An Algorithmic Approach for Solving the Grouping Problem

Bachelor's Thesis

Andrin Müller

Viola Meier

Universität Bern

Bern, February 6, 2025

First Reviewer: **Prof. Dr. Timo Kehrer**
University of Bern
Software Engineering Group

Second Reviewer: **Roman Bögli**
University of Bern
Software Engineering Group

Abstract

An Algorithmic Approach for Solving the Grouping Problem

by Andrin Müller
and Viola Meier

The grouping problem, a challenging combinatorial optimization problem classified as NP-hard, seeks to divide a set of objects into optimal groups while satisfying predefined constraints. This thesis explores algorithmic approaches to solve this problem for fixed group sizes and in scenarios involving must-link and cannot-link constraints, which enforce mandatory co-membership or exclusion between specific individuals.

The primary focus is on developing novel methods for encoding features and constraints that minimize total intra-group distances, ensuring the desired homogeneity or heterogeneity at the feature level. One of the key contributions of this work is the development of a unified feature encoding technique that accommodates various feature types, enhancing the accuracy of member compatibility analysis by seamlessly integrating both homogeneous and heterogeneous features.

Another significant contribution is the transformation of compatibility measurements into a single-objective optimization framework, which simplifies the computational process. This transformation enables the evaluation of member compatibility with greater efficiency, ensuring that the minimization of intra-group distances remains the central objective.

Additionally, the thesis presents three algorithmic approaches for group assembly: Ant Colony Optimization (ACO), a customized PCKMeans clustering algorithm, and the Occurrence Ranking (OCCR) approach. Each of these algorithms provides a different strategy for solving the grouping problem and can be adapted for various optimization objectives. The effectiveness and efficiency of these algorithms are thoroughly evaluated, with performance metrics tailored specifically for group formation tasks. This evaluation demonstrates their capability to generate optimal groupings in a reasonable computational time frame.

The experimental results highlight the potential applications of these methods in collaborative learning, team-building, and other domains that require structured group formation. The findings also lay the groundwork for future work, including improvements to algorithm scalability, the integration of more complex constraints, and the exploration of alternative optimization techniques.

Contents

Abstract ii

1. Introduction 2
- 1.1. Grouping Problem . . . 3
  - 1.1.1. Combinatorial Optimization Problems . . . 3
  - 1.1.2. Representation . . . 4
  - 1.1.3. Terminology . . . 6
- 1.2. Related Work . . . 7
  - 1.2.1. Group Forming Methods . . . 7
  - 1.2.2. Group Forming Tools . . . 8
- 1.3. Contribution . . . 9
  - 1.3.1. Individual Attribution . . . 10

2. Member Characteristics 11
- 2.1. Describing Features . . . 12
  - 2.1.1. Feature Types . . . 12
  - 2.1.2. Feature Class . . . 12
  - 2.1.3. Answer Cardinality . . . 13
- 2.2. Encoding . . . 13
- 2.3. Distance Metric . . . 14
  - 2.3.1. Componentwise Distance . . . 14
  - 2.3.2. Pairwise Distance . . . 16
- 2.4. Global Constraints . . . 18
- 2.5. Limitations . . . 20

3. Algorithmic Approaches 21
- 3.1. Ant Colony Optimization . . . 21
  - 3.1.1. Algorithm Overview . . . 21
  - 3.1.2. Algorithm to Solve the Grouping Problem . . . 23
- 3.2. Custom PCKMean . . . 28
  - 3.2.1. Algorithm Overview . . . 28
  - 3.2.2. Calculating Mean . . . 29
  - 3.2.3. Algorithm to Solve the Grouping Problem . . . 30

3.3. Occurrence Ranking Algorithm 35
3.3.1. Algorithm Overview 35
3.3.2. Algorithm to Solve the Grouping Problem 35

4. Validation and Evaluation **39**
4.1. Setup and Criteria 39
4.2. Results 42
4.2.1. Custom PCKMean (CPCK) 44
4.2.2. Occurrence Ranking (OCCR) 46
4.2.3. Ant Colony Optimization (ACO) 47
4.2.4. Lower Bound and Gini-Index 50
4.3. Discussion 50

5. Conclusion **52**

A. Appendix **54**

List of Figures **57**

Bibliography **61**

1. Introduction

Forming groups is a common task in various fields, ranging from assigning employees to subtasks to assembling project teams. While automatically creating randomized groups is straightforward, the task becomes more complex when groups need to be formed based on specific criteria or desired attributes. In collaborative environments, it is often crucial to incorporate specific criteria into the group formation process. This applies both in academic settings, where a structured group composition enhances the learning experience, and in professional contexts, where a diverse knowledge base within a team supports efficient project execution. The criteria can include multiple factors, such as the individual preferences of members who may have a preferred group in mind as well as key attributes deemed important by the Group Manager. Depending on the desired distribution within the groups, attributes must be treated differently. For example, certain attributes such as similar preferences may be grouped homogeneously, while others, like skills, may benefit from a heterogeneous distribution to ensure a well-balanced team.

Balancing multiple attributes while optimizing group composition quickly becomes computationally challenging. The grouping problem is a combinatorial optimization problem, classified as NP-hard [RFQCMMS20]. As the number of members increases, the factorial growth in possible groupings makes a manual assignment into optimal groups an infeasible challenge, especially when multiple attributes need to be considered. Given this complexity, a structured and efficient approach becomes indispensable.

This thesis addresses the group formation problem by exploring algorithmic approaches that enable structured and near-optimal solutions. We develop a generalized framework that integrates diverse criteria and constraints into a unified optimization model, facilitating systematic analysis and comparison of different algorithms. This methodology provides a solid foundation for enhancing the group formation process and evaluate the effectiveness of different approaches.

1.1. Grouping Problem

Generally, the grouping problem refers to the question of how objects in a set can be optimally divided into groups [RFQCMMS20]. In this context, the objects are referred to as members. According to [RFQCMMS20], the grouping problem is defined as follows. Consider a set V of n members and its subsets (groups) G_i, where $i \in \{1, 2, \ldots, l\}$. The grouping problem involves finding an optimal partition of the set V into groups such that:

$$\begin{aligned} G_i \subseteq V, \quad & \text{for each } i \in \{1, 2, \ldots, l\} \\ & G_i \cap G_j = \emptyset, \quad \text{for } i \neq j \\ & \bigcup_{i=1}^{l} G_i = V \end{aligned}$$

1.1.1. Combinatorial Optimization Problems

As described in [IG04] combinatorial optimization problems are generally difficult to solve due to their complexity. The runtime of an exact algorithm for an NP-hard problem is in the worst case exponential. If we consider n members, l groups with $g = \frac{n}{l}$ members each, we have

$$\frac{\binom{n}{g}\binom{n-g}{g}\ldots\binom{n-(l-1)g}{g}}{l!} \tag{1.1.1}$$

different combinations to partition the members into groups. The calculation is derived from the fact that for the first group, g members can be selected from n members, for the second group, g members from $n-g$ members, and so on, until there are g members left for the last group. Since there are l different groups, the result is divided by the $l!$ possible permutations for the arrangement of the l groups (cf. [MZ20]). By rearranging Equation 1.1.1, we derive

$$\frac{n!}{(g!)^l l!} \tag{1.1.2}$$

This equation shows that the number of possible combinations increases factorially with the number of members. This factorial growth makes it infeasible to compute all possible combinations within a realistic time frame. As a result, determining the globally optimal solution with deterministic methods becomes impractical, making ap-

proximate approaches essential for obtaining near-optimal solutions within a reasonable computational time.

1.1.2. Representation

In this section, we discuss the fundamental definitions necessary to formally represent the grouping problem. The definitions were derived from the general definitions of optimization problems in [IG04] and [DS], and specifically adapted for the grouping problem.

Search Space and Objective Function

Optimization problems define a search space S and an objective function $f_{obj} : S \to \mathbb{R}$ which assigns a score to each possible solution in the search space. The search space S represents the set of all possible solutions. A feasible solution $s \in S$ for the grouping problem is the complete partitioning of all members into disjoint groups. The optimal solution is searched with respect to the objective function, which determines the quality of a solution. In order to find an optimal solution for the grouping problem, we define the following objective function, which is minimized throughout the process:

Let $s \in S$ be a feasible solution for the grouping problem. The objective function $f_{obj} : S \to \mathbb{R}$ is given by:

$$f_{obj}(s) = d(s) \tag{1.1.3}$$

where $d(s)$ is the total distance of the solution s, computed as:

$$d(s) = \sum_{i=1}^{l} d(G_i) \tag{1.1.4}$$

Here, $d(G_i)$ denotes the intra-group distance of group G_i, calculated as the sum of all pairwise distances between its members:

$$d(G_i) = \sum_{\substack{v_q, v_w \in G_i \\ w \neq q}} pdist(v_q, v_w), \tag{1.1.5}$$

A detailed definition of the pairwise distance function $pdist(v_q, v_w)$ between members v_q and v_w is provided in Section 2.3.

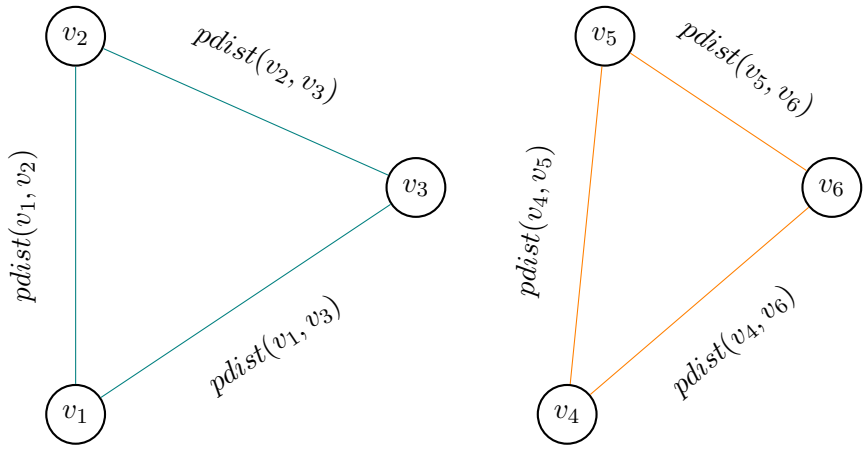


Figure 1.1.: Distance of a solution

As an example consider a given solution $s \in S$, as illustrated in Figure 1.1. We consider two groups $G_1 = \{v_1, v_2, v_3\}$ and $G_2 = \{v_4, v_5, v_6\}$. The intra-group distances are computed using the pairwise distances between members:

$$d(G_1) = pdist(v_1, v_2) + pdist(v_1, v_3) + pdist(v_2, v_3)$$
$$d(G_2) = pdist(v_4, v_5) + pdist(v_4, v_6) + pdist(v_5, v_6)$$

The objective function is the sum of the intra-group distances:

$$f_{obj}(s) = d(s) = d(G_1) + d(G_2)$$

Construction Graph

The grouping problem can be conceptualized as a fully connected graph, where each node represents a member, and the edges between two members are weighted by their respective pairwise distances. Formally, this can be expressed as follows. For a finite set $V = \{v_1, v_2, \ldots, v_n\}$ of n members the construction graph G_V is defined as $G_V = (V, L)$, where V is a finite set of members, and L is the set of edges that fully connect the members of V. With this setup, we obtain a weighted adjacency matrix, later termed as the distance matrix, where the pairwise distances between the members are stored.

1.1.3. Terminology

Tables 1.1 and 1.2 provide an overview of the terminology used in this work, which we will consistently follow.

Designation	**Definition**
member	An individual with specific attributes.
Group Assembly	Complete partitioning of n members into l disjoint groups.
Group Manager	Administrator of a Group Assembly. Defines all criteria for group formation.
Feature	Measures a characteristic of a member.
Group Size Constraint	Ensures that all groups are of equal size.

Table 1.1.: General terminology

Notation	**Definition**
$S = \{s_1, s_2, \cdots, s_h\}$	Search space: set of all h possible solutions.
$f_{obj} : S \to \mathbb{R}$	Objective function: assigns a score to each solution $s \in S$.
G_V	Construction graph: graph representation of the grouping problem.
n	Number of members in one Group Assembly.
l	Number of groups one Group Assembly has.
g	Number of members one group has.
$V = \{v_1, v_2, \cdots, v_n\}$	Set of n members. Each member is represented as a vector.
m	Number of features or equivalently: The dimension of the vector representing the member.
$G = \{G_1, G_2, \cdots, G_l\}$	Set of all individual groups $G_i, i \in \{1, 2, \cdots, l\}$
F	Set of all features within a Group Assembly.
$f_{ij} \in F$	Single feature of a Group Assembly. $i \in \{1, 2, \cdots, n\}$ is the index of the member, $j \in \{1, 2, \cdots, m\}$ is the index of the feature.
C	Set of must-links and cannot-links.
$c_{=}(v_i, v_q)$	Must-Link Constraint: requires, that member v_i and v_q belong to the same group.
$c_{\neq}(v_i, v_q)$	Cannot-Link Constraint: requires, that member v_i and v_q cannot be assigned to the same group.
$d(s)$	Total distance from a solution $s \in S$.
$d(G_i), i \in \{1, 2, \cdots, l\}$	intra-group distance: distance between all members of one group.
$pdist(v_i, v_q)$	Pairwise distance between member v_i and v_q.

Table 1.2.: Mathematical terminology

1.2. Related Work

In our work, we focus on the question of how individuals can be grouped while respecting certain constraints. However, grouping problems are central to a wide range of domains, from industry and logistics to education [RFQCMMS20]. In [RFQCMMS20], seventeen metaheuristics from different studies addressing grouping problems are compared. This demonstrates the considerable effort dedicated to this topic in the research community. This section provides an overview of the wide range of studies that consider the grouping problem and the methods used to solve it.

In the context of collaborative learning, the literature generally distinguishes between homogeneous and heterogeneous groups [BDC09, ZCAY+10, GB06, PTC+20, OMB19]. Homogeneous groups are understood to be groups whose members have similar skills, knowledge, or preferences. Heterogeneous groups, on the other hand, are made up of members who contribute as many different skills, knowledge, and preferences as possible. Most of the reviewed studies in [OMB19] advocate the formation of heterogeneous groups in terms of collaborative learning. This aligns with the findings of [ZCAY+10], which emphasize improved learning outcomes achieved through balanced groups in programming courses. An exception is the approach from CATME Team-Maker described in [LLOR10]. In [LLOR10], it is proposed that individual criteria can be classified as either homogeneous or heterogeneous. We will delve deeper into this approach in Subsection 1.2.2

1.2.1. Group Forming Methods

The standard method for discovering natural groupings within a data set is clustering. In [ViC21], heterogeneous groups are formed using K-Means clustering in combination with a heterogeneous grouping algorithm.

Another approach that is mentioned in different studies to form groups, is the genetic algorithm, a search algorithm based on natural selection [OMB19, ZCAY+10, MOV12]. Genetic algorithms prove to be good optimization methods for systematically forming balanced groups [ZCAY+10]. The generality of this algorithm has the advantage that it can be applied to a wide range of different grouping problems. The implementation varies depending on the problem. It has been shown that the genetic algorithm produces high quality results for a wide range of grouping problems [RFQCMMS20].

Another class of algorithms to solve the grouping problem are swarm intelligence algorithms such as Ant Colony Optimization (ACO) and Particle Swarm Optimization (PCO) [OMB19, RFQCMMS20]. In [GB06], ACO is used to maximize the heterogeneity of the groups based on the goodness heterogeneous values of all groups.

