A Hybrid Heuristic for the Maximum Dispersion Problem

Alex Gliesch[a], Marcus Ritt[a,*]

[a]*Federal University of Rio Grande do Sul, Av. Bento Gonçalves 8500, 91501-970 Porto Alegre, Rio Grande do Sul, Brazil*

Abstract

In this paper we propose a hybrid heuristic for the Maximum Dispersion Problem of finding a balanced partition of a set of objects such that the shortest intra-part distance is maximized. In contrast to clustering problems, dispersion problems aim for a large spread of objects in the same group. They arise in many practical applications such as waste collection and the formation of study groups. The heuristic alternates between finding a balanced solution, and increasing the dispersion. Balancing is achieved by a combination of a minimum cost flow algorithm to find promising pairs of parts and a branch-and-bound algorithm that searches for an optimal balance, and the dispersion is increased by a local search followed by an ejection chain method for escaping local minima. We also propose new upper bounds for the problem. In computational experiments we show that the heuristic is able to find solutions significantly faster than previous approaches. Solutions are close to optimal and in many cases provably optimal.

Keywords: Heuristics, Dispersion Problem, Ejection Chain, Branch-and-Bound Algorithm

*Corresponding author.

Email addresses: alex.gliesch@inf.ufrgs.br (Alex Gliesch), marcus.ritt@inf.ufrgs.br (Marcus Ritt)

1. Introduction

The Maximum Dispersion Problem (MaxDP) is to find a partition of a set of weighted objects such that the total weight of each part is close to a given target weight, and the *dispersion*, defined as the minimum distance between two objects in the same group, is maximal. Dispersion problems can be understood as the opposite of clustering problems, and arise in many practical situations. We give two example applications from the literature. The first is the design of waste collection territories in Germany in accordance with the European Waste Electrical and Electronic Equipment (WEEE) recycling directive [9]. This directive requires to assign waste collection stations to companies such that stations assigned to the same company are as dispersed as possible, in order to prevent regional monopolies. The problem extends the MaxDP by considering additional application-specific criteria, for example allowing the same station to be assigned to multiple companies. The main model for this application has been introduced by Fernández et al. [11], who also propose a GRASP algorithm to find heuristic solutions. Recently, Ríos-Mercado et al. [28] proposed a tabu search heuristic and Ríos-Mercado & Bard [29] an exact approach by mixed integer programming (MIP) to solve the same model.

The second application was considered first by Baker & Powell [2] and concerns the design of heterogeneous learning groups. Here, study groups comprised of students with backgrounds as different as possible are desired, to promote intercultural and interdisciplinary exchanges. The distances between students are given by the weighted differences between binary attribute vectors associated with each student. Thus, the problem can also be modeled as finding balanced groups of maximum dispersion.

The MaxDP was introduced by Fernández et al. [10] as a generalization of the two applications above. They propose an exact solution method based on mixed integer programming. It uses the fact that an instance of the MaxDP with n objects has at most $\binom{n}{2}$ solution values and iteratively solves a series of models with fixed upper and lower bounds on the solution value. The exact method is reported to solve instances of sizes up to 700 objects for instances of the WEEE application, and 300 objects for study group instances. Fernández et al. [10] also show that the MaxDP is NP-hard, by reduction from the Partition Problem. Moeini et al. [25] have proposed a variable neighborhood search (VNS) to heuristically find solutions for the MaxDP. However, the authors consider only instances of the WEEE application of relatively small size (up to 500 objects), which can be solved exactly by the method of Fernández et al. [10].

Upper bounds on the MaxDP can be obtained by relaxing the constraints on the total group weight. The resulting *unrestricted MaxDP* (UMaxDP) problem can

be solved by reduction to a series of graph coloring subproblems. Fernández et al. [10] propose an upper bounding scheme which considers the optimal solution of a set of smaller, heuristically generated subinstances. Ríos-Mercado & Bard [29] consider the design of recycling districts based on MIP models and also use upper bounds to UMaxDP in order to improve the models.

There are a number of related grouping problems in the literature. In contrast to the MaxDP, which asks for dispersed groups, districting problems (cf. Kalcsics [17] for a good overview) generally ask for units to be assigned to contiguous groups which meet weighing criteria and minimize a distance function, so as to produce compact districts. The Capacitated Clustering Problem [26] seeks groups of a given maximum overall weight and minimizing the sum of distances from the median object of each group. In terms of dispersion-based objectives, the Maximally Diverse Grouping Problem, first studied by Arani & Lotfi [1], asks that objects be partitioned into groups meeting balancing criteria and that maximize the sum of intra-group distances. Brimberg et al. [3] and Lai & Hao [19] have proposed heuristics based on incremental neighborhoods to solve it. In the context of location theory, the p-Dispersion Problem [8] asks for facilities to be located such that the minimum distance between two facilities is maximized. Some problem variants consider equity measures such as the minimum differential dispersion among selected facilities among a set of candidates [27, 7].

In this paper we propose a heuristic to solve the MaxDP. Starting from a greedy initial solution, it iteratively alternates between improving the dispersion and balancing the solution. The heuristic stops when the dispersion cannot be improved, or optimality can be shown, by a matching upper bound.

The main contributions of this paper are: i) a hybrid method combining a local search to improve dispersion with an effective ejection chain algorithm to resolve conflicts, which operates in the space of imbalanced solutions, ii) a matching balancing algorithm, that repeatedly selects two promising groups based on information computed by solving a minimum cost flow problem and applies a truncated branch-and-bound that searches for the best strategy to balance them, iii) a proof that two existing upper bounds from the literature are the same, and a new family of better upper bounds, together with algorithms to compute them efficiently, iv) a new set of large, challenging instances, and v) a detailed computational analysis that shows that the overall heuristic can solve more instances than previous methods in one to two orders of magnitude less time, and expands the limit of solvable MaxDP instances from 800 to about 4000 objects. Although the method is heuristic, combined with the new upper bound many solutions are provably optimal; in particular, we solve more instances to optimality than previous approaches.

The rest of this paper is organized as follows. In the next section we give a formal definition of the problem. In Section 3 we review existing upper bounds and

propose an improved upper bound. The proposed heuristic algorithm is outlined in Section 4, and Sections 5 and 6 explain in detail the algorithms used to improve the dispersion and to balance solutions. In Section 7 we report on experiments that assess the effectiveness of the proposed heuristic, as well as each of its components. We conclude in Section 8.

2. Problem definition

In the following we use the set notation $[n] = \{1, \dots, n\}$. An instance $I = (V, m, d, a, \alpha)$ of the MaxDP is defined by a set of objects V of size $n = |V|$, the desired number of groups m, a matrix $d \in \mathbb{R}_+^{n \times n}$ of distances the objects, a vector $a \in \mathbb{R}_+^n$ of object weights, and a balancing tolerance parameter $\alpha \in [0, 1]$. Let $R = \{d_{ij} \mid i, j \in V\}$ be the set of unique distance values, and $d^1 < \dots < d^{|R|}$ denote these distances in increasing order. For convenience, we also introduce a value $d^0 < d^1$.

A solution is a function $S : V \to [m]$ mapping objects to groups. We allow S to be partial. In a partial solution the objects in $V \setminus \mathrm{dom}(S)$ are *unassigned* or *free*. A solution S is *complete* if all objects are assigned, i.e., $\mathrm{dom}(S) = V$. We write $S_k = S^{-1}(k)$ for the set of objects assigned to group k in solution S.

The dispersion of group k is defined as $D(S_k) = \min_{i,j \in S_k} d_{ij}$, and the dispersion of solution S as $D(S) = \min_{k \in [m]} D(S_k)$. Given some dispersion value d we call a pair of objects $\{i, j\}$ a *d-pair* in S if $d_{ij} = d$ and $S(i) = S(j)$, and a $D(S)$-pair a *critical* pair. Objects i and j are *conflicting* in S if $d_{ij} < D(S)$. Note that, by definition of D, conflicting objects cannot belong to the same group. Let $C(S)$ be the set of critical pairs in S. During optimization we often use a extended objective function $D'(S) = D(S) - \epsilon|C(S)|$, where $\epsilon \ll \min_{i,j \in V} d_{ij}$ is a small constant, to break ties and order solutions by the number of moves necessary to increase their dispersion. This is especially important for instances of type "study" (see Section 7.1) where object distances are integer and $|R| \ll \binom{n}{2}$, making $|C(S)|$ generally large.

Let $w(C) = \sum_{i \in C} a_i$ be the total weight of the set of objects C, and M_k be a given target weight for group k. We assume that $\sum_{k \in [m]} M_k = \sum_{i \in V} a_i$. The *imbalance* of group k is the excess of the relative deviation of its weight from the target over α, namely $B(S_k) = \max\{0, -\alpha + |w(S_k) - M_k|/M_k\}$, and the imbalance of a solution S is the total imbalance $B(S) = \sum_{k \in [m]} B(S_k)$. The imbalance B represents the constraint violations of the classical balancing constraints given in the problem definition [10]. We say that group k is *balanced* if $B(S_k) = 0$, otherwise it is *imbalanced*. A solution S is *balanced* or *feasible* if all its groups are balanced. The tolerance parameter α allows a small deviation from the target weights, since the underlying packing problems may not always be feasible.

The goal of MaxDP is to find a balanced solution of maximum dispersion D. We also define the *unrestricted* MaxDP (*UMaxDP*, for short), which simply asks for a solution of maximum dispersion without balancing constraints. The UMaxDP is NP-hard, by reduction from the Vertex k-Coloring Problem [10]. Note that, since it is a relaxation of the original problem, the optimal solution value u^*_{ur} of the UMaxDP is an upper bound on the optimal value of the MaxDP.

For completeness we next give a mathematical model for the MaxDP. Let $x_{ik} \in \{0, 1\}$ be a variable that indicates that object $i \in V$ is part of group $k \in [m]$. Then we can formulate

$$\textbf{maximize} \quad \min_{i,j \in V} \sum_{k \in [m]} d_{ij} x_{ik} x_{jk} \tag{1}$$

$$\textbf{subject to} \quad \sum_{k \in [m]} x_{ik} = 1, \qquad \forall i \in V, \tag{2}$$

$$M_k(1 - \alpha) \leq \sum_{i \in V} a_i x_{ik} \leq M_k(1 + \alpha) \qquad \forall k \in [m], \tag{3}$$

$$x_{ik} \in \{0, 1\} \qquad \forall i \in V, k \in [m]. \tag{4}$$

In this formulation, the objective function (1) maximizes the distance among any two object assigned to the same group. Note that the objective function is quadratic and contains a minimum; both elements can be linearized by standard techniques. Constraints (2) guarantee that each object is assigned to exactly one group, and constraints (3) make sure that solutions are balanced.

Finally, we define two operators over solutions. A *shift* $i \to k$ reassigns object i to group $k \in [m] \setminus \{S(i)\}$. Similarly, a *swap* $i \leftrightarrow j$ swaps the assignments of objects i and j with $S(i) \neq S(j)$, i.e., i will be shifted to group $S(j)$ and j to $S(i)$. We use the notation $S[i \to k]$ and $S[i \leftrightarrow j]$ to refer to solution S after applying the respective operation, and denote the sets of all possible operations for solution S as $N_{shift}(S)$ and $N_{swap}(S)$, for shifts and swaps, respectively.

In the rest of the paper, when solution S is clear from the context we will omit S in unary functions such as B or D. Further, to make all procedures completely defined, unless otherwise stated we break ties by giving preference to a smaller object index i followed by a smaller group index k, when applicable.

