Periodica Polytechnica
Electrical Engineering and
Computer Science

x(y) pp. x-x, (year)
DOI: 10.3311/PPee.XXXX
Creative Commons Attribution
b

RESEARCH ARTICLE

3D Object Detection and Scene Optimization for Tangible Augmented Reality

Márton Szemenyei[1], Ferenc Vajda[1]

Received xx Xxxx 201x

Abstract
Object recognition in 3D scenes is one of the fundamental tasks in computer vision. It is used frequently in robotics or augmented reality applications [1]. In our work we intend to apply 3D shape recognition to create a Tangible Augmented Reality system that is able to pair virtual and real objects in natural indoors scenes. In this paper we present a method for arranging virtual objects in a real-world scene based on primitive shape graphs. For our scheme, we propose a graph node embedding algorithm for graphs with vectorial nodes and edges, and genetic operators designed to improve the quality of the global setup of virtual objects. We show that our methods improve the quality of the arrangement significantly.

Keywords: *Object Detection, Shape Recognition, Graph Embedding, Genetic Algorithm*

[1]Department of Control engineering and Information Technology,
Faculty of Electrical Engineering and Informatics,
Budapest University of Technology and Economics
H-1521 Budapest, P.O.B. 91, Hungary (e-mail: szemenyei@iit.bme.hu, vajda@iit.bme.hu)

1 Introduction

Object recognition and detection is one of the vital tasks in computer vision. While most methods use 2D visual information only [2], there are numerous 3D shape based recognition techniques [3] [4], as well as methods that use both visual and shape information [5] [6]. Object detection methods are essential for scene understanding [7], which has a number of applications in different fields, such as robotics [1] or augmented reality. [8]

One such application is Tangible Augmented Reality (TAR) [9], in which virtual objects are attached to real ones, and the real-world objects serve as input devices for user manipulation. While most TAR systems use real objects with artificial markers for their systems, [10] [11] there are a few that are able to use any object with sufficient natural features. [12] Still, even these systems do not pair real and virtual objects intelligently in order to ensure easy user manipulation.

In this paper we present an algorithm that performs the matching of virtual and real objects in a scene with no artificial features using 3D shape recognition. This way, virtual objects can be paired with real ones with similar shape, resulting in easily learnable interaction techniques.

Our method describes the shape of objects and scenes using graphs of primitive shapes. [13] This ensures that the actual segmentation of objects is also learnable. First, we use a support vector machine to classify the segments individually, then we optimize the labels over the entire scene using a genetic algorithm. The cost function used for the second part ensures that the setup of the scene satisfies multiple criteria in addition to shape similarity (such as the presence of certain categories, relative position of objects, etc.).

One of our main contributions in this paper is a graph node embedding framework for graph that have vectorial node and edge weights. We show, that this embedding significantly improves the segment-by-segment classification accuracy. Our other contribution is the proposal of genetic operators specifically designed to improve the performance of evolutionary optimization methods for the problem presented in this paper.

In the next section, we discuss relevant results of other workshops in the areas of 3D shape recognition, graph recognition, and global optimization. Then, in section 3, we present our shape description method, with emphasis on the graph node embedding framework. In section 4, we present the optimization algorithm used to determine the arrangement of virtual objects in the scene, including the cost function, and the genetic operators used. Finally, we present and evaluate the results of our methods on several datasets.

2 Related Work

In this chapter, we discuss results related to our own work. We begin by elaborating on the various methods for 3D shape recognition, while in the second part we discuss classification and recognition in graphs. In the last part of the chapter, we discuss various global optimization methods.

2.1 3D shape recognition

Classifying objects based on 2D shape is a relatively common task, for which numerous methods exist. Most of these methods, however, cannot be generalized easily to 3D shapes [14]. Still, there are a few, such as the Generalized Hough Transform [4], or RANSAC [15] work reliably for 3D recognition as well. Nonetheless, these algorithms require a reference model for matching, which cannot be easily obtained, mainly due to high intra-class variation.

When no reference model is available, learning algorithms are more appropriate for the task. There are numerous approaches for this, such as using local features [16] [17] to create a learning algorithm. Another approach is described in [14], using shape distributions. These algorithms, however, require a segmentation step in order to produce object candidates for classification. In relatively simple 3D scenes, with helpful prior information (such as urban scenes, where the ground is easy to segment) this may be easy to do. On the other hand, in complex, cluttered scenes (such as indoors scenes) segmentation might become unreliable, resulting in inferior detection performance.

Recently deep convolutional neural networks (CNN) have become increasingly popular amongst researchers working on object recognition and detection, mainly due to their superb efficiency. [2] [18] Unsurprisingly, there is significant work on 3D object detection using either depth-based [5] [6] or volumetric [18] [19] data. Since CNNs are able to perform (super)pixel classification, multi-object detection in larger scenes using CNNs is relatively straightforward. [5] [20] Nonetheless, CNNs are notoriously difficult to train [21] [22], since they are fraught with numerical difficulties. CNNs also require large amounts of training data and computational resources.

Another approach is presented by Schnabel et. al [23], who use a RANSAC variant to segment a scene into primitive shapes (such as plane, sphere, cylinder, etc.), which they treat as the “building blocks” of the objects and the scene. Their algorithm uses local sampling and inlier detection, in order to increase the chance of finding local shapes. They proceed by constructing a topology graph of the object, where the nodes of the graph are the primitives, and edges represent the geometric relations between the nodes. The adjacency between the shapes is determined by their distance. [13]

In order to detect objects in a larger scene, they construct a reference graph for each category, and apply brute-force graph matching. Since a single reference graph has relatively low number of nodes, the matching algorithm remains feasible. [13] They further decrease the number of possible matches between the scene and the reference graphs, by introducing a number of constraints. Node constraints ensure that only nodes of the same primitive type are matched, while edge constraints enforce the similarity of the relations between adjacent nodes. A third type or constraints – graph constraints can be used to take the relationship of non-adjacent nodes into account as well.

2.2 Graph classification

Graph based learning has numerous applications, including bioinformatics [24] and network analysis. [25] Recognizing objects visually using graph-based learning is relatively common as well, since objects can usually be described using graph of (visual) features. [26] In this section we discuss relevant results of graph-based learning.

The difficulty of graph classification is that most standard learning algorithms require a vector (or tensor) of features as their input. Since these methods cannot take graphs as inputs, a way to convert it to a vectorial representation – to embed the graph into a vector space – is needed. This, however, is not a simple task, since the ordering of graph nodes is arbitrary, and any simple method of vectorizing a graph would yield a vector that is not invariant to the ordering of nodes. [27] A related difficulty is, that graphs of different sizes yield vectors of different dimensions, while standard learning algorithms assume, that all data is in the same vector space.

