Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 30, no. 1, pp. 1–24 (2026)
DOI: 10.7155/jgaa.v30i1.3166

Engineering Hypergraph b-Matching Algorithms[1]

Ernestine Großmann[1] *Felix Joos*[1] *Henrik Reinstädtler*[1] *Christian Schulz*[1]

[1]Institute for Informatics
Heidelberg University, Germany

Submitted: July 2024 Accepted: January 2026 Published: January 2026

Article type: Regular paper Communicated by: Petra Mutzel

Abstract. Recently, researchers have extended the concept of matchings to the more general problem of finding b-matchings in hypergraphs, broadening the scope of potential applications and challenges. The concept of b-matchings, where b is a function that assigns positive integers to the vertices of the graph, is a natural extension of matchings in graphs, where each vertex v is allowed to be matched to up to $b(v)$ edges, rather than just one. The weighted b-matching problem then seeks to select a subset of the hyperedges that fulfills the constraint and maximizes the weight.

In this work, we engineer novel algorithms for this generalized problem. More precisely, we introduce exact data reductions for the problem as well as a novel greedy initial solution and local search algorithms. These data reductions allow us to significantly shrink the input size. This is done by either determining if a hyperedge is guaranteed to be in an optimum b-matching and thus can be added to our solution or if it can be safely ignored. Our iterated local search algorithm provides a framework for finding suitable improvement swaps of edges. Experiments on a wide range of real-world hypergraphs show that our new set of data reductions are highly practical, and our initial solutions are competitive for graphs and hypergraphs as well.

[1]The Code of this publication can be found at `https://github.com/HeiHGM/Bmatching`. The dataset and code is archived at `https://doi.org/10.5281/zenodo.18225669`.

We acknowledge support by DFG grant SCHU 2567/8-1 and by the state of Baden-Württemberg through bwHPC.

E-mail addresses: E.Grossmann@informatik.uni-heidelberg.de (Ernestine Großmann) joos@informatik.uni-heidelberg.de (Felix Joos) henrik.reinstaedtler@informatik.uni-heidelberg.de (Henrik Reinstädtler) christian.schulz@informatik.uni-heidelberg.de (Christian Schulz)

1 Introduction

Graph theory has long been a crucial discipline in the world of mathematics and computer science, providing insights into numerous complex problems. One of the most well-known problems in graph theory is the matching problem. A matching in a graph is a set of pairwise vertex-disjoint edges. Computing (these) matchings in a graph is a ubiquitous combinatorial problem that has a myriad of applications in various fields [36]. By now, maximum weight/cardinality matchings in graphs in the internal-memory model have been extensively studied, leading to various breakthroughs. However, finding maximum (weight) matchings in graphs in the internal-memory model is only the tip of the iceberg. Recently, researchers have extended the concept of matchings to the more general problem of finding b-matchings in hypergraphs, broadening the scope of potential applications and challenges. A hypergraph is a natural graph extension in which edges can have more than two endpoints and thus model more complex relationships. For example, online hypergraph matching can be used to model auctions of advertisement campaigns [45]. Moreover, the concept of b-matchings (with $b : V \to \mathbb{N}$) is a natural extension of matchings, where each vertex v is allowed to be matched with up to $b(v)$ edges, rather than just one. The weighted b-matching problem then seeks to select a subset of the hyperedges that fulfill the constraint and maximize the weight.

Many applications require the computation of a matching $\mathcal{M}$ with certain properties, like being maximal (no edge can be added without violating the matching property), having maximum cardinality, or having maximum total weight $\sum_{e\in\mathcal{M}} \omega(e)$. For example, in multi-level (hyper)graph partitioning, the problem of coarsening a (hyper)graph without losing the characteristics of the original (hyper)graph in multi-level decomposition algorithms can be solved by computing a hypergraph matching problem [60]. Similarly, hypergraph b-matching plays a critical role in agglomerative hypergraph clustering [56], where hyperedges are evaluated based on the likelihood of merging adjacent clusters. In this context, the function b assigned to the vertices can serve as a mechanism to regulate the pace of agglomeration. Other important example applications include allocating resources to machines or auctioning goods [15], ride-sharing [58] and load balancing [38].

Currently, however, researchers have only developed approximation algorithms [55] for the weighted case and practical implementations of heuristics to tackle the hypergraph matching problem are only limited to special classes (d-uniform and d-partite) of hypergraphs without weight [21]. One powerful technique for tackling **NP**-hard graph problems is to use *data reduction rules*, which remove or contract local (hyper)graph structures to reduce the input instance to an equivalent but smaller instance. Originally developed as a tool for parameterized algorithms [14], data reduction rules have been effective in practice for computing a (weighted) maximum independent set [11, 48, 62] / minimum vertex cover [2], maximum clique [10, 65], and maximum k-plex [12, 41], as well as solving graph coloring [65, 50] and clique cover problems [33, 63], among others [1]. However, recent work has only scratched the surface for *weighted* problems, with some examples being [49, 30, 35, 67].

Our Results. In this work, we devise and engineer data reduction rules and new greedy initial solution algorithms for the weighted hypergraph b-matching problem in general hypergraphs. Furthermore, we present a local search for this problem. While our main focus is on the most general weighted hypergraph b-matching problem, we also compare our greedy initial solutions on graphs against solvers that are restricted to graphs. Our experiments show that we are able to obtain better initial solutions by greedy heuristics of up to 10 %, a speedup of 6.85 for exactly solving hypergraph b-matching, and quality improvements of up to 30 % by our local search algorithm for the 1-matching case.

2 Preliminaries

Basic Concepts. A *weighted undirected hypergraph* $H = (V, E, \omega)$ is defined as a set of n vertices V and a multiset of m hyperedges E with edge weights $\omega : E \to \mathbb{R}_{>0}$, where each edge $e \in E$ is a subset of the vertex set V. We define $\hat{\omega}(S) := \{\omega(x) \mid x \in S\}$ for some subset $S \subset E$. We assume hyperedges to be sets rather than multisets; that is, a vertex can only be contained in a hyperedge *once*, while multiple edges can contain the same set of vertices. Therefore, we write e for the set of vertices of a hyperedge e and define $|e|$ as the edge size. The maximum edge size is denoted by $\Delta_E := \max_{e \in E} |e|$.
We refer to the edges of a vertex by $E(v) := \{e \in E \mid v \in e\}$ and for a (multi-)set M of edges we define $M(v) := E(v) \cap M$. A vertex v is *incident* to an edge e if $v \in e$. The degree of a vertex v is $|E(v)|$ and $\Delta_V := \max_{v \in V} |E(v)|$ is the maximum degree. Two vertices u, v are *adjacent* if at least one edge is incident to both of them. Furthermore, two edges e, f are *adjacent* if $e \cap f \neq \emptyset$. We call two edges e, f *linked* if there are only vertices of degree 2 incident to e and f. A set of edges S in H is *independent* if for all distinct $f, g \in S$ f and g are disjoint. We define $\mathcal{N}(e) := \bigcup_{v \in e} E(v)$ as the closed neighborhood of an edge. We extend the weight function ω to sets naturally, that is, $\omega(F) := \sum_{e \in F} \omega(e)$. Given a subset $V' \subset V$, the *subhypergraph* $H_{V'}$ is defined as $H_{V'} := (V', \{e \cap V' \mid e \in E : e \cap V' \neq \emptyset\})$. For a set of edges R of a hypergraph $H = (V, E, \omega)$ we write $H \setminus R$ short for $(V, E \setminus R, \omega)$. A hypergraph is called d-partite if its node can be partioned into d sets such that no edge is adjacent to two vertices in the same set. If each edge has the same number d of vertices, a hypergraph is called d-uniform. A *weighted undirected graph* $G = (V, E, \omega)$ is defined as a set of n vertices V and a set of m edges E with edge weights $\omega : E \to \mathbb{R}_{>0}$. In contrast to hypergraphs, the size of the edges is restricted to 2. Throughout this paper, we use *edges* in the context of hypergraphs and graphs. An overview of the notations can be found in Table 1.

Definition	Description
$\omega : E \to \mathbb{R}_{>0}$	edge weight
$\hat{\omega}(S) := \{\omega(x) \mid x \in S\}$	edge weight on elements
$E(v)$	edges adjacent to v
Δ_E	maximum edge size
Δ_V	maximum node degree
$\mathcal{N}(e) := \bigcup_{v \in e} E(v)$	closed neighborhood
$b(v) : V \to \mathbb{N}$	capacity
$\beta := \max b(v)$	maximum capacity
$blocked(e, M)$	exhausted vertices of e

Table 1: Notation overview.

(Hyper)graph b-Matching. A matching $\mathcal{M} \subset E$ in a (hyper)graph is a set of (hyper)edges that are pairwise disjoint. The *cardinality* or *size* of a matching is simply the cardinality of the (hyper)edge subset $\mathcal{M}$. We call a matching *maximal* if there is no (hyper)edge in E that can be added to $\mathcal{M}$. A *maximum cardinality matching* $\mathcal{M}_c$ is a matching that contains the largest possible number of (hyper)edges of all matchings. A *maximum weight matching* $\mathcal{M}_\omega$ is a matching that maximizes $\omega(\mathcal{M}_\omega)$ among all possible matchings. In a *perfect* matching, every vertex is incident to an edge contained in the matching. For a given function $b : V \to \mathbb{N}$, the b-matching problem relaxes the edge-disjointness constraint so that each vertex can be incident to up to $b(v)$ edges. We define $b(v)$ as the capacity of vertex v and denote $\beta = \beta(b) = \max_{v \in V} b(v)$. For $b \equiv 1$, this is equivalent to the standard matching problem. By $blocked(e, \mathcal{M}) := \{v \in e \mid |\mathcal{M}(v)| = b(v)\}$ we refer to the set of all vertices of an edge e where the capacity is exhausted, in other words, we cannot add further edges to the matching. Similarly, we define $blockedEdges(e) := \bigcup_{v \in e : b(v) = 1} E(v) \setminus \{e\}$ as the edges blocked by an edge e. An edge e for which $blocked(e, \mathcal{M}) = \emptyset$ is called *free*. Finally, for a finite set $X \subset \mathbb{R}_{>0}$ let $\text{nmax}(X, k)$ denote its k-th largest value if it exists, otherwise 0.