3. Upper bounds

In this section we introduce new upper bounds for the UMaxDP. Since α is not relevant for the UMaxDP, we consider instances $I = (V, m, d)$ and write $D(I) = \max_{S \in S(I)} D(S)$ for the optimal solution value over all complete solutions $S(I)$

of I. We call $I' = (V', m, d)$ a *k-subinstance* if $V' \subseteq V$ has size $k = |V'|$. Let I^k be the set of all k-subinstances of instance I. Note that the optimal value of a k-subinstance I' of I is an upper bound on the optimal value of I, since any solution of I with given dispersion d when restricted to I' has dispersion at least d.

Fernández et al. [10] have proposed an upper bound $\mathrm{U}^C = \min_{I' \in I^{m+1}} D(I')$, which is the least upper bound obtained by solving all $(m+1)$-subinstances of an instance I. An $(m+1)$-subinstance can be solved optimally in time $O(m^2)$ by computing the maximum distance between two objects and putting them in the same group, with all remaining $m-1$ items in a group by themselves. Since it is impractical to consider all $\binom{|V|}{m+1}$ subinstances, the authors propose a relaxed upper bound U_h^C which considers only a sample of all subinstances. Samples are generated heuristically by a greedy algorithm and improved by a local search.

Ríos-Mercado & Bard [29] show that $(m+2)$-subinstances can also be solved in time $O(m^2)$, and consequently extend the above idea to consider all $(m+2)$-subinstances. They define an upper bound $\mathrm{U}^{RB} = \min_{I' \in I^{m+2}} D(I')$ as the least upper bound over all $(m+2)$-subinstances, and an efficiently computable heuristic version U_h^{RB}, which samples $(m+2)$-subinstances generated by a greedy algorithm. Empirically U_h^{RB} outperforms U_h^C on the tested instances. We refer the reader to the original papers for details on how U_h^C and U_h^{RB} are computed.

We generalize this idea to a family of decreasing upper bounds $\mathrm{U}^1 \geq \mathrm{U}^2 \geq \mathrm{U}^3 \geq \cdots \geq \mathrm{U}^{|V|-m}$, where U^σ is the least upper bound over all $(m+\sigma)$-subinstances. Observe that $\mathrm{U}^1 = \mathrm{U}^C$ and $\mathrm{U}^2 = \mathrm{U}^{RB}$, and $\mathrm{U}^{|V|-m} = u^*_{ur}$ is the optimal solution of UMaxDP, since it contains all vertices.

In Section 3.1 we first explain the relation between UMaxDP and graph coloring and show that $\mathrm{U}^1 = \mathrm{U}^2$, and then propose in Section 3.2 a heuristic to generate $(m+\sigma)$-subinstances for a given parameter σ, which can be solved by exact graph coloring algorithms in order to obtain better upper bounds.

3.1. The unrestricted MaxDP and graph coloring

The UMaxDP has a solution S with dispersion $D(S) > d$ iff the intersection graph $G_d = (V, E_d(V))$, where $E_d(V) = \{\{u, v\} \mid u, v \in V, d_{uv} \leq d\}$, is m-colorable. Such a solution can be obtained by computing a minimum vertex coloring for G_d. If there are at most m colors, each color can form a group, and by definition of the intersection graph the distance between vertices in the same group cannot be d or less, so $D(S) > d$ follows. Thus, the UMaxDP can be solved by finding the largest d^i, $i \in [|R|]$ for which $G_{d^{i-1}}$ is m-colorable.

Bounds can also be formulated by considering cliques. If the intersection graph G_d contains an $(m+1)$-clique it is not m-colorable, and thus d is an upper bound on the dispersion. Thus we obtain an upper bound by finding the smallest d^i such that contains an $(m+1)$-clique. Fernández et al. [10] show that upper bound U^C

is equivalent to the smallest d^i such that $\omega(G_{d^i}) \geq m+1$, where $\omega(G)$ is the size of the largest clique in G.

Since the size of any clique of a graph G is a lower bound on the size of any coloring of G, the size of its largest clique, $\omega(G)$, is a lower bound on the size of the smallest coloring, its *chromatic number* $\chi(G)$. Note that if $G_{u^*_{ur}}$ has no chromatic gap (i.e., $\omega(G_{u^*_{ur}}) = \chi(G_{u^*_{ur}}) = m+1$) then $\mathrm{U}^1 = \mathrm{U}^C = u^*_{ur}$ is optimal. Otherwise, $G_{u^*_{ur}}$ has no $(m+1)$-clique and $u^*_{ur} < \mathrm{U}^1$. Graphs with a chromatic gap tend to be harder to color by exact coloring algorithms, since the lower bound given by the clique number is weak [5]. In general, U^σ (and thus all U^k with $k \geq \sigma$) can still be optimal for larger chromatic gaps of $G_{u^*_{ur}}$. However, this does not hold for U^2, as the following theorem shows.

Theorem 3.1. *Upper bounds U^1 and U^2 are the same, i.e., $U^1 = U^2$.*

Thus, $\mathrm{U}^2 = \mathrm{U}^{RB}$ can never improve over $\mathrm{U}^1 = \mathrm{U}^C$. Clearly, the sampling-based heuristic versions U^C_h and U^{RB}_h still can produce different results, if they fail to find the right subinstances that witness these upper bounds.

For the proof Theorem 3.1 we use the following result of Dirac [6]:

Theorem 3.2. *If $0 \leq n \leq k-1$, a k-chromatic graph either contains a complete $(k-n)$-graph as a subgraph or has at least $k+n+2$ vertices.*

Proof of Theorem 3.1. If $\mathrm{U}^2 < \mathrm{U}^1$, there must be an $(m+2)$-subinstance $I' = (V', m)$ that witnesses this, i.e., the intersection graph $W = G_{U^2}(V', E(V'))$ has $m+2$ vertices, and $\chi(W) = m+1$.

Now by Theorem 3.2 with $n = 0$ and $k = m+1$, W either contains a complete $m+1$-subgraph or has at least $m+3$ vertices. Since it has $m+2$ vertices it must contain a complete $(m+1)$-subgraph K. But then $\mathrm{U}^1 = \min_{I' \in I^{m+1}} D(S) \leq D(K) \leq \mathrm{U}^2$, since $K \in I^{m+1}$, contradicting our assumption. □

3.2. An improved upper bound

Given a parameter $\sigma > 2$ and instance $I = (V, m, d)$, we search for an improved upper bound U^σ_h by searching for an $(m+\sigma)$-subinstance of I whose graph induced by the current upper bound is not $(m+1)$-colorable. If the search is successful, the current upper bound can be reduced, since the graph induced by u^*_{ur} must be $(m+1)$-colorable. The choice of σ must be small enough such that the coloring subproblems can be solved optimally within reasonable time, but at least as large as the smallest subgraph of $G_{u^*_{ur}}$ of chromatic number $m+1$. We calibrate parameter σ in Section 7.2.1.

Starting from an initial upper bound $d^u = \mathrm{U}^C_h$ algorithm UB iteratively generates, for each $i \in V$, an $(m+\sigma)$-subinstance $I_i = (V_i, m, d) \subseteq I$ using a

constructive heuristic seeded by object i, followed by a local search procedure, as explained below. For each I_i it then computes an exact coloring of the induced subgraph $H_u^i = (V_i, E_{d^u}(V_i))$. Here, we use the implementation of Lewis et al. [21] of the exact coloring algorithm of Korman [18]. If H_u^i is not m-colorable, then d^u is not optimal, and can be reduced. Algorithm UB then binary searches for the largest $1 \leq u' < u$ such that $\chi(H_{u'}^i) \leq m$, and updates u with this value. At most $O(n + \log_2 |R|)$ exact colorings are computed. It processes the graphs H_u^i in order of decreasing degree of node i and stops if an improving subinstance is found or if a time limit is reached. In the end, UB returns the final upper bound $U_\sigma^{GR} = d^u$. The full method is given in Algorithm UB.

Input : an instance $I = (V, m, d)$ and a subinstance size $\sigma > 0$.
Output : an improved upper bound $\mathrm{U}_h^\sigma \leq \mathrm{U}_h^C$.
$d^u \leftarrow \mathrm{U}_h^C(I)$
let $V = \{v_1, \ldots, v_n\}$ such that $\delta(v_1) \geq \delta(v_2) \geq \cdots \geq \delta(v_n)$
for $i \in [n]$ **do**
    $V_i \leftarrow$ generateSubinstance(I, σ, d^u, v_i)
    $H_u^i \leftarrow (V_i, E_{d^u}(V_i))$
    if $\chi(H_u^i) > m$ **then**
        $u \leftarrow$ binarySearch$(V_i, 1, u-1)$
        break
return d^u

Procedure UB(I, σ)

Each set V_i is constructed by starting with $V_i = \{i\}$ and iteratively adding the next object $j \in V \setminus V_i$ such that the number of edges $|E_{d^u}(V_i)|$ is maximized, with ties broken by maximum degree, until $|V_i| = m + \sigma$. Then, algorithm UB performs a first-improvement local search that swaps objects in V_i with objects in $V \setminus V_i$, again aiming to maximize $|E_{d^u}(V_i)|$. It iterates over the candidates in a round-robin fashion, starting from the first object index. Here, we search for subgraphs with as many edges as possible, as opposed to other criteria, since $|E_{d^u}(V_i)|$ is fast to compute and empirically it has a large influence on the chromatic numbers. Smith-Miles et al. [30] have thoroughly studied the relevance of several other metrics to the difficulty of graph coloring instances, and found the density (and thus the number of edges, for a graph of fixed size) to be among the three most important. Algorithm generateSubinstance outlines this procedure.

Note that since algorithm UB only considers a sample of n subgraphs, U_h^σ may not be as low as U^σ; we show in Section 7.2.2, however, that with an appropriate choice of σ it always improves existing upper bounds in practice. By starting with U_h^C, we ensure U^σ is at most U_h^C, but any feasible upper bound can be used as an initial value for d^u. Here, we chose U_h^C rather than U_h^{RB} as it is faster to compute

Input : an instance $I = (V, m, d)$, a subinstance size $\sigma > 0$, the current upper bound index u, and a seed object $v_i \in V$. **Output :** a subset $V_i \subseteq V$ of size σ such that $v_i \in V_i$.

$V_i \leftarrow \{v_i\}$
do
  $j \leftarrow \arg\max\{|E_{d^u}(V_i \cup \{j\})| + \delta(j)/n, j \in V \setminus V_i\}$
  $V_i \leftarrow V_i \cup \{j\}$
while $|V_i| < \sigma$
do
  $V_i' \leftarrow V_i$
  for $j \in V_i$ *cyclically, in increasing order of index* **do**
    for $k \in V \setminus V_i$ *cyclically, in increasing order of index* **do**
      if $|E_{d^u}((V_i \setminus \{j\}) \cup \{k\})| > |E_{d^u}(V_i)|$ **then**
        $V_i \leftarrow (V_i \setminus \{j\}) \cup \{k\}$
while $V_i \neq V_i'$
return V_i

Procedure generateSubinstance(I, σ, d^u, v_i)

and empirically smaller.

4. A hybrid heuristic for the MaxDP

We propose a hybrid heuristic Hase to solve the MaxDP. Its overall structure follows the alternating approach of Gliesch & Ritt [13] for solving districting problems. Algorithm Hase summarizes the method. Hase alternates between balancing solutions and increasing the dispersion. This is repeated until a time limit is reached or a feasible solution with a dispersion equal to the upper bound is found.

Algorithm optBal tries to balance a solution. Since balancing a solution may reduce its dispersion arbitrarily, optBal is restricted to produce only solutions with a minimal acceptable dispersion d. In the first trial d is set to the dispersion of the current solution, and thus the dispersion is not allowed to decrease. However, if this fails, Hase uses a binary search to find the largest dispersion that can be successfully balanced. The search interval includes all dispersion values between the last solution that could be balanced, and the first solution that could not. In this case a regress in dispersion is allowed in order to find a feasible solution. After finding the largest feasible dispersion, algorithm Hase assumes that no larger dispersion is feasible and stops.