One class of learning algorithms employ so called kernel functions in order to implicitly use a higher dimensional representation of the data for learning. These kernel functions are symmetric, positive semi-definite functions that can usually be interpreted as a similarity measure between objects. [28] When using kernel learning methods (such as SVM), an elegant solution presents itself: defining a kernel function between graphs. Since a kernel function does not require a vectorial input, nor does it explicitly produce a vectorial representation, the entire problem can be circumvented.

Perhaps the most widely-known graph kernel is the random walk kernel [28], which interprets the edge weights of the graph as the probabilities of taking that edge during a random walk. It performs simultaneous random walks on the two graphs, and derives a similarity score based on the probability of performing the same walk. This probability is computed on the direct product of the two graphs using the following equation:

$$K(X,Y) = \sum_{i=1}^{N} w(i)\boldsymbol{e}^T\boldsymbol{A}^i\boldsymbol{s} \tag{1}$$

$$w(i) = e^{-\vartheta i}, \tag{2}$$

where $\boldsymbol{A}$ is the adjacency matrix of the direct product, $\boldsymbol{s}$ and $\boldsymbol{e}$ contain the probabilities of starting and ending the walk on a given node respectively, N is the maximum length of the walks considered, and ϑ is a hyperparameter controlling the slope of the weight w.

However, when using non-kernel learning algorithms the graphs must be embedded explicitly. Perhaps the most widely used method for explicit vectorial embedding is the spectral representation. [29] The simplest version of spectral embedding is to compute the spectral decomposition of the adjacency matrix of the graph. (3) If the graph has weights on the nodes, these can be inserted in the diagonal of the adjacency matrix. [29]

$$\boldsymbol{A} = \boldsymbol{V\Lambda V}^T \tag{3}$$

If the eigenvalues and the corresponding eigenvectors are ordered, then this representation will be partially invariant to node ordering. Since this invariance is only partial, and alignment step is still needed. [30] Furthermore, spectral embedding is able to handle graphs of different sizes, by enlarging the smaller graph using dummy nodes. [31]

Aside from the adjacency matrix, other matrices may be used for the spectral decomposition. The Laplacian matrix of graphs is a rather frequent choice, which is computed (4) using the adjacency ($\boldsymbol{A}$), and the degree ($\boldsymbol{D}$) matrices of the graph. One other method for embedding graphs is the heat kernel (5). The t parameter heat kernel controls the trade-off between local and global representation of the graph. According to Zhu and Wilson [32] the heat kernel outperforms the other two. It is also possible to mix different spectral representations [31] in order to create a more robust method.

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A} \tag{4}$$

$$\boldsymbol{H} = \boldsymbol{V}e^{-t\boldsymbol{\Lambda}}\boldsymbol{V}^T \tag{5}$$

However, we intend to classify a graph on a node-by-node basis, which means that instead of embedding entire graphs, we need to embed nodes into a vector space. In contrast with embedding graphs, there has been very little work done on the topic of embedding nodes. For instance, [26] has used a low-distortion node embedding framework to perform many-to many feature matching using the earth movers distance. Riba et. al. [33] use binary embedding to produce hash keys for graph retrieval.

These methods, however, place limitations on the structure of the graphs or the weights of nodes and edges. Since our shape description method yields full graphs with vectorial weights on both the nodes and edges, the previous methods are insufficient for our application. To our best knowledge, no work has been done yet on embedding graph nodes of vectorial weighted graphs, with no restrictions on the topology.

2.3 Global optimization

Optimization problems are frequent in learning computer vision, and scene understanding. [34] In many cases, the optimization problem is relatively simple (for instance, if the cost function and constraints are convex), and may be solved using standard gradient-based, or second order methods. In constrained cases, methods for solving linear or quadratic programs may be used. [35]

These algorithms, however rely on the gradient of the cost function to some extent. If the gradient cannot be computed, these methods are unusable. Moreover, there may be constraints on some variables that make the problem NP-hard, as is the case, when some variables are required to be integers or binary. A final problem is the tendency of simple optimization methods to get stuck in local minima.

To solve these complex optimization problems, one needs to rely on heuristic methods. These methods are capable of finding global optima even in difficult problems, however, it is difficult (and in most cases impossible) to guarantee their convergence. Moreover, according to the No Free Lunch Theorem (NFLT) [36], on average no optimization method can outperform brute-force search on all the optimization problems.

It is possible, however, to make a heuristic method that performs well on a subset of problems. Variations of hill-climbing method are very popular, such as shotgun-gradient methods, or simulated annealing. [37] One of the most popular heuristic methods are genetic, or evolutionary methods. [38] The idea behind these methods is to implement a scheme similar to biological evolution: simulating subsequent generation of solution candidates, using fitness based selection for offspring generation, and random mutation to create a (hopefully) better population.

One great advantage of genetic algorithms is that by implementing problem specific crossover and mutation operators, it is possible to apply these algorithms efficiently to almost any kind of problem. Weise [37] details the most commonly used operators for genetic methods. There are also numerous fitness scaling and selection schemes [37], which influence convergence greatly.

There is a great number of further heuristics that aim to increase the efficiency of genetic algorithms. On such idea is to use multiple populations, instead of just one, and implement some form of migration between them. This enables the algorithm to explore the parameter space more easily, since populations share similar genes, therefore tend to converge. [39]

Genetic algorithms are not guaranteed to improve the best individual every situation. This might result in the optimal solution disappearing from the population. In order to avoid this it is possible to introduce elitism to the selection strategy. [40] This means, that the best few individuals always survive, and become a part of the next generation unchanged.

3 Graph based shape description

In multi-instance object detection algorithms one of the first steps is usually a segmentation procedure, aiming to produce "object candidates" for a subsequent classification method. This is a viable way in shape recognition, especially in scenes, where segmentation is relatively straightforward. In urban scenes, for instance, one can easily remove the ground, resulting in most of the objects becoming disjoint in the point cloud. [3]

This, however, cannot be done in indoors scenes, since it is significantly more frequent for objects to be cluttered in this context. Therefore, we use a different approach: we segment our scene into primitive shapes (Fig. 1), and classify these 'building blocks' individually first. Since primitive shapes have several features (depending on the primitive type), it would be straightforward to use these features to classify each primitive.

