Explanations for Combinatorial Optimization Problems

Martin Erwig* and Prashant Kumar
Oregon State University
{erwig*,kumarpra}@oregonstate.edu
EECS, Oregon State University, Corvallis, OR 97330,
Corresponding Author: Martin Erwig

Abstract

We introduce a representation for generating explanations for the outcomes of combinatorial optimization algorithms. The two key ideas are (A) to maintain fine-grained representations of the values manipulated by these algorithms and (B) to derive explanations from these representations through merge, filter, and aggregation operations. An explanation in our model presents essentially a high-level comparison of the solution to a problem with a hypothesized alternative, illuminating why the solution is better than the alternative. Our value representation results in explanations smaller than other dynamic program representations, such as traces. Based on a measure for the conciseness of explanations we demonstrate through a number of experiments that the explanations produced by our approach are small and scale well with problem size across a number of different applications.

Keywords: contrastive explanation, explanation simplification

1. Introduction

In this paper we introduce a representation for explanations of the results obtained by combinatorial optimization algorithms. Moreover, we provide various visualizations for these explanations. The basis for this explanation representation is the so-called *joint value decomposition*, which is the underlying structure for all the explanations that are generated. Our approach is based on retaining a granular representation of values that are otherwise aggregated during the computation performed by the algorithm. The explanations that are created from the granular representations can answer questions about why one result was obtained instead of another and therefore can increase the confidence in the correctness of program results.

Combinatorial optimization problems are discrete optimization problems with a finite number of solutions [38]. They have roots in combinatorics, operations research, and theoretical computer science. Combinatorial optimization problems have numerous real-life applications, and adding explanation techniques for them can have a far-reaching impact on improving the understanding and acceptance of computational systems.

For example, algorithms for finding the shortest paths [10] are used by hundreds of millions of people worldwide in applications such as Google Maps and vehicle navigation systems. Moreover, companies like Amazon use algorithms for vehicle routing problems [49] to find optimal routes for package deliveries. The importance of this

problem is highlighted by the *Amazon Last Mile Routing Research Challenge* [4], a collaboration with MIT that offered a total award of 175,000 dollars to improve the state of the art in the domain. Other common applications of combinatorial optimization include project scheduling, cash-flow management, message routing in communication systems, and traffic flow optimization. The last example is an instance of the maximum-flow problem [25], which is also used for determining the maximum steady-state flow in pipelines or electrical networks. Applications of matching problems [50] can be found in a range of scenarios, including matching roommates to hostels, matching pilots to compatible airplanes, scheduling airline crews for available flight legs, and assigning routes to bus drivers.

Given the predominant role that optimization algorithms play in our lives, we should be able to trust and understand the results they produce. Automation systems can earn their users' trust by *explaining* results because explanations give users confidence in the correctness and reliability of algorithm and computation processes [24]. Realizing the increasing role of algorithms in the lives of citizens, governments around the world are starting to enact laws providing citizens with a *right to explanation* for the algorithmic results impacting them [45]. The General Data Protection Regulation (GDPR) of the European Union [28] and the Digital Republic Act of France [48] are examples of such regulations. While explanation as a fundamental right is still in its infancy, we can expect more such regulations and requirements as part of the legal framework in the years to come.

When presented with a result by an algorithm, it is natural for users to ask "Why is the result X and not Y?". Such explanations are an instance of what the philosophy literature calls *contrastive explanations* [34, 35]. A contrastive explanation compares two specific phenomena, the actual result (called *fact* or *solution*) and a hypothetical alternative (called *foil*) and justifies "Why this [fact] rather than that [foil]?"[27]. A nice property of contrastive explanations is that the explanations generated may be tailored to different users who may be wondering about different aspects of a solution. Different (parts of) results may result in different foils and consequently in different and more fitting explanations. Alternatives (that is, foils) against which decisions are to be justified and explained are typically provided by users, but sometimes they can be anticipated, which means that comparative explanations can also be generated automatically.