Otherwise, if optBal is able to balance the current solution of highest dispersion, algorithm optDisp tries to increase the dispersion. To achieve this, it is allowed to produce imbalanced solutions. If the dispersion cannot be improved, the heuristic terminates. Otherwise, the next iterations starts to balance the improved

$ub \leftarrow$ UB()
$S^0 \leftarrow$ infeasible solution; $D(S^0) \leftarrow d^0$
$i \leftarrow 1$
$S^1 \leftarrow$ greedyConstructive()
repeat
  $S^i \leftarrow$ optBal$(S^i, D(S^i))$
  if S^i *is imbalanced* **then**
    binary search for the largest $d \in (D(S^{i-1}), D(S^i)) \cap R$
     s.t. $B(\text{optBal}(S^i, d)) = 0$
    if *no such* d *exists* **then**
      if $i = 1$ **then**
        return optimal solution of model (F), or S^0 if none found
      return S^{i-1}
    return optBal(S^i, d)
  if $D(S^i) = ub$ *or time limit reached* **then**
    return S^i
  $i \leftarrow i + 1$
  $S^i \leftarrow$ optDisp(S^{i-1}, ub)
until $D'(S^i) = D'(S^{i-1})$
return S^i

Algorithm Hase

solution, if required.

Algorithm optDisp starts from a balanced solution and stops whenever the dispersion improves, which typically entails only a small change. This is done to avoid a large regress in the solution's imbalance, such that from the second iteration on optBal starts from a nearly balanced solution, and thus tends to be much faster. Further, by doing small, incremental improvements the algorithm avoids terminating with an imbalanced solution when reaching the time limit.

In the special case where optBal fails with $d = d^1$, in a final effort to find any feasible solution Hase tries to solve a feasibility MIP model (F) given by constraints (2), (3), and (4). If model (F) is infeasible, then Hase returns S^0 with a proof of infeasibility; otherwise, it returns solution x. If (F) cannot be solved within the remaining time limit, Hase also returns S^0, but without an infeasibility guarantee.

In the following subsection we explain the greedy construction of an initial, possibly imbalanced solution that will be balanced in the first iteration. Next, in Section 5.2 we explain algorithm optDisp, which tries to increase the dispersion by alternating between a local search and a tree-based ejection procedure, followed algorithm optBal in Section Section 6, which tries to balance a solution by a truncated branch-and-bound method.

4.1. Initial solutions

A greedy constructive algorithm generates initial solutions. It first seeds the m groups with m objects from an $(m+1)$-subinstance $I_{\mathrm{U}_h^C}$ witnessing U_h^C (i.e., whose maximum pairwise distance is equal to $I_{\mathrm{U}_h^C}$) generated using the method of Fernández et al. [10], excluding the object of lowest index that is part of the maximum pairwise distance. This is done to select initial seeds which are as close to each other as possible. No additional step is required to find these seeds, as $I_{\mathrm{U}_h^C}$ is obtained when computing the upper bound in Algorithm UB.

The algorithm then iteratively selects the group k with the smallest fraction of achieved target weight $w(S_k)/M_k$ and assigns to it a free object i which maximizes $D'(S[i{\to}k])$, with ties broken by minimum imbalance. This choice of the next assignment gives preference to initial solutions of high dispersion, as opposed to more balanced ones. Still, some initial balancing is ensured by preferring assignments with minimum imbalance. The algorithm stops when all objects have been assigned. Because both D' and B are recomputed in constant time upon object insertions, the solution is constructed from the seeds in time $O(n^2)$. Algorithm greedyConstructive outlines the method.

Output : an initial solution.
let $(v_1, \ldots, v_{m+1}, k, d)$ be an $(m+1)$-subinstance witnessing U_h^C,
  s.t. $\exists j : v_j > v_{m+1} \wedge d_{v_j v_{m+1}} \geq d_{ik}\ \forall i, k \in [m]$
$S_k \leftarrow \{v_k\}\ \forall k \in [m]$, inducing solution S
do
    $k \leftarrow \arg\min_{k \in [m]} w(S_k)/M_k$
    $i \leftarrow \arg\max_{i \in [n] \setminus \bigcup_{l \in [m]} S_l} D'(S[i{\to}k]) - \epsilon B(S[i{\to}k])$
    $S_k \leftarrow S_k \cup \{i\}$
while $\mathrm{dom}(S) \neq V$
return S

Algorithm greedyConstructive

5. Improving dispersion

Algorithm optDisp improves the dispersion of a solution by alternating between a local search (LS) and a recursive exchange procedure (EX) based on a tree search. LS repeatedly performs shift and swap operations, and stops if it cannot further improve the incumbent (see Section 5.1). Then, EX attempts to find a longer sequence of moves to escape the current local minimum (see Section 5.2). Algorithm optDisp alternates between LS and EX until either the current extended dispersion D' cannot be improved, dispersion D has improved, the upper bound is achieved or a time limit is reached. At the end, if D has improved optDisp runs

LS once more. Note that since optDisp ignores balancing constraints, it may produce and work with infeasible solutions. Because it requires D' to improve at each iteration and stops once D is improved, optDisp iterates at most $|C(S)| = O(\binom{n}{2})$ times. Algorithm optDisp outlines the method.

Input : a solution S^0 and an upper bound ub.
Output : a solution S such that $D'(S) \geq D'(S^0)$.
$S \leftarrow S^0$
do
    $S' \leftarrow S$
    $S \leftarrow$ LS(S)
    $S \leftarrow$ EX(S)
while $D'(S) \neq D'(S') \wedge D(S^0) = D(S) < ub \wedge$ *in time limit*
if $D(S^0) < D(S) \wedge$ *in time limit* **then**
    $S \leftarrow LS(S)$
return S

Algorithm optDisp(S^0, ub)

5.1. Local search

Algorithm LS searches the neighborhoods N_{shift} and N_{swap} iteratively performing a shift or a swap which increases the dispersion D' until no more improving moves exist. Since $|N_{swap}(S)| \gg |N_{shift}(S)|$, LS first explores N_{shift} fully and considers swaps only if there is no improving shift. It uses a first-improvement strategy and explores each neighborhood in a round-robin fashion, i.e., each call to LS continues the search cyclically from the point where the previous call stopped. Note that only operations on objects in critical pairs can improve $D'(S)$, so LS considers the reduced neighborhoods $\{i \rightarrow k \mid i \in \bigcup C(S), k \in [m] \setminus \{S(i)\}\}$ for shifts and $\{i \leftrightarrow j \mid i \in \bigcup C(S), j \in [n] \setminus S(i)\}$ for swaps. These neighborhoods have sizes $O(|C|m)$ and $O(|C|n)$, respectively. Algorithm LS outlines the method.

The number of critical pairs $|C|$ varies according to the instance and can potentially be as large as $\binom{n}{2}$, if all objects are in the same location. In practice, however, instances of the WEEE application typically have $|C| < 10$, while instances of study group design have more duplicate distances and average $|C| \approx n/m$. Using a dynamic data structure to update D' in time $O(n^2 \log n)$ and evaluate D' for candidate moves in time $O(m)$, each iteration takes $O(n^2(m + \log n))$ time, since the swap neighborhood dominates.

5.2. An ejection chain algorithm for improving dispersion

Algorithm EX searches for an improving ejection chain [15]. The idea is to eject some object $i \in \bigcup C(S)$ from its current group (i.e., free that object), obtaining a partial solution with better $D'(S)$, and then to insert i into some other group

Input : a solution S^0. **Output :** a solution S such that $D'(S) \geq D'(S^0)$.
$S \leftarrow S^0$
do
  $S' \leftarrow \arg\max\{D'(S[i \to k]) \mid i \in \bigcup C(S), k \in [m] \setminus \{S(i)\}\}$
  if $D'(S') > D'(S)$ **then**
    $S \leftarrow S'$
    continue
  $S' \leftarrow \arg\max\{D'(S[i \leftrightarrow j]) \mid i \in \bigcup C(S), S(j) \neq S(i)\}$
  if $D'(S') > D'(S)$ **then**
    $S \leftarrow S'$
while $D'(S) = D'(S') \wedge$ *in time limit*
return S

Algorithm LS(S^0)

$k \neq S(i)$. This can lead to a sequence of accommodating moves. Insertions are recursively done as follows. Let d' be the dispersion $D(S)$ before the ejection of i and $\kappa(i, k, S) = \{j \mid j \in S_k, d_{ij} \leq d'\}$ be the set of conflicting objects induced by the placement of object i into some group k. When attempting to insert i into k, EX ejects every object $j \in \kappa(i, k, S)$ from k, and recursively tries to reinsert each object j into the solution using the same procedure. Note that $\kappa(i, k, S)$ is non-empty for all k at recursion depth 0, since we assume that the current solution is a local minimum of N_{shift}. If an object is moved to another group, it is *fixed*, and cannot be moved again in that recursion branch. Reinserting an object i is considered *successful* if i is not fixed and inserting all $j \in \kappa(i, k, S)$ is successful for some $k \in [m]$ (the basic case is $\kappa(i, k, S) = \emptyset$, i.e., there are no conflicts).

This process can also be described by an AND/OR tree. Here, OR nodes branch on the possible groups an ejected object can be moved into, and only one branch must succeed for the OR node to be considered feasible, whereas AND nodes branch on the conflicts induced by an insertion, and all branches (each, in turn, an OR node representing a conflict) must be successfully placed for the AND node to be feasible.

Algorithm EX considers the possible receiving groups $k \in [m]$ in order of non-decreasing number of conflicts $|\kappa(i, k, S)|$, i.e., it prefers groups which will eject the least number of objects, and thus likely require fewer recursive calls. We have found this to be significantly better than an arbitrary group ordering. EX discards groups k where $\kappa(i, k, S)$ contains a fixed object, since they cannot be successful. When reinserting the ejected objects, it first tries objects which are more likely to fail: since all objects in κ must be inserted, a single failure allows to fathom unsuccessful nodes early. To this end, EX keeps track of the number $\varphi(i, k)$ of unsuccessful placement attempts for each object i and group k and the

total number of failed insertions $\varphi(i) = \sum_{k \in [m]} \varphi(i,k)$, and attempts to reinsert ejected objects in decreasing order of $\varphi(i)$, with ties broken by i. The value of φ is maintained across multiple calls to EX.

Since at least one object is fixed at each node in the recursion tree, the recursion depth is bounded by n. However, this still leads to a search space that is too large to be explored in practical time. Therefore, inspired by the well-known heuristic of Lin & Kernighan [23] for the Traveling Salesman Problem, we introduce parameters p_1, p_2 and p_3 such that:

- at depth $d \leq p_1$ EX expands all possible insertions of the current object into other groups,
- at depth $p_1 < d \leq p_2$ EX only recurses on groups k with a single conflict ($|\kappa(i,k,S)| \leq 1$),
- at depth $p_2 < d \leq p_3$ EX adopts an aggressive backtracking approach and recurses only on the group k with smallest lexicographic $(\varphi(i,k),k)$ among the groups with a single conflict,
- at depth $d > p_3$, if there are still conflicts the node is pruned, i.e., the current branch is declared unsuccessful.

Algorithm EX shows the ejection chain approach. It processes the objects $i \in \bigcup C(S)$ in order of non-decreasing $\varphi(i)$, i.e., it prefers objects which have been easy to place, since only one successful insertion is needed. Each of these objects is ejected and then EX attempts to reinsert it, and stops as soon as a reinsertion is successful (since in this case $D'(S)$ must have improved; otherwise, there would still be unassigned conflicting nodes). Assuming groups have on average n/m objects, each recursive call takes time $O(n)$ at depth $d \leq p_2$, and $O(n/m)$ time at depth $p_2 < d \leq p_3$.

