GECCO – Additional Evaluation Details & Results

Adrian Rebmann
University of Mannheim, Germany
rebmann@informatik.uni-mannheim.de

Matthias Weidlich
Humboldt-Universität zu Berlin, Germany
matthias.weidlich@hu-berlin.de

Han van der Aa
University of Mannheim, Germany
han@informatik.uni-mannheim.de

***Abstract*—This document provides additional details regarding the evaluation of our work. §I gives a detailed overview of the baselines employed in the paper including their implementation. §II provides all detailed results for all configurations and baseline experiments conducted on real-life event logs, which could not all be included in the paper due to space limitations. §III presents additional results from experiments conducted on a synthetic data collection containing 100 event logs with very diverse behavioral characteristics to illustrate the general applicability of our work, thus complementing the results using the real-life log collection employed in the paper.**

I. Baseline Details

This section provides further details about the baselines used to evaluate our work.

A. Graph Querying Baseline (BL_Q)

This baseline replaces the first step of GECCO by querying the DFG using constraints formulated as graph queries. We employ the *Neo4J* graph database[1] and the graph query language *Cypher* [1] to implement this baseline. The DFG of each event log used in the experiments is created in a *Neo4J* database using the node type `EventClass` representing the event classes C_L in the event log L and the relation type `followedby` representing directly-follows relations between events. To allow for constraints to be imposed, nodes are stored with a *name* property and any class-level event attributes as properties, which are constant for events of a particular class throughout L, such as the *role* that performs an event. Note in particular that it does not make sense to store instance-level event attributes in the graph, as the mapping of a particular attribute value to the specific case to which the event instance having said attribute value belongs is necessary to check for constraint satisfaction of instance-based constraints.

Therefore, we compare against three different kinds of class-based constraints. Listing 1 represents the constraint on the group size, such that at most 5 event classes can be grouped ($BL1$). Listing 2 is an example of a *cannotlink* constraint combined with a constraint on the group size ($BL2$), where in this example the event classes 'a' and 'b' cannot be grouped. Listing 3 provides an example of a query representing the constraint that only event classes having the same value for a class-level attribute can be grouped ($BL3$). All of these are so-called *pattern queries*, which allow to describe the shape of the data required in terms of patterns. For instance, the query in Listing 1 finds data in the graph with a shape that fits the following pattern: a node (with the name property 'a') and all the `followedby` related nodes, from 0 up to 5 hops away. Finally, the output is limited to at most 5 nodes. In all example queries, the variable `depth` can be replaced by the number of hops that should be taken at most, i.e., a number between 0 and $|C_L|$. This allows us to obtain candidate groups of different sizes.

Note that for a fair comparison, we issue these queries for every node, i.e, event class, in C_L in the *WHERE* statement (as shown for 'a' in the example queries). Per node, we issue queries with all values for `depth` from 0 up to the group size limit, respectively maximum depth ($|C_L|$). Furthermore, the *DISTINCT* statement enforces the set nature of candidate groups considered in our work.

Listing 1: Examplary query to get a candidate group of at most size 5 based on nodes reachable from a specific event class 'a' in the graph ($BL1$)

```
MATCH (ec:EventClass)-[:followedby*depth]
->(reachable)
WHERE ec.name = 'a'
RETURN DISTINCT reachable
LIMIT 5
```

Listing 2: Examplary query to get a candidate group that respects a cannotlink constraint between 'a' and 'b' ($BL2$)

```
MATCH (ec:EventClass)-[:followedby*depth]
->(reachable)
WHERE ec.name = 'a'
AND reachable.name <> 'b'
RETURN DISTINCT reachable
LIMIT 5
```

Listing 3: Examplary query to get a candidate group that resprects the class-level event attribute constraint from a specific event class 'a' in the graph ($BL3$)

```
MATCH (ec:EventClass)-[:followedby*depth]
->(reachable)
WHERE  ec.name = 'a'
AND eventclass.attribute =
 reachable.attribute
RETURN DISTINCT reachable
```

[1]https://neo4j.com

After having obtained the candidate groups using the queries, steps 2 and 3 of GECCO are applied.

