Synchronization of Group-labelled Multi-graphs
Supplementary material

Andrea Porfiri Dal Cin[1], Luca Magri[1], Federica Arrigoni[2], Andrea Fusiello[3] and Giacomo Boracchi[1]
[1]DEIB - Politecnico di Milano (Italy) [2]DISI - University of Trento (Italy) [3]DPIA - University of Udine (Italy)

This document provides additional materials that were omitted from the main manuscript due to space restrictions. In particular, in Section 1 we detail the graph-expansion algorithm presented in Section 3.2 of the main paper, while Section 2 provides a more detailed description of the proof of Theorem 1 in Section 3.3. Full-sized tables are reported in Section 3.

1. Multi-graph expansion algorithm

The multi-graph expansion algorithm is an iterative greedy procedure that turns a Σ-labeled multi-graph Γ_{in} given in input into a simple graph $\Gamma_{\text{out}} = (\mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}}, z)$. Algorithm 1 describes the main steps. Please, refer to Definitions 1 and 2 in the main paper for the notion of a multi-graph and multi-edges, respectively. Intuitively, a multi-edge is a collection of edges with the same source and target nodes.

At first, in lines 1-3, we initialize the vertex set and the edge set of the output graph to the values of the input multi-graph. At high level, the algorithm cycles through all the vertices of the input graph and populates $\mathcal{V}_{\text{out}}$ and $\mathcal{E}_{\text{out}}$ adding the vertex and its corresponding edges, but when a multi-edge is encountered this is first expanded to simple-edges between replicated nodes, via the EXPANDNODE routine, and then it is added to $\mathcal{E}_{\text{out}}$ together with the replicas which are in turn added to $\mathcal{V}_{\text{out}}$. For reasons of efficiency, the procedure starts by expanding the nodes with more than one multi-edge, (lines 4-17) while the expansion of nodes with a single multi-edge is deferred until the end of the procedure (line 19 – 24). This delayed expansion mechanism is implemented by means of a queue $\mathcal{S}$ that is initialized as empty (line 3).

The first loop (lines 4-17) consists in iterating through the nodes in the input multi-graph and replicating those with more than one multi-edge updating $\mathcal{V}_{\text{out}}$ and $\mathcal{E}_{\text{out}}$ accordingly for the output graph. Specifically, in line 5 we gather the multi-edges for a node v and, if their count is greater than 1, we proceed to replicate v. In lines 8-9, we remove v from $\mathcal{V}_{\text{out}}$ and all its edges from $\mathcal{E}_{\text{out}}$. In line 11, we invoke the EXPANDNODE procedure, which returns $\mathcal{V}_{\text{rep}}$ and $\mathcal{E}_{\text{rep}}$. $\mathcal{V}_{\text{rep}}$ contains the replicas of v, while $\mathcal{E}_{\text{rep}}$ contains the previous edges of v distributed for $\mathcal{V}_{\text{rep}}$, including the identity constraints between replicas (more details in Alg.2).

Algorithm 1 Multi-graph expansion

Input: A Σ labeled multi-graph $\Gamma_{\text{in}} = (\mathcal{V}_{\text{in}}, \mathcal{E}_{\text{in}}, s, t, z)$
Output: The expanded graph $Go = (\mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}}, z)$