While software has predominantly been judged according to *correctness* and *efficiency*, the *understandability* of the results delivered by computations should be also considered an important criterion, since programs may efficiently produce correct results that still leave users wondering whether or why they are correct. In previous work we have proposed the notion of *explanation-oriented programming* that has the explicit goal of creating programs that not only produce correct results but also explanations why the results are correct and thus can be trusted. In this context, we have explored representations that can add explanatory context to program results. These representations were created for specific domains, such as probabilistic computations [20, 21, 23], causality [22, 51], regular expressions [16], spreadsheets [13], decision making [18], and dynamic programming [17, 19]. Notably, all these approaches exploited in some way the structures of the specific domains to obtain effective explanatory representations.

In this paper, we present and explore a representation that is not tied to a specific domain and can work for a wide range of different applications, which provides software developers a generic go-to representation for creating more explanatory software. The work in the area of programming languages that is concerned with representation is often focused on either language constructs in the context of language design or data structures for efficient computations. This paper is focused on representations that can support the understandability of software.

The main contributions of this paper are the following.

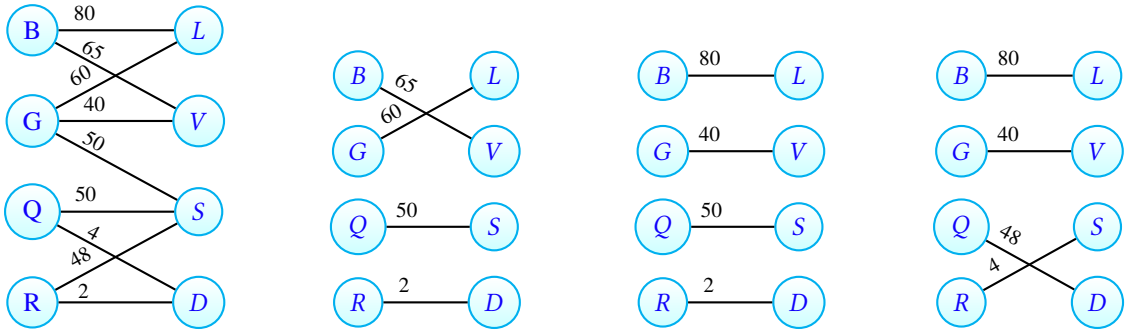
- A general representation for explaining the results of combinatorial optimization algorithms
- Tailored visualizations for explanations
- Experimental evidence of the efficacy of our representation
- A method for the automatic generation of foils for contrastive explanations

In Section 2, we examine a representation for formalizing combinatorial optimization problems and build upon it in Section 3 to incorporate explanations and their accompanying visualizations. The compositional structure of explanations facilitates the definition of operations for simplifying them. We introduce these operations in Section 4 together with a measure of explanation complexity. In Section 5, we show through experiments the scalability of our explanation approach for the maximum weighted bipartite matching problem. We then demonstrate in detail how our explanation representation can be applied to various other optimization problems, such as shortest paths (Sections 6), transportation problems (Section 7), and maximum-flow problems (Section 8). For each of these problems we also perform experiments to measure the efficacy of the various explanation simplification techniques. In Section 9, we show that our techniques work not only for the aforementioned computationally tractable problems but also for computationally hard (NP-hard) problems. We demonstrate with the travelling salesman problem. In Section 10, we survey the existing literature on explanations and compare them to our work. We provide conclusions in Section 11.

2. Combinatorial Optimization Problems

In the following, we adopt the definitions from the standard text by Ausiello et al. [6] (adjusting the notation slightly for convenience). Formally, a combinatorial optimization problem is defined as a four-tuple (I, f, m, μ) where

- I is a set of problem *instances*,
- $f : I \to 2^S$ is a function that maps instances to sets of *feasible solutions* from a set S,
- $m : I \times S \to \mathbb{R}$ is the *measure function* that assigns values to instance/solution pairs, and
- $\mu \in \{\min, \max\}$ is the *goal function*.



(a) Buyers and bids (b) Optimal Assignment O (c) Foil Assignment F (d) Foil Assignment F'

Figure 1: Optimal and alternative matching for a maximum bipartite matching problem.

The measure function is used to compare the various solutions for a problem instance. The goal is to find for some problem instance $p \in I$ the best feasible solution $s \in f(p)$, that is, we want to compute $\mu\{m(p,s) \mid s \in f(p)\}$.

