this safe runtime execution has large overhead, so it cannot be used in production.

Valgrind is a comprehensive tool for detecting problems at runtime, but its primary aim is memory leak detection. It distinguishes different memory leak types:

- Definitely lost: no pointer points to the allocation when the program terminates
- Indirectly lost: no pointer points to that space which were able to access and deallocates the current allocation (e.g. if a root element of a binary tree is

definitely lost than every other nodes in the tree is indirectly lost).

- Still reachable: there is at least one pointer that points to the allocation
- Possibly lost: there is at least one pointer that points to the allocation but it is not exactly the same address that the `new` or `malloc` returns.

2.2. Clang Static Analyzer

Clang is a compiler infrastructure that is based on LLVM [5]. It has many related tools. Clang Static Analyzer uses static analysis and symbolic execution to detect different problems in the code [4]. It can realize divison-by-zero problems, using of uninitialized variables based by examining the source code. As a static analyser it does not execute the code.

The Clang Static Analyzer has three checkers that aim at detecting memory leaks in the source code:

- `unix.MismatchedDeallocator` – searches for incorrect deallocation, when `new`/`delete` and `malloc`/`free` are used together.
- `unix.Malloc` – searches for incorrect `malloc` allocated heap usage (e.g. double `free`, memory leak, etc.)
- `alpha.cplusplus.NewDeleteLeaks` – finds memory leak when the memory is allocated with `new`.

2.3. Boehm Garbage Collector

Many programming languages use garbage collector to ensure the minimalization of memory leak. C/C++ does not offer standard garbage collection (C++11 introduces a minimal ABI [1]). The Boehm garbage collector is able to work in C and C++ programs. It keeps track all variables in the program to check when it can safely execute the deallocation in the background. It uses a modified mark-and-sweep algorithm [3]. It has different interfaces for C and C++-like memory management.

2.4. Smart pointers

The standard smart pointers are able to deallocate memory when the smart pointer objects go out of scope. Smart pointers take advantage of the C++ template construct, so they are independent of the type of the managed memory. C++ template construction is very important feature regarding the performance. Effectiveness of C++ template constructs is still evaluated. The basic operations of smart pointers are those of the raw pointers but smart pointers offer some convenience methods. Different standard smart pointer types are available. However, dealing with memory usage optimization in concurrent execution is still problematic.

The smart pointers are based on the RAII (resource acquisition is initialization) principle: constructors and destructors are automatically executed in a well-defined moment [6]. Invocation of these operations is based on the smart pointer objects lifetime. The major standard smart pointers are `std::unique_ptr<T>`, `std::shared_ptr<T>` and `std::weak_ptr<T>`.

3. Examples

In this section we define examples that are used for evaluation. We have about sixteen use cases. In this section we present some of these. We have analysed how we can modify the examples for garbage collection or smart pointers.

The very first example is a simple one:

```
int main()
{
  int * p = new int;
}
```

Valgrind and Clang Static Analyzer detect the memory leak. However, a minimal modification presents the strong limitation of static analysis. If the memory allocation is executed in a function in a different compilation unit then the static analyser does not detect it. This modification does not affect Valgrind because Valgrind works at runtime and does not mind compilation units.

This example can be modified using a smart pointer to avoid memory leak:

```
#include <memory>

int main()
{
  std::unique_ptr<int> p(new int);
}
```

This example can be modified using garbage collector as well. This GC takes advantage of the fact that operator `new` can be overloaded:

```
#include "./gc_cpp.h"

int main()
{
  int * p = new(UseGC) int;
}
```

The following example presents the differences between C and C++ memory routines:

```
#include <cstdlib>

class Vec
{
public:
  Vec() {}
  ~Vec() { delete[] p; }
  void init(int i) { p = new int[i]; }
private:
  int * p;
};

int main()
{
  Vec * vec_pointer_new = new Vec;
  vec_pointer_new->init(5);
  delete vec_pointer_new; // no memory leak

  Vec * vec_pointer_malloc = (Vec*)malloc(sizeof(Vec));
  vec_pointer_malloc->init(5);
  free(vec_pointer_malloc); // memory leak
}
```

The `malloc` and `free` is responsible only for the memory management and cannot deal with C++'s constructs like constructor and destructor. The `new` and `delete` related to objects' lifetime, so these constructs call constructor and destructor, respectively. This can result in memory leak. Valgrind can detect this leak, but Clang Static Analyser does not report it.

One can think that memory leak obviously can be avoided with the help of smart pointers, but it is not true actually:

```
#include <cstdlib>
#include <memory>

int main()
{
  int * p = (int*)malloc(sizeof(int));
  std::unique_ptr<int> u_p(p); // mismatched malloc()/delete
}
```

However, smart pointers offer possibility to pass custom deallocation code snippet but it is not enforced. This problem can be realized at runtime with Valgrind, but cannot be realized with Clang Static Analyser. The major problem is that the C++ standard does not offer functor for `free`. One can develop it, but there is no standard approach for this scenario. On the other hand, functors have other difficulties [9].

Static analysers have an important advantage. These tools do not deal with runtime parameters and are able to check every execution paths. Let us consider the following code snippet:

```
#include <cstdlib>
#include <iostream>

int main()
{
  int * p = new int;

  int input;
  std::cin >> input;

  switch(input)
  {
   case 0: // do something
     break;
   default: // do something else
     delete p; break;
  }
} // memory leak if input==0
```

This potential memory leak is not guaranteed to be checked with Valgrind, but can be detected with Clang Static Analyzer because it does not deal with execution.

4. Evaluation

We have analysed the tools based on the extended set of test cases that we presented in the previous section. The tests revealed the following results:

- Valgrind detects most of the memory leaks in the examples. If the leak is occured on the execution the tool was able to find it. One of the major problems with Valgrind that complex applications are difficult from the viewpoint of execution.
- Clang Static Analyzer finds less memory leak than Valgrind. The major problem is related to cross-translation units and destructor calls. However, this approach does not mind execution paths.
- All memory leaks can be overcome with the garbage collector.
- Smart pointers do a good work in most cases but there are some use cases when smart pointers can also be used erroneously.

After the test cases, we have evaluated the tools based on the following characteristics as well:

- Setup – How difficult is to start the work with this tool
- Documentation – How detailed, well-structured the documentation of the tool is
- Portability – Which platforms can be used
- Further improvements – Is it a mature tool or is it a continuously improving one
- Appreciation – Is it a widespread tool
- Runtime overhead – How the tool affects the runtime
- Memory consumption overhead – How the tool affects the memory consumption
- Compilation time overhead – How the tool affects the compilation time
- False positives – Does the tool report a problem that is not problem actually
- Green field projects or code legacies – Does the tool support big code legacies or is it useful for new projects
- C or C++ support - How the tool is affected by C or C++ code

Our experiences based on the previous characteristics:

- Valgrind is an honored, well-known, widely-used tool. (For instance, during the development of Mozilla Firefox, OpenOffice, MySQL, NASA Mars Exploration Rover, Blender and CMake Valgrind has been used [11].) It is a mature tool, but there are limitations in portability. It does not affect the compilation time, but has overhead, so it cannot be used in production environment. It supports C and C++, as well.
- Clang Static Analyser does not affect the runtime circumstances, but the usage can take long time and has a rather big memory consumption. The documentation is not perfect. It can report false positives. It can be used with C and C++ code, as well. It can be used with code legacies.
- Boehm GC is not a well-documented one, it is hard to set up. It is difficult to use with code legacies. The garbage collector cannot be a standardized one, the community does not support it.
- Smart pointers: well-documented, portable because of the standard. Smart pointers increase the productivity but they cannot work together with pure C.

5. Conclusion

The memory management can be still problematic in C and C++ code. However, there are many tools that can help the programmers to avoid memory leak. We presented some tools for avoiding or detecting memory leaks: static analyser, runtime validation, smart pointers and a garbage collector. We defined a set of test cases to evaluate these tools. After this, we defined other aspects to evaluate and measure the tools' convenience. Based on these we have a comprehensive evaluation of these tools.

References

[1] Boehm, H. J., Spertus, M.: *Garbage collection in the next standard of C++*, in Proc. of the 2009 international symposium on Memory management (2009), pp. 30–38.

[2] Dewhurst, S. C.: "C++ Gotchas Avoiding Common Problems in Coding and Design", Pearson Education (2003).

[3] Edelson, D. R.: *A mark-and-sweep collector C++*, In Proc. of the 19th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '92) (1992), pp. 51–58.

[4] Horváth, G., Pataki, N.: *Source Language Representation of Function Summaries in Static Analysis*, In Proc. of the 1th Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems Workshop (ICOOOLPS 2016), Paper Nr. 6.

[5] Lopes, B. C., Auler, R.: "Getting Started with LLVM Core Libraries", Packt Publishing (2014).

[6] Meyers, S.: "Effective Modern C++", O'Reilly (2015).

[7] Nethercote N., Seward J. Valgrind: *A program supervision framework*, Electronic Notes in Theoret, Comput. Sci, **89(2)** (2003), pp. 44–66.

[8] Nethercote N., Seward J. Valgrind: *A framework for heavyweight dynamic binary instrumentation*, in Proc. of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI' 07), San Diego, (2007), pp. 89–100.

[9] Pataki, N.: *Advanced Functor Framework for C++ Standard Template Library*, Studia Universitatis Babeş-Bolyai, Informatica, **LVI(1)** (2011), pp. 99–113.

[10] Stroustrup, B.: "The C++ Programming Language", Addison-Wesley Publishing Company, Fourth edition (2013).

[11] Valgrind Official Home, `http://valgrind.org/`

Annales Mathematicae et Informaticae
***48** (2018) pp. 51–64*
http://ami.uni-eszterhazy.hu

Reflection in quadratic surfaces*

Hellmuth Stachel

Institute of Discrete Mathematics and Geometry, Vienna University of Technology
stachel@dmg.tuwien.ac.at

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

"The transparent cup" is the title of pictures which show an interesting phenomenon: The circular boundary c of the depicted plate appears as an ellipse which seems to coincide with the view of the reflection of c in the coffee-cup. Is this just by chance or is there a geometric theory behind?

In one example the circle c is the focal circle of the reflecting one-sheet hyperboloid, and for this particular case the displayed phenomen is a consequence of focal properties of quadratic surfaces. The tangent cones drawn from a fixed point P to a family of confocal quadrics are confocal and have therefore coinciding axes. These axes are the surface normals to the particular quadrics passing through P. Also the cones connecting P with the focal conics are included in the considered set of confocal cones. Therefore, all focal conics share the property: In each perspective, the images of these curves and their reflections belong to the same conic.

The goal of the paper is to highlight the geometric background, i.e., to focus on confocal conics and their spatial counterparts.

Keywords: confocal conics, confocal quadrics, reflection in quadrics

MSC: 51N20, 51N15, 68U05

*The author is grateful to Georg Glaeser, University of Applied Arts Vienna, for pointing his attention to this problem, and to Marko Knöbl, who was the first who identified this surprising phenomenon and gave a geometric proof. Furthermore, the author expresses his thanks to these persons and to Boris Odehnal for the permission to reprint their illustrations.

1. Reflection in conics and vertical cylinders

The stimulus for this article is a photograph showing a coffee-cup, which is made of ceramics and stands on a plate[1]. The cup looks transparent since the circular boundary of the plate is completely visible, even its section behind the cup. This apparent transparency is caused by the reflection in the cup: The mirror of the plate's visible boundary appears as an exact continuation of itself. Similar effects can be seen in Figure 1. Is this incidental, or is there a theory behind?

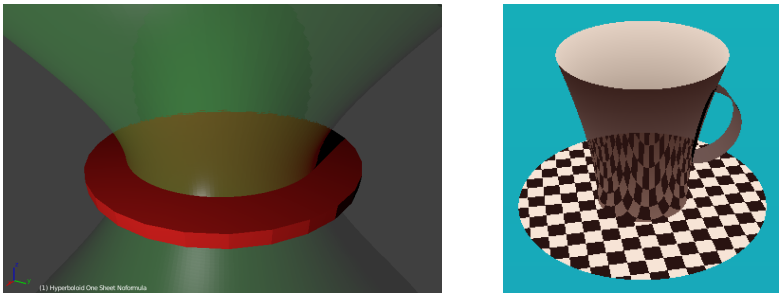


Figure 1: Why does the bounding circle of the plate continue in the reflection? (By courtesy of Kuno Knöbl [4])

Just to fix the terminology, we emphasize that under *'reflection'* in a conic or quadric we understand the physical reflection and not the projective inversion in a quadric[2]. We study the physical reflection in its geometric idealization, which is defined as a transformation applied, in general, to non-directed lines l in the following way: at each point P of intersection with the mirror $\mathcal{R}$, i.e., the reflecting curve or surface, the line l is reflected in the tangent plane τ_P or the normal line n_P to $\mathcal{R}$ at P.[3] The line l can have more than one point of intersection with $\mathcal{R}$ and hence more than one image. Note that each tangent line at P to $\mathcal{R}$ remains fixed.

To begin with, we recall the optical property of conics (see Figure 2, left). The reflection in an ellipse transforms rays emanating from one focus onto rays passing through the other focus. The same holds for hyperbolas when we ignore the orientation of the line. And finally, this optical property is also valid for each parabola when the ideal point of its axis is accepted as the second focus. Since the tangents drawn from a point X to an ellipse share the angle bisectors with the pair of lines connecting X with the focal points [1, p. 42], we can formulate a more general optical property (see Figure 2, right).