1: $\mathcal{V}_{\text{out}} = \mathcal{V}_{\text{in}}$
2: $\mathcal{E}_{\text{out}} = \mathcal{E}_{\text{in}}$
3: $\mathcal{S} = \emptyset$ ▷ queue
4: **for** $v \in \mathcal{V}_{\text{in}}$ **do**
5: $\quad \mathcal{E}_v = \text{GETMULTIEDGES}(v, \Gamma_{\text{out}})$
6: $\quad$ **if** $|\mathcal{E}_v| > 1$ **then** ▷ there are many multi-edges
7: $\quad\quad$ ▷ Remove nodes and edges
8: $\quad\quad \mathcal{V}_{\text{out}} = \mathcal{V}_{\text{out}} \setminus \{v\}$
9: $\quad\quad \mathcal{E}_{\text{out}} = \mathcal{E}_{\text{out}} \setminus \{e \in \mathcal{E}_{\text{out}} \colon s(e) = v \text{ or } t(e) = v\}$
10: $\quad\quad$ ▷ Add replicated nodes and edges
11: $\quad\quad (\mathcal{V}_{\text{rep}}, \mathcal{E}_{\text{rep}}) = \text{EXPANDNODE}(v, \Gamma_{\text{out}})$
12: $\quad\quad \mathcal{V}_{\text{out}} = \mathcal{V}_{\text{out}} \cup \mathcal{V}_{\text{rep}}$
13: $\quad\quad \mathcal{E}_{\text{out}} = \mathcal{E}_{\text{out}} \cup \mathcal{E}_{\text{rep}}$
14: $\quad$ **else if** $|\mathcal{E}_v| = 1$ **then**
15: $\quad\quad \mathcal{S} = \mathcal{S} \cup \{v\}$ ▷ add to queue
16: $\quad$ **end if**
17: **end for**
18: ▷ Process nodes with a single multi-edge
19: **for** $v \in \mathcal{S}$ **do**
20: $\quad \mathcal{E}_v = \text{GETMULTIEDGES}(v, \Gamma_{\text{out}})$
21: $\quad$ **if** $|\mathcal{E}_v| > 0$ **then**
22: $\quad\quad$ repeat lines 8-13
23: $\quad$ **end if**
24: **end for**
25: $\Gamma_{\text{out}} = (\mathcal{V}_{\text{out}}, \mathcal{E}_{\text{out}}, z)$

The elements of $\mathcal{V}_{\text{rep}}$ and $\mathcal{E}_{\text{rep}}$ are merged with those of $\mathcal{V}_{\text{out}}$ (line 12) and $\mathcal{E}_{\text{out}}$ (line 13) respectively, updating the output graph. If v has only a single incoming or outgoing multi-edge, then v is added to the queue $\mathcal{S}$ (line 15) and its replication is deferred after the other nodes with a greater number of multi-edges have been expanded.

The second loop (lines 19-24) consists in iterating through the nodes v in the queue $\mathcal{S}$. The procedure is similar to the first loop: v is replicated when it is still involved

with a multi-edge. This check (line 21) is necessary because, during the first pass, the multi-edge involved with v may have been expanded by the nodes adjacent to v.

The algorithm outputs the expanded graph Γ_{out}, which is defined by the updated vertex set $\mathcal{V}_{\text{out}}$ and edge set $\mathcal{E}_{\text{out}}$.

Expansion of node and multi-edge conversion The procedure EXPANDNODE, detailed in Algorithm 2, aims at replicating a node i originally involved in a multi-edge with a set of returned replicas $\mathcal{V}_{\text{rep}}$. In addition, EXPANDNODE converts the original multi-edges involving i in simple-edges connecting the replicas. Specifically, incoming edges, outgoing edges and identity constraints between replicas are instantiated and added to the returned set of simple edges $\mathcal{E}_{\text{rep}}$.

First, we compute the maximum number m between the cardinality of incoming and the cardinality of outgoing multi-edges involved with i (lines 2 - 7). This value is used to initialize a ordered sequence $\mathcal{V}_{\text{rep}} = \{v_1, v_2, \ldots, v_m\}$ with m replicas of i (line 8). The r-th element of $\mathcal{V}_{\text{rep}}$ is referenced by using the array-like notation $\mathcal{V}_{\text{rep}}[r]$. The set of edges for the expanded node is initially empty (line 9).

The next step is to distribute the original constraints among replicas, starting from the incoming edges (lines 11-19). For every incoming multi-edge E, we take every simple edge $e \in E$ and add a new edge f to $\mathcal{E}_{\text{rep}}$ with the same label and source node as e, but with a different target node in $\mathcal{V}_{\text{rep}}$ (lines 11- 19). In other words, we are rerouting every edge that makes up the multi-edge so that each of them has a different replica $\mathcal{V}_{\text{rep}}[r]$ of i, as target node. Notice that simple edges can be seen as multi-edges with cardinality equal to 1, thus they are distributed as well.

We apply the same steps to distribute the constraints among replicas for outgoing multi-edges. The main difference is that the re-routed edges f have the same label and target node as their counterparts, but have a replica $\mathcal{V}_{\text{rep}}[r]$ of i as a source node (lines 20-35) (see Fig. 2 in the main paper).

Finally, we add the identity constraints between replicas (lines 30 - 35). To this end, we iterate through the $k-1$ pairs of replicas (v_1, v_2), indicated as $\mathcal{P}_2(\mathcal{V}_{\text{rep}})$ (line 31), and a new edge f is added to $\mathcal{V}_{\text{rep}}$ with source node v_1, target node v_2 with the identity 1_Σ as label.

