On an Ordering Problem in Weighted Hypergraphs

Peter Damaschke

[1] Department of Computer Science and Engineering,
Chalmers University, 41296 Göteborg, Sweden
[2] Fraunhofer-Chalmers Research Centre for Industrial Mathematics
ptr@chalmers.se

Abstract. We consider the problem of mapping the n vertices of an edge-weighted hypergraph to the points $1, \ldots, n$ on the real line, so as to minimize the weighted sum of the coordinates of right ends of all edges. This problem naturally appears in warehouse logistics: n shelves are arranged in one row, every shelf can host one type of items, the edges are sets of items requested together, their weights are the request frequencies, and items must be picked from the shelves and brought to a collection point at the left end of the row. The problem is to place all items so as to minimize the average length of the collection tours. It is NP-complete even for graphs, but it can be solved in $O^*(2^n)$ time by dynamic programming on subsets. In the present work we focus on hypergraphs with small connected components, which also has a practical motivation: Typical requests comprise related items from only one of many small disjoint groups. As a first result we solve, in polynomial time, an auxiliary problem with prescribed ordering in every component. For the unrestricted problem we conclude some worst-case time bounds that beat $O^*(2^n)$ for components of sizes up to 6. Some simple preprocessing can further reduce the time in many instances. Furthermore, the case of star graphs can be solved via bipartite matchings. Finally, there remain various interesting open problems.

Keywords: Hypergraph linear arrangement · Dynamic programming on subsets · Convex hull · Bipartite matching · Warehouse logistics

1 Introduction

Let $G = (V, E)$ be a given hypergraph, consisting of a set V of vertices and a set E of edges, which are non-empty subsets of V. (As this work mainly deals with hypergraphs, for brevity we prefer the term "edge" to "hyperedge".) Every edge $e \in E$ has a positive weight $w(e)$. For edges with only one vertex, such as $e = \{v\}$, we may simply write $w(v)$ instead of $w(e)$ or $w(\{v\})$, and we say that the vertex v has this weight, without risk of confusion. We will study the following arrangement problem. We first introduce it technically, because it is then easier to explain its motivation.

MinSumEnds

Given: a hypergraph $G = (V, E)$ with n vertices, and positive weights $w(e)$ of all edges $e \in E$.

Find: a bijective mapping π of V onto the set of integers $\{1, \ldots, n\}$ that minimizes the sum $\sum_{e \in E} w(e) \cdot \mu(e)$, where $\mu(e) := \max\{\pi(v) : v \in E\}$ for every edge $e \in E$.

Informally, our problem is to place the n vertices of V on n distinct points $1, \ldots, n$ on the real line, so as to minimize the weighted sum of the right ends of all edges. We can also view π as an ordering of V from left to right.

The objective function $\sum_{e \in E} w(e) \cdot \mu(e)$ can be rewritten as $\sum_{i=1}^{n} i \cdot L(i)$, where $L(i)$ denotes the total weight of all edges that end in point i, that is, $L(i) = \sum_{e:\, \mu(e)=i} w(e)$. For any ordering π we also define a function L_π on V by $L_\pi(v) := L(i)$ when $\pi(v) = i$. Hence the value $L_\pi(v)$ is the total weight of all edges having v as their rightmost vertex.

A hypergraph is called connected if, for any two vertices u and v, there exists a sequence $u = u_0, e_0, u_1, e_1, u_2, \ldots, u_{k-1}, e_{k-1}, u_k = v$ with $u_i \in e_i$ for all i with $0 \leq i \leq k-1$, and $u_i \in e_{i-1}$ for all i with $1 \leq i \leq k$. For brevity, connected components of a hypergraph are just called *components* in this paper.

The main motivation of MinSumEnds comes from efficient warehouse operations. In one scenario, n types of items shall be located in n equidistant shelves along a wall. At a collection point to the left of all shelves, a collector (robot or human worker) is waiting for requests. When a subset of items is requested, the collector must retrieve the requested items from the shelves and bring them to the collection point. Hence the length of the walk is twice the distance to the farthest requested item, whereas the positions of the other items are irrelevant, and so are the amounts of requested items of each type. The weighted hypergraph models the typical requests and their frequencies. More precisely, every edge is a set of (types of) items in a request, and its weight is proportional to the frequency of exactly this request. Thus, placing the items in the shalves so as to minimize the average walking distance leads to the MinSumEnds problem.

