Computers in Biology and Medicine 119 (2020) 103695

Contents lists available at ScienceDirect

Computers in Biology and Medicine

journal homepage: http://www.elsevier.com/locate/compbiomed

# A partition-based optimization model and its performance benchmark for Generative Anatomy Modeling Language

Doga Demirel [a], Berk Cetinsaya [b], Tansel Halic [c,*], Sinan Kockara [c], Dirk Reiners [b], Shahryar Ahmadi [d], Sreekanth Arikatla [e]

[a] Department of Computer Science, Florida Polytechnic University, Lakeland, FL, USA
[b] Department of Computer Science, University of Central Florida, Orlando, FL, USA
[c] Department of Computer Science, University of Central Arkansas, Conway, AR, USA
[d] Department of Orthopedic Surgery, University of Arkansas for Medical Sciences, Little Rock, AR, USA
[e] Kitware Inc., Carrboro, NC, USA

A R T I C L E I N F O

*Keywords:*
Modeling language for human anatomy
Endoscopic submucosal dissection
Virtual human anatomy
non-linear programming
Partition-based optimization

A B S T R A C T

*Background:* This paper presents a novel iterative approach and rigorous accuracy testing for geometry modeling language - a Partition-based Optimization Model for Generative Anatomy Modeling Language (POM-GAML). POM-GAML is designed to model and create anatomical structures and their variations by satisfying any imposed geometric constraints using a non-linear optimization model. Model partitioning of POM-GAML creates smaller sub-problems of the original model to reduce the exponential execution time required to solve the constraints in linear time with a manageable error.
*Method:* We analyzed our model concerning the iterative approach and graph parameters for different constraint hierarchies. The iteration was used to reduce the error for partitions and solve smaller sub-problems generated by various clustering/community detection algorithms. We empirically tested our model with eleven graph parameters. Graphs for each parameter with increasing constraint sets were generated to evaluate the accuracy of our method.
*Results:* The average decrease in normalized error with respect to the original problem using cluster/community detection algorithms for constraint sets was above 63.97%. The highest decrease in normalized error after five iterations for the constraint set of 3900 was 70.31%, while the lowest decrease for the constraint set of 3000 was with 63.97%. Pearson correlation analysis between graph parameters and normalized error was carried out. We identified that graph parameters such as diameter, average eccentricity, global efficiency, and average local efficiency showed strong correlations to the normalized error.
*Conclusions:* We observed that iteration monotonically decreases the error in all experiments. Our iteration results showed decreased normalized error using the partitioned constrained optimization by linear approximation to the non-linear optimization model.

## 1. Background

Generative Anatomy Modeling Language (GAML) is particularly used to generate 3-D virtual human anatomy variations on a web browser via the use of WebGL (3-D rendering application programming Interface for web browsers) technology. In GAML, anatomical constraints can be incorporated prior to any geometry modification to achieve authentic 3D models [1]. The purpose of GAML is to create a platform for the rapid generation of variations in anatomical or biological structures by fulfilling their structural constraints. These constraints can stem from anatomy or any restrictions imposed by users (e.g. physicians or researchers). Our ultimate goal is to minimize any need for bio-illustrators and provide a free platform where modeling and modification, with respect to the anatomy, can be performed at ease without any technical or modeling expertise. We often experience that even slight modifications over 3-D anatomy models require significant design reiterations. This is due to the fact that the accuracy of these models needs to be authenticated after each change done by expert

* Corresponding author. Computer Science Department, University of Central Arkansas, 201 Donaghey Ave, Conway, AR, 72035, USA.
*E-mail address:* tanselh@uca.edu (T. Halic).

https://doi.org/10.1016/j.compbiomed.2020.103695
Received 13 January 2020; Received in revised form 3 March 2020; Accepted 3 March 2020
Available online 5 March 2020

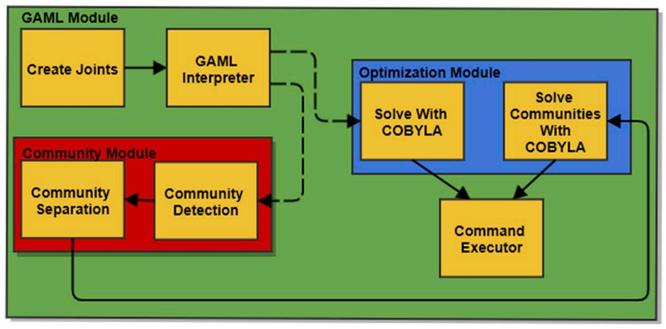

**Fig. 1.** Overall architecture of POM-GAML. Dashed arrows show that either step can be performed, and solid arrows show the order of operations.

physicians to ensure the models are anatomically valid. However, GAML could be able to make modifications on 3-D models respecting the correctness of the anatomy/biostructure with the imposed constraints. The authentic model generation will significantly increase the efficiency of the model for surgical simulation use. Design and modeling of such an anatomically authentic 3-D model require significant man-hours including back and forth modifications of the model between medical experts and bio-illustrators [1].

GAML uses Powell's non-linear derivative-free constrained and non-linear optimization solver called Constrained Optimization by Linear Approximation (COBYLA) [2,3] to satisfy anatomy (geometry) constraints. GAML provides linear solution times up to 100 constraints and exhibits exponential time beyond that. We proposed Partition-based Optimization Model for Generative Anatomy Modeling Language (POM-GAML) [4] in our previous work. POM-GAML successfully decreases the exponential execution time to linear time for GAML. In a scene with the twenty-five 3-D models and a total of 160,392 vertices with 5000 constraints, the speed up was calculated as 2689-fold and the normalized error was calculated as less than 0.1% using POM-GAML. However, the error reduction was not the primary focus of these previous works. The error was within the acceptable error margin and showed no noticeable differences compared with the actual results.

POM-GAML is used to partition the optimization problem into sub-problems by introducing additional virtual constraints for each sub-problem. Using various community detection algorithms, Clauset, Newman, and Moore (CNM) [5], k-means clustering [6], density peaks clustering [7], and density-based spatial clustering of applications with noise (DBSCAN) [8], we find possible candidate nodes for a split. Each split is carried out on the constraint hierarchy which is formed depending on the connectivity of joints with these constraints. After the split operation, two separate geometries and new optimization models are introduced. Partitioning is imperative in enabling the concurrent computation of the solution for the sub-problems. The generation of the sub-problems removes the dependency while retaining the original optimization problem. In large constraint sets, solving the sub-problems of a problem compared to solving the original problem decreases the exponential execution time to linear time. In POM-GAML, the solution was computed once for each sub-problem to achieve near real-time performance. The overall workflow of POM-GAML is seen in Fig. 1.

In general, regarding non-linear programming problems, there is no single algorithm that is more favorable than the other approaches [9]. Partitioning a problem is a widely applied technique; one of the well-studied approaches is the Divide and Conquer approach. Divide and Conquer creates extra computations for each sub-problem due to its recursive nature [10,11]. Anand et al. [10] use a backtracking algorithm to reduce additional computations. In this approach, until the global constraint is satisfied, sub-problems were solved recursively using backtracking and combining solutions. By eliminating redundant checks, the execution speed can be increased. Reimann et al. [12] introduced D-Ants, which applies problem decomposition to the vehicle routing problem. For efficiency, the number of clusters were predefined and the clustering was generated with the sweep algorithm [13] which then solved with the savings based on the ant system framework [14]. In regards to the performance benchmark, the D-Ants algorithm was slower than the granular tabu search [15], which uses candidate lists with tabu search [16]. Mackey et al. [17] introduced divide-factor-combine for noisy matrix factorization in an application for video background modeling context. In this work, the matrix is randomly divided into sub-problems and the sub-problems are computed in parallel. The speedup is twenty times faster than the proximal gradient algorithm, which uses unconstrained non-smooth convex optimization [18].

Furthermore, for constraint-partitioning, Burkard et al. [19] considered the partition sets as bases of matroids. Matroid is a finite set that generalizes the idea of linear independence [20]. Burkard et al. [19] extended a solution that can perform a greedy-like algorithm repeatedly for matroids. The complexity of the algorithm is $O(k^2)$ while all given matroids have the same structure with $k$ partitioning problems. In addition, when the involved matroids are not all equal, they need to find good approximation algorithms. Furthermore, certain problems, such as the matrix decomposition problem, have a special sequence for matroids, where the complexity is in polynomial time. Wah et al. [21] presented a theory of penalty methods for mixed-integer, discrete, and continuous optimization, and its application in solving temporal planning problems partitioned by the constraints. Their approach reduces the complexity of non-linear constrained planning problems. Since each sub-problem has a smaller number of constraints, their algorithm leads to sub-problems that are simpler to solve.