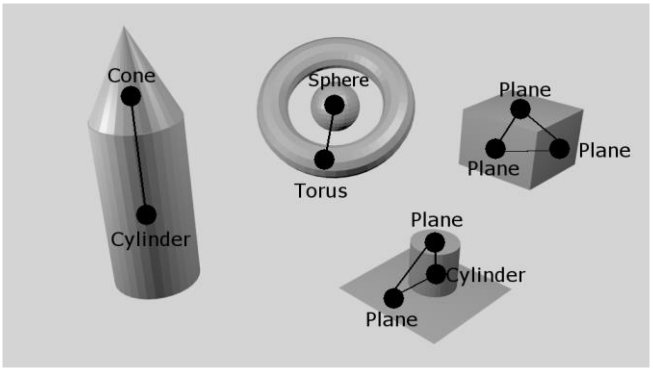

Fig. 1 A graph constructed from primitive shapes (only close edges are shown)

However, this way, geometric relations between the primitives, and the local context of each primitive would be ignored. This could lead to high classification errors, if two classes contain very similar shapes, albeit in different contexts. Therefore, we construct a graph from the primitive shapes, and use a graph node embedding procedure in order to produce a feature vector for each primitive that encodes the local context of the primitive as well.

In this section we briefly discuss the graph construction process, then we describe our graph node embedding framework in detail.

3.1 Construction of shape graphs

The first step of constructing shape graphs is to segment the 3D point cloud of the scene using the algorithm proposed by Schnabel et. al. [23] Their implementation is able to detect five different primitive shapes: planes, cylinders, cones, spheres and tori. We then construct a graph using the primitive shapes as nodes, while the edges represent the geometric relations between the nodes.

Each primitive shape type has a few distinct features that further define the exact shape of the point cloud they represent. (Table 1) By computing these features we are able to assign a feature vector to each primitive shape. Since the features of the different primitive types are incompatible, it makes sense to construct a unified feature vector for each primitive by concatenating the features of the individual types. Of course, for every primitive shape the features of the other types will be set to zero.

Table 1 Primitive shapes and their features

Primitive	Plane	Cylinder	Sphere	Cone	Torus
Features	Area, Diameter, Bounding Box Area	Radius, Height	Radius	Radius, Height, Angle	Inner Radius, Outer Radius

Furthermore, each primitive shape can be easily assigned with a coordinate system, consisting of an origin and at least a single direction in the 3D space. (The only exception is the sphere, where there is no special direction.) This means, that we may describe the geometric relations between the primitives by computing the rigid transform between their coordinate systems. (Table 2) However, since we want our algorithm to be invariant to rotation, we only consider the distance between the origins, and the angle of the rotation between their special directions. Since spheres do not have a special direction, the angle between spheres and other primitives is always set to zero.

Table 2 Coordiante Systems for pirmitives

Primitive	Plane	Cylinder	Sphere	Cone	Torus
Origin	Centroid	Centroid	Centroid	Peak	Centroid
Direction	Normal	Axis	N/A	Axis	Normal

It is worth noting, that both the nodes and the edges of the constructed graph have vectorial weights. Furthermore, the edges of the graph have two different types of weights. The first type is the traditional "distance" type, meaning, that if this feature is larger, then the two nodes are less connected. The second type is the "feature" type, which describes other qualities of the connection, but its magnitude does not influence the strength of the connection. Moreover, we construct full graphs, therefore we do not make explicit decisions on the adjacency of the nodes, we simply store the distance between them amongst the features. This way the subsequent steps can make use of a continuous adjacency property, without loss of information.

3.2 Graph node embedding framework

In this subsection we present our graph node embedding framework in detail. Our goal is to create descriptors for nodes in graphs that have vectorial node and edge weights. We also wish to place no restrictions on the graph structure, that is, we propose a framework that is applicable to full, directed graphs as well.

Our embedding method aims to describe 'what the graph looks like' from the perspective of the node that is being embedded (the central node). Therefore the framework needs to include information on the features of the central node, as well as the surrounding ones. It also needs to incorporate information on the geometric relations between the nodes.

Since our goal is to embed the local context of the node, the influence of nodes farther from the central node must be less than that of the immediate neighbours. This means that if the edge features of the graph include a parameter that can be interpreted as “distance” or “connection strength”, then this parameter may be used to weight the influence of the nodes. Since shape graph edges have a distance parameter, we will use it for our discussion without loss of generality, since connection strength may be understood as the inverse of distance.

The first step of the embedding process is to order the nodes of the graph in based on the distance from the central node. Since the spectral embedding is only partially invariant to the node ordering, this step alone ensures that the feature vectors are different for separate nodes. If the ordering is not obvious due to some nodes being too close, then two separate feature vectors may be made with the different orderings and averaged. In order to create descriptors of the same size for all nodes, the maximum number of nodes included must be set. Distant nodes are clipped from larger graphs, while smaller ones are padded with zero nodes and edges.

It is important to note, that padding should not alter the original shape of the scene. However, if features are normalized, then padding the graph with all-zero nodes means that we are adding average shapes to average distance, which might affect the embedding adversely. Luckily, there is a simple way to avoid this. Since our node and edge features are non-negative, we divide the features with the standard deviation, but do not subtract the mean. Thus, the non-negative property of our features is preserved, and adding zero nodes to the graph is equivalent with leaving the shape of the scene unchanged.

The next step of our embedding method is to construct a graph feature matrix ***F*** according to the equation below.

$$\boldsymbol{F} = \begin{bmatrix} T_{11} & \cdots & T_{1N} \\ \vdots & \ddots & \vdots \\ T_{N1} & \cdots & T_{NN} \end{bmatrix}$$
$$T_{ij} = T\left(n_i, n_j, e_{1i}, e_{1j}, e_{ij}\right) \qquad (6),$$

where T is a feature transform function, n_i is the i_{th} node of the graph, while e_{ij} is the edge pointing from the i_{th} to the j_{th} node. N is the maximum number of neighbouring nodes considered in the embedding. Note, that there are very few restrictions on the properties of T, leaving the choice of feature transform to the researcher.

Nonetheless, there are a few helpful guidelines for constructing the transform function. First, we have previously divided edge features into “distance” (e_d) and “feature” (e_f) types. We treat feature type values the same way we treat node features, therefore we concatenate them to the node feature vectors. (7) Second, we use the distance type features to scale the result of the feature transform, therefore distant nodes will not affect the graph feature matrix significantly. Our choice for the feature transform function is shown in the equation below.

$$T_{ij} = w_{ij} \begin{bmatrix} n_i \\ e_{1if} \\ e_{ijf} \end{bmatrix} \begin{bmatrix} n_j & e_{1jf} & e_{ijf} \end{bmatrix}$$
$$w_{ij} = \frac{1}{1 + \mu\left({e_{1id}}^2 + {e_{1jd}}^2 + {e_{ijd}}^2\right)} \qquad (7),$$

where μ is a hyperparameter controlling the distance scaling. It is important to note, that we do not only scale the feature transform using the distance from the central node, but also the distance between the two interacting nodes. This is because the interaction between distant nodes is not significant for a local context-based embedding.