The Python implementation of BL_P is provided alongside the implementation of GECCO in `eval/graphquerying_bl.py`. To run, it requires a running *Neo4J graph database* with the configurations listed in the `README` of the repository.

B. Graph Partitioning Baseline (BL_P)

We compare GECCO to graph partitioning, since its goal to find a disjoint set of cohesive groups for log abstraction is similar to that of graph partitioning, which aims to partition a graph such that edges between different groups have a low weight [2]. Given a DFG, BL_P therefore aims to minimize the sum of directly-follows frequencies of cut edges, while cutting the graph into n partitions.

To implement BL_P, we employ *sci-kit-learn*'s[2] implementation of spectral clustering, a common approach to partition similarity graphs [2]. We precompute the adjacency matrix of the similarity graph of a particular DFG using its normalized directly-follows frequencies (sim_{df}). As undirected graphs are required in this approach, we sum the frequencies in case of bi-directional edges between classes or, in case of uni-directional edges, simply remove the edge direction, as suggested by Luxburg [2]. Given two event classes a and b, we compute the normalized directly-follow frequency as follows

$$sim_{df}(a,b,L) = 1 - \frac{1}{|a >_L b| + |b >_L a| + 1} \qquad (1)$$

For instance, if *a* is followed by *b* twice, and *b* is followed by *a* once in an event log *L*, the directly-follows frequency of the {a,b} is 3. The normalized directly-follows frequency is then $1 - \frac{1}{3+1}$ The specific number of desired groups in the output can be specified as a parameter n. As such, we can compare against grouping constraints, which require an exact number of groups in the result. Specifically, we compare against a grouping constraint that aims to reduce the number of activities in the abstracted log by half ($BL4$).

The Python implementation of BL_P is provided alongside the implementation of GECCO in `eval/graphpartitioning_bl.py`.

C. Greedy Baseline (BL_G)

In the following, we provide an extended description of the greedy baseline BL_G. It takes as input an event log L over a set of event classes C_L and proceeds as follows:

1) *Initialization*: BL_G starts with all event classes from C_L as singleton groups to establish the initial set of groups G_0.
2) *Candidate assessment and distance computation*:
   a) In each iteration i, we establish potential candidates by merging two groups $g_x, g_y \in G_i$ into a new group g_{xy}. This is done for each pair $g_x, g_y \in G_i$ that does not violate the imposed constraints, i.e., for which $\texttt{holds}(g_{xy}, L)$ is true.
   b) Then, we establish the next grouping G_{i+1} based on the merged group g_{xy} that leads to the lowest overall distance score $\texttt{dist}(G_{i+1}, L)$, when replacing the subgroups g_x and g_y in G_i with their merged version, i.e., $G_{i+1} = (G_i \cup \{g_{xy}\}) \setminus \{g_x, g_y\}$.
   c) Afterwards, we determine if this iteration $i+1$ is favorable in comparison to iteration i, by checking if the distance of G_{i+1} is lower than the distance of G_i. If so, BL_G continues, otherwise it returns G_i.
3) *Termination*:
   a) If singleton groups are contained in G_i, these are checked for constraint violation now, as these have not been assessed yet. If there are violations, the respective groups are removed from G_i.
   b) Finally, the baseline checks whether all event classes C_L are covered by G_i. If so the final grouping $G = G_i$ is returned, else *null* is returned, since the log-abstraction problem was not solved.

   In case a solution was found, Step 3 of GECCO is applied to create the abstracted event log *L'*.

The Python implementation of BL_G is provided alongside the implementation of GECCO in `eval/greedy_bl.py`.

