Conjugate Product Graphs for Globally Optimal 2D-3D Shape Matching
– Appendix –

Paul Roetzer\(^1\) Zorah Lähner\(^2\) Florian Bernard\(^1\)
University of Bonn\(^1\) University of Siegen\(^2\)

A1. Segmentation Pre-Matching

In Fig. A1, we visualise the pre-matching which is used by the approach in [3]. It is obvious that the injection of such information into the objective function makes finding valid solutions substantially easier.

Figure A1. Visualisation of pre-matched segmentation information on cat from the TOSCA dataset. Different colours encode different segments, while darkest blue encodes the transition between different segments.

For a fair comparison, in addition to our method that does not use this information, we also evaluate "Ours-Seg.", in which we incorporate the above segmentation information as an additional feature descriptor, see [3] for details.

A2. Branch and Bound Algorithm

Algorithm A1 describes our optimisation strategy. We adapt the branch and bound algorithm introduced in [3] to conjugate product graphs and implement runtime improvements by increasing chances of finding tighter upper bounds earlier.

The final goal of the optimisation is to find a cyclic path with minimal cost. In general, this requires to solve a total of \(2|E_N| + |V_N|\) (ordinary) shortest path problems, and is computationally more expensive than the branch-and-bound strategy that we pursue.

The main idea of branch-and-bound is to iteratively subdivide the search space, while tightening upper and lower bounds using the results of previous iterations. In that sense, instead of searching for shortest paths from each vertex on the first layer to each respective vertex on the last layer, we search for the shortest path from a set of vertices \(B \subset V^*\) on the first layer to the respective set of vertices \(B\) on the last layer, see Fig. A2 (left). There is no guarantee that the path \(C = (v_1^*, \ldots, v_{|C|}^*)\) from \(B\) (first layer) to its duplicate \(B\) (last layer) with minimal energy is indeed cyclic, i.e. that the final vertex \(v_{|C|}^*\) in the last layer is indeed the same as the starting vertex \(v_1^*\) in the first layer. If \(C\) is not cyclic, we partition \(B\) into smaller, disjunct subsets \(B_1\) and \(B_2\) (with \(B_1 \cup B_2 = B\) and \(B_1 \cap B_2 = \emptyset\)) until a cyclic path is found (this is the branching strategy, see Fig. A2). The partitioning is done by calculating Voronoi cells around edges \(e_N^1\) and \(e_N^c\) on 3D shape assuming \(e_N^1\) and \(e_N^c\) are not identical (where the conjugate product vertex \(v_1^* = (e_1^M, e_N^1)\) contains edge \(e_N^1\) on 3D shape and conjugate product vertex \(v_{|C|}^* = (e_{|C|}^M, e_{|C|}^N)\) contains edge \(e_{|C|}^N\) on 3D shape). If \(e_N^1\) and \(e_{|C|}^N\) are identical we partition according to \(B_1 = B \setminus \{v_{|C|}^*\}\) and \(B_2 = \{v_{|C|}^*\}\).

The path cost \(d_C\) of non-cyclic paths (i.e. \(v_1^* \neq v_{|C|}^*\)) is a lower bound \(b(\cdot)\) on the path cost of the globally optimal cyclic path. Whenever \(v_1^*\) and \(v_{|C|}^*\) are equal (meaning that \(C\) is a cyclic path), an upper bound \(b_{upper}\) is found, which might already be the globally optimal path, but can only be identified as such if all other branches do not yield cyclic paths with lower costs. Hence, the algorithm has to explore all other branches, which in the worst case are as many as there are vertices on one layer (i.e. \(2|E_N| + |V_N|\) many).

While searching for the optimal path, the algorithm only explores paths with cost \(d_C < b_{upper}\) and thus performance can be improved if tighter upper bounds \(b_{upper}\) are found as early as possible. We improve the branch-and-bound algorithm of [3] by computing all paths \(c_{all}\) of a branch, and then
Figure A2. Illustration of the branching strategy in Algorithm A1. First, the shortest path from all vertices in \( B \) on the first layer to the same vertices on the duplicate first layer (which amounts to the last layer) is computed. The resulting shortest path from \( B \) on the first layer to \( B \) on the last layer might not start and end on the same vertex (since we are searching for a shortest path from a set of vertices to a set of vertices). Whenever this is the case, \( B \) is partitioned into two sets \( B_1 \) and \( B_2 \), for which in subsequent iterations shortest paths are computed analogously as for \( B \).