[1] See `http://imgur.com/N10ESfl`, retrieved April 2017.

[2] The latter is also known under the name 'projective inversion'; it is a rational transformation where corresponding points are conjugate with respect to ('w.r.t.', in brief) a given quadric and collinear with a given center.

[3] In the two-dimensional case, the reflection in any smooth curve preserves the density $\mathrm{d}p \wedge \mathrm{d}\varphi$ of oriented lines (satisfying $x\cos\varphi + y\sin\varphi = p$). For further details note [3, p. 6].

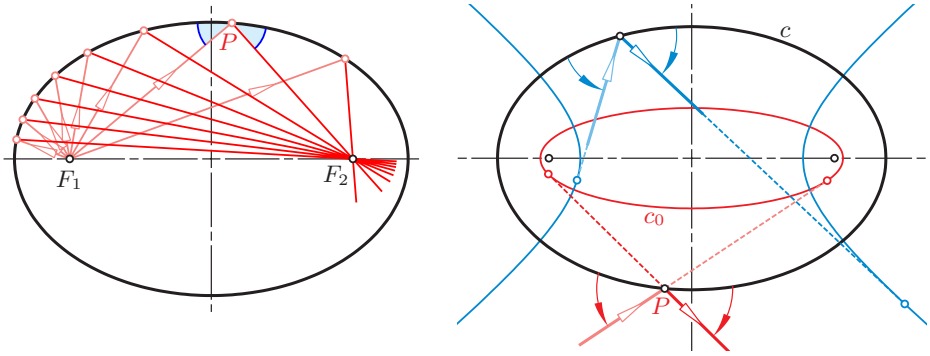


Figure 2: Optical properties of ellipses

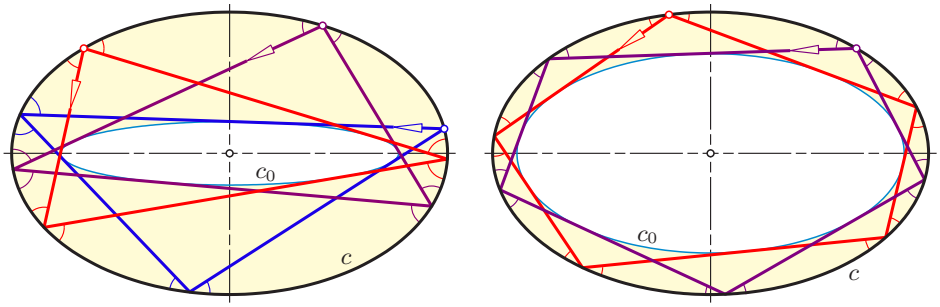


Figure 3: Closed billiards with three or five reflections in an ellipse

Theorem 1.1. *If any ray is reflected in a conic c then the incoming and the outgoing ray are tangent to the same conic c_0 being confocal with c.*

We recall that two conics are called *confocal* if they share the focal points. If in the family of confocal ellipses the minor semi-axis tends to zero the ellipse degenerates into the segment bounded by the two foci. This reveals that the statement of Theorem 1.1 includes the original optical property, too. Analogous degenerations show up as limits of confocal hyperbolas or parabolas.

Iterated reflections of any ray produce *billiards*. Due to Theorem 1.1, billiards in an ellipse c are always circumscribed to another ellipse c_0 being confocal with c. If one billiard inscribed in c and circumscribed to c_0 closes after n reflections then all these billiards close, independently of the choice of the initial point on c (Figure 3). This is a well known example of a Poncelet porism [1, p. 429ff]. All these closed billiards have even the same length, due to Graves' theorem (see [3] or [9] with much more details on billiards and reflections). By the same token, similar properties hold for billiards between two confocal ellipses (Figure 4).

We continue with a rather popular case of a reflection which is often used for producing anamorphoses [5]: Let a right cylinder $\mathcal{R}$ in vertical position be the

reflector. As illustrated in Figure 5, if observed from the center C, a point Q of the horizontal ground plane is visible at $P \in \mathcal{R}$. We call P an *reflected image of Q in $\mathcal{R}$ w.r.t. the center C*. The surface normal n_P to the cylinder at P is horizontal. Therefore the two segments PQ and PC of the reflected ray have the same inclination, and n_P is the interior angle bisector of $\angle QPC$, also, when seen in the top view.

As a consequence, for given center C and point Q, a reflected image $P \in \mathcal{R}$ has its top view P' on a *strophoid*, a curve of degree 3 [7]. This is the locus of points X in the ground plane such that a bisector of the angle QXC' passes through a given center M', which in our case coincides with the top view of the axis of $\mathcal{R}$ (Figure 6). Obviously, there is a second point of intersection between the strophoid and the cylinder $\mathcal{R}$ such that the interior angle bisector of $\angle QP'C'$ passes through M'. This shows that point Q can (theoretically) have two reflected images $P, \overline{P} \in \mathcal{R}$; the second one $\overline{P}$ lies on the back wall.

Figure 7 shows also the trajectory q of Q when a reflected image P on $\mathcal{R}$ runs along the horizontal circle $p \subset \mathcal{R}$. These trajectories are circular only in two particular cases: Either $P \in \mathcal{R}$ lies in the ground plane or P has exactly half of the height of C over the ground plane. Otherwise, the trajectories are *Pascal limaçons*.

This can be proved as follows (see Figure 7, left): The reflection at $P \in \mathcal{R}$ acts like the reflection in the surface normal n_P and maps the line PC onto the line PQ. If P has the height z over the ground plane, then the reflection in n_P maps Q onto a point P_2 in the height $2z$ on the line PC . Let P run with angular velocity ω along the parallel circle $p \subset \mathcal{R}$. Then the intersection point P_2 of CP with the plane in the height $2z$ runs with the same angular velocity ω on a horizontal circle p_2 with center M_2 on the cone connecting p with C.

In the top view we obtain Q' when $P_2' \in p_2'$ is reflected in n_P', which rotates with angular velocity ω about M'. This shows that the trajectory q of Q is traced when a first bar $M'M_2'$ rotates about M' with angular velocity 2ω while a second bar

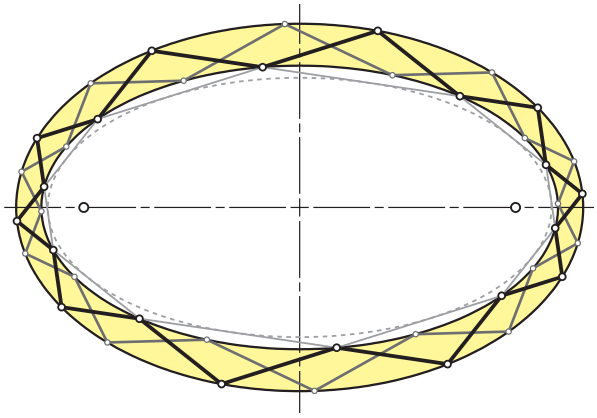


Figure 4: Closed billiards between confocal ellipses (20 reflections)

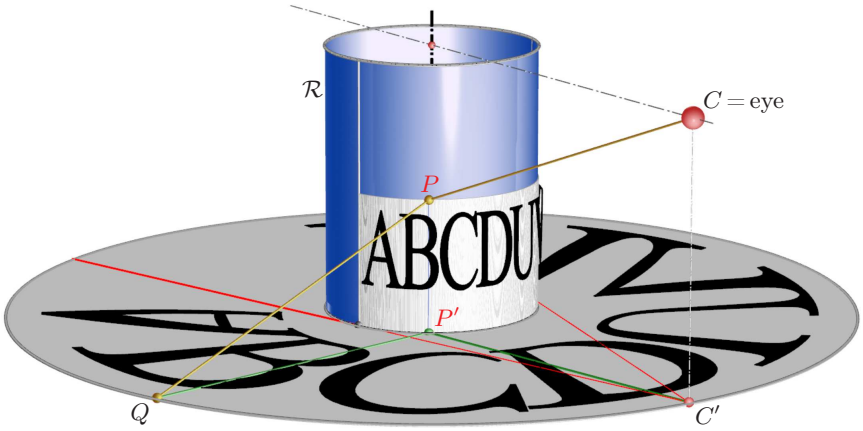


Figure 5: Reflection in a right cylinder $\mathcal{R}$: point Q in the ground plane and a reflected image P (by courtesy of Georg Glaeser)

$M_2'Q_0'$ rotates with the (absolute) velocity ω. A dyad $M'M_2'Q_0'$ moving this way generates as path of its endpoint a particular trochoid, namely a Pascal limaçon q' [10, p. 155], provided that no moving bar has length zero.

2. Confocal quadrics

The word 'quadric' stands now for regular surfaces of degree 2, i.e., for those of full rank 4. Of course, surfaces of degree 2 can also be cylinders or cones (rank 3), pairs of planes (rank 2), or double-counted planes (rank 1). In the projective setting, when cones of degree 2 are regarded as sets of tangent planes, they are dual to conics.

Definition 2.1. Two quadrics are called *confocal* if they have common axes and they intersect each plane of symmetry along confocal conics.

Let $\mathcal{E}$ be a tri-axial ellipsoid with semiaxes a, b and c in standard position. Then the one-parameter set of quadrics being confocal with $\mathcal{E}$ is given as

$$\frac{x^2}{a^2+k}+\frac{y^2}{b^2+k}+\frac{z^2}{c^2+k}=1 \quad \text{for} \quad k\in\mathbb{R}\setminus\{-a^2,-b^2,-c^2\}. \tag{2.1}$$

In the case $a>b>c>0$ this family includes (see Figure 8)

$$\text{for}\quad \begin{array}{ll} -c^2<k<\infty & \text{tri-axial ellipsoids } \mathcal{E},\\ -b^2<k<-c^2 & \text{one-sheet hyperboloids } \mathcal{H}_1,\\ -a^2<k<-b^2 & \text{two-sheet hyperboloids } \mathcal{H}_2. \end{array}$$

Their intersections with the plane $z=0$ share the focal points $(\pm\sqrt{a^2-b^2},\,0,\,0)$. In $y=0$ the common foci are $(\pm\sqrt{a^2-c^2},\,0,\,0)$, and in $x=0$ $(0,\pm\sqrt{b^2-c^2},\,0)$.

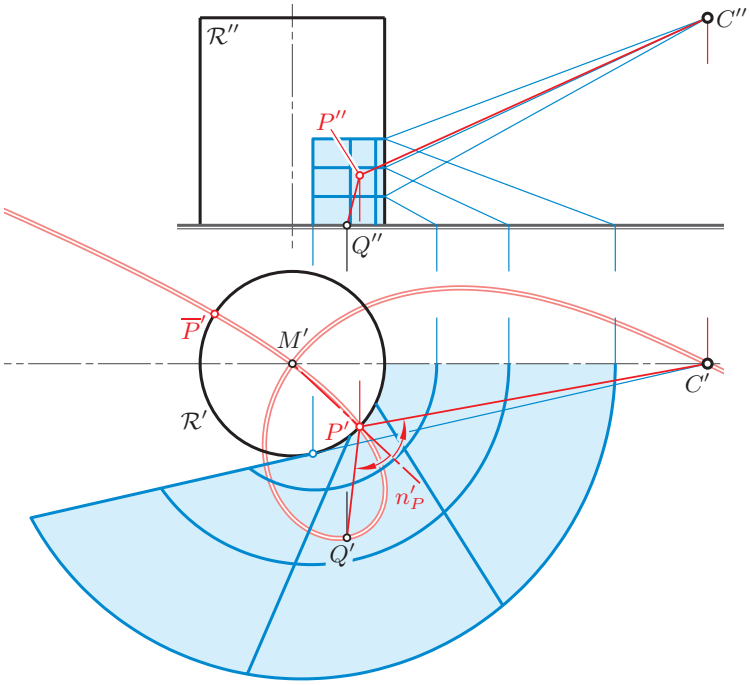


Figure 6: For a given center C and point Q in the ground plane the top views of the reflected images P and $\overline{P}$ lie on a strophoid

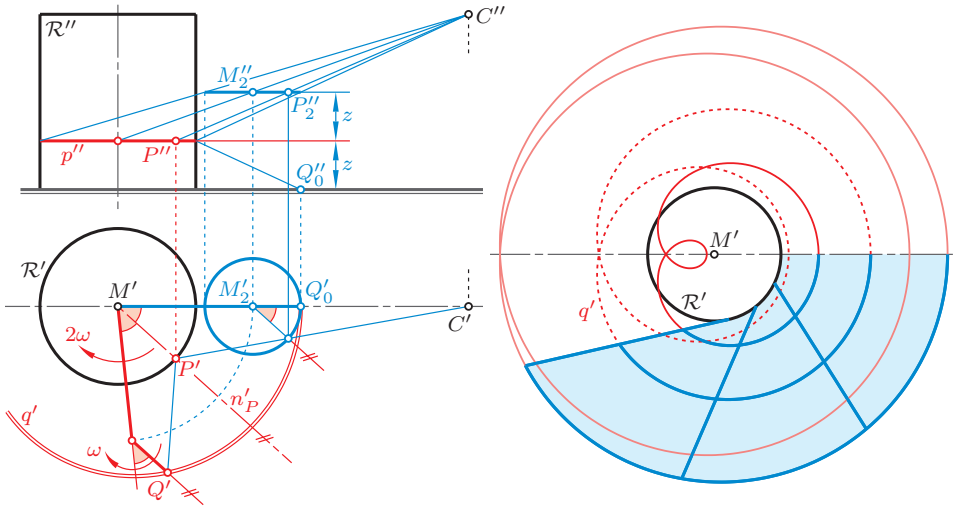


Figure 7: Parallel circles p on $\mathcal{R}$ are the reflected images of Pascal limaçons q in the ground plane