[2]https://scikit-learn.org/stable/

II. Extended results of experiments on real data

A. Results of the different configurations

Due to space limitations we could only show results per constraint set for one configuration of GECCO in the paper. Here we provide the results for all configurations we tested. For convenience, we provide the table listing the constraint sets used in the experiments again in Table I.

TABLE I: Constraints used in the experiments.

ID	Categories	Constraint(s)
A	R_I	$\|g.role\| \leq 3$
M	R_I	$\texttt{sum}(g.duration) \geq 10^1$
N	R_I	$\texttt{avg}(g.duration) \leq 5 * 10^5$
Gr	R_G	$\|G\| \leq 3$
$C1$	R_I, R_G	$A \wedge N \wedge Gr$
$C2$	R_I, R_G	$A \wedge M \wedge N \wedge Gr$
$BL1$	R_C	$\|g\| \leq 5$
$BL2$	R_C	$BL1 \wedge \texttt{cannotLink}(e_1.C, e_2.C)$
$BL3$	R_C	$\|g.D\| = 1$
$BL4$	R_G	$\|G\| = \|L\|/2$

Table II depicts results for the configurations averaging only over solved cases, whereas Table III shows results averaged over all cases. Note that the latter only includes constraint sets for which there were unsolved problems.

All configurations were able to solve the same problems. The only exception is a single problem using the non-monotonic N constraint on one of the event logs [3], which could not be solved by DFG_k.

It can be seen that the results regarding size and complexity reduction as well as silhouette coefficient are stable across configurations for all constraint sets. When dealing with anti-monotonic and grouping constraints (such as in A, Gr, and

TABLE II: Results for all GECCO configurations per constraint, averaged over solved cases.

Const.	Config.	Solved	S. red.	C. red.	Sil.	T(m)
A	Exh	1.00	0.68	0.63	0.15	146
	DFG_∞	1.00	0.69	0.64	0.19	117
	DFG_k	1.00	0.63	0.55	0.09	64
M	Exh	0.31	0.58	0.55	0.15	75
	DFG_∞	0.31	0.50	0.45	0.04	92
	DFG_k	0.31	0.47	0.43	0.01	37
N	Exh	0.77	0.68	0.65	0.12	154
	DFG_∞	0.77	0.69	0.63	0.19	123
	DFG_k	0.69	0.58	0.52	0.09	68
Gr	Exh	1.00	0.66	0.61	0.13	144
	DFG_∞	1.00	0.67	0.61	0.18	120
	DFG_k	1.00	0.60	0.55	0.09	57
C1	Exh	0.54	0.68	0.59	0.12	134
	DFG_∞	0.54	0.68	0.61	0.18	130
	DFG_k	0.54	0.61	0.53	0.08	79
C2	Exh	0.23	0.50	0.40	0.09	100
	DFG_∞	0.23	0.43	0.33	0.07	100
	DFG_k	0.23	0.39	0.32	0.03	65
BL1	Exh	1.00	0.67	0.61	0.12	122
	DFG_∞	1.00	0.67	0.55	0.18	79
	DFG_k	1.00	0.59	0.53	0.11	25
BL2	Exh	1.00	0.66	0.61	0.12	121
	DFG_∞	1.00	0.67	0.59	0.19	90
	DFG_k	1.00	0.59	0.53	0.11	25
BL3	Exh	1.00	0.38	0.29	-0.02	38
	DFG_∞	1.00	0.36	0.36	0.05	29
	DFG_k	1.00	0.33	0.28	-0.02	4
BL4	Exh	1.00	0.51	0.46	0.05	147
	DFG_∞	1.00	0.51	0.45	0.11	146
	DFG_k	1.00	0.51	0.44	0.07	68

$BL1$ to 4), DFG_∞ performs as well or even slightly better than Exh. Better results for size and complexity reductions are caused by the fewer timeouts that the DFG-based configurations encounter. With respect to the silhouette coefficient, Exh is outperformed by DFG_∞ for all of these constraints as well, showing the efficacy of the DFG-based configuration to find candidate groups that are cohesive and well separated.