We search within these for cyclic paths to find lower values of the upper bound \( b_{\text{upper}} \) earlier. We want to point out that no additional computational effort is required to compute \( C_{\text{all}} \) using the implementation of [3], since all paths are already available (see Fig. A4 for runtime comparisons).

### A3. Number of Conjugate Product Edges

As mentioned in the main paper, the conjugate product graph \( P^* \) has \( 7 \) times more vertices than \( P \) and \( c \approx 11 \) times more edges. In the following we derive \( c \). To this end, we count outgoing edges of each conjugate product vertex (which is sufficient since \( P^* \) is cyclic). Further, we assume that on average each vertex \( j \) of the 3D shape \( N \) is connected to 6 edges [1]. Thus, each (directed) edge on 3D shape is connected to 5 other directed edges via their shared vertex, see Fig. A3.

![Subset of a triangle mesh](image)

Figure A3. Subset of a triangle mesh. Directed pink edge is connected to all directed black edges via blue vertex.

In conclusion, each conjugate product vertex is connected to 5 conjugate product vertices on the same layer (reflecting degenerate 2D conjugate product vertices) and 6 conjugate product vertices on the next layer (reflecting 5 non-degenerate conjugate product vertices and 1 degenerate 3D conjugate product vertex). In total, each conjugate

---

### Algorithm A1: Branch and Bound for Cyclic Shortest Path on (Conjugate) Product Graph

**Input**: 2D shape \( M = (\mathcal{V}_M, \mathcal{E}_M) \).

3D shape \( N = (\mathcal{V}_N, \mathcal{E}_N) \).

**Output**: Optimal Path \( C_{\text{opt}} \subset \mathcal{V}^* \)

// First branch is complete first layer

\( B_0 \leftarrow \{v^* = (e^M, e^N) | i_0 = 0, e^M = (i_0, i_1)\} \);

// Initialise bounds and branches

\( b(B_0) \leftarrow 0; \)

\( b_{\text{upper}} \leftarrow \infty; \)

\( B_{\text{Branches}} \leftarrow B_0; \)

// Run until no branches with a gap between lower and upper bound exist

while \( \min_{B \in B_{\text{Branches}}} b(B) < b_{\text{upper}} \) do

\( B \leftarrow \text{argmin}_{B \in B_{\text{Branches}}} b(B); \)

\( \mathcal{B}_{\text{Branches}} \leftarrow \mathcal{B}_{\text{Branches}} \setminus B; \)

Compute all paths \( C_{\text{all}} = \{C_1, C_2, \ldots\} \) with path cost \( d_C \)

if \( C_{\text{all}} = \emptyset \) then

// No path which meets \( d_C < b_{\text{upper}} \)

continue;

\( C \leftarrow \text{argmin}_{C \in C_{\text{all}}} d_C; \)

// Check if current path is cyclic

if \( v^*_1 = v^*_{|C|} \) then

if \( d_C < b_{\text{upper}} \) then

\( b_{\text{upper}} \leftarrow d_C; \)

\( C_{\text{opt}} \leftarrow C; \)

else

// Cut current branch into two parts

if \( e^N_1 = e^N_{|C|} \) then

\( B_1 \leftarrow B \setminus \{v^*_1\}; \)

\( B_2 \leftarrow \{v^*_1\}; \)

else

Compute \( B_1, B_2 \) as Voronoi cells around \( e^N_1 \) and \( e^N_{|C|} \) respectively;

// Add new branches

\( \mathcal{B}_{\text{Branches}} \leftarrow \mathcal{B}_{\text{Branches}} \cup \{B_1, B_2\}; \)

// Update lower bounds

\( b(B_1) = b(B_2) = d_C; \)

// Try to tighten upper bound

for \( C \leftarrow C_{\text{all}} \) do

if \( v^*_1 = v^*_{|C|} \) then

if \( d_C < b_{\text{upper}} \) then

\( b_{\text{upper}} \leftarrow d_C; \)