Another approach to solve large non-linear optimization problems is sub-space techniques [22]. Sub-space techniques are used to reduce the computation cost and memory size. According to Refs. [23], the generalized Lagrangian function method [24], modified sequential simplex pattern search [25], and the generalized reduced gradient methods [26] are noted as useful on large-scale non-linear programming problems. Sequential quadratic programming is a sub-space technique that solves non-linearly constrained problems by minimizing quadratic approximations and applying the Lagrangian function to the linearized constraints [27].

### 1.1. Contributions of this work

In this work, we introduce an iterative approach where the sub-problems are expected to converge to an original solution that significantly lowers the error in POM-GAML. In addition to the iterative approach, we tested the optimization model within the spectrum of clustering techniques and hierarchical graph parameters. We wanted to understand the response of the optimization model with distinct clusters of constraints and joints/constraints hierarchy graphs. We believed that the formation of the partitions might have an impact on the solution of the optimization model. We hypothesize that the clustering and partitioning the optimization model and then solving in an iterative manner among the partitions could further improve the error. We thereby derived and created various joint hierarchies using graph connectivity parameters to generate test cases for our hypothesis. We wanted to investigate the influence of joints/constraints hierarchy in our approach using graph parameters and tested with common clustering techniques. We experimented with constraint sets varying from 2000 to 4200 and determined the errors. We used our iterative approach on human digestive anatomy models for real-life cases as well. We tested our models used for Virtual Endoluminal Surgery Simulator (VESS) [28]for Endoscopic Submucosal Dissection (ESD) and Virtual Colorectal Surgical Trainer (VCoST) training and assessment using 4200 constraints in a scene with five 3-D models (large intestine, small intestine, stomach, liver, and spleen) with a total of 80,847 vertices.

**Table 1**
Optimization model for partitions.

| | | |
|---|---|---|
| $Arg\ Min:\ P(\sum_{l=1}^{N} k_l \lvert p_l - p_{Destination} \rvert)_t$ | | |
| *Subject to:* | | |
| $Dist_{ij} - \lvert p_i - p_j \rvert = 0$ | for (*i,j*) ∈ $A_t$ | *(1)* |
| $cos^{-1}\left(\frac{(p_{io} - p_j)\cdot(p_i - p_j)}{\lVert(p_{io} - p_j)\rVert \times \lVert(p_i - p_j)\rVert}\right) - \theta_{ij} < 0$ | for (*i,j*) ∈ $B_t$ | *(2)* |
| $(p_{io} - p_j)_{axis} - (p_i - p_j)_{axis} = 0$ | for (*i,j*) ∈ $B_t$ | *(2a)* |
| $Dist_{ij} - \Delta d_{max} - \lvert p_i - p_j \rvert < 0$ | for (*i,j*) ∈ $C_t$ | *(3)* |
| $\lvert p_i - p_j \rvert - Dist_{ij} - \Delta d_{max} < 0$ | for (*i,j*) ∈ $C_t$ | *(4)* |
| $2\mathrm{sgn}(p_i p_j . p_i v_j) \times tan^{-1}\left(\frac{r}{\lVert v_j \rVert}\right) - a_{ij} < 0$ | for (*i,j*) ∈ $D_t$ | *(5)* |
| $\left(\frac{\tan(a_{ij})}{\lVert v_j \rVert}\right) < 0$ | for (*i,j*) ∈ $D_t$ | *(6)* |
| $\lvert p_i - v_j \rvert < 0$ | for (*i,j*) ∈ $D_t$ | *(7a)* |
| $\lvert p_i - v_i \rvert = 0$ | for (*i,j*) ∈ $D_t$ | *(7b)* |
| $i,j \in J\ and\ i,j \subseteq M,\ k_l > 0, A \cap C = \emptyset,$ $P(A)m \cap P(A)n = \emptyset, v_i = p_i, v_j = p_j, v \in V, v \notin A,B,C$ | | |

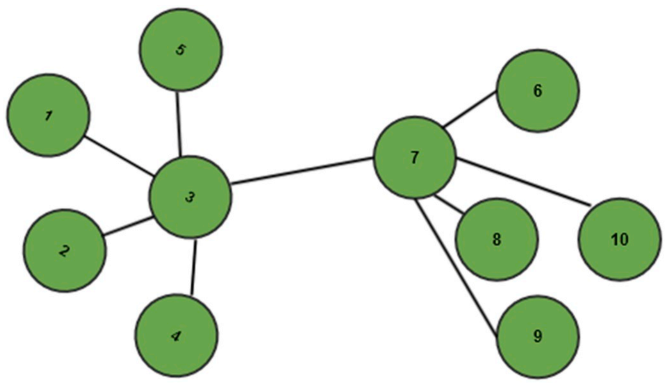

**Fig. 2.** Connected joints.

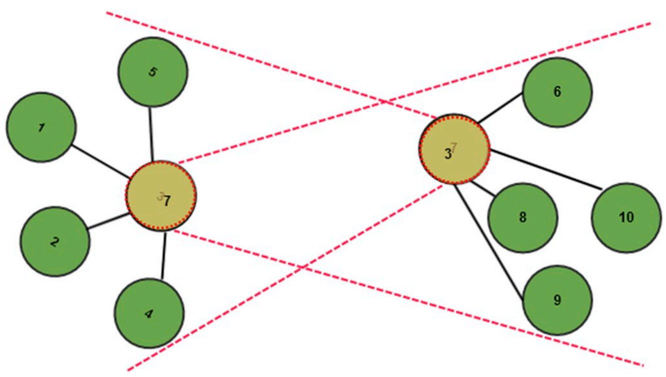

**Fig. 3.** Partition of joints. Yellow joints are virtual joints and red lines represent the possible motion of the joints.

## 2. Methods

In this section, we briefly describe the model and present the iterative approach followed by a description of graph parameterization metrics for the generation of test cases.

### 2.1. Virtual joints

Joint is an abstract term that holds the movement information and constraints for a region of the 3-D model. The dimension of a joint indicates the region of interest that can be enlarged or contracted over the 3-D model. This region affects the spatial position of the vertices when the joint undergoes any motion. Joints can be linked together to form more complex structures. The formation of these joints is represented in a hierarchical graph structure. POM-GAML [4] uses joint graphs to model anatomies or biostructures. In our joint graph, joints are used as nodes and constraints/connections are used as edges between the nodes. A joint graph can be formally defined as; $G = (V, E); E \subseteq \{(J_i, J_k) : J_i, J_j \in V\}$, where $(J_i, J_j)$ is a tuple of joints, $V$ is the vertex set, and $E$ is the edge set of 2-element subsets of the vertex set $V$.

The optimization model (as seen in Table 1) tries to compute the nearest possible location to the desired motion of each joint. With the desired location of a joint or multiple joints given, our optimization model computes the optimal solution that satisfies all the other joints' constrained motions in the joint hierarchy graph.

In our optimization model, $p_i$ is a 3-D position of a joint ($J_i$), where $J_i$ can be an arbitrary node (vertex) in or a node attached to a 3-D Mesh (*M*). Number of joints (*N*), can be dynamically modified by adding and removing joints. In between each joint couple $J_i$ and $J_j$ with corresponding 3-D $p_i$ and $p_j$ positions, there could be up to four different types of constraints (Equations 1-4 in Table 1) to restrain the movement of a node. In our model, unique constraint sets are formed for each constraint type. For instance, Equation 1 in our optimization model is defined for the absolute distance constraint and set *A* indicates the set of absolute distance constraints. Equation 2 and 2a are for the angle constraints and