One may doubt that the problem arises in exactly this form in practice, as a single line of shelves is a special case, the problem may be intertwined with other types of constraints, data on frequencies may only be rough estimates, etc. However, combinations of workflow and layout planning in factory halls and warehouses are definitely a subject of industrial projects, where users want to optimize layouts for work sequences and vice versa, possibly in several iterations. Extracting basic combinatorial optimization problems and trying to understand their complexity is a meaningful activity accompanying the practical developments.

MinSumEnds is NP-complete even for graphs, i.e., the case when all edges have at most two vertices. On the positive side, it can be solved in $O^*(2^n)$ time by dynamic programming on subsets [4]. Naturally, we are interested in relevant special cases that can be solved faster. One such practical case appears when the

items are partitioned into small disjoint groups of related items, and only these items are typically requested together. In such cases, the sizes of the components of our hypergraphs are small integers, however, the frequencies of requests are still arbitrary numbers. Therefore we will study the complexity of MinSumEnds for weighted hypergraphs whose components have some limited fixed size k.

A problem being closely related to MinSumEnds has been studied much more extensively: The Minimum Linear Arrangement (MLA) problem asks to order the vertices of a graph or hypergraph so as to minimize the sum, over all edges, of the distances between the leftmost and rightmost vertex in the edge. Key results can be found, e.g., in [1–3, 5, 8–12]. (Here we do not summarize them all.) Apparently, the use of dynamic programming on subsets for that problem was first discovered in [3].

The vast majority of graph and hypergraph problems, including MLA, can be solved independently on the components. A remarkable issue it that this is no longer the case for MinSumEnds. Loosely speaking, since all vertices compete for the best positions, there is heavy interaction between the components, which makes the problem tricky even in hypergraphs with components of fixed size, which we mainly consider in this work.

Our contributions can be outlined as follows. Using some exchange arguments we solve, in polynomial time, a restricted version of MinSumEnds that we call MinSumEnds$<$. In that auxiliary problem, the vertices within every component must appear in some prescribed ordering (whereas all permutations are allowed in MinSumEnds). The MinSumEnds$<$ problem also has some nice geometric interpretation in terms of convex functions. With the help of MinSumEnds$<$ we obtain time bounds that beat the standard $O^*(2^n)$ bound in hypergraphs with components of at most 6 vertices. For concrete instances, the actual running times can be further improved by excluding several candidate orderings within the components, as candidates for optimal solutions have to pass some simple tests with linear inequalities in the given weights. Besides small components it is also worth considering structural restrictions. The case of star graphs can be solved via bipartite matchings, and this idea can be generalized. We conclude with various open problems.

2 An Exchange Property

The following lemma is simple, but it will be central to our approach.

Lemma 1. *Let π be an ordering, and let P and Q be the sets of vertices at points $i+1,\ldots,i+p$ and $i+p+1,\ldots,i+p+q$, respectively. Suppose that no $u \in P$, $v \in Q$, and $e \in E$ exist with $u, v \in e$. Then we have: If the inequality*

$$\sum_{u \in P} L_\pi(u)/p \geq \sum_{v \in Q} L_\pi(v)/q$$

is violated, then placing the vertices of Q at points $i+1,\ldots,i+q$ and the vertices of P at points $i+q+1,\ldots,i+q+p$ while preserving the orderings within P and Q, will make the objective smaller.

Proof. Let us swap P and Q. Since no edge contains vertices from both P and Q, every vertex $u \in P$ and $v \in Q$ is the rightmost vertex of the same edges as it was before the swap. Hence all $L_\pi(u)$ and $L_\pi(v)$ are preserved. Thus, the objective function changes by adding the amount $q \sum_{u \in P} L_\pi(u) - p \sum_{v \in Q} L_\pi(v)$. From this, the assertion obviously follows. □

In words, Lemma 1 says that neighbored and consecutive sets P and Q of sizes $p = |P|$ and $q = |Q|$, such that no edge contains vertices from both P and Q, can be swapped if the mentioned inequality is violated. Hence, this inequality must hold in any optimal ordering π. If the inequality is an equation, then P and Q can also be swapped without destroying optimality.

3 An Auxiliary Problem with Ordered Components

Now we consider an auxiliary problem named MinSumEnds<. It is defined exactly as MinSumEnds, but with the extra condition that the vertices within every component C of the hypergraph must appear in π in some prescribed ordering. That is, $\pi(v_1) < \ldots < \pi(v_k)$ is required, where $\{v_1, \ldots, v_k\}$ is the ordered vertex set of C. (We do not repeat the entire formal definition, as the objective is the same as in MinSumEnds.) Since the orderings of vertices in the components are given, the components are already part of the input. Of course, they must be the true components of the input hypergraph G, but consistency can be checked in linear time.