For constraint sets containing a monotonic constraint (M and $C2$), the abstraction degree DFG_∞ achieved is about 7% lower than that achieved by Exh. DFG_k, which imposes a beam width, on the other hand scores only about 3% less with respect to size and complexity reduction than the configuration with unlimited beam width. Yet, it improves the runtime efficiency by a factor of 2.

An interesting observation is that for the monotonic M constraint set DFG_∞ takes on average 17 minutes longer than the exhaustive candidate computation Exh, when considering only solved cases. Taking a closer look at the four logs for which solutions were found reveals that this difference in runtime is caused by a single log [4], which contains only 11 event classes, but has a highly dense DFG. Therefore, simply computing all candidates exhaustively is considerably faster in this particular case. Note that these longer runtimes can be easily avoided by only applying the DFG-based candidate computation if a certain number of event classes is reached, as, for instance, checking 2^{11} [4] candidates is manageable, whereas checking 2^{70} [5] candidates is intractable.

TABLE III: Results for all GECCO configurations per constraint, averaged over all cases.

Const.	Config.	Solved	S. red.	C. red.	Sil.	T(m)
M	Exh	0.31	0.18	0.17	0.05	141
	DFG_∞	0.31	0.15	0.14	0.01	123
	DFG_k	0.31	0.15	0.13	0.01	70
N	Exh	0.77	0.53	0.50	0.09	147
	DFG_∞	0.77	0.53	0.49	0.14	123
	DFG_k	0.69	0.40	0.36	0.06	54
C1	Exh	0.54	0.37	0.32	0.06	147
	DFG_∞	0.54	0.37	0.33	0.10	116
	DFG_k	0.54	0.33	0.28	0.04	69
C2	Exh	0.23	0.11	0.09	0.02	148
	DFG_∞	0.23	0.10	0.08	0.02	120
	DFG_k	0.23	0.09	0.07	0.01	75

B. Results of the baseline comparison

In the paper, we show the results of the comparison of one configuration of GECCO against the three baselines for brevity. Table IV complements Table VII of the paper showing the comparison of all three GECCO configurations against all three baselines. Results show that all GECCO configurations outperform all baselines with respect to complexity reduction as well as silhouette score.

TABLE IV: Results of the baselines compared to all GECCO configurations. Only constraints possible for the resp. baseline were used. Results are averaged over solved cases.

Const.	Config.	Solved	S. red.	C. red.	Sil.	T(m)
BL[1-3]	Exh	1.00	0.63	0.56	0.10	109
	DFG_∞	1.00	0.63	0.55	0.17	77
	DFG_k	1.00	0.56	0.50	0.09	23
	BL_Q	0.96	0.55	0.43	-0.20	24
BL4	Exh	1.00	0.51	0.46	0.05	147
	DFG_∞	1.00	0.51	0.45	0.11	146
	DFG_k	1.00	0.51	0.45	0.07	68
	BL_P	1.00	0.51	0.42	0.01	1
A,M,N	Exh	0.69	0.67	0.63	0.14	140
	DFG_∞	0.69	0.66	0.61	0.17	115
	DFG_k	0.67	0.59	0.52	0.08	58
	BL_G	0.64	0.45	0.37	0.02	24

With respect to the number of solved problems, BL_Q failed to solve one problem using the $BL3$ constraint set, which GECCO was able to solve, indicating the more comprehensive sets of candidate groups GECCO can find compared to this baseline. Furthermore, BL_G failed to solve a problem using the N constraint set, which our most efficient configuration DFG_k was able to solve.

III. Experiments on synthetic data

This section present results of experiments that were conducted on a collection of synthetic event logs. In the following we present the data set and the results obtained.

A. Setup

Synthetic event logs. For our experiments, we generated a collection of synthetic event logs by first using an established technique [6] to generate 100 random process models using its default configuration. Then, we simulated 1,000 traces for each process model, yielding a log collection with diverse behavioral characteristics and complexity, as shown in Table V.