As a concrete example, let's consider the maximum weighted bipartite matching problem. A weighted bipartite graph $G = (U \cup V, E, W)$ is a labeled graph whose vertices can be divided into two disjoint sets U and V such that each edge $(u,v) \in E$ connects a vertex $u \in U$ with a vertex $v \in V$ and has an associated weight $W(u,v) \in \mathbb{R}$ (that is, $W : E \to \mathbb{R}$). A matching M of graph G is a subset of E such that no two edges in M share a common vertex. The maximum (minimum) weighted bipartite matching of G is a matching whose sum of the weights of the edges is maximal (minimal).

Formally, the weighted maximum bipartite matching problem can be defined as a combinatorial optimization problem as follows (we use the notation $E_v^+ = \{w \mid (v,w) \in E\}$ to denote the set of successors of v in E and $E_w^- = \{v \mid (v,w) \in E\}$ for set of predecessors of w in E).

- $I = \{G = (U \cup V, E, W) \mid G \text{ is a weighted bipartite graph}\}$
- $f(G) = \{M \subseteq E \mid (\forall (u,v) \in M.M_v^+ = \{w\} \wedge M_w^- = \{v\}) \wedge |M| = \min\ \{|U|, |V|\}\}$
- $m(G,M) = \sum_{(u,v)\in M} W(u,v)$
- $\mu = \max$

To illustrate the need for explanations and suggest a possible approach for how to generate them, we use a concrete example in the following.

Assume that the National Basketball Association (NBA) was trying to add 4 new basketball franchises with teams for the cities of Las Vegas (L), Montreal (M), Seattle (S), and Dayton (D). After initial assessments 4 interested parties, Bayer AG (B), Glazer family (G), Qatar Investment Authority (Q), and Red Bull (R), are shortlisted for the bidding process. Each interested party can submit bids for multiple teams but can only get ownership of one team. The owners are not decided by the highest bid but rather the combination of bids that generates the highest total revenue for the NBA. This is an instance of the maximum weighted bipartite matching problem. The bipartite graph

showing the teams that each buyer is interested in owning is shown in Figure 1a: The presence of an edge between two nodes u and v suggests that the buyer u is interested in owning the team v for a price equal to the edge weight of (u, v) (in tens of millions of dollars).

The optimal assignment of buyers to respective teams is shown in Figure 1b where the total bid for all the teams is 177 (that is, 1.77 billion dollars). This result is surprising because Bayer bid 80 for the Las Vegas team, the highest bid for any team in the whole process, and was still not allocated that team. Either Bayer or someone in NBA might want to know why that is the case. In the next section we discuss different strategies to generate explanations for questions such as this and formalize the explanation process.

Next, we walk through a sequence of steps of a potential explanation, which helps to point out some aspects of a general approach to generating explanations. The overall strategy is to hypothesize an alternative assignment, guided by the user question/concern, and then show that it does not have a higher value. We can structure the explanation process into the following four steps.

Contrast Create a hypothesis about an alternative solution.

In our example, we assign the Las Vegas team to Bayer for a value of 80, that is, we make sure the edge (B, L) is contained in the alternative assignment.

Reconcile Since the previous step may have violated some constraint within the problem domain, we have to adapt the example data so that all constraints are again satisfied.

For our example, this means to remove the nodes B and L together with their incident edges from G (since they are already contained in the assignment), leaving a residual graph, say G'.

Remeasure Recompute the measurement for the alternative.

In the example, we compute the weighted maximum bipartite matching on G', which turns out to have a maximum value of 92, and add it to the value for the alternative solution, that is, 80, to obtain a total of 172.

Reaffirm Compare the measurement for the alternative with that of the solution to confirm the solution (and reject the alternative).

For our example, the highest possible revenue for the alternative assignment is 172, which is lower than the solution, thus demonstrating that the hypothesis should be rejected.

Figure 1c displays the optimal assignment after forcing the assignment of the Las Vegas team to Bayer. Following [36], we call such an alternative assignment a *foil assignment*. A foil assignment is created based on a specific question by a user who expected an alternative solution, especially one with a specific feature (here: a particular

edge). The notion of a foil has its roots in literature and is a powerful device for illustration through contrast [5].

Based on the foil assignment and its value, we can explain why the NBA did not assign the Las Vegas team to Bayer. The best alternative team assignment would have only produced a total revenue of 172 for the NBA, instead of the 177 it receives from the optimal assignment. This explanation, although correct, may not fully convince a user. They may want to know which specific assignments lead to the higher total value of the optimal assignment, but the current explanation does not address that. To provide a more fine-grained explanation, we can use the weights of individual assignments instead of just considering the total value. In the next section, we formalize this idea.