Within any component C of the input hypergraph G, we call a subset M of vertices of C a *module* if the vertices of M appear consecutively in the prescribed ordering of C, and they remain consecutive also in every optimal ordering π of our hypergraph G.

Note that, due to the last condition, the modules are not "obvious" from the input; below we will show how to compute them.

We define the *density* of any subset M (module or not) of the vertex set of C as $D(M) := \sum_{v \in M} L_\pi(v)/|M|$. Since the ordering of C is fixed, actually the values $L_\pi(v)$, $v \in M \subseteq C$, do not depend on π, therefore $D(M)$ is well defined and easy to compute from the input. For single vertices v we write $D(v)$ instead of $D(\{v\}) = L_\pi(v)$.

Lemma 2. *Let M and N be disjoint modules in C with the following properties: $\pi(u) < \pi(v)$ for all $u \in M$ and all $v \in N$, no other vertex of C is between M and N in the ordering, and $D(M) < D(N)$. Then $M \cup N$ is a module, too.*

Proof. Assume for contradiction that π is some optimal ordering where the set I of vertices between M and N is non-empty. By assumption, all vertices in I belong to other components than C. We can uniquely partition I into subsets called bags, where every bag is a maximal subset of vertices of I that are consecutive in π and belong to the same component. M and N are considered bags as well. Let J and J' denote any two neighbored bags in this ordering (J to the left of J'). By Lemma 1, if $D(J) < D(J')$ then we can swap them to make

the objective smaller, which contradicts the optimality of π. (Since J and J' are from different components, they have no common edges, hence they satisfy the assumptions of Lemma 1, and swapping also yields a valid solution, since the orderings in the components are not changed.) Hence the bags in the sequence from M to N have non-increasing densities. But this contradicts $D(M) < D(N)$. It follows $I = \emptyset$ in every optimal ordering π. By the definition of modules this implies that $M \cup N$ is a module. □

Whenever two modules M and N satisfy the assumptions of Lemma 2, we can *merge* them to the module $M \cup N$ according to the conclusion of Lemma 2. Consider the following process based on this observation:

We start from the sequence of the single vertices of C (which are, trivially, modules) in the prescribed ordering, and we merge two arbitrarily chosen neighbored modules that satisfy the assumptions of Lemma 2. This step is repeated as long as possible. The final result is a sequence of modules that we call *blocks*.

We claim that the sequence of blocks does not depend on the arbitrary choices, that is, the resulting blocks are uniquely determined by the densities $D(v)$ of all single vertices v. Below we give a proof that also yields a geometric characterization of blocks.

The idea is to represent any partitioning of the ordered vertex set $\{v_1, \ldots, v_k\}$ of C into modules as a function f on the interval $[0, k)$, defined as follows: For every module $M = \{v_i, \ldots, v_j\}$ in this partitioning, let $f(x) := D(M)$ for all $x \in [i-1, j)$. Furthermore, let $g(x) := \int_0^x f(t)\,dt$.

Note that g is a monotone increasing and piecewise linear continuous function, where the slopes equal the densities. The initial function f denoted f_0 has values $f_0(x) = D(v) = L_\pi(v_i)$ for all $x \in [i-1, i)$, and for all i. Let $g_0(x) := \int_0^x f_0(t)\,dt$.

Yet another technical definition makes the proof convenient: On the graph of g we *mark* all endpoints of the modules. Then, the effect of merging two modules M and N to the graph of g can be figuratively described as follows. Let p and q be the left and right endpoint, respectively, of the straight-line segment corresponding to M and N, and let r denote the point separating these segments. Remember that the slope increases in the point r. We unmark r and move the two segments upwards, thereby transforming them continuously into the straight-line segment connecting p and q, Note that only unmarked points move upwards to the new segment. With these notations and observations we can state:

Lemma 3. *The endpoints of the blocks are exactly those points $(x, g_0(x))$ with integer x that are on the upper convex hull h of the graph of the function g_0.*

Proof. In the following, two functions are said to be in $\leq$ relation if their function values are in $\leq$ relation for every argument.

