A General Framework for Multiplets Selection: Algorithmization and Complexity Analysis

DAŇKOVÁ Martina and ŠUSTEK Jan

Institute for Research and Applications of Fuzzy Modeling, University of Ostrava, Centre of Excellence IT4Innovations, 30. dubna 22, 70200 Ostrava, Czech Republic
E-mail: Martina.Dankova@osu.cz, Jan.Sustek@osu.cz

Abstract

In this contribution, we present the `multiplets` algorithm for constructing and selecting optimal sets of disjoint hyperedges across multiple groups in tabular data. We describe its main computational steps and provide a complexity analysis covering both the edge construction and optimization phases, based on the Linear Sum Assignment method and the Constraint Programming SAT-based solver.

1 Introduction

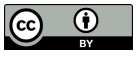
Forming balanced and representative teams across several predefined groups is a common problem in many areas of data analysis, education, and optimization. Typical applications include matching individuals from different categories or cohorts so that each resulting team (also called *multiplet*) contains one member from each group and the overall configuration is as coherent as possible according to some similarity or cost measure. This task can be viewed as finding a set of disjoint *hyperedges* in a multipartite graph, where vertices represent individuals and edge weights represent pairwise similarities or distances [1, 2]. Such matching problems appear not only in team assignment, but also in cross-domain record linkage, cohort pairing, or multi-modal data alignment [3, 4].

To address this, we introduce the `multiplets` algorithm, which systematically constructs and selects an optimal set of disjoint multiplets across multiple groups in tabular data. The method generalizes bipartite matching to a multipartite setting and combines efficient data manipulation (through the Polars framework) with exact optimization using either the Linear Sum Assignment method or the Constraint Programming SAT-based solver from OR-Tools. The following example illustrates the core idea in an intuitive context. For practical implementation we developed the `multiplets` Python module [5], which supports partitioning a dataframe into groups, computing pairwise edges and hyperedges, and selecting optimal set of disjoint hyperedges via exact optimization.

Imagine you have several groups of students (for example, different classes or training groups). The goal is to form *multiplets* — small teams where each member comes from a different group, and the members of the team are chosen so that they "fit together" according to some criterion (for example, similar performance, body height, or test score).

Our algorithmic approach to solve this problem consists of three main steps:

Acknowledgement The contribution has been funded from the project "Research of Excellence on Digital Technologies and Wellbeing CZ.02.01.01/00/22_008/0004583", which is co-financed by the European Union.



https://www.doi.org/10.15452/978-80-7599-515-5.2026.07

1. **Score pairs across groups:** For every possible pair of students from different groups, the algorithm computes how similar they are (a "weight"). Pairs that do not meet the condition (for example, weight is too large) are discarded.

2. **Build multiplets:** From these pairs the algorithm constructs all possible teams (multiplets), always including one student from each group.

3. **Optimize the selection:** The algorithm then uses a suitable optimization method to find the best set of disjoint multiplets (i.e. the largest set with the smallest total weight).

So in simple words: the result should help to divide students into balanced, mixed teams.

In more detail, the procedure first verifies the validity of the input dataframe, then constructs pairwise edges and corresponding hyperedges across all groups, and finally applies an exact optimization step—using either the Linear Sum Assignment method or the Constraint Programming SAT-based solver—to select the optimal set of disjoint multiplets.

2 Description of the Multiplets Algorithm

The `multiplets` module implements functions for constructing, weighting, and selecting sets of *hyperedges* (multiplets) in a multipartite graph derived from tabular data.

- **Initialization:** Given a Polars `DataFrame` containing a unique identifier column (`id`), a grouping attribute (`group`) and other columns, the constructor partitions the vertices according to their group membership using the `partition_by` function. It validates the input and raises an error if any of the following conditions hold: the dataframe is empty, required columns are missing, identifiers are non-unique or null, or the number of groups is less than 2 or exceeds 10 (unless `check_too_big=False`). Each partition is internally indexed by an integer column for efficient joining.

- **Edges and hyperedges construction (`init_edges`):** For every pair of groups, all possible inter-group edges are generated via a Cartesian product. A user-specified `weight` expression (column name, numeric literal, or Polars expression from the other columns) is evaluated for each pair and stored in column `_weight`. An optional `filter` condition removes edges that do not satisfy a constraint (e.g., an upper bound for `_weight`). Hyperedges are then constructed as complete combinations of vertices containing exactly one vertex from each group, such that all required pairwise edges exist. The total weight of a hyperedge is obtained by an aggregation function (defaulty by `sum_horizontal`) of the weights of the pairwise edges. The resulting dataframe `hyperedges` is deterministically sorted by vertex identifiers.

- **Selection of optimal multiplets (`find_multiplets`):** The algorithm searches for a set of disjoint hyperedges that maximizes the number of multiplets (primary objective) and minimizes the total weight (secondary objective). Parameter `penalty` can be used to prefer smaller set of multiplets when the weights in a bigger set are too large. Two solvers are implemented:

  - For **two groups**, the Linear Sum Assignment (**LSA**) algorithm from OR-Tools is used by default. Algorithm LSA works only with integer weights, so the weights are multiplied by an optional parameter `multiplier` to reduce rounding errors. Algorithm LSA searches for a perfect matching in a graph, therefore some virtual edges are internally added.