In order to finalize the embedding process, we compute the singular value decomposition of the graph feature matrix, and concatenate the first couple singular values and vectors (8). When using quadratic, antisymmetric ($T_{ji} = T_{ij}^T$) feature transform functions the graph feature matrix is guaranteed to be symmetric. In this case, the eigendecomposition may be used in order to reduce the size of the feature vector. However, we do not place such restrictions on the transform function, hence using SVD is recommended.

$$v = \begin{bmatrix} \sigma_1 u_1 \\ \sigma_1 v_1 \\ \vdots \\ \sigma_k u_k \\ \sigma_k v_k \end{bmatrix} \qquad (8),$$

where σ_i, u_i, and v_i are the i_{th} singular value, left and right singular vectors respectively, and k is the maximum number of singular values considered.

4 Object detection in scenes

The second part of our object matching algorithm is the optimization of the object pairings in the entire scene. This is needed, since the individual and independent classification of primitive shapes might result in inferior performance. This is caused by the fact, that we are classifying segments individually, without considering the labels of nearby segments. However, nearby primitive shapes are likely to belong to the same object, thus have the same label. By introducing a compactness requirement, we encourage close primitive shapes to take the same label.

Moreover, there are several requirements unique to the TAR application, which need to be taken into consideration when determining the final setup of the scene. For instance, the presence of some virtual objects might be necessary for the AR application to work, while others may be optional. In other cases it may be appropriate to encourage the algorithm to place more than one instance of certain objects.

A further possible addition to the requirements is category context. With this additional part, we may encourage or discourage certain categories to be near to each other. There are cases in which objects tend to be near, or even touching in scenes (items are usually placed on tables, for instance). In this case, the aforementioned compactness requirement might punish the system for making sensible placement decisions.

In this section we discuss our method to solve the scene optimization problem. First, we detail the cost function and the constraints we use. Then we discuss our optimization method, focusing on the problem specific genetic operators proposed for our application. In the final part of the section we elaborate on the problem of introducing and learning context information from labelled scenes.

4.1 Cost function for scene optimization

It is worth noting, that without the compactness and context parts, our problem can be formed as a binary linear program (BiLP). The formulation is shown below.

$$\min_{x\in\{0,1\}} c^T x$$

$$s.t. \qquad \sum_j x_{ij} = 1 \quad \forall\, i, \qquad \sum_i x_{ij} \geq a_j \quad \forall\, j$$

$$a_j = \begin{cases} 1 & if\ class\ j\ is\ required \\ 0 & otherwise \end{cases} \qquad (9),$$

where x_{ij} is the binary label indicating whether the i_{th} node has a class label j. The first constraint forces all nodes to have exactly one class label, which is necessary due to the one-hot coding used. The second set of constraints determines whether the object is required or not. Since our variables are non-negative, the second constraint is ignorable for classes where a_j is zero.

The costs of the i_{th} node belonging to the j_{th} class (c_{ij}) can be derived from the node-by-node classification method. For the purpose of normalization it is more sensible to use costs ranging from zero to one instead of raw classification scores. These can be easily attained by transforming raw scores using a softmax function. This way, the c_{ij} cost factors can be interpreted as the probabilities of the i_{th} node belonging to the j_{th} class.

By adding extra parts to the cost function, the problem loses its linearity. Yet, this is not a serious loss, since integer constraints on the variables already make this problem difficult to solve, due to the NP hard nature of ILPs. The first such extra component is compactness, which is computed as follows.

$$C_{comp} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{\mu}{2{d_{ij}}^2} \left(n_i - n_j\right)^T \left(n_i - n_j\right) \qquad (10),$$

where d_{ij} is the distance between the i_{th} and j_{th} nodes, N is the number of nodes, μ is the hyperparameter controlling the relative importance of this criterion, while n_i is the vector containing the binary labels of the i_{th} node. By adding this extension to the cost function, the problem becomes quadratic.

It is also possible to introduce soft requirements for the presence of objects, instead of a hard constraint. This makes sense if the presence of a virtual object is not essential, but we still want to reward the algorithm for placing as many different types of objects as possible. To ensure this, we might add a reward r_j to the cost function, if there is at least one node has the label j.

Another important use of a soft constraint of class presence is the possibility of rewarding the algorithm for placing more than one instance of a virtual object. Introducing this extension is quite tricky, however, since the number of nodes having the label i is not the same as the number of objects. To solve this problems, we introduce node clusters, and count the number of occurrences of the label i in different clusters only. A single cluster of nodes contains nodes that are closer to each other than a predefined threshold. This threshold may be computed adaptively, using the distances in the scene.

The final extension of the cost function is the addition of context. This step allows the algorithm to encourage or discourage the closeness of certain categories. This extension may counter the compactness criterion for certain class combinations only. As mentioned before, this is essential in indoors scenes. The context reward is computed as follows:

$$C_{cont} = \beta \sum_{i=1}^{N_c-1} \sum_{j=i+1}^{N_c} \sigma_{ij} d_{ij,min} \qquad (11),$$

where σ_{ij} is the context coefficient for the i_{th} and j_{th} classes, $d_{ij,min}$ is the minimum distance between nodes belonging to the i_{th} and j_{th} classes, N_c is the number of classes, and β is the hyperparameter controlling the relative importance of this part of the cost function. With all extensions covered, we present the complete cost function.

$$C_{total} = c^T x + \ C_{comp} + C_{cont} - \sum_{i=1}^{N_c} r_i N_i \qquad (12),$$

where r_i is the reward for the presence of the i_{th} class, and N_i is the number of node clusters that have the label i.

4.2 Methods for scene optimization

The next step in arranging the virtual objects in the scene is finding the optimum of the cost function. Because of the NP-hard nature of the optimization problem presented in the previous subsection, heuristic optimization methods are needed. Perhaps the simplest heuristic method available is a greedy neighbourhood search. This algorithm iteratively evaluates all the neighbours of the initial point, and moves to the one with the lowest cost function. The performance of this algorithm depends greatly on the choice of initial point, and the definition of neighbourhood.