1.2.2. Group Forming Tools

Feedbackfruits

TU Delft launched a group forming tool called *feedbackfruits*[1]. They mainly focus on online learning tools and research, but they also have a group formation tool. The grouping tool allows instructors to efficiently and automatically form groups of students. The groups are created based on the instructor's desired criteria. When configuring the groups, it must be determined whether the groups should be homogeneous or heterogeneous. For homogeneous groups, a normalized similarity score is calculated, while for heterogeneous groups, a normalized dissimilarity score is used. A more detailed description of the algorithm used can be found on their webpage [Fee].

Team Maker CATME

The web-based system CATME Team-Maker [CAT] offers automated group formation based on criteria defined by the instructor. It is available for free upon request to faculty accounts. Detailed descriptions of the supported criteria and how the algorithm forms groups are published in [LLOR10]. Team-Maker adopts the approach that groups can include both homogeneous and heterogeneous criteria. In our work, we pursue this approach as well. The algorithm employs a max-min heuristic, enabling instructors to assign weights to each criterion and classify them as either homogeneous or heterogeneous. The web tool supports four types of questions:

- Multiple-choice questions: students select exactly one option from a list of possible options.
- Choose-any-or-all questions: students select all applicable statements.
- Schedule compatibility questions: students indicate their availability for the upcoming project.

[1] https://help.feedbackfruits.com/en/articles/9556175-group-formation-algorithm-explainer

- Pairing of underrepresented team members: this focuses on attributes such as gender and/or ethnicity.

Each question type is associated with a specific heuristic, which assigns a question score in the range [0,1]. A score of 0 corresponds to homogeneity in the considered group, while a score of 1 corresponds to heterogeneity. Detailed calculations for these heuristics are outlined in [LLOR10]. Additionally to the question scores for each criterion, the algorithm computes a compliance score for each group, representing the average of all question scores for that group. A higher compliance score indicates a better match among group members. The algorithm initially assigns members to groups randomly, based on the desired group size. It then employs a hill-climbing optimization method to maximize the smallest compliance score across all groups, ensuring balanced and effective group composition.

1.3. Contribution

In this work, we follow a similar approach to group formation as in [LLOR10]. Unlike the approach proposed by Feedbackfruits, which considers homogeneity and heterogeneity at the group level, we consider these aspects at the level of individual characteristic. This allows for more flexible group formation, where both similarities and differences between members can be considered simultaneously.

Team-Maker applies distinct heuristics for each question type to compute individual question scores. The overall compliance score is derived through a multi-objective optimization process (cf. [IG04]), where each criterion is either maximized or minimized, with the overall function being maximized [LLOR10]. In contrast, our method transforms all compatibility measurements into a minimization problem based on a single factor. We define a member-to-member distance as a comprehensive measure of compatibility, integrating all characteristics while distinguishing between homogeneous and heterogeneous features. These distance values can be precomputed and stored, allowing group formation to be explored using different algorithms without re-evaluating compatibility from scratch. This design ensures that the core principles of group formation remain independent of the algorithmic implementation, enabling any optimization algorithm to operate solely as a minimization problem, regardless of how a given feature contributes to compatibility.

Our key contributions can be summarized as follows:

- A unified feature encoding technique that accommodates various feature types, enhancing the accuracy of member compatibility analysis.
- A transformation of compatibility measurements into a single-objective optimization framework, simplifying the computational process.
- Three algorithmic approaches for group assembly: Ant Colony Optimization (ACO), a customized PCKMeans clustering method (CPCK), and the Occurrence Ranking (OCCR) approach.
- A comprehensive performance evaluation of the proposed algorithms, assessing both effectiveness and efficiency.

1.3.1. Individual Attribution

As this thesis is a joint effort by two authors, we outline their individual contributions. Chapter 1, the ACO algorithm presented in Section 3.1, and its evaluation in Subsection 4.2.3 were primarily led by Viola Meier, who also developed the graphical user interface (GUI), shown in the Appendix A. Chapter 2, the CPCK algorithm described in Section 3.2 along with its evaluation in Subsection 4.2.1, as well as the OCCR algorithm presented in Section 3.3 and its corresponding evaluation in Subsection 4.2.2, were primarily led by Andrin Müller. The remainder of this thesis was a collaborative effort.

2. Member Characteristics

In order to make a grouping decisions, information about the individual members is required. Many aspects relevant to the group formation, such as personality, compatibility, or preferences, are inherently subjective. These cannot be observed or measured through external data sources like social media profiles or online activity. For example, understanding someone's willingness to collaborate or their intrinsic motivation requires direct input rather than inferred behavior. For this reason we propose the use of a survey with predefined questions, defined by the Group Manager. This allows the collection for grouping-specific information. Furthermore, the questions can be asked in such a way that they fit into the algorithmic data format.

As a starting point, each member is represented as a vector. Mapping each member to vector facilitates mathematical operations, such as distance calculations, weight (scaling) or addition, making it easier to model relationships and patterns within the data set. All questions defined by the Group Manager for a given Group Assembly constitute the feature space F, which is an m-dimensional vector space. Hence, each member v_i is represented as a vector in this m-dimensional space. One survey questions represents one dimension (one feature) $f_{ij} \in F$, where i is the member and j the feature index.

$$\mathbf{v}_i = \begin{pmatrix} f_{i1} \\ f_{i2} \\ \vdots \\ f_{im} \end{pmatrix} \tag{2.0.1}$$

In the following chapter we define different types of survey questions (features), how they are encoded and finally how we measure distances between them. At the end of this chapter, we provide an illustrative example that demonstrates the process from a set of survey questions (and answers) to a pairwise distance between each participating member. It is important to note, that our work does not address the specific content or topic of the survey questions themselves.

2.1. Describing Features

This section explores the key characteristics of a feature: the *feature type*, which defines the data format; the *feature class*, which determines its role in grouping and the *number of possible answers*, specifying how members can respond.

2.1.1. Feature Types

The feature type refers to the kind of data used to measure a characteristic of a member. In our case, we use two distinct feature types:

Discrete Numerical Interval Features: These features are measured on a finite numerical scale without the presence of a true zero. They represent interval data, where differences between values are meaningful, but absolute comparisons (like ratios) are not. For example, this type allows for questions such as "Rate your skill in X on a scale from 1 to 3."

Categorical Nominal Features: These features are based on discrete, unordered categories. Since there is no inherent ranking or ordering of the values, they fall under the nominal category. This type is suitable for questions like "What is your favorite programming language?"

2.1.2. Feature Class

When forming groups, a key consideration is the distinction between homogeneous and heterogeneous groups [OMB19]. In homogeneous groups, the goal is for members to be similar in their features, whereas in heterogeneous groups, members should be dissimilar. Our work extends this concept by introducing the ability to make this distinction at the feature level. Depending on the Group Assembly, we can designate a feature as homogeneous if members should align on this characteristic, or as heterogeneous if members should differ in it. For example, consider a scenario where one question assesses a member's motivation and another evaluates a specific skill. Using our approach, we can assemble groups such that individuals with diverse talents but a shared motivation are grouped together, ensuring both alignment and diversity within the group.

FeatureID	Feature Type	Feature Class	# Answers	Encoding	Metric
hom	discrete numerical, interval	homogeneous	single	Min-Max Scaling	absolute difference
het	discrete numerical, interval	heterogeneous	single	one-hot	SPMD
hot_hom	discrete categorical, nominal	homogeneous	single	one-hot	SPMMD
hot_het	discrete categorical, nominal	heterogeneous	single	one-hot	SPMD
multi_hot_hom	discrete categorical, nominal	homogeneous	multiple	one-hot	SMC
multi_hot_het	discrete categorical, nominal	heterogeneous	multiple	one-hot	MMD

Table 2.1.: Possible features

2.1.3. Answer Cardinality

In the context of our feature distinction, the number of possible answers plays an important role. Features can allow for either single answers or multiple answers:

Single-answer features restrict members to select only one option from the available choices. For instance, when asked "What is your primary programming language?" only one selection is permitted.

Multiple-answer features enable members to choose multiple options, allowing for richer representation of their characteristics. For example, when asked "Which days do you prefer to work?" members can select all applicable options.

This flexibility ensures that features can accurately capture the complexity and diversity of member characteristics based on the type of question. Based on the feature type, the feature class and the number of answers we can determine the encoding of the feature as well as the metric used for the componentwise distances. A brief overview is shown in Table 2.1.

2.2. Encoding

The `hom` features are represented as single integers, with each `hom` feature having its own range. To ensure equal contributions from all features when calculating distances, we scale them using Min-Max Scaling as described in [HKP12] chapter 3. This transformation maps each `hom` feature to the range [0, 1]. Here, f_{ij} represents the original value, and f'_{ij} is the scaled `hom` feature j of member i.

$$f'_{ij} = \frac{f_{ij} - \min(f_j)}{\max(f_j) - \min(f_j)} \tag{2.2.1}$$

Note that the `het` features can be intervals as well. The actual encoding, however, is one-hot, where the range of the intervals represents the number of categories, and the

selected interval value corresponds to the position of the one-hot encoded value. This means that the `het` and `hot_het` features are treated in the same way, but can have different input formats.

Categorical features are also converted into binary representations using one-hot encoding, where each category is represented by a separate binary variable with values of 1 or 0, where 1 represents "category chosen" and 0 "category not chosen". In the next section we define appropriate distance metrics for each type of feature (see Table 2.1) to ensure compatibility with a single unified pairwise distance function.

2.3. Distance Metric

In this section, we first discuss the different componentwise metrics. These metrics are functions that allow us to compare the same component (feature) of a member's vector between two different members. In the second part, we introduce the pairwise distance, which aggregates each componentwise distance into a single value

2.3.1. Componentwise Distance

First, we define the componentwise distance functions for each kind of feature. Those functions can then be used for a pairwise distance function. We use a similar approach as described by [WM97]. In their work they describe the function Heterogeneous Euclidean-Overlap Metric (HEOM) that allows for the combined use of continuous and categorical features. In the literature heterogeneous distance is often refereed to as the mixed use of continuous and categorical variables. It is important to note, that we use the term "heterogeneous" only in the context of a feature class as defined in Section 2.1.2. The following function defines the distance between a feature j from two members i and q as:

$$d_j(f_{ij}, f_{qj}) = \begin{cases} \text{abs}(f_{ij} - f_{qj}), & \text{if } j \text{ is } hom, \\ \text{SPMMD}(f_{ij}, f_{qj}), & \text{if } j \text{ is } hot\_hom \\ \text{SPMD}(f_{ij}, f_{qj}), & \text{if } j \text{ is } het \text{ or } hot\_het, \\ \text{MMD}(f_{ij}, f_{qj}), & \text{if } j \text{ is } multi\_hot\_hom. \\ \text{SMC}(f_{ij}, f_{qj}), & \text{if } j \text{ is } multi\_hot\_het, \end{cases} \tag{2.3.1}$$

Single-Position Mismatching Distance (SPMMD)

The Single-Position Mismatching Distance (SPMMD) is used for the `hot_hom` features. These features allow only a single valid answer, meaning the comparison results in either a perfect match or a complete mismatch. Due to the homogeneity of this feature, a perfect match corresponds to a minimal distance of 0, whereas a mismatch results in a maximal distance of 1.

Single-Position Matching Distance (SPMD)

The Single-Position Matching Distance (SPMD) is applied to the `het` and `hot_het` features. These features also permit only a single valid answer. Due to the heterogeneity of this feature, a perfect match corresponds to a distance of 1, whereas a mismatch results in a distance of 0.

Mean Manhattan Distance (MMD)

The Mean Manhattan Distance (MMD) described by [CCT09] is used for the `multi_hot_hom` category.

$$MMD = \frac{\text{number of mismatching answers}}{\text{number of total answers}} = \frac{b+c}{a+b+c+d} \tag{2.3.2}$$

As shown in Table 2.2, b represents the count of mismatches where one answer is positive and the other is negative (1-0), while c represents mismatches where one answer is negative and the other is positive (0-1). The denominator $(a+b+c+d)$ is the total number of possible answers. The MMD returns a value of 0 if all answers match perfectly and a value of 1 if there are no matching answers. A fractional value between 0 and 1 indicates the proportion of mismatching answers relative to the total number of answers. A distance of 0 is only achieved when both members select exactly the same answers, making it a strict measure of homogeneity.

Simple Matching Coefficient (SMC)

The Simple Matching Coefficient (SMC), also described by [CCT09], is a similarity measure used for the `multi_hot_het` category.

$$SMC = \frac{\text{number of matching answers}}{\text{number of total answers}} = \frac{a+d}{a+b+c+d} \tag{2.3.3}$$

$f_{ij} \backslash f_{qj}$	1 (Presence)	0 (Absence)	Sum
1 (Presence)	a	b	$a+b$
0 (Absence)	c	d	$c+d$

Table 2.2.: Binary feature j of member i and q

As shown in Table 2.2, a represents the count of matching positive answers (1-1), and d represents the count of matching negative answers (0-0). The denominator $(a+b+c+d)$ is the total number of possible answers. The SMC returns a value of 1 if all answers match perfectly (both 1-1 and 0-0), a value of 0 if there are no matching answers, and a fractional value between 0 and 1 that represents the proportion of matching answers (both positive and negative) to the total number of answers. A distance of 0 is only achieved when both members select exactly opposite answers, making it a strict measure of heterogeneity.

2.3.2. Pairwise Distance

To derive an overall distance between two members, it is necessary to aggregate the distances computed for individual features into a single distance value. The aggregation process combines the outputs of the various componentwise distance functions into a single value, ensuring that the contributions of each feature are appropriately weighted and interpreted. The foundation of our pairwise distance function is based on Gower's work [Gow71]. It presents a widely used aggregation function for data sets with mixed variable types, including binary, ordinal, and continuous features. However, several modifications were made to accommodate the concepts of homogeneity and heterogeneity. In its original formulation, Gower's Distance assigns continuous (or in our case discrete) features a scaled similarity measure denoted as:

$$scaled\_similarity\_measure = 1 - \frac{|f_{ij} - f_{qj}|}{\max(f_j) - \min(f_j)} \tag{2.3.4}$$

If two feature values (f_{ij} and f_{qj}) are exactly the same, the output of this function is 1, indicating similarity. However, in our case, this value needs to be 0 if the two values are the same. Hence we use the absolute difference of the scaled feature values as described in Equation 2.3.1. For categorical features, Gower's Distance assigns a value of 1 for complete similarity and 0 for complete dissimilarity between categorical variables, effectively treating them as binary matches or mismatches. We split this case into two different componentwise functions to account for homogeneity and heterogeneity,

resulting in the SPMMD and SPMD functions. For multiple-answer features, we require a more nuanced measure of distance, leading to the SMC and MMD distance measures. The aggregation process, however, is the same. We calculate the weighted sum of each componentwise distance, and divide it by the number of features. It is also possible to assign a weight for each componentwise distance given by w_j. This gives the Group Manager, the ability to prioritize certain features, even after the survey is completed. Since all componentwise distances are between 0 and 1, the mean will also be in this range. This will make further comparisons and analysis easier. Therefore the pairwise distance between member v_i and member v_q is defined as:

$$pdist(v_i, v_q) = \frac{1}{m} \sum_{j=1}^{m} w_j \cdot d_j(f_{ij}, f_{qj}), \tag{2.3.5}$$

where w_j is a weight that can be defined by the Group Manager.

Based on this pairwise distance function, we seek optimal solutions to the grouping problem described in Section 1.1.