Initially, the marked points on g_0 are all points $(x, g_0(x))$ with integer values x, because the modules we start from are all single vertices. Trivially we have $g_0 \leq h$. If $g \leq h$, and we replace two incident straight-line segments in the graph

of g with increasing slopes by one straight-line segment connecting the same endpoints, then this modified function g still satisfies $g \leq h$, since h is an upper convex hull (and hence a concave function).

As long as $g = h$ does not yet hold, another merge operation can be done, and the total number of merge operations is finite (actually, at most k). Thus we always eventually obtain $g = h$, regardless of the choice of merge operations in every step.

All marked points that were strictly below the graph of h got unmarked, since otherwise they could not have moved upwards to the graph of h. Furthermore, marked points on the graph of h never got unmarked. Thus, exactly those marked points that were already initially on the graph of h are still marked. Together this yields the assertion. □

Note that the blocks have non-increasing densities, in the prescribed ordering in the component C. We arrive at the complexity result for our auxiliary problem:

Theorem 1. MinSumEnds$<$ *can be solved in* $O(e+n\log n)$ *time, for arbitrary hypergraphs with* n *vertices, edges of total size* e*, and ordered components. Moreover, every sequence of the blocks from all components, sorted by non- increasing densities (where ties are broken arbitrarily), is an optimal ordering.*

Proof. First we compute all densities $D(v)$, $v \in V$, in $O(e)$ time. Then we compute the blocks in every component, by pairwise merging of modules, starting from the single vertices. The blocks are uniquely determined due to Lemma 3, and with some care this part can be implemented to run in $O(n)$ time in all components.

Recall that the blocks are modules. Hence, in an optimal ordering, every block is a consecutive set, and due to Lemma 1, also blocks from different components appear in non-increasing order of densities, where the order of blocks with equal densities is arbitrary. Thus, it only remains to sort the blocks by their densities and concatenate them. In the worst case of many small blocks this incurs a logarithmic factor. □

Due to the non-increasing densities we refer to the algorithm in Theorem 1 as the *sedimentation algorithm*.

The arbitrary tie breaking in Theorem 1 suggests to slightly re-define the notion of blocks as follows:

Whenever blocks from the same component have equal densities, we can place them consecutively and merge them to one block, without missing an optimal solution.

The advantage is that now the blocks from the same component appear with strictly decreasing densities. From now on we use the concept of *blocks* in this stricter sense, without risk of confusion.

4 Domination Relation

Next we apply the sedimentation algorithm for MinSumEnds$<$ to the solution of the original MinSumEnds problem. A very naive way would be to exhaustively try all combinations of orderings of the vertices in the components of the input hypergraph G, solve every case in polynomial time, and finally take the solution with the best objective value. In the following we try to reduce the large number of orderings to examine.

In an instance of MinSumEnds$<$, let $v_1, \ldots, v_k$ again denote the vertices of some component C in their prescribed ordering, that is, $\pi(v_1) < \ldots < \pi(v_k)$ is required. Recall that the densities $D(v_i)$ uniquely determine the blocks of C, and the $D(v_i)$ in turn depend only on this internal ordering of C. We abbreviate the densities by $d_i := D(v_i)$.

Now let $(d_1, \ldots, d_k)$ and $(d'_1, \ldots, d'_k)$ be two different sequences of densities (resulting from different possible orderings of C in an instance of MinSumEnds). We say that $(d_1, \ldots, d_k)$ *dominates* $(d'_1, \ldots, d'_k)$ if their prefix sums satisfy $\sum_{i=1}^{j} d_i \geq \sum_{i=1}^{j} d'_i$ for all $j = 1, \ldots, k$, and the inequality is strict for some index j. An equivalent characterization is that $(d_1, \ldots, d_k)$ is obtained from $(d'_1, \ldots, d'_k)$ by "moving some amounts" from some positions to other positions to the left, i.e., with smaller indices, while preserving the sum. We also say that the ordering of C with densities $(d_1, \ldots, d_k)$ dominates the ordering of C with densities $(d'_1, \ldots, d'_k)$. Finally, we call two orderings *equivalent* if they yield the same sequence of densities; note that they do not dominate each other.

From the objective function of MinSumEnds, the following is obvious:

Lemma 4. *In an optimal ordering π solving an instance of* MinSumEnds*, the ordering of the vertices of any component C is not dominated by any other ordering of the vertices of C.*

In simpler words, only non-dominated orderings of components may appear in an optimal solution π. Among equivalent orderings we need to consider only one, since the objective value solely depends on the densities. This suggest the following definition and observation:

A set of orderings of a component C is said to be a *candidate set* if every optimal solution to MinSumEnds uses one of these orderings of C, or an equivalent ordering. We refer to the orderings in a fixed candidate set as *candidate orderings*.