Auxiliary function for multi-edges Algorithm 3 collects auxiliary functions that are used to get the incoming, the outgoing and all the multi-edges respectively for a node i in the graph Γ. We recall that the notation $E(i, j)$ denotes the multi-edges connecting node i and j.

Computational complexity Let us begin by analyzing the complexity of the EXPANDNODE procedure. EXPANDNODE includes three for-loops. The first and second loops iterate through the incoming and outgoing multi-edges of vertex i respectively. The number of outgoing and incoming edges for i depends on its degree d_i. The upper bound for the degree of a node is the number of vertices n in the graph. Each iteration contains a nested loop that, for every edge e in E, adds a new edge to $\mathcal{E}_{rep}$. Thus, the average number of iterations for this loop is equal to the average multiplicity of the multi-edges in the multigraph. Let us assume that the cost function c represents the unit time taken to run an operation. From these considerations, we estimate the run time t_1 and t_2 for the first two loops as follows:

$$t_1 = t_2 = nm\mathsf{c} \tag{1}$$

Algorithm 2 Expand node and convert multi-edges

1: **function** EXPANDNODE(i, $\mathcal{V}$, $\mathcal{E}$)
2: ▷ Get max cardinality of multi-edges
3: $\mathcal{E}_{\to i} = \text{GETINCOMINGMULTIEDGES}(v, \mathcal{V}, \mathcal{E})$
4: $m_{\to i} = \max_{e \in \mathcal{E}_{\to i}} |e|$
5: $\mathcal{E}_{i \to} = \text{GETOUTGOINGMULTIEDGES}(v, \mathcal{V}, \mathcal{E})$
6: $m_{i \to} = \max_{e \in \mathcal{E}_{i \to}} |e|$
7: $m = \max(1, m_{\to i}, m_{i \to})$ ▷ Number of replicas
8: $\mathcal{V}_{\text{rep}} = \{v_1, \ldots, v_m\}$ ▷ Ordered set of replicas
9: $\mathcal{E}_{\text{rep}} = \emptyset$
10: ▷ Distribute incoming edges among replicas
11: **for** $E \in \mathcal{E}_{\to i}$ **do** ▷ E is a multi-edge
12: $r = 1$ ▷ replicas' counter
13: **for** $e \in E$ **do** ▷ e is a simple edge in E
14: instantiate f s.t. $s(f) = t(e), t(f) = \mathcal{V}_{\text{rep}}[r]$
15: $\mathcal{E}_{\text{rep}} = \mathcal{E}_{\text{rep}} \cup \{f\}$
16: $z(f) = z(e)$
17: $r = r + 1$
18: **end for**
19: **end for**
20: ▷ Distribute outgoing edges among replicas
21: **for** $E \in \mathcal{E}_{i \to}$ **do**
22: $r = 1$
23: **for** $e \in E$ **do**
24: instantiate f s.t. $s(f) = \mathcal{V}_{\text{rep}}[r], t(f) = t(e)$
25: $\mathcal{E}_{\text{rep}} = \mathcal{E}_{\text{rep}} \cup \{f\}$
26: $z(f) = z(e)$
27: $r = r + 1$
28: **end for**
29: **end for**
30: ▷ Add identity constraints between replicas
31: **for** $(v_1, v_2) \in \mathcal{P}_2(\mathcal{V}_{\text{rep}})$ **do**
32: instantiate f s.t. $s(f) = v_1, t(f) = v_2$
33: $\mathcal{E}_{\text{rep}} = \mathcal{E}_{\text{rep}} \cup \{f\}$
34: $z(f) = 1_\Sigma$
35: **end for**
36: return $(\mathcal{V}_{\text{rep}}, \mathcal{E}_{\text{rep}})$
37: **end function**