**Table 2**
Distance properties and corresponding ranges.

| Distance Properties | | | | | |
|---|---|---|---|---|---|
| Constraint Set of | Radius | Diameter | Min # of Central Vertex | Min # of Peripheral vertex | Average eccentricity |
| 3000 | $x \leq 2$ | $4 \leq x \leq 6$ | $x \geq 21$ | $x \leq 5$ | $x \leq 4.0$ |
| 3300 | $5 \leq x \leq 6$ | $7 \leq x \leq 9$ | $5 \leq x \leq 10$ | $x \leq 5$ | $6.01 \leq x \leq 8.0$ |
| 3600 | $3 \leq x \leq 4$ | $7 \leq x \leq 9$ | $x \leq 4$ | $x \leq 5$ | $4.01 \leq x \leq 6.0$ |
| 3900 | $x \geq 7$ | $x \geq 10$ | $x \leq 4$ | $x \leq 5$ | $x \geq 8.01$ |
| 4200 | $5 \leq x \leq 6$ | $7 \leq x \leq 9$ | $5 \leq x \leq 10$ | $x \leq 5$ | $6.01 \leq x \leq 8.0$ |
| 2000 | $x \leq 2$ | $4 \leq x \leq 6$ | $5 \leq x \leq 10$ | $6 \leq x \leq 10$ | $x \leq 4.0$ |
| 2200 | $3 \leq x \leq 4$ | $4 \leq x \leq 6$ | $5 \leq x \leq 10$ | $6 \leq x \leq 10$ | $x \leq 4.0$ |
| 2400 | $x \leq 2$ | $x \leq 3$ | $x \geq 21$ | $11 \leq x \leq 15$ | $x \leq 4.0$ |
| 2600 | $5 \leq x \leq 6$ | $7 \leq x \leq 9$ | $11 \leq x \leq 20$ | $x \geq 16$ | $6.01 \leq x \leq 8.0$ |
| 2800 | $x \leq 2$ | $x \leq 3$ | $x \geq 21$ | $x \leq 5$ | $x \leq 4.0$ |

**Table 3**
Connection properties and corresponding ranges.

| Connection Properties | | | | |
|---|---|---|---|---|
| Constraint Set of | Characteristic Path Length | Global Efficiency | Average Local Efficiency | Clustering Coefficient |
| 3000 | $x \leq 2.0$ | $x \geq 0.71$ | $x \geq 0.61$ | $x \geq 0.61$ |
| 3300 | $3.01 \leq x \leq 4.0$ | $0.31 \leq x \leq 0.5$ | $x \leq 0.2$ | $x \leq 0.2$ |
| 3600 | $3.01 \leq x \leq 4.0$ | $0.31 \leq x \leq 0.5$ | $x \leq 0.2$ | $x \leq 0.2$ |
| 3900 | $x \geq 4.01$ | $0 \leq 0.3$ | $x \leq 0.2$ | $x \leq 0.2$ |
| 4200 | $3.01 \leq x \leq 4.0$ | $0.31 \leq x \leq 0.5$ | $x \leq 0.2$ | $x \leq 0.2$ |
| 2000 | $x \leq 2.0$ | $x \geq 0.71$ | $x \geq 0.61$ | $x \geq 0.61$ |
| 2200 | $2.01 \leq x \leq 3.0$ | $0.51 \leq x \leq 0.7$ | $0.21 \leq x \leq 0.4$ | $0.21 \leq x \leq 0.4$ |
| 2400 | $x \leq 2.0$ | $x \geq 0.71$ | $x \geq 0.61$ | $x \geq 0.61$ |
| 2600 | $3.01 \leq x \leq 4.0$ | $0.31 \leq x \leq 0.5$ | $0.41 \leq x \leq 0.6$ | $0.41 \leq x \leq 0.6$ |
| 2800 | $x \leq 2.0$ | $x \geq 0.71$ | $x \geq 0.61$ | $x \geq 0.61$ |

**Table 4**
Constraint metrics and corresponding values used.

| Constraint Properties | | |
|---|---|---|
| Constraint Set of | Constraints per Joint | # of Joints |
| 3000 | 50 | 60 |
| 3300 | 55 | 60 |
| 3600 | 60 | 60 |
| 3900 | 65 | 60 |
| 4200 | 70 | 60 |
| 2000 | 50 | 40 |
| 2200 | 55 | 40 |
| 2400 | 60 | 40 |
| 2600 | 65 | 40 |
| 2800 | 70 | 40 |

set $B$ indicates the set of angle constraints. Equations 3-4 are for the flexibility constraints and set $C$ is designated for the flexibility constraints. Sets $A_t, B_t, C_t,$ and $D_t$ indicate the partitioned sets for sets $A$, $B$, $C$, and $D$, respectively where $t$ is the partition. $Dist_{ij}$ is the original distance between joints $J_i$ and $J_j$ and $\Delta d_{max}$ is the maximum displacement allowed between the joint couple $J_i$ and $J_j$ using the stiffness ratio $k$. In Equation 2, $\theta_{ij}$ is the maximum angle that $J_i$ is allowed to pivot about $p_j$, and $p_{io}$. For the angle constraint, $p_{io}$ is the original position of $p_i$. In Equation 2a, $(p_{io} - p_j)_{axis}$ and $(p_i - p_j)_{axis}$ are the directional vectors in the x, y, or z axes to accommodate any lock along the axis of rotation. The direction vector between $p_{io}$ and pivot point of $p_j$ is $(p_{io} - p_j)$. The direction vector between $p_i$ and pivot point of $p_j$ is $(p_i - p_j)$.

We partitioned the problem into sub-problems by introducing the concept of virtual joints. Partitioning requires a split to remove the connectivity among the partitions. In each split operation, the joints at the split location are duplicated. Subsequently, the edge and the constraints between the joints are removed. The duplicated nodes will serve as virtual joints (as shown in Fig. 2 and Fig. 3). The virtual joints carry information regarding the disconnected joints and apply the constraints with other partition to its sub-problem. In the model, $V$ is the set of virtual joints, $v_i$ is the virtual joint of joint $p_i$, and $v_j$ is the virtual joint of joint $p_j$. $P_t$ is the partition and none of the joints in one partition can be shared or exist in another partition unless it is a virtual joint. In Equation 5, $r$ is the radius of the cone defined for a virtual joint, while $a_{ij}$ is the angle between joints $J_i$ and $J_j$. The angle of the cone is preset to 30°and change in preset angle shows no significant difference in the constraint sets over 50°.

## 2.2. Iterative approach

In our iterative approach, the optimization model is run for each partition. For each iteration, joints and virtual joints with updated constraint, connection, position, and partition information are passed into the optimization model. From joints and virtual joints, constraint information is gathered. The feasible region for each joint in each partition is calculated and the geometric boundaries are checked. After each iteration, properties of all joints and virtual joints such as constraint, connection, and position information are updated. Virtual joint locations are synchronized and updated at the end of each iteration at the synchronization step. The synchronization of the virtual joints is performed by averaging their current locations. After the first iteration, the 3-D location of joint $J_i$ is $p_{1i}$ and after the second iteration, the 3-D location of joint $J_i$ is $p_{2i}$, where $|p_{1i} + p_{2i}| \geq 0$. The derivative of the position and summative positional change of the partition joints with respect to the iteration can be also used for updating the virtual joint. We iteratively compute the solution where the virtual joints gradually converge to each other. For the constraint sets varying from 2000 to 4,200, we have utilized our iteration-based method. The next section describes these joint graph parameters that are used to generate various complexities that enable us to test and validate our iterative approach.

To summarize our approach, we traverse through our joint hierarchy and use various community detection/clustering approaches to seek possible candidate nodes for partitioning. Once candidate nodes are determined, they are duplicated in the hierarchy. This duplication generates virtual nodes. The virtual nodes contain information regarding disconnected joints and mimic the original node by imposing original constraints. The duplication and followed by the split operation continues until the constraints between the partitions are completely separated. These operations will result in separate partitions where no nodes are shared in their respective joint hierarchies. In the iterations, solutions to each sub-problem; optimization problem pertaining to each partition, is solved separately and iteratively. The synchronization, information exchange phase through virtual joints, is performed at each iteration to reduce the overall error.