Example: Consider the following survey questions:

FeatureID	Question	Possible Answers
hom	what grade are you aiming for?	0;4, 1;5, 2;6
hot_hom	how do you prefer working?	0;in person, 1;remote, 2;hybrid
hot_het	how good are you at Python?	0;beginner, 1;intermediate, 2;pro
multi_hot_hom	at which days are you available?	0;mon, 1;tue, 2;wed, 3;thurs, 4;fri
multi_hot_het	what skills can you bring to the team?	0;leading, 1;writing, 3;presenting

Table 2.3.: Example survey questions

Let v_i and v_q be two distinct members, each of whom has answered the survey questions shown in Table 2.3 with the following responses:

Question	Answer member v_i	Answer member v_q
what grade are you aiming for?	6	5
how do you prefer working?	remote	in person
how good are you Python?	beginner	pro
on which days do you have time?	mon, tue, wed	tue, wed
what skills can you bring to the team	leading	leading, writing

Table 2.4.: Example survey answers for two members v_i and v_q

Based on the given answers, shown in Table 2.4, we apply the encoding described in the section 2.2 and obtain the following encoded vectors:

$$\mathbf{v}_i = \begin{pmatrix} 1 \\ [0,1,0] \\ [1,0,0] \\ [1,1,1,0,0] \\ [1,0,0] \end{pmatrix} \mathbf{v}_q = \begin{pmatrix} 0.5 \\ [1,0,0] \\ [0,0,1] \\ [0,1,1,0,0] \\ [1,1,0] \end{pmatrix} \tag{2.3.6}$$

Each component of those vectors gets now passed to its suitable componentwise distance function, based on the Equation 2.3.1

Feature	FeatureID	Componentwise Distance Function	Distance
1	hom	abs(1 - 0.5)	0.5
2	hot_hom	SPMMD([0,1,0],[1,0,0])	1
3	hot_het	SPMD([1,0,0],[0,0,1])	0
4	multi_hot_hom	MMD([1,1,1,0,0],[0,1,1,0,0])	0.2
5	multi_hot_het	SMC([1,0,0],[1,1,0])	0.65

Table 2.5.: Componentwise distances for example members v_i and v_q

We set all the weights w_j to one and obtain the weighted, mean sum of each distance from Table 2.5:

$$pdist = \frac{1}{5}(0.5 + 1 + 0 + 0.2 + 0.65) = 0.47 \tag{2.3.7}$$

Since the *pdist* always lies between 0 and 1, we can interpret the result as an approximate 47% mismatch or an 53% match between member v_i and member v_q.

2.4. Global Constraints

In our work, we impose three hard global constraints that are strictly enforced and independent of any distance calculations. These constraints cannot be violated under any circumstances. These restrictions are group size constraint, must-link constraint and cannot-link constraint.

Group Size Constraint: This constraint ensures that all groups are of equal size g, a crucial requirement for the group formation process.

Must-Link Constraint: Denoted as $c_{=}(v_i, v_q)$. This constraint requires that two members, v_i and v_q, must belong to the same group.

Cannot-Link Constraint: Denoted as $c_{\neq}(v_i, v_q)$. This constraint ensures that member v_i and v_q cannot be assigned to the same group.

Furthermore the set of all must-link and cannot-link constraints for one Group Assembly is denoted as C. In their book [BDW09] Chapter 1, Basu et al. highlighted two key observations regarding must-link and cannot-link constraints, which form the basis of their application in clustering tasks. In our work we make use of those observations.

Observation 1: Transitive Inference of Must-Link Constraints

Let G_M be the must-link graph for data set V, with a node for each $v \in V$ and an edge between nodes v_i and v_q for each $c_{=}(v_i, v_q)$ in C. Let CC_1 and CC_2 be two connected components in this graph. If there exists a must-link constraint $c_{=}(v_i, v_q)$, where $v_i \in CC_1$ and $v_q \in CC_2$, then we can infer $c_{=}(v_a, v_b)$ for all $v_a \in CC_1, v_b \in CC_2$.

Observation 2: Transitive Inference of Cannot-Link Constraints Let G_M be the must-link graph for data set V, with a node for each $v \in V$ and an edge between nodes v_i and v_q for each $c_{=}(v_i, v_q)$ in C. Let CC_1 and CC_2 be two connected components in this graph. If there exists a cannot-link constraint $c_{\neq}(v_x, v_y)$, where $v_x \in CC_1$ and $v_y \in CC_2$, then we can infer $c_{\neq}(v_a, v_b)$ for all $v_a \in CC_1, v_b \in CC_2$.

2.5. Limitations

The first limitation concerns must-links. All the algorithms discussed in Chapter 3 are designed to handle must-link constraints. However, for each set of members with must-link relationships, there must also be enough additional members without must-link constraints to complete the group. This means that merges cannot occur between two separate sets of must-linked members.

For example, consider a Group Assembly with a required group size of four members. There are two members that are connected by a must-link relationship. To complete the group, two additional members are needed. These additional members must not have a must-link relationship with each other.

A second limitation concerns the number of members. When the total number of members, n, is not divisible by the group size g, we address this issue by artificially creating additional members and imposing cannot-link constraints between them. The number of artificially added members is given by:

$$artificially\_added\_members = g - (n \bmod g) \tag{2.5.1}$$

If more than one artificial member is added, a pairwise cannot-link constraint is imposed between each pair of these artificial members. After the grouping process, these artificial members are removed from the final solution.

For example, consider $n = 25$ members that should be grouped into groups of $g = 3$. We apply the formula $3 - 25\ mod\ 3 = 2$, so we add two artificial members that have a cannot-link constraint between them. After solving the grouping problem, and removing the artificial members we obtain $l = 9$ groups with sizes $2, 2, 3, 3, 3, 3, 3, 3, 3$.

3. Algorithmic Approaches

3.1. Ant Colony Optimization

The Ant Colony Optimization (ACO) metaheuristic was developed by Dorigo, Maniezzo and Colorni [IG04]. Many of the existing variants were originally tested using the Traveling Salesman Problem as a benchmark [DS], but shows excellent results for various discrete problems [RFQCMMS20]. In [GB06] the algorithm is applied to the formation of heterogeneous groups. We adopt this approach and adjust the algorithm to minimize the objective function defined in Equation 1.1.3. The algorithm implemented is an adaptation of the algorithm implemented on Datacamp [McK], where the algorithm solves the Travel Salesman Problem.

3.1.1. Algorithm Overview

The ACO algorithm is inspired by the collective behavior of ant colonies. This metaheuristic approach solves combinatorial optimization problems that can be represented as graphs [GB06]. Ants solve complex tasks by indirect communication using pheromone trails. By laying and following pheromone trails, ants can find the shortest path to a food source [DS]. In the absence of any pheromone trails, the ants choose a random path. If they detect a trail, they follow it with a probability proportional to the intensity of the trail [IG04]. To illustrate this, consider the two ants in Figure 3.1: ant a_1 follows the lower path of distance d_1 and another ant, a_2, follows the upper path of distance $d_2 = \frac{1}{2}d_1$. a_1 reaches the food source at time $t_1 = 1$, while a_2 has only traveled halfway through. At time $t_2 = 2$, a_1 has completed its journey and a_2 has reached the food source. Given the deposited pheromone, as shown in Figure 3.2, a_2 will choose the shorter path back with higher probability. This principle applies to the entire colony (cf. description in [DS]).

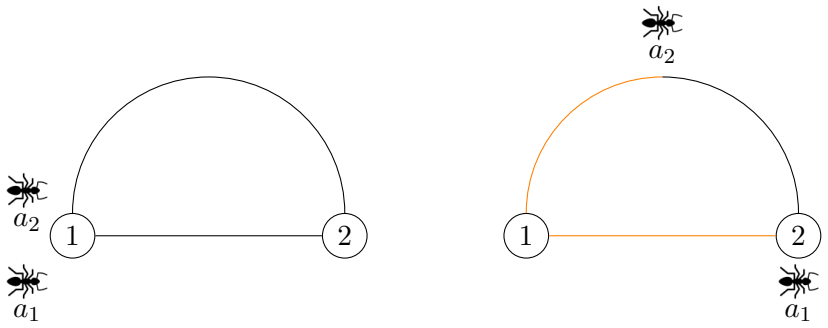


Figure 3.1.: Trail laying

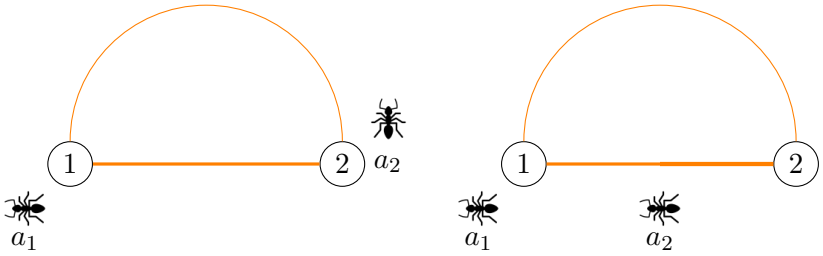


Figure 3.2.: Trail following

Key Concepts

The following explanations were mainly taken from the book [DS] to explain the key concepts and steps of the algorithm. Equations and definitions have been adopted from [DS] to ensure completeness and consistency. Adjustments in solution construction for the grouping problem follow the principle described in [GB06], with modifications to the equations made to align with those principles as well as our specific problem setting.

The ACO method employs a stochastic constructive process where each 'artificial ant' builds a solution step by step. These solutions evolve through randomized walks across the fully connected construction graph G_V defined in Section 1.1. At each step in the solution process, the next member to be added to the solution is selected according to probability rules. The probabilities are based on pheromone trails and on heuristic information.

Pheromone trails, defined as τ_{ij} for the edge between the member v_i and v_j, store information about the quality of previously discovered solutions. They promote paths that have led to successful outcomes. The trails are represented in an adjacency matrix, initially set to one.

Heuristic information, defined as η_{ij} for the edge between member v_i and v_j, provides knowledge of the current problem instance. At each decision point, it refers to the benefit of adding a member to an existing group. Let G_q be a pre-established group with v_i being the latest member added. The heuristic information for considering the addition of another member v_j to this group is defined as:

$$\eta_{ij} = \frac{1}{\sum_{v \in G_q} pdist(v, v_j)} \tag{3.1.1}$$

Although the most recently added member is v_i, the heuristic value considers the overall compatibility of v_j with the entire existing group G_q. This is achieved by defining the heuristic information η_{ij} as the inverse of the sum of pairwise distances between member v_j and all members v in the group G_q.

3.1.2. Algorithm to Solve the Grouping Problem

In this subsection, the individual steps of the algorithm are described in detail. As in the previous subsection, the equations and concepts from [DS] were adopted and adapted to the construction of groups following [GB06]. The difference compared to [GB06] lies in the optimization approach. Our definition of distances in Section 2.3.2 allows homogeneous and heterogeneous features to be combined in such a way that the entire problem is reduced to a minimization problem. In contrast, [GB06] defined an objective function that maximizes group heterogeneity, thereby framing it as a maximization problem. The remainder of this subsection describes how solutions are constructed in an iterative process and how the interaction of pheromone trails and heuristic information influences the decision making for the next component of the solution. The key steps are outlined in Algorithm 1. The individual steps will be discussed in more detail below.

Algorithm 1 ACO main procedures

```
Initialize empty solution s = [[None,...,None],...,[None,...,None]] with length(s) =
  number of groups and each length(s[i]) = number of members within one group
for k iterations do
    for l solutions do
        Solution Construction
        Evaluate Solution
        Update Pheromone
    end for
end for
```

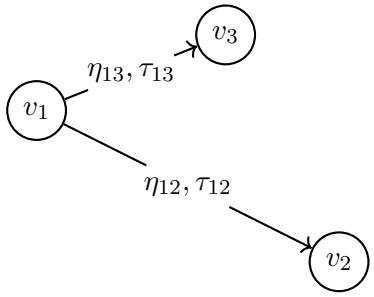


Figure 3.3.: Decision making

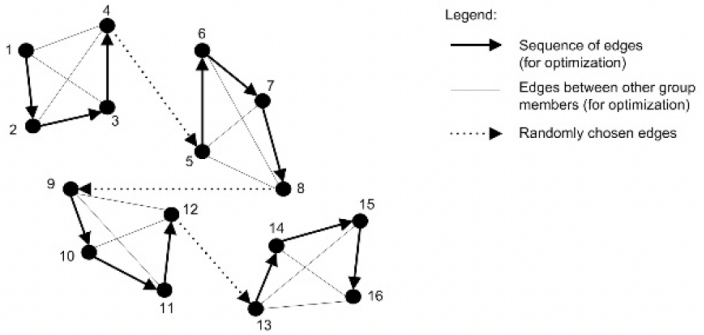


Figure 3.4.: Representation of a solution (from [GB06])

Solution Construction

Each group $G_q \in G, q \in \{1, 2, \cdots, l\}$ is started with a randomly chosen member. Similar to a Greedy Algorithm the solution is constructed step by step via intermediate solutions [IG04]. Each construction step determines the optimal addition of members to an existing group. The probability of choosing member v_i after v_j is given by:

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \ \eta_{ij}^{\beta}}{\sum_{v_w \in N_i} \ \tau_{iw}^{\alpha} \ \eta_{iw}^{\beta}} \qquad \text{if } v_w \in N_i \tag{3.1.2}$$

where N_i is the set of ungrouped members. The parameters α and β determine the influence of the pheromone trail and heuristic information on the decision making. Edges between members are used to indicate the quality of the group in terms of the overall solution, encoded in pheromone trails, and carry heuristic information. Both factors are taken into account in every decision-making process, as illustrated in Figure 3.3.

A new group member v_j is always assigned to every other group member in G_q. The construction of the solution and the relevant edges within a group is illustrated in Figure 3.4 as published in [GB06]. The order in which the members are added to a group has no significance, the important criterion is that all members within a group are connected.

Evaluate Solution and Update Pheromone

Pheromone trails are increased for each constructed solution and are inversely proportional to the quality of the solution, quantified by the objective function $f_{obj} : S \to \mathbb{R}$. The pheromone trail update between member v_i and member v_j is therefore defined as

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{h} \Delta\tau_{ij}^{k} \tag{3.1.3}$$

where $\Delta\tau_{ij}^{k}$ is the amount of pheromone from solution s_k and h is the number of solutions constructed in one iteration. $\Delta\tau_{ij}^{k}$ is defined as:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{f(s_k)}, & \text{if } v_i \text{ and } v_j \text{ are grouped together in } s_k \\ 0, & \text{otherwise} \end{cases} \tag{3.1.4}$$

The objective function is intended to be minimized, meaning that a good solution is characterized by a lower value. Consequently, the definition in Equation 3.1.4 ensures that higher pheromone levels are deposited when a solution is considered favorable.

Solution Convergence

Figure 3.5 illustrates how the algorithm's solution construction evolves over iterations. In this example, six members are divided into two groups. Nine iterations are shown, with four solutions constructed in each iteration. The heatmap of pheromone trails visualizes which members are grouped together. While in iteration 1 to 5 a wide range of potential solutions is considered, the algorithm converges toward an optimal solution as iterations progress. Poorly rated solutions are increasingly excluded. Starting from iteration 6, the algorithm begins to focus on a specific grouping solution that is clearly favored by iteration 9.