Algorithm 3 Auxiliary function for multi-edges

function GETINCOMINGMULTIEDGES($i, \mathcal{V}, \mathcal{E}$)
Input: A node i and a multi-graph $\Gamma = (\mathcal{V}, \mathcal{E}, s, t)$.
Output: The set of incoming multi-edges for i.
    $\mathcal{T} = \{e \in \mathcal{E} \mid t(e) = i\}$
    $\mathcal{U} = \{v \in \mathcal{V} \mid \exists e \in \mathcal{T} \text{ s.t. } s(e) = v\}$
    return $\mathcal{E}_{\to i} = \{E(v, i) : v \in \mathcal{U}\}$
end function

function GETOUTGOINGMULTIEDGES($i, \mathcal{V}, \mathcal{E}$)
Input: A node i and a multi-graph $\Gamma = (\mathcal{V}, \mathcal{E}, s, t)$.
Output: The set of outgoing multi-edges for i.
    $\mathcal{S} = \{e \in \mathcal{E} \mid s(e) = i\}$
    $\mathcal{U} = \{v \in \mathcal{V} \mid \exists e \in \mathcal{S} \text{ s.t. } t(e) = v\}$
    return $\mathcal{E}_{i \to} = \{E(i, v) : v \in \mathcal{U}\}$
end function

function GETMULTIEDGES($i, \mathcal{V}, \mathcal{E}$)
Input: A node i and a multi-graph $\Gamma = (\mathcal{V}, \mathcal{E}, s, t)$.
Output: The set of multi-edges for i.
    $\mathcal{E}_{\to i}$ = GETINGOINGMULTIEDGES(i, Γ).
    $\mathcal{E}_{i \to}$ = GETOUTGOINGMULTIEDGES(i, Γ).
    return $\mathcal{E}_{\to i} \cup \mathcal{E}_{i \to}$.
end function

where m is the average multiplicity of the multi-edges and n is the number of vertices in the graph. The run time t_3 of the third loop, that enforces equality constraints, is estimated as:

$$t_3 \leq \frac{m \times (m-1)}{2} \mathsf{c}. \tag{2}$$

Hence the overall execution time t_{exp} for EXPANDNODE is estimated as:

$$t_{exp} \leq nm\mathsf{c} + \frac{m^2}{2}\mathsf{c} - \frac{m}{2}\mathsf{c} \tag{3}$$

and the procedure is of time complexity $\mathcal{O}(nm + m^2)$

The Multi-graph expansion algorithm is made of two for-loops. The first loop iterates over n elements, where n is the number of vertices in the input multigraph. The second loop iterates over a subset of the vertex set of the input multigraph, hence it iterates over a maximum of n elements as well. Both loops perform a similar set of operations. For every iteration, the multi-edges of the current vertex v are queried. We assume that this operation is supported by a data structure that can access the edges of a vertex directly, thus getting the multi-edges of v in constant time. The EXPANDNODE procedure is invoked once per iteration and the vertex and edge sets of the graph are updated accordingly.

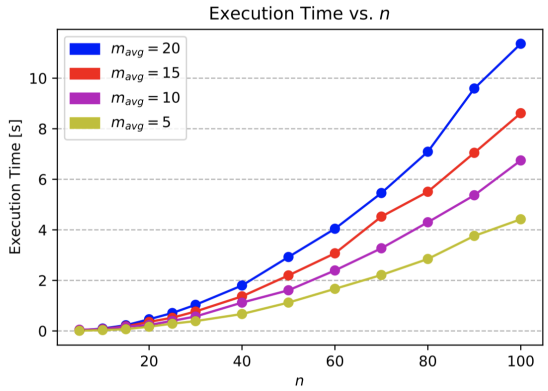

Figure 1: Execution time varying n the number of nodes and the average multiplicity m.

The overall run time estimate for the expansion of the graph is therefore:

$$t \leq 2n(\mathsf{c} + t_{exp}) = 2n\mathsf{c} + 2n^2 m\mathsf{c} + 2n\frac{m^2}{2}\mathsf{c} - 2n\frac{m}{2}\mathsf{c} \tag{4}$$

and the algorithm if of time complexity $\mathcal{O}(n^2 m + nm^2)$.

In general, the number of vertices n in the input multi-graph is much larger than the average multiplicity of the multi-edges in the same graph ($n \gg m$), therefore:

$$n^2 m + nm^2 = nm(n + m) \approx n^2 m \tag{5}$$

and the time complexity is $\mathcal{O}(n^2 m)$ in general, as confirmed by the simulation reported in Fig. 1 where the execution time with respect to different number of nodes n is computed for different values of average multiplicity m.