Related Work. There is a vast amount of literature for matchings in graphs [22, 53, 29, 44, 19, 9, 59, 18, 51, 20]. We refer the reader to the respective papers for more details. For results in data reduction, we refer the reader to the recent survey [1]. We now cover related work closer to our main contribution, which are problem variations of the most general weighted hypergraph b-matching problem.

Graph b-Matching. The b-matching problem can be reduced to the simple matching problem according to Gabow [28] by substituting vertices, but this is impractical on large graphs. An overview of exact approaches can be found in Müller-Hannemann and Schwartz [54]. Grötschel and Holland [34] use the cutting plane technique to tackle the problem. Based on belief propagation and assuming a unique solution exists, Huang and Jebara [39] developed an exact algorithm for the b-matching problem. Mestre [52] proved that the greedy algorithm is a half-approximation and generalized the PGA algorithm by Drake and Hougardy [18] to achieve an $O(\beta m)$ time half-approximation. The LD algorithm was generalized to b-matching by Georgiadis and Papatriantafilou [31] in a distributed fashion. Khan et al. [43] introduced an approximation algorithm that can be executed in parallel called bSuitor, inspired by the results of Manne and Halappanavar [51] for normal matchings. Ferdous et al. [25] consider parallel algorithms for b-matchings with submodular objectives.

Hypergraph Matching. According to Hazan et al. [37], the maximum d-set packing problem and, therefore, the matching problem on d-partite, d-uniform hypergraphs can be poorly approximated, and there is no approximation within a factor of $\mathcal{O}(d/\log d)$. In general, as proven by Håstad [40], the matching problem in non-uniform hypergraphs and the maximum independent set problem are NP-hard and there is no $n^{1-\epsilon}$ factor approximation unless $P = NP$. There is a polynomial $(k+1+\epsilon)/3$-approximation algorithm for k-set packing, and therefore the matching problem in d-uniform, d-partite hypergraphs proposed by Cygan [13] using local search. Furthermore, Fürer and Yu [27] improved these results with respect to the run-time. Dufosse et al. [21] introduce several heuristics to reduce the complexity of the uniform problem by extending the two well-known Karp-Sipser [42] rules to hypergraphs. Dufosse et al. [21] present the idea of using the Sinkhorn-Knopp algorithm [61] for the normalization of incident tensors as a third selection rule. They perform practical experiments, but are limited to only d-partite, d-uniform hypergraphs with uniform edge weights. Anneg et al. [5] give an improved optimality bound for LP-relaxation for the non-uniform case. This extends to b-matching. For the weighted k-set packing problem Thiery and Ward [64] show improved approximation bounds of 1.786 for $k = 3$. Recently, Neuwohner [55] showed how to prove an approximation threshold lower than $\frac{k}{2}$ by a factor in $\Omega(k)$. Both approaches are improvements on long-standing local search ideas presented by Berman [7] for maximum weight independent set in d-claw free graphs.

Hypergraph b-Matching. The b-matching cardinality problem in hypergraphs has also no approximation scheme according to El Ouali and Jäger [24], even if the degree of vertices is bounded. Similarly, El Ouali et al. [23] showed that in k-uniform hypergraphs for the cardinality problem with $2 \leq b \leq k/\log k$, there is no polynomial-time approximation within any ratio smaller than $\Omega(\frac{k}{b \log k})$. For weighted b-matching on k-uniform hypergraphs Krysta [47] gave a greedy $k+1$ approximation, while Parekh and Pritchard [57] achieve a $(k-1+\frac{1}{k})$ approximation algorithm via linear programming. Koufogiannakis and Young [46] developed a k-approximation in a distributed fashion for weighted k-uniform hypergraphs. We are not aware of any practical implementation of those algorithms.

3 Hypergraph b-Matching Algorithm

We now give an overview of our algorithm to solve the general weighted hypergraph b-matching problem. In Section 3.1 we introduce the exact integer linear program for this problem. Our approach shown in Algorithm 1 starts by using exact data reductions devised in Section 3.2 to reduce the instance size. The reduced instances can then be used as input to the exact solver (based on the ILP) or our heuristic algorithm. Our heuristic algorithm computes a good initial b-matching using a greedy strategy, see Section 3.3. In Section 3.4 we introduce a local search, that improves solution quality by swaps. Once a solution is computed on the reduced instance we reconstruct it to a solution for the original instance by unfolding the reductions in reverse order.

3.1 Optimal Solutions

To solve the b-matching problem to optimality, either on the original or exactly reduced hypergraph, we use the following integer linear program:

$$\max \sum_{e \in E} x_e \cdot \omega(e) \quad s.t. \forall v \in V: \sum_{e \in E(v)} x_e \leq b(v) \quad x_e \in \{0,1\} \quad \forall e \in E. \tag{1}$$

For every edge $e \in E$, the integer linear program has a variable x_e, which is set to $x_e = 1$ iff the edge e is part of the matching and zero otherwise. The maximization term is the sum of the weights of the selected edges. The main constraint restricts the number of selected edges to satisfy the capacity at each vertex in the original hypergraph.

3.2 Exact Reduction Rules

Only few reductions are known that can be used for the hypergraph matching problem [21]. These data reductions are based on Karp-Sipser rules and are a) not applicable to weighted problems and b) not applicable to the more general b-matching problem in hypergraphs. However, especially for large instances, applying exact data reductions is a very important technique to decrease the problem size. In general, reductions allow the classification of edges as either (1) part of a solution, (2) non-solution edges, or (3) deferred, i. e., the decision for this edge depends on additional information about neighboring edges that will be obtained later. We denote by $\mathcal{K}$ the resulting *reduced hypergraph*, where no reduction rule applies anymore. In the following, we introduce a large set of new reductions for the weighted b-matching problem. An overview over all reductions can be found in Table 2. We now propose our first data reduction rule, that is the removal of abundant vertices. Intuitively, if the degree of a vertex is smaller than or equal to its capacity, the

Algorithm 1 b-Matching Algorithm.

```
1: procedure BMatching($H = (V, E, \omega)$)
2:     $\mathcal{K}, \mathcal{M}_{exact}, E_{folded} \leftarrow$ Reductions($H$)
3:     $\mathcal{M}_{Kernel} \leftarrow$ Greedy($\mathcal{K}$) or ILP($\mathcal{K}$)
4:     $\mathcal{M}_{post} \leftarrow$ Unfold($H, E_{folded}, \mathcal{M}_{Kernel}$)
5:     return $\mathcal{M}_{exact} \cup \mathcal{M}_{Kernel} \cup \mathcal{M}_{post}$
```

Name	**Condition**	**Action**	**Complexity**
Abundant Vertices (AV)	$\lvert E(v)\rvert \leq b(v)$	Remove v	$\mathcal{O}(\lvert V\rvert)$
Neighborhood Removal (NR)	$\omega(e)$ is larger than sum $b(v)$-heaviest neighbors	e is in the solution	$\mathcal{O}(\min(n\beta, m)\Delta_E + n\Delta_V + \log \Delta_V)$
Weighted Isolated Edge Removal (WIER)	e heaviest edge and neighbors share common $b(v) = 1$ vertex	e is in the solution	$\mathcal{O}(\min(m, n)\Delta_E^2)$
Weighted Edge Folding (WEF)	e has neighbors N independent with $b(v) = 1$,$\omega(e) > \omega(N) - \min_{f\in N} f$	fold e and N into e' with $\omega(e') := \omega(N) - \omega(e)$, defer if e or N is in solution	$\mathcal{O}(\min(m, n)\Delta_E)$
Weighted Twin (WT)	e_1, e_2 non adjacent, same neighbors N that are independent and $\omega(\{e_1, e_2\}) > \omega(N) - \min_{f\in N}\omega(f)$	Combine e_1 and e_2 into one edge and apply WEF or NR	$\mathcal{O}(m\Delta_E\Delta_V + m\log m)$
Weighted Domination (WD)	$\omega(e) \geq \omega(f)$ and $e \subset f$	Remove f	$\mathcal{O}(m\Delta_E\Delta_V \log \Delta_V)$

Table 2: Overview over all reductions presented in this paper.

vertex can be removed. A vertex $v \in V$ is considered *abundant* if its capacity $b(v)$ is equal to or exceeds its degree.

Reduction 3.1 (Abundant Vertices (AV)) *Any abundant vertex can be removed from the hypergraph. Moreover, any edge that becomes empty in the process can be included in an optimal solution.*

Proof: An abundant vertex v can be removed from the hypergraph as v does not restrict the selection problem, that is, all incident edges of v could in principle be contained in an optimum matching as the capacity is larger than the number of adjacent edges. Thus, we can remove v from the hypergraph. If there is an edge $e \in E$, only containing v, it is part of an optimal solution, since it cannot be blocked at any other vertex. □

Reduction 3.2 (Neighborhood Removal (NR)) *An edge $e \in E$ is in an optimal matching $\mathcal{M}_{opt}$ if e has a higher weight than the total sum of weights of the $b(v)$-th heaviest edge (excluding e) in each of its vertices v:* $\omega(e) \geq \sum_{v\in e} \text{nmax}(\hat{\omega}(E(v) \setminus \{e\}), b(v))$.

Proof: Let $\mathcal{M}_{opt}$ be an optimal solution and assume $e \notin \mathcal{M}_{opt}$. For each $v \in e$ with $b(v) = \lvert\mathcal{M}_{opt}(v)\rvert$ (conflicting vertices), we remove the lightest incident edge that is in the solution. Call this matching $\mathcal{M}'$. It follows for all vertices $v \in e$: $\lvert\mathcal{M}'(v)\rvert < b(v)$. This implies that $\mathcal{M}'' = \mathcal{M}' \cup \{e\}$ is also a valid matching. We now show $\omega(\mathcal{M}'') \geq \omega(\mathcal{M}_{opt})$. Let e'_v be a lightest edge removed from $\mathcal{M}_{opt}$ at v. Its weight $\omega(e'_v)$ contributing to $\mathcal{M}_{opt}$ is smaller or equal to the $b(v)$ heaviest edge incident to v, since $b(v)$ edges have been in $\mathcal{M}_{opt}$. Thus, $\omega(e'_v) \leq \text{nmax}(\hat{\omega}(E(v) \setminus \{e\}), b(v))$.