## 2.3. Graph parameters

For testing purposes, we require a variety of joint hierarchies to analyze the impact of the joints and constraints distribution over the optimization model. We aim to perform benchmark tests using these hierarchies to quantify the performance and error. The motivation of creating these cases has two folds; we want to eliminate any bias in the test cases, where the model could coincidentally perform well in one hierarchy and poor on the other one. Secondly, we want to ensure that

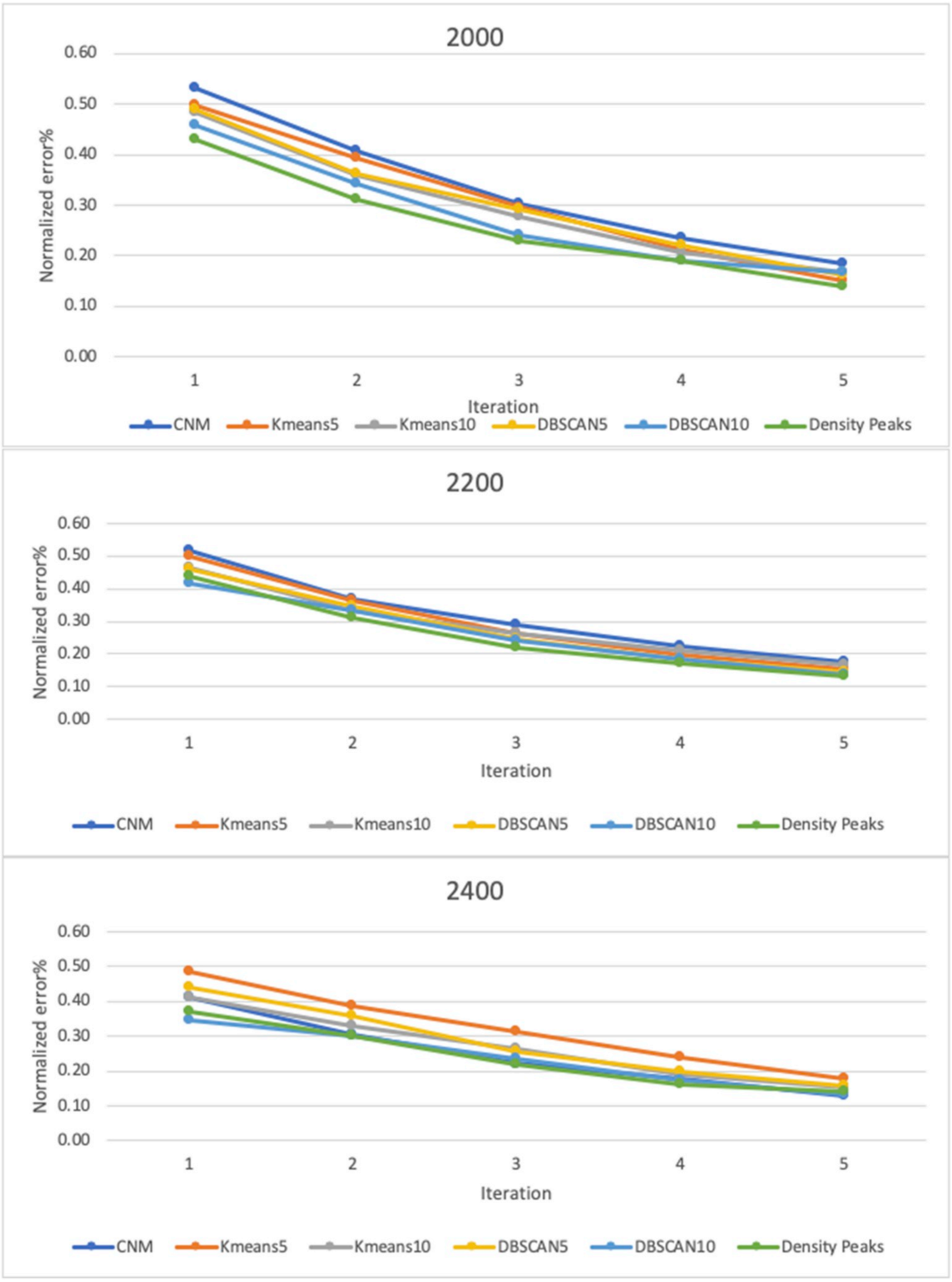

**Fig. 4.** Normalized iteration graphs for constraint sets 2000, 2200, and 2400.

the optimization model can be applicable regardless of the hierarchy formation. We also want to analyze the error and error distribution respecting hierarchy generation parameters.

We used graph generation parameters for automatically generating joint hierarchies. These parameters are categorized into three metrics; distance, connection, and constraint. For each parameter, four different ranges (e.g. easy, mild, moderate, and difficult complexity terms used for readability) were used to generate increasingly complex constraint graphs. We will briefly describe these metrics.

### 2.3.1. *Distance metrics*

Radius [29], diameter [30], minimum number of central vertices [31], minimum number of peripheral vertices [32], and average eccentricity [33] were used as distance metrics. Distance metrics were derived from the eccentricity of each vertex. The eccentricity $\varepsilon(V)$ of a vertex $V$ is the longest distance between $V$ and any other joint node in the graph. The radius $r$ of a graph is the minimum eccentricity of any node, $r = \min(\varepsilon(V))$. The diameter $d$ of a graph is the maximum eccentricity of any joint node in the graph, $d = \max(\varepsilon(V))$. To find the diameter of a graph, we find the shortest path between each pair of nodes. The greatest length of any of these paths is the diameter of the graph. A central node in a graph of radius $r$ is one whose eccentricity is $r$ and that is a node that achieves the radius such that $\varepsilon(V) = r$. Peripheral nodes are defined as the nodes that are $d$ distance away from some other node. Formally, $V$ is peripheral if $\varepsilon(V) = d$. Table 2 shows the distance properties and corresponding ranges.

### 2.3.2. *Connection metrics*

Characteristic path length [34], global efficiency [35], average local efficiency [36] and clustering coefficient [37,38] were used as graph

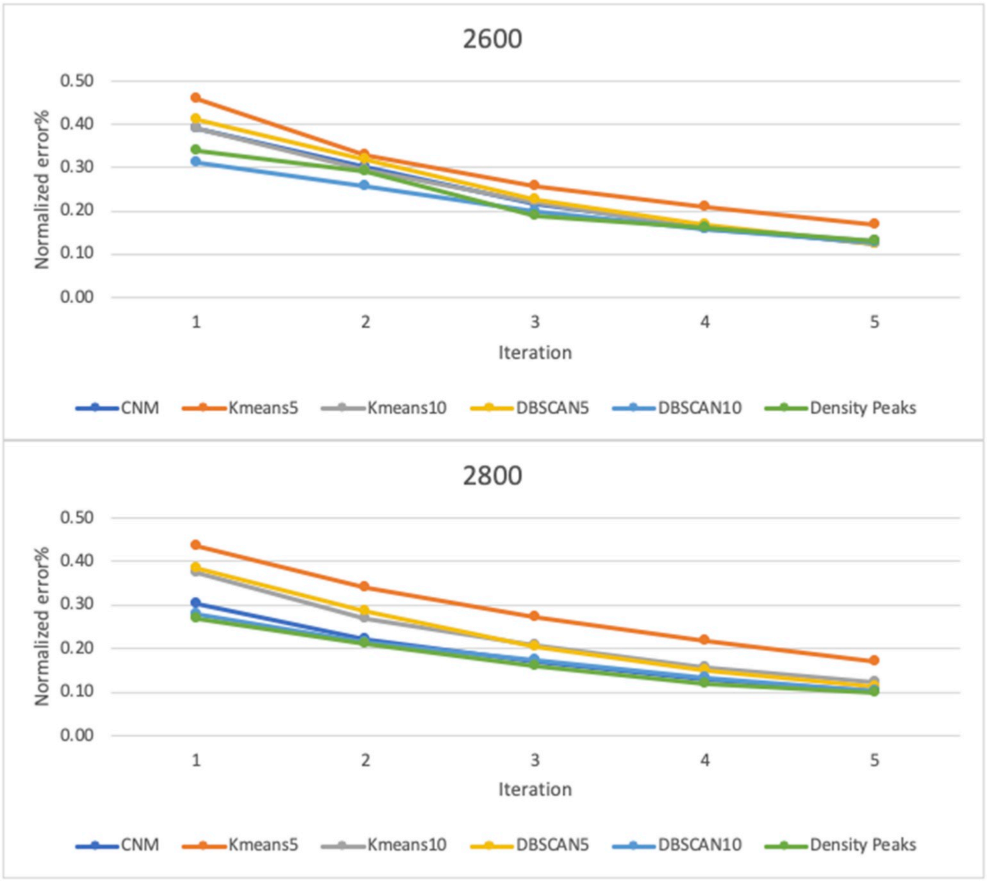

**Fig. 5.** Normalized iteration graphs for constraint sets 2600 and 2800.

creation metrics. The characteristic path length (or the average path length) is the average of all the distances between the node pairs( $(V_i, V_j)$, $i \neq j$ ) in the graph, where $i$ and $j$ are nodes in the graph. The global efficiency is the average of all the reciprocals of the non-zero distances in a graph. The local efficiency is the average efficiency of the local subgraphs. Efficiency is defined as $\varepsilon_{i,j} = \frac{1}{distance(i,j)}$, $i \neq j$, [39,40]. A clustering coefficient is a degree of connectivity measure of two nodes that are connected to the same node. Clustering coefficient of a graph is the average of the local clustering coefficients for all nodes in the graph. The clustering coefficient is defined as $C = \frac{1}{N}\sum_{n \in N} \frac{y_n}{\binom{d_n}{2}}$, where $N$ is the list of all nodes, $y_n$ is the number of links between neighbors of a node $n \in N$, and $d_n$ is the degree of a node $n \in N$. Table 3 shows the connection metrics and corresponding ranges.