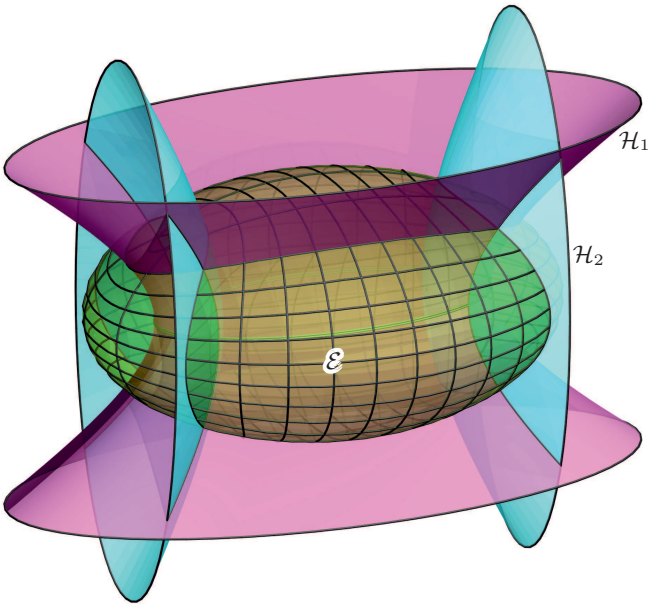


Figure 8: Confocal quadrics intersect mutually along their curvature lines (by courtesy of Boris Odehnal)

As limits for $k \to -c^2$ and $k \to -b^2$ we obtain 'flat' quadrics, i.e., the

$$\textit{the focal ellipse} \quad f_e\colon \quad \frac{x^2}{a^2-c^2}+\frac{y^2}{b^2-c^2}=1,\ z=0\,,$$
$$\textit{the focal hyperbola} \quad f_h\colon \quad \frac{x^2}{a^2-b^2}-\frac{z^2}{b^2-c^2}=1,\ y=0\,.$$

These two conics form a pair of focal conics: each is the locus of apices of right cones passing through the other conic [1, p. 137ff]. As a member of the confocal family, the two focal conics have to be seen as sets of tangent planes. Then they are rank 3 quadrics. According to this interpretation, all lines in space which meet any focal conic f in at least one point, are *tangent lines* of f. When below we speak of a *proper tangent line*, then we mean an ordinary tangent of the plane curve f.

The quadrics being confocal with an elliptic paraboloid $\mathcal{P}_e$ can be represented as

$$\frac{x^2}{a^2+k}+\frac{y^2}{b^2+k}-2z-k=0 \quad \text{for} \quad k\in\mathbb{R}\setminus\{-a^2,-b^2\}. \tag{2.2}$$

In the case $a>b>0$ this one-parameter set includes

$$\text{for} \quad \begin{array}{cc} -b^2<k<\infty \quad \text{or} \quad k<-a^2 & \text{elliptic paraboloids } \mathcal{P}_e,\\ -a^2<k<-b^2 & \text{hyperbolic paraboloids } \mathcal{P}_h. \end{array}$$

For all k, the vertices of the paraboloids have the coordinates $(0, 0, -k/2)$. Point $(0, 0, b^2/2)$ is the common focal point of the principal sections in the plane $x = 0$, and $(0, 0, a^2/2)$ is the analogue for the sections with $y = 0$.

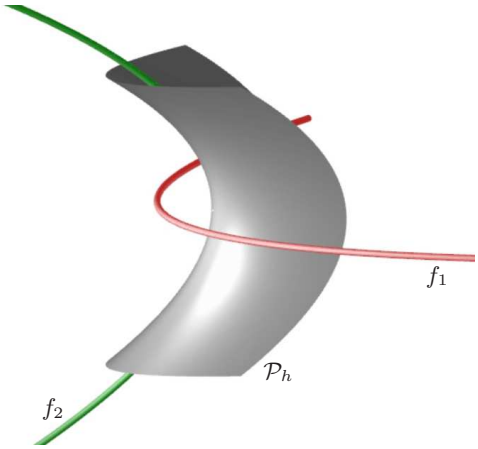


Figure 9: A hyperbolic paraboloid $\mathcal{P}_h$ together with its focal parabolas f_1 and f_2 (by courtesy of Georg Glaeser)

The limits for $k \to -a^2$ or $k \to -b^2$ define the pair of *focal parabolas*

$$\frac{y^2}{a^2 - b^2} - 2z + b^2 = 0, \quad y = 0\,,$$
$$\frac{x^2}{a^2 - b^2} + 2z + a^2 = 0, \quad x = 0$$

within the confocal family (Figure 9). For this pair of parabolas (compare with [1, Fig. 4.15] holds the same as mentioned above for an ellipse and its focal hyperbola.

For the sake of brevity, we ignore here the special cases of confocal quadrics of revolution. However, we recall that confocal quadratic cones can be given as

$$\frac{x^2}{a^2 + k} + \frac{y^2}{b^2 + k} - \frac{z^2}{c^2 - k} = 0, \quad k \in \mathbb{R} \setminus \{-a^2, -b^2, c^2\}. \tag{2.3}$$

Their intersections with the unit sphere result in confocal spherical conics. If $a > b > 0$ then for $k \geq c^2$ and $k \leq -a^2$ the cones do not contain real points other than the origin. The 'flat' limit for $k \to -b^2$ is a sector bounded by the lines

$$\frac{x}{\sqrt{a^2 - b^2}} \pm \frac{z}{\sqrt{b^2 + c^2}} = 0 \tag{2.4}$$

in the plane $y = 0$. These lines g_1, g_2 are called *focal lines* or *focal axes* of the cones, since they pass through the focal points of the corresponding spherical conics [1, p. 436ff]. The optical property, as shown in Figure 2, left, is also valid for spherical

conics. Therefore the reflection in a quadratic cone transforms planes through one focal axis g_1 into planes through the other axis g_2.

In the case $a = b$ we obtain confocal cones of revolution. Their focal axes coincide in the common axis of revolution.

Theorem 2.2. *In dual setting, confocal quadrics form a one-parametric linear system (range) of quadrics sharing the isotropic tangent planes. Hence, the range includes the absolute conic as a rank-3 dual quadric.*
Similarily, confocal quadratic cones form a range, which includes the isotropic cone with the same apex. Since pairs of isotropic tangent planes of a quadratic cone intersect along a focal axis, confocal cones have common focal axes.

Proof. In order to obtain the tangential equations, we note that the plane satisfying

$$u_0 + u_1 x + u_2 y + u_3 z = 0$$

is tangent to any surface of the confocal family (2.1) if and only if

$$(-u_0^2 + a^2 u_1^2 + b^2 u_2^2 + c^2 u_3^2) + k(u_1^2 + u_2^2 + u_3^2) = 0.$$

This is a linear combination of the homogeneous dual equation of $\mathcal{E}$ and that of the set of isotropic planes. The homogeneous dual equations of confocal parabolas satisfying (2.2) have a similar form, namely

$$(a^2 u_1^2 + b^2 u_2^2 - 2u_0 u_3) + k(u_1^2 + u_2^2 + u_3^2) = 0.$$

Finally, the dual equations of confocal cones, as given in (2.3), are

$$u_0 = 0, \quad (a^2 u_1^2 + b^2 u_2^2 - c^2 u_3^2) + k(u_1^2 + u_2^2 + u_3^2) = 0,$$

and they show again a range, spanned by the given cone ($k = 0$) and the isotropic cone with their common apex at the origin. □

Theorem 2.3. *The cones or cylinders drawn from any finite or ideal point P tangent to the quadrics of a confocal family or connecting P with one of the included focal conics are confocal. For finite P, the common and mutually orthogonal planes of symmetry of these confocal cones are tangent to one of the three quadrics passing through P.*

Proof. The considered tangent cones share all isotropic planes which are common to the confocal quadrics and pass through P. Hence, the cones are confocal, too. This is a classical result attributed to C. G. J. Jacobi 1834 [8, p. 204] and a special case of a theorem concerning ranges of surfaces of degree 2.

The tangent cone from P to a quadric $\mathcal{Q}$ splits into pencils of planes with two real or complex conjugate axes if and only if $\mathcal{Q}$ passes through P. Then the two axes are generators of $\mathcal{Q}$ and span the tangent plane at P. On the other hand, the planes spanned by the axes of singular cones are the common planes of symmetry of the confocal cones. This confirms that confocal quadrics form a triply-orthogonal system of surfaces. □

Let a tangent line l of a quadric $\mathcal{Q}_0$ pass through any point P on the quadric $\mathcal{Q}$ being confocal with $\mathcal{Q}_0$. Then, by virtue of Theorem 2.3, the reflection of l at P in $\mathcal{Q}$ is again tangent to $\mathcal{Q}_0$, since the tangent plane τ_P to $\mathcal{Q}$ is a plane of symmetry of the cone of tangents drawn from P to $\mathcal{Q}_0$. Thus we obtain the spatial analogue of Theorem 1.1.

Corollary 2.4. *Let $\mathcal{Q}$ and $\mathcal{Q}_0$ be two different quadrics in a confocal family. Then the reflection in $\mathcal{Q}$ maps the line complex of tangents of $\mathcal{Q}_0$ onto itself. In particular, the complex of lines meeting any focal conic f of $\mathcal{Q}$ remains fixed.*

We only report that, in general, a given line contacts two surfaces of a confocal family, and the tangent planes at the respective points of contact are orthogonal (see, e.g., [9, p. 65]). This can be concluded from the spatial version of the Desargues involution theorem. However, there are exceptions, called *focal axes* [8, pp. 205–206]: Such a line l has the property that the isotropic planes through l are tangent to any quadric and therefore to all confocal quadrics.

Lemma 2.5. *Each focal axis l of a quadric $\mathcal{Q}$ is either a generator of a ruled quadric confocal with $\mathcal{Q}$ or a proper tangent of a focal conic of $\mathcal{Q}$. At each point $P \in l$, the focal axis l of $\mathcal{Q}$ is also a focal axis of the cone drawn from P tangent to $\mathcal{Q}$ or to any other confocal quadric.*

Proof. Each plane through a generator l of a ruled quadric is tangent to this quadric at a particular point of l. Therefore also the isotropic planes through l touch the quadric.
The tangent cone or cylinder with apex P comprises all tangent planes of $\mathcal{Q}$ which

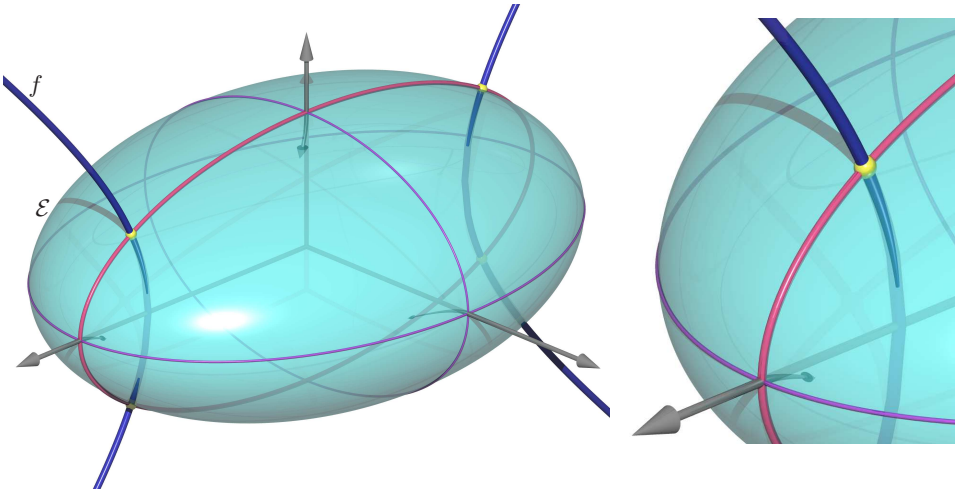


Figure 10: The perspective of the focal hyperbola coincides with its reflected image in the ellipsoid $\mathcal{E}$ (by courtesy of Boris Odehnal)

pass through P. If l is a focal axis of such a cone or cylinder then the isotropic planes through l are tangent to the cone and, hence, also to $\mathcal{Q}$. □

Corollary 2.4 is the main reason for the optical effects mentioned at the beginning (Figure 1): Let a quadric $\mathcal{Q}$ and a central projection with center C be given. If any line l of sight, which meets a focal conic f of $\mathcal{Q}$ at a point Q_1, is reflected at the point $P \neq C$ in $\mathcal{Q}$, then the transformed line still meets f at any point Q_2. Hence, the perspective images of point Q_1 and P are coinciding, where P is the reflected image of Q_2 w.r.t. C. This holds for all $Q_1 \in f$. Therefore in the perspective the focal conic f and its reflected image in $\mathcal{Q}$ w.r.t. C belong to the same conic ("Theorem of the Transparent Cup").

The quadric in Figure 1 is a one-sheet hyperboloid of revolution, and f passes through the focal points of the meridians. In Figure 10 we have a reflecting ellipsoid $\mathcal{E}$ and its focal hyperbola f.