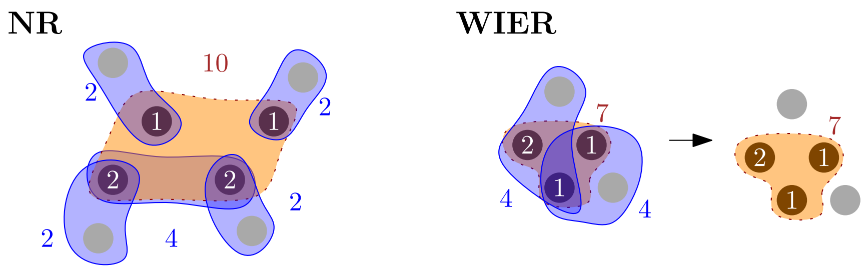


Figure 1: Examples for the second and third reductions. **Neighborhood Removal (NR):** The weight of the orange edge dominates the sum of the $b(v)$-th heaviest weights per vertex (white), in this case the blue ones($10 \geq 8$). There exists an optimal solution that contains the orange edge. **Weighted Isolated Edge Removal (WIER):** All edges overlap with each other and have a common vertex with capacity 1 each. The orange edge is part of an optimal matching, because it has the highest weight of the clique.

The weight of all edges removed from $\mathcal{M}_{opt}$ is smaller or equal to $\omega(e)$, if the equation holds. This yields $\omega(\mathcal{M}'') \geq \omega(\mathcal{M}_{opt})$. □

Figure 1 (NR) shows an example of Reduction 3.2 and in Algorithm 2 a naive algorithm is presented. We iterate over each vertex and check up to $b(v)$ incident edges. For each edge e, we calculate the sum of weights it needs to dominate and break early if the condition cannot be satisfied anymore. If we find a candidate, we can include it as part of an optimal matching and update the hypergraph and capacity accordingly. The time complexity for this algorithm is $\mathcal{O}(\min(n\beta, m)\Delta_E + n\Delta_V \log \Delta_V)$. At each vertex, we have to check up to β edges, and using a map for skipping already checked edges, we have in total up to $\mathcal{O}(\min(n\beta, m))$ candidates. The nmax operation is implementable in $\mathcal{O}(1)$ if the edge vector in each vertex is sorted. This initial sorting requires $\mathcal{O}(n\Delta_V \log \Delta_V)$ steps. We can find multiple reductions in the same pass.

Reduction 3.3 (Weighted Isolated Edge Removal (WIER)) *Let $e \in E$ be an edge that has the heaviest weight among its neighbors: $\omega(e) \geq \max_{f \in \mathcal{N}(e)} \omega(f)$. If for all $f, g \in \mathcal{N}(e)$ there exists a common vertex $v \in f \cap g$ with capacity $b(v) = 1$ then e is part of an optimal solution.*

Proof: Because e and all its adjacent edges contain at least one vertex with capacity 1, we can select at most one edge in $\mathcal{N}(e)$ while excluding all other edges in this neighborhood. An optimal solution $\mathcal{M}_{opt}$ must contain at least one edge of $\mathcal{N}(e)$, otherwise e can directly be added yielding a heavier matching which is a contradiction to the optimality of the matching. Given any optimal solution $\mathcal{M}_{opt}$ containing a neighbor f then $\mathcal{M}_{opt} \setminus \{f\} \cup \{e\}$ is also optimal since $\omega(e) \geq \max_{f \in \mathcal{N}(e)} \omega(f)$. □

A hypergraph where this reduction is applicable, is depicted in Figure 1 (WIER). In Algorithm 3, we give a procedure to detect isolated edges and apply the reduction efficiently in detail. At each vertex we have to scan the heaviest edge, but only if it has not been scanned before. We simultaneously check whether it has maximum weight at each other vertex and collect its neighbors. After checking if the weight condition is satisfied, we mark each edge incident to the currently scanned edge also as scanned. Because all neighbors weigh less than the currently scanned edge, they cannot be candidates themselves for this reduction at other vertices later. For neighbors at capacity 1 vertices, we save the position we scanned this edge in a bit mask. We add neighboring

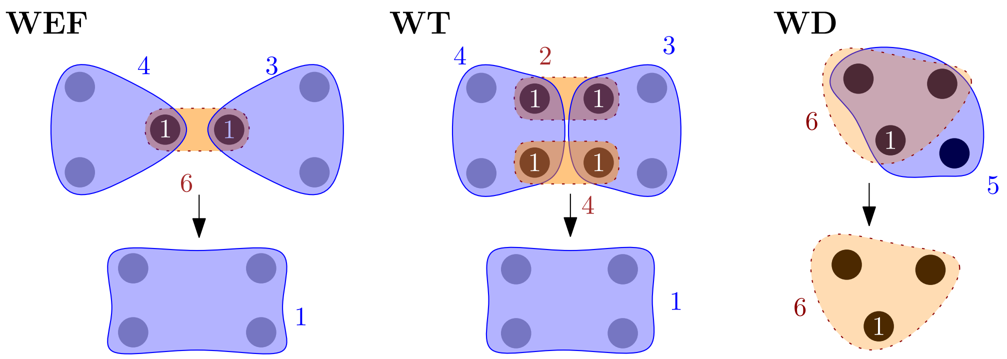


Figure 2: Examples for three reductions. **Weighted Edge Folding (WEF):** The orange edge has exactly two non-adjacent neighbors (blue), that it dominates one by one, but not in total. The three edges can be folded and later be decided on. **Weighted Twin (WT):** The orange edges have exactly two independent non-adjacent neighbors (blue), that they dominate one by one, but not in total. The four edges can be folded and later be decided on. The Weighted Edge Folding is directly applied. **Weighted Domination (WD):** The orange edge is a subset of the blue edge, has a higher weight, and they share a common vertex with capacity 1.

edges at higher capacity vertices in a vector N_l. Afterwards, we check if all neighbors are incident to a vertex with capacity 1 (property S) and if all pairs have a (blocking) common vertex of capacity 1. We use the bit mask to check, if all edges in N_l are incident to a capacity 1 vertex. This greatly reduces the number of checks required. This is accomplished by a *bitwise and* of the masks in N_b. If the result is zero the edges have no common capacity 1 vertex with e, and we have to do a detailed check. The overall complexity of this algorithm is $\mathcal{O}(\min(m,n){\Delta_E}^2)$ because in the worst case, we would collect Δ_E distinct neighbors at a vertex with $b(v) = 1$ that we have to check for a common vertex.

The previous data reductions work by removing vertices (respectively, edges) from the graph. The following reduction modifies the structure of the hypergraph and postpones some decisions to a later point.

Reduction 3.4 (Weighted Edge Folding (WEF)) *Let $e \in E$ be a edge and $N = \mathcal{N}(e) \setminus \{e\}$ be the edges adjacent to e. Suppose the following holds:*

1. *Each edge in N is linked to e via a vertex v with capacity $b(v) = 1$,*
2. *N is independent, that is the vertices in all distinct $f, g \in N$ are disjoint,*
3. *The Neighborhood Removal is not applicable as $\omega(N) > \omega(e)$, but it holds $\omega(e) > \omega(N) - \min_{f \in N} \omega(f)$*

then we can "fold" e and N into a new edge e', inducing an altered hypergraph H'. The hypergraph H' contains a new edge e' instead of N and e with $\omega(e') := \omega(N) - \omega(e)$ and $e' = \bigcup_{f \in N} f$. Let $\mathcal{M}'_{opt}$ be the optimal solution for H'. The weight of the maximum b-matching in H is $\omega(\mathcal{M}'_{opt}) + \omega(e)$. If the matching $\mathcal{M}'_{opt}$ contains e' then N is contained in an optimal solution for H. Otherwise, e is contained in a maximum matching in H.

Proof: We first show, that either e or all edges in N are contained in a maximum b-matching $\mathcal{M}_{opt}$. Assumption 2 guarantees that all edges in N are possible candidates for $\mathcal{M}_{opt}$. Assumption 1 allows

Algorithm 2 Neighborhood Removal

```
procedure NR($H = (V, E, \omega), b$)
    $C \leftarrow \emptyset$
    for $v \in V$ do
        $checked \leftarrow 0$
        for $e \in E(v)$ ordered by weight desc do
            $checked \leftarrow checked + 1$
            if $checked > b(v)$ then
                break {only the $b(v)$ heaviest}
            if $e \in C$ then
                continue
            $C \leftarrow C \cup \{e\}$
            $\omega_d \leftarrow 0$
            for $w \in e$ do
                $\omega_d \leftarrow \omega_d + \text{nmax}(\hat{\omega}(E(w) \setminus \{e\}), b(w))$
                if $\omega_d > \omega(e)$ then
                    break {Must dominate}
            if $\omega_d \leq \omega(e)$ then
                $b'(v) := \begin{cases} b(v) - 1 & \text{if } v \in e \\ b(v) & \text{else} \end{cases}$
                $R \leftarrow \{e\} \cup blockedEdges(e)$
                yield $(H \setminus R, b'), \{e\}, \emptyset$
```

Algorithm 3 Weighted Isolated Edge Removal

```
procedure WIER($H = (V, E, \omega), b$)
    $C \leftarrow \emptyset$
    for $v \in V$ with $b(v) = 1$ do
        $e \leftarrow \text{argmax}_{e \in E(v)} \omega(e)$
        if $e \in C$ then
            continue
        $N_b \leftarrow \{\}$ {map for faster neighbor check}
        $N_l \leftarrow \emptyset$
        $count \leftarrow 0$
        $B \leftarrow True$
        for $w \in e$ do
            if $\omega(e) < \max_{f \in E(w)} \omega(f)$ then
                $B \leftarrow False$ {exit, edge too light}
                break
            for $f \in E(w)$ do
                $C \leftarrow C \cup \{f\}$
            if $b(w) = 1$ then
                for $f \in E(w)$ do
                    $N_b[f] += 2^{count}$ {binary encode}
                $count \leftarrow count + 1$
            else
                $N_l \leftarrow N_l \cup E(w)$
        $S \leftarrow \forall f \in N_l : N_b[f] > 0$ {Check $N_l \subseteq N_b$}
        for $f, g \in N_b$ do
            if $N_b[g] \& N_b[f] = 0$ then
                if $\nexists w \in g \cap f : b(w) = 1$ then
                    $B \leftarrow False$
                    break {no shared vertex with cap. 1}
        if $S \wedge B$ then
            yield $(H \setminus \{N_b\}, b), \{e\}, \emptyset$
        else
            $C \leftarrow C \cup \{e\}$ {Exclude at future vertices}
```