### 2.3.3. *Constraint metrics*

In GAML, a joint holds constraint and attachment information. Constraints per joint and number of joints were used as constraint metrics for the graph generation. The numbers used in constraint metrics are derived from our previous empirical tests where the average constraints per joints tabulated here were sufficient for modeling complex human anatomy. The number of joints were set to 40 and 60, while constraints per joint were modified from 50 to 70. The reason is to determine graph test cases for the number of joints and constraints per joints separately. Table 4 shows the constraint metrics and corresponding values.

## 2.4. *Clustering and community detection*

We use hierarchical joint graph structure to partition the optimization model. Before we partition the problem, we determine the optimal location for the splitting operation. The assumption here is that in a graph the joint communities with higher constraint density could be good candidates for model partition. As a result, we used common partition algorithms such as; CNM [5], k-means clustering [6], density peaks clustering [7], and DBSCAN [8] to detect clusters/communities. We would like to note that we use cluster and community terms interchangeably to stay with the terminology in both graph and data mining disciplines.

CNM is an agglomerative hierarchical method based on greedy optimization. In CNM, modularity is used as a measure to calculate the strength of communities. Modularity calculates the divisibility of a community by checking the ratio of the number of edges in each community to the edges between the communities. Density peaks clustering is a density-based clustering approach. In density peaks, it is assumed that cluster centers are away from other points with high densities and cluster centers have a higher density than their neighbors. For each data point in density peaks, we used two parameters local density($\rho$) and the distance of each point from points belonging to higher density. The distance parameter was populated from the connectivity of joints in the graph. Another clustering approach used is DBSCAN. DBSCAN uses parameters (MINPTS, ε, DISTFUNC) to find highly dense areas of joints. MINPTS expresses the amount of joints needed in the radius (ε) to specify the area as a high density area. DISTFUNC is the distance function between joints. In our case, we used 2 for MINPTS to minimize noise, we selected 5 (DBSCAN5) and 10 (DBSCAN10) for the radius(ε) of connectivity of joints in the graph, and we used connectivity between joints as the DISTFUNC. The last technique used for joint clustering is k-means. In the k-means algorithm, n-dimensional data is partitioned into k clusters. Cluster number “k” was set to the same number of clusters as DBSCAN clusters.

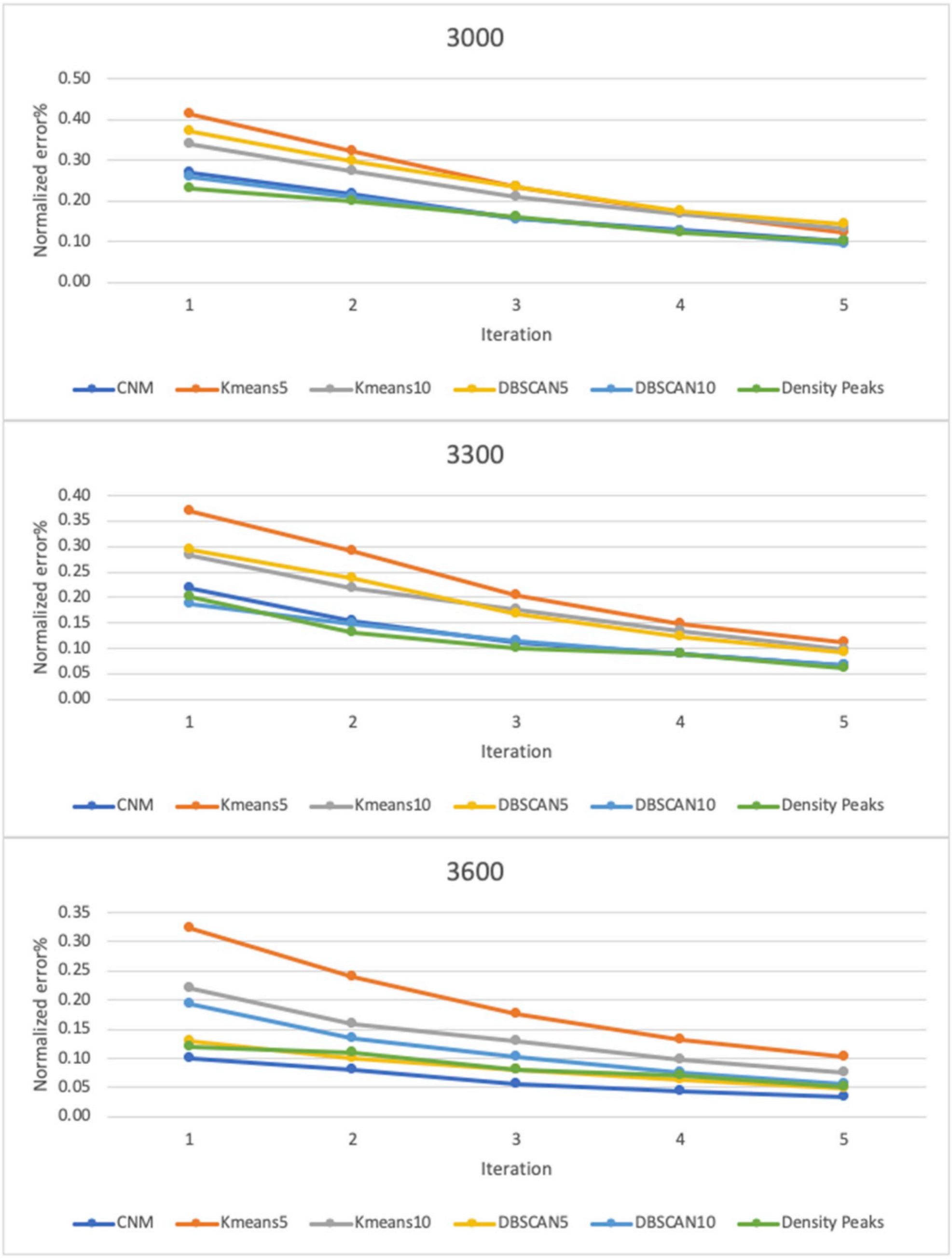

**Fig. 6.** Normalized iteration graphs for constraint sets 3000, 3300, and 3600.

After the graph structure is created for the scene, the communities need to be detected before the partitioning and iteration. For the selected community detection/clustering algorithm, parameters are given as an input. Once the communities/clusters are detected, the edges between the communities are deleted and virtual joints are added for each separated joint couple.

## 3. Results

### 3.1. Time and error

For each constraint set and clustering/community detection algorithm, we performed five iterations. We observed that further iterations beyond five do not significantly decrease the error percentage. Each iteration was performed five times to determine the average computation time. The magnitude of the test scenes was 40.86-unit distance, which was used to compute the normalized error. The normalized error was computed with regard to the dimension of the scene (Equation (8)). In Equation (8), *Partition* is the 3-D position of the joints using the partitions, *NonPartition* is the 3-D position of the joints without the partitions, and *SceneMag* is the scene magnitude.

$$\%Normerror = Avg\left(\left|\frac{\frac{Partition-NonPartition}{NonPartition}}{SceneMag}\right|_i\right)x100 \tag{8}$$

The normalized error in all constraint sets and all clustering/community detection algorithms decreased as the number of iterations has increased (as seen in Figs. 4–7). In the first iteration for the smallest constraint set, constraint set of 2,000, the normalized error varied from 0.53% to 0.46%. The highest normalized error was for CNM while DBSCAN10 had the lowest normalized error. At the fifth and final