We can even replace the focal conic f by any other quadric in the confocal family and claim, as given below.

Corollary 2.6. *Let a reflecting quadric $\mathcal{Q}$ be given together with a confocal quadric $\mathcal{Q}_0$. Then in a perspective with any center C, the quadric $\mathcal{Q}_0$ and its reflected image in $\mathcal{Q}$ w.r.t. C have coinciding contours. This is also valid when $\mathcal{Q}_0$ degenerates into a focal conic f: The perspective of f coincides with that of its reflected image in $\mathcal{Q}$.*

3. Reflecting cones in a quadric

By virtue of Theorem 2.4, a line meeting a pair of focal conics f_1 and f_2 keeps this property after reflection in any quadric being confocal with f_1 and f_2. The set of such lines is the union of cones of revolution with apices on the focal conics. Now we check what happens if the generators of one of these cones are reflected.

Theorem 3.1. *Let $\mathcal{Q}$ be a quadric with focal conics f_1 and f_2. The cone $\mathcal{C}_0$ of revolution, which connects any point $S_0 \in f_1$ with f_2, intersects $\mathcal{Q}$ along two conics c_1 and c_2. The reflection in $\mathcal{Q}$ along the conic c_i, $i = 1, 2$, transforms $\mathcal{C}_0$ again in a cone $\mathcal{C}_i$ of revolution passing through f_2 with an apex $S_i \in f_1$ (Figure 11).*

Proof. The tangent cones drawn from point $S_0 \in f_1$ to the quadrics of the given confocal family are confocal with the cone $\mathcal{C}_0$ connecting S_0 with f_2. Since the latter one is a cone of revolution, they all are cones of revolution with the proper tangent t_{S_0} to f_1 at S_0 as their common axis. These cones are tangent to the isotropic planes through t_{S_0}; the respective lines of contact are isotropic lines in the plane orthogonal to t_{S_0} through S_0.
On the other hand, the poles of a fixed plane w.r.t. the quadrics of a range are collinear. For each isotropic plane through t_{S_0}, which touches all quadrics confocal with $\mathcal{Q}$, the points of contact are alined with two points: S_0 as the touching point with f_1, and the respective absolute point as the touching point with the absolute

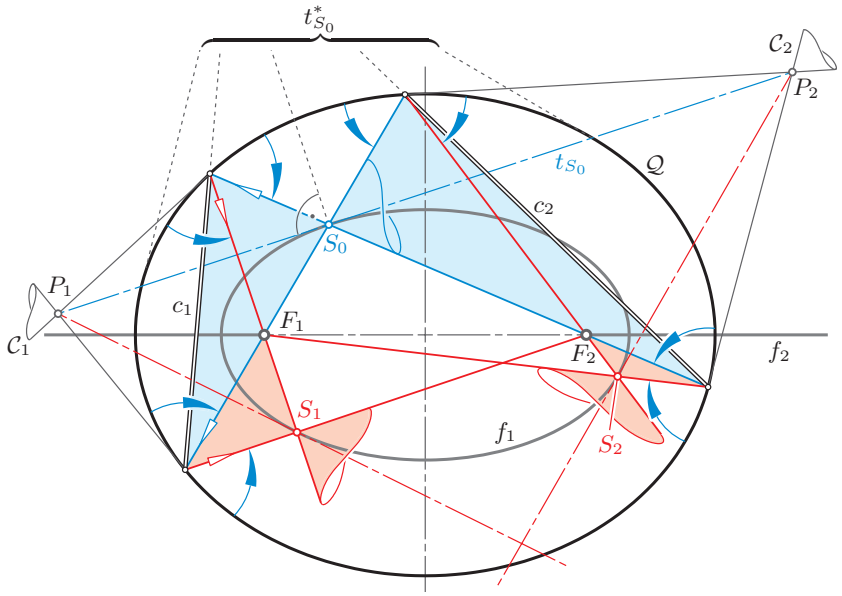


Figure 11: The reflection in the quadric $\mathcal{Q}$ transforms the right cone with apex $S_0 \in f_1$ onto two right cones with apices $S_1, S_2 \in f_1$

conic. Hence, the quadric $\mathcal{Q}$, like any other confocal quadric, contacts the cone $\mathcal{C}_0$ at two points. Consequently, the curve of intersection $\mathcal{Q} \cap \mathcal{C}_0$ splits into two conics c_1 and c_2, both passing through the points of contact on the line $t^*_{S_0}$, polar to t_{S_0} w.r.t. $\mathcal{Q}$. Figure 11 shows the scene after being orthogonally projected into the plane of the focal conic f_1.

Let P_i denote the apex of the tangent cone $\mathcal{C}_i$ of $\mathcal{Q}$ along c_i for $i = 1, 2$. In accordance with Lemma 2.5, the two proper tangents drawn from P_i to f_1 are the focal axes of $\mathcal{C}_i$. One of them is t_{S_0}, the other contacts f_1 at S_i (Figure 11). As already noted, the reflection in $\mathcal{C}_i$ transforms planes through t_{S_0} into planes through $P_i S_i$. Due to the contact between $\mathcal{C}_i$ of $\mathcal{Q}$ along c_i, for each point $X \in c_i$ the reflection in $\mathcal{Q}$ maps the line S_0X onto a line meeting the axis $P_i S_i$. On the other hand, by virtue of Corollary 2.4, the reflected line must also meet f_1 (and f_2). Hence, the reflection of S_0X coincides with S_iX, as stated in Theorem 3.1. For all $X \in c_i$, the planes spanned by the incoming and outgoing ray, which contain also the surface normal n_X to $\mathcal{Q}$, have the common trace S_0S_i in the plane of f_1. □

The given proof reveals that Theorem 3.1 can be generalized by replacing the focal conic f_2 with any confocal quadric $\mathcal{Q}_0$.

Theorem 3.2. *Let $\mathcal{Q}$ and $\mathcal{Q}_0$ be two confocal quadrics. Then the reflection in $\mathcal{Q}$ transforms each cone of revolution, which is tangent to $\mathcal{Q}_0$, into two cones of the same type.*

Remark 3.3. It can be shown that, conversely, the only smooth cones which by reflection in a general quadric correspond again to a cone, are those mentioned in Theorem 3.2.

From a limiting case of Theorem 3.1 we learn how the well known reflecting property of a satellite-TV receiving dish changes when the paraboloid of revolution is replaced with a general elliptic paraboloid.

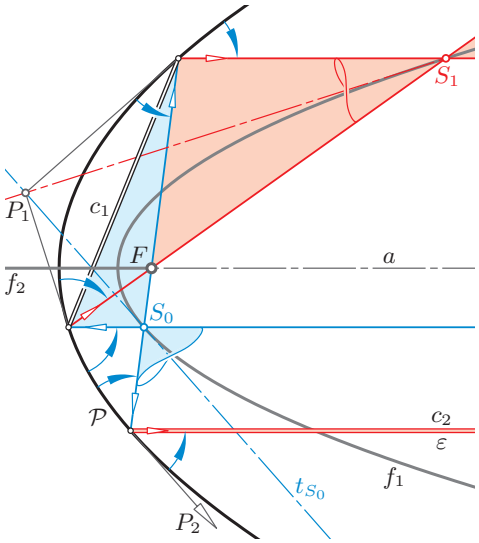


Figure 12: The reflection in the elliptic paraboloid $\mathcal{P}$ transforms the right cone with apex $S_0 \in f_1$ onto the right cone with apex $S_1 \in f_1$ and a pencil of lines parallel to the axis a in the plane ε

Theorem 3.4. *Let $\mathcal{P}$ be any paraboloid other than a paraboloid of revolution. Then the reflection in $\mathcal{P}$ maps all lines l being parallel to the axis a of $\mathcal{P}$ onto lines meeting both focal parabolas f_1 and f_2 of $\mathcal{P}$. The pencil of those parallels l to a, which lie in a plane ε orthogonal to the plane of f_1, is mapped onto a cone of revolution with apex $S_0 \in f_1$.*

The latter can also be concluded as follows (see Figure 12). Let c_2 denote the parabola $\mathcal{P} \cap \varepsilon$. The tangent cone of $\mathcal{P}$ along c_2 is a parabolic cylinder $\mathcal{C}_2$ with apex P_2 at infinity. After an orthogonal projection with center P_2 the cylinder $\mathcal{C}_2$ appears as a parabola $\mathcal{C}_2^n$. In this view the reflection in $\mathcal{Q}$ along c_2 is seen as a planar reflection in $\mathcal{C}_2^n$ which transforms lines parallel to the parabola's axis onto lines through the focus of $\mathcal{C}_2^n$. This focus coincides with the view of S_0, which is the point of f_1 with the proper tangent t_{S_0} passing through P_2.

Remark 3.5. The bundle of parallels to the axis a of the paraboloid $\mathcal{P}$ consists of all lines orthogonal to a plane. By virtue of the Theorem of Malus and Dupin [6, p. 446], the property of being a *normal line congruence* is preserved under reflection

in a surface. The surfaces orthogonal to the lines meeting the pair of focal parabolas of $\mathcal{P}$ are parabolic Dupin cyclides [1, p. 147ff]. We recall that the surfaces, whose normals intersect an ellipse and its focal hyperbola, are general Dupin cyclides.

References

[1] Glaeser, G., Stachel, H., Odehnal, B., *The Universe of Conics*, Springer Spectrum, Berlin Heidelberg 2016.

[2] Glaeser, G., Odehnal, B., Stachel, H., *The Universe of Quadrics*, Springer Spectrum (in preparation).

[3] Izmestiev, I., Tabachnikov, S., *Ivory's Theorem revisited*, Journal of Integrable Systems 2/1, xyx006 (2017) (https://doi.org/10.1093/integr/xyx006).

[4] Knöbl, M., *The Transparent Cup Theorem*, Retrieved from [http://karuga.eu/transparent-cup.html], 2016.

[5] Mazzalai, S., *Between Memory and Innovation: Algorithmic Analysis of some Catoptric Anamorphoses by Jean François Niceron,* Proceedings of the 17th ICGG, Beijing 2016, no. 27.

[6] Pottmann, H., Wallner, J., *Computational Line Geometry,* Springer-Verlag, Berlin 2001.

[7] Stachel, H., *Strophoids are auto-isogonal cubics.* G – Slovak Journal for Geometry and Graphics, ISSN 1336-524X, **12**, no. 24, 45–59 (2015).

[8] Staude, O.J., *Flächen 2. Ordnung und ihre Systeme und Durchdringungskurven*, in *Encyklopädie der math. Wiss.* III.2.1, no. C2, 161–256, B.G. Teubner, Leipzig 1915.

[9] Tabachnikov, S., *Geometry and Billiards,* American Mathematical Society, Providence/Rhode Island 2005.

[10] Wunderlich, W., *Ebene Kinematik,* Bibliographisches Institut, Mannheim 1970.

Annales Mathematicae et Informaticae
***48** (2018) pp. 65–73*
http://ami.uni-eszterhazy.hu

Functional model of a decision support tool for Air Traffic Control supervisors

Bence Számel, Géza Szabó

Budapest University of Technology and Economics
szamel.bence@mail.bme.hu
szabo.geza@mail.bme.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

The optimization of operator workload in Air Traffic Control systems is of high importance regarding both safety and efficiency of air transportation. The default means of optimizing workload in practice is division of traffic among controllers by splitting the airspace into sectors. Decisions about opening or closing sectors are made by a human supervisor who can sometimes encounter difficulties during decision making, especially when facing uncommon traffic situations.

This makes it advisable to create a decision support tool designed to automate decision making by suggesting a sector configuration based on traffic complexity to the supervisor. The main modules of this tool are centered on a model of cognitive functions that have to be executed by the supervisor throughout the decision making process. The functions include prediction of future traffic, calculation of complexity and determining optimal sector states to finally produce the optimal configuration. In this paper, we outline the formal description of the functionality of the tool's modules, focusing on the algorithms they should implement.

Keywords: Air Traffic Control, decision support, neural network

MSC: 68T99

1. Introduction

The Air Traffic Control (ATC) system is responsible for ensuring the safe and efficient movement of aircraft in controlled airspace. Controlled airspace is the

section of the atmosphere in which all air vehicles are compulsively required to comply with the instructions of Air Traffic Control Officers (ATCOs). ATCOs observe traffic in the airspace in real time and issue clearances in order to separate aircraft on conflicting routes or help them evade thunderstorm areas or restricted airspace. As ATCOs can pay attention to a limited number of aircraft at the same time, when traffic increases, the airspace is divided into sectors with different responsible ATCOs.

Nowadays the division of airspaces is done by using deterministic geographical and/or altitudinal borders. If such a border is actually in use for dividing the airspace, it will be referred to as an active border, otherwise it will be referred to as an inactive border. If a sector does not contain any inactive borders, it will be called an elementary sector. A certain combination of active borders will be referred to as a sector configuration. A theoretical combination of one or more elementary sectors will be simply referred to as a sector, while sectors that are present in a specifed sector confguration will be referred to as active sectors. In general, sectors can assume the "split" (contains more than one active sector), "armed" (used on its own) or "merged" (part of an active sector) state.

Making and executing decisions related to sector configuration change is done by a designated person, known as the supervisor. The supervisor is responsible for assigning ATCOs to newly activated sectors, replacing personnel in already active sectors and above all for making decisions about when to change sector configuration and which should be the new configuration to use.