There are two obvious ways of setting the initial point. The first is setting all binary labels to zero (creating an infeasible solution), and letting the algorithm build its way greedily to a feasible one. The second method is random initialization. In our case, however, there is a third solution: initializing each node using the node-by-node classification. Arguably, this way the initialization will be relatively close to the optimal solution, making it very likely for the algorithm to converge to the optimum.

For the greedy algorithm, neighbourhood is defined as a single difference in the labels of the nodes. It is important to

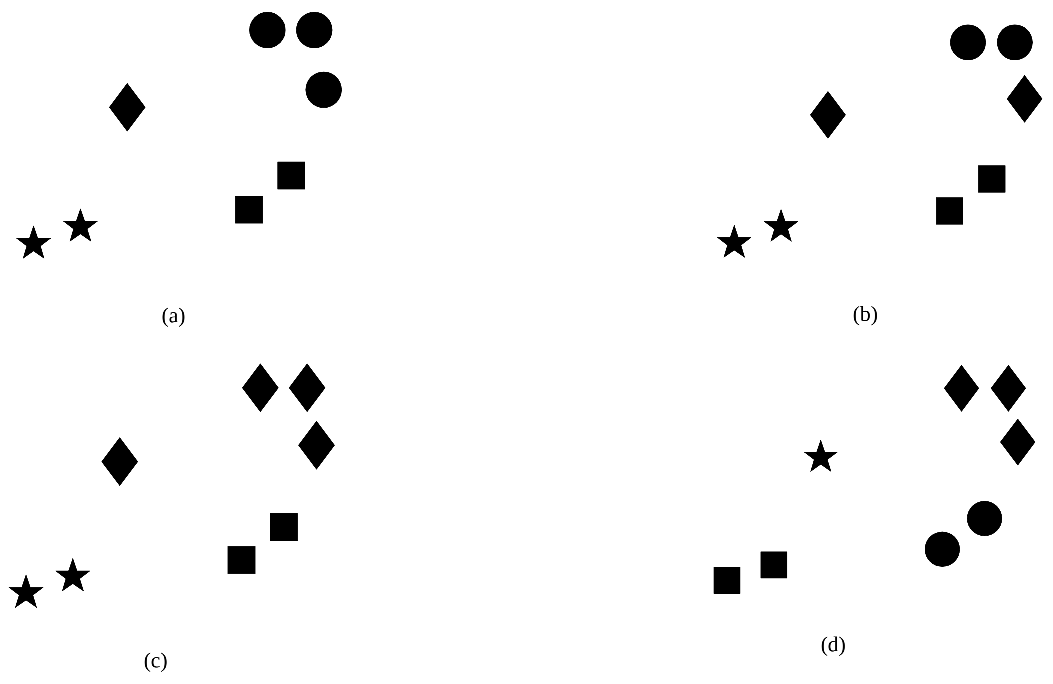

Fig. 2 The mutation operator: (a) is the original setup, (b) is the traditional flip mutation. (c) is flip mutation using drag, and (d) shows label permutation

note, however, that not all neighbours of a feasible solution are feasible, since they are not guaranteed to contain an instance of all required classes. One significant drawback of the greedy search is that it finds local minima. If the classification method has low accuracy, then the initialization may be too far from the true solution for this algorithm to converge reliably.

For this reason, we used a simple single population genetic algorithm with elitist strategy for scene optimization. Moreover, we developed specific crossover and mutation operators, as well as a custom initialization strategy.

Class Score Optimal Initialization (CSOI): The first step of initialization is to determine the population size. Since the number of parameters depends on the size of the scene, using a fixed population size is not recommended. For our problem, setting the population size to 10 times the number of nodes worked relatively well. To create the initial population, we selected the individual that maximizes the classification scores, and added it to the population. The rest of the population consists of mutated versions of this seed solution.

Random Drag&Shuffle Mutation (RDSM): (Fig. 2) The point of the mutation operator is to allow the genetic algorithm to escape local minima by randomly finding better regions of the parameter space. The standard mutation operator for binary/nominal integer genomes is the random flip operator, which randomly changes the label of a single node. In our case, however, the random flip operator is very likely to create a significantly worse candidate solution, because of the compactness element in the cost function. This means, that the mutated solution is not very likely to survive many generations, and allow the algorithm to explore other regions of the parameter space.

To solve this, we introduce the concept of random drag: the node whose label is changed will “drag” other nearby nodes with it with a certain probability. The drag probability influences the trade-off between node mutation and cluster mutation. By allowing both kinds of mutation to occur, the parameter space can be explored more efficiently.

A further idea is to allow the mutation operator to permute the labels themselves with a given probability. This allows the algorithm to consider other combinations of classes, which would require several rounds of consequent mutations otherwise. The RDSM operator is used for generating the initial population.

Clustered N-Point Crossover (CNPC): (Fig. 3) The standard crossover operator for binary/nominal integer genomes is the N-point intersection. This operator randomly selects N-1 intervals in the genome, and inherit them from the two parents in an alternating way. The problem with this operator is similar to the problem we encountered with the mutation, namely, that defining the crossover on the level of nodes may lead to the creation of a high number of inferior offspring.

We solve this problem using a similar idea: we define the intersection operator on node clusters, instead of the nodes themselves. This means, that all nodes are assigned to a cluster based on their proximity, using an adaptive threshold to divide clusters. Then, the clusters are ordered randomly, and divided into N intervals. The labels are inherited from the parents alternatingly.