Figure 1 shows the EX procedure on an example with $m = 3$ and $(p_1, p_2, p_3) = (1, 2, 3)$. We represent OR nodes as squares and AND nodes as circles. Consider the critical pair set $C = \{\{u_1, u_2\}\}$ with $\{u_1, u_2\} \subseteq S_3$, and assume this is the first call to EX, i.e., $\varphi(u) = 0$ for all u. Since $\varphi(u_1) = \varphi(u_2)$, the tie is broken lexicographically and EX starts by trying to eject u_1. At depth $d = 1$ it can move u_1 to either S_1 or S_2. Since $|\kappa(u_1, S_2)| < |\kappa(u_1, S_1)|$, it explores the branch $u_1 \to S_2$ first. This branch induces the ejection of u_3, which can, in turn, be placed in S_1 or S_3. Since the search is now at depth $d = 2 > p_1$, subtree $u_3 \to S_3$ is pruned since $|\kappa(u_3, S_3)| > 1$. Next, branch $u_3 \to S_1$ induces the ejection of u_4, which can be placed in either S_2 or S_3. Because the search depth now is $3 > p_2$, however, EX prunes $u_4 \to S_3$ since it can only explore the one branch (since $\kappa(u_4, S_2) = \kappa(u_4, S_3) = 1$ and $\varphi(u_4, S_2) = \varphi(u_4, S_3) = 0$ the tie was broken by

Input : a complete solution S^0.
Output : a complete solution S with $D'(S) \geq D'(S^0)$ if successful, or S^0 on failure.

for $i \in \bigcup C(S^0)$ *in order of non-decreasing* $\varphi(i)$ **do**
  $S \leftarrow$ ejectionChain$(S^0, \emptyset, i, 0)$
  if $S \neq S^0$ **then**
    return S
return S^0

Function *ejectionChain(S^0, F, i, d)*
  $S \leftarrow S^0; F \leftarrow F \cup \{i\}$
  $first \leftarrow$ true
  for $k \in [m] \setminus \{S^0(i)\}$ *in order of increasing* $(|\kappa(i,k,S)|, \varphi(i,k), k)$ **do**
    if $\kappa(i,k,S) \cap F \neq \emptyset$ **then**
      continue
    if $(d > p_3 \wedge \kappa(i,k,S) \neq \emptyset)$ ***or*** $(d > p_2 \wedge \neg first)$ ***or*** $(d > p_1 \wedge |\kappa(i,k,S)| > 1)$ **then**
      break
    for $j \in \kappa(i,k,S)$ *in order of decreasing* $\varphi(j)$ **do**
      $S' \leftarrow$ ejectionChain$(S, F, j, d+1)$
      if $S' = S$ **then**
        continue next k-loop
      $S \leftarrow S'$
    return S
    $first \leftarrow$ false
  return S^0

Algorithm EX(S^0)

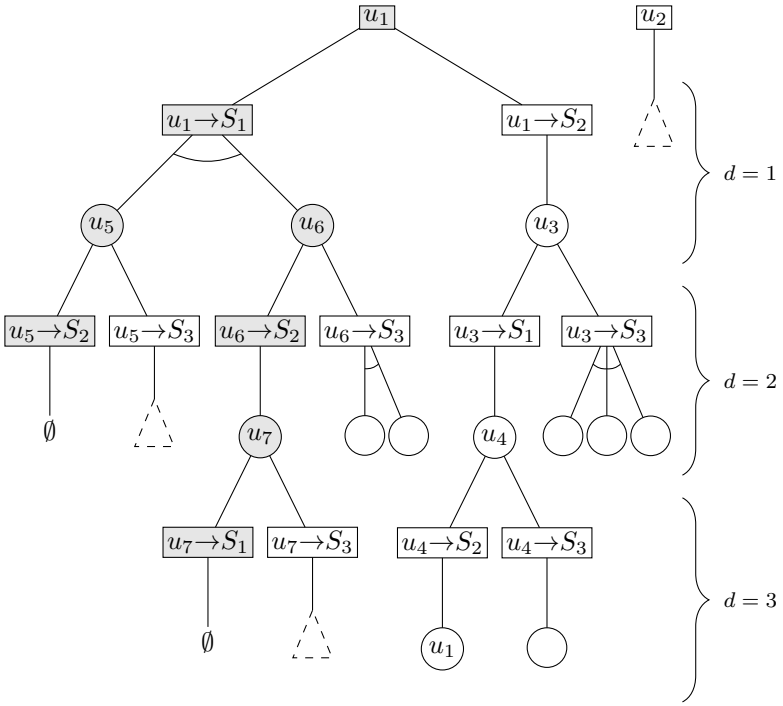

Figure 1: An example of the EX procedure.

group index). When visiting branch $u_4 \to S_2$ EX finds that u_1 generates a conflict. However, since u_1 is fixed and may not be ejected again, this branch is pruned and EX backtracks to $u_1 \to S_1$. Movement $u_1 \to S_1$ induces ejections u_5 and u_6. Because these are AND nodes, both u_5 and u_6 must be successfully placed in order to find a feasible sequence. Moving $u_5 \to S_2$ yields no conflicts and can be done immediately, regardless of subtree $u_5 \to S_3$. When reinserting u_6, EX prunes subtree $u_6 \to S_3$ since at depth $2 > p_1$ it has more than one child. Moving $u_6 \to S_2$ therefore causes u_7 to be ejected. Finally, u_7 is immediately placed in S_1 as $\kappa(u_7, 1, S) = \emptyset$. Since all AND nodes led to a leaf node with no conflicts, the set of movements $\{u_1 \to S_1, u_5 \to S_2, u_6 \to S_2, u_7 \to S_1\}$ is feasible and increases the dispersion.

6. Balancing solutions

A common approach to improve balance in similar problems is a variable neighborhood search (VNS) that first examines neighborhood N_{shift}, and if no improving shift can be found, neighborhood N_{swap} [3, 28]. If no improving move

is found, the search continues from a perturbed solution. In our experience VNS-like methods for the MaxDP quickly reach local minima, and the perturbation step is typically not enough to escape them. This is mainly because these two operators can fail to explore large sections of the search space through improving paths. Although shifts are universal, in the sense that any solution can be transformed into any other using them, often finer adjustments to group weights can only be achieved by long, not strictly improving sequences of shifts. Different operators such as 3-chain [3] or double (triple) shifts [4], or other, more elaborate ones often help mitigate the issue, but they tend to have neighborhoods too large to be fully explored at each iteration.

We therefore propose a more flexible approach to generate longer move sequences which includes by design the shift and swap neighborhoods, outlined in Algorithm optBal. It repeatedly selects two groups k and l and searches for a sequence of exchanges between them, which we denote as a *group pair exchange*, such that their relative imbalances are reduced, and the dispersion does not fall below some given limit d. If such a sequence can be found, it is applied, and the algorithm considers the next pair of groups. Group pair exchanges are obtained by a truncated branch-and-bound procedure bb (explained in Section 6.2) which aims at an optimal exchange of objects between two groups. Because the entire branch-and-bound tree must be exhausted if no improving exchange exists, which usually is too costly, this procedure expands at most BB_{max} nodes. Initially BB_{max} is set to $\mathrm{BB}^{\mathrm{lo}}_{max}$.

After each iteration, regardless of whether an improving exchange between groups k and l was found, the pair $\{k, l\}$ is marked as *tabu* (i.e., it can only be selected if no non-tabu moves exist) for τ iterations. We set $\tau = m$ following Lei et al. [20] and Gliesch et al. [14]. Further, we denote a pair of groups $\{k, l\}$ as *hopeless* if an improving exchange between them is impossible. This happens if k and l are both balanced, or if the last exchange between k and l did not find an improving sequence and no other improving sequences incident to either k or l have been found since. Algorithm optBal never selects hopeless pairs. In Algorithm optBal tabu and hopeless pairs are represented by the sets T and H, respectively. The algorithm stops if all pairs of groups are hopeless, meaning either the incumbent is balanced or exchange attempts for all imbalanced groups have failed. If the solution is still imbalanced, optBal doubles BB_{max} and re-executes starting from the current solution, up to a maximum $\mathrm{BB}_{max} = \mathrm{BB}^{hi}_{max}$, at which it stops. In our implementation, we use $\mathrm{BB}^{lo}_{max} = 2^{11}$ and $\mathrm{BB}^{hi}_{max} = 2^{18}$.

In early tests we found that the main algorithm sometimes quickly converges to solutions where balancing fails even for low dispersion bounds, causing an early termination of the algorithm, but that small modifications to the input solution are often enough to allow balancing, if the instance is feasible. To exploit this,

if balancing fails it performs a deterministic shuffling step to slightly modify the current solution, and re-executes the balancing algorithm. This is done at most ξ times, or until the resulting imbalance is larger than a threshold θ, above which another attempt at balancing is unlikely to succeed. In the end the solution with lowest imbalance B is returned.

To keep the all steps deterministic, shuffling is done as follows. The algorithm cyclically iterates over pairs of groups in lexicographical order, starting from $(1, 1)$ in the first call and later from the point where the previous execution stopped. Given pair (k, l) it considers shifts $u \to l, u \in S_k$ in the order which the nodes in S_k are stored in memory. (This order is mostly arbitrary, and is determined by the sequence of operations done so far on S_k.) It applies the first shift s encountered for which $D(S[s]) \geq d$, and moves on to the next pair of groups. The algorithm stops after ρ shifts. We calibrate parameters ξ, θ and ρ in Section 7.4.1.

Input : a solution S, and a minimum dispersion d.
Output : a solution S' such that $B(S') \leq B(S)$ and $D(S') \geq d$.
$S' = S$
for $i \in [\xi]$ **do**
  $S' \leftarrow$ successiveGroupExchanges$(S', d, \text{BB}^{lo}_{max})$
  $S \leftarrow \arg\min\{B(S), B(S')\}$
  if $B(S) = 0$ ***or*** $B(S') > \theta$ **then**
    break
  $S' \leftarrow$ deterministicShuffle(S', ρ)
return S
Function *successiveGroupExchanges(S^0, d, BB_{max})*
  $S \leftarrow S^0; T \leftarrow \emptyset; H \leftarrow \{\{k, l\} \mid k \text{ and } l \text{ are balanced in } S^0\}$
  repeat
    $k, l \leftarrow$ selectTwoGroups(S, T, H)
    $T \leftarrow$
     $(T \cup \{\{k, l\}\}) \setminus \{\text{pairs that have been tabu for more than } \tau \text{ iterations}\}$
    $S' \leftarrow \text{bb}(S, k, l, \text{BB}_{max})$
    if $B(S') < B(S)$ **then**
      $S \leftarrow S'$
      $H \leftarrow H \setminus \{\{i, j\} \mid \{i, j\} \cap \{k, l\} \neq \emptyset, i \text{ or } j \text{ is imbalanced in } S\}$
    if $B(S') = B(S)$ *or k and l are balanced in S* **then**
      $H \leftarrow H \cup \{\{k, l\}\}$
  until $B(S) = 0 \vee |H| = \binom{m}{2}$
  if $B(S) > 0$ *and* $BB_{max} \leq BB^{hi}_{max}$ **then**
    return successiveGroupExchanges$(S, d, 2\text{BB}_{max})$
  return S

Algorithm optBal(S, d)

6.1. Selecting two groups

At each iteration, optBal selects the next pair of groups by solving a minimum cost flow problem on an auxiliary bipartite graph. The main idea is to select the two groups that would exchange the most weight if all possible shifts were allowed, regardless of decreases in the dispersion.