Clearly, some candidate set for C can be obtained as follows: Take the set of all orderings not dominated by others, but for any equivalent orderings, keep only one arbitrarily selected representative.

An obvious idea is now to compute candidate sets in advance, which reduces the total number of orderings to consider. We elaborate on this idea in the next section.

5 Hypergraphs With Small Components

One use of the concept of blocks and of the above results would be to simplify some proofs from [6] (and rewrite them from a more general perspective). In that work we considered the case of unweighted graphs. However, in the following we derive new results for another class of hypergraphs, namely such with components whose size is bounded by some constant, but where the weights are arbitrary positive numbers.

Let $\{u, v\}$ be the vertex set of some component where $w(u) \geq w(v)$. Then, obviously, the ordering (u, v) dominates (v, u), unless $w(u) = w(v)$, in which case the orderings are equivalent. Thus, in either case we have to consider only one candidate ordering, and since $e = O(n)$, Theorem 1 yields immediately:

Theorem 2. MinSumEnds *in weighted hypergraphs where all components have at most two vertices can be solved in* $O(n \log n)$ *time.*

Already components with three vertices turn out to be more tricky. However, we will obtain some upper bounds on the number of candidate orderings. We use the following convenient notation. Let $v_1, \ldots, v_k$ be the vertices of the considered component C. We abbreviate the weights by $w_i := w(v_i)$ and $w_{ij} := w(\{v_i, v_j\})$, and similarly, we use more subscripts for larger edges. Furthermore, we can assume $w_1 \leq \ldots \leq w_k$ by re-indexing.

Lemma 5. *Let* c_k *denote a number with the property that every component with* k *vertices admits a candidate set with at most* c_k *orderings. Then we have:*

$$c_2 = 1,\ c_3 \leq 2,\ c_4 \leq 5,\ c_5 \leq 16,\ c_6 \leq 62.$$

Moreover, for any fixed k *and given weights, these candidate sets can be identified in constant time.*

Proof. Before we stated Theorem 2 we have already shown $c_2 = 1$. Generalizing the *exchange argument* used there we see: Any ordering of C beginning with $(v_i, v_j, \ldots)$, where $i < j$, is dominated by or equivalent to the ordering $(v_j, v_i, \ldots)$ obtained by swapping the first two vertices.

Assume that some prefix of the ordering of C is already fixed. Let P denote the vertex set of this prefix. We consider the residual hypergraph defined as follows. Every vertex $v \in P$ becomes an edge whose weight is the sum of all original weights of edges with v as the rightmost vertex. Every subset $e \subseteq C \setminus P$ gets the weight $\sum_{Q \subseteq P} w(Q \cup e)$, where $w(.)$ denotes the original weights, and with the understanding that a non-existing edge is the same as an edge with zero weight. The rest of the hypergraph outside C is not affected. We remark that $C \setminus P$ in the residual hypergraph is not necessarily connected, but we will not make use of connectivity.

Since, in MinSumEnds with the restriction that the ordering of C begins with the assumed prefix P, the costs of all vertices in $C \setminus P$ depend only on the residual hypergraph, the exchange argument also applies to $C \setminus P$ and the

new weights. This allows us to bound the numbers c_k recursively as follows. The claimed constant time bound is trivial.

For $k = 3$, a candidate ordering starting with v_2 must continue with v_1, and any other candidate ordering starts with v_3, which yields $c_3 \leq 1+c_2 \leq 1+1 = 2$. For $k = 4$, a candidate ordering starting with v_2 must continue with v_1, and the other candidate orderings start with either v_3 or v_4, which yields $c_4 \leq c_2+2c_3 \leq 1 + 4 = 5$. For $k = 5$, a candidate ordering must start with one of (v_2, v_1), (v_3, v_1), (v_3, v_2), (v_4), (v_5), which yields $c_5 \leq 3c_3+2c_4 \leq 6+10 = 16$. For $k = 6$, a candidate ordering must start with one of (v_2, v_1), (v_3, v_1), (v_3, v_2), (v_4, v_1), (v_4, v_2), (v_4, v_3), (v_5), (v_6), which yields $c_6 \leq 6c_4 + 2c_5 \leq 30 + 32 = 62$. □

Now we can beat the standard $O^*(2^n)$ time bound that holds for arbitrary hypergraphs (via dynamic programming on subsets), in the case when all components are small. In order to focus on the interesting exponential part only, we use the O^* notation that suppresses polynomial factors.