Algorithm 4 Weighted Edge Folding

```
procedure WEF($H = (V, E, \omega), b$)
    $C \leftarrow \emptyset$
    for $v \in V$ with $b(v) = 1$ and $|E(v)| = 2$ do
        for $e \in E(v)$ with $|e| = 2$ do
            if $e \in C$ then
                continue {skip, edge already checked}
            $C \leftarrow C \cup \{e\}$
            $c \leftarrow True$
            $N \leftarrow \emptyset$
            for $w \in e$ do
                if $|E(w)| > 2 \vee b(w) > 1$ then
                    $c \leftarrow False$ {edge not suitable}
                else
                    $N \leftarrow N \cup E(w) \setminus \{e\}$
            if $c \wedge N$ independent then
                if $\omega(N) > \omega(e) \wedge \max_{e_n \in N} \omega(e_n) \leq \omega(e)$ then
                    $e_n \leftarrow \bigcup_{e \in N} e$
                    $\omega' \leftarrow \omega$
                    $\omega'(e_n) \leftarrow \omega(N) - \omega(e)$
                    $E' \leftarrow E \setminus (N \cup \{e\}) \cup \{e_n\}$
                    yield $((V, E', \omega'), b), \emptyset, \{e\}$
```

Algorithm 5 Weighted Twin

```
procedure TwinReduction($H = (V, E, \omega), b$)
    $C \leftarrow \emptyset$
    for $v \in V$ with $b(v) = 1$ and $|E(v)| = 2$ do
        $X \leftarrow \emptyset$
        for $e \in E(v)$ do
            if $e \in C$ then
                continue {skip, already checked}
            $C \leftarrow C \cup \{e\}$
            $candidate \leftarrow True$
            $neighbors \leftarrow \emptyset$
            for $w \in e$ do
                if $|w| > 2 \vee b(w) > 1$ then
                    $candidate \leftarrow False$
                else
                    $neighbors \leftarrow neighbors \cup E(w) \setminus \{e\}$
            if $candidate$ then
                $X \leftarrow X \cup \{(e, neighbors)\}$
        for $\exists N, e_1 \neq e_2 : (e_1, N), (e_2, N) \in X$ do
            if $N$ is independent then
                $e_n \leftarrow \bigcup_{e \in N} e$
                $\omega' \leftarrow \omega$
                $\omega'(e_n) \leftarrow \omega(N) - \omega(\{e_1, e_2\})$
                $E' \leftarrow (E \setminus \{e_1, e_2\}) \cup \{e_n\}$
                $H' = (V, E', \omega')$
                if $\omega'(e') \geq \omega(N)$ then
                    yield NR($H', b$)
                else if $\omega'(e') > \omega(N) - \min_{e \in N} \omega(e)$ then
                    yield EdgeFolding($H', b$)
```

either e or any edge of N to be in a matching. Let $F = N \cap \mathcal{M}_{opt}$ be a part of an optimal solution, we show that $F = N$ or $e \in \mathcal{M}_{opt}$. We now assume that F is nonempty and a strict subset of N, that is $F \subsetneq N$. Due to Assumption 1 this implies $e \notin \mathcal{M}_{opt}$. Since F is a strict subset of N and we have $\omega(e) > \omega(N) - \min_{f \in N}\{\omega(f)\} \geq \omega(F)$ (Assumption 3), we could swap it for e in $\mathcal{M}_{opt}$ and gain a better result. This is a contradiction to $\mathcal{M}_{opt}$ being optimal and $F \subsetneq N$ being a strict subset. If none of the edges in N are part of $\mathcal{M}_{opt}$ e is free and we can include it in the matching.

The vertices of e' in H' correspond to those of N in H. The vertices of e in H are only contained in e' in H' without further neighbors. Therefore, if e' is not in $\mathcal{M}'_{opt}$ the edge e must be in an optimal solution for H and otherwise N is included in an optimal solution for H.

The formula for the weight is correct by the following case distinction. When e' is not contained in $\mathcal{M}'$, e is free in the corresponding matching $\mathcal{M}$ and must be included in the optimal matching for H. Otherwise the weight of $\mathcal{M}'$ contains $\omega(e')$ and thus the optimal solution in H has weight $\omega(M')+\omega(e) = \omega(\mathcal{M}' \setminus \{e'\})+\omega(e')+\omega(e) = \omega(\mathcal{M}' \setminus \{e'\})+\omega(N)-\omega(e)+\omega(e) = \omega(\mathcal{M}' \setminus \{e'\})+\omega(N)$. □

Algorithm 4 finds edges with two adjacent edges to fold in order to reduce the problem size. Only edges of size two are considered for complexity reasons. The complexity of this algorithm is $\mathcal{O}(\min(m,n)\Delta_E)$, because we have to check for $\mathcal{O}(\min(m,n))$ candidates if the two neighbors are independent which requires $\mathcal{O}(\Delta_E)$ checks. Without constraining the edge size, it would be $\mathcal{O}(\min(m,n)\Delta_V\Delta_E)$ since we would collect more neighbors. We collect the neighbors on the two vertices with capacity 1 and check if they are independent. If so, we merge the independent neighboring edges and replace the neighbors and e by this merged e_n with a new weight. Figure 2 shows a (sub-)hypergraph where this reduction is applicable.

The following data reduction groups non-adjacent edges with the same independent neighborhood together. Afterwards, we can directly check again if Neighborhood Removal (NR) or Weighted Edge Folding (WEF) applies.

Reduction 3.5 (Weighted Twin (WT)) *Suppose $e_1, e_2 \in E$ are non-adjacent. Let $L_i = \mathcal{N}(e_i) \setminus \{e_i\}$ be the set of neighboring edges that are linked with e_i via a vertex with capacity of 1. Assume each set L_i is independent, $L_1 = L_2$ and $\omega(\{e_1, e_2\}) > \omega(L_1) - \min_{f \in L_1} \omega(f)$ holds. Then, we can solve the problem on a modified hypergraph H' of H, replacing e_1, e_2 with an edge e' with $\omega(e') = \omega(e_1) + \omega(e_2)$ and $e' = e_1$.*

Proof: Since $L_1 = L_2$ and all edges in L_2 are linked via a capacity 1 vertex we do not need to include the vertices in e_2. Because any capacity constraint for an edge in L_2 at a vertex in e_2 is also present at a vertex in e_1. If $\omega(e') > \omega(L_1)$ the new edge e' is weighing more than all of its neighbors combined, satisfying the condition of the Neighborhood Removal (NR). Otherwise, $\omega(e') > \omega(L_1) - \min_{n \in L_1} \omega(n)$ still holds and the properties for the Weighted Edge Folding (WEF) are satisfied. Indeed, the neighbors L_1 and e_1 are linked, L_1 is independent and the weight inequality for Assumption 3 hold. □

Figure 2 (WT) shows two edges that can be folded in such a way. An algorithm for detecting twins is listed in Algorithm 5. The algorithm first identifies all possible candidates that have only degree two vertices. Afterwards, we identify twins and either apply the Neighborhood Removal and add them directly to the matching or apply the Edge Folding. In this case, they only dominate their neighborhood except for one edge, and we can merge the edges and assign a new weight to the new combined edge. In order to quickly find identical neighborhoods, we have to sort the candidates by neighborhood size. Each independence check requires $\mathcal{O}(\Delta_E\Delta_V)$ checks. The overall complexity

Algorithm 6 Weighted Domination

```
1: procedure WD($H = (V, E, \omega), b$)
2:   $C \leftarrow \emptyset$
3:   for $v \in V$ with $b(v) = 1$ do
4:     for $e_s \in E(v)$ ordered by weight desc. do
5:       if $e_s \in C$ then
6:         continue
7:       $C \leftarrow \{e_s\}$
8:       $D \leftarrow \emptyset$
9:       for $e \in E(v), \omega(e_s) \geq \omega(e)$ do
10:        if $|e| \leq |e_s| \wedge \text{hash}(e, e_s)$ then
11:          $D \leftarrow D \cup \{e\}$
12:          break {Stop collecting candidates}
13:      for $w \in e_s$ do
14:        $D \leftarrow \{c \in D \mid w \in c\}$
15:      $E' \leftarrow E \setminus D$
16:      return $((V, E', \omega), b), \emptyset, \emptyset$
```

of this algorithm is $\mathcal{O}(m\Delta_E\Delta_V + m \log m)$. The algorithm for Weighted Edge Folding (WEF) and Weighted Twin (WT) share common steps and could be combined.

The original domination proposed by Fomin et al. [26] for the maximum independent set reasoned that a vertex v having neighbors N_v is superfluous if there is a vertex w with subset neighbors $N_w \subset N_v$. We extend this idea to edges in a weighted hypergraph.

Reduction 3.6 (Weighted Domination (WD)) *Let $e, f \in E$ be two edges with $\omega(e) \geq \omega(f)$. Suppose e is a subset of f and there is a vertex $v \in e \subseteq f$ with capacity $b(v) = 1$. Then, the edge f can be removed from the hypergraph.*

Proof: Since there is one vertex v with $b(v) = 1$ in $e \cap f$ the edges e and f cannot be both in the maximum matching at the same time. Now assume an optimal solution $\mathcal{M}_{opt}$ containing f. Since e is a subset of f and it holds $\omega(e) \geq \omega(f)$, in any optimal solution e can replace f. Thus, f can be removed from the hypergraph. □

Figure 2 (WD) visualizes an example of this reduction, and in Algorithm 6 we show an implementation in pseudocode for finding a Weighted Domination. We iterate over each vertex and its incident edges in descending weight. We check if the following edges satisfy the natural size constraint and, using a simple hash function, check for the subset criterion. After collecting all candidates, we check which of these candidates are super sets and remove them from the graph. The complexity with the shown subset check is $\mathcal{O}(m(\Delta_E\Delta_V \log \Delta_V))$ if we do not have a hashing function. For each edge, we can collect at most $\Delta_V - 1$ candidates. For each candidate, we have to check up to Δ_E vertices of e that they are indeed incident requiring $\log \Delta_V$ comparisons if the list is sorted. By multiplying the id of vertices contained in an edge, storing these results in a wide integer and using the modulo operator to check for division without remainder, we can reduce the time complexity to $\mathcal{O}(m(\Delta_E + \Delta_V))$ in the ideal case. This requires a growable wide integer resulting in larger memory cost, so using a hash function, like multiplying only the k least significant bits of the ids, seems reasonable. If B is the width of the wide integer and we want to use the k least significant bits, we can hash all edges if $\Delta_E < (\frac{B}{k} \log(2) - 1)$ holds.