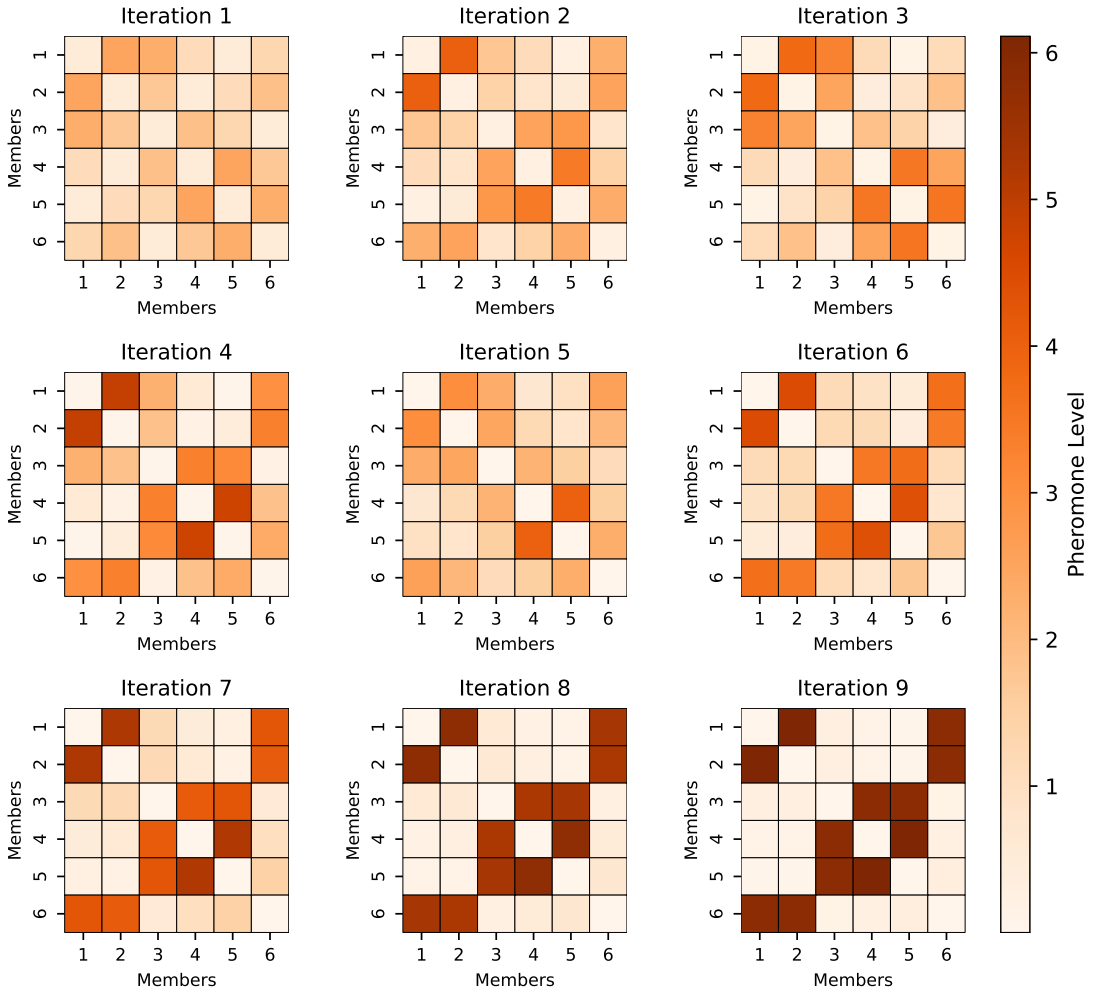


Figure 3.5.: Solution convergence

Constraints

The following explains how the constraints from Section 2.4 have been incorporated into the algorithm's implementation.

Group Size Constraint: The structure of the initially empty solution is defined in such a way that the group size is maintained (cf. Algorithm 1). The number of members and the group size can be provided as parameters to the algorithm. Once set, the structure is fixed. This represents a limitation of the algorithm, as the group size must be a divisor of the total number of members, ensuring that all groups have the same size. This restriction limits the flexibility in handling group sizes.

Must-Link Constraint: must-link constraints can be provided to the algorithm as a set of tuples (v_i, v_j) for each must-link $c_{=}(v_i, v_j)$. A list of lists is then created, containing all the members that should belong to the same group. The construction of each solution begins not with an initially empty solution as in Algorithm 1, but with the groups already formed, containing the members that are supposed to be together.

Cannot-Link Constraint: cannot-link constraints can be provided to the algorithm as a set of tuples (v_i, v_j) for each cannot-link $c_{\neq}(v_i, v_j)$. Instead of starting the grouping process with a random member, the algorithm begins by processing the cannot-links. The grouping starts with members that have cannot-links to prevent any ungrouped members from remaining at the end of the algorithm that cannot be assigned to any group due to the constraints. At each step of the algorithm, the ungrouped members are tracked and stored as a set of potential group partners. When a member is grouped having cannot-links, the respective members are removed from this set to ensure that this constraint is not violated.

Parameter Settings

The key parameters of the algorithm are the weighting of the pheromone trails (α) and the heuristic information (β), as well as the number of solutions per iteration and the number of iterations. In [DS], values in the range of 1 to 5 are recommended for the weighting of the heuristic information (β), and a value of $\alpha = 1$ is suggested for the pheromone trails to prevent the algorithm from converging too quickly to a solution. According to Dorigo [DS], these values were found to yield good solutions. The most promising results during testing were obtained with $\beta = 5$, $\alpha = 1$, along with 30 iterations and 30 solution constructions per iteration.

3.2. Custom PCKMean

In this section we discuss the algorithm Custom PCKMean, or short CPCK which at its heart is a version of a clustering algorithm. Clustering is an unsupervised learning technique that groups similar data points based on a chosen similarity measure (distances). It aims to partition data into clusters where points (members) within the same cluster are more similar to each other than to those in different clusters. Clustering methods often require adaptations to address specific constraints, such as ensuring fixed cluster sizes or incorporating relational constraints between data points. Those adaptations reflect to some degree a solution for our grouping problem. Several approaches in the literature provide solutions to these challenges, which serve as the foundation for developing the CPCK algorithm. This approaches are briefly discussed in Subsection 3.2.1, followed by a definition of the mean of two members. In the last subsection we present each step of the CPCK.

3.2.1. Algorithm Overview

[BBD00] proposed a method to prevent the formation of empty clusters by introducing a minimum size constraint. This is achieved through a constrained cluster assignment step, which is solved using linear programming. Another method for enforcing fixed cluster (or group) sizes is described by [GCT14], where each cluster is constrained to have a predefined size. In this approach, the cluster assignment step assigns each data point to the closest cluster center, provided the cluster size constraint is not violated. If the closest cluster is full, the data point is assigned to the next closest cluster. When all clusters have the same size constraint, the method resembles a Greedy Algorithm, as earlier data points can freely select their clusters, while later points have fewer options as clusters fill up. The COP-KMeans algorithm, introduced by Wagstaff et al. [WCRS01], extends the standard K-Means clustering by incorporating must-link and cannot-link constraints. The goal of COP-KMeans is to produce clusters that minimize the distance-based objective while respecting these additional constraints. During the cluster assignment step, a data point that violates a constraint (e.g., being assigned to a cluster containing a cannot-link member) is reassigned to the next closest cluster that satisfies the constraints. A semi-supervised approach to clustering is explored by Sugato Basu et al. in [BBM04], which also integrates must-link and cannot-link constraints into the clustering process. In their method, neighborhoods are formed by computing the transitive closure of must-link relationships, and the mean of each neighborhood is used as the initial cluster center. A penalty is added to the distance calculation for any

constraint violations, thereby influencing the clustering objective. The CPCK algorithm is a version of a constrained clustering algorithm that builds upon ideas and methods from the aforementioned approaches.

Algorithm 2 provides an overview of the steps involved in the proposed CPCK algorithm. Before diving into the algorithm, we first introduce how the mean of two members is calculated, an important yet non-trivial operation due to the nature of our features.

3.2.2. Calculating Mean

Because of our features, calculating the mean in a standard way is only feasible for the `hom` features. Therefore, we need a different definition for the one-hot encoded features. For the one-hot encoded features with only one possible answer (`hot_hom, hot_het`), the mean is defined as the category that occurs most often. In the case of a tie, the mean is determined by randomly selecting one of the most frequently occurring categories. A tie is handled in this manner because these features allow for only one valid answer, meaning that only a single category can have a value of 1.

For the one-hot encoded features with multiple possible answers let $v_1, v_2, \ldots, v_q$ be a set of members, and let f_j be either the j-th `multi_hot_hom` or `multi_hot_het` feature with t possible categories to choose from. Each category is denoted as b_{io}, where i represents the member and o represents the category, with $o \in \{1, \ldots, t\}$. To calculate the mean of the subset of q members, we sum the values for each category across all members and divide by q. The mean value for a category o is represented by b_{mo}. If the result is greater than or equal to 0.5, the category is assigned a value of 1; otherwise, it is assigned a value of 0:

$$b_{mo} = \begin{cases} 1 & \text{if } \frac{1}{q} \sum_{i=1}^{q} b_{io} \geq 0.5, \\ 0 & \text{otherwise.} \end{cases} \tag{3.2.1}$$

Let us illustrate calculating the mean with two examples. First we calculate the mean for `hot_hom` or `hot_het` features. Let v_i and v_q be two members with $f_{ij} = [1, 0, 0]$ and $f_{qj} = [0, 1, 0]$ The mean is a random choice between $[1, 0, 0]$ and $[0, 1, 0]$.

For the `multi_hot_hom` or `multi_hot_het` features, lets extend the previous example by member v_w and new features $f_{ij} = [1, 0, 1]$, $f_{qj} = [1, 1, 0]$ and $f_{wj} = [0, 0, 1]$. The mean of those features is $[1, 0, 1]$.

Algorithm 2 CUSTOM PCKMEANS

Require: Set of members V, integer *group_size*, integer *n_clusters*, integer *max_iterations*, list of tuples must_links, list of tuples *cannot_links*

1: **Step 1: Create neighborhoods**
2: *neighborhoods* ← `create_neighborhoods`*(must_links, cannot_links, |V|)*
3: **Step 2: Fill neighborhoods**
4: (*neighborhood_clusters*, *V_without_neighborhoods*) ← `fill_neighborhoods`*(neighborhoods)*
5: **Step 3: Perform clustering on remaining data**
6: *unbalanced_clusters* ← `kmeans`*(V_without_neighborhoods, n_clusters, max_iterations)*
7: **Step 4: Balance clusters**
8: *balanced_clusters* ← `balance_clusters`*(unbalanced_clusters)*
9: **Step 5: Solve optimization problem for each cluster**
10: *lp_solution* ← `solve_all`*(all_distance_matrices)*
11: **return** *lp_solution + neighborhood_clusters*

3.2.3. Algorithm to Solve the Grouping Problem

In the subsequent section we will discuss each step of the CPCK in detail. Algorithm 2 provides an overview of those steps.

Step 1: Create neighborhoods In the first step, we create neighborhoods following the method proposed by [BBM04]. Using the must-link constraints among members, we group them into neighborhoods, where each neighborhood corresponds to a connected component. This process relies on the Transitive Inference of Must-Link Constraints outlined in Section 2.4 Observation 1. For example, consider the members v_i, v_q and v_w. If there is a must-link constraint between v_i and v_q, $c_=(v_i, v_q)$, and another between v_q and v_w, $c_=(v_q, v_w)$, we can infer that v_i and v_w must also be grouped together, resulting in $c_=(v_i, v_w)$. This ensures that all indirect must-link relationships are captured, forming complete and consistent neighborhoods.

We further augment the cannot-link relationships using the Transitive Inference of Cannot-Link Constraints outlined in Section 2.4 Observation 2. Specifically, if member v_i and v_q share a must-link relationship, $c_=(v_i, v_q)$, and member v_i and v_w have a cannot-link relationship, $c_{\neq}(v_i, v_w)$, we can infer that v_q and v_w also have a cannot-link relationship, $c_{\neq}(v_q, v_w)$. This augmented set of cannot-link relationships is used in all algorithms.

Step 2: Filling neighborhoods We calculate the center (mean) of each neighborhood using the previously mentioned method from Subsection 3.2.2. Next, we construct a distance matrix between each unassigned member (i.e., those without must-link relationships) and the center of each neighborhood. In the subsequent neighborhood assignment step, we employ a *snake draft*, also known as a serpentine draft. This drafting method alternates the selection order in each round. In the first round, each neighborhood selects the closest unassigned member based on the distance matrix, starting with the first neighborhood, then the second, and so on. After all neighborhoods have made their first selection, the drafting order reverses for the next round, allowing the neighborhood that picked last in the first round to pick first in the second round. This process continues until each neighborhood reaches the designated group size g. The snake draft ensures fairness by giving neighborhoods that pick later in one round the opportunity to pick earlier in the next, mitigating the disadvantage of a shrinking pool of potential group members. After the snake draft all member assigned to a neighborhood are removed from the data set V. The grouping of those members is done.

Step 3: Perform clustering on remaining data K-Means++ is an initialization method for the K-Means clustering algorithm that aims to improve the choice of initial centroids [AV06]. The first centroid is selected randomly from the members. For each subsequent centroid, the algorithm calculates the distance of each data point to its nearest centroid already chosen. The probability of selecting the next centroid is proportional to the square of the distance to the nearest centroid. This means that members that are farther from the existing centroids are more likely to be chosen as new centroids. This process is repeated until k centroids are chosen. K-Means++ ensures a more spread-out initialization of centroids compared to random initialization, reducing the likelihood of poor clustering results and speeding up the convergence of the K-Means algorithm.

After the initialization of the clusters, a constrained K-Means algorithm is applied, as proposed by [WCRS01]. Here, if a cannot-link constrain is to be violated the algorithm will reassign the member to the next closest cluster that satisfies the constraints. The default number of clusters is 10% of the total number of members, rounded up to the nearest integer.

Step 4: Balance clusters For now, we do not have any control over the size of the clusters. Although [GCT14] proposed a solution for fixed cluster sizes, this would require prior knowledge of the size of each cluster, which we do not have. Furthermore, this restriction would limit the search for natural clusters in the data. For this reason, we decided to balance the clusters afterwards. The goal is to make every clusters size divisible by the group size g, so that we can apply a linear program in a later step.

To achieve balanced cluster sizes without losing too much of the result obtained during the clustering process, we aim to perform as few swaps between the clusters as possible. The `balance_clusters` function is designed to achieve balance with a minimal number of swaps. While we have not mathematically proven that it achieves the absolute minimum, extensive testing has shown no instances where the number of swaps exceeded the theoretical minimum. Note that the sum of all cluster sizes is always divisible by the required group size, because all members that have been removed until here, where removed in groups of g and the initial data set was divisible by g. First, we create two lists: one called *give* and the other called *take*. The *give* list holds the sizes of all clusters whose size modulo g is less than $g/2$, while the *take* list holds all clusters with sizes for which the modulo g is greater than or equal to $g/2$. All clusters that are already divisible by g are not considered and at every swap we check if a cannot-link constraint is violated. Now, we distinguish between three cases:

$$\textit{balance(give, take)} = \begin{cases} \textit{balance}_1(\textit{give}, \textit{take}), & \text{if } \textit{give} \neq \emptyset \wedge \textit{take} \neq \emptyset \\ \textit{balance}_2(\textit{give}, \textit{take}), & \text{if } \textit{give} \neq \emptyset \wedge \textit{take} = \emptyset \\ \textit{balance}_3(\textit{give}, \textit{take}), & \text{if } \textit{give} = \emptyset \wedge \textit{take} \neq \emptyset \end{cases} \tag{3.2.2}$$

We update the clusters according to the function $\textit{balance}(\textit{give}, \textit{take})$ as follows. If $\textit{balance}_1(\textit{give}, \textit{take})$ applies, we remove the first member from the first cluster in *give* and add it to the first cluster in *take*. If $\textit{balance}_2(\textit{give}, \textit{take})$ applies, we remove the first member from the first cluster in *give* and add it to the second cluster in *give*. If $\textit{balance}_3(\textit{give}, \textit{take})$ applies, we remove the first member from the first cluster in *take* and add it to the second cluster in *take*. These steps are repeated until both *give* and *take* are empty.