Theorem 3. MinSumEnds *in weighted hypergraphs with n vertices, consisting only of components with at most k vertices, can be solved in $O^*(b_k^n)$ time, where $b_k = c_k^{1/k}$, in particular:*

$$b_3 \leq 1.26,\ b_4 \leq 1.5,\ b_5 \leq 1.7412,\ b_6 \leq 1.9895.$$

Proof. The numbers come from Lemma 5. In the special case when all components have exactly k vertices, we can apply the sedimentation algorithm to all $c_k^{n/k}$ combinations of candidate orderings of the components. The time bounds for the sedimentation algorithm remain valid also if all components have at most k vertices, by the monotonicity of the b_k and straightforward algebra. □

For $k > 6$, the time bounds obtained in this way would exceed $O^*(2^n)$. In order to avoid bases larger than 2 despite some large components we can, however, combine the benefits of small components with dynamic programming on subsets. The statement of the following result looks somewhat technical, but our aim was to make it as general as possible.

Theorem 4. *Let $C_1, \ldots, C_h$ be some components of a given hypergraph G, where C_i has k_i vertices. Suppose that we can, in polynomial time, compute for each component C_i a candidate set with $a_i < 2^{k_i}$ orderings. Then some optimal ordering of the given hypergraph can be computed in time $O^*(\prod_{i=1}^{h} a_i \cdot 2^{n-\sum_{i=1}^{h} k_i})$.*

Proof. First we compute the mentioned candidate sets and take all $\prod_{i=1}^{h} a_i$ combinations of the candidate orderings therein. For every such combination we apply Theorem 1 to compute an optimal ordering σ of $C_1 \cup \ldots \cup C_h$, using the candidate orderings as prescribed orderings within the components C_i.

We observe that the vertices of $C_1 \cup \ldots \cup C_h$ have the same ordering σ also within an optimal ordering of the entire hypergraph G. This "context-free" property holds since, by Theorem 1, an optimal ordering is characterized by

the blocks having non-increasing densities, and neither blocks nor their densities depend on the remainder R of G outside $C_1 \cup \ldots \cup C_h$.

To R we apply dynamic programming on subsets: We generate all possible ordered subsets R incrementally from left to right, insert the resulting blocks into σ, discard solutions that violate the property that also blocks in R have non-increasing densities, and most importantly, whenever two ordered subsets of R are identical as sets and they end at the same position in σ, we keep only one ordering with minimal cost until that position. Correctness and time bound are straightforward. □

Note that the time bound in Theorem 4 is never higher than $O^*(2^n)$, but it can be significantly smaller. Theorem 4 can be used, for instance, with $C_1, \ldots, C_h$ being the components with at most 6 vertices (due to Lemma 5), and Theorem 3 is the special case when larger components do not exist. But also large components might have much fewer than 2^{k_i} candidate orderings due to their specific edge weights.

6 Linear Inequalities Can Rule Out Candidate Orderings

Lemma 5 provides general upper bounds on the number of candidate orderings in components of sizes from 3 to 6. However, the given edge weights may rule out further candidate orderings, and thus some quick and simple preprocessing can make the main algorithms for MinSumEnds faster, not in the worst case but for many specific instances. In the following we illustrate these possibilities for size 3 only, but similar measures can be taken in larger components, too.

Remember from Section 5 that, in a component with three vertices with weights $w_1 \leq w_2 \leq w_3$, at most two candidate orderings exist: (v_2, v_1, v_3), and either (v_3, v_1, v_2) or (v_3, v_2, v_1). Assume that the candidate set is, in fact, of size 2, that is, the two sequences are neither equivalent nor is any of them dominated by another ordering. By the characterization of dominating sequences in Section 4, the resulting two sequences of densities of vertices must have the form $(a, b+c+d, e)$ and $(a+b, c, d+e)$, for some non-negative numbers a, b, c, d, e. (Actually, b and d are positive.) From these sequences we get the following necessary conditions for a candidate set of size 2:

$a = w_2$ and $a + b = w_3$, hence $b = w_3 - w_2$.
$e = w_3 + w_{13} + w_{23} + w_{123}$.
$b + c + d = w_1 + w_{12}$, hence $c + d = w_1 + w_2 + w_{12} - w_3$.
$w_3 + w_{23} \leq w_1 + w_{12}$, since otherwise (v_3, v_2, v_1) would dominate (v_2, v_1, v_3).