3. Explaining Algorithm Results

In this section, we extend the definition of combinatorial optimization problems to construct a framework for explaining algorithm results. To appreciate the scope and purpose of these explanations, it's important to distinguish between two different target groups for algorithm explanations.

3.1. Different Explanations for Different Stakeholders

There are various stakeholders involved in the development, implementation, and use of algorithms. While an algorithm designer is primarily concerned with the theoretical properties of an algorithm, such as correctness and time and space complexity, a programmer focuses on an accurate and properly documented implementation of the algorithm, resulting in code that is maintainable and thoroughly tested. The algorithm designer and the programmer are both concerned with how the algorithm/program works in general, for arbitrary input.

In contrast, an end user applies a program to specific input, representing a particular problem they want to solve, and they are typically only concerned with the output that solves the specific problem at hand.

Therefore, the notion of an *explanation* has different meanings to the different stakeholders. We can at least distinguish two different forms of explanations. First, a *generic* (or *static*) explanation describes how the algorithm or program works in general. Such an explanation may provide a general argument for how the algorithm works and why it is correct for arbitrary inputs. It may also justify the control structures that realize the algorithmic ideas and present general reasons for the algorithm's runtime complexity.

Second, a *specific* (or *dynamic* or *value-based*) explanation demonstrates why a specific output is the correct solution to the problem represented by the specific input. As the name suggests, concrete values play an important role in value-based explanations, and the challenge for an explanation approach is to select the "right" values that can convince a user of the correctness of the output. The focus of our work is on dynamic, value-based explanations.

3.2. Explainable Optimization Problems

To explain the result of combinatorial optimization algorithms with value explanations, we introduce the notion of *component-based* combinatorial optimization problem.

Specifically, we add a set of components C to allow the measure function to return a collection of values, indexed by the values of C, which are then aggregated using an operation $\oplus$ into the measurement that is to be optimized by μ. The operation $\oplus$ must be a binary operation of a group $(\mathbb{R}, \oplus, 0)$ where the set of real numbers $\mathbb{R}$ is the underlying set and 0 is the identity element. We also assume that $\ominus$ is the inverse operation of $\oplus$.

A *component-based combinatorial optimization problem* is given by a six-tuple $(I, f, \vec{m}, \mu, C, \oplus)$ where:

- I is a set of problem instances
- $f : I \to 2^S$ is a function that maps instances to sets of feasible solutions from a set S
- C is a set of *components* resulting from the optimization process
- $\vec{m} : I \times S \to (C \to \mathbb{R})$ is an *itemizing measure function* that maps instance/solution pairs to mappings from components to real numbers
- $\oplus$ is used to aggregate the decomposed values produced by $\vec{m}$
- $\mu \in \{\min, \max\}$ is the goal function

We call the mappings produced by $\vec{m}$ *value decompositions*, a concept borrowed from [17]. We can recover a "plain" (non-itemizing) measure function from the itemizing one by aggregating the values of all components into a single value. To this end we define for each value decomposition δ its aggregated value $\delta^{\oplus}$ as follows.

$$\delta^{\oplus} = \oplus_{v \in rng(\delta)} v$$

We can then define the *aggregated measure function* based on the aggregated values from the mappings of an itemizing measure function $\vec{m}$.

$$\hat{m} = \{(p, s) \mapsto \delta^{\oplus} \mid (p, s) \mapsto \delta \in \vec{m}\}$$

With this definition, $(I, f, \hat{m}, \mu)$ is a (plain) combinatorial optimization problem.

The formalization of the maximum weighted bipartite matching problem as a component-based problem keeps I, f, and μ the same as in the original version and just modifies the measure function as follows. We use the set of edges corresponding to the assignments of owners to the teams as components C and the weights associated with edges as values for the components. Component values are aggregated with addition.

- $C = E$
- $\vec{m}(G, M) = \{e \mapsto W(e) \mid e \in M\}$
- $\oplus = +$

The underlying group here is the group of real numbers with addition and 0 as the unit element. Note that while all the examples in this paper use this particular group, the current formalization allows for any appropriate aggregation

operation. For example, it can be used to explain the outputs of the Viterbi algorithm [26] where the underlying aggregation operation is multiplication, and the unit element is 1 ($\oplus = \times, 0 = 1$).