For the sake of clarity, take the following example. Consider three clusters with sizes $[5, 10, 1]$ and the group size $g = 4$. So the remainders of the cluster sizes modulo 4 are $[1, 2, 1]$. We create the two lists $\textit{give} = [5, 1]$ and $\textit{take} = [10]$. In the first iteration, $\textit{balance}_1(\textit{give}, \textit{take})$ applies, meaning we remove one member from the cluster with size 5 and add it to the cluster with size 10. After this, we recalculate both lists and obtain $\textit{give} = [4, 1]$ and $\textit{take} = [11]$. Since 4 is already divisible by 4, we can ignore it. This results in $\textit{give} = [1]$ and $\textit{take} = [11]$. In the second iterations the remainders modulo 4 are 1 respectively 3. Since both lists are not empty $\textit{balance}_1(\textit{give}, \textit{take})$ is applied again. The first cluster from the list *give* transfers its only remaining member to the

first cluster in the list *take*. We now obtain two clusters with size 4 and 12. Thus, we have balanced the cluster sizes with two swaps.

Step 5: Solve optimization problem for each cluster This step splits each cluster into groups of g members, such that the total pairwise distances within groups are minimized. Note that each cluster is treated separately, meaning there are no more swaps between two clusters. Therefore we do not have to consider cannot-link constraints anymore, since each cluster satisfies all constraints. To achieve this, the problem is formulated as a Mixed-Integer Linear Programming (MILP) problem. In our implementation we use the CBC_CMD solver from PuLP [Mit11] that is based on the simplex method. Below, we describe the components of the method:

Problem Definition

The problem is defined as minimizing the sum of pairwise distances within groups, subject to constraints that ensure valid group assignments. This is achieved by using two types of decision variables:

- **Binary Assignment Variables ($y[i, h]$)**: These variables indicate whether member i is assigned to group h.
- **Binary Pairing Variables ($z[i, q, h]$)**: These auxiliary variables capture whether two members i and q are assigned to the same group h.

Objective Function

For a given pair of members i and q assigned to the same group h, the contribution to the objective is based on their distance in the distance_matrix[i, q]. The size of the cluster (amount of members in this cluster) is denoted as $\hat{n}$ and the amount of groups we obtain from this cluster (cluster size divided by g) is given by $\hat{l}$ [2]. The objective function sums these weighted distances across all groups:

$$\text{Minimize: } \sum_{h=1}^{\hat{l}} \sum_{i=1}^{\hat{n}} \sum_{q=i+1}^{\hat{n}} z[i, q, h] \cdot \text{distance\_matrix}[i, q]$$

[2] In Chapter 1.1.3, n denotes all members of a Group Assembly and l denotes the number of groups. We can interpret one cluster as a new, reduced Group Assembly.

Constraints

The formulation is subject to the following constraints to ensure valid group assignments:

1. **Each Point Must Be Assigned to Exactly One Group**:

$$\sum_{h=1}^{\hat{l}} y[i,h] = 1 \quad \forall i \in \{1,\ldots,\hat{n}\}$$

2. **Each Group Must Have Exactly g members**:

$$\sum_{i=1}^{\hat{n}} y[i,h] = g \quad \forall h \in \{1,\ldots,\hat{l}\}$$

3. **Auxiliary Variable Definition for Pairwise Assignment**: The auxiliary variable $z[i,q,h]$ is defined to be 1 if and only if both members i and q are assigned to the same group h. This is enforced by the following constraints:

$$z[i,q,h] \leq y[i,h], \quad z[i,q,h] \leq y[q,h], \quad z[i,q,h] \geq y[i,h] + y[q,h] - 1$$

   These constraints collectively ensure that $z[i,q,h] = 1$ only when both $y[i,h] = 1$ and $y[q,h] = 1$.

It is important to note that the cluster sizes plays a key role in the feasibility of the linear program. The same complexity applies to each cluster as shown in Section 1.1. Therefore it can happen, that solving the optimization problem shown in step 5 might take a very long time to find a solution. To address this issue, a time limit should be imposed. Our experiments showed, that a default value of 180 seconds is enough to find good solutions.

3.3. Occurrence Ranking Algorithm

In this section, we present the final algorithm of our work: the Occurrence Ranking algorithm (OCCR). This represents our approach to designing an algorithm entirely from scratch. The first part of this section discusses the underlying ideas that form the basis of the algorithm, while the second part provides a detailed explanation of each step of the OCCR and in the last part we provide an illustrative example.

3.3.1. Algorithm Overview

The OCCR is based on two key observations we had during our work on this thesis. First, while the grouping problem is computationally intensive, it is feasible to compute all possible individual groups and their respective within-group distances within a reasonable time frame. The complexity of calculating all possible groups is given by $\binom{n}{g}$, which is significantly smaller than the complexity of the grouping problem, represented by $\frac{n!}{(g!)^l l!}$. This computation provides valuable insights into which groups are better or worse in terms of their within-group distances.

Second, we observed that some members can form good groups with many other members (also referred to as cluster centers), while others can only form good groups with a few members (also referred to as outliers). By exploiting these two observations, we can create a ranking where each member's rank is determined by their occurrence in the best groups. Based on this ranking we can form our groups with respect to all constraints and the objective function.

3.3.2. Algorithm to Solve the Grouping Problem

Algorithm 3 presents a step-by-step overview of the OCCR. In the following section, each of these steps is described in detail.

Step 1: Create all possible groups In the first step, we generate all possible groups from the given set of n members with the desired group size g. This is achieved using the binomial coefficient $\binom{n}{g}$, which represents the number of ways to select g members from a set of n members without regard to order. Each combination forms a unique group, ensuring that all potential group configurations of size g are considered. We then calculate the within-group distance for each group. This step has a time complexity of $\mathcal{O}(n^g)$. The value of g determines the rate of growth, with higher values of g resulting in a much steeper increase in time complexity as the input size grows. This can be observed in our experiments in Chapter 4.

Algorithm 3 OCCURRENCE RANKING

Require: Set of members V, integer $group\_size$, float α, integer max_iter, list of tuples $must\_links$, list of tuples $cannot\_links$

1: **Step 1: Create all possible groups**
2: $all\_possible\_groups \leftarrow$ `get_all_possible_groups`$(V, must\_links, cannot\_links, group\_size)$
3: **Step 2: Create occurrence ranking**
4: $occurrence\_ranking \leftarrow$ `create_occurrence_ranking`$(all\_possible\_groups, \alpha)$
5: **Step 3: Create initial grouping using a greedy approach**
6: $initial\_grouping \leftarrow$ `greedy_grouping`$(occurrence\_ranking, must\_links, cannot\_links)$
7: **Step 4: Optimize the grouping using random permutations**
8: $solution \leftarrow$ `random_permutations`$(initial\_grouping, max\_iter, must\_links, cannot\_links)$
9: **return** *solution*

Step 2: Create occurrence ranking As shown in Algorithm 3, we now create the occurrence ranking based on the parameter α. This parameter, provided by the Group Manager, represents the percentage of the best groups to consider for the ranking. Using `create_occurrence_ranking`, we first sort `all_possible_groups` in ascending order based on their within-distance. Next, we select the top $\alpha\%$ of these groups. In this reduced set of groups, we calculate the occurrence of each member. The member that appears most frequently is the one that groups well with the majority of other members. Conversely, members that occur less frequently are those that group well with only a few others — or perhaps none at all. To exploit this characteristic, we sort the occurrence ranking in descending order, ensuring that members who occur less frequently are placed at the beginning of the ranking. It is important to note that the reduced set of best groups, determined by α, must include every member at least once. Otherwise, some members would be excluded from the ranking. To ensure full coverage, α is iteratively increased by 5% until every member occurs at least once in the selected set of the best groups.

Based on extensive testing, we recommend using a default starting value of $\alpha = 10\%$. Generally speaking, a smaller value of α results in a more nuanced ranking, as it focuses only on the most optimal groups. To illustrate this, consider α set to 100%. This means that each possible group is considered and hence each member occurs exactly the same amount of times in the ranking. If we now gradually decrease α we can observe that some members will occur disproportionately more or less. This effect is what the ranking is based on.

Step 3: Create initial grouping using a greedy approach In this step, we create an initial grouping solution based on the previously ranked members. This is achieved using a greedy approach. The first member, v_i, from the `occurrence_ranking` (the member that groups the least well with the other members), initiates the process. member v_i selects the $g-1$ closest members based on the distances in the distance matrix, adhering to must-link and cannot-link constraints. If v_i has any must-link constraints, all members in its must-link set are added to the group. Additionally, if v_i selects a member v_q that has its own must-link constraints, all members in v_q's must-link set are also added to the group. However, if v_q has too many must-links, making it impossible to satisfy the group size g, v_q is discarded, and the next closest member is chosen. Similarly, if a selected member violates a cannot-link constraint, it is skipped, and the next closest member is selected.

Once a group is formed, all members in the group are removed from further consideration, and the greedy process continues with the next available member in the `occurrence_ranking`. This process repeats until all members have been grouped.

Step 4: Optimize the grouping using random permutations The final step of the Occurrence Ranking Algorithm begins with the initial_grouping and involves swapping two randomly chosen members from different groups. A swap is performed if, and only if, no must-link or cannot-link constraints are violated and the grouping solution is improved (the total sum of intra-group distances decreases). The maximum number of swaps is controlled by the parameter `max_iterations` given by the user. Our default value is 10000 iterations.

In the following example we consider a Group Assembly with four members v_1, v_2, v_3 and v_4, a required group size of $g = 2$, no must-link or cannot-link constraints and α set to 50%. With `get_all_possible_groups` we obtain a sorted dictionary that holds the group members as key and the within group distance in the corresponding values, leading to
$\{(v_1, v_2) : 0.1, (v_1, v_3) : 0.1, (v_1, v_4) : 0.3, (v_2, v_3) : 0.4, (v_2, v_4) : 0.5, (v_3, v_4) : 0.8\}$.

For the next step we only consider the best 50% of the groups, represented by $\{(v_1, v_2) : 0.1, (v_1, v_3) : 0.1, (v_1, v_4) : 0.3 :\}$. After applying the `occurrence_ranking` we retrieve the dictionary `occurrence_ranking` sorted descending, which holds the member as key and its numbers of occurrences in the subset of the best 50% of groups as value $\{v_2 : 1, v_3 : 1, v_4 : 1, v_1 : 3\}$. Because v_2 is at the beginning of the ranking (meaning v_2 is a member that is difficult to group), this member can start the `greedy_grouping` process. The member v_2 can choose its best fitting group member, which is v_1. Next, v_3 will chose v_4. This leads us to a first solution given by $\{(v_1, v_2) : 0.1, (v_3, v_4) : 0.8\}$.

In the last step we will randomly chose two members from different groups and swap them, if they do not violate any constraints and if the overall distance gets smaller. How often the OCCR should try to make a valid swap, can be determined by the user. In the inital solution a swap between v_2 and v_3 leads to the optimal solution.

4. Validation and Evaluation

Given the NP-hardness of the grouping problem, it is computationally infeasible in most real-world scenarios to determine the global optimum. Furthermore, due to the defined objective function, $f_{obj} : S \to \mathbb{R}$ in Equation 1.1.3, which takes into account both homogeneous and heterogeneous features, it is challenging to find ground truth data sets that correspond to our problem statement. In the first step, the algorithms will be validated against a baseline. This baseline represents a simple method in which groups are formed randomly. Testing against a simple random selection serves as a counterpoint to the algorithms we have implemented and aims to confirm the added value of the optimization. To evaluate the algorithms we introduce synthetic benchmarks for controlled assessment and compare the solutions of the algorithms with each other.

4.1. Setup and Criteria

This section outlines the setup for the validation and evaluation of the grouping algorithms and defines the criteria considered. To validate the solutions of our algorithms against random groupings, we consider the distance from a solution according to Equation 1.1.3. For the evaluation, we consider the following criteria:

1. **Objective Function:** The comparison of the objective function serves as an internal evaluation and provides information about the quality of a group. Comparing the performance of individual algorithms based on the objective function is intended to reveal which algorithm achieves the minimum of this function and thus comes closest to meeting the optimization goal.

2. **Runtime:** The runtime includes the calculation of the distance matrix, as well as all other calculations each algorithm computes. We run each algorithm with their corresponding default values. Note that fine-tuning the parameters for a specific configuration may lead to improved results. The CPCK has a native time limit parameter, set to the default value of 180 seconds during the test runs. This default value was determined by extensive testing.

member Id	hom_1	hom_2	hot_het_1	hot_het_2	multi_hot_hom_1	multi_hot_het_2
1						
...						
n						

Table 4.1.: Benchmark structure

3. **Gini-Index:** The calculation of the total distance of a solution does not take into account how the group distances are distributed within the grouping. To obtain a measure of the fairness of the grouping, we use the concentration measure κ, also known as the Gini-Index, according to Lorenz-Muenzer as defined in [JB15]. The Gini-Index κ represents the degree of inequality in the distribution of group distances, with a value of 0 indicating perfect equality and a value of 1 indicating complete inequality. Applied to the grouping problem, the characteristic carriers are the individual groups and the concentration of the individual within-group distances is computed. An exact definition of the calculation can be found in [JB15]. The algorithms are not optimized for the distribution of individual group distances. We consider the Gini-Index in our evaluation to assess whether the groups are formed fairly and to determine if the distribution of within-group distances should be taken into account for future optimization.

4. **Lower Bound of a Group Assembly:** As described above, it is not possible to determine the global minimum value of the objective function. However, we can calculate a lower bound inspired by the OCCR. First, we compute all possible groups and their respective within-group distances. Then, we select the l groups (where l is the number of members divided by the group size) with the lowest within-group distances. The sum of these l groups represent the lower bound. Note, that the lower bound is unlikely to correspond to a valid solution for the grouping problem, since there is a high probability that some members are in more than one group. Nevertheless, we can confidently state that no feasible solution will have a lower total distance than the sum of those groups.

Table 4.1 shows the structure of the evaluation benchmarks. There are two homogeneous features, one with random values between 1 and 3 and a second with values between 1 and 4. The one-hot encoded heterogeneous features are random permutations of [1,0,0] and [1,0,0,0]. The multiple-answer features consists of a list with length 4, where each entry is randomly set to 0 or 1. To include the must-link and cannot-link constraints in the evaluation, 20 percent of the members are randomly selected and

rounded up to the next number divisible by 4. From this subset, pairs of tuples are formed, half of which are assigned as must-links and the other half as cannot-links. The structure of the benchmarks remains consistent in terms of feature type and class, while the number of members varies.

For a comprehensive evaluation, we consider various group sizes and different numbers of members to be grouped. The evaluation parameters are specified as follows:

- Each group $G_i, i \in \{1, ..., l\}$ with group size g: for $g \in \{2, 3, 4\}$.
- Set V of n members: for $n \in \{12, 36, 60, 84, 96, 216\}$. n is selected such that it is divisible by each considered group size g.
- For each combination of the number of members n and group size g, the algorithms are run ten times. Subsequently, the average value of the previously defined evaluation criteria for each algorithm is calculated.