Case 1. The second candidate ordering is (v_3, v_1, v_2), since $w_2 + w_{23} \leq w_1 + w_{13}$. Then $c = w_1 + w_{13}$, hence $d = w_2 + w_{12} - w_3 - w_{13}$, thus also $w_3 + w_{13} < w_2 + w_{12}$.

Case 2. The second candidate ordering is (v_3, v_2, v_1), since $w_1 + w_{13} \leq w_2 + w_{23}$. Then $c = w_2 + w_{23}$, hence $d = w_1 + w_{12} - w_3 - w_{23}$, thus also $w_3 + w_{23} < w_1 + w_{12}$.

We conclude that the component has only one candidate ordering if some of the derived inequalities are violated.

Getting back to the case when two candidate orderings exist: Define x to be the distance (in π) between the first two vertices of the component, and define y similarly for the last two vertices. Then (v_2, v_1, v_3) is strictly cheaper than the other candidate ordering (v_3, v_i, v_j), $\{i, j\} = \{1, 2\}$, if and only if $b \cdot x < d \cdot y$. In particular, the ordering in the component depends only on the ratios of these distances. Furthermore, in any optimal ordering using (v_2, v_1, v_3), since $w_2 \leq w_3 + w_{23} \leq w_1 + w_{12}$, the vertices v_2 and v_1 are in the same block, thus $x = 1$ and therefore $y > b/d$.

By checking these inequalities we can, in concrete instances, efficiently rule out certain partial solutions before or during dynamic programming when applying the algorithm from Theorem 4, and thus speed up its execution.

7 The Star Graph

So far we have focused attention on hypergraphs with small components. Another meaningful direction is to consider hypergraphs with structural restrictions. Here we provide such an example.

A *star graph* is a graph with n vertices where one vertex called the *center* is joined by $n-1$ edges to all other vertices called *leaves*. As a practical motivation of MinSumEnds in the warehouse context, suppose that the center represents some main product, and the leaves represent optional accessoires only one of which may be chosen and used together with the main product. The main product as well as each accessoire may also be ordered separately. Hence all possible requests are the vertices and edges of a star graph.

We denote the center by c and the edge weights by $y(v) = w(\{c, v\})$. Recall that c and the leaves v also have vertex weights $w(c)$ and $w(v)$, respectively, as earlier.

Furthermore, let $B(n, m)$ denote a time bound of a minimum-weight perfect matching algorithm in bipartite graphs with n vertices on either side, and with m edges. For instance, we have $B(n, m) = O(n^2 \log n + nm)$ from [13].

Theorem 5. MinSumEnds *on star graphs with n vertices can be solved witin $O(n \cdot B(n, n^2))$ time.*

Proof. For all n possible positions $\pi(c)$ of the center c, we compute an optimal solution with this fixed $\pi(c)$, and finally we take the best of these n solutions. For any fixed $\pi(c)$ we proceed as follows.

We construct a complete bipartite graph $(L, M; F)$ with vertex sets L and M of size $n-1$, and edge set $F = L \times M$. The vertices in L represent the leaves of the star, and the vertices in M represent the positions $i \in \{1, \ldots, n\} \setminus \{\pi(c)\}$. The weight $z(v, i)$ of any edge $\{v, i\} \in F$ is the cost of placing the leaf v at position $\pi(v) = i$. It is specified by $z(v, i) := i \cdot w(v) + \pi(c) \cdot y(v)$ if $i < \pi(c)$, and $z(v, i) := i \cdot (w(v) + y(v))$ if $i > \pi(c)$. The total cost of a solution is obviously $\pi(c) \cdot w(c) + \sum_{v \in L} z(v, \pi(v))$. Since $\pi(c)$ is fixed, it only remains to compute a minimum-weight perfect matching in $(L, M; F)$. □

We conjecture that MinSumEnds on star graphs can be solved faster than by doing n unrelated computations of bipartite matchings, encouraged by the observation that the n bipartite graphs for the different positions $\pi(c)$ are only slight variations of each other, and that their edge weights are also far from being arbitrary, e.g., they have obvious monotonicty properties. It should be possible to take advantage of that. In fact, the problem looks quite similar to the one in [7] (which has applications in scheduling), but the structure of edge weights is somewhat more complicated in our case, as M has vertices of two different types, namely those before and after $\pi(c)$. We must leave the question open.