TABLE V: Properties of the synthetic log collection.

Property	**Min.**	**Max.**	**Avg.**	**Med.**
Nodes in DFG	9	28	18.12	18.0
Edges in DFG	10	391	85.39	54.0
Trace variants	6	1,000	308.97	145.5
Min. Trace length	1	16	6.18	6.0
Max. Trace length	6	119	22.45	17.5

To support various abstraction constraints, we augment each event with a categorical *role* and a numerical *duration* attribute. For the *role* attribute, we pick the size of its domain $n \in [2, |C_L|]$, for each log L, and then randomly assign these n roles to the event classes in that log. This way, our collection contains event logs in which many events are performed by the same role (allowing for a high degree of abstraction) and cases with a lot variety (less abstraction). For the *duration* attribute, we ensure that each log contains event classes with a range of distributions. Therefore, each event class is assigned either a normal or an exponential distribution, having a mean duration of 10^i minutes with $i \in [0, 4]$.[3] In this way, event classes have an average duration ranging from just a minute to approximately one week.

Constraints, Configurations, Baselines, and Measures. We conduct experiments on the synthetic logs using the same constraint sets as for the real-life event logs, listed in Table I. The same configurations are tested and we compare against the same baselines using the same evaluation measures.

B. GECCO *results*

This section reports on the results of the experiments using synthetic data. Table VI depicts results for the different constraint sets and the different GECCO-configurations. Compared to the real-life event logs, all configurations solve almost all problems with monotonic and non-monotonic constraint sets (M and N) as well, indicating that the constraint sets are non-restrictive with respect to the data collection at hand. Yet, the trends are similar as the DFG-based configurations perform comparably well as the exhaustive approach (Exh). This also holds for the degree of abstraction, evidenced by the stable numbers for size and complexity reduction across configurations. For most constraint sets, a size and complexity reduction of >0.7 is achieved. The identified groups are also cohesive and separated for all configurations, with silhouette coefficients well above 0.2 for constraint sets containing anti-monotonic and grouping constraints. As there are almost no timeouts, the efficiency gains that can be achieved using the DFG-based configurations are even clearer, when looking at the results of the synthetic event logs. Runtimes improve by a factor of 2 using DFG$_\infty$ compared to Exh. For DFG$_k$ the efficiency gains are even more considerable, as it is at least 20 times faster than Exh for most constraint sets (all except the class-based $BL1$-3 constraint sets).

[3]For the normal distribution, we set the standard deviation σ to 10^{i-1}.

TABLE VI: Results using the synthetic data collection. We show results per GECCO configurations and per constraint, averaged over solved cases.

Const.	**Config.**	**Solved**	**S. red.**	**C. red.**	**Sil.**	**T(m)**
A	**Exh**	1.00	0.74	0.71	0.20	10
	DFG$_\infty$	1.00	0.71	0.69	0.23	5
	DFG$_k$	1.00	0.70	0.69	0.23	0.5
M	**Exh**	0.98	0.78	0.79	0.28	26
	DFG$_\infty$	0.95	0.73	0.74	0.25	13
	DFG$_k$	0.93	0.67	0.67	0.23	0.5
N	**Exh**	1.00	0.78	0.79	0.28	35
	DFG$_\infty$	1.00	0.78	0.78	0.27	13
	DFG$_k$	1.00	0.73	0.68	0.25	0.8
Gr	**Exh**	1.00	0.77	0.79	0.27	30
	DFG$_\infty$	1.00	0.77	0.77	0.27	9
	DFG$_k$	1.00	0.75	0.76	0.26	1
C1	**Exh**	1.00	0.72	0.71	0.20	31
	DFG$_\infty$	1.00	0.71	0.69	0.23	14
	DFG$_k$	1.00	0.70	0.69	0.23	0.8
C2	**Exh**	0.98	0.77	0.77	0.28	33
	DFG$_\infty$	0.95	0.67	0.70	0.21	15
	DFG$_k$	0.93	0.63	0.64	0.21	0.8
BL1	**Exh**	1.00	0.74	0.75	0.26	4
	DFG$_\infty$	1.00	0.73	0.73	0.27	1
	DFG$_k$	1.00	0.72	0.71	0.27	0.3
BL2	**Exh**	1.00	0.74	0.75	0.26	3
	DFG$_\infty$	1.00	0.73	0.73	0.27	1
	DFG$_k$	1.00	0.72	0.71	0.27	0.3
BL3	**Exh**	1.00	0.37	0.26	-0.18	0.1
	DFG$_\infty$	0.96	0.25	0.17	-0.08	0.1
	DFG$_k$	0.96	0.25	0.17	-0.08	0.1
BL4	**Exh**	1.00	0.51	0.52	0.13	49
	DFG$_\infty$	1.00	0.51	0.48	0.16	30
	DFG$_k$	1.00	0.51	0.48	0.16	1