3.3 Initial Solutions

We now present our algorithms to compute initial solutions. Roughly speaking, we use a greedy algorithm that sorts the edges by a priority function and adds free edges in this order. Note that the most intuitive order of adding heavy edges first may yield poor results. This is because any edge may block a wide range of other edges from being added, for example, if the current edge has many vertices. Thus, other priority functions are necessary. The core idea of our algorithm is to assign each edge a positive priority value and then add edges greedily in descending order of their priority (edges with the highest priority are added first). To overcome the problems of the weight priority function explained above, we scale the weight function with the number of their incident vertices and with the capacity at each vertex of the edge. This ensures that we select edges first that a) have a high weight, b) do not block a lot of other edges, and c) have vertices with a lot of remaining capacity. We scale by capacity $h_{cap}(e) := \omega(e)\prod_{v\in e} b(v)$, inverse edge size $h_{pin}(e) := \frac{\omega(e)}{|e|}$, the combination of both $h_{pin,cap}(e) := \frac{\omega(e)}{|e|}\prod_{v\in e} b(v)$ and finally the capacity and inverse vertex degree $h_{scaled}(e) := \omega(e)\prod_{v\in e}\frac{b(v)}{|E(v)|}$. This results in four algorithms cap, pin, pin,cap and scaled, defined by the general framework and the respective objective function.

3.4 Local Search

We now give a brief overview over our local search algorithm based on swapping. More detailed pseudocodes for the algorithms can be found in Algorithm 7. A swap consists of removing one edge e in favor of two edges s_1, s_2 that become unblocked by the removal of e. A swap is feasible if the combined weight of s_1, s_2 is greater than the weight of e.

Swapping Algorithm. In our swapping algorithm, we are searching for a matched edge and two non-matched adjacent edges that are only blocked by the matched edge and can be admitted to the matching simultaneously without conflict.

$(1, 2)$**-Swaps.** For each edge in the matching e, we first collect all neighboring edges that satisfy the condition of being only blocked by e. In a second step, we identify a pair of edges that can be added without conflict when e is removed from the matching and also constitute an improvement. If we find such an edge pair, we include them in the matching and remove the edge e from it. After a successful swap we maximize, i. e., add all free edges to the solution.

Perturbation. The swapping algorithm ends up in a local optimum if executed repeatedly. Therefore, we propose to perturb the solution (forcing hyperedges into the solution) similarly to Andrade et al. [4] who do this for the independent set problem. The geometric distribution and the number of unmatched edges forced into the solution are directly adopted from Andrade et al. [4].

Iterated Local Search. The whole process is driven by the iterated local search. As long as the stopping criterion, in our case, a number of unsuccessful searches is not met, the current solution $\mathcal{M}$ gets perturbed and then exhaustively improved by $(1, 2)$-swaps. We call the obtained solution $\mathcal{M}_*$. A better solution automatically becomes the new starting point for the next iteration. Similarly to Andrade et al. [4], we allow with a low probability[2] a slightly worse solution to be accepted as starting point of our next local search.

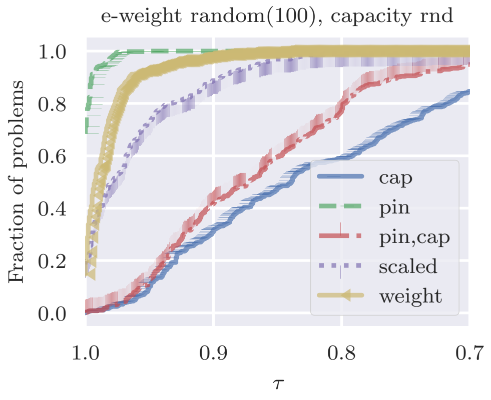
[2] $\frac{1}{(\omega(\mathcal{M}_{\text{best}}) - \omega(\mathcal{M}_*))(\omega(\mathcal{M}) - \omega(\mathcal{M}_*))}$

Algorithm 7 Local Search

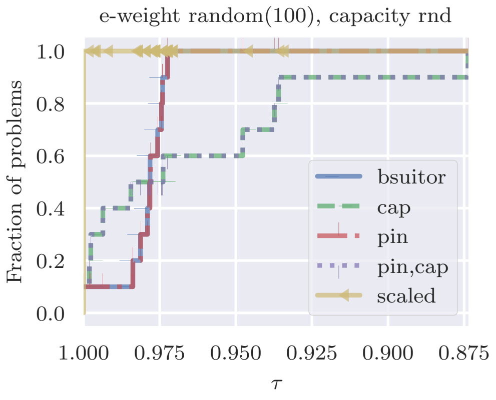
```
procedure OneTwoSwap($H = (V, E, \omega), \mathcal{M}$)
  for $c \in \mathcal{M}$ do
    $l \leftarrow \emptyset$
    for $p \in c$ do
      for $e \in E(p) \setminus \mathcal{M}$ do
        if $\mathit{blocked}(e, \mathcal{M}) \subseteq \mathit{blocked}(c, \mathcal{M})$ then
          $l \leftarrow l \cup \{e\}$
    if $|l| > 1$ then
      $\Phi(x) := \mathit{blocked}(x, \mathcal{M} \setminus \{c\} \cup \{x\})$
      if $\exists x, y \in l : \Phi(y) \cap \Phi(x) = \emptyset$
      and $\omega(x) + \omega(y) > \omega(c)$ then
        $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\} \cup \{x, y\}$
        $\mathcal{M} \leftarrow \text{maximize}(\mathcal{M})$
procedure ExhaustiveOneTwoSwap($H, \mathcal{M}$)
  while solution improved do
    OneTwoSwap($H, \mathcal{M}$)
procedure ILS($H = (V, E, \omega), \mathcal{M}$)
  $\mathcal{M}_{\text{best}} \leftarrow \mathcal{M}$
  while stopping criterion not met do
    $\mathcal{M}_* \leftarrow \text{Perturb}(H, \mathcal{M})$
    ExhaustiveOneTwoSwap($H, \mathcal{M}_*$)
    $\mathcal{P} \leftarrow \frac{1}{(\omega(\mathcal{M}_{\text{best}}) - \omega(\mathcal{M}_*))(\omega(\mathcal{M}) - \omega(\mathcal{M}_*))}$
    if $\omega(\mathcal{M}_*) > \omega(\mathcal{M})$ then
      $\mathcal{M} \leftarrow \mathcal{M}_*$
    else if $x \in \text{U}(0, 1) < \mathcal{P}$ then
      $\mathcal{M} \leftarrow \mathcal{M}_*$
    if $\omega(\mathcal{M}) > \omega(\mathcal{M}_{\text{best}})$ then
      $\mathcal{M}_{\text{best}} \leftarrow \mathcal{M}$
  return $\mathcal{M}_{\text{best}}$
```

4 Experimental Evaluation

Methodology. We implemented our algorithms and data structures in C++17. We compiled our program and all competitors using g++-12.1 with full optimization turned on (-O3 flag). In our experiments, we have used machines provided by a cluster, equipped with two 20-core Intel Xeon Gold 6230 processors running at 2.10 GHz and having a cache of 27.5 MB. Each machine was either equipped with 96 or 192 GB of main memory. In general, we perform experiments with uniform capacities of 1, 3, 5 and random capacity. Random in this context means, that we assign each vertex a capacity uniformly distributed between 1 and its degree. We run the experiments 10 times and take the arithmetic mean as result per instance. We use SCIP [8], one of the fastest open-source ILP-solvers, as a black box solver for ILPs. Deterministic experiments were only executed once iff the results (size, weight) and not the time was measured. The experiments were scheduled in parallel up to the numbers of physical cores of the machine, and the number of cores used by SCIP concurrently was limited to one. Each experiment run has a memory budget of 60 GB per instance and 140 GB in total. For comparison, we are using performance profiles as



(a) Quality results for the different proposed priority functions on 488 hypergraphs.



(b) Performance profile for $b(v)$ = rnd on graph instances selected by Khan et al. [43].

Figure 3

proposed by Dolan and Moré [17]. We plot which fraction of instances is solved by an algorithm to an objective value of at least $\tau\omega(\mathcal{M}^\star)$, $0 < \tau \leq 1$ and $\mathcal{M}^\star$ being the best matching reported by all heuristics. Thus, having a fraction near 1.0 for a high τ is considered a good performance because a high fraction of instances is then solved to near optimum. Similarly, we are using these profiles to compare execution duration of approaches. Here τ is greater than 1; for each instance, the time is marked as a multiple of the minimum time needed to solve the instance exactly by any approach.

Instances. We use a wide range of instances collected from various sources to evaluate our algorithms and to compare against state-of-the-art competitors. We use the M_{HG} dataset of *hypergraph instances*, provided by Gottesbüren et al. [32] containing *general hypergraphs*. It consists of 488 instances for four different use cases of hypergraphs, including 18 for circuit design (Ispd98,[3]), 10 routability-driven placement (Dac2012,[66]), 184 instances derived from general matrices from the Suite Sparse Collection (SPM,[16]) and 276 instances derived from SAT solving problems (SAT,[6]). When comparing against state-of-the-art b-matching algorithms in graphs, we use the 10 *graphs* from the Florida Sparse Matrix Collection by Davis and Hu [16], as proposed by Khan et al. [43] who also propose the algorithm that we compare against.

Edge Weights & Capacity. For general experiments, we assigned weights uniformly at random to the edges between 1 and 100. If we use random capacity, we sample for each vertex the capacity uniformly at random between one and its vertex degree.

Overview. The section is organized as follows. First, we compare our greedy priority functions from Section 3.3 on hypergraphs. Furthermore, we compare our results with those of the b-matching graph algorithm bSuitor by Khan et al. [43]. Then, we study the effectiveness of our exact data reductions on SCIP [8], a state-of-the-art open-source ILP solver. Finally, we benchmark our local search and compare against the competitors by Dufosse et al. [21] on 6-partite, 6-uniform hypergraphs.

Reduction	Constraint	Default
Neighborhood Removal (NR)	edge size	10
Weighted Isolated Edge Removal (WIER)	edge size	8
Weighted Isolated Edge Removal (WIER)	clique size	80
Weighted Domination (WD)	subedge size	6
Weighted Domination (WD)	candidate	6
Weighted Twin (WT)	edge size	4
All	iterations	10

Table 3: Parameter constraints of the reductions.