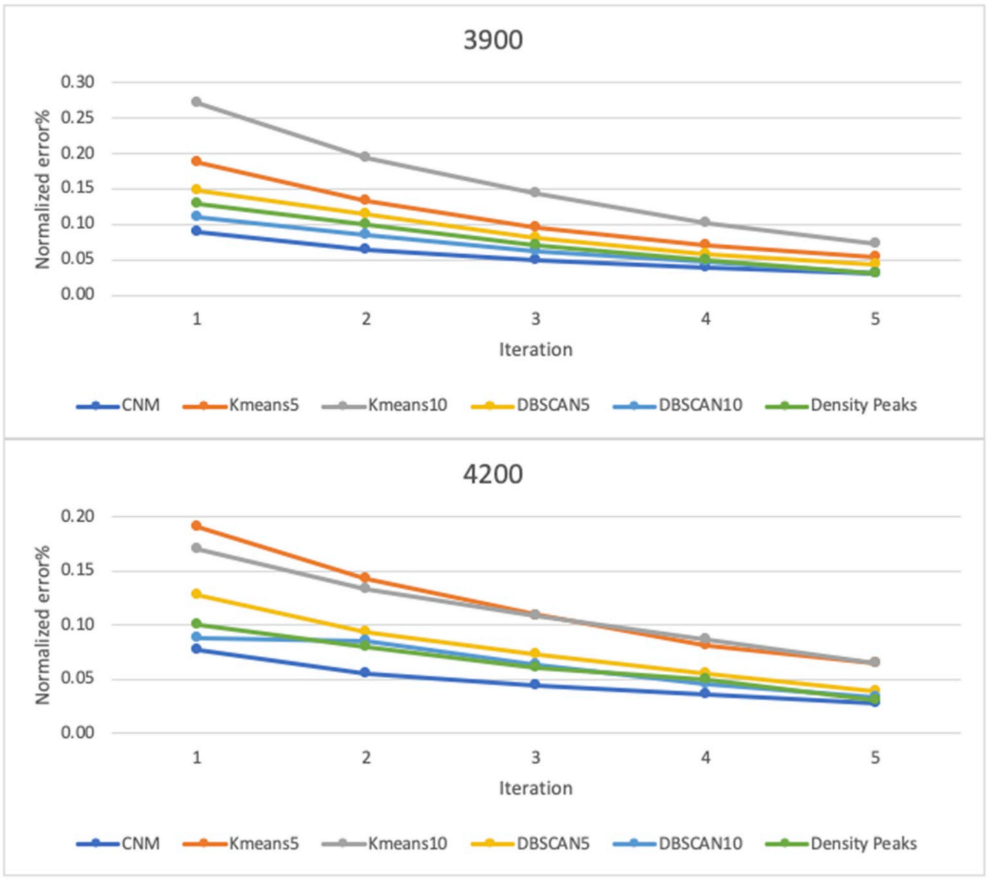

**Fig. 7.** Normalized iteration graphs for constraint sets 3900 and 4200.

**Table 5**
The decrease in error percentage from one to five iterations.

| | Decrease in error percentage (%) | | | | | |
|---|---|---|---|---|---|---|
| Constraint Set of | CNM | k-means5 | k-means10 | DBSCAN5 | DBSCAN10 | Density Peaks |
| 3000 | 63.07 | 70.53 | 60.85 | 61.64 | 63.76 | 56.50 |
| 3300 | 69.07 | 70.04 | 65.27 | 68.24 | 64.78 | 70.00 |
| 3600 | 67.18 | 68.54 | 65.20 | 63.23 | 71.39 | 58.30 |
| 3900 | 65.57 | 71.17 | 73.34 | 70.13 | 71.35 | 76.92 |
| 4200 | 63.16 | 66.38 | 61.76 | 69.75 | 63.26 | 70.00 |
| 2000 | 65.19 | 69.49 | 65.72 | 66.80 | 63.25 | 67.44 |
| 2200 | 66.13 | 69.08 | 64.20 | 68.39 | 66.90 | 70.45 |
| 2400 | 68.72 | 63.71 | 62.74 | 64.50 | 61.36 | 62.16 |
| 2600 | 68.37 | 63.31 | 67.05 | 70.13 | 59.08 | 61.76 |
| 2800 | 65.85 | 60.75 | 67.40 | 70.88 | 63.03 | 62.96 |

**Table 6**
Speed-up(x-times) for each community detection/clustering algorithm compared to non-partitioned performance.

| Constraint Set of | Community Detection/Clustering Algorithms | | | | | |
|---|---|---|---|---|---|---|
| | CNM | k-means5 | k-means10 | DBSCAN5 | DBSCAN10 | Density Peaks |
| 3000 | 16.12 | 210.18 | 324.97 | 187.79 | 9.17 | 106.35 |
| 3300 | 12.76 | 290.30 | 26.82 | 223.32 | 5.12 | 5.59 |
| 3600 | 16.43 | 285.09 | 20.75 | 241.57 | 3.04 | 8.38 |
| 3900 | 81.67 | 219.89 | 18.18 | 196.12 | 2.68 | 7.45 |
| 4200 | 124.78 | 278.20 | 24.98 | 143.42 | 2.99 | 7.79 |
| 2000 | 9.52 | 99.63 | 31.79 | 96.32 | 18.77 | 43.62 |
| 2200 | 7.36 | 89.50 | 29.48 | 121.84 | 19.91 | 29.44 |
| 2400 | 8.38 | 100.28 | 25.82 | 124.63 | 14.82 | 23.58 |
| 2600 | 9.02 | 106.60 | 28.67 | 130.74 | 10.82 | 17.35 |
| 2800 | 10.54 | 131.66 | 28.30 | 8.87 | 138.30 | 38.73 |

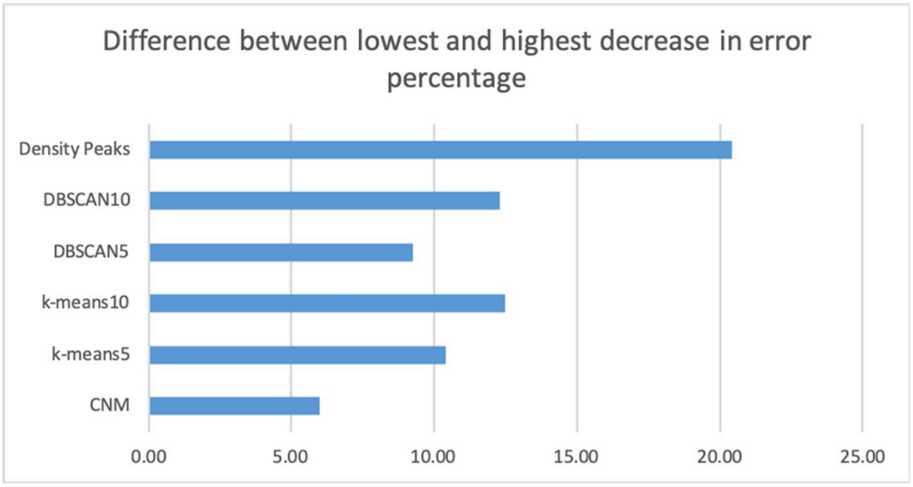

**Fig. 8.** Differences between the lowest and highest decrease in error percentage for each clustering/community detection algorithm.

**Table 7**
Strong Pearson correlations of graphs parameters with the error.

| Algorithm | Graph Parameter | Iteration | | | | |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| k-means5 | Diameter | −0.83(0.003) | −0.87(0.001) | −0.89(0.0006) | −0.87(0.001) | −0.84(0.0024) |
| | Average Eccentricity | −0.78(0.0078) | −0.82(0.0037) | −0.83(0.0029) | −0.81(0.0045) | −0.77(0.0092) |
| DBSCAN5 | Diameter | −0.78(0.0078) | −0.78(0.0078) | −0.80(0.0055) | −0.80(0.0055) | −0.81(0.0045) |
| | Global efficiency | 0.79(0.0065) | 0.78(0.0078) | 0.80(0.0055) | 0.81(0.0045) | 0.83(0.0029) |
| | Average Local Efficiency | 0.77(0.0092) | 0.77(0.0092) | 0.80(0.0055) | 0.81(0.0045) | 0.81(0.0045) |
| | Clustering Coefficient | 0.79(0.0065) | 0.79(0.0065) | 0.82(0.0037) | 0.82(0.0037) | 0.82(0.0037) |
| DBSCAN10 | Diameter | −0.75(0.012) | −0.77(0.0092) | −0.80(0.0055) | −0.80(0.0055) | −0.77(0.0092) |
| Density Peaks | Average Local Efficiency | 0.80(0.0049) | 0.82(0.0035) | 0.78(0.0083) | 0.77(0.0087) | 0.79(0.0072) |
| | Clustering Coefficient | 0.81(0.0045) | 0.83(0.0033) | 0.78(0.0076) | 0.79(0.0071) | 0.80(0.0058) |

*P*-values indicated in parenthesis.

iteration, the normalized error varied between 0.18% (CNM) to 0.15% (k-means5). After five iterations, the constraint set of 2000 constraints was the only constraint set that k-means5 had the lowest error percentage. Even though k-means5 had the highest error percentages in the first iteration for the constraint sets 2,400, 2,600, 2,800, 3,000, 3,600, and 4,200, in constraint sets 2,000, 3,000, and 3300 k-means5 had the largest decrease in error percentage (as seen in Table 6). The results for k-means5 showed that having the highest error percentage for a specific constraint set (2,400, 2,600, 2,800, 3,000, 3,600, and 4200) did not account for the largest decrease in error percentage in the constraint sets (2,000, 3,000, and 3300). Table 5 shows the decrease in error percentage in the iterations starting from one to five. Iteration one is similar to the results noted in our previous work, POM-GAML [4].