2. Constrained eigenvalue problem

Let us resume from Theorem 1 (Section 3.3), whose statement is reported here for the reader's convenience:

Theorem 1. *The stationary points of the cost function*

$$\min_{\mathsf{X}} \|\mathsf{MX}\|_F^2 \quad \textit{subject to } \mathsf{X}^\top \mathsf{X} = \mathsf{I}_d, \quad \mathsf{C}^\top \mathsf{X} = 0,$$

are given by the eigenvectors of $\left(\mathsf{I} - \mathsf{C}\mathsf{C}^\dagger\right) \mathsf{M}^\top \mathsf{M}$, *where* $\mathsf{C}^\dagger$ *is the pseudo-inverse of* C

Recall that $\mathsf{M}^\top \mathsf{M}$ is the matrix whose eigenvectors we are pursuing and C is the constraint matrix of rank k.

Consider the *complete orthogonal decomposition* of C:

$$\mathsf{C} = \mathsf{Q} \begin{bmatrix} \mathsf{L} & 0 \\ 0 & 0 \end{bmatrix} \mathsf{Z}^\top \tag{6}$$

where Q and Z are orthogonal and L is $k \times k$ non-singular. With the change of variable $\mathsf{Y} = \mathsf{Q}^\top \mathsf{X}$ the constraint

rewrites:

$$\mathsf{Z}\begin{bmatrix}\mathsf{L}^\top & 0\\ 0 & 0\end{bmatrix}\mathsf{Y}=0 \iff \begin{bmatrix}\mathsf{L}^\top & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\mathsf{Y}_1\\ \mathsf{Y}_2\end{bmatrix}=0 \iff \mathsf{Y}_1=0 \tag{7}$$

and the original quadratic form $\mathsf{X}^\top\mathsf{M}^\top\mathsf{M}\mathsf{X}$ becomes: $\mathsf{Y}^\top\mathsf{G}\mathsf{Y}$ where

$$\mathsf{G}=\mathsf{Q}^\top\mathsf{M}^\top\mathsf{M}\mathsf{Q}=\begin{bmatrix}\mathsf{G}_{11} & \mathsf{G}_{12}\\ \mathsf{G}_{12}^\top & \mathsf{G}_{22}\end{bmatrix}. \tag{8}$$

Let the columns of Y_2 be the d orthogonal eigenvectors corresponding to the least d eigenvalues of the $(\nu-k)\times(\nu-k)$ symmetric matrix G_{22} (where ν denotes the size of $\mathsf{M}^\top\mathsf{M}$), then it is easy to see that the solution is given by

$$\mathsf{X}=\mathsf{Q}\begin{bmatrix}0\\ \mathsf{Y}_2\end{bmatrix}. \tag{9}$$

Please note that only the Q matrix of the complete orthogonal decomposition of C is used in computing the solution. It turns out that this matrix is the same Q of the *rank revealing* QR decomposition:

$$\mathsf{C}=\mathsf{Q}\begin{bmatrix}\mathsf{R} & \mathsf{S}\\ 0 & 0\end{bmatrix}\mathsf{E}^\top \tag{10}$$

where E is a permutation and R_1 is $k\times k$ upper-triangular, non-singular. This decomposition can be computed with the Matlab command `[Q,R,E] = qr(C)`.

This formulation, described in [2], sidesteps the matrix P – which entails computing the pseudo-inverse of C – at the cost of performing the QR decomposition of C.

3. Tables

For the reader's convenience, we report here a more readable version of the tables appearing in the main paper.

References

[1] David Crandall, Andrew Owens, Noah Snavely, and Daniel P. Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3001–3008, 2011. 5

[2] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, third edition, 1996. 4

[3] K. Wilson and N. Snavely. Robust global translations with 1DSfM. In *Proceedings of the European Conference on Computer Vision*, pages 61–75, 2014. 5

Table 1: Partitioned synchronization in $\mathrm{SO}(3)$ on real data sets from [3, 1]. The average error $\bar{\epsilon}$, median error $\hat{\epsilon}$, and running time t are reported for MULTISYNC and edge averaging. Full synchronization performances are included but are intended only as an ideal reference as it works on different assumptions, having at disposal the full graph instead of partitioning it.