Let each group $k \in [m]$ be represented by two nodes v_k^s and v_k^t in different parts of the graph, where v_k^s is a source node with supply $w(S_k)$ and v_k^t is a sink node with demand M_k. A pair of nodes v_k^s and v_l^t which is not hopeless is connected by an arc (v_k^s, v_l^t) of different capacity and cost:

- If $k = l$ the capacity is $(1 - \alpha)M_k$ and the cost 1. The low cost is an incentive for groups to maintain their lower target bound, and the capacity of $(1 - \alpha)M_k$ forces them to send the excess over the lower bound to other groups.
- If $k \neq l$ and $\{k, l\}$ is not tabu, the capacity is ∞ and the cost 2. This allows groups to send any excess weight to other groups at a slightly higher cost.
- If $k \neq l$ and $\{k, l\}$ is tabu, the capacity is ∞ and the cost is an upper bound $f_u = 2\sum_{i \in V} a_i$ on the maximum cost of all other edges. These arcs heavily penalize flow exchange on tabu pairs.

After finding the minimum cost flow in this graph, which can be done in time $O(f_u m^2 \log m)$ using the algorithm of Ford & Fulkerson [12], optBal selects the two non-tabu groups incident to the edge of highest flow, with ties broken lexicographically. If no flow passes through edges incident to non-tabu pairs, tabu pairs are also allowed to be selected.

Figure 2 illustrates four iterations of the main loop of successiveGroupExchanges on an example with $m = 3$. In this example we consider $M = (5, 5, 5)$ and $\alpha = 0.2$ (i.e., groups can assume weights between 4 and 6). For each iteration we show the flow graph with edge labels representing flow/capacity/cost, and node labels name/flow, with the name of the vertex and the flow passing through it. We omit edges which do not receive flow. Initially we have $w = (10, 2, 3)$, $T = H = \emptyset$, and the edge of highest flow is $\{v_1^s, v_2^t\}$ with flow 3 (here, the tie with $\{v_1^s, v_3^t\}$ has been broken lexicographically), so optBal selects pair $\{1, 2\}$ to exchange objects (Figure 2a). Suppose now that the branch-and-bound method fails to find an improving exchange between 1 and 2. In this case, $\{1, 2\}$ is hopeless and optBal recomputes the minimum cost maximum flow without edge $\{v_1^s, v_2^t\}$, obtaining a highest flow 5 on edge $\{v_1^s, v_3^t\}$ (Figure 2b). Procedure bb now finds an improving exchange between groups 1 and 3, with new weights $w = (7, 2, 6)$. Pair $\{1, 2\}$ is no longer hopeless, since an improving exchange incident to 1 was

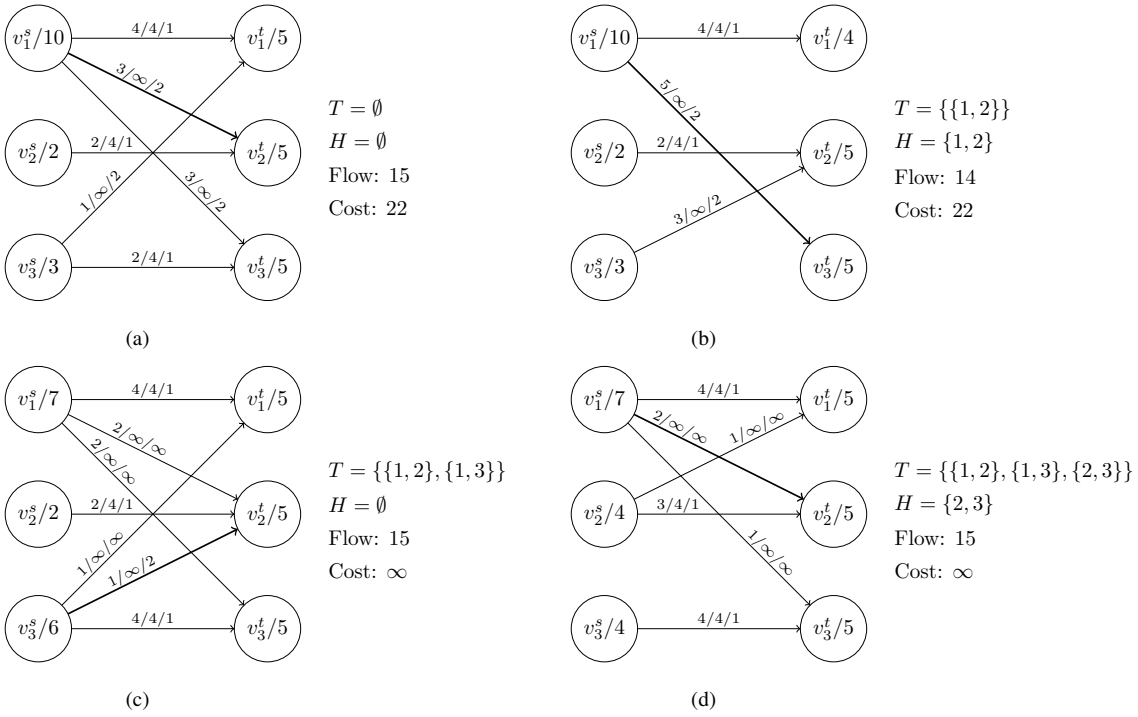

Figure 2: Four example iterations of the successiveGroupExchanges procedure of Algorithm optBal.

found, but $\{1,2\}$ and $\{1,3\}$ remain tabu. In the next iteration, edge $\{v_1^s, v_3^t\}$ has the highest flow of 2 but is tabu and therefore optBal selects the only non-tabu edge receiving flow, $\{v_3^s, v_2^t\}$ (Figure 2c). Executing procedure bb on groups 2 and 3 yields weights $w = (7,4,4)$. Since groups 2 and 3 are now balanced, no improving exchange between them is feasible and the pair $\{2,3\}$ is hopeless. In the fourth iteration all pairs are tabu, so optBal selects the edge with highest flow $\{v_1^s, v_2^t\}$ for the next exchange (Figure 2d).

6.2. Finding improving exchanges between two groups

The subproblem of optimally exchanging objects between two groups is: given a solution S and two groups k and l, find a sequence of shifts between them such that new the dispersion is not less than $D(S)$ and the new contribution $B_{kl}(S) = B(S_k) + B(S_l)$ of k and l to the imbalance of S is minimized. This problem is NP-complete, since the special case where $m = 2$, $D(S) = 0$, $M_1 = M_2$ and $\alpha = 0$ generalizes the Subset Sum Problem.

Given S, k and l we represent a solution by $x \in \{0,1\}^{|S_k \cup S_l|}$, where $x_i = 1$ if $i \in S_k \cup S_l$ is to be shifted to the other group, and $x_i = 0$ if i is to stay in its current group. We use a branch-and-bound algorithm to solve the problem. At each node in the search tree, the algorithm branches on the assignment of an unfixed variable x_i associated with the object of largest weight a_i, as empirically this leads to fewer expanded nodes. Since $D(S)$ is not allowed to decrease, fixing a single variable can potentially lead to a large sequence of fixations. When fixing $x_i = 1$, all conflicting x_j (i.e., $d_{ij} < D(S)$) may also be set to 1, since i and j cannot be placed in the same group. Similarly, fixing $x_i = 0$, all conflicting x_j can fixed to 0 as well. This is done recursively: when fixing a conflicting x_j, all of j's conflicts are also fixed, and so on. If we find that two conflicting objects should be fixed to the same group, the node is infeasible and can be pruned. The algorithm expands nodes depth first, giving preference to the child node of least lower bound, since this empirically leads to fewer expanded nodes. Because the decision variables represent the exchange of objects between the groups, we can obtain an upper bound at any node by setting all unfixed x_i to 0 (i.e., they remain in their current group).

For each new node a lower bound on B_{kl} is computed and the node is fathomed if it is not less than the current best value. Lower bounds are computed by ignoring conflict and integrality constraints of the unfixed objects, i.e., solving a relaxed version of the problem where any two objects can be placed together, and objects can be partially assigned. Let v_k and v_l be the total weights of the objects fixed to groups k and l, respectively, and let $r = (\sum_{i \in S_k \cup S_l} a_i) - v_k - v_l$ be the weight of the remaining objects that could still be assigned to the groups. The lower bound is then computed by finding a real-valued split $r = r_k + r_l$ of the remaining weight which minimizes B_{kl}, where r_k and r_l are the weights that will be (partially) assigned to groups k and l, respectively. This split is determined by minimizing the sum of expected imbalances

$$\max\{0, |v_k + r_k - M_k|/M_k - \alpha\} + \max\{0, |v_l + r - r_k - M_l|/M_l - \alpha\},$$

over all $r_k \in [0, r]$ (note that v_k, v_l, M_k, M_l and α are constants). This function is piecewise linear and therefore the optimum value always occurs either at the bounds of the domain or the extremes of some segment (see Figure 3 for an example). In other words, the optimal r_k can be found by examining, in $O(1)$, the six values 0, r, $-v_k + M_k(1 \pm \alpha)$, $v_l + r - M_l(1 \pm \alpha)$ for r_k.

7. Computational experiments

In this section, we report on computational experiments. We first describe in Section 7.1 the instances and the general experimental methodology. In Section 7.2

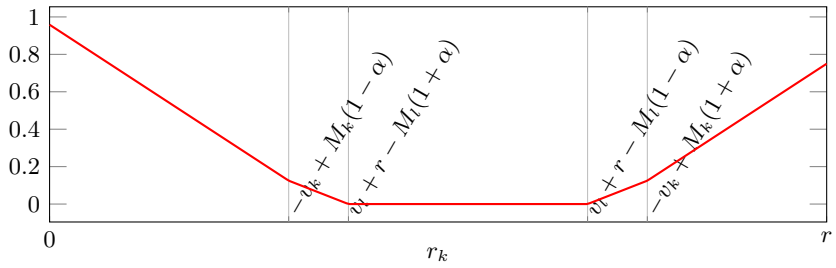

Figure 3: Example objective function when fractionally balancing two groups.

we calibrate upper bound U_h^σ and compare it to upper bounds from the literature. Next, in Sections 7.3 and 7.4 we calibrate and analyze the effectiveness of two algorithmic components, namely solving the UMaxDP and balancing solutions, separately. We also compare our approach to solving the UMaxDP to the literature. Finally, in Sections 7.5 and 7.6 we present the main results. In Section 7.5 we compare our approach to the exact approach of Fernández et al. [10], and in Section 7.6 we present and analyze results on large test instances. We summarize our results in Section 7.7.

7.1. Test instances and methodology

We use two types of instances, "WEEE" and "study", originating from the applications discussed in Section 1. For each type our test set comprises instances of sizes $n = 400i$ and $m = 5+6i$, for $i \in [10]$ generated as proposed by Fernández et al. [10]. The number of objects n was chosen to match the range of instances solvable by our heuristic algorithm (instances smaller than $n = 400$ were almost always solved optimally, while instances larger than $n = 4000$ were intractable within an hour). For group size m we chose the largest (relative to n) of the three values used by Fernández et al. [10], since this leads to harder instances.

The WEEE instances consist of objects uniformly distributed in the plane, and distances d are Euclidean. The object weights are drawn uniformly from $[1000, 4000]$, and the target weights M_k from $[(1-\beta)\bar{A}, (1+\beta)\bar{A}]$, where $\bar{A} = \sum_{i \in V} a_i/m$ is the average group weight and $\beta \in [1, 4]/4$ a slack parameter. For each combination (n, β) the test set contains 10 replicates, for a total of 400 WEEE instances.

The distances of the study instances are generated using Likert scales [22]. Each object is associated with a random vector in $[5]^{25}$, and the distance between two objects is the l_1 distance between these vectors. The target weights are chosen in the same way as for the WEEE instances, but here we use $\beta \in \{0, 0.1\}$. Again,

for each combination (n, β) the test set contains 10 replicates, for a total of 200 study instances. As noted above, by definition of the distances we have $|R| \leq 125$ and therefore the set L of critical pairs of a solution is usually large.