In a general airspace, a great amount of possible sector configurations may exist and different sector configurations usually mean a different distribution of traffic among ATCOs. The basic aim of sectorization is to keep the workload of ATCOs near the optimum level which means that they should neither be assigned too difficult nor too simple traffic situations as both can lead to higher error rates [1, 2]. Since workload is not only influenced by traffic volume (i.e. the number of aircraft) but also by traffic complexity (and other complexity factors related to airspace, equipment, operators etc.)[3], finding the optimal sector configuration for a certain traffic situation can not be done via a simple algorithm based on the number of aircraft in sectors. Instead, the optimal sector configuration can be interpreted as a product of a complex function based on the above mentioned complexity factors. Note that many Air Navigation Service Providers use the number of aircraft to determine a suitable basic sector configuration. Supervisors can then change this basic configuration taking into account other factors based on their experience.

In ATC centers, the above function is realized by the supervisor. Due to the complexity of this decision and the fact that supervisors may differ in experience, skills and/or decision-making preferences, the actual sector configuration produced depends on the supervisor. Therefore, it is prone to human error, especially in case of uncommon traffic situations. In order to increase the probability of using safe and efficient sector configurations, it is advisable to provide an automated advisory tool that can suggest sector configuration solutions to the supervisor who can decide to approve or reject them. In this paper, we provide an overview of the

functional requirements of a tool that fulfils the mentioned purpose by presenting a model to formally describe the decision making process (Section 2), providing a more detailed description about the tool's modules which are based on this model (Sections 3 and 4) and drawing conclusions (Section 5).

2. Model of the decision making process

As it was adumbrated in the previous section, the tool needs to use the same data that is available to supervisors and produce a sector configuration as an output. In other words, it should mimic the decision making mechanism of a human supervisor.

The first step in the decision making process is the collection and evaluation of different data that can be relevant to ATCO workload and thus sector configuration. The data used by supervisors as input for their decisions can originate from any of the following sources:

- Radar data (real time position, flight level and speed vector of aircraft)
- Flight plan data (scheduled route of aircraft)
- Airspace restriction data (due to thunderstorm or scheduled restrictions)

In order to create an appropriate model of the data evaluation process, it is important to consider the timely nature of the supervisor's decision. When making their decisions, supervisors are aware that performing a sector configuration change requires a certain amount of time, because ATCOs have to get familiar with the traffic situation before assuming control and newly assigned ATCOs even have to man their workstations first. In real life situations, this procedure requires 10-20 minutes to complete. Such an amount of time is enough for considerable changes to happen in the characteristics of air traffic. This means that the supervisor can not simply rely on the data that is valid at the time of the decision but should make projections and create a mental picture of traffic expected in 10-20 minutes and then use this for actual decision making.

Formalizing the above, let D_R, D_{FP} and D_{TRA} be sets of radar data, flight plan data and temporary airspace restriction data respectively, which are available at the moment of the decision, although D_{FP} and D_{TRA} contain information about the future. In the model of the decision making process, D_R and D_{FP} are used first by a projection function (F_P) that creates a set of projected radar data (D_R^P) whith a structure similar to D_R but it is valid at the time of configuration change.

$$F_P : D_R \times D_{FP} \to D_R^P \tag{2.1}$$

After projecting future traffic based on actual aircraft movement and flight plans, the supervisor has to update the mental picture by considering areas of the airspace that are restricted to flight by thunderstorm or scheduled restrictions by other parties using the airspace. This correction is modelled by the corrected projection function (F_{CP}) which transforms the set of projected radar data into

corrected projected radar data (D_R^{PC}) by using a set of data about temporary airspace restrictions (D_{TRA}).

$$F_{CP} : D_R^P \times D_{TRA} \rightarrow D_R^{PC} \tag{2.2}$$

Once the mental picture of the (corrected) projected situation is available, the supervisor analyzes it by evaluating the characteristics of air traffic and the airspace. In the model, these characteristics are represented by so called complexity factors. Complexity factors each describe a certain attribute of traffic (or airspace structure) by a numeric value. The set of complexity factors (C) can contain different parameters such as the number of descending aircraft or the number of aircraft pairs on conflicting routes (further examples can be found in [3, 6, 7]). Set C turns out as the product of the complexity calculation function (F_C) which transforms corrected projected radar data into complexity factors.

$$F_C : D_R^{PC} \rightarrow C \tag{2.3}$$

When the supervisor has successfully evaluated the projected traffic situation, the most complex (and thus the most difficult to model) stage of the decision making process takes place. In this stage, the supervisor tries to figure out which would be the optimal sector configuration by taking all the complexity factors into consideration simultaneously with different "weights" depending on their significance. In the model, this process is represented by two successively executed functions. The first function is called the sector state function (F_{SS}) and it is responsible for assigning a set of sector states (S) to the sectors based on the set of complexity factors. S contains the state (i.e. "split", "armed" or "merged") that is considered optimal for each sector under the given circumstances.

$$F_{SS} : C \rightarrow S \tag{2.4}$$

The second function is the sector configuration function (F_{SC}) which transforms the sector state set produced by F_{SS} into a sector configuration which can be interpreted as a corrected set of sector states (S^C).

$$F_{SC} : S \rightarrow S^C \tag{2.5}$$

Including F_{SC} in the model is necessary because F_{SS} models a highly complex function and thereby it is likely to produce some erroneous states in S. Errors in S can mean that there is no applicable sector configuration in which every sector would have the state considered optimal. This can happen for example if two partially overlapping sectors are both assigned "armed" as their optimal state. These errors can be corrected by F_{SC} if it applies certain restrictions about possible sector configurations and rules to handle inconsistent sector states.

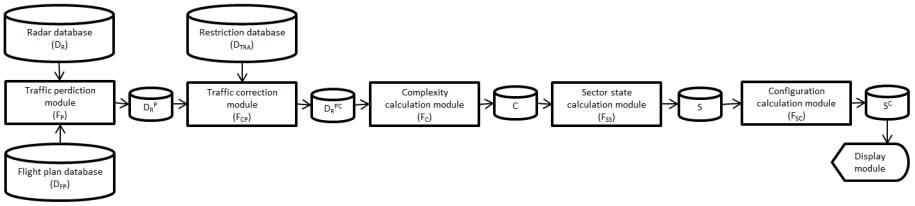


Figure 1: Modules of the decision support tool

3. Creating and correcting projected radar data

Projected radar data is created by using actual radar data and flight plan data in a similar manner as supervisors would use such data types. Therefore, it has to be understood how supervisors create their mental picture about future traffic in reality. When it comes to predicting future position, flight level and speed, aircraft can be classified into the following three categories:

1. Approaching aircraft outside the airspace with radar data only
2. Approaching aircraft outside the airspace with radar data and flight plan data synchronized
3. Aircraft already in the airspace

The future parameter values of aircraft in the first category can only be calculated by using actual radar data, because there is no information about the expected changes in direction, flight level or speed. A radar data record for a particular aircraft in a specified point in time contains information about the geographic distance from a specified point in two perpendicular directions (integer x and y), the speed vector's coordinates in the same directions (integer v_x and v_y) and flight level (integer l). These data can be used by the tool for the simple projection of the aircraft's position by implementing a function that uses the following equation (with t_0 representing the time belonging to the data record and t_P representing the time of the predicted situation 15 minutes after t_0):

$$x_{t_P} = x_{t_0} + v_x(t_P - t_0), y_{t_P} = y_{t_0} + v_y(t_P - t_0) \tag{3.1}$$

In case of aircraft in the second and third category, data prediction is primly based on flight plan data, because it provides more information about an aircraft's expected future characteristics. The flight plan of a particular aircraft contains the identifiers of the next n waypoints to cross ($W_1..W_n$) along with the expected speeds ($v_{W_1}..v_{W_n}$) and expected flight levels ($l_{W_1}..l_{W_n}$) upon crossing each waypoint. Besides these data, we can also assume that the tool has access to the geographic coordinates of the waypoints ($x_{W_1}..x_{W_n}$; $y_{W_1}..y_{W_n}$).

Based on the cruising speeds and the distance between subsequent waypoints, the tool can calculate the expected time of crossing each waypoint ($t_{W_1}..t_{W_n}$) and

thus determine which will be the last (W_i) and next (W_j) waypoint of the aircraft at t_P. If this information is available, the aircraft's expected position at t_P can be obtained from the following equations where $\hat{v}(t_P)$ is the average speed between t_{W_i} and t_P supposing that the aircraft changes speed from v_{W_i} to v_{W_j} by constant acceleration:

$$\alpha = \arctan \frac{y_{W_j} - y_{W_i}}{x_{W_j} - x_{W_i}} \tag{3.2}$$

$$\hat{v}(t_P) = v_{W_i} + \frac{\frac{v_{W_j} - v_{W_i}}{t_{W_j} - t_{W_i}}(t_P - t_{W_i})}{2} \tag{3.3}$$

$$x(t_P) = x_{W_i} + \hat{v}(t_P)\cos\alpha(t_P - t_{W_i}), y(t_P) = y_{W_i} + \hat{v}(t_P)\sin\alpha(t_P - t_{W_i}) \tag{3.4}$$

Once the radar data is produced for t_P, it also has to be produced for time $t_P^{'}$ which represents a moment in time 10 seconds after t_P. This is necessary because some of the complexity factors' values can only be produced by functions that require information about the vertical dynamics of traffic which can only be obtained from the comparison of flight levels for the same aircraft at two different points in time.

The correction of projected radar data is required if certain sections of the airspace are restricted to the aircraft controlled by the ATC center in scope. The restriction can be planned if airspace is in use by other parties (e.g. military) in a predefined time and manner or unplanned if thunderstorm activity is present making the section unsafe to fly through. In this phase of designing the advisory tool, the two types of restriction will be handled by using the same algorithm.

In a real life air traffic situation, aircraft avoid entering restricted airspace sections by flying a modified route planned by the ATCO. When the supervisor makes a prediction about the future traffic situation, he or she has to take such modified routes into consideration and so does the tool. In order to achieve this, the tool has to implement an algorithm that simulates avoidance of restricted airspaces by modifying flight plan data. This algorithm would first decide whether a given aircraft is expected to cross a restricted section if it flies according to its initial flight plan data. This can be decided by analyzing whether any of the expected route's upcoming sections intersects with the restricted airspace section (modelled as a polygon). If an intersecting section is found, the given aircraft's flight plan data set has to be extended by replacing the intersecting section with multiple sections that do not intersect with the polygon. Calculating coordinates of the replacment sections can either be done by using an off-the-shelf model (like the geometric recourse model in [4] or the dynamic rerouting model in [5]) or by developing a purpose-made algorithm which is outside the scope of this paper. Once the additional sections are added, the aircraft's projected radar data has to be calculated by using the newly modified flight plan data.

4. Calculating sector states and –configuration

When projected radar data is available for a future situation, it is used by the tool's complexity calculation module for producing the actual values of complexity parameters. Complexity parameter values are calculated by applying simple geometric functions which are described in [6] along with the set of complexity factors planned to be used by the tool. Complexity calculations have to be performed for the whole airspace as well as each sector inside it given that they can be used as active sectors in a practical sector configuration.

Complexity values of each sector are then passed on to the tool's central logic module as real numbers. The central logic module's main function uses neural network based estimation to produce the optimal state for each sector. To make this function applicable, training of the neural network has to be performed before starting to use the tool in order to obtain the values of the network's weight parameters. These weight parameters have to be made accessible to the tool in a database and they should be continuously modified in accordance with the user's feedback. A more detailed description about applying neural network logic for sector state estimation – including the training process of the networks – can be seen in [7].

The trained neural network's function (F_{SS}) transforms a vector of complexity factors ($\mathbf{c}$) into a sector state matrix (S). The rows of S each represent a sector while the columns represent possible sector states (split, armed and merged). As an example, in case of the Hungarian airspace where there are 31 usable sectors, S is a 31x3 matrix. S consists of real numbers between 0 and 1 with each number providing information about how close the given state is to the optimal one in case of the given sector.

$$F_{SS} : c \rightarrow S \tag{4.1}$$

Getting the optimal sector configuration requires turning the sector state matrix (S) into a sector border matrix (B) via function F_SS which is responsible for eliminating errors from the state matrix.

$$F_{SC} : S \rightarrow B \tag{4.2}$$

In case of the Hungarian airspace, B is a 2x4 matrix with the two lines representing the east ('E') and west ('W') sectors of the airspace while the columns represent the altitudinal borders in E and W. Elements of the matrix assume the value 1 if the represented border is active in the configuration and 0 otherwise. F_{SC} is only executed by the tool if the airspace has to be split according to S, otherwise the airspace itself will be the optimal configuration. F_{SC} is represented in the tool's logic as a function that iterates through the lines of S and compares the split, armed and merged values in each line. Based on the comparison results, it modifies the elements of B from 0 to 1 in accordance with the following algorithm:

1. If the 'E' (or 'W') sector's state with the highest value is not the split state, 'E' (or 'W') should be used as an armed sector.