			Edge averaging			MULTISYNC			Full sync		
Data set	n	c	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t	$\bar{\epsilon}$	$\hat{\epsilon}$	t
Ellis Island	247	9	3.56	0.73	1.02	3.49	0.68	1.07	3	0.47	3.4
Piazza del Popolo	345	11	5.62	1.86	1.27	5.22	1.41	1.38	3.3	0.86	3.47
NYC Library	376	11	4.91	3.35	0.93	4.24	2.32	1.01	3.16	1.27	4.8
Madrid Metropolis	394	11	7.84	3.19	1.13	7.14	2.52	1.18	6.6	1.12	1.18
Yorkminster	458	12	5.96	3.98	1.33	4.98	2.91	1.37	3.5	1.58	4.83
Montreal N. Dame	474	12	2.67	1.02	1.32	2.11	0.87	1.41	1.12	0.5	10.1
Tower of London	489	13	6.43	3.51	1.07	5.54	2.74	1.12	4.21	2.33	3.91
Notre Dame	553	13	4.25	1.92	3.82	3.43	0.85	3.87	2.7	0.65	22
Alamo	627	14	6.89	1.63	4.21	6.42	1.57	4.28	3.7	1.02	26.5
Gendarmenmarkt	742	15	39.54	21.18	2.29	34.32	12.68	2.34	40.82	6.09	39.9
Vienna Cathedral	918	17	15.73	4.87	3.88	11.01	3.73	3.92	6.2	1.27	50.2
Union Square	930	17	7.71	3.88	3.65	7.25	3.67	3.71	6.18	3.6	6.61
Roman Forum	1102	17	10.03	9.12	4.95	6.91	3.39	5.01	2.81	1.4	12.1
Piccadilly	2508	21	19.89	10.21	17.45	17.47	8.21	17.61	4.42	1.94	241
Cornell Arts Quad	5530	41	8.64	3.29	17.88	6.25	2.71	18.02	3.2	1.71	191

Table 2: Performances of MULTISYNC combined with different techniques, for synchronization in $\mathrm{SO}(3)$ on real data sets [3, 1]. The median error $\hat{\epsilon}$, and running time t are reported.

			EIG-IRLS				L1-IRLS				R-GoDec			
			MULTISYNC		Full sync		MULTISYNC		Full sync		MULTISYNC		Full sync	
Data set	n	c	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t	$\hat{\epsilon}$	t
Ellis Island	247	9	1.15	0.43	1.18	0.82	1.05	0.41	0.57	2.35	1.48	0.23	1.00	0.23
Piazza del Popolo	345	11	1.78	1.18	1.02	2.24	1.84	0.53	0.98	3.55	1.86	0.24	1.48	0.49
NYC Library	376	11	3.24	3.15	1.98	1.65	2.02	0.39	1.33	2.36	2.82	0.47	3.20	1.35
Madrid Metropolis	394	11	4.01	3.83	4.43	1.79	2.68	0.57	1.01	4.20	3.94	0.29	4.07	0.49
Yorkminster	458	12	2.91	2.85	1.81	3.18	2.86	1.07	1.69	2.29	2.85	0.43	2.69	2.03
Montreal N. Dame	474	12	2.12	6.92	0.59	4.09	1.01	3.67	0.58	7.10	1.02	0.52	0.85	1.05
Tower of London	489	13	2.98	3.74	2.79	2.43	2.83	1.20	2.63	1.94	2.89	0.46	3.28	2.11
Notre Dame	553	13	1.23	5.22	0.74	7.46	1.22	25.94	0.65	29.11	1.57	1.79	1.05	1.10
Alamo	627	14	1.99	2.01	1.19	11.05	1.87	1.84	1.09	32.22	1.61	0.79	1.48	1.67
Gendarmenmarkt	742	15	26.88	8.19	76.97	11.30	14.81	2.12	28.85	12.01	37.36	1.01	28.70	5.83
Vienna Cathedral	918	17	4.72	2.88	1.62	18.23	3.92	1.68	1.37	56.80	2.53	1.59	2.08	9.26
Union Square	930	17	18.95	4.13	4.93	6.48	4.33	1.05	3.97	4.82	20.17	1.57	7.16	10.58
Roman Forum	1102	17	7.89	6.11	1.86	15.46	3.85	2.20	2.27	12.46	5.54	1.12	7.54	14.77
Piccadilly	2508	21	39.55	54.05	24.87	284.87	9.01	8.93	1.89	287.23	11.79	12.24	13.36	47.13
Cornell Arts Quad	5530	41	-	-	-	-	5.49	30.10	1.98	73.51	17.13	28.24	13.21	586.6