Priority Functions/Initial Solutions. The priority functions for this experiment are defined in Section 3.3. We test them on the general hypergraph data set. In Figure 3a, the results are shown for random capacity. For reference, we include the simple greedy weight function without any scaling as a baseline. For uniform capacity of 1, the pin,cap and pin compute the same best result since their objective function for this capacity is the same. Our other two proposals are nearly identical to the simple greedy weight algorithm. For a higher uniform capacity of 3 or 5 and random capacities, the quality of all of our proposals except for the pin is worse than the weight function. The pin function finds the best solution in around 70 to 80 % of the cases and requires a maximum $\tau = 0.9$ across all capacities. Since we only need to precompute the values of pin once, the running time of pin only slightly exceeds the running time of the simple weight algorithm, on average by 0.4% of the weight function, which we consider negligible.

We *conclude* that pin is a natural good choice to compute initial solutions on general hypergraphs. The other proposals are not viable since they yield worse results than the greedy weight algorithm.

Graph b-Matching. In order to show the versatility of our approaches, we also tested them on normal graphs and compared our results to those by Khan et al. [43]. We compiled and linked their program into our benchmark suite. The results of this first experiment are shown in Figure 3b. The competitor bsuitor by Khan et al. [43] is run in comparison to our approaches introduced in Section 3.3. Because of the fixed edge size of two in normal graphs and the capacity being static in all but one experiments, all but one function from Section 3.3 report the same results. Only the scaled approach differs by taking the number of incident edges into account and yielding a better result. We can report an improvement of up to 2 % over bsuitor on random capacity and up to 7 % on capacity 1. The running time of bsuitor on a single core in sequential cannot be matched by our algorithms on graphs, these are on average 3.19 times slower. This is however expected since our data structure supports hypergraphs and that introduces an additional overhead when only considering graphs.

Reductions and Speedup. In this experiment, we investigate how well our reductions can speedup solving b-matching problems with SCIP [8] as an open source black-box solver. As some of the data reductions can be expensive, we restrict some parameters in the search. The parameters we use can be found in Table 3. We empirically chose the parameters to balance between the reduction run time and success rate. The most sensitive tuning parameters were the ones for the Weighted Domination, since allowing bigger edges causes more checks to be needed. We apply

		Instances	# Edges (b.)	# Edges	# Vertices (b.)	# Vertices	Avg Speedup	Geom. Speedup
Type	$b(v)$							
ISPD98	1	15	66 149,73	63 802,47	61 454,80	51 575,80	1,48	1,43
	3	12	54 146,83	34 399,25	49 819,42	21 146,42	1,10	1,02
	5	16	84 121,75	27 443,81	78 844,19	15 681,44	1,71	1,65
	rnd	18	87 241,94	57 598,33	82 194,44	26 716,67	2,50	2,48
SPM	1	76	90 291,61	83 653,75	66 371,99	30 211,46	2,60	1,74
	3	85	106 133,60	88 644,16	93 424,06	42 218,27	1,64	1,21
	5	95	121 452,67	62 287,86	109 070,25	19 574,86	2,86	1,57
	rnd	149	101 263,43	86 516,18	91 509,01	53 173,65	1,52	1,29
dac2012	3	9	870 493,78	339 460,11	883 178,44	267 180,00	3,14	2,96
	5	10	912 788,00	58 781,30	924 053,70	24 115,70	7,12	6,66
	rnd	10	912 788,00	456 581,90	924 053,70	143 152,90	6,98	6,85
sat14	1	88	174 182,83	171 749,31	132 746,41	121 557,12	2,61	1,50
	3	154	378 729,73	210 010,55	980 506,36	83 418,93	3,71	2,18
	5	164	515 168,57	226 712,89	1 010 003,43	34 002,97	5,77	3,00
	rnd	155	281 539,58	211 308,96	302 720,74	76 921,45	2,38	2,05
all	1	179	129 511,16	125 299,78	98 590,93	76 909,13	2,51	1,59
	3	260	291 653,62	166 708,88	644 174,86	73 436,33	2,89	1,75
	5	285	373 682,38	154 825,18	654 400,99	27 818,10	4,62	2,40
	rnd	332	209 111,81	154 356,59	214 688,53	65 536,53	2,14	1,74

Table 4: Removal effectiveness and speed up broken down for classes of hypergraphs. We report the average edge and vertex count per class. Only exactly solvable instances are considered.

our search algorithms for the reductions up to ten times and pass the resulting core problem to SCIP. We then compare its total running time to the time it takes to solve the whole (hyper-)graph without applying reductions. The running time of the program with reductions includes the time to find and apply the data reductions. We set a timeout for the SCIP computations of 1 800 seconds and test it on uniform capacities of 1, 3 and 5, as well as on random capacity. We ran this experiment once to determine which instances are solvable in the given time with any of these capacities and repeated the experiment on the solvable subset of 395 hypergraphs ten times.
Only instances on which SCIP returns the optimum within the time limit are considered in the time performance profile for random capacity shown in Figure 4b, similar results were obtained for the static capacities. The performance profiles show that there are some instances that are strongly affected by the reductions. Some instances are so small that the time to search for reductions is bigger than the solving itself. Overall, 80% of instances benefit from running the reductions search and 40% have a speedup of at least factor 2. In Table 4 we report the average and geometric speedup sorted by hypergraph class. Our reductions achieve the best average speed up on the dac2012 instances of nearly 7 for random capacity. For these instances, the average edge count is halved by our reductions. The least speed up is observed on the ISPD98 instances for uniform capacities.

A plot of the removal effectiveness is shown in Figure 4a. On the y-axis, we display the relative edge count of the hypergraphs after applying our reductions. Similarly, on the x-axis, we plot the relative vertex count. Different shapes signify different capacities and colors are used for the different classes of hypergraphs. The majority of instances are located above the diagonal, meaning that the nodes are more reduced than the edges. There are two major clusters, one of not reducible instances and one for nearly completely reducible instances.
Figure 5 shows the relative running time of the reductions and effect on all 488 instances. The relative effect is computed on how many edges are included, excluded or deferred by the respective

$b(v)$	Sort	S,cap	S,pin	S,pin,cap	S,scaled	bweight
rnd	ISPD98	0.40	0.39	0.39	0.42	**0.38**
	SPM	1.58	1.27	1.63	1.84	**1.22**
	dac2012	2.51	2.37	2.62	2.56	**2.36**
	sat14	16.49	**15.26**	16.61	17.42	15.33
1	ISPD98	0.52	**0.44**	0.51	0.54	0.45
	SPM	2.96	**2.27**	2.82	2.92	2.27
	dac2012	4.84	4.31	4.85	5.22	**4.04**
	sat14	8.56	6.98	8.56	9.34	**6.84**
3	ISPD98	0.27	0.23	0.26	0.27	**0.22**
	SPM	1.05	0.67	1.08	1.27	**0.63**
	dac2012	1.98	1.75	1.98	2.21	**1.66**
	sat14	16.77	**14.26**	14.84	16.02	14.38
5	ISPD98	0.16	**0.16**	0.18	0.19	0.16
	SPM	1.02	0.67	1.05	1.23	**0.66**
	dac2012	0.78	**0.74**	0.76	0.79	0.76
	sat14	15.76	**14.13**	16.29	15.86	14.50

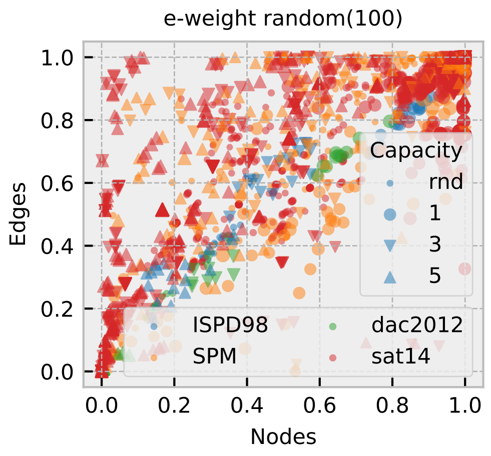
Table 5: Average running time in seconds for reductions and greedy initialization.

$b(v)$	ISPD98	SPM	dac2012	sat14
rnd	**0.36**	1.10	2.17	14.72
1	**0.42**	2.27	3.64	6.27
3	**0.19**	0.53	1.41	13.72
5	**0.14**	0.56	0.71	14.06

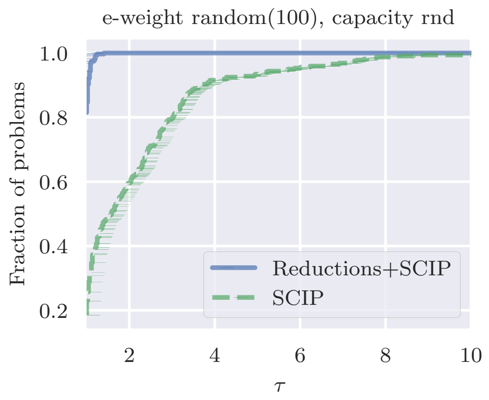
Table 6: Average running time in seconds for reductions only.

reductions. Table 5 and 6 report the running time in seconds for the reductions and the different greedy initialization functions. For all capacities the Abundant Vertices, Neighborhood Removal and Weighted Domination are the most significant reductions and take up the most time. The impact of the other reductions (Weighted Edge Folding, Weighted Isolated Edge Removal and Weighted Twin) is limited, but also the share of running time is small. For randomly assigned capacity the Neighborhood Removal reduction has the biggest impact, having over 50 % of the affected edges, while taking only a small fraction of running time. The Weighted Domination contributes the most affected edges for uniform capacity of 1, but also takes the most of the running time. The Weighted Isolated Edge Removal finds many candidates at capacity of 1 and uses around one quarter of the running time, finding only few affected edges. For higher or random capacity the Abundant Vertices reduction requires most of the running time, but also contributes more than half of the affected edges in the static capacity case.

Local Search. The iterated local search (ILS) approach from Section 3.4 is the main focus of this experiment. The ILS has only a threshold parameter k determining after how many fruitless perturbations and local searches, the search is canceled. The best result so far is returned. In our experiment, we used the best solution obtained by pin function and tried to improve it. We chose



(a) Relative removal effectiveness. Some instances can be completely reduced, while many are not reduced at all.