– For **three or more groups** (or when `force_CPSAT=True`), a Constraint Programming formulation is solved by the OR-Tools **CP-SAT** optimizer. Each candidate hyperedge corresponds to a binary variable, and mutual exclusivity between overlapping hyperedges is enforced as linear constraints. The objective function maximizes the sum of (`penalty` − `_weight`) terms. Optional parameters include `max_time` which forces the solver to stop after a given time, returning a sub-optimal result.

The method returns the Polars `DataFrame` with selected multiplets. If no feasible solution is found, a `multipletsErrorNoSolution` is raised.

- **Reproducibility:** The current implementation of `find_multiplets` is not deterministic for both solvers and the results are not reproducible if there is more than one optimal solution, which is often the case. This is caused by stochastic effects in OR-Tools heuristics.

In summary, the `multiplets` module constructs a multipartite hypergraph from tabular data, evaluates feasible cross-group combinations, and employs exact optimization (LSA or CP-SAT) to obtain a maximum set of disjoint hyperedges with minimal total weight. This framework provides a reproducible and interpretable approach to structured matching across multiple categorical partitions.

The `multiplets` module also contains several functions for easier tabular manipulation with the dataframe with selected multiplets, and with other dataframes.

3 Computational Complexity

Let the input consist of k groups with sizes $n_0, \ldots, n_{k-1}$ and total size $N = \sum_{i=0}^{k-1} n_i$. Denote by $M = \sum_{0 \le i < j \le k-1} m_{ij}$ the total number of *kept* pairwise edges after filtering, with $m_{ij} \le n_i n_j$, and by $H \le \prod_{i=0}^{k-1} n_i$ the number of feasible hyperedges (complete k-cliques across groups).

(1) Partitioning and input checks. Building group partitions and validating input (non-empty dataframe, required columns, uniqueness of `id`, and bounds on k) runs in

$$\text{time } O(N), \qquad \text{space } O(N).$$

(2) Pairwise edge construction (init_edges). For each pair (i, j) the algorithm performs a Cartesian product, evaluates the `weight` expression, and applies the `filter`. Let m_{ij} be the number of kept edges for (i, j). Then

$$\text{time } O\Big(\sum_{i<j} n_i n_j\Big) \text{ (worst case) or } O(M) \text{ (after filtering)}, \qquad \text{space } O(M).$$

(3) Hyperedge (clique) construction. The pairwise edge tables are joined to form cliques with exactly one vertex from each group; the $\binom{k}{2} = O(k^2)$ pair weights are aggregated horizontally (by default, by sum). With H feasible hyperedges, we have

$$\text{time } O(Hk^2) \text{ for aggregation} + \text{(hash-join overhead)}, \qquad \text{space } O(H).$$

In dense settings, H can approach $\prod_i n_i$; in practice, filtering controls H.

(4) Selection (find_multiplets).

- **LSA algorithm** ($k = 2$ unless `force_CPSAT=True`): after augmenting to a square weight matrix of size $p = n_0 + n_1$, the Hungarian (LSA) algorithm yields

$$\text{time } O(p^3), \qquad \text{space } O(p^2),$$

  with $O(p^2)$ to build the augmented table and $O(p)$ to extract matches.

- **CP-SAT algorithm** ($k \geq 3$ or `force_CPSAT=True`): the CP-SAT model has one binary variable per hyperedge (H total) and $\leq N$ at-most-one constraints (one per person), where each constraint can touch up to $\prod_{j \neq i} n_j$ hyperedges in the dense worst case. Model building is

$$\text{time } O(Hk) + O(\text{nnz}), \qquad \text{space } O(H + \text{nnz}),$$

  where nnz = number of nonzero coefficients in all constraints; and the solve time is exponential in the worst case (NP-hard set-packing); CP-SAT performs well empirically but has no polynomial worst-case bound. An optional time limit `max_time` caps the solve time.

(5) Summary. The build phase costs $O\big(\sum_{i<j} n_i n_j\big)$ time and $O(M)$ space for edges, then $O(Hk^2)$ time and $O(H)$ space for hyperedges (up to join constants). Selection is $O((n_0 + n_1)^3)$ for $k = 2$ (after $O(n_0 n_1)$ build) and NP-hard for $k \geq 3$ via CP-SAT. Aggressive filtering that reduces m_{ij} (hence M) typically shrinks H dramatically and dominates practical scalability.

References

[1] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1–2):83–97, 1955.

[2] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003.

[3] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[4] L. Perron, V. Furnon, and the OR-Tools Team. OR-Tools CP-SAT solver: A modern constraint programming approach. *Google Optimization Tools Documentation*, 2020. Available at: `https://developers.google.com/optimization`.

[5] M. Daňková, J. Šustek. `multiplets` — a Python module for multipartite hyperedge selection. GitHub repository, `https://github.com/jsustek/multiplets`, 2025.