4.3 Optimizing the context reward

In chapter 3.1, we introduced the idea of using context

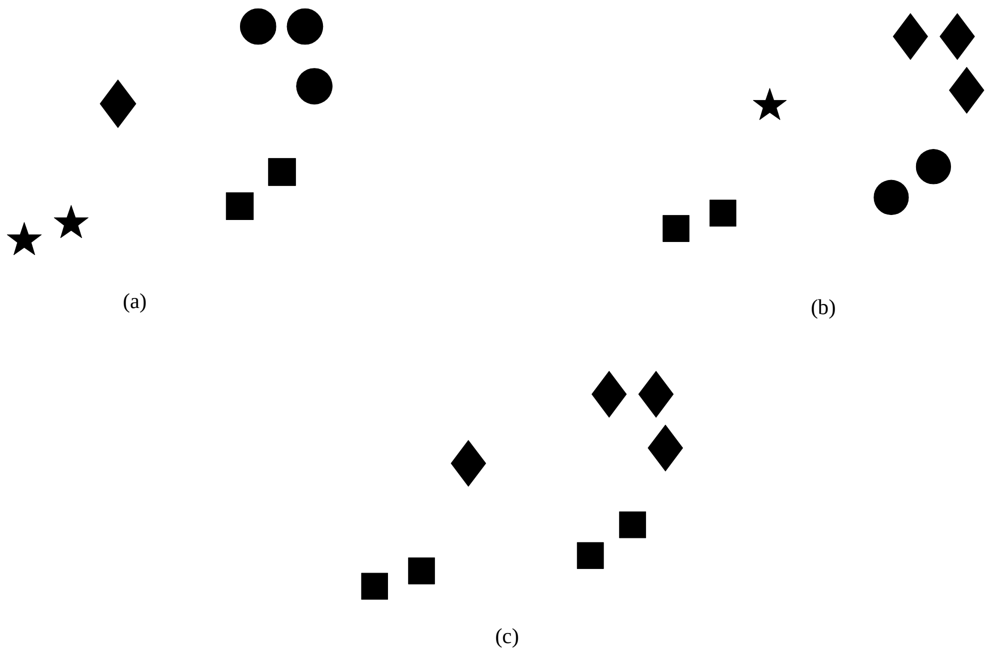

Fig. 3 The crossover operator: (a) and (b) are the parents, (c) is the offspring.

information in the cost function. This way, we allow the optimization algorithm to prefer certain virtual objects being close or distant. This is especially useful for countering the compactness criterion for certain categories only.

In our earlier discussion we have not clarified how the context rewards (σ_{ij} from Eq. 11) are determined for all the possible class combinations. The simplest solution is to let the researcher set them using trial and error methods. This, however, is not only time consuming, it will also likely underperform automated optimization.

For this reason, we use a simple optimization procedure to determine the context reward parameters. In order to achieve this, we run the genetic algorithm and operators introduced in the previous subsection, and select the *N* best unique solutions from the last generation. With their help, we propose the following cost function:

$$C = -\sum_{i=1}^{N} c_i - c_{true} \qquad (13),$$

where c_i is the cost of the i_{th} best solution, while c_{true} is the cost of the ground truth solution. Using this cost function, we can run the stochastic gradient descent (SGD) algorithm to find the optimal values of the context rewards, according to the following derivative:

$$\frac{\partial C}{\partial \sigma_{ij}} = \beta \sum_{k=1}^{N} d_{k,ij} - d_{true,ij} \qquad (14),$$

where $d_{k,ij}$ and $d_{true,ij}$ are the minimum distance between nodes belonging to the class *i* and *j* in the k_{th} best, and the true candidates respectively.

There is one important modification we made to the standard algorithm, namely, that after every few epochs, we rerun the genetic optimization algorithm, and set the best *N* candidates. This technique prevents the algorithm from returning context rewards that although make the true solution better than the original best, but only to result in a new incorrect solution to become the optimum.

5 Experimental Results

In this chapter we present the result of the experiments based on our methods. In the first part, we test the graph node embedding algorithm and show that it outperforms using the vectorial descriptors of the nodes. We use support vector machines to perform node-by-node classification. In the second part of the chapter we evaluate the scene optimization algorithm, showing that the proposed genetic operators significantly improve the performance of the optimization. We also show, that the context optimization is helpful in cluttered scenes.

We use four different datasets for training the classification algorithms. The first dataset consists of synthetic shape graphs. This dataset is meant to be easy to learn, for the five classes use different types of nodes, with some random noise added. Some instances have additional “noise” nodes, while others are missing some nodes to represent errors of the segmentation.

The second dataset is also synthetic, however, the five classes use the same pool of nodes, albeit in different combinations. This dataset is meant to demonstrate that the embedding algorithm significantly outperforms the simple node descriptor based classification in cases where the different classes have very similar nodes, but in different configurations.

The third dataset consists of images of synthetic objects created in Blender. There are five classes in ten variations each, with series of images taken with a multiple camera. We created hundreds of partial 3D reconstructions for each category using

VisualSfM. [41] The last database uses real images in four categories and 4-10 variations for each category. Here, the categories are relatively simple (box/book, mug/can, sphere, and horizontal surface).

For the scene optimization, we created one scene database for each training database using the same objects. We have also created two databases, where certain categories are deliberately cluttered, making context learning essential. These two databases use the synthetic and real images respectively.

5.1 Graph node embedding

We evaluated the performance of our node embedding method using support vector machines with RBF kernels. We applied Bayesian optimization to find good hyperparameter values, using 20% holdout validation error (e_{ho}) as the objective function. We then compute the training (e_{class}) and 10-fold cross-validation errors (e_{cv}) with the acquired hyperparameters. We compare the classification and validation errors with and without applying node embedding to the datasets. The results for all datasets are shown in Table 3.

Table 3 Node-by-node classifiaction errors

Metric	e_{ho}		e_{cv}		e_{class}	
Embedding	No	Yes	No	Yes	No	Yes
Synthetic	9.6	**1.9**	10.1	**2.1**	6.6	**0.8**
Synthetic2	67.5	**2.4**	68.1	**2.4**	64.6	**0.6**
Synthetic Images	39.2	**32.8**	39.2	**31.7**	33.6	**14.6**
Real Images	11.4	**7**	10.4	**6.4**	9.2	**5.3**

It is easy to see, that the graph node embedding algorithm significantly decreases the loss of the node-by-node classification in all cases. Interestingly, the first synthetic database shows some improvement, even though it was was created so, that it would be easy to classify without embedding. The reason for this is that the embedding method encodes the local context into the noise nodes, making them easy to classify. Arguably, the node embedding method is most helpful for the second synthetic dataset that contains classes with similar nodes in different configurations.

5.2 Object detection

For testing the scene optimization method, we used the scene databases mentioned earlier. We compared the performance of two algorithms: the simple greedy method, and the genetic algorithm using our operators. We have computed four performance indicators for each run. First, we compute the classification error (e_c) of nodes in the scenes, and compare it with the SVM-only classification error (e_{svm}).

Furthermore, we compute the percentage of scenes, where the cost of the optimal scene labelling is lower than the cost of the ground truth (e_{cost}). This metric shows the frequency of 'cost function failures', however, it is also dependant on the performance of the optimization method. The fourth metric is the percentage of scenes, where the solution found by the optimization algorithm has higher cost than the true solution (e_{opt}). This metric evaluates the performance of the optimization method. These last two metrics are not independent, however. As the optimization algorithm gets better at finding the optimum, it also becomes more likely to find a solution that is better than the true scene arrangement, if such a solution exists. This means that as e_{opt} decreases e_{cost} will likely increase.