2. If an elementary sector's armed state has a higher value than its merged state, it should be armed, so the values representing its lower and upper border in B should be set to 1.
3. If a non-elementary sector's split value is higher than the armed and merged value, it has to be split. Sectors that have to be split are evaluated in the following steps, which are repeated until none of the conditions are fulfilled and no additional split operations are necessary.
4. If a sector contains 4 borders, it has to be split at the border with the highest average split value (i.e. the average split value of all sectors that contain the given border), so the given border's value in B should be set to 1.
5. If a sector contains 2 or 3 borders, it has to be split at the border with the highest average split value but only if it exceeds 0,5.
6. If a sector contains exactly 1 border, it has to be split if the border's average split value exceeds 0,8.

When B has been created with the above algorithm, the configuration has to be displayed visually to the supervisor as graphical and/or textual information.

5. Conclusion

A decision support tool that can suggest sector configurations to supervisors would be useful for enhancing ATC safety. Developing such a tool requires creating a functional model of the real decision making process and implementing the functions of this model. The latter can not be done without solving such problems as the lack of flight plan data for certain aircraft, the presence of restricted airspace sections or the errors in sector states calculated by neural networks. Thes can be solved via different algorithms presented (or referred) in this paper.

Due to the simplifications and ommitances in the design of the model and its functions, the dependability of the tool's suggestions can not be guaranteed in the first phase of usage. In order to subdue this issue and continuously improve dependability, the tool should contain a feedback function (by requesting direct feedback from the supervisor or simply monitoring the actual configurations used) through which it can modify its own parameters (e.g. weights of the neural network). Details of the feedback function are expected to be presented in later works.

References

[1] Rodgers, M.D., Mogford, R.H., Mogford, L.S., The relationship of sector characteristics to operational errors, *FAA Aviation Medicine Report* Vol. 98/14 (1998)

[2] Stager, P., Hameluck, D., Ergonomics in air traffic control, *Ergonomics* Vol. 33(4) (1990), 493–499.

[3] Gianazza, D., Guittet, K., Evaluation of air traffic complexity metrics using neural networks and sector status, *Proceedings of the 2nd International Conference on Research in Air Transportation, ICRAT 2006*

[4] Yoon, Y., Hansen, M., Ball, M. O., Optimal route decision with a geometric ground-airborne hybrid model under weather uncertainty, *Transportation Research Part E* Vol. 48 (2012), 34–49.

[5] Mukherjee, A., Hansen, M., A dynamic rerouting model for air traffic flow management, *Transportation Research Part B* Vol. 43 (2009), 159–171.

[6] Számel, B., Szabó, G., Towards safer air traffic: Optimizing ATC controller workload by simulation with reduced set of parameters„ *Safety and Reliability: Methodology and Applications: ESREL2014* (2014), 979–987.

[7] Számel, B., Mudra, I., Szabó, G., Applying Airspace Capacity Estimation Models to the Airspace of Hungary, *Periodica Polytechnica: Transportation Engineering* Vol. 43(3) (2015), 120–128.

Annales Mathematicae et Informaticae
***48** (2018) pp. 75–82*
http://ami.uni-eszterhazy.hu

Finding frequent closed itemsets with an extended version of the Eclat algorithm

Laszlo Szathmary

University of Debrecen, Faculty of Informatics, Department of IT
H-4002 Debrecen, Pf. 400, Hungary
szathmary.laszlo@inf.unideb.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Apriori is the most well-known algorithm for finding frequent itemsets (FIs) in a dataset. For generating interesting association rules, we also need the so-called frequent closed itemsets (FCIs) that form a subset of FIs. *Apriori* has a simple extension called *Apriori-Close* that can filter FCIs among FIs. However, it is known that vertical itemset mining algorithms outperform the *Apriori*-like levelwise algorithms. *Eclat* is another well-known vertical miner that can produce the same output as *Apriori*, i.e. it also finds the FIs in a dataset. Here we propose an extension of *Eclat*, called *Eclat-Close* that can filter FCIs among FIs. This way *Eclat-Close* can be used as an alternative of *Apriori-Close*. Experimental results show that *Eclat-Close* performs much better than *Apriori-Close*, especially on dense, highly-correlated datasets.

Keywords: data mining, frequent itemsets, association rules, algorithms

MSC: 97R40

1. Introduction

In data mining, frequent itemsets (FIs) and association rules play an important role [1]. Due to the high number of patterns, various concise representations of FIs have been proposed, of which the most well-known representations are the frequent generators (FGs) and the frequent closed itemsets (FCIs) [2, 3]. There are a number of methods in the literature that target FCIs and/or FGs, but most of these

algorithms are levelwise methods [4, 5]. It is known that depth-first algorithms usually outperform their levelwise competitors.

In this paper, we present an algorithm called *Eclat-Close*, which is a single-pass, depth-first, vertical FI+FCI miner. The approach behind *Eclat-Close* is derived from the one used in the *Eclat* [6] vertical miner. *Eclat* outputs the entire family of FIs hence what we needed to design was a mechanism to recognize FCIs among FIs. The task performed by *Eclat-Close* can be described as the computation of frequent equivalence classes. The nice surprise with *Eclat-Close* came when we measured its performance. Despite its relatively low level of optimization, our algorithm systematically outperformed its levelwise competitor, *Apriori-Close*.

The remainder of the paper is organized as follows. Basic concepts are provided in Section 2. Section 3 presents the *Apriori-Close* algorithm, which is our levelwise competitor. Section 4 introduces *Eclat-Close*, which is the main contribution of the paper. Experimental results are provided in Section 5. Finally, Section 6 concludes the paper.

2. Basic Concepts

The following 5×5 sample dataset: $\mathcal{D} = \{(1,\ ABDE),\ (2,\ AC),\ (3,\ ABCE),\ (4,\ BCE),\ (5,\ ABCE)\}$ will be used as a running example. Henceforth, we refer to it as dataset $\boldsymbol{\mathcal{D}}$.

We consider a set of *objects* or *transactions* $\mathcal{O} = \{o_1, o_2, \ldots, o_m\}$, a set of *attributes* or *items* $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, and a relation $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$. A set of items is called an *itemset*. Each transaction has a unique identifier (*tid*), and a set of transactions is called a *tidset*. The tidset of all transactions sharing a given itemset X is its *image*, denoted by $t(X)$. For instance, the image of $\{A, B\}$ in $\mathcal{D}$ is $\{1, 3, 5\}$, i.e., $t(AB) = 135$ in our separator-free set notation. The *length* of an itemset X is $|X|$, whereas an itemset of length i is called an i-itemset. The (absolute) *support* of an itemset X, denoted by $supp(X)$, is the size of its image, i.e. $supp(X) = |t(X)|$. An itemset X is called *frequent*, if its support is not less than a given *minimum support* (denoted by min_supp), i.e. $supp(X) \geq min\_supp$. An equivalence relation is induced by t on the power-set of items $\wp(\mathcal{A})$: equivalent itemsets share the same image ($X \cong Z$ iff $t(X) = t(Z)$). Consider the equivalence class of X, denoted $[X]$. The equivalence class $[X]$ has a unique maximum w.r.t. set inclusion (a *closed* itemset).

Definition 2.1. An itemset X is *closed* if it has no proper superset with the same support.

A *closure* operator underlies the set of closed itemsets; it assigns to X the maximum of $[X]$ (denoted by $\gamma(X)$). Naturally, $X = \gamma(X)$ for closed X. For instance, in our dataset $\mathcal{D}$, the closure of B is BE, while the closure of BC is BCE.

3. Levelwise Exploration with Apriori-Close

The most well-known levelwise algorithm, without doubt, is *Apriori* [7]. This algorithm addresses the problem of finding all frequent itemsets in a dataset. *Apriori* has been followed by lots of variations, and several of these levelwise algorithms concentrate on a special subset of frequent itemsets, like closed itemsets or generators. Mannila and Toivonen provided a general framework for levelwise algorithms in [8]. The levelwise algorithm for finding all FIs is a breadth-first, bottom-up algorithm, which means the following. First it finds all 1-long frequent itemsets[1], then at each i^{th} iteration it identifies all i-long frequent itemsets. The algorithm stops when it has identified the largest frequent itemset. Frequent itemsets are computed iteratively, in ascending order by their length. At each iteration one database pass is needed to count support values, thus the number of database passes is equal to the length of the largest frequent itemset. This approach is very simple and efficient for sparse, weakly correlated data. The levelwise algorithm is based on two basic properties.

Property 3.1 (downward closure)**.** *All subsets of a frequent itemset are frequent.*[2]

Property 3.2 (anti-monotonocity)**.** *All supersets of a non-frequent itemset are non-frequent.*

Apriori-Close was proposed in [9]. This algorithm is an extension of *Apriori* and it can identify not only frequent, but *frequent closed itemsets* too simultaneously. The idea is the following. By definition, a closed itemset has no proper superset with the same support. At each i^{th} step all i-long frequent itemsets are marked as "closed". At the next $(i+1)^{th}$ iteration for each $(i+1)$-long itemset we test if it has an i-long subset with the same support. If so, then the i-long frequent itemset is not a closed itemset and we mark it as "not closed". When the algorithm terminates with the enumeration of all frequent itemsets, the itemsets still marked as "closed" are the frequent closed itemsets of the dataset. Experiments show that this kind of filtering of closed itemsets does not induce any serious additional computation time.

4. The Eclat-Close Algorithm

In this section we present the *Eclat* algorithm [6], which serves as a basis for *Eclat-Close*. *Eclat* can only find FIs, while *Eclat-Close* makes it possible to filter FCIs among FIs. *Eclat-Close* is our extension and this section is the main contribution of the paper.

[1] That is, first it identifies all frequent items (attributes).

[2] The name of the property comes from the fact that the set of frequent itemsets is closed w.r.t. set inclusion.

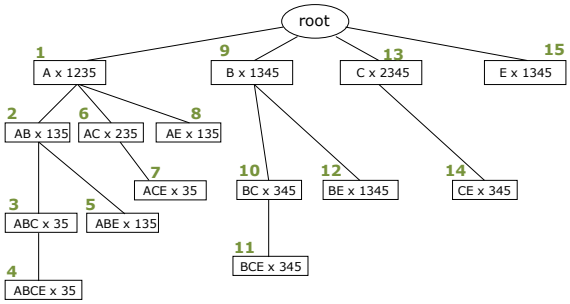


Figure 1: IT-tree: Itemset-Tidset search tree of dataset $\mathcal{D}$ with $min\_supp = 2$

4.1. Eclat

Eclat was the first FI-miner using a vertical encoding of the database combined with a depth-first traversal of the search space (organized in a prefix-tree) [6].

Vertical miners rely on a specific layout of the database that presents it in an item-based, instead of a transaction-based, fashion. Thus, an additional effort is required to transpose the global data matrix in a pre-processing step. However, this effort pays back since afterwards the secondary storage does not need to be accessed anymore. Indeed, the support of an itemset can be computed by explicitly constructing its tidset which in turn can be built on top of the tidsets of the individual items. Moreover, in [10], it is shown that the support of any k-itemset can be determined by intersecting the tid-lists of any two of its $(k-1)$-long subsets.

The central data structure in a vertical FI-miner is the IT-tree that represents both the search space and the final result. The IT-tree is an extended prefix-tree whose nodes are $X \times t(X)$ pairs. With respect to a classical prefix-tree or trie, in an IT-tree the itemset X provides the entire prefix from the root to the node labeled by it (and not the difference with the parent node prefix).

Example. Figure 1 presents the IT-tree of our example. The traversal order is indicated above the nodes. Observe that the node $ABC \times 35$ for instance can be computed by combining the nodes $AB \times 135$ and $AC \times 235$. To that end, tidsets are intersected and itemsets are joined. The support of ABC is readily established to 2.

4.2. Eclat-Close

In this subsection we present the *Eclat-Close* algorithm in detail. As mentioned before, *Eclat-Close* is based on *Eclat*. *Eclat-Close* traverses the IT-tree in a pre-order way, from left to right (see Figure 1), and it filters FCIs while extracting FIs from a dataset. The output of *Eclat-Close* is the list of frequent equivalence classes (see Table 1).

tidset	eq. class members (optional)	closure	support
1235	A	A	4
135	AB, ABE, AE	ABE	3
35	ABC, $ABCE$, ACE	$ABCE$	2
235	AC	AC	3
1345	B, BE, E	BE	4
345	BC, BCE, CE	BCE	3
2345	C	C	4

Table 1: *Eclat-Close* builds this table, which is actually a hash table. The key is a tidset and the value is a row

Eclat-Close builds a hash table, as depicted in Table 1. The key of the hash is a tidset, while the value of the hash is a row object. A row object represents an equivalence class and it has the following fields: **(1)** tidset (by definition all itemsets in an equivalence class have the same tidset), **(2)** equivalence class members, **(3)** closure (the largest element in an equivalence class; this is a unique element), and **(4)** support (this is the cardinality of the tidset).

The algorithm works the following way. When a new FI is found in the IT-tree, it is tested if it belongs to an already discovered equivalence class, i.e. we test if its tidset is in the hash. If it is not present in the hash, then it belongs to a new equivalence class, thus a new row is added to the hash. If its tidset is in the hash, then the following steps are performed. First, the itemset is added to the row's list of equivalence class members. Second, the itemset is added to the row's closure using a union operation.