Further, we use an additional set of instances, which is used to calibrate the different components of the algorithms. The calibration set consists of 10 instances for each of the 10 values of n and m used in the test set, each with the largest value of β (1 for WEEE instances, 0.1 for study instances), for a total of 200 instances. We used the largest β because these instances are more difficult to balance.

Finally, we use a baseline set of 860 instances consisting of the same number of replicates and parameters n, m and β as the ones used by Fernández et al. [10], where n ranges from 100 to 300 for study instances and from 200 to 700 for WEEE instances. We use this data set to compare to the exact method of Fernández et al. [10], since it cannot handle larger instances in practical time.

We have executed all experiments on a PC with an 8-core AMD FX-8150 processor and 32 GB of main memory, running Ubuntu Linux 18.04. For each test, only one core was used. The heuristic algorithms were implemented in C++ and compiled with GCC 7.2 using maximum optimization. Model (F) is solved using CPLEX 12.7.1. Source code, data sets, instance generators, and detailed results are available at `http://www.inf.ufrgs.br/algopt/maxdp`.

7.2. Experiment 1: upper bounds

7.2.1. Calibrating parameter σ for upper bound U_h^σ

In this experiment we calibrate the subinstance size σ of upper bound U_h^σ on the calibration data set. We have considered values $\sigma = \min\{3, i\,m\}$ for $i \in \{0.05, 0.1, 0.25, 0.5, 1.0, 1.5, 2.0\}$. Each run was limited to 30 minutes, and executes only the algorithm to compute U_h^σ. Table 1 shows the results. For each value of σ and instance type we report the running time t in seconds, the average relative deviation of the upper bound from the best known upper bound (UB (r.d.)), the average number of iterations i_{best} to find the upper bound, and the average running time t_{col} in seconds of the exact coloring algorithm.

We see that up to a certain point, larger subinstance sizes tend to yield better upper bounds. This is expected, since larger subinstances are more likely to contain intersection graphs which are not m-colorable. After $\sigma/m > 2$ for study instances and $\sigma/m > 1.5$ for WEEE instances, however, the intersection graphs become too large to be colored optimally in feasible time, and thus the quality of the upper bounds decreases, as the algorithm reaches the time limit before considering all n possible subinstances. This is supported by the coloring times t_{col}: the values of σ/m where the best upper bounds are found ($\sigma/m = 2$ for study and $\sigma/m = 1.5$ for WEEE instances) appear to also be a sweet spot for coloring, where smaller σ lead to trivial coloring subproblems and larger σ to subproblems which are too hard

Table 1: Calibrating parameter σ of U_h^σ.

	σ/m	t	UB (r.d.)	i_{best}	t_{col}
study	1.05	4.9	3.43	15.0	0.0
	1.10	4.9	3.39	13.7	0.0
	1.25	5.3	3.09	14.6	0.0
	1.50	5.4	2.65	13.5	0.0
	2.00	388.7	**1.38**	8.4	28.7
	2.50	992.0	7.20	4.2	206.2
	3.00	1281.7	10.69	2.8	291.9
WEEE	1.05	11.7	0.21	13.0	0.0
	1.10	11.7	0.20	34.5	0.0
	1.25	12.1	0.17	36.9	0.0
	1.50	114.4	**0.06**	29.4	5.8
	2.00	583.8	0.08	15.4	237.1
	2.50	731.4	0.13	5.9	438.0
	3.00	857.0	0.15	4.9	479.0

to solve in feasible time. We therefore set $\sigma = 1.5m$ in the following experiments, as this value performs well for both instance types.

7.2.2. Comparing U_h^σ, U_h^C and U_h^{RB}

In this experiment we compare the new upper bound U_h^σ to upper bounds U_h^C and U_h^{RB} from the literature on the test set. For a fair comparison, we start optimizing U_h^σ from $d^u = d^R$ instead of $d^u = \mathrm{U}_h^C$. Running times were limited to 30 minutes.

Table 2 shows the results. For each instance type and size we report averages of the running time t in seconds, the relative deviation of the obtained bound from the best known upper bound (UB (r.d.)) and the optimality rate (opt. (%)), i.e., the empirical probability of the upper bound to match the value found by our heuristic (using results of Section 7.3).

We find that the proposed bound U_h^σ was smallest for all instance sizes, on average, though often at the cost of being more computationally expensive. In particular, for WEEE instances of size $n \geq 2800$ we running times increase sharply as the sampled subinstances are larger and thus lead to harder coloring subproblems. Nonetheless, we will show in Section 7.3.3 that using U_h^σ even in these cases is still effective, as it leads to an overall faster execution. Regarding the other two bounds, U_h^C performed poorly for study instances, but was slightly better than U_h^{RB} for WEEE instances. As we saw in Theorem 3.1 upper bounds $\mathrm{U}^C = \mathrm{U}^1$ and $\mathrm{U}^{RB} = \mathrm{U}^2$ are the same, so these differences are due to the sampling heuristics.

Table 2: Comparison of upper bounds U_h^C, U_h^{RB} and U_h^{σ}.

	n	U_h^C			U_h^{RB}			U_h^{σ}		
		t	UB (r.d.)	opt.	t	UB (r.d.)	opt.	t	UB (r.d.)	opt.
study	400	0.0	11.74	0	0.1	2.14	0	0.1	**0.53**	0
	800	0.1	15.32	0	0.5	2.38	0	0.2	**0.00**	0
	1200	0.2	16.97	0	1.9	0.87	0	0.5	**0.00**	0
	1600	0.4	20.35	0	4.8	2.16	0	1.2	**0.00**	0
	2000	0.8	20.19	0	10.1	2.05	0	2.3	**0.00**	0
	2400	1.3	18.93	0	19.1	**0.00**	0	3.8	**0.00**	0
	2800	2.1	19.03	0	32.7	1.17	0	6.0	**0.00**	0
	3200	3.1	18.04	0	52.6	1.17	0	8.9	**0.00**	0
	3600	4.5	20.72	0	80.0	1.59	0	12.7	**0.00**	0
	4000	6.7	18.97	0	117.5	2.61	0	18.0	**0.00**	0
	Avg.	1.9	18.03	0	31.9	1.61	0	5.4	**0.05**	0
WEEE	400	0.0	**0.00**	100	0.1	2.11	60	0.1	**0.00**	100
	800	0.1	0.37	90	0.5	0.39	80	0.3	**0.00**	100
	1200	0.4	0.03	90	1.9	3.60	60	0.8	**0.00**	100
	1600	0.9	1.12	90	5.1	1.86	40	2.2	**0.00**	100
	2000	1.8	0.18	80	11.6	0.25	70	4.4	**0.00**	90
	2400	3.2	0.12	70	23.0	0.95	50	7.9	**0.00**	80
	2800	5.3	0.41	40	41.5	2.93	0	266.0	**0.00**	68
	3200	8.3	0.93	58	68.8	3.60	10	199.3	**0.00**	68
	3600	12.4	0.22	30	108.4	3.35	8	55.2	**0.00**	50
	4000	18.7	0.25	28	162.0	3.17	10	411.6	**0.00**	28
	Avg.	5.1	0.36	68	42.3	2.22	39	94.8	**0.00**	78

Looking at optimality rates, we can see that for the WEEE instances the upper bounds frequently match the best heuristic value, and are useful for proving optimality. This is not the case for the study instances, where upper bounds never match the heuristic value. For this reason, in the experiments that follow we do not compute upper bounds for study instances.

7.3. Experiment 2: solving UMaxDP

7.3.1. Calibrating parameters p_1, p_2 and p_3

We used irace [24] to calibrate parameters p_1, p_2 and p_3 of algorithm EX. To test a parameter setting we generate an initial solution with the greedy constructive algorithm of Section 4.1 and execute Algorithm optDisp repeatedly until the current solution cannot be improved or a time limit is reached. We ran irace on the calibration data set with a budget of 4000 runs limited to 600 seconds each, over

parameter ranges $p_1 \in [8], p_2 \in [16]$ and $p_3 \in [24]$. We did not use upper bounds in the calibration. The best values found were $p_1 = 4$, $p_2 = 8$ and $p_3 = 8$, and are used in further experiments.

7.3.2. Comparison to Fernández et al. [10]'s method for the UMaxDP

In this experiment we compare the proposed ejection chain approach to improving dispersion to the approach of Fernández et al. [10], here called FKNU, to obtain lower bounds to the unrestricted MaxDP. To obtain a lower bound with our method, we start on an initial solution obtained by the algorithm of Section 4.1, and repeatedly execute the EX procedure until it fails to improve D'. FKNU starts from a dispersion $d^u = \mathrm{U}_h^C$ and iteratively decrements index u. At each iteration, it attempts to color the intersection graph G_{d^u} with the heuristic algorithm TabuCol [16]. If TabuCol reports $\chi(G_{d^u}) \leq m$ the algorithm returns the dispersion value d^{u+1}, otherwise it continues until $u = 1$, in which case coloring is trivial since G_{d^1} has no edges. We use the implementation of Lewis et al. [21] of TabuCol, with a fixed seed. Both algorithms were executed with a time limit of 30 minutes.

Table 3 shows the results. For both approaches we report the time t in seconds and the relative deviation (D (r.d.)) of the dispersion from the best known value. Since in the last iteration our algorithm will continue optimizing until the search tree is exhausted, which often consumes all available time, we also report the time to find the best value t_{best}, in seconds. This is not applicable to FKNU, since it optimizes top-down and stops as soon as a feasible solution is found.

We find that algorithm EX leads to better lower bounds, with two exceptions for study instances of size 1200 and 4000, where it is slightly worse. On the WEEE instances, in particular, lower bounds are significantly better. Finding better lower bounds takes longer, often considerably, and we also can see that EX usually consumes the available time, since it continues optimizing until the search tree is exhausted. Algorithm EX scales better on the WEEE instances, finding better upper bounds faster for $n \geq 2800$ objects. For such instances FKNU does not scale well with n since the number of different distances $|R|$, and thus the average number of iterations required, grows quadratically.

7.3.3. Effectiveness of upper bounds for the WEEE instances

In this experiment we evaluate the effectiveness of upper bound U_h^σ in reducing total running time by allowing optimization to stop as soon as the upper bound is reached, and thus skip the last iteration of EX. As reported in Section 7.2.2, upper bounds were not effective for study instances, so here we consider WEEE instances only. We use the results of the experiments of Sections 7.2.2 and 7.3.2. Table 4 shows, for each value of n, averages of the optimality rate opt. (%) of U_h^σ, the time

Table 3: Comparison of our ejection chain-based algorithm to the approach of [10] for the UMaxDP.

	n	EX			FKNU	
		t	t_{best}	D (r.d.)	t	D (r.d.)
study	400	99.5	0.4	**0.00**	0.4	1.84
	800	1727.9	2.6	**0.00**	0.7	**0.00**
	1200	1725.4	13.4	0.26	1.4	**0.00**
	1600	1800.0	31.0	**0.00**	2.8	**0.00**
	2000	1800.0	64.2	**0.00**	4.7	**0.00**
	2400	1800.0	32.0	**0.00**	7.5	**0.00**
	2800	1800.0	102.9	**0.00**	10.9	**0.00**
	3200	1800.0	48.6	**0.00**	15.4	**0.00**
	3600	1800.0	81.5	**0.00**	20.9	**0.00**
	4000	1800.0	92.5	0.98	27.2	**0.00**
	Avg.	1615.3	46.9	**0.12**	9.2	0.18
WEEE	400	17.0	0.1	**0.00**	0.0	**0.00**
	800	1384.9	0.4	**0.00**	1.1	0.10
	1200	1676.2	1.6	**0.00**	1.2	**0.00**
	1600	1792.1	11.2	**0.00**	79.3	0.95
	2000	1800.0	68.8	**0.00**	150.7	1.08
	2400	1800.0	189.1	**0.00**	588.4	2.24
	2800	1800.0	340.1	**0.00**	1244.0	45.30
	3200	1800.0	347.1	**0.00**	1417.8	42.11
	3600	1800.0	768.1	**0.00**	1547.5	69.60
	4000	1800.0	1138.8	**0.00**	1800.0	86.89
	Avg.	1567.0	286.5	**0.00**	683.0	24.83

t (EX) needed to compute EX, the time t (EX, last iter.) of the last iteration of EX, the time t (U_h^σ) to compute U_h^σ, and the time t (EX+U_h^σ) which is the average time needed to compute U_h^σ plus the time to best, for instances which were solved optimally, or the total time, for instances which were not.