Table 4 Result of the scene optimization

Metric	e_c				e_{svm}			
Algorithm	Greedy		Genetic		Greedy		Genetic	
Embedding	No	Yes	No	Yes	No	Yes	No	Yes
Synthetic	0.1	0.2	**0**	**0**	6.5	**0.9**	6.5	**0.9**
Synthetic2	51.1	0.2	3.8	**0**	64.6	**0.6**	64.6	**0.6**
Synth. Images	23.9	12.2	16.2	**6.3**	33.4	**14.4**	33.4	**14.4**
Real Images	4.2	1.2	3.5	**0.7**	8.9	**5.1**	8.9	**5.1**
Metric	e_{cost}				e_{opt}			
Synthetic	0.1	**0**	0.1	**0**	0.3	0.3	**0**	**0**
Synthetic2	**0**	**0**	2.7	**0**	97.5	0.7	7.5	**0.1**
Synth. Images	18.9	**12.3**	35.3	19.5	45.9	28.9	3.6	**0.8**
Real Images	1.3	**1.1**	1.5	1.4	7.5	6.4	**0**	**0**

The results (Table 4) show, that the scene optimization method reliably outperforms the node-by-node classification. It is also obvious, that the graph node embedding method improves the final classification accuracy as well. Apparently, the genetic algorithm outperforms the greedy building, especially in cases, where the node-by-node classification error is high, therefore the true optimum is presumably far from the initialization point.

We have also compared the performance of the proposed genetic operators with the vanilla genetic operators for one-hot coded nominal genomes. The results (Table 5) show, that each operator improves the performance of the genetic optimization on its own. The Random Drag and Shuffle Mutation (RDSM), and the Class Score Optimal Initialization (CSOI) achieve the highest decrease in error, while the improvement caused by the Cluster N-Point Crossover (CNPC) is more modest. Note that the operators were enabled in the same order they are presented in the table. The changes in the table show the decrease in error *in addition to the previous operators' result.*

Table 5 Change in errors caused by the special genetic operators

Method	RDSM		CNPC		CSOI	
Metric	e_c	e_{opt}	e_c	e_{opt}	e_c	e_{opt}
Synthetic	59.4	65.5	0.8	1.5	9.5	29.2
Synthetic2	55.5	52	0.7	0.3	14.1	47.2
Synthetic Images	57.2	66.9	0.8	2.9	2.9	11.5
Real Images	43.3	62.7	0.2	0.4	1.7	2.4

We also evaluated the performance of context optimization, using the two scene databases created for this purpose. We present the final classification error and the cost function error for these two datasets, with and without context optimization. The results (Table 6) show, that context optimization is able to reliably improve the scene optimization method performance, as long as the closeness of certain objects is dependent on their categories.

Table 6 Results before and after the context optimization

Metric	e_c		e_{cost}	
Context	No	Yes	No	Yes
Synthetic Images	28.5	**17.9**	92.7	**66.4**
Real Images	17.3	**8.7**	47.9	**27.9**

6 Conclusion

In this paper, we have presented an algorithm for pairing virtual and real objects for an adaptive Tangible Augmented Reality (TAR) system using shape recognition. Our method builds primitive shape graphs from the 3D point clouds, and uses a graph node embedding method to allow for superb node-level classification. We then use a genetic algorithm with special operators to optimize the arrangement of virtual objects globally.

We have shown, that node embedding improves classification accuracy significantly, especially when there are similar nodes in different classes, albeit in different configurations. We have also shown, that constructing a cost function for TAR and optimizing it can significantly improve the accuracy of the pairing algorithm compared to the node level classification. The class score optimal initialization scheme, random drag and shuffle mutation and clustered n-point crossover operators improve the chance of the genetic optimization algorithm to find the optimal solution significantly.

References

[1] C. Wojek, S. Walk, S. Roth, K. Schindler and B. Schiele, “Monocular Visual Scene Understanding: Understanding Multi-Object Traffic Scenes,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(4), pp. 882-897, 2012. DOI: 10.1109/tpami.2012.174.

[2] C. Szegedy, W. Liu and Y. Jia, “Going deeper with convolutions,” in IEEE Conference on Computer Vision and Pattern Recognition, 2015. DOI: 10.1109/CVPR.2015.7298594.

[3] A. Golonivskiy, V. G. Kim and T. Funkhouser, “Shape-based Recognition of 3D Point Clouds in Urban Environments,” in IEEE 12th International Conference on Computer Vision, Kyoto, Japan, 2009. DOI: 10.1109/iccv.2009.5459471.

[4] F. Tombari and L. Di Stefano, “Object recognition in 3D scenes with occlusions and clutter by Hough voting,” in Fourth Pacific-Rim Symposium on Image and Video Technology, Singapore, 2010. DOI: 10.1109/psivt.2010.65.

[5] C. Wu, I. Lenz and A. Saxena, “Hierarchical Semantic Labeling for Task-Relevant RGB-D Perception,” in Robotics: Science and Systems, 2014. DOI: 10.15607/rss.2014.x.006.

[6] M. Schwarz, H. Schulz and S. Behnke, “RGB-D Object Recognition and Pose Estimation based on Pre-trained Convolutional Neural Network Features,” in Proceedings of the IEEE International Conference on Robotics and Automation, Seattle, 2015. DOI: 10.1109/icra.2015.7139363.

[7] L.-J. Li, R. Socher and L. Fei-Fei, “Towards Total Scene Understanding:Classification, Annotation and Segmentation in an Automatic Framework,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2009. DOI: 10.1109/cvprw.2009.5206718.

[8] O. Pauly, B. Diotte, P. Fallavollita, S. Weidert, E. Euler and N. Navab, “Machine learning-based augmented reality for improved surgical scene understanding,” Computerized Medical Imaging and Graphics, 41, pp. 55-60, 2015. DOI: 10.1016/j.compmedimag.2014.06.007.

[9] M. Billinghurst, H. Kato and I. Poupyrev, “Tangible Augmented Reality,” in ACM SIGGRAPH ASIA, 2008. DOI: 10.1145/1508044.1508051.

[10] M. Billinghurst, H. Kato and S. Myojin, “Advanced Interaction Techniques for Augmented Reality Applications,” in Lecture Notes in Computer Science, Berlin, Springer, 2009, pp. 13-22.DOI: 10.1007/978-3-642-02771-0_2.

[11] G. A. Lee, M. Billinghurst and G. J. Kim, “Occlusion based Interaction Methods for Tangible Augmented Reality Environments,” in Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry, Los Angeles, CA USA, 2004. DOI: 10.1145/1044588.1044680.

[12] W. Broll, E. Meier and T. Schardt, “The Virtual Round Table - a Collaborative Augmented Multi-User Environment,” in Proceedings of the ACM Collaborative Virtual Environments, San Fransisco, CA USA, 2000. DOI: 10.1145/351006.351011.