Example. *Eclat-Close* builds a hash table, as depicted in Table 1. A row object represents an equivalence class. The algorithm starts enumerating the 15 FIs of $\mathcal{D}$ using the traversal strategy of *Eclat* (as seen in Figure 1). The first node is $A \times 1235$. The tidset 1235 is not yet in the hash, thus a new row is added in the hash table (tidset: 1235; eq. class members: A; closure: A; support 4). The nodes $AB \times 135$ and $ABC \times 35$ are also added as new rows. The next FI is $ABCE \times 35$, but its tidset is an existing key in the hash. Let r denote the row whose tidset is 35. $ABCE$ is added to r's "eq. class members" and "closure" fields. The "closure" column is the union of its former value ABC and $ABCE$, which yields $ABCE$. The end result is shown in Table 1.

When the algorithm stops, the itemsets in the "closure" field are completed, i.e. they represent the closures of the equivalence classes. If we are only interested in FCIs, the column "eq. class members" can be omitted. This way *Eclat-Close* can be used as a pure FCI-miner algorithm. The pseudo code of *Eclat-Close* is provided in Algorithm 1.

Algorithm 1 (pseudo code of Eclat-Close):

hashTable: the table structure (as seen in Table 1)

```
1)  start the Eclat algorithm and assign the current node to the variable curr
2)  {
3)      if curr.tidset not in hashTable:
4)          row.tidset ← curr.tidset
5)          row.eq_class_members ← curr.itemset   // optional
6)          row.closure ← curr.itemset
7)          row.support ← cardinality(row.tidset)
8)          hashTable.add(row)
9)      else:
10)         row ← hashTable.get(curr.tidset)
11)         row.eq_class_members.add(curr.itemset)   // optional
12)         row.closure ← row.closure ∪ curr.itemset
13) }
14) // hashTable is filled; it contains all the frequent equivalence classes
```

5. Experimental Results

In our experiments, we compared *Eclat-Close* with *Apriori-Close* and *Charm* [11]. *Apriori-Close* was presented in Section 3. *Charm* is a very efficient vertical algorithm, also based on *Eclat*. *Charm* reduces the search space to the minimum, i.e. it explores FCIs only. Since *Charm* is a state-of-the-art algorithm, we also compare *Eclat-Close* against it. All three algorithms were implemented in Java. The experiments were carried out on an Intel Quad Core i5-2500 3.3 GHz machine running under Manjaro GNU/Linux with 4 GB RAM. All times reported are real, wall clock times as obtained from the Unix *time* command between input and output. For the experiments we have used the following datasets: T20I6D100K, T25I10D10K, C20D10K, C73D10K and MUSHROOMS. The T20 and T25[3] are sparse datasets, constructed according to the properties of market basket data that are typical weakly correlated data. The C20 and C73 are census datasets from the PUMS sample file, while the MUSHROOMS[4] describes mushrooms characteristics. The last three are highly correlated datasets.

The execution times of the algorithms are illustrated in Table 2. The table also shows the number of FIs and FCIs. *Apriori-Close* finds all FIs and filters FCIs. *Eclat-Close* does the same thing, but in a vertical, depth-first fashion. *Charm* is similar to *Eclat*, but it reduces the search space to FCIs only.

The experiments show that *Eclat-Close* outperforms *Apriori-Close* on all datasets. The difference is especially spectacular on dense datasets. *Eclat-Close* performs very similarly to *Charm*, though it explores a much larger search space. It

[3] `http://www.almaden.ibm.com/software/quest/Resources/`
[4] `http://kdd.ics.uci.edu/`

min_supp	execution time (sec.)			# FIs	# FCIs
	Apriori-Close	Eclat-Close	Charm		
T20I6D100K					
10%	1.30	0.87	0.69	7	7
0.75%	10.15	1.01	2.70	4,710	4,710
0.50%	16.98	1.44	3.64	26,836	26,208
0.25%	43.03	4.62	7.38	155,163	149,217
T25I10D10K					
10%	0.42	0.31	0.27	20	20
0.75%	2.97	0.47	0.79	17,073	7,841
0.50%	16.54	1.04	1.30	302,284	52,033
C20D10K					
30%	7.90	0.31	0.29	5,319	951
20%	19.82	0.35	0.34	20,239	2,519
10%	48.58	0.55	0.47	89,883	8,777
5%	106.75	0.77	0.65	352,611	21,213
C73D10K					
95%	10.09	0.58	0.37	1,007	93
90%	144.64	0.63	0.43	13,463	942
85%	440.65	0.70	0.53	46,575	2,359
MUSHROOMS					
40%	1.00	0.29	0.27	505	124
30%	2.83	0.32	0.28	2,587	425
15%	45.20	0.43	0.34	99,079	2,210
10%	184.84	0.76	0.40	600,817	4,850

Table 2: Response times of Eclat-Close

can be due to the fact that *Charm* needs to apply several tests on an itemset to decide if it is closed. These extra tests give some overhead to *Charm*.

As a conclusion, we can say that *Eclat-Close* performs much better than its levelwise competitor *Apriori-Close*, and it is comparable to *Charm*.

6. Conclusion

In this paper we presented a vertical, depth-first algorithm that finds frequent equivalence classes, i.e. it explores FIs and filters FCIs among them.

When testing the performance of *Eclat-Close* w.r.t. efficient comparable alternatives from the literature, it came out that our algorithm performed surprisingly well for its level of optimization. We tend to see that fact as an indication that any improvement that avoids exploring the entire family of FIs has good chances of becoming the best performing algorithm in its class.

Currently, we only concentrated on FCIs among FIs, but it would be interesting to filter frequent generators as well. Having the generators in an equivalence class

as well, we could produce interesting association rules easily. We plan to extend *Eclat-Close* in this direction.

References

[1] Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: Proceedings of the 20th International Conference on Very Large Data Bases (VLDB '94), San Francisco, CA, Morgan Kaufmann (1994) 487–499

[2] Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., Lakhal, L.: Mining Minimal Non-Redundant Association Rules Using Frequent Closed Itemsets. In: Proceedings of the Computational Logic (CL '00). Volume 1861 of LNAI., Springer (2000) 972–986

[3] Kryszkiewicz, M.: Concise Representations of Association Rules. In: Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery. (2002) 92–109

[4] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering Frequent Closed Itemsets for Association Rules. In: Proceedings of the 7th International Conference on Database Theory (ICDT '99), Jerusalem, Israel (1999) 398–416

[5] Stumme, G., Taouil, R., Bastide, Y., Pasquier, N., Lakhal, L.: Computing Iceberg Concept Lattices with Titanic. Data and Knowl. Eng. **42**(2) (2002) 189–222

[6] Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W.: New Algorithms for Fast Discovery of Association Rules. In: Proceedings of the 3rd International Conference on Knowledge Discovery in Databases. (August 1997) 283–286

[7] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in knowledge discovery and data mining. American Association for Artificial Intelligence (1996) 307–328

[8] Mannila, H., Toivonen, H.: Levelwise Search and Borders of Theories in Knowledge Discovery. Data Mining and Knowledge Discovery **1**(3) (1997) 241–258

[9] Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Closed set based discovery of small covers for association rules. In: Proceedings 15emes Journees Bases de Donnees Avancees (BDA). (1999) 361–381

[10] Zaki, M.J.: Scalable Algorithms for Association Mining. IEEE Transactions on Knowledge and Data Engineering **12**(3) (2000) 372–390

[11] Zaki, M.J., Hsiao, C.J.: CHARM: An Efficient Algorithm for Closed Itemset Mining. In: SIAM International Conference on Data Mining (SDM' 02). (Apr 2002) 33–43

Annales Mathematicae et Informaticae
***48** (2018) pp. 83–91*
http://ami.uni-eszterhazy.hu

Generating Distance Fields from Parametric Plane Curves

Gábor Valasek

Eötvös Loránd University
valasek@inf.elte.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Distance fields have been presented as a general representation for both curves and surfaces [4]. Using space partitioning, adaptive distance fields (ADF) found their way into various applications, such as real-time font rendering [5].

Computing approximate distance fields for implicit representations and mesh objects received much attention. Parametric curves and surfaces, however, are usually not part of the discussion directly. There are several algorithms that can be used for their conversion into distance fields. However, most of these are based converting parametric representations to piecewise linear approximations [7].

This paper presents two algorithms to directly compute distance fields from arbitrary parametric plane curves. One method is based on the rasterization of general parametric curves, followed by a distance propagation using fast marching. The second proposed algorithm uses the differential geometric properties of the curve to generate simple geometric proxies, segments of osculating circles, that are used to approximate the distance from the original curve.

Keywords: Computer Graphics, Distance Fields, Geometric Modeling

MSC: 65D17, 65D18, 68U07

1. Introduction

Distance fields are samples of the signed or unsigned distance function to a geometric object. They are a versatile representation of curves and surfaces and have been

successfully applied in many areas of geometric modeling and computer graphics. Distance fields can simplify otherwise sophisticated operations, such as morphing between geometries with different topologies [3], and also speed up costly queries like collision detection.

Computation of distance fields from implicit representations has been extensively studied in the literature [7], but direct conversion of parametric curves or surfaces is less often exposed.

This paper presents two methods to convert parametric plane curves to sampled distance fields over a uniform grid.

The first method discretizes the parametric curve to grid cells. These cells are then used to form the boundary condition of the eikonal equation that defines the signed distance function of the parametric curve. Solving this equation using fast marching results in a highly efficient, approximate solution to the distance field.

The second method uses osculating circle segments to create a piecewise circular approximation to the original curve and extracts gridpoint distances using these circle segments as proxies for the original curve. This approach provides simple means to control the trade-off between accuracy and speed of execution.

These two methods are related in that they consist of a sampling and an approximation step. In rasterization, we sample the continouos problem first and then compute an approximate solution. In the osculating circle scenario, first we approximate the original curve with continuous proxies and then sample grid distances from these entities.

2. Rasterization-based distance field generation

One of the key components in real-time computer graphics is the high speed rasterization of geometric primitives. There are highly efficient algorithms to rasterize line segments [1], conic sections, and even higher order algebraic curves [6]. Nevertheless, research on the rasterization of general parametric curves is less prominent in the literature and they are usually discussed only in terms of lower degree polynomials, e.g. cubic Bezier curves [2].

The first method proposed in this paper to compute the distance field of a parametric curve consists of two steps:

1. Rasterize the input curve onto the distance field grid
2. Propagate distances over the grid from the rasterized cells

Step 1 requires a robust, gap-free, as in 8-connected, discretization of the input curve $\mathbf{r}(t) : [0, 1] \to \mathbb{E}^3$. This operation is inherently representation-aware, that is, it needs to know the particular basis and control data by which $\mathbf{r}(t)$ is represented.

For the sake of simplicity, we assume we are given a uniform grid, that is, a collection of rectangles of equal width and heigh. This way, the grid can be considered as collection of cells accessed by two dimensional integer indices, i.e. $Grid = [0, 1, .., W - 1] \times [0, 1, .., H - 1]$, where $W, H \in \mathbb{N}$ are the number of colums

and rows, respectively. Rasterization should generate a gap-free subset $Rasters \subset Grid$.

Step 2 starts with the cells in $Rasters$ and sets the distances to zero for these. Then it proceeds by using the fast marching method to compute the distance of every cell from $Rasters$. It does not rely on any prior information about the original representation of the curve, all it requires is the set of rasterized cells and the grid onto which the distances should be propagated.

2.1. Rasterization

Let $\mathbf{r}(t) : [0,1] \to \mathbb{E}^2$ be a regular parametric curve, i.e. $|\mathbf{r}'(t)| \neq 0, t \in [0,1]$. Let $D > 0$ denote the common width and height of all cells, measured in the units of the space in which the curve is embeded in.

The proposed rasterization method relies on a naive, curve traversal approach: starting from the $t = 0$ parameter, let us increase the parameter value by some $\Delta t > 0$ such that we travel approximately D units along the curve, i.e. the arc length between t and $t + \Delta t$ is $\approx D$. Evaluate the curve at this new point, find the closest grid cell, and if it is connected to the last cell in $Rasters$, add its index to it as well. If the connection is broken, we have to decrease Δt until the new point on the curve is rasterized onto a cell that is connected to $Rasters$.

To find the required $\Delta t > 0$ change of parameter, we have to compute the inverse of the arc-length function. The arc-length function is defined as $s(t) = \int_0^t |\mathbf{r}'(x)|dx : [0,1] \to [0,L]$ and since $\mathbf{r}(t)$ is regular, $s(t)$ is strictly increasing, i.e. it has an inverse denoted by $t(l) = s^{-1}(l) : [0,L] \to [0,1]$.

The n-th derivative of the arc-length function can be expressed in terms of the derivatives of the curve up to n. More precisely, in the case of the first three derivatives, we have

$$
\begin{aligned}
s'(t) &= |\mathbf{r}'(t)| = (\mathbf{r}'(t) \cdot \mathbf{r}'(t))^{\frac{1}{2}} \\
s''(t) &= \left((\mathbf{r}'(t) \cdot \mathbf{r}'(t))^{\frac{1}{2}}\right)' = \frac{\mathbf{r}'(t) \cdot \mathbf{r}''(t)}{|\mathbf{r}'(t)|} \\
s'''(t) &= \left(\frac{\mathbf{r}'(t) \cdot \mathbf{r}''(t)}{|\mathbf{r}'(t)|}\right)' = \frac{\mathbf{r}''(t) \cdot \mathbf{r}''(t) + \mathbf{r}'(t) \cdot \mathbf{r}'''(t)}{|\mathbf{r}'(t)|} - \frac{(\mathbf{r}'(t) \cdot \mathbf{r}''(t))^2}{|\mathbf{r}'(t)|^3}
\end{aligned}
$$

The above expressions can be simplified by using the Frenet coordinates of the derivatives: let $\mathbf{t}, \mathbf{n} : [0,1] \to \mathbb{R}^2$ denote the unit tangent and normal vectors of the curve and let $x_i(t) = \mathbf{r}^{(i)}(t) \cdot \mathbf{t}(t)$ and $y_i(t) = \mathbf{r}^{(i)}(t) \cdot \mathbf{n}(t)$. It can be easily shown that

$$
s'(t) = x_1(t) \ , \quad s''(t) = x_2(t) \ , \quad s'''(t) = \frac{y_2^2(t)}{x_1(t)} + x_3(t) = \kappa^2(t) x_1(t)^3 + x_3(t) \ ,
$$

where $\kappa(t)$ denotes the curvature function of the curve.