In the fifth iteration for constraint sets, 3,300, 3,900, and 4,200, CNM and DBSCAN10 had the same error percentages, 0.07%, 0.03%, and 0.03% respectively. In the first iteration for the largest constraint set; constraint set of 4,200, the normalized error varied from 0.19% to 0.08%. The highest normalized error was for k-means5 while CNM had the lowest normalized error. The normalized error varied between 0.03% (CNM and DBSCAN10) to 0.07% (k-means10). Out of all clustering/community detection algorithms CNM had the smallest difference between the lowest and highest decrease in error percentage with 6%, while density peaks had the highest difference with 20.42% (as seen in Fig. 8). In the fifth iteration, in the constraint sets; 2,600, 2,800, 3,900, and 4,200, DBSCAN10 and density peaks had the same error percentages 0.10%, 0.13%, 0.03%, and 0.03% respectively.

For time analysis, an Intel Core i7-5820 K CPU with 16 GB RAM and a GeForce GTX 970 GPU with the driver version 419.67 was used. Table 7 shows the speed-up times for each partition algorithm for each constraint set. The maximum overall speed-up (compared to non-partitioned performance) was achieved with k-means5 for the constraint sets of 3,300, 3,600, 3,900, 4,200, and 2,000, with k-means10 for the constraint set of 3,000, with DBSCAN5 for constraint sets of 2,200, 2,400, and 2,600, and with DBSCAN10 for the constraint set of 2800 (as seen in Table 6).

## 3.2. *Parameter analysis*

We aim to understand the correlation between graph parameters and computed errors to recognize model behavior for large constraints. We run the Pearson's correlation test for each graph parameter against the normalized error for each constraint set and iteration. For the Pearson correlation test, $r$ value is the correlation coefficient and $p$ value is the significance. According to our results, the minimum number of central and peripheral nodes did not show a strong correlation to the normalized error results. Pearson's correlation values, $r$, for the minimum number of central nodes varied between 0.32 ($p = 0.367$) to 0.656 ($p = 0.039$) while the minimum number of peripheral nodes varied between 0.51($p = 0.132$) to 0.65 ($p = 0.042$). The rest of the parameters had at least an $r$ value of 0.6 or −0.6 against the normalized error results. For k-means5 normalized error results, diameter, and average eccentricity had a strong negative correlation for each iteration (as seen in Table 7). For DBSCAN5 normalized error results, global efficiency, average local efficiency, and clustering coefficient had a strong positive correlation, while diameter had a strong negative correlation for each iteration (as seen in Table 7). For DBSCAN10, normalized error results and diameter had a strong negative correlation for each iteration (as seen in Table 7). For density peaks normalized error results, average local efficiency, and clustering coefficient had a strong positive correlation for each iteration (as seen in Table 7).

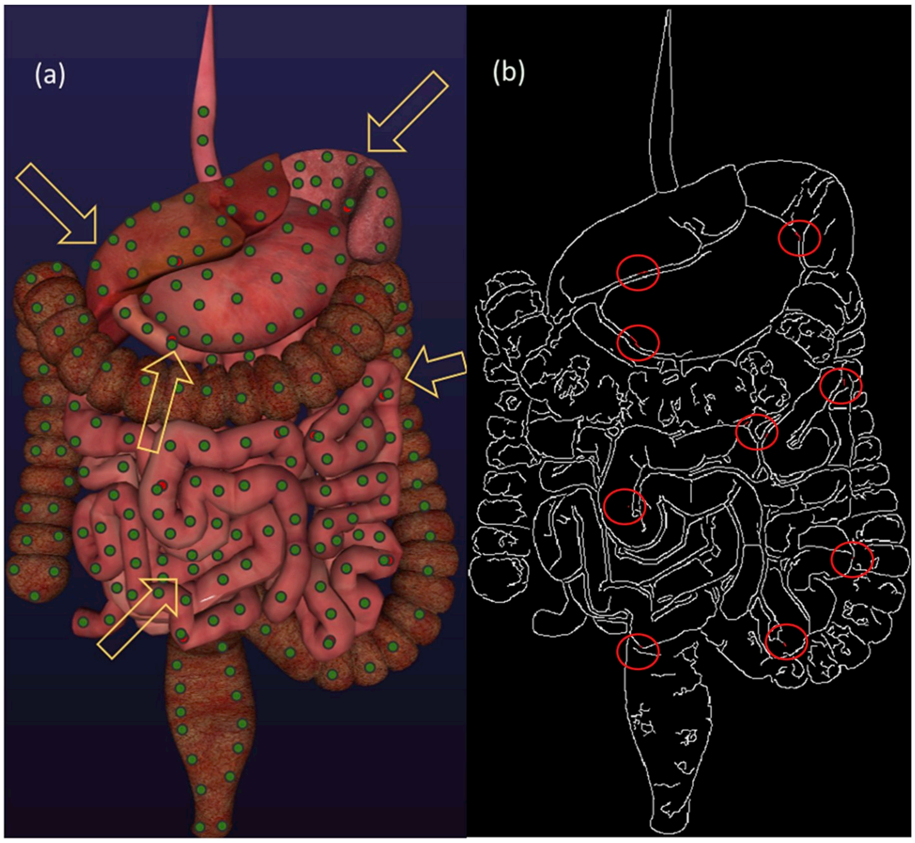

**Fig. 9.** (a) 3-D scene with joints after solving with five-iteration partitioned and non-partitioned optimization model, (b) Scene error comparison using the Canny edge detection algorithm.

## 4. Discussion

Based on our results, the diameter has a significant impact on the error. We believe that efficient partitioning of the joints is possible with a bigger diameter. As seen in the results section, k-means5, DBSCAN5, and DBSCAN10 have strong negative correlations between normalized error and diameter. Even though not strong, k-means10 (average r = −0.73), CNM (average r = −0.72), and density peak (average r = −0.71) algorithms all have negative correlations. Another graph parameter, clustering coefficient, shows the degree of nodes in a graph that is inclined to cluster together. Between normalized error and clustering coefficient, density peaks and DBSCAN5 showed strong positive correlations, while CNM (average r = 0.69), k-means5 (average r = 0.79), k-means10 (average r = 0.74), and DBSCAN10 (average r = 0.73) showed positive correlations. These results imply that proper partitioning can significantly reduce the error.

Different variations of anatomies in a virtual surgical simulator allow for surgeons to practice on a variety of difficult surgical scenarios. These variations are generated with our approach introduced in this study to ensure anatomical relevance. The digestive anatomy was constructed for our surgery simulations for ESD training and assessment, VESS [41] and VCoST. ESD is an endoscopic technique for en bloc resection of gastrointestinal lesions bigger than >20 mm [28]. VCoST is a virtual trainer for colorectal surgery skills. We created a virtual scene of the digestive anatomy and attached joints equivalent to 4200 constraints. The scene had five 3-D models (large intestine, small intestine, stomach, liver, and spleen) related to the human digestive anatomy with a total of 80,847 vertices. We executed nodal transformations for each model in the scene. Fig. 9a shows the 3-D scene with the joints after motion using the partitioned iterative approach and non-partitioned optimization models. Arrows in Fig. 9a show the transformation motion (direction of the intended transformation), green circles represent five-iteration partitioned and red circles represent non-partitioned solver results. The visible red circle indicates an error at that location. The visual errors between our approach and non-partitioned solver results are not clearly noticeable to the eye. Therefore, we used the Canny edge detection algorithm [42] using MATLAB and marked the errors in the scene with red (also with red circles to point them out) as seen in Fig. 9b.