On another front, Theorem 5 can be generalized straightforwardly: In graphs with a vertex cover of small fixed size γ we can decide on the γ positions of its vertices and then compute bipartite matchings for the other vertices, as they form an independent set, and thus the costs of each vertex depends only on its own position. This yields a time bound $O(n^\gamma \cdot B(n, n^2))$.

However, we do not see a way to solve MinSumEnds on graphs with many such *simple* graphs as components, as we did with *small* components. (This would be interesting for the storage of several products with accessoires.)

8 More Open Problems

Driven by practical questions we have presented algorithms being faster than standard dynamic programming on subsets, but it remains the intriguing problem whether MinSumEnds is NP-complete on weighted hypergraphs with components of at most three vertices (or any other constant size limit). There might exist a polynomial-time reduction from a "number problem" like Partitioning, but we did not manage to establish one.

A potentially interesting combinatorial question related to small components is whether the bounds in Lemma 5 are already tight. One may either refine the recurive argument or construct examples that enforce the obtained numbers of candidate orderings.

Besides graphs with small vertex covers, it would be worthwhile to find other natural cases (e.g., limited integer edge costs, or components with other restrictions such as tree structures) that can be solved in polynomial time or by FPT algorithms, using the concepts developed here.

Acknowledgment

This work extends initial ideas from Pedram Shirmohammad's master's thesis [14] supervised by the author, and it is also inspired by collaboration with Raad Salman and Fredrik Ekstedt at the Fraunhofer-Chalmers Research Centre for Industrial Mathematics.

References

1. Ambühl, C., Mastrolilli, M., Svensson, O.: Inapproximability Results for Maximum Edge Biclique, Minimum Linear Arrangement, and Sparsest Cut. SIAM J. Comp. 40, 567–596 (2011)
2. Arora, S., Frieze, A., Kaplan, H.: A New Rounding Procedure for the Assignment Problem with Applications to Dense Graphs Arrangements. Math. Progr. 92, 1–36 (2002)
3. Bhasker, J., Sahni, S.: Optimal Linear Arrangement of Circuit Components. In: HICSS 1987, vol. 2, pp. 99–111 (1987)
4. Boysen, N., Stephan, K.: The Deterministic Product Location Problem under a Pick-by-Order Policy. Discr. Appl. Math. 161, 2862–2875 (2013)
5. Cohen, J., Fomin, F.V., Heggernes, P., Kratsch, D., Kucherov, G.: Optimal Linear Arrangement of Interval Graphs. In: Kralovic, R., Urzyczyn, P.(Eds.) MFCS 2006. LNCS, vol. 4162, pp. 267–279. Springer, Heidelberg (2006)
6. Damaschke, P.: Ordering a Sparse Graph to Minimize the Sum of Right Ends of Edges. In: Klasing, R., Gasieniec, L., Radzik, T. (Eds.) IWOCA 2020. LNCS, vol. 12126, pp. 224–236, Springer, Cham (2020)
7. Domanic, N.O., Lam, D.K., Plaxton, C.G.: Bipartite Matching with Linear Edge Weights. In: Hong, S.K. (Ed.) ISAAC 2016. LIPIcs, vol. 64, pp. 28:1–28:13. Dagstuhl (2016)
8. Eikel, M., Scheideler, C., Setzer, A.: Minimum Linear Arrangement of Series-Parallel Graphs. In: Bampis, E., Svensson, O. (Eds.) WAOA 2014. LNCS, vol. 8952, pp. 168–180, Springer, Heidelberg (2014)
9. Esteban, J.L., Ferrer-i-Cancho, R.: A Correction on Shiloach's Algorithm for Minimum Linear Arrangement of Trees. SIAM J. Comput. 46, 1146–1151 (2017)
10. Feige, U., Lee, J.R.: An Improved Approximation Ratio for the Minimum Linear Arrangement Problem. Info. Proc. Letters 101, 26–29 (2007)
11. Fellows, M.R., Hermelin, D., Rosamond, F.A., Shachnai, H.: Tractable Parameterizations for the Minimum Linear Arrangement Problem. ACM Trans. Comp. Theory 8, 6:1–6:12 (2016)
12. Fernau, H.: Parameterized Algorithmics for Linear Arrangement Problems. Discr. Appl. Math. 156, 3166–3177 (2008)
13. Fredman, M.L., Tarjan, R.E.: Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, J. ACM 34, 596–615 (1987)
14. Shirmohammad, P.: Linear Arrangements with Closeness Constraints, Master's thesis, Chalmers and Univ. of Gothenburg, 2020