4.2. Results

The following section presents the evaluation results and discusses the performance of each algorithm. The objective function (distance) is reported as the average pairwise distance within each group. Specifically, the total sum of all pairwise distances — produced by the algorithms — is divided by the total number of pairwise distances in a given solution. This normalization ensures that each resulting distance lies between 0 and 1, allowing for easier interpretation. Figure 4.1 shows the validation against randomly selected grouping. The set of 84 members was chosen because it represents the midpoint of our member sizes. Figure 4.2a, Figure 4.2b and Figure 4.2c show the average pairwise distances and the runtime per algorithm for different evaluation settings and the corresponding error bars.

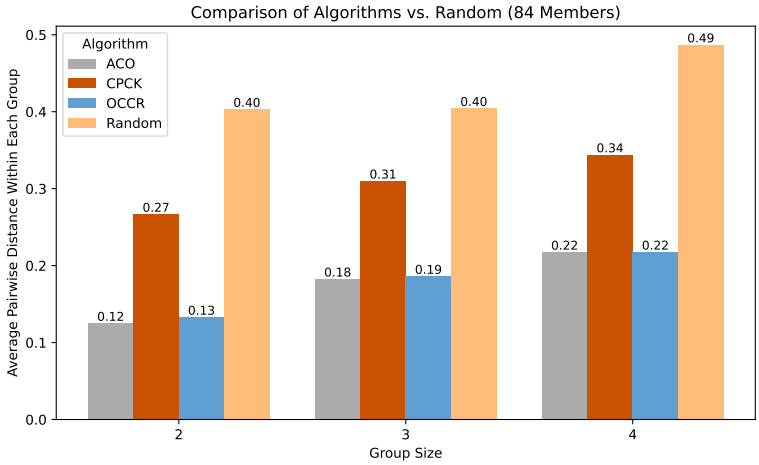
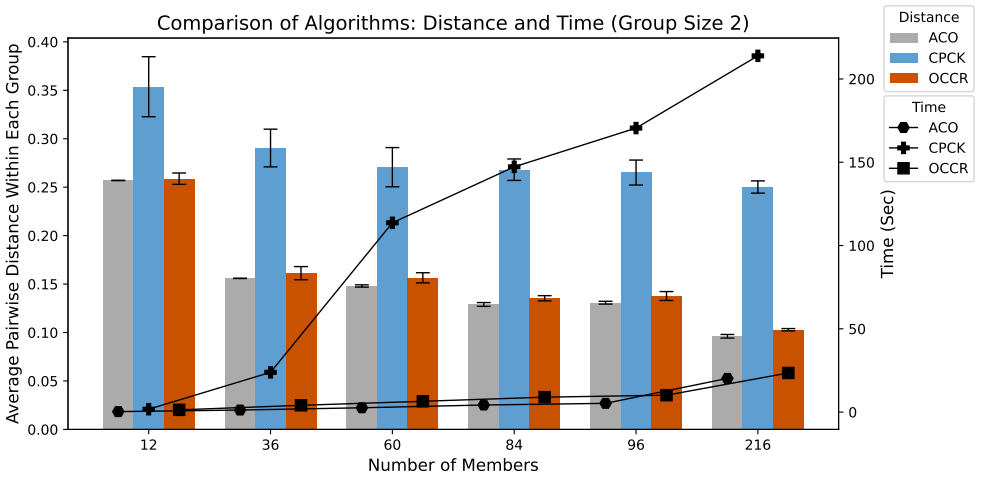
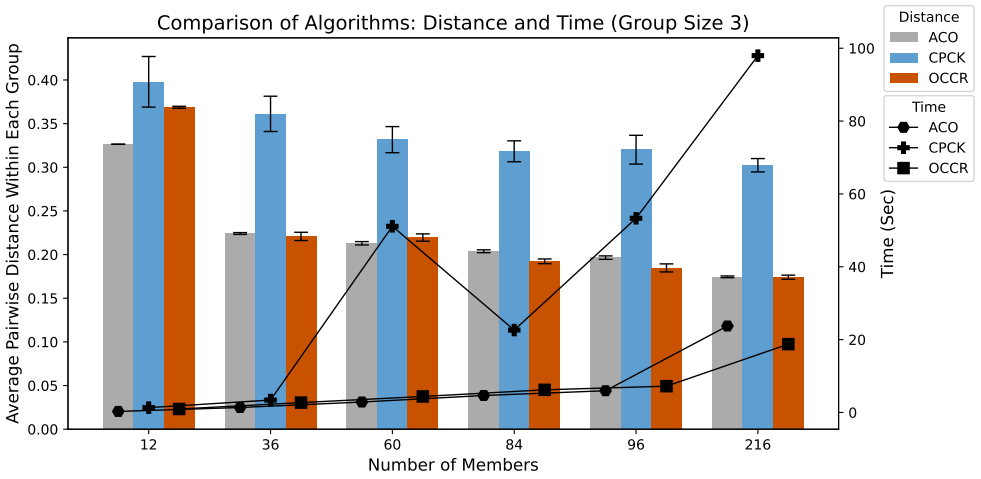


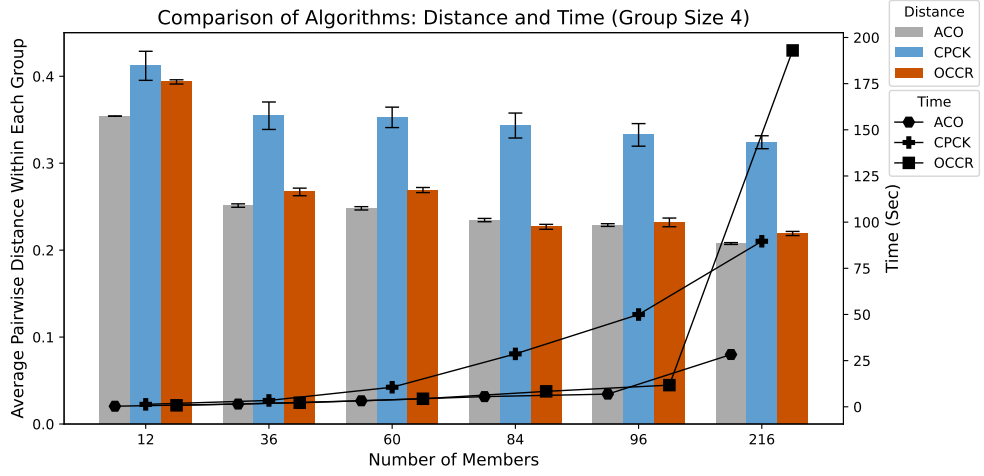
Figure 4.1.: Validation of the three algorithms versus randomly selected groups



(a) Group size 2



(b) Group size 3



(c) Group size 4

Figure 4.2.: Comparison of average time vs. average pairwise distance for different group sizes

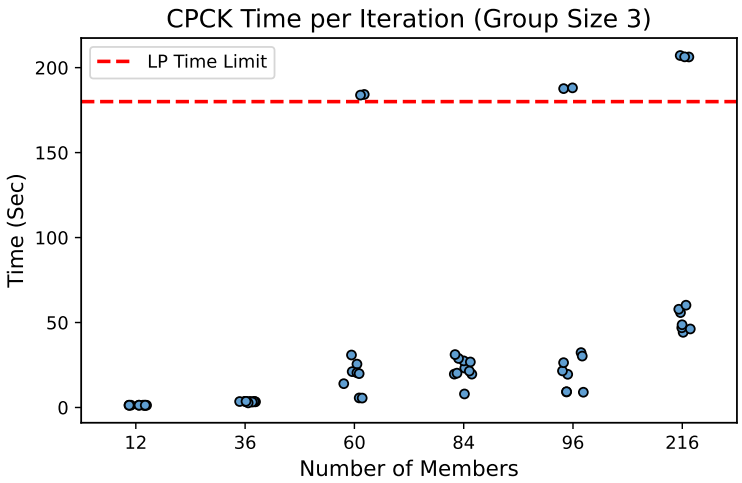


Figure 4.3.: Actual time for each iteration compared to the time limit set by the linear program for group size 3

4.2.1. Custom PCKMean (CPCK)

Out of all three algorithms, the CPCK has the longest runtime. What is particularly interesting is the spike in the average time observed in Figure 4.2b, where the number of members is 60. The algorithm was executed with the default value for the number of clusters, which is 10% of the members, resulting in six clusters. After creating these clusters, they are balanced to ensure divisibility by the group size. However, it cannot be ruled out that some clusters end up significantly larger than others. If this is the case, the larger clusters will take substantially longer to be resolved by the linear program and are likely to cause an early stop by reaching the time limit before the optimal solution is found. This can be observed in Figure 4.3, which shows the actual runtime for each of the 10 iterations. All dots above the red dotted line represent runs where a big cluster caused an early stop. If we now compare the average pairwise distances resulting from each iteration with the actual time the CPCK took, shown in Figure 4.4, we can see that reaching the time limit does not have a significant impact on the result. We can cautiously interpret this to mean that the clustering process does not work in a satisfactory manner, but the linear program step yields good results.

Another key observation is that for increased group sizes and a fixed amount of must-links, CPCK tends to exhibit faster runtimes. This can be observed, when comparing Figure 4.2a and Figure 4.2b with Figure 4.2c . This behavior can be attributed to the way must-links are processed. In Step 2 of the CPCK algorithm, each neighborhood is

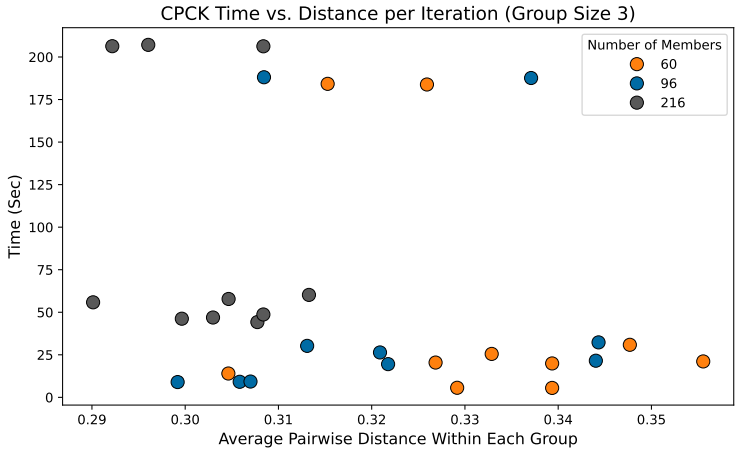


Figure 4.4.: Actual time versus pairwise distance for each iteration for group size 3

filled. Thus, the number of members assigned to a neighborhood increases, the remaining set of members to be processed decreases. This effect becomes apparent when comparing the average runtime of CPCK for 216 members between group sizes 3 and 4. In the case of 216 members, there are 22 members with must-links. These must-links are filled, resulting in a total of 44 members being removed before the clustering process begins (compared to only 33 members removed in the case of group size 3). This reduces the probability of forming large clusters, thereby increasing the likelihood of solving the linear program before the time limit is reached.

When analyzing the average pairwise distance, CPCK again ranks the lowest. In comparison to the other algorithms, CPCK narrows down the search space of solutions quickly due to its clustering approach. Once clusters are formed and balanced, there are no further interactions between members of different clusters. As a result, poorly formed clusters or a set of members that can not be well clustered can have a significant impact on CPCK's performance.

In conclusion, the CPCK algorithm shows promise in terms of its clustering approach but exhibits notable challenges, particularly in runtime and cluster balancing. The larger clusters tend to cause delays, resulting in early stops when the time limit is exceeded. However, despite these spikes in runtime, the impact on the quality of the results is minimal. Furthermore, CPCK demonstrates improved stability with larger group sizes due to better management of must-links. Although it ranks lowest in terms of average pairwise distance and time, its clustering process helps to narrow down the solution space quickly, highlighting both its strengths and areas for further refinement.

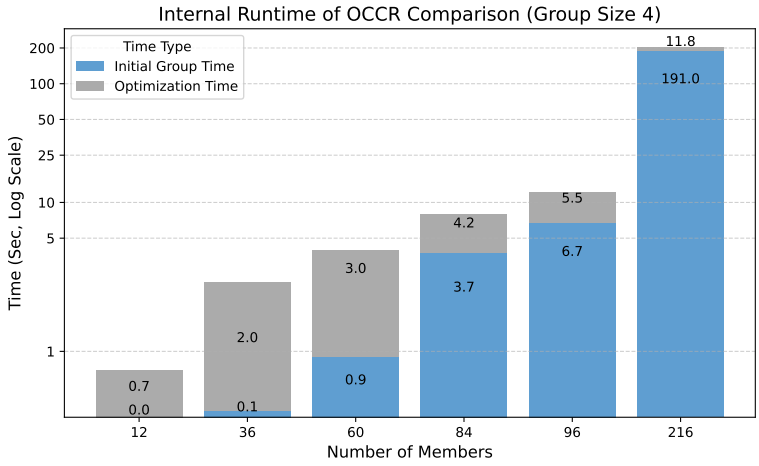


Figure 4.5.: OCCR's internal run times compared for group size 4

4.2.2. Occurrence Ranking (OCCR)

Based on our evaluation, time and distance wise, the OCCR algorithm holds the second place when compared to CPCK and ACO. In a few instances, the OCCR algorithm shows better results than the ACO with regard to distances, and in one case, shown in Figure 4.2b, at 216 members, it even achieves a better run time. However, we can observe that the runtime increases significantly with the number of possible groups. This is particularly noticeable when comparing the runtime for 216 members with group sizes 3 and 4, as shown in Figures 4.2b and 4.2c. Here, we observe a tenfold increase in runtime, caused by the rise in the number of possible groups, from 1,656,360 for group size 3 to 88,201,170 for group size 4. Figure 4.5 compares the internal runtimes for one run of OCCR with a group size of 4 (as an example). The Initial Group Time refers to the time it took for OCCR to complete steps 1 to 3, as described in Subsection 3.3.2, during which the initial grouping is formed. The optimization time corresponds to step 4. It can be observed that the optimization time grows underproportionally to the number of members. This increase is due to the fact that in the optimization step, the distances need to be recalculated to check if the swap results in a better solution.

While the OCCR algorithm performs competitively with respect to both time and distance, its runtime does not scale well with the increase in the number of possible groups. This scalability issue highlights the challenges of using OCCR in large-scale scenarios, though it remains an efficient choice in cases where the number of members is manageable.