In our example, the optimal (O) and foil (F) assignments are given by the following two subsets of edges (see also Figures 1b and 1c). For clarity, we will also write B-V for the edge (B, V), and $\vec{m}_s$ for $\vec{m}(p, s)$ if the problem p is understood from context. (Similarly, we write $\hat{m}_s$ for $\hat{m}(p, s)$.) This will make some examples easier to read.

$$O = \{B\text{-}V, G\text{-}L, Q\text{-}S, R\text{-}D\}$$

$$F = \{B\text{-}L, G\text{-}V, Q\text{-}S, R\text{-}D\}$$

Accordingly, the itemizing and aggregated measure functions yield the following value decompositions and values for O and F, respectively.

$$\vec{m}_O = \{B\text{-}V \mapsto 65, G\text{-}L \mapsto 60, Q\text{-}S \mapsto 50, R\text{-}D \mapsto 2\} \qquad \hat{m}_O = 177$$

$$\vec{m}_F = \{B\text{-}L \mapsto 80, G\text{-}V \mapsto 40, Q\text{-}S \mapsto 50, R\text{-}D \mapsto 2\} \qquad \hat{m}_F = 172$$

We can notice that the difference between the two assignments O and F occur in the components (that is, edges) B-V and G-L in O and the alternatives B-L and G-V in F, which means that those edges are relevant to explaining why O and not F is the best assignment. Specifically, we can see that changing B-V to B-L in the foil assignment results in a gain of 15, but it also leads to the change of G-L to G-V, which incurs a cost of 20. Thus, this change leads to a net loss of 5 compared to the optimal assignment, making O the better choice.

How can we automate this kind of reasoning and effectively create a corresponding explanation? A simple approach is to compute the component-wise difference between the value decompositions of the solution and the foil, focusing on the components in which they differ. To achieve this, we define the *join* of two value decompositions δ and δ' as the union of three mappings: (i) the intersection of δ and δ', which gives the pair of values for the components that δ and δ' have in common, (ii) the components that are only contained in δ, and (iii) the components that are only contained in δ'. As auxiliary functions, we define the intersection and difference of value decompositions as follows.

$$\delta \cap \delta' = \{c \mapsto (v, v') \mid (c \mapsto v \in \delta \wedge c \mapsto v' \in \delta'\}$$

$$\delta - \delta' = \{c \mapsto v \in \delta \wedge c \notin dom(\delta')\}$$

Note that the operation $\cap$ is *not* commutative, as the order of the values (v, v') in the pairs is important. We use the notation $\dot{\delta}$ to denote a value decomposition δ as a foil. The values in the mapping inherit this notation to track their origin in joint decompositions and to distinguish between optimal and foil values. (In the visualizations of explanations, the solution and foil values will be represented in green and red, respectively.)

With these two operations we can define the join as a combination of three components.

$$\delta \bowtie \delta' = (\delta \cap \dot{\delta}') \cup (\delta - \dot{\delta}') \cup (\dot{\delta}' - \delta)$$

Please note that the join operation is not commutative. The foil is considered as the second argument, and its values are subtracted from the solution. In the case of our example, the join operation results in the following joint value decomposition.

$$\begin{aligned}\delta_{OF} &= \vec{m}_O \bowtie \vec{m}_F \\ &= \{Q\text{-}S \mapsto (50, \dot{50}), R\text{-}D \mapsto (2, \dot{2})\} \cup \{B\text{-}V \mapsto 65, G\text{-}L \mapsto 60\} \cup \{B\text{-}L \mapsto \dot{80}, G\text{-}V \mapsto \dot{40}\} \\ &= \{Q\text{-}S \mapsto (50, \dot{50}), R\text{-}D \mapsto (2, \dot{2}), B\text{-}V \mapsto 65, G\text{-}L \mapsto 60, B\text{-}L \mapsto \dot{80}, G\text{-}V \mapsto \dot{40}\}\end{aligned}$$

We can observe that the components of each value pair in the intersection are identical, indicating that any edge that is present in both the optimal and foil solutions contributes the same value to the overall result. In simpler terms, the edges in the intersection are not relevant for explaining the difference between the solutions. This is a property of the optimization problem at hand. We will encounter an example later on where the values in the solution and foil result in different values in the intersection.