(b) Impact of reductions on optimum ILP solver SCIP on a subset of 395 hypegraphs.

Figure 4

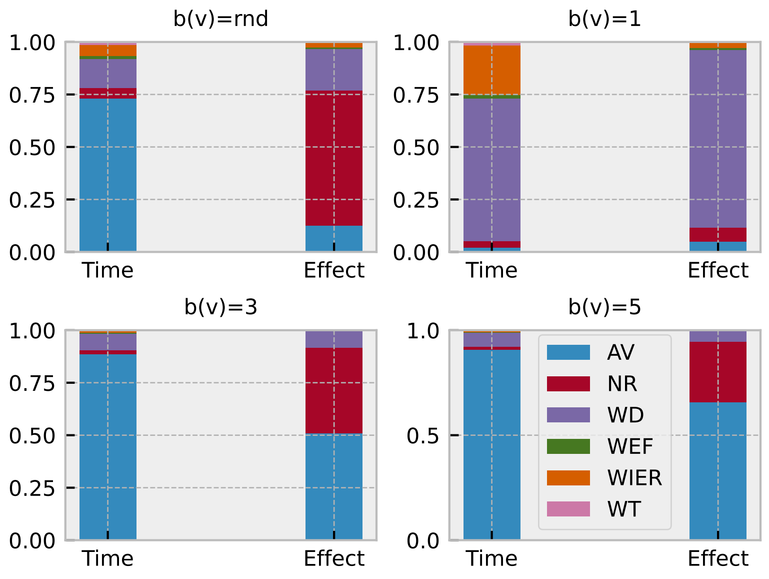
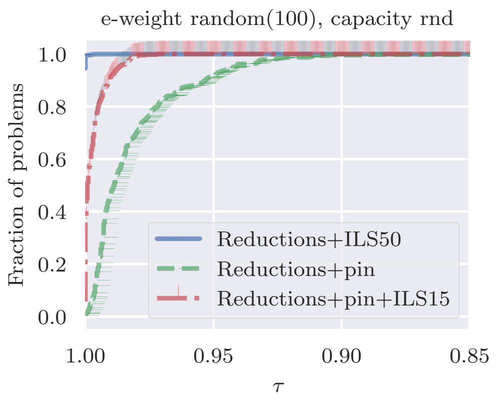
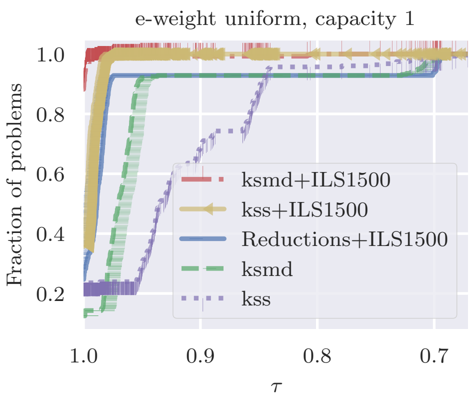


Figure 5: Relative running time and effect of reductions.



(a) Iterated Local Search improves the solution quality by up to 8%.



(b) Solution quality of Reductions+ILS, ksmd, kss by Dufosse et al. [21] and combination with ILS.

Figure 6

$k = 15$, as it showed a good balancing between running time and improvement on this collection of hypergraphs. We report an improvement of up to eight percent at max over all capacities. Running time is around ten times on average longer than computing the initial solution. This is however not surprising as the initial algorithm only sorts by the priority function and adds edges greedily. The performance plot for this experiment is shown in Figure 6a. For reference, we include $k = 50$ in the performance profile, which has an average running time tenfold in comparison to $k = 15$. Some instances are only improved by a negligible amount, while the best instances are improved by 8%. The average improvement is around 3%.

1-Matching on d-partite, d-uniform Hypergraphs. This experiment compares and combines our algorithm with those developed by Dufosse et al. [21] on a special class of hypergraphs with uniform edge weight and uniform structure. They implemented kss and ksmd, which employ two configurations of the Karp-Sipser approach. We are using kss with 20 scaling repetitions as proposed by the authors. We built and linked both algorithms into our benchmark program to ensure equal compile flags and settings. Figure 6b shows the performance profile for solution quality for $k = 1500$, when combining ILS with their approaches or our approaches as initial solution. The number of retries needed to see an improvement is significantly higher than the one for the non-uniform problem. This is due to the uniform structure of the problem, we need more tries in the perturbation phase to find an improvement. For a low $k = 50$, we can only report minimal improvements while running time stays closely to the base approaches. Our Reductions+ILS1500 approach, which combines the reductions, a greedy initial matching by pin, and iterated local search with $k = 1500$ has a better solution quality than the kss and ksmd approaches. Therefore, we want to combine both approaches.

Our data reductions cannot be combined with kss and ksmd since these algorithms require uniform edge size and our data reductions do not ensure this. We can considerably improve the quality of kss and ksmd by using our ILS as post-processing step. In case of ksmd, we can report a quality improvement of more than 30% on around 8% of instances, even surpassing the results of the Reductions+ILS1500 approach. Setting $k = 1500$ unsuccessful tries can result on a few instances in a hundredfold running time while consistently yielding better quality. The improvement for kss is more pronounced than for ksmd, which has a specific reduction rule for this kind of hypergraph. Improved quality comes at the expense of running time. The ksmd is the fastest approach and ten times faster than the kss approach. We need ten times more time to compute the solution than kss on average. Both, ksmd and kss are deterministic and thus can not be repeated multiple times to get a fairer comparison. In conclusion, the choice of k is sensitive for determining quality and running time and is dependent of the structure of the problem. Uniform weight problems require more tries than their non-uniform counterparts.

5 Conclusion

We developed a scalable algorithm for the general weighted b-matching problem in hypergraphs, utilizing novel data reductions. Our reductions can identify and incorporate optimum edges into a preliminary solution, reduce the problem size by combining the decision for multiple edges or, lastly, eliminate non-optimal edges from the input. We engineered new heuristic initial solution algorithms in a greedy framework and a local search framework to improve solutions. Experiments show that our data reductions scale well to large instances and accelerate state-of-the-art black-box solver. The new initial heuristic solutions by a greedy framework are up to 10 % better on general hypergraphs. On graphs these approaches also yield good results. The local search is able to improve solutions up to 30 % over recent results by Dufosse et al. [21] on some instances. Given the good results, we will release our software as an open source project. Future work includes parallelization of our algorithms, research in nonlinear optimization objectives, improving local search techniques, and application of our algorithms to related **NP**-hard problems.

References

[1] Faisal N. Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. Recent advances in practical data reduction. In Hannah Bast, Claudius Korzen, Ulrich Meyer, and Manuel Penschuck, editors, *Algorithms for Big Data: DFG Priority Program 1736*, pages 97–133. Springer Nature Switzerland, Cham, 2022. doi:10.1007/978-3-031-21534-6_6.

[2] Takuya Akiba and Yoichi Iwata. Branch-and-reduce exponential/fpt algorithms in practice: A case study of vertex cover. *Theor. Comput. Sci.*, 609:211–225, 2016. doi:10.1016/j.tcs.2015.09.023.

[3] Charles J. Alpert. The ispd98 circuit benchmark suite. In *Proceedings of the 1998 International Symposium on Physical Design*, ISPD '98, page 80–85, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/274535.274546.

[4] Diogo Vieira Andrade, Mauricio G. C. Resende, and Renato Fonseca F. Werneck. Fast local search for the maximum independent set problem. *J. Heuristics*, 18(4):525–547, 2012. doi:10.1007/s10732-012-9196-4.

[5] Georg Anegg, Haris Angelidakis, and Rico Zenklusen. Simpler and stronger approaches for non-uniform hypergraph matching and the Füredi, Kahn, and Seymour conjecture. In Hung Viet Le and Valerie King, editors, *4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11-12, 2021*, pages 196–203. SIAM, 2021. doi:10.1137/1.9781611976496.22.

[6] Anton Belov, Daniel Diepold, Marijn Heule, and Matti Järvisalo. The SAT competition 2014. http://www.satcompetition.org/2014/index.shtml, 2014.

[7] Piotr Berman. A d/2 approximation for maximum weight independent set in d-claw free graphs. In *Algorithm Theory - SWAT 2000*, pages 214–219, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. doi:10.1007/3-540-44985-X_19.

[8] Ksenia Bestuzheva, Mathieu Besançon, Weikun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros M. Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. Enabling research through the SCIP optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2):22:1–22:21, December 2023. doi:10.1145/3585516.

[9] M. Birn, V. Osipov, P. Sanders, C. Schulz, and N. Sitchinava. Efficient parallel and external matching. In *Proc. of Euro-Par 2013*, volume 8097 of *LNCS*, pages 659–670. Springer, 2013. URL: http://dx.doi.org/10.1007/978-3-642-40047-6_66, doi:10.1007/978-3-642-40047-6_66.

[10] Lijun Chang. Efficient maximum clique computation and enumeration over large sparse graphs. *VLDB J.*, 29(5):999–1022, 2020. doi:10.1007/s00778-020-00602-z.

[11] Lijun Chang, Wei Li, and Wenjie Zhang. Computing A near-maximum independent set in linear time by reducing-peeling. In Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pages 1181–1196. ACM, ACM, 2017. doi:10.1145/3035918.3035939.

[12] Alessio Conte, Donatella Firmani, Maurizio Patrignani, and Riccardo Torlone. A meta-algorithm for finding large k-plexes. *Knowl. Inf. Syst.*, 63(7):1745–1769, 2021. doi:10.1007/s10115-021-01570-8.

[13] Marek Cygan. Improved approximation for 3-dimensional matching via bounded pathwidth local search. In *FOCS*, pages 509–518. IEEE, 2013. doi:10.1109/FOCS.2013.61.

[14] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

[15] Marek Cygan, Fabrizio Grandoni, and Monaldo Mastrolilli. How to sell hyperedges: The hypermatching assignment problem. In *SODA*, pages 342–351. SIAM, 2013. doi:10.1137/1.9781611973105.25.

[16] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), Dec 2011. doi:10.1145/2049662.2049663.

[17] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002. doi:10.1007/s101070100263.

[18] Doratha E Drake and Stefan Hougardy. A simple approximation algorithm for the weighted matching problem. *Information Processing Letters*, 85(4):211–213, 2003. doi:10.1016/S0020-0190(02)00393-9.