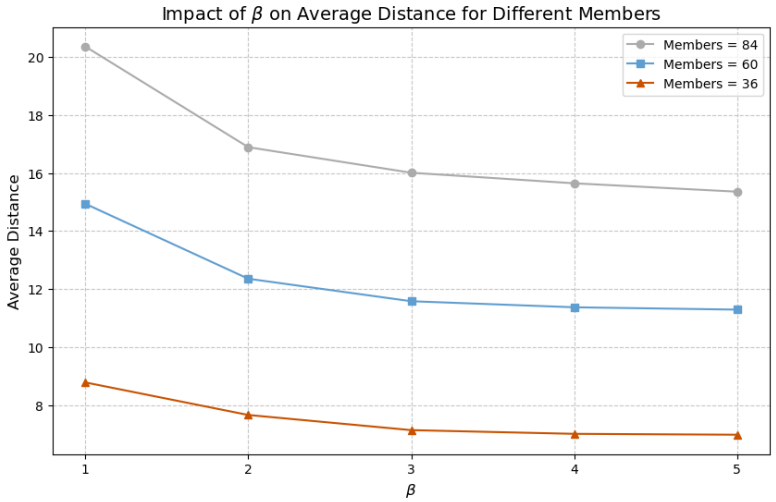


Figure 4.6.: Impact of β on average distance for group size 3

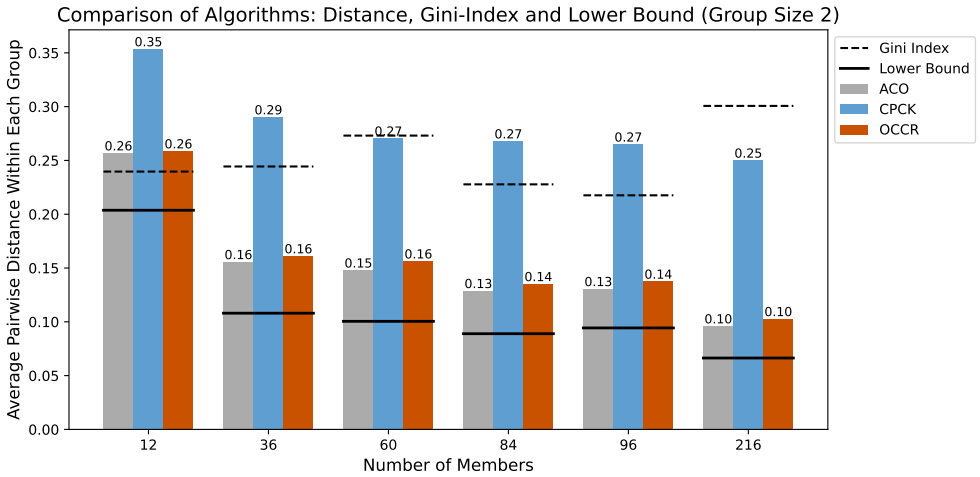
4.2.3. Ant Colony Optimization (ACO)

Through testing, it has been shown that the parameter settings of the ACO algorithm have a significant impact on optimization. The most important parameters are the weighting of the pheromone trails, the heuristic information, the number of iterations and the number of solution construction per iteration. For the evaluation, 30 iterations and 30 solution constructions per iteration were chosen. These parameters strongly depend on the number of members. Considering the possible combinations for grouping depending on the number of members, it is reasonable to adjust the number of solution construction accordingly to achieve better results. When a larger number of members is present, more potential solutions can be explored per iteration, which enables better optimization. The number of iterations and solution constructions per iteration were set in such a way that good results can also be achieved for a larger number of members using the same settings.

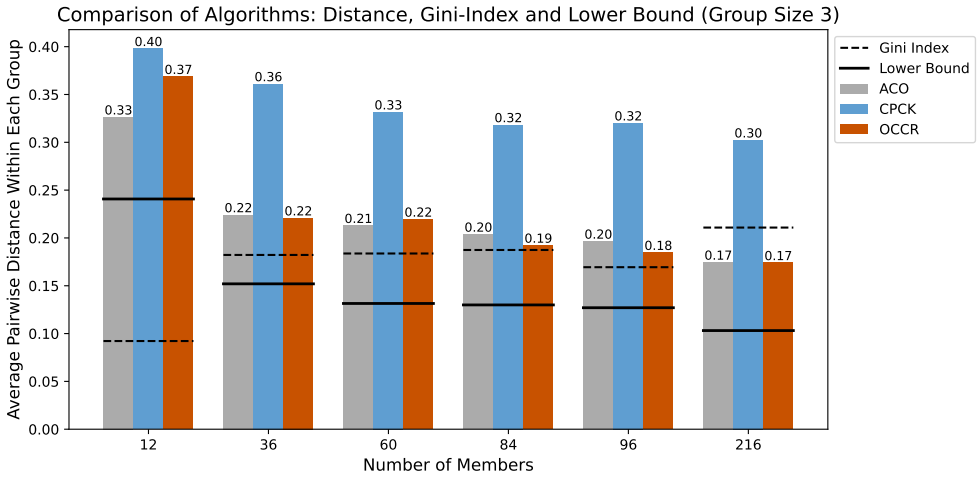
The weighting of the heuristic information (β) also has a significant impact on the solution. Figure 4.6 shows the average total distance of solutions based on 10 runs for different values of β. The results are shown for 36, 60 and 84 members, each with a group size of 3. The group size of 3 was selected because it represents the midpoint of the considered group sizes and the number of members represents a range that lies in the middle of our member sizes. The best solutions were achieved with $\beta = 5$. This aligns with the statement in [DS], where a value of β between 2 and 5 is recommended.

Figure 4.2a, 4.2b and 4.2c show that the ACO algorithm effectively minimizes the objective function and provides comparable results to the OCCR algorithm regarding both distance and runtime. In Figure 4.2a, the ACO shows slightly better results than the OCCR in terms of distance and runtime. However, for group sizes of 3 and 4, the distance results are comparable, as shown in Figure 4.2b and 4.2c.

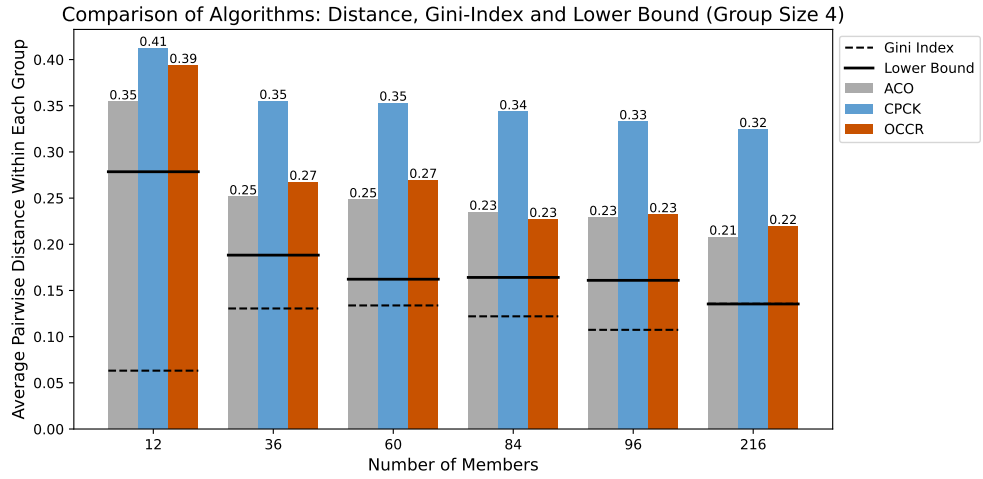
Regarding runtime, ACO is generally fast and efficient. When the number of members reaches 216, a noticeable increase in runtime is observed, particularly evident in Figure 4.2b. The increase is due to the step by step probability calculations for adding members to groups as described in Section 3.1.2. However, as shown in Figure 4.2c, the runtime of the ACO algorithm does not increase at the same rate for larger group sizes as the other algorithms. This is because, for example, the OCCR algorithm has to compute all combinations for a possible group, whereas the ACO only calculates the distance to each other members in each step. In summary, the ACO approach provides an efficient and effective optimization for the grouping problem.



(a) Group size 2



(b) Group size 3



(c) Group size 4

Figure 4.7.: Comparison of average time vs. average pairwise distance for different group sizes

4.2.4. Lower Bound and Gini-Index

The Gini-Index and the lower bound of a Group Assembly were considered to make general statements about the quality of the solutions. Since we do not have a ground truth for the minimum of the objective function, the lower bound provides us with a benchmark for how close the solutions are to an absolute minimum, even though this might not be achievable with very high probability. In Figures 4.7a, 4.7b and 4.7c, both ACO and OCCR produce distance values consistently close to the lower bound, regardless of the group size. This is a positive indicator, as it suggests that the solutions found are near-optimal. In contrast, the CPCK algorithm is far from the lower bound, especially with a larger number of members, as shown in Figure 4.7a, 4.7b and 4.7c.

The Gini-Index provides insights into whether the groups are balanced. In Figure 4.7a, it is shown that with a group size of two members, a relatively high value is reached. This indicates less balanced groups. However, it is difficult to draw a clear conclusion, as more groups are formed with smaller group sizes and a constant number of members. This leads to more values being compared than when the groups are larger. This is further supported by Figure 4.7b and 4.7c, as the Gini-Index decreases with increasing group size.

4.3. Discussion

The first limitation discussed in Section 2.5 is that two independent sets of must-linked members cannot be merged. This limitation arises from the way our algorithms are designed. However, if a larger number of members already have connections with others they wish to group with, this feature could become particularly useful. Consider a scenario where every member has exactly one must-link relationship, and the goal is to form groups of four members. One approach we propose to address this is the use of the mean, as discussed in Subsection 3.2.2. By calculating the mean for each set of must-linked members (also referred to as neighborhoods), we can treat the resulting “mean member” as a single entity. This respects the structural requirements of our algorithms. However, an additional mechanism is needed to track the number of members represented by each “mean member” to ensure equal sized groups.

An alternative and more nuanced approach to handling cases where the number of members is not divisible by the desired group size could be explored. In our current work, described in Section 2.5, this issue is addressed by adding artificial members, which inherently results in some groups having fewer members than the required group

size. However, one can envision scenarios where the opposite is desired — creating some groups with more members than the required group size. This approach could be particularly useful in situations where the number of groups, l, must remain fixed (e.g., when there is a predetermined number of available projects). Implementing such a solution would require substantial modifications to the current algorithms, as they are designed to accommodate only fixed group sizes.

For some Group Assemblies it might be desirable to form evenly balanced groups with regard to their intra-group distances. The objective functions of the algorithms could be modified in such a way, that they would not only minimize the total distance of a solution, but also the Gini-Index. This would lead to fairer groups.

For CPCK, it would be useful to explore alternative clustering methods on the remaining data. Other clustering approaches might lead to more representative clusters, potentially improving overall performance.

For larger sets of members, we observed that OCCR experiences increasingly high runtimes. In such cases, we could experiment with creating a representative sample of the members, which could then be used to form the initial grouping. This approach might help manage and potentially limit the runtime.

For practical applications of our thesis, we recommend presenting multiple solutions to the Group Manager. This is because the best solution may not always yield the fairest groups, as measured by the Gini-Index. Providing multiple options would allow the Group Manager to select the solution that best fits their specific purpose.

5. Conclusion

The research presented in this thesis aimed to find feasible solutions to the grouping problem. To achieve this goal, precise definitions of the grouping problem were elaborated. A framework was created to measure various distances between individual members, facilitating comparison. Three different algorithms – Ant Colony Optimization, Custom PCKMean, and Occurrence Ranking – were adapted, customized, and invented. An evaluation system was built to assess runtime, the average total distance of a solution, lower bounds, and the Gini-Index. Finally, the algorithms were evaluated, and the results were presented.

The evaluation showed that both ACO and OCCR effectively minimize the objective function. Furthermore, as the group sizes increases, the runtime of ACO increases at a slower rate than that of OCCR, while the CPCK approach consistently falls shorter of optimal performance for the grouping problem.

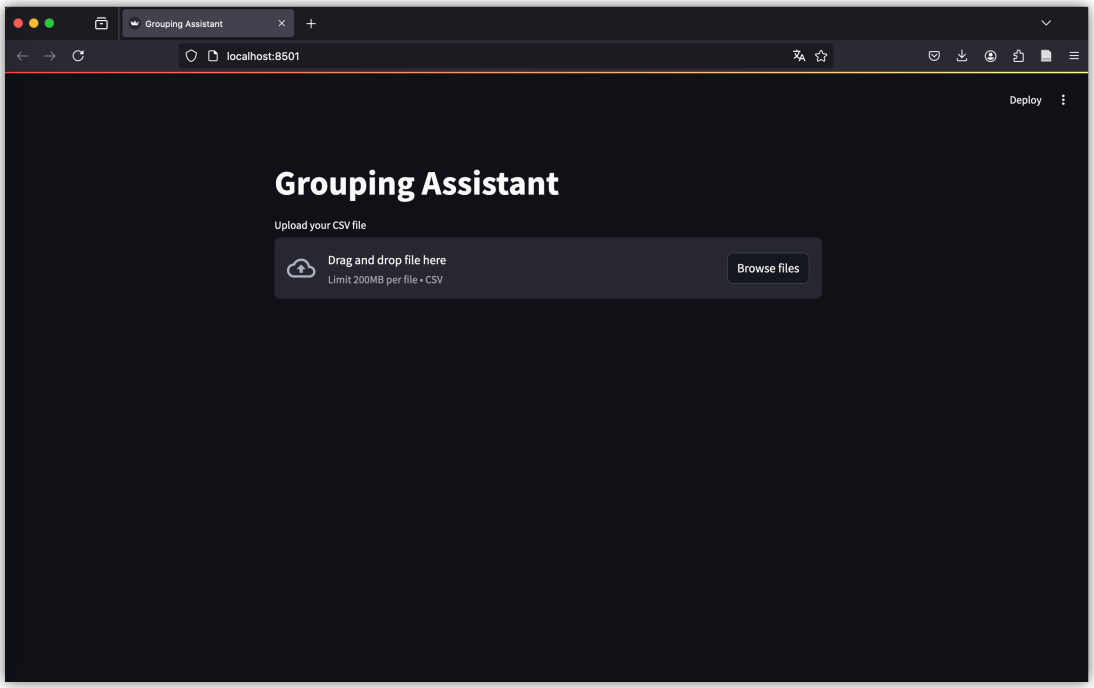
For future work, additional question types will be explored. In particular, allowing members to respond to open-ended questions could provide deeper insights into their preferences. For instance, a member could describe their personal interest in a specific topic, and this text could then be analyzed by a language model to assess similarities and compatibility with responses from other members.

To further diversify the algorithmic approach, we propose exploring Genetic Algorithms as a potential method for group formation. Genetic Algorithms are inspired by the principles of natural selection and evolution, making them particularly well-suited for solving complex optimization problems. Adapting them to our grouping problem has the potential to produce highly effective results.

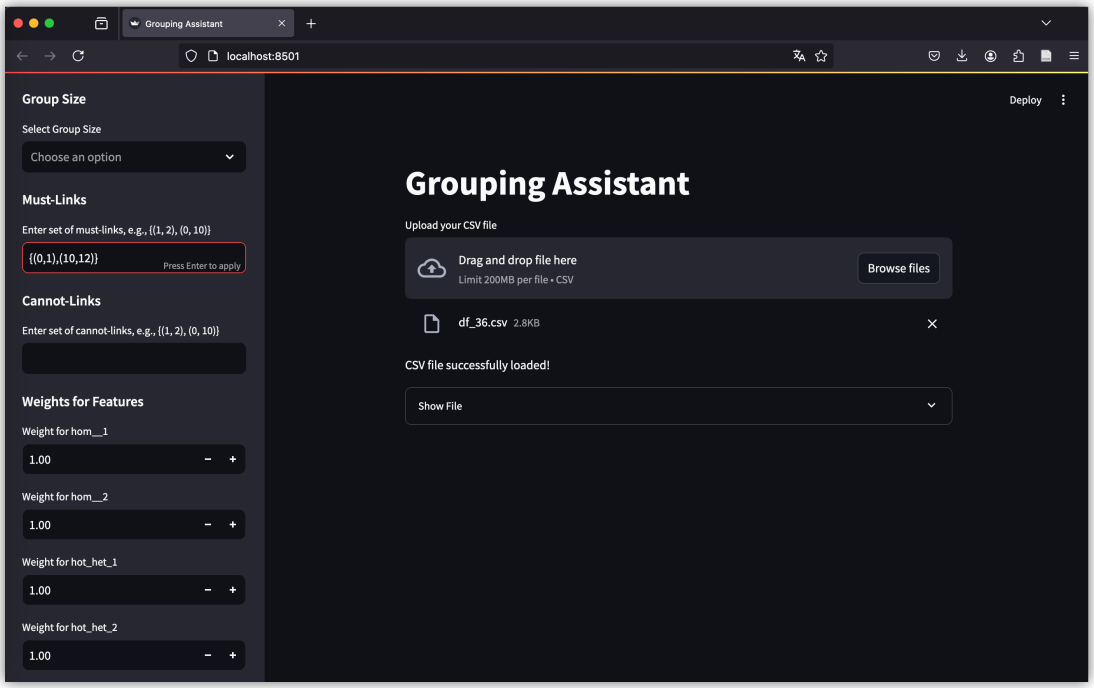
In our work, we focused on defining a unified feature encoding technique as well as on algorithmic approaches. Nevertheless, we conceptualized a minimal prototype for the Group Manager, where a CSV file can be uploaded and the grouping is carried out using the three algorithms described in this thesis. This prototype (cf. Appendix A) allows to configure parameters such as group size, must-links, cannot-links and weights for individual features. For future work, it would be worthwhile to develop a tool that enables the Group Manager to collect member information and seamlessly generate groups using

one or all of our algorithms. Additionally, providing members with the ability to manually swap groups after they have been formed could further refine the grouping process, leading to more flexible and personalized group formations.