C. *Results of the baseline comparison*

Table VII depicts the results of the comparison of all three GECCO configurations against all three baselines. The results show that all GECCO configurations outperform all baselines with respect to complexity reduction as well as silhouette score, essentially mirroring the results obtained for the real-life event logs.

TABLE VII: Results of the baselines compared to all GECCO configurations using the synthetic data collection. Only constraints possible for the resp. baseline were used. Results are averaged over solved cases.

Const.	**Config.**	**Solved**	**S. red.**	**C. red.**	**Sil.**	**T(m)**
BL[1-3]	**Exh**	1.00	0.62	0.58	0.12	2
	$\mathbf{DFG}_{\infty}$	0.99	0.58	0.56	0.15	1
	$\mathbf{DFG}_{k}$	0.99	0.57	0.54	0.15	0.3
	$\mathbf{BL}_{Q}$	0.93	0.50	0.49	-0.23	0.1
BL4	**Exh**	1.00	0.51	0.52	0.13	49
	$\mathbf{DFG}_{\infty}$	1.00	0.51	0.48	0.16	146
	$\mathbf{DFG}_{k}$	1.00	0.51	0.48	0.16	68
	$\mathbf{BL}_{P}$	1.00	0.51	0.45	0.09	<0.1
A,M,N	**Exh**	0.99	0.76	0.76	0.25	24
	$\mathbf{DFG}_{\infty}$	0.98	0.75	0.75	0.26	10
	$\mathbf{DFG}_{k}$	0.98	0.74	0.73	0.25	0.8
	$\mathbf{BL}_{G}$	0.88	0.50	0.35	0.09	0.2

References

[1] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 1433–1445. [Online]. Available: https://doi.org/10.1145/3183713.3190657

[2] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[3] B. van Dongen, "Bpi challenge 2014: Activity log for incidents," Apr 2014. [Online]. Available: https://data.4tu.nl/articles/dataset/BPI_Challenge_2014_Activity_log_for_incidents/12706424/1

[4] M. M. de Leoni and F. Mannhardt, "Road traffic fine management process," Feb 2015. [Online]. Available: https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249/1

[5] A. A. Augusto, R. R. Conforti, M. M. Dumas, M. M. La Rosa, F. F. M. Maggi, A. A. Marrella, M. M. Mecella, and A. A. Soo, "Data underlying the paper: Automated discovery of process models from event logs: Review and benchmark, bpic 2015 log 1," Jun 2019. [Online]. Available: https://data.4tu.nl/articles/dataset/Data_underlying_the_paper_Automated_Discovery_of_Process_Models_from_Event_Logs_Review_and_Benchmark/12712727/1

[6] T. Jouck and B. Depaire, "Generating artificial data for empirical analysis of control-flow discovery algorithms," *Business & Information Systems Engineering*, vol. 61, no. 6, pp. 695–712, 2019.