We see that the last iteration is the most time-consuming part of the EX method. The reason is that when no improvement is possible, EX must exhaust the search tree before halting. Consequently, the last iteration consumes in average more than 75% of the running time. We see in column t (U_h^σ) that the average time to compute the upper bound is usually smaller than the time for the last iteration of EX. Therefore, when the optimal solution is found and matches the upper bound, the overall running time decreases. This is confirmed by the results in column t (EX+U_h^σ), where running times are on average about 50% smaller than t (EX). As n grows, however, particularly after $n \geq 4000$, optimality rates decrease and

Table 4: Effectiveness of upper bounds for algorithm optDisp, for WEEE instances

n	opt. (%)	t (EX)	t (EX, last iter.)	t (U_h^σ)	t (EX+U_h^σ)
400	100	17.0	8.9	0.1	8.2
800	100	1,384.9	745.7	0.3	639.5
1,200	100	1,676.2	1,406.3	0.8	270.7
1,600	100	1,792.1	1,714.2	2.2	80.2
2,000	90	1,800.0	1,731.2	4.4	249.5
2,400	80	1,800.0	1,610.9	7.9	443.9
2,800	68	1,800.0	1,459.9	266.0	992.3
3,200	68	1,800.0	1,452.9	199.3	897.4
3,600	50	1,800.0	1,031.9	55.2	1,171.1
4,000	28	1,800.0	661.3	411.6	1,853.9
Avg.	78	1567	1182.3	94.8	660.7

upper bounds becomes less advantageous.

7.4. Experiment 3: balancing solutions

7.4.1. Calibrating parameters ξ, θ and ρ

Parameters ξ, θ and ρ of optBal have been calibrated by irace, using the calibration data set, with a budget of 4000 runs and a time limit of 30 minutes per run. Each run executes the full algorithm with $\alpha = 0.001$ and reports the dispersion value, or $-\infty$ if no feasible solution was found. We used parameter ranges $\xi \in 5 \times [10], \theta \in \{0.1, 0.25, 0.5, 1, 2, 5\}$ and $\rho \in [0.01, 0.2]$. The best values found were $\xi = 35$, $\theta = 0.5$ and $\rho = 0.01$, which are used in the remaining experiments.

7.4.2. Comparison to a variable neighborhood search-based approach

To evaluate the effectiveness of our balancing method, we implemented a VNS approach as a baseline. The VNS iteratively searches for an improving operation in the neighborhoods $N^*_{shift} = N_{shift} \cap \{i \to k \mid w_{S(i)} > M_{S(i)} \wedge w(S_k) < M_k\}$ (i.e., we only allow shifts where a group exceeding its target weight sends an object to a group below its target weight), and N_{swap}. We use N^*_{shift} since we found it to be more effective than of N_{shift} in preliminary tests. The search explores the candidates of each neighborhood in a round-robin fashion and applies improving candidates as soon as they are encountered, proceeding then to the next iteration. If no improving candidates are found, the current best solution is "shaked" and the search continues from there. Shaking is done by performing up to $n\rho_{vns}$ random shifts which do not lower the current dispersion. The search stops after a maximum ξ_{vns} shakes. We used irace to calibrate parameters ρ_{vns} and ξ_{vns} as in Section 7.4.1, and found the best configuration $\rho_{vns} = 0.03$ and $\xi_{vns} = 50$.

Table 5: Comparison of our algorithm for balancing solutions to a VNS approach.

	n	Hase			VNS		
		t	D (r.d.)	opt. (%)	t	D (r.d.)	opt. (%)
study	400	0.3	**0.28**	0.0	3.9	0.42	0.0
	800	0.5	**0.00**	0.0	0.5	**0.00**	0.0
	1200	1.3	**0.00**	0.0	1.9	**0.00**	0.0
	1600	3.0	**0.00**	0.0	3.7	0.06	0.0
	2000	5.8	**0.00**	0.0	7.6	**0.00**	0.0
	2400	9.8	**0.00**	0.0	12.2	0.03	0.0
	2800	15.8	**0.00**	0.0	18.3	**0.00**	0.0
	3200	24.7	**0.00**	0.0	25.4	**0.00**	0.0
	3600	35.1	**0.00**	0.0	37.8	0.05	0.0
	4000	47.7	**0.00**	0.0	45.4	**0.00**	0.0
	Avg.	14.4	**0.03**	0.0	15.7	0.06	0.0
WEEE	400	0.4	0.71	97.5	0.7	**0.65**	89.6
	800	36.4	**0.53**	90.8	24.7	1.50	77.4
	1200	60.6	**0.27**	85.8	46.5	1.43	79.6
	1600	144.4	**0.94**	88.9	301.1	2.59	60.0
	2000	224.2	**1.10**	56.7	535.5	3.55	47.4
	2400	271.3	**1.03**	44.2	687.9	5.75	38.9
	2800	471.2	**0.41**	78.4	771.8	8.21	33.6
	3200	618.7	**0.67**	56.2	920.7	9.69	27.1
	3600	828.8	**0.56**	23.3	1050.9	13.53	17.5
	4000	779.8	**0.65**	56.3	839.6	10.75	15.6
	Avg.	343.6	**0.69**	67.8	517.9	5.77	48.7

For the comparison, we executed both algorithms on the test set with $\alpha \in \{5, 1, 0.1\} \times 10^{-2}$ for WEEE instances and $\alpha = 10^{-3}$ for study instances. We use these values of α to be consistent with Fernández et al. [10]. Table 5 shows the results. Since the VNS is stochastic, each test was replicated 7 times and we report averages. For both approaches we report the time t in seconds of the balancing procedure, the relative deviation (D (r.d.)) of the dispersion from the best known value, and the optimality rate (opt. (%)), which accounts the cases where upper bounds were met and cases where the instance was proven infeasible by model (F). For each instance size n we report averages over all values of α and β.

The results show that Hase produces better results than VNS, in less time. On study instances both algorithms find solutions of almost the same quality in the same time, but on WEEE instances Hase finds significantly better solutions and is about $40\,\%$ faster, with a single exception at $n = 400$ where the relative

deviation of the dispersion is slightly higher. In all study instances the execution stopped because EX timed out, and every lower bound on UMaxDP found by EX was made feasible by both approaches. This suggests that study instances are easy to balance, but that improving their dispersion is hard. The running times also corroborate this. Since $|R|$ is small there are typically several critical pairs, and thus an increasingly larger number of moves are necessary for each improvement to D. The better dispersion values of Hase on WEEE instances can be explained by its higher success rate in balancing solutions, so it can continue optimizing the dispersion. For both instance types, the number of feasible solutions was the same, i.e., either both approaches found a feasible solution, or none did. In 39% of executions our approach obtained the best solution, while the VNS approach was better in 10% of cases.

7.5. Experiment 4: comparison to the exact approach of Fernández et al. [10]

In this experiment we compare our hybrid heuristic to the exact algorithm of Fernández et al. [10], called FKN here, on the baseline data set, with a time limit of one hour. We compare only to FKN because it dominates the heuristic of Moeini et al. [25]. We implemented FKN using Julia 0.6.1 and CPLEX 12.7.1. The upper and lower bounds of the algorithm are computed in C++ and given as parameters. Since solving the MIP models are the bottleneck of FKN, the performance loss of using Julia as opposed to C++ is negligible. Tables 6 and 7 show the results for WEEE and study instances, respectively. We report averages for each set of instances with the same number of objects n and slack β, since the number of groups m and slack α have little influence on difficulty. The number of instances of each configuration is given in column #. For each algorithm we report the running time t in seconds, and the number of optimal solutions found (opt.). For the hybrid heuristic, column (opt.) considers both the case where optimality was proven by matching the upper bound, and the case where the optimal solution was found but not proven. We also report for Hase the number of solutions with optimality proofs (prv.), and the average relative deviation (D (r.d.)) in percent from the optimal solution, considering only cases where FKN found optimality. For FKN we report the average running time (t (opt.)), considering only instances which were solved optimally. In all cases, FKN either found the optimal solution, proved infeasibility or reached the time limit without obtaining any feasible solution.

For study instances, we see that FKN finds optimal solutions for all instances of $n = 100$ and around half the instances of $n = 200$ and $n = 300$. Using upper bounds the hybrid heuristic was able to find provenly optimal solutions on 54 tests, while in the remaining 29 it found the optimal solution but was not able to prove it. For $n = 200$ and $\beta = 0.1$ the hybrid heuristic found proven optimal solutions to 2 additional instances for which FKN did not. This suggests that,

though results from Section 7.2.2 indicated upper bounds were not effective on study instances of the test set, they are sometimes useful on smaller instances. This is due to the baseline data set also having instances with fewer groups m, while the test set considers only the largest value of m used by Fernández et al. [10]. Overall, Hase found slightly fewer optimal solutions, 83 compared to 86 of FKN. On the instances which FKN solved optimally, the hybrid heuristic produces solutions which deviate, on average, 0.26% from the optimal values.

For WEEE instances the hybrid heuristic found a more optimal solutions than FKN (2012 compared to 1981), and most these, namely 1979, were proven optimal by the heuristic. Again, this confirms that upper bounds are much tighter on WEEE instances. After $n = 600$, Hase consistently finds more optimal solutions than FKN, as instances become too large for the exact algorithm to solve in one hour. Considering only instances where FKN found the optimal solution, our method produced solutions within 0.22% of optimality, on average. We can also notice that, for WEEE instances, there is a slight increase in difficulty for $\beta = 0.75$ and a sharp increase for $\beta = 1$, for both approaches. For study instances, as expected $\beta = 0.1$ was slightly harder than $\beta = 0$.

Concerning running times, the hybrid heuristic was significantly faster in all cases, as expected, with running times 24.8 times smaller for study instances and 10.4 times smaller for WEEE instances, on average.

7.6. Experiment 5: final results on test instances

In this section we report full results for the hybrid heuristic on the test set. We have run Hase with a time limit of one hour, and report in Table 8 the relative deviation of the dispersion from the best upper bound (D (r.d. ub)), the optimality rate (opt. (%)), the rate of feasibility (feas. (%)) over instances which were not proven infeasible by model (F), the total running time t in seconds, and the number of iterations (iter.). For each instance type and number of objects n we report averages over all values of α and β. As can been seen in Tables 6 and 7, the running times of the exact approach of Fernández et al. [10] increase sharply with the number of objects n, and the optimality rates decrease accordingly. We therefore do not report results for FKN on the test set.

We observe that dispersion values for study instances deviate about 25% from the upper bound, corroborating previous findings that the upper bounds are never effective for these instances. As a consequence no instance was solved to proven optimality. For WEEE instances, Hase usually either finds the optimal solution, or terminates with a relatively large relative deviation from the upper bound (about 10%). This is likely due to the heuristic reaching a difficulty barrier with respect to balancing, as upper bounds only consider the subproblem without balance constraints. As expected, optimality rates decrease with increasing n.