\( C_{\text{opt}} \leftarrow C; \)

end

end

end

end

end

end

end

end
comparision to product graph

A4. Runtime
A4.1. Runtime Analysis

In the following we estimate runtime complexity of our branch-and-bound algorithm for conjugate product graphs. To this end, we use $|E_P| \approx 3|V_P|$ [1] to obtain $|V_P^*| \approx 7 \cdot |V_M||V_N|$ and $|E_P^*| \approx c \cdot 7 \cdot |V_M||V_N|$. The runtime of Dijkstra on an arbitrary graph $G = (V_G, E_G)$ is $O((|E_G| + |V_G|) \cdot \log(|V_G|))$ where $(|E_G| + |V_G|)$ indicates the number of update steps to be made, and $\log(|V_G|)$ indicates the complexity to access the priority heap that is used to keep track of the next nodes to be updated.

In our case, the number of update steps is $(|E_P^*| + |V_P^*|) \approx c \cdot 14 \cdot |V_M||V_N|$ (with $c \approx 11$). We make use of the strictly directed order of the $|V_M|$ layers of $P^*$, which allows to use a heap that scales with the number of vertices of one layer $O(|V_M|)$ (also see [3]). In summary, the runtime complexity of a single Dijkstra run in our conjugate product graph $P^*$ is $O(|V_M||V_N| \log(|V_N|))$.

To find the optimal cyclic path among all possible cyclic paths, we run Dijkstra not just once but at most $O(|V_N|)$ times (without any branch-and-bound optimization), which leads to a final runtime complexity of $O(|V_M||V_N|^2 \log(|V_N|))$.

A4.2. Runtime Comparison

In Fig. A4, we show the median runtime for the approach by Lähner et al. [3] and ours (on conjugate product graph $P^*$ as well as product graph $P$ for a fair comparison). The vertical axis shows the runtime in minutes. Points (light colours) are individual experiments, while thick lines are median runtimes. Spikes in computation time stem from a varying number of branches needed to compute the optimal path. Left: We fix the size of various 3D shapes and gradually increase the number of vertices (horizontal axis) of respective 2D shapes (by subsampling). Right: We fix the size of various 2D shapes and gradually increase the number of vertices (horizontal axis) of respective 3D shapes (also by subsampling).

A5. 2D to 3D Deformation Transfer

We compute 2D to 3D deformation transfer by applying the following steps:

- **2D-3D Matching** We find a matching between 2D and 3D shape using our approach.
- **2D Deformation** We deform the 2D shape by using a skeleton which allows for different articulation of arms, legs and head. In combination with biharmonic weights [2, 7], we obtain a smooth deformation of the 2D shape (we tessellate the interior of the contour for biharmonic weight computation [4]).

**2D-3D Alignment** We find the optimal alignment $T^{3D}_{2D}$ of 2D shape and matched vertices on 3D shape by introducing a third, constant coordinate for 2D vertices and solving the (orthogonal) Procrustes problem [6].

**3D Deformation** We apply the deformation to the 3D shape by transforming the deformation on the 2D shape using $T^{3D}_{2D}$, applying the transformed deformation to a small subset of 3D vertices (chosen by furthest distance) and using their new positions as a constraint when deforming all other vertices of the 3D shape with the as-rigid-as-possible method of [5].

A6. Ablation Studies
A6.1. Cost Function

We evaluate the performance of different parts of our cost function in Tab. A1 as well as the performance of local rigidity when using multidimensional spectral features.

A6.2. Discretisation

In Tab. A2 we evaluate the robustness of our method w.r.t. to different discretisations. For all our experiments in the main paper we reduce influence of discretisation by decimating 3D shapes to half of their original resolution, which results in more uniform edge lengths [2]. Additionally, we re-sample 2D shapes with edge lengths according to the mean edge length of the decimated 3D shape.