The joint value decomposition allows us to produce several forms of contrastive explanations. Specifically, we can produce traces that illustrate how the advantage of the solution over the foil arises. Any such trace will contain the components of $\delta - \dot{\delta}'$ and $\dot{\delta}' - \delta$, but components of $\delta \cap \dot{\delta}'$ would be included only if the two value components differ. In that case we compute the difference between the two values and mark the result as counting toward the solution or the foil. This is done using a "foil-arithmetic" difference operation $\dot{\ominus}$, which is defined as follows.

$$x \mathbin{\dot{\ominus}} \dot{y} = \begin{cases} \dot{z} & \text{if } \mu(x, y) = y \\ z & \text{otherwise} \end{cases}$$

$$\textbf{where } z = |x \ominus y|$$

Note that the semantics of $\dot{\ominus}$ depends on the function μ that is defined as part of the component-based problem.

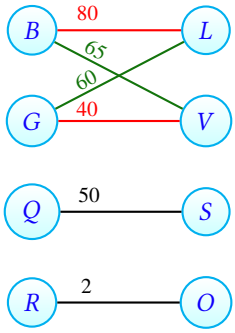
The focus on values that are relevant for explaining the difference between the solution and foil is captured in the concept of a *focused value decomposition*, which we denote by δ° and which is defined as follows.

$$\delta^\circ = \{c \mapsto v^\circ \mid c \mapsto v \in \delta \wedge v^\circ \neq 0\} \qquad\qquad v^\circ = \begin{cases} x \mathbin{\dot{\ominus}} \dot{y} & \text{if } v = (x, \dot{y}) \\ v & \text{otherwise} \end{cases}$$

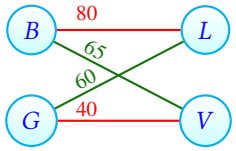
In our example we get the following focused value decomposition.

$$\delta^\circ_{OF} = \{B\text{-}V \mapsto 65, G\text{-}L \mapsto 60, B\text{-}L \mapsto \dot{80}, G\text{-}V \mapsto \dot{40}\}$$

There are several possibilities for grouping and ordering the components into an explanatory trace. One option is to show how the sum of solution values cannot be matched by the sum of foil values, which can be arithmetically expressed as the difference $65 + 60 - (80 + 40) = 125 - 120 = 5$. This expression shows that the combined values of the Las Vegas and Vancouver teams in the optimal and foil assignment result in a difference of 5 in favor of the



(a) Complete explanation



(b) Focused explanation

Figure 2: Explanations for the solution to the problem from Figure 1 with respect to the foil F.

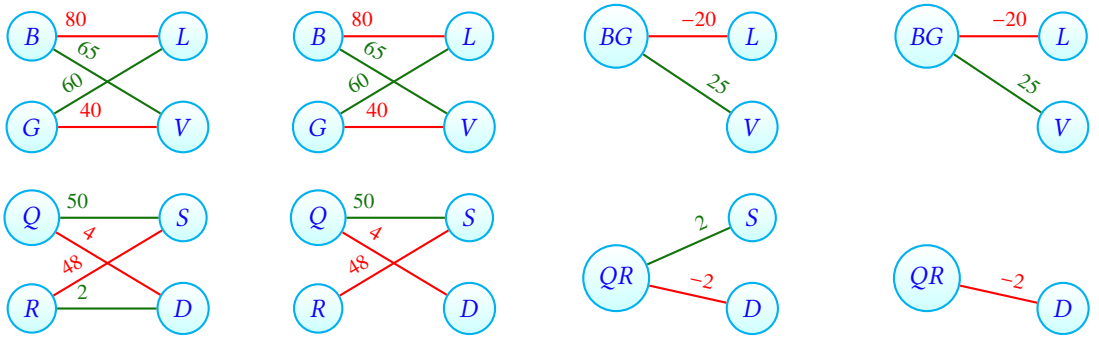
optimal solution, making it a better choice. Another option is to group components based on common elements (in this case, nodes) to emphasize the specific alternative (and its consequences) being questioned by the foil. For example, in our scenario, this would mean grouping the components B-L and G-L and the components B-V and G-V. This produces $60 - 80 + 65 - 40 = -20 + 25 = 5$, which illustrates how the apparent gain of the foil in one area is more than offset by a loss in another. This expression suggests that the NBA's loss of 20 from the Las Vegas team is more than compensated by the profit of 25 from the Vancouver team.