Let us now omit the parameter of evaluation from the formulas. Using that the derivative of the inverse function can be written as

$$t' = (s^{-1})' = \frac{1}{(s' \circ s^{-1})}$$
$$t'' = (s^{-1})'' = -\frac{s'' \circ s^{-1}}{(s' \circ s^{-1})^3}$$
$$t''' = (s^{-1})''' = -\frac{s''' \circ s^{-1} \cdot (s' \circ s^{-1}) + 3(s'' \circ s^{-1})^2}{(s' \circ s^{-1})^5}$$

the derivatives of the inverse of the arc-length function can be expressed as

$$t' = \frac{1}{x_1} \ , \ t'' = -\frac{x_2}{x_1^3} \ , \ t''' = -\frac{(\kappa^2 x_1^3 + x_3)x_1 - 3x_2^2}{x_1^5} = -\frac{y_2^2 + x_1 x_3 - 3x_2^2}{x_1^5}$$

that is

$$t(l + \Delta l) \approx t(l) + \Delta l \frac{1}{x_1} - \frac{\Delta l^2}{2} \frac{x_2}{x_1^3} - \frac{\Delta l^3}{6} \frac{y_2^2 + x_3 x_1 - 3x_2^2}{x_1^5} - ...$$

is the Taylor expansion of $t = s^{-1}$ about $l \in [0, L]$.

We can compute the required Δt change of parameter to advance approximately D units along the curve by evaluating the Taylor expansion with D

$$\Delta t = t(D) = D \frac{1}{x_1} - \frac{D^2}{2} \frac{x_2}{x_1^3} - \frac{D^3}{6} \frac{y_2^2 + x_3 x_1 - 3x_2^2}{x_1^5} - ...$$

Note that this means that this method is only rasterizing grid cells through which the curve passes through at least D units long consequitively sans approximation error. As a result, generally, we do not rasterize every cell that the curve touches, but the total arc-length of the curve within the omitted cells is small, provided the inverse arc-length approximation is precise enough and D is selected properly.

In practice, we have to keep track of whether the truncation introduced such an error that creates a gap during rasterization. This can be verified by comparing the integer grid coordinates of the previous and the new rasters. Should a gap occur, we can use a simple binary search between t and $t + \Delta t$ to find the parameter that corresponds to a curve point that rasterizes onto a grid cell that does not break the connectivity of the rasters or, better yet, until the arc-length between the two points is indeed closer to D. Using the latter as a stop criteria may or may not be possible, depending on our computational and time constraints.

2.2. Distance propogation

The second step of the algorithm can be formulated as an eikonal equation: the rasterized cells, denoted by *Rasters*, form the zero set of the unknown signed

distance function $f(x, y)$ of the curve $\mathbf{r}(t)$, and we are looking for a solution over the entire *Grid* such that

$$|\nabla f(\mathbf{x})| = 1 \text{ for } \mathbf{x} \in Grid$$
$$f(\mathbf{x}) = 0 \text{ for } \mathbf{x} \in Rasters$$

This can be solved on the grid using fast marching in $O(N \log N)$ time [8], where N denotes the number of grids, but there are linear solvers for the problem [9].

3. Osculating cirlce-based distance field generation

The rasterization based method proves to be an efficient sequential algorithm, however, its accuracy, as shown in the test results section, may be lacking for certain applications. The method proposed in this section aims to improve on accuracy and provides a user-controllable parameter to balance between speed and precision.

The method constitutes of the following steps:

1. Adaptively sample the curve and compute osculating circles at samples
2. Select segments of the osculating circles to approximate the curve
3. Compute cell distances from osculating circle segments

Step 1 and 2 do a pre-processing of the input curve. The goal is to create a collection of circular segments as the approximation of the original curve from the osculating curcles. The number of osculating circles can be set by the user with or without taking into account the particular geometric complexity of the input curve. The more osculating circles there are, the smaller the mean and maximum error becomes, but the execution time is also increasing.

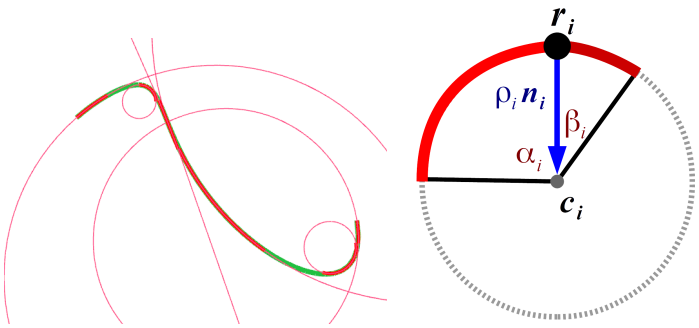
Step 3 speeds up distance computations by taking advantage of the piecewise circular structure of the curve approximant.

The following subsections discuss these three steps in detail.

3.1. Sampling the curve

In order to adapt the number of segments to the particular curve in question, one has to find a measure of complexity for parametric curves. Initially, we used the curvature function plot to determine this complexity scalar by counting the number of local extrema of the curvature function. The user could select the number of osculating circles we sample between these extrema.

However, our tests have shown that for the case of distance field computation, an equidistant parameter sampling of the curve generates roughly the same quality distance approximation as sampling the curve adapted to the curvature extrema. As a result, we suggest the use of equidistant parameter sampling in time critical situations. Figure 1a illustrates the osculating circles obtained by this process.



(a) Osculating circles. (b) Circular segment.

Figure 1: Osculating circles sampled along the curve (left). The representation of a single circular segment consists of $\alpha, \beta > 0$ angles, associatied with the osculating curve at sample $\mathbf{r}_i$ (right).

3.2. Determining the piecewise circular curve approximant

If all osculating circles are determined, we have to select segments on each, such that they put together form at least a globally G^0 approximation to the curve.

Let $t_0 = 0 < t_1 < t_2 < ... < t_N = 1$ denote the selected sampling parameters along the curve. Then $\mathbf{r}_i = \mathbf{r}(t_i)$ are the curve points and the unit tangent and normal vectors are $\mathbf{t}_i = [\mathbf{r}'(t_i)]_0$ and $\mathbf{n}_i = [(\mathbf{r}'(t_i) \times \mathbf{r}''(t_i)) \times \mathbf{r}'(t_i)]_0$, respectively, where $[\mathbf{a}]_0$ denotes normalization, i.e. $[\mathbf{a}]_0 = \frac{\mathbf{a}}{|\mathbf{a}|}$ if $\mathbf{a} \neq \mathbf{0}$.

Curvature at t_i is denoted by κ_i, their recipocals, i.e. the radii of the osculating circles are $\rho_i = \frac{1}{\kappa_i}$, and the centers of the osculating circles are $\mathbf{c}_i = \mathbf{r}_i + \rho_i \mathbf{n}_i$.

Since the circular segments have to include $\mathbf{r}_i$, we can define the segments used for approximation as two angles measured away from $\mathbf{n}_i$, the left angle α_i and the right angle β_i, see Figure 1b.

At the two endpoints we do not want to extend beyond the original curve, i.e. $\alpha_0 = 0$ and $\beta_N = 0$.

In case of a general point, consider the three osculating circles at t_{i-1}, t_i, t_{i+1}. If the $(i-1)$-th and i-th osculating circles intersect, let α_i be the angle that corresponds to the intersection point that is closer to $\mathbf{r}_i$ and still lies on the left of it, i.e. on the opposite direction of $\mathbf{t}_i$. If the circles are non-intersecting, or the intersection lies in the opposite direction, let us pick α_i such that it points to the point that is the closest to $\frac{\mathbf{r}_{i-1}+\mathbf{r}_i}{2}$. The β_i angle is determined similarly.

3.3. Distance computation

The reason for the gain in execution time lies in the fact that distance computation from a circular arc can be resolved by three point-point distance checks. We can use the distance of the point from the circular segment in the cone determined by $\mathbf{c}_i$ and the α_i, β_i spans, which is either $|\mathbf{x}-\mathbf{c}|-\rho_i$ or $|\mathbf{x}-\mathbf{c}|+\rho_i$, depending whether

$\mathbf{x} - \mathbf{c}$ forms and angle less than 90 degrees with $\mathbf{n}_i$. For points that are outside of this cone, we can check against the endpoints of the circular segments.

4. Test results

We benchmarked the execution time of both proposed algorithms and used a brute force, geometric Newton-Raphson based distance field as the ground truth for accuracy tests. A pure Python, sequential CPU implementation was used to compare the run times. After running a batch of 1000 random curves, we obtained the runtime and accuracy statistics for the distance field generation methods. The osculating circle based method used 7 osculating circles to approximate the curve.

	execution time (seconds)		error		
method	mean	std. dev.	max	mean	std. dev.
brute force	0.36	0.03	0	0	0
raster	**0.014**	**0.0029**	30.64%	21.53%	24.65%
osculating	0.16	0.036	**7.28%**	**0.46%**	**4.23%**

The errors are relative errors with respect to the distance values obtained from the brute force method. Both tests were conducted on 1000 random quntic Bézier curves and the distance transform was carried out on a 16×16 grid.

In a sequential environment, the rasterization based approach proved to be superior, however, the accuracy suffered greatly from the initial quantization of distance values, i.e. the setting of distances to zero for rasterized cells. This, however, is necessary for most fast marching algorithms.

The accuracy (measured as mean error) of the osculating circle based approach is two orders of magnitude better than that of the rasterization method.

Another focus of our tests was to determine how does the number of osculating circles correlate with the error statistics. The following table shows these as we increase the number of equidistantly sampled osculating circles. Interestingly, replacing the equidistant sampling strategy did not alter the error significantly, however, it is subject to future research to find computationally viable sample parameter selection methods that could increase distance field accuracy.

no. of circles	**mean error**	**std. dev.**
5	10.54%	88.86%
10	1.2%	11.95%
15	0.26%	1.79%
25	0.15%	1.4%
35	0.11%	1.08%

It is also important to note that creating a massively parallel implementation of the osculating circle based method is trivial, whereas the rasterization method suffers from the difficulties that arise when migrating fast marching to this setting, making it much harder to implement on e.g. GPUs.

Simple parallelization of the brute force method is also possible, however, one advantage of the osculating circle based method is that the actual distance transform part, where we measure the cell distances from the curve approximation, does not depend on the original representation of the curve, while the brute force algorithm has to rely on it throughout its execution.

5. Conclusion

This paper proposed two algorithms for the problem of direct conversion of parametric plane curves to distance fields. We compared these methods with each other and to a brute force distance field generation algorithm.

In sequential environments, the rasterization based method proved to be the fastest. However, if accuracy is required, the osculating circle based approach offers a superior solution. It relies on the prior knowledge of the representation of the curve, even though to a lesser extent than the brute force approach, which has to be taken into account when it comes to implementation.

References

[1] J. E. Bresenham: Algorithm for computer control of a digital plotter, IBM Systems Journal, Volume 4 Issue 1, March 1965, Pages 25-30

[2] S.-L. Chang, M. S. R. Rocchetti: Rendering cubic curves and surfaces with integer adaptive forward differencing, SIGGRAPH '89 Proceedings of the 16th annual conference on Computer graphics and interactive techniques, Pages 157 - 166, ISBN:0-89791-312-4

[3] D. Cohen-Or , D. Levin , A. Solomovici: Three-Dimensional Distance Field Metamorphosis, *ACM TRANSACTIONS ON GRAPHICS* 1998, Volume 17, Number 2, Pages 116–141

[4] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, Thouis R. Jones: Adaptively sampled distance fields: a general representation of shape for computer graphics, *Proceeding SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques* 2000, Pages 249-254

[5] C. Green: Improved Alpha-Tested Magnification for Vector Textures and Special Effects, *Proceeding SIGGRAPH '07 ACM SIGGRAPH 2007 courses*, 2007, pages 9–18.

[6] J. D. Hobby: Rasterization of nonparametric curves, ACM Transactions on Graphics, Volume 9 Issue 3, July 1990, Pages 262-277

[7] Mark W. Jones, J. Andreas Baerentzen, Milos Sramek: 3D Distance Fields: A Survey of Techniques and Applications, *IEEE Transactions on Visualization and Computer Graphics* , Volume 12 Issue 4, July 2006, Page 581-599

[8] J.A. Sethian: A Fast Marching Level Set Method for Monotonically Advancing Fronts, Proc. Nat. Acad. Sci., 93, 4, pp.1591-1595, 1996.

[9] L. Yatziv, A. Bartesaghi, G. Sapiro: O(N) implementation of the fast marching algorithm, Journal of Computational Physics, Volume 212, Issue 2, 1 March 2006, Pages 393-399