[13] R. Schnabel, R. Wahl, R. Wessel and R. Klein, “Shape Recognition in 3D Point Clouds,” in The 16-th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision'2008, Bory, 2008. Available: http://cg.cs.uni-bonn.de/aigaion2root/attachments/schnabel-2008-shape.pdf.

[14] R. Osada, T. Funkhouser, B. Chazelle and Dobkin David, “Matching 3D Models with Shape Distributions,” in SMI 2001 International Conference

on Shape Modeling and Applications, Genova, 2001. DOI: 10.1109/sma.2001.923386.

[15] M. A. Fishler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” Magazine Communications of the ACM, 24(6), pp. 381-395, 1981. DOI: 10.1145/358669.358692.

[16] R. Rusu, N. Blodow and M. Beetz, “Fast Point Feature Histograms (FPFH) for 3D Registration,” in In Proceedings of the IEEE International Conference on Robotics and Automation, Kobe, 2009. DOI: 10.1109/ROBOT.2009.5152473.

[17] S. Lazebnik, C. Schmid and J. Ponce, “A sparse texture representation using local affine regions,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 27, pp. 1265-1278, 2005. DOI: 10.1109/TPAMI.2005.151.

[18] Z. Wu, S. Song, A. Khosla, L. Z. F. Yu, X. Tang and J. Xiao, “3D ShapeNets: A Deep Representation for Volumetric Shape Modeling,” in Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition, 2015. DOI: 10.1109/CVPR.2015.7298801.

[19] S. Bai, X. Bai, Z. Zhou, Z. Zhang and L. J. Latecki, “GIFT: A Real-time and Scalable 3D Shape Search Engine.,” in Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition, 2016. DOI: 10.1109/CVPR.2016.543.

[20] S. Gupta, R. Girshick, P. Arbel´aez and J. Malik, “Learning Rich Features from RGB-D Images for Object Detection and Segmentation,” in European Conference on Computer Vision, 2014. DOI: 10.1007/978-3-319-10584-0_23.

[21] Y. Bengio, “Practical Recommendations for Gradient-Based Training of Deep Architectures,” in Neural Networks: Tricks of the Trade, Springer, 2012, pp. 437-478.DOI: 10.1007/978-3-642-35289-8_26.

[22] L. Bottou, “Stochastic Gradient Descent Tricks,” in Neural Networks, Tricks of the Trade, Reloaded, Springer, 2012, p. 430–445.DOI: 10.1007/978-3-642-35289-8_25.

[23] R. Schnabel, R. Wahl and R. Klein, “Efficient RANSAC for Point-Cloud Shape Detection,” Computer Graphics Forum, 26(2), pp. 214-226, 2007. DOI: 10.1111/j.1467-8659.2007.01016.x.

[24] R. Sharan and T. Ideke, “Modeling cellular machinery through biological network comparison,” Nature Biotechnology, 24(4), p. 427–433, 2006. DOI: 10.1038/nbt1196.

[25] R. Kumar, J. Novak and A. Tomkins, “Structure and evolution of online social networks,” in Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006. DOI: 10.1145/1150402.1150476.

[26] M. F. Demirci, Y. Osmanlioglu, A. Shokoufandeh and S. Dickinson, “Efficient many-to-many feature matching under the l1 norm,” Journal of Computer Vision and Image Understanding, 115(7), pp. 976-983, 2011. DOI: 10.1016/j.cviu.2010.12.012.

[27] R. C. Wilson, E. R. Hancock and B. Luo, “Pattern vectors from algebraic graph theory,” IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(7), p. 1112–1124, 2005. DOI: 10.1109/tpami.2005.145.

[28] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor and K. M. Borgwardt, “Graph Kernels,” Journal of Machine Learning, 11, pp. 1201-1242, 2010. Available: http://www.jmlr.org/papers/volume11/vishwanathan10a/vishwanathan10a.pdf.

[29] F. Chung, Spectral graph theory, American Mathematical Society, 1997. DOI: 10.1090/cbms/092.

[30] M. Ferrer, F. Serratosa and A. Sanfeliu, “Synthesis of median spectral graph,” in Lecture Notes in Computer Science, vol. 3523, Springer, 2005, p. 139–146.DOI: 10.1007/11492542_18.

[31] D. White and R. C. Wilson, “Mixing Spectral Representations of Graphs,” in Proceedings of the 18th International Conference on Pattern Recognition, 2006. DOI: 10.1109/icpr.2006.803.

[32] P. Zhu and R. C. Wilson, “Stability of the Eigenvalues of Graphs,” in 11th International Conference, CAIP 2005, Versailles, 2005. DOI: 10.1007/11556121_46.

[33] P. Riba, J. Llad´os, A. Forn´es and A. Dutta, “Large-scale Graph Indexing using Binary Embeddings of Node Contexts,” in Proceedings of the 10th IAPR-TC-15 International Workshop, Beijing, 2015. DOI: 10.1007/978-3-319-18224-7_21.

[34] M. Sofka, J. Zhang, S. K. Zhou and D. Comaniciu, “Multiple object detection by sequential monte carlo and Hierarchical Detection Network,” in IEEE Conference on Computer Vision and Pattern Recognition, 2010. DOI: 10.1109/cvpr.2010.5539842.

[35] S. Boyd and L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004. Available: http://stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

[36] D. Wolpert and W. Macready, “No Free Lunch Theorems for Optimization,” IEEE Transactions on Evolutionary Computation, 1(1), pp. 67-82, 1997. DOI: 10.1109/4235.585893.

[37] T. Weise, Global Optimization Algorithms - Theory and Application, Self-Published, 2009. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.569.5284&rep=rep1&type=pdf.

[38] A. S. Fraser, “Simulation of genetic systems by automatic digital computers,” Australian Journal of Biological Science, 10(1), p. 484–491, 1957. DOI: 10.1071/bi9570484.

[39] J. P. Cohoon, S. U. Hegde, W. N. Martin and D. Richards, “Punctuated equilibria: a parallel genetic algorithm,” in Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, 1987.

[40] M. Laumanns, E. Zitzler and L. Thiele, “A unified model for multi-objective evolutionary algorithms with elitism,” in Proceedings of the 2000 Congress on Evolutionary Computation, 2000. DOI: 10.1109/CEC.2000.870274.

[41] C. Wu, “Towards Linear-time Incremental Structure From Motion,” in International Conference on 3DTV, 2013. DOI: 10.1109/3dv.2013.25.