## 5. Conclusion

In this work, we introduced an iterative approach and performance benchmark for a Partition-based Optimization Model for Generative Anatomy Modeling Language (POM-GAML). POM-GAML is an anatomy modeling language that incorporates and solves a non-linear optimization model to create anatomically correct structures when the geometry undergoes any modifications. The optimization model in POM-GAML fulfills any requested variations by satisfying geometric constraints. This model can be used in any biological structure other than human anatomy. We experimented our approach with hierarchical graphs that are computationally generated cases representing various joint formations. Our results showed that as the iteration amount increases, the solution converges to the original solution as the normalized error monotonically decreases. We have utilized our approach with four distinct clustering/community detection algorithms (CNM, density peaks, DBSCAN, and k-means) to analyze partition formation with regards to error reduction and speed-up. In graph parameter analysis, we discovered results showing a correlation between parameters and normalized error results. We also further tested our iterative approach for modeling 3-D human digestive anatomy for our VESS and VCoST surgery simulators. As a future work of our study, we plan to evaluate our approach with more comprehensive graph clustering approaches such as Mean-Shift, Highly Connected Subgraphs, Affinity Propagation, with further graph complexity parameters such as assortativity, coreness, cliques, etc. We will also further validate the effectiveness of POM-GAML with other regions such as shoulder, knee, elbow, etc. physiology where anatomy comprises many cartilages, joints and very elastic membranes.

## Declaration of competing interest

The authors declare that they have no conflict of interest.

## Acknowledgment

This publication was made possible by the Arkansas INBRE program, supported by a grant from the National Institute of General Medical Sciences, P20 GM103429 from the National Institutes of Health. This project was also supported by NIH/NIAMS R44AR075481-01, NIH/NCI 5R01CA197491 and NIH/NHLBI NIH/NIBIB 1R01EB025241, R56EB026490.

## Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.compbiomed.2020.103695.

## References

[1] D. Demirel, A. Yu, S. Baer-Cooper, T. Halic, C. Bayrak, Generative anatomy modeling language (GAML), Int. J. Med. Robot. Comput. Assist. Surg. 13 (4) (2017) e1813.
[2] M.J. Powell, A fast algorithm for nonlinearly constrained optimization calculations, Numerical analysis (1978) 144–157. Springer.
[3] M.J. Powell, A direct search optimization method that models the objective and constraint functions by linear interpolation, in: Advances in Optimization and Numerical Analysis, Springer, 1994, pp. 51–67.
[4] D. Demirel, B. Cetinsaya, T. Halic, S. Kockara, S. Ahmadi, Partition-based optimization model for generative anatomy modeling language (POM-GAML), BMC Bioinf. 20 (2) (2019) 105.
[5] A. Clauset, M.E. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (6) (2004), 066111.
[6] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability 1, 1967, pp. 281–297.
[7] A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, Science 344 (2014) 1492–1496, 6191.
[8] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, 1996, pp. 226–231.
[9] D.M. Himmelblau, Applied Nonlinear Programming, McGraw-Hill Companies, 1972.
[10] S. Anand, W.-N. Chin, S.-C. Khoo, A lazy divide and conquer approach to constraint solving, in: 14th IEEE International Conference on Tools with Artificial Intelligence, 2002. (ICTAI 2002). Proceedings, 2002, pp. 91–98.
[11] C.-J. Tsai, A.K. Katsaggelos, Dense disparity estimation with a divide-and-conquer disparity space image technique, IEEE Trans. Multimed. 1 (1) (1999) 18–29.
[12] M. Reimann, K. Doerner, R.F. Hartl, " D-ants, Savings based ants divide and conquer the vehicle routing problem, Comput. Oper. Res. 31 (4) (2004) 563–591.
[13] B.E. Gillett, L.R. Miller, A heuristic algorithm for the vehicle-dispatch problem, Oper. Res. 22 (2) (1974) 340–349.
[14] K. Doerner, M. Gronalt, R.F. Hartl, M. Reimann, C. Strauss, M. Stummer, SavingsAnts for the vehicle routing problem, in: Workshops on Applications of Evolutionary Computation, 2002, pp. 11–20.
[15] P. Toth, D. Vigo, The granular tabu search and its application to the vehicle-routing problem, Inf. J. Comput. 15 (4) (2003) 333–346.
[16] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. 13 (5) (1986) 533–549.
[17] L.W. Mackey, M.I. Jordan, A. Talwalkar, Divide-and-conquer matrix factorization, Adv. Neural Inf. Process. Syst. (2011) 1134–1142.
[18] K.-C. Toh, S. Yun, An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems, Pac. J. Optim. 6 (2010) 15, 615–640.
[19] R.E. Burkard, E.-Y. Yao, Constrained partitioning problems, Discrete Appl. Math. 28 (1) (1990) 21–34.
[20] D.L. Neel, N.A. Neudauer, Matroids you have known, Math. Mag. 82 (1) (2009) 26–41.
[21] B.W. Wah, Y. Chen, Constraint partitioning in penalty formulations for solving temporal planning problems, Artif. Intell. 170 (3) (2006) 187–231.
[22] Y. Yuan, Subspace techniques for nonlinear optimization, in: Some Topics in Industrial and Applied Mathematics, World Scientific, 2007, pp. 206–218.
[23] F.A. Tillman, C.-L. Hwang, W. Kuo, Optimization techniques for system reliability with Redundancy\A review, IEEE Trans. Reliab. 26 (3) (1977) 148–155.
[24] H. Everett III, Generalized Lagrange multiplier method for solving problems of optimum allocation of resources, Oper. Res. 11 (3) (1963) 399–417.
[25] M.J. Box, A new method of constrained optimization and a comparison with other methods, Comput. J. 8 (1) (1965) 42–52.
[26] L.S. Lasdon, R.L. Fox, M.W. Ratner, Nonlinear optimization using the generalized reduced gradient method, Rev. Fr. Autom. Inform. Rech. Opérationnelle Rech. Opérationnelle 8 (V3) (1974) 73–103.
[27] Y. Yuan, A review on subspace methods for nonlinear optimization, in: Proceedings of the International Congress of Mathematics, 2014, pp. 807–827.
[28] B. Cetinsaya, M.A. Gromski, S. Lee, Z. Xia, D. Demirel, T. Halic, C. Bayrak, C. Jackson, S. De, S. Hegde, J. Cohen, M. Sawhney, S.N. Stavropoulos, D.B. Jones, A task and performance analysis of endoscopic submucosal dissection (ESD) surgery, Surg. Endosc. 33 (2) (Feb. 2019) 592–606.
[29] J.M. Hernández, P. Van Mieghem, Classification of Graph Metrics, Delft Univ. Technol. Mekelweg Neth., 2011, pp. 1–20.
[30] R. Albert, H. Jeong, A.-L. Barabási, Internet: diameter of the world-wide web, Nature 401 (6749) (1999), 130.
[31] R.C. Entringer, D.E. Jackson, D.A. Snyder, Distance in graphs, Czech. Math. J. 26 (2) (1976) 283–296.
[32] G. Chartrand, D. Erwin, G.L. Johns, P. Zhang, Boundary vertices in graphs, Discrete Math. 263 (1–3) (2003) 25–34.
[33] P. Van Mieghem, Performance Analysis of Communications Networks and Systems, Cambridge University Press, 2009.
[34] M.A. Nascimento, J. Sander, J. Pound, "Analysis of SIGMOD's co-authorship graph, ACM Sigmod Rec. 32 (3) (2003) 8–10.
[35] J.T. Pastor, J.L. Ruiz, I. Sirvent, An enhanced DEA Russell graph efficiency measure, Eur. J. Oper. Res. 115 (3) (1999) 596–607.
[36] S. Achard, E. Bullmore, Efficiency and cost of economical brain functional networks, PLoS Comput. Biol. 3 (2) (2007) e17.
[37] D.J. Watts, S.H. Strogatz, "Collective dynamics of 'small-world'networks, nature 393 (6684) (1998) 440.
[38] T. Schank, D. Wagner, Approximating Clustering-coefficient and Transitivity. Universität Karlsruhe, Fakultät für Informatik, 2004.
[39] V. Latora, M. Marchiori, Efficient behavior of small-world networks, Phys. Rev. Lett. 87 (19) (Oct. 2001) 198701.
[40] B. Ek, C. VerSchneider, D.A. Narayan, Global efficiency of graphs, AKCE Int. J. Graphs Comb. 12 (1) (Jul. 2015) 1–13.
[41] B. Cetinsaya, M.A. Gromski, S. Lee, Z. Xia, M. Turkseven, D. Demirel, T. Halic, C. Bayrak, C. Jackson, S. De, Design of virtual endoluminal surgery simulator (VESS): colorectal endoscopic submucosal dissection training module, in: American Journal of Gastroenterology vol. 112, 2017, pp. S452–S453.
[42] J. Canny, A computational approach to edge detection, in: Readings in Computer Vision, Elsevier, 1987, pp. 184–203.