Looking at the running times, we see that the hybrid heuristic tends to terminate well before the time limit on WEEE instances, since upper bounds are tighter and often match the dispersion values found. On study instances, on the other hand, due to upper bounds not being as tight the hybrid heuristic times out for $n \geq 1200$, since without optimality detection algorithm EX must be executed until completion.

Concerning feasibility rates, the hybrid heuristic either found a feasible solution or proved infeasibility in 97% of the cases, except for a few difficult instances of size $n = 2800$ and $n = 4000$. Out of the 1400 configurations (instance, α), 20 were proven infeasible by model (F).

7.7. Summary of results

In summary, we have shown that the proposed upper bound U_h^σ is consistently better than bounds U_h^{RB} and U_h^C from the literature, and matched lower bounds found by our heuristic more often. For WEEE instances, U_h^σ was effective in allowing algorithm EX to skip its final iteration whenever it reached optimality, reducing running time by 58%. For study instances, no upper bound matched the lower bounds found heuristically. In an analysis of individual components we have found that the proposed ejection chain method EX finds significantly better dispersion values than algorithms from the literature, which were optimal in 78% of cases, and that the proposed truncated branch-and-bound strategy is significantly better than a baseline VNS approach. An effective balancing strategy allows heuristic Hase to execute more iterations, and thus improve the dispersion further. Compared to the exact algorithm of Fernández et al. [10] on smaller instances, Hase is up to two orders of magnitude faster and finds more optimal solutions. In cases where Hase did not find optimal values, their values deviate less than 0.26% from optimality, on average. Hase is also able to solve instances with up to $n = 4000$ objects.

8. Conclusions

We have proposed a new, hybrid heuristic and an improved upper bound for the Maximum Dispersion Problem and have shown experimentally that good solutions can be found one to two orders of magnitudes faster than with previous methods. These solutions are often provably optimal since they match the upper bounds.

Our strategy alternates between balancing the groups and increasing the maximum dispersion. We have applied similar strategies to other location problems, and believe that such an approach may be generally useful [14, 13] for solving problems with conflicting objectives of dispersion and balancing. The variable-depth neighbor search proposed here may also be useful in other facility location problems with similar max-min objective functions or constraints that can lead to

cascading conflicts. We have further shown that two previously proposed upper bounds are in fact the same, and that the new upper bound is considerably tighter and enables the heuristic to often prove optimality. Our heuristic algorithms use truncated and complete exact algorithms as subprocedures, a strategy we believe is of general interest, and we show that they lead to better results than simpler strategies. We believe that similar methods can lead to better heuristics for related problems such as the Maximally Diverse Grouping Problem [19, 3].

References

[1] Arani, T., & Lotfi, V. (1989). A three phased approach to final exam scheduling. *IIE Transactions (Institute of Industrial Engineers)*, *21*, 86–96.

[2] Baker, K. R., & Powell, S. G. (2002). Methods for assigning students to groups: a study of alternative objective functions. *Journal of the Operational Research Society*, *53*, 397–404.

[3] Brimberg, J., Mladenović, N., & Urošević, D. (2015). Solving the maximally diverse grouping problem by skewed general variable neighborhood search. *Information Sciences*, *295*, 650–675.

[4] Butsch, A., Kalcsics, J., & Laporte, G. (2014). Districting for arc routing. *INFORMS Journal on Computing*, *26*, 809–824.

[5] Coudert, O. (1997). Exact coloring of real-life graphs is easy. In *Proceedings of the 34th annual conference on Design automation conference - DAC '97* (pp. 121–126). ACM Press.

[6] Dirac, G. A. (1952). Some theorems on abstract graphs. *Proc. London Math. Soc.*, *s3-2*, 69–81.

[7] Duarte, A., Sánchez-Oro, J., Resende, M. G. C., Glover, F., & Martí, R. (2015). Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. *Inform. Sci.*, *296*, 46–60.

[8] Erkut, E. (1990). The discrete p-dispersion problem. *European Journal of Operational Research*, *46*, 48–60.

[9] European Parliament and The Council of the European Union (2012). Directive 2012/19/EU on waste electrical and electronic equipment (WEEE).

[10] Fernández, E., Kalcsics, J., & Nickel, S. (2013). The maximum dispersion problem. *Omega (United Kingdom)*, *41*, 721–730.

[11] Fernández, E., Kalcsics, J., Nickel, S., & Ríos-Mercado, R. Z. (2010). A novel maximum dispersion territory design model arising in the implementation of the weee-directive. *Journal of the Operational Research Society*, *61*, 503–514.

[12] Ford, L., & Fulkerson, D. (1956). Maximal flow through a network. *Canadian Journal of Mathematics*, *8*, 399–404.

[13] Gliesch, A., & Ritt, M. (2019). A generic approach to districting with diameter or center-based objectives. In *Genetic and Evolutionary Computation Conference - GECCO '19* (pp. 249–257).

[14] Gliesch, A., Ritt, M., & Moreira, M. C. O. (2018). A multistart alternating tabu search for commercial districting. In *Lecture Notes in Computer Science* (pp. 158–173). Springer volume 10782.

[15] Glover, F. (1996). Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, *65*, 223–253.

[16] Hertz, A., & Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing*, *39*, 345–351.

[17] Kalcsics, J. (2015). Districting problems. In *Location Science* (pp. 595–622). Cham: Springer International Publishing.

[18] Korman, S. M. (1979). The graph-colouring problem. *Combinatorial optimization*, (pp. 211–235).

[19] Lai, X., & Hao, J. K. (2016). Iterated maxima search for the maximally diverse grouping problem. *European Journal of Operational Research*, *254*, 780–800.

[20] Lei, H., Laporte, G., Liu, Y., & Zhang, T. (2015). Dynamic design of sales territories. *Computers and Operations Research*, *56*, 84–92.

[21] Lewis, R., Thompson, J., Mumford, C., & Gillard, J. (2012). A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers and Operations Research*, *39*, 1933–1950.

[22] Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, *140*.

[23] Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, *21*, 498–516.

[24] López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

[25] Moeini, M., Goerzen, D., & Wendt, O. (2018). A local search heuristic for solving the maximum dispersion problem. In *LNCS* (pp. 362–371). Springer volume 10752.

[26] Mulvey, J. M., & Beck, M. P. (1984). Solving capacitated clustering problems. *European Journal of Operational Research*, *18*, 339–348.

[27] Prokopyev, O. A., Kong, N., & Martinez-Torres, D. L. (2009). The equitable dispersion problem. *Eur. J. Oper. Res*, *197*, 59–67.

[28] Ríos-Mercado, R., Maldonado-Flores, J., & González-Velarde, J. L. (2017). *Tabu Search with Strategic Oscillation for Improving Recollection Assignment Plans of Waste Electric and Electronic Equipment*. Technical Report PISIS-2017-01, Universidad Autónoma de Nuevo León San Nicolás de los Garza, Mexico.

[29] Ríos-Mercado, R. Z., & Bard, J. F. (2019). An exact algorithm for designing optimal districts in the collection of waste electric and electronic equipment through an improved reformulation. *European Journal of Operational Research*, *276*, 259–271.

[30] Smith-Miles, K., Baatar, D., Wreford, B., & Lewis, R. (2014). Towards objective measures of algorithm performance across instance space. *Computers and Operations Research*, *45*, 12–24.

Table 6: Comparison of our heuristic algorithm to the exact algorithm of [10], for instances of type WEEE.

n	β	#	Hase				FKN		
			t	opt.	prv.	D (r.d.)	t	t (opt.)	opt.
200	0.25	90	0.1	88	88	0.04	11.3	11.3	**90**
	0.5	90	0.1	**90**	90	0.00	11.7	11.7	**90**
	0.75	90	0.5	88	88	0.01	17.4	17.4	**90**
	1	90	35.9	84	83	0.14	134.3	95.4	**89**
300	0.25	90	0.1	89	84	0.01	16.9	16.9	**90**
	0.5	90	0.1	88	83	0.03	15.8	15.8	**90**
	0.75	90	0.1	**90**	84	0.00	28.8	28.8	**90**
	1	90	1.1	84	76	0.51	141.3	62.7	**88**
400	0.25	90	0.2	88	88	0.29	43.5	43.5	**90**
	0.5	90	0.1	**90**	90	0.00	137.6	98.7	89
	0.75	90	0.2	**89**	89	0.04	205.1	167.0	**89**
	1	90	44.1	71	70	1.22	621.2	163.0	**78**
500	0.25	90	0.3	81	79	0.19	213.4	136.5	**88**
	0.5	90	0.3	87	84	0.00	265.7	228.2	**89**
	0.75	90	0.3	**86**	84	0.00	367.2	94.6	83
	1	90	183.7	59	59	0.82	1234.1	220.2	**63**
600	0.25	90	0.3	82	82	0.31	421.4	194.4	**84**
	0.5	90	0.4	**88**	88	0.09	510.1	208.7	82
	0.75	90	0.5	**89**	89	0.03	533.2	274.6	83
	1	90	48.7	**76**	76	0.38	1136.8	339.9	68
700	0.25	90	0.6	**85**	85	0.17	806.4	376.6	78
	0.5	90	0.6	**85**	85	0.05	881.1	422.0	77
	0.75	90	17.9	**85**	85	0.11	1037.8	305.8	70
	1	90	56.6	**70**	70	0.91	1805.6	553.0	53
	Total		16.4	**2012**	1979	0.22	441.6	170.3	1981

Table 7: Comparison of our heuristic algorithm to the exact algorithm of [10], for instances of type study.

n	β	#	Hase				FKN		
			t	opt.	prv.	D (r.d.)	t	t (opt.)	opt.
100	0	20	1.2	18	14	0.31	21.1	21.1	**20**
	0.1	20	0.4	17	13	0.46	135.3	135.3	**20**
200	0	30	5.5	**16**	8	0.00	1826.3	274.3	**16**
	0.1	30	11.1	**15**	9	0.00	2063.0	53.2	13
300	0	20	38.0	**9**	5	0.38	2310.9	735.3	**9**
	0.1	20	24.1	**8**	5	0.43	2470.5	776.3	**8**
Total			13.4	83	54	0.26	1471.2	332.6	**86**

Table 8: Results of Hase for large instances.

	n	D (r.d. ub)	opt. (%)	feas.	t	iter.
study	400	12.85	0.0	100.0	687.5	15.2
	800	19.11	0.0	100.0	3,339.2	4.2
	1200	21.78	0.0	100.0	3,600.0	3.3
	1600	22.42	0.0	100.0	3,600.0	3.4
	2000	22.22	0.0	100.0	3,600.0	3.6
	2400	24.01	0.0	100.0	3,600.0	3.5
	2800	24.41	0.0	100.0	3,600.0	3.4
	3200	25.68	0.0	100.0	3,600.0	3.5
	3600	24.83	0.0	100.0	3,600.0	3.6
	4000	25.67	0.0	100.0	3,600.0	3.5
	Avg.	22.30	0.0	100.0	3,282.7	4.7
WEEE	400	13.52	90.8	100.0	0.5	2.7
	800	13.66	84.2	100.0	48.2	9.1
	1200	6.41	85.8	100.0	65.5	11.8
	1600	9.62	69.2	100.0	194.6	29.2
	2000	9.42	56.7	100.0	470.9	46.0
	2400	9.97	44.2	100.0	677.3	60.0
	2800	8.57	37.5	90.0	1,422.5	76.4
	3200	8.91	32.5	100.0	1,468.3	102.3
	3600	9.50	28.3	100.0	1,970.1	114.4
	4000	11.01	19.2	80.0	2,231.5	119.2
	Avg.	10.06	54.8	97.0	854.9	57.1