In Fig. A5, we compare the performance of our method w.r.t. to different discretised 3D shapes. We observe that our method is relatively robust to different discretisation.
**Table A1.** Various cost functions on FAUST. The score is the area under the curve (AUC) of the cumulative segmentation errors. All introduced components increase performance. Our one-dimensional local thickness outperforms the multi-dimensional spectral features due to different intrinsic properties of 2D and 3D shapes.

| Method                                      | AUC ↑  |
|---------------------------------------------|--------|
| Local Rigidity & Spectral                  | 0.95   |
| Local Rigidity                             | 0.76   |
| Local Thickness                             | 0.92   |
| Local Rigid. & Local Thick. ($\psi_1(x) = \psi_2(x) = |x|$) | 0.89   |
| Ours                                        | 0.98   |

**Table A2.** Ablation study on the sensitivity of our approach to different discretisations. The score is the area under the curve (AUC) of the cumulative segmentation errors. We fix the discretisation of 3D shape and vary edge lengths of 2D shape. $\bar{e}$ depicts the mean edge length of 3D shape.

| Mean Edge Length 2D Shape | AUC ↑  |
|----------------------------|--------|
| 0.5-$\bar{e}$             | 0.96   |
| 0.75-$\bar{e}$            | 0.97   |
| 1-$\bar{e}$               | **0.98** |
| 1.25-$\bar{e}$            | 0.97   |
| 1.5-$\bar{e}$             | 0.95   |

Figure A5. Comparison of the matching performance of a uniformly sampled 3D shape (red line, edge lengths approx. equal) vs. a 3D shape with non-uniform density (blue line, regions with high curvature have smaller edge lengths) on the entire FAUST dataset. The vertical axis shows % of points in correct segment (↑) while the horizontal axis shows the geodesic error threshold.

In Fig. A6 we show results for matchings across different resolution of 2D and 3D shapes. The experiment confirms that our method is robust in reasonable settings. In addition, we can see that matchings on the coarsest and on the finest level result in similar correspondences (cf. shape visualisations on the right of Fig. A6).

**A6.3. Shape Discrepancies**

In Fig. A7, we show results of cross-category matchings.

Figure A6. Mesh resolution. We downsample various 2D and 3D shapes (FAUST). The values with green and yellow background are AUC (↑) of % of points in correct segment. The values with grey background are the number of vertices of respective 2D and 3D shapes. On the right we show that matchings from coarsest ($|V_M| = 432, |V_N| = 39$) and finest resolution ($|V_M| = 3446, |V_N| = 318$) are consistent.

Figure A7. Cross-category matching. We match the 2D shape of woman to the 3D shape of a woman and man, as well as to the 3D shape of dog. We can see that matching of woman to man results in a plausible matching, while matching of woman to dog does not yield meaningful results (as expected). The values are the resulting path-costs for each pair.

**A6.4. Noise**

In Fig. A8, we evaluate our method’s robustness to noise when the 3D shape is disturbed by Gaussian noise. We plot the AUC (↑) of % of points in correct segment for each noise level.

In Fig. A8 we apply Gaussian noise with $\sigma = \{0.001, \ldots, 0.008\}$ to 3D shapes and subsequently match them to respective 2D shapes. Even under severe noise (see qualitative example on the left) our method is able to compute meaningful results which is confirmed by the plot on the right which shows AUC (↑) of % of points in correct segment (vertical axis) for increasing standard deviations (horizontal axis).

Figure A8. Robustness w.r.t. Noise. We apply Gaussian noise with $\sigma = \{0.001, \ldots, 0.008\}$ to 3D shapes and subsequently match them to respective 2D shapes. Even under severe noise (see qualitative example on the left) our method is able to compute meaningful results which is confirmed by the plot on the right which shows AUC (↑) of % of points in correct segment (vertical axis) for increasing standard deviations (horizontal axis).
A7. Qualitative Results on FAUST

In Fig. A9 we show additional qualitative results.

Figure A9. Qualitative results on instances of FAUST dataset. We can see that left-right-flips occur (indicated with $\leftarrow\rightarrow$) which nevertheless are plausible matchings.

References

[1] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. Polygon Mesh Processing. CRC press, 2010. 2, 3
[2] Alec Jacobson et al. gptoolbox: Geometry processing toolbox, 2021. http://github.com/alecjacobson/gptoolbox. 3
[3] Zorah Lähner, Emanuele Rodolà, Frank R Schmidt, Michael M Bronstein, and Daniel Cremers. Efficient globally optimal 2d-to-3d deformable shape matching. In CVPR, 2016. 1, 2, 3
[4] Per-Olof Persson and Gilbert Strang. A simple mesh generator in matlab. SIAM review, 2004. 3
[5] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In SGP, 2007. 3
[6] Jos MF Ten Berge. Orthogonal procrustes rotation for two or more matrices. Psychometrika, 1977. 3
[7] Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. TOG, 2015. 3