A. Appendix

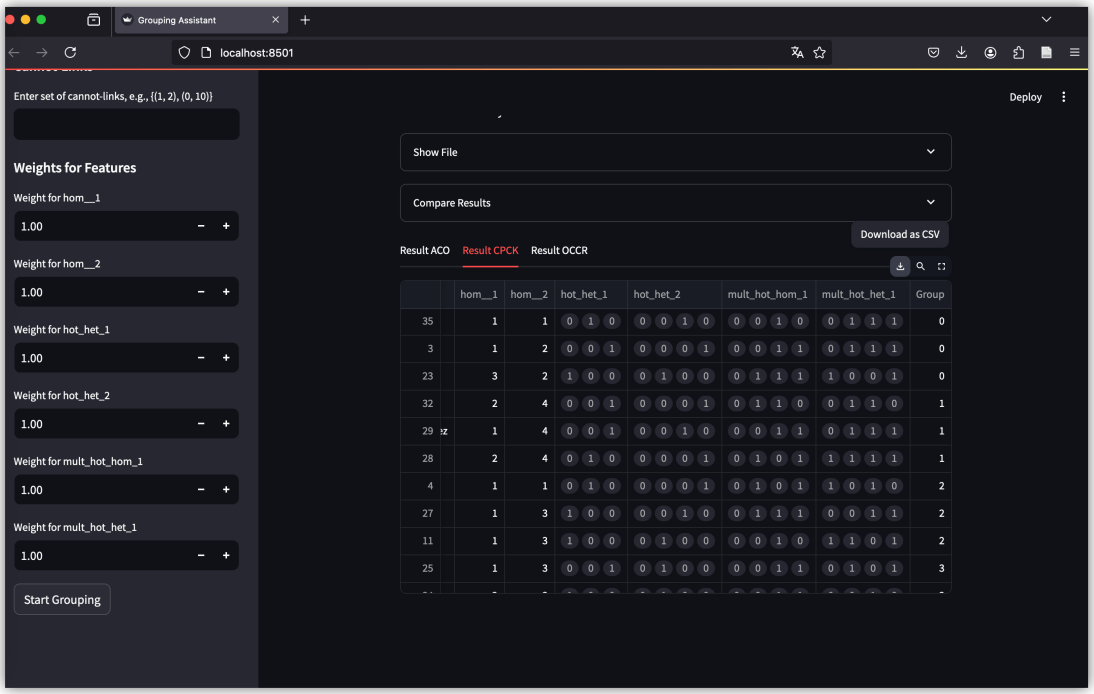


(a) Upload CSV file

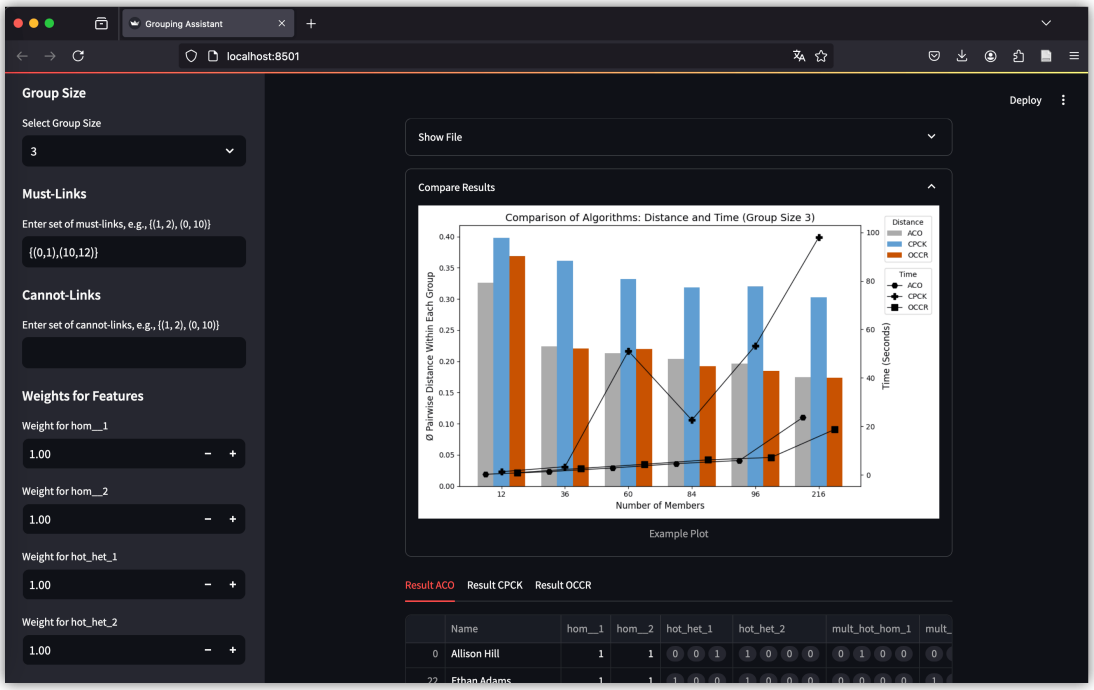


(b) Configurations

Figure A.1.: Minimal prototype for the Group Manager (i)



(a) Results



(b) Comparison results with example plot

Figure A.2.: Minimal prototype for the Group Manager (ii)

List of Figures

1.1. Distance of a solution 5

3.1. Trail laying 22
3.2. Trail following 22
3.3. Decision making 24
3.4. Representation of a solution (from [GB06]) 24
3.5. Solution convergence 26

4.1. Validation of the three algorithms versus randomly selected groups 42
4.2. Comparison of average time vs. average pairwise distance for different group sizes 43
4.3. Actual time for each iteration compared to the time limit set by the linear program for group size 3 44
4.4. Actual time versus pairwise distance for each iteration for group size 3 . . 45
4.5. OCCR's internal run times compared for group size 4 46
4.6. Impact of β on average distance for group size 3 47
4.7. Comparison of average time vs. average pairwise distance for different group sizes 49

A.1. Minimal prototype for the Group Manager (i) 55
A.2. Minimal prototype for the Group Manager (ii) 56

Bibliography

[AV06] Arthur, David ; Vassilvitskii, Sergei: k-means++: The Advantages of Careful Seeding / Stanford InfoLab. Version: June 2006. http://ilpubs.stanford.edu:8090/778/. Stanford, June 2006 (2006-13). – Technical Report

[BBD00] Bennett, K.P. ; Bradley, P.S. ; Demiriz, A.: Constrained K-Means Clustering / None. Version: May 2000. https://www.microsoft.com/en-us/research/publication/constrained-k-means-clustering/. 2000 (MSR-TR-2000-65). – Forschungsbericht. – 8 S.

[BBM04] Basu, Sugato ; Banerjee, Arindam ; Mooney, Raymond J.: Active Semi-Supervision for Pairwise Constrained Clustering. In: *SDM*, 2004

[BDC09] Borges, José ; Dias, Teresa G. ; Cunha, João Falcão E.: A new group-formation method for student projects. 34 (2009), 12, Nr. 6, 573–585. http://dx.doi.org/10.1080/03043790903202967. – DOI 10.1080/03043790903202967. – ISSN 0304–3797, 1469–5898

[BDW09] Basu, Sugato (Hrsg.) ; Davidson, Ian (Hrsg.) ; Wagstaff, Kiri L. (Hrsg.): *Constrained clustering: advances in algorithms, theory, and applications*. Boca Raton : CRC Press, 2009 (Chapman & Hall/CRC data mining and knowledge discovery series). – ISBN 9781584889960. – OCLC: ocn144226504

[CAT] CATME: *Team-Maker*. https://info.catme.org/features/team-maker/, . – [Accessed: 2025-01-15]

[CCT09] Choi, SHC ; Cha, Sung-Hyuk ; Tappert, Charles: A Survey of Binary Similarity and Distance Measures. In: *J. Syst. Cybern. Inf.* 8 (2009), 11

[DS] Dorigo, Marco ; Stützle, Thomas: *Ant colony optimization*. MIT Press. – ISBN 9780262042192

[Fee] Feedbackfruits: *Group Formation Algorithm Explainer*. https://help.feedbackfruits.com/en/articles/9556175-group-formation-algorithm-explainer, . – [Accessed: 2025-01-15]

[GB06] Graf, Sabine ; Bekele, Rahel: Forming Heterogeneous Groups for Intelligent Collaborative Learning Systems with Ant Colony Optimization, 2006. – ISBN 978–3–540–35159–7, S. 217–226

[GCT14] Ganganath, Nuwan ; Cheng, Chi-Tsun ; Tse, Chi K.: Data Clustering with Cluster Size Constraints Using a Modified K-Means Algorithm. In: *2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2014, S. 158–161

[Gow71] Gower, J. C.: A General Coefficient of Similarity and Some of Its Properties. In: *Biometrics* 27 (1971), Dezember, Nr. 4, 857. http://dx.doi.org/10.2307/2528823. – DOI 10.2307/2528823. – ISSN 0006341X

[HKP12] Han, Jiawei ; Kamber, Micheline ; Pei, Jian: *Data mining: concepts and techniques*. 3rd ed. Amsterdam Boston : Elsevier/Morgan Kaufmann, 2012 (Morgan Kaufmann series in data management systems). – ISBN 9780123814791

[IG04] Ingrid Gerdes, Rudolf K. Frank Klawonn K. Frank Klawonn: *Evolutionäre Algorithmen Genetische Algorithmen – Strategien und Optimierungsverfahren – Beispielanwendungen*. Wiesbaden Vieweg+Teubner Verlag 2004, 2004 (Computational Intelligence). – ISBN 9783322868398. – OCLC: 863941601

[JB15] Josef Bleymüller, Rafael W.: *Statistik für Wirtschaftswissenschaftler*. München, Deutschland : Verlag Franz Vahlen GmbH 2015, 2015. – 237 – 241 S. – ISBN 9783800649600

[LLOR10] Layton, Richard A. ; Loughry, Misty L. ; Ohland, Matthew W. ; Ricco, George D.: Design and Validation of a Web-Based System for Assigning Members to Teams Using Instructor-Specified Criteria. 2

(2010), Nr. 1. https://eric.ed.gov/?id=EJ1076132. – ERIC Number: EJ1076132

[McK] McKee, Amberle: *Swarm Intelligence Algorithms: Three Python Implementations*. https://www.datacamp.com/tutorial/swarm-intelligence, . – [Accessed: 2025-01-17]

[Mit11] Mitchell, Stuart: *PuLP: A Linear Programming Toolkit for Python*. https://github.com/coin-or/pulp, 2011. – Accessed: YYYY-MM-DD

[MOV12] Moreno, Julián ; Ovalle, Demetrio A. ; Vicari, Rosa M.: A genetic algorithm approach for group formation in collaborative learning considering multiple student characteristics. In: *Computers & Education* 58 (2012), Nr. 1, 560-569. http://dx.doi.org/https://doi.org/10.1016/j.compedu.2011.09.011. – DOI https://doi.org/10.1016/j.compedu.2011.09.011. – ISSN 0360–1315

[MZ20] Molchanov, Ilya ; Ziegel, Johanna: *Skript zur Vorlesung Kombinatorik und Wahrscheinlichkeitsrechnung*. Im IMSV-Sekretariat erhältlich : FS 2020, Vorlesung, Universität, 2020. – Überarbeitete Version eines Skriptes von Prof. H. Carnal, Dr. D. Assoulin und Dr. M. Collenberg

[OMB19] Odo, Chinasa ; Masthoff, Judith ; Beacham, Nigel: Group Formation for Collaborative Learning. In: Isotani, Seiji (Hrsg.) ; Millán, Eva (Hrsg.) ; Ogan, Amy (Hrsg.) ; Hastings, Peter (Hrsg.) ; McLaren, Bruce (Hrsg.) ; Luckin, Rose (Hrsg.): *Artificial Intelligence in Education*. Cham : Springer International Publishing, 2019. – ISBN 9783030232078, S. 206–212

[PTC+20] Peng, Chun-Cheng ; Tsai, Cheng-Jung ; Chang, Ting-Yi ; Yeh, Jen-Yuan ; Lee, Meng-Chu: Novel heterogeneous grouping method based on magic square. 517 (2020), 05, 340–360. http://dx.doi.org/10.1016/j.ins.2019.12.088. – DOI 10.1016/j.ins.2019.12.088. – ISSN 0020–0255

[RFQCMMS20] Ramos-Figueroa, Octavio ; Quiroz-Castellanos, Marcela ; Mezura-Montes, Efrén ; Schütze, Oliver: Metaheuristics to solve grouping problems: A review and a case study. 53 (2020), 03, 100643. http://dx.doi.org/10.1016/j.swevo.2019.100643. – DOI 10.1016/j.swevo.2019.100643. – ISSN 2210–6502

[ViC21] Victor-ikoh, Maudlyn ; Catherine, Ogunmodimu: Students' Group Formation Using K-Means Clustering in Combination with a Heterogeneous Grouping Algorithm. In: *American Journal of Engineering Research* 10 (2021), 11, S. 01–09

[WCRS01] Wagstaff, Kiri ; Cardie, Claire ; Rogers, Seth ; Schrödl, Stefan: Constrained K-means Clustering with Background Knowledge. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2001 (ICML '01). – ISBN 1558607781, S. 577–584

[WM97] Wilson, D. R. ; Martinez, T. R.: Improved Heterogeneous Distance Functions. In: *Journal of Artificial Intelligence Research* 6 (1997), Januar, 1-34. http://dx.doi.org/10.1613/jair.346. – DOI 10.1613/jair.346. – ISSN 1076–9757

[ZCAY+10] Zhamri, Che ; Che Ani, Zhamri ; Yasin, Azman ; Husin, Mohd Z. ; Hamid, Zauridah: A Method for Group Formation Using Genetic Algorithm. In: *International Journal on Computer Science and Engineering* 02 (2010), 01, S. 3060–3064

Disclaimer

In the interest of transparency, we acknowledge the use of large language model (LLM) technology, such as ChatGPT, to optimize text passages and assist in code generation for this thesis. While the research, analysis, and intellectual contributions are our own, we utilized LLM assistance to refine phrasing, enhance clarity, improve readability, and translate conceptual ideas into functional code. All key decisions regarding content, structure, argumentation, and methodology remain solely our responsibility.