[19] Andre Droschinsky, Petra Mutzel, and Erik Thordsen. Shrinking trees not blossoms: A recursive maximum matching approach. In Guy E. Blelloch and Irene Finocchi, editors, *Proc. of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020*, pages 146–160. SIAM, 2020. doi:10.1137/1.9781611976007.12.

[20] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *J. ACM*, 61(1), Jan 2014. doi:10.1145/2529989.

[21] Fanny Dufossé, Kamer Kaya, Ioannis Panagiotas, and Bora Uçar. Effective heuristics for matchings in hypergraphs. In *International Symposium on Experimental Algorithms*, pages 248–264. Springer, 2019. doi:10.1007/978-3-030-34029-2_17.

[22] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[23] Mourad El Ouali, Antje Fretwurst, and Anand Srivastav. Inapproximability of b-matching in k-uniform hypergraphs. In *WALCOM: Algorithms and Computation*, pages 57–69, Berlin, Heidelberg, 2011. Springer. doi:10.1007/978-3-642-19094-0_8.

[24] Mourad El Ouali and Gerold Jäger. The b-matching problem in hypergraphs: Hardness and approximability. In Guohui Lin, editor, *Combinatorial Optimization and Applications*, pages 200–211. Springer, 2012. doi:10.1007/978-3-642-31770-5_18.

[25] S. M. Ferdous, Alex Pothen, Arif Khan, Ajay Panyala, and Mahantesh Halappanavar. A parallel approximation algorithm for maximizing submodular b-matching. In *ACDA*, pages 45–56. SIAM, 2021. doi:10.1137/1.9781611976830.5.

[26] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5):25:1–25:32, 2009. doi:10.1145/1552285.1552286.

[27] Martin Fürer and Huiwen Yu. Approximating the k-set packing problem by local improvements. In *ISCO*, LNCS, pages 408–420, Germany, 2014. Springer. doi:10.1007/978-3-319-09174-7_35.

[28] Harold N Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC*, pages 448–456, 1983. doi:10.1145/800061.808776.

[29] Harold N Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320229.

[30] Alexander Gellner, Sebastian Lamm, Christian Schulz, Darren Strash, and Bogdán Zaválnij. Boosting data reduction for the maximum weight independent set problem using increasing transformations. In *ALENEX*, pages 128–142. SIAM, 2021. doi:10.1137/1.9781611976472.10.

[31] Giorgos Georgiadis and Marina Papatriantafilou. Overlays with preferences: Distributed, adaptive approximation algorithms for matching with preference lists. *Algorithms*, 6(4):824–856, 2013. doi:10.3390/a6040824.

[32] Lars Gottesbüren, Tobias Heuer, Nikolai Maas, Peter Sanders, and Sebastian Schlag. Scalable high-quality hypergraph partitioning. *CoRR*, abs/2303.17679, 2023. doi:10.48550/arXiv.2303.17679.

[33] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *ACM J. Exp. Algorithmics*, 13, Feb 2009. doi:10.1145/1412228.1412236.

[34] Martin Grötschel and Olaf Holland. Solving matching problems with linear programming. *Mathematical Programming*, 33:243–259, 1985. doi:10.1007/BF01584376.

[35] Jiewei Gu, Weiguo Zheng, Yuzheng Cai, and Peng Peng. Towards computing a near-maximum weighted independent set on massive graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 467–477, 2021. doi:10.1145/3447548.3467232.

[36] Mahantesh Halappanavar. *Algorithms for Vertex-Weighted Matching in Graphs.* PhD thesis, Old Dominion University, May 2009. URL: https://digitalcommons.odu.edu/computerscience_etds/57.

[37] Elad Hazan, Shmuel Safra, and Oded Schwartz. On the complexity of approximating k-set packing. *computational complexity*, 15(1):20–39, 2006. doi:10.1007/s00037-006-0205-6.

[38] Wei Hou, Limin Meng, Xuqing Ke, and Lingjun Zhong. Dynamic load balancing algorithm based on optimal matching of weighted bipartite graph. *IEEE Access*, 10:127225–127236, 2022. doi:10.1109/ACCESS.2022.3226885.

[39] Bert Huang and Tony Jebara. Fast b-matching via sufficient selection belief propagation. In *Intl. Conf. Artificial Intelligence and Statistics*, pages 361–369, 2011. URL: http://proceedings.mlr.press/v15/huang11a/huang11a.pdf.

[40] Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Mathematica*, 182(1):105 – 142, 1999. doi:10.1007/BF02392825.

[41] Hua Jiang, Dongming Zhu, Zhichao Xie, Shaowen Yao, and Zhang-Hua Fu. A new upper bound based on vertex partitioning for the maximum k-plex problem. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1689–1696. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track. doi:10.24963/ijcai.2021/233.

[42] Richard M. Karp and Michael Sipser. Maximum matching in sparse random graphs. In *Foundations of Computer Science, 1981. SFCS'81. 22nd Annual Symp. on*, pages 364–375. IEEE, 1981. doi:10.1109/SFCS.1981.21.

[43] Arif Khan, Alex Pothen, Md. Mostofa Ali Patwary, Nadathur Rajagopalan Satish, Narayanan Sundaram, Fredrik Manne, Mahantesh Halappanavar, and Pradeep Dubey. Efficient approximation algorithms for weighted b-matching. *Scientific Computing*, 2016. doi:10.1137/15M1026304.

[44] Viatcheslav Korenwein, André Nichterlein, Rolf Niedermeier, and Philipp Zschoche. Data reduction for maximum matching on real-world graphs: Theory and experiments. In *ESA*, volume 112 of *LIPIcs*, pages 53:1–53:13, 2018. doi:10.4230/LIPIcs.ESA.2018.53.

[45] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 508–520. Springer, 2009. doi:10.1007/978-3-642-02930-1_42.

[46] Christos Koufogiannakis and Neal E Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. In *DISC*, pages 221–238. Springer, 2009. doi:10.1007/978-3-642-04355-0_23.

[47] Piotr Krysta. Greedy approximation via duality for packing, combinatorial auctions and routing. In *MFCS*, pages 615–627. Springer, 2005. doi:10.1007/11549345_53.

[48] Sebastian Lamm, Peter Sanders, Christian Schulz, Darren Strash, and Renato F. Werneck. Finding near-optimal independent sets at scale. *J. of Heuristics*, 23(4):207–229, Aug 2017. doi:10.1007/s10732-017-9337-x.

[49] Sebastian Lamm, Christian Schulz, Darren Strash, Robert Williger, and Huashuo Zhang. Exactly solving the maximum weight independent set problem on large real-world graphs. In *2019 Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 144–158. SIAM, 2019. doi:10.1137/1.9781611975499.12.

[50] Jinkun Lin, Shaowei Cai, Chuan Luo, and Kaile Su. A reduction based method for coloring very large graphs. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 517–523, 2017. doi:10.24963/ijcai.2017/73.

[51] Fredrik Manne and Mahantesh Halappanavar. New effective multithreaded matching algorithms. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 519–528. IEEE, 2014. doi:10.1109/IPDPS.2014.61.

[52] Julián Mestre. Greedy in approximation algorithms. In *Algorithms–ESA 2006: 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006. Proceedings 14*, pages 528–539. Springer, 2006. doi:10.1007/11841036_48.

[53] Silvio Micali and Vijay V. Vazirani. An o(sqrt(|v|) |e|) algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.12.

[54] Matthias Müller-Hannemann and Alexander Schwartz. Implementing weighted b-matching algorithms: Insights from a computational study. *ACM J. Exp. Algorithmics*, 5:8, 2000. doi:10.1145/351827.384250.

[55] Meike Neuwohner. Passing the limits of pure local search for weighted k-set packing. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1090–1137. SIAM, 2023. doi:10.1137/1.9781611977554.ch41.

[56] David A. Papa and Igor L. Markov. Hypergraph partitioning and clustering. In *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007. doi:10.1201/9781420010749.ch61.

[57] Ojas Parekh and David Pritchard. Generalized hypergraph matching via iterated packing and local ratio. In Evripidis Bampis and Ola Svensson, editors, *Approximation and Online Algorithms - 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014, Revised Selected Papers*, volume 8952 of *Lecture Notes in Computer Science*, pages 207–223. Springer, Springer, 2014. URL: https://doi.org/10.1007/978-3-319-18263-6_18, doi:10.1007/978-3-319-18263-6_18.

[58] Marco Pavone, Amin Saberi, Maximilian Schiffer, and Matt Wu Tsao. Online hypergraph matching with delays. *Operations Research*, 70(4):2194–2212, 2022. doi:10.1287/opre.2022.2277.

[59] R. Preis. Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs. In *STACS*, volume 1563 of *LNCS*, pages 259–269. Springer, 1999. doi:10.1007/3-540-49116-3_24.

[60] S. Schlag, V. Henne, T. Heuer, H. Meyerhenke, P. Sanders, and C. Schulz. k-way hypergraph partitioning via n-level recursive bisection. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments, ALENEX*, pages 53–67, 2016. doi:10.1137/1.9781611974317.5.

[61] Richard Sinkhorn and Paul Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[62] Darren Strash. On the power of simple reductions for the maximum independent set problem. In *Intl. Computing and Combinatorics Conf.*, pages 345–356. Springer, 2016. doi:10.1007/978-3-319-42634-1_28.

[63] Darren Strash and Louise Thompson. *Effective Data Reduction for the Vertex Clique Cover Problem*, pages 41–53. SIAM, 2022. doi:10.1137/1.9781611977042.4.

[64] Theophile Thiery and Justin Ward. An improved approximation for maximum weighted k-set packing. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1138–1162. SIAM, 2023. doi:10.1137/1.9781611977554.ch42.

[65] Anurag Verma, Austin Buchanan, and Sergiy Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. *INFORMS Journal on Computing*, 27(1):164–177, 2015. doi:10.1287/ijoc.2014.0618.

[66] Natarajan Viswanathan, Charles J. Alpert, Cliff C. N. Sze, Zhuo Li, and Yaoguang Wei. The DAC 2012 routability-driven placement contest and benchmark suite. In Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun, editors, *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, DAC '12, pages 774–782, New York, NY, USA, 2012. ACM. doi:10.1145/2228360.2228500.

[67] Yiyuan Wang, Shaowei Cai, Shiwei Pan, Ximing Li, and Monghao Yin. Reduction and local search for weighted graph coloring problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2433–02441, Apr. 2020. doi:10.1609/aaai.v34i03.5624.