A visual representation of the joint value decomposition, shown in Figure 2, can serve as a summary for the contrastive explanation. The green edges and labels show the components and values of the solution. Similarly, the red edges and labels show the components and values of the foil. The black edges in Figure 2a represent the common components and values. We call this visualization a *complete explanation*. The black edges in a complete explanation can help set the context and show the extent of the changes required to consider when comparing the foil against the solution. But since the black edges don't contribute anything to the explanation of the value difference, users might prefer to see a more focused representation shown in Figure 2b that is based on the focused value decomposition and shows only the differences in components. We call this visualization a *focused explanation*.

4. Explanation Simplification

We have already seen that the common components in a joint value decomposition, which have the same values, do not contribute much to an explanation, even though they can provide context. Since conciseness is an important feature of explanations to be effective [36, 40, 47], we would like for explanations to contain as few details as possible, but as many details as needed. Focused explanations (see Figure 2b) are one way to support this goal. The effect of focused explanations will, of course, be noticeable only in sufficiently large problems. We will provide some data in this regard in Section 5.

Explanations that already focus on relevant differences can often be simplified further and deliver their explanatory contribution with still fewer details. One method, described in [17], is to show a smallest subset of the solution components whose sum exceeds the negative values of the foil components. Such a subset is called *minimal*



(a) Complete (b) MDS (c) Aggregated (d) Aggregated MDS

Figure 3: Complete, MDS, and aggregated explanations for the problem from Figure 1 with respect to the foil F'.

dominating set or *MDS*. We can define an MDS value decomposition as a refinement of a focused value decomposition as follows.[1]

First, we determine the negative (= foil) parts ($\delta^\bullet$) in a decomposition δ and then compute those positive subsets Δ (called *dominating sets*) whose value ($\hat{\Delta}$) exceeds the total of the negative components ($\hat{\delta^\bullet}$).

$$\delta^\bullet = \{c \mapsto \dot{v} \in \delta\} \qquad \delta^\Delta = \{\Delta \subseteq (\delta - \delta^\bullet) \mid \hat{\Delta} > \hat{\delta^\bullet}\}$$

Any element of δ^Δ with the fewest number of elements is a *minimal* dominating set.

$$\delta^{\text{MDS}} = \Delta \in \delta^\Delta \text{ such that } \forall \Delta' \in \delta^\Delta . |\Delta| \leq |\Delta'|$$

Since we compute MDSs from focused decompositions, they will always be a subset of those. However, only in some cases are MDSs proper subsets and thus smaller than focused decompositions. Despite its simplicity, focused decompositions are still a valuable form of explanation that can be used effectively.

In our small matching example, the MDS with regard to the foil F turns out to be the same as the focused value decomposition and cannot further simplify the explanation (which is already quite small). But we can illustrate the effect of MDSs with the alternative foil F' that is shown in Figure 1d: The value 50 of component Q-S in the solution results (together with the values 65 for component B-V and 60 for the component G-L) in a value of 175, which exceeds the total value of the alternative assignment, which is 172. The reduced joint decomposition with respect to this slightly smaller minimal dominating set is as follows.

$$\delta^{\text{MDS}}_{OF'} = \{B\text{-}V \mapsto 65, G\text{-}L \mapsto 60, Q\text{-}S \mapsto 50, B\text{-}L \mapsto \dot{8}0, G\text{-}V \mapsto \dot{4}0, Q\text{-}D \mapsto \dot{4}, R\text{-}S \mapsto \dot{4}8\}$$

This means it is sufficient to show just the edge Q-S to convincingly demonstrate that the solution is better than the foil F'. The edge R-D is not needed, and omitting it leads to a (ever so slightly) simplified explanation, shown in Figure 3b.

[1] As discussed in Section 10, we are using this method in a slightly different way than described in [17].

Yet another approach to simplifying the presentation of explanation information is to factor common parts of structured components. Specifically, if components are tuples, we can view C as $C = C_1 \times C_2$. Given an equivalence relation $\equiv \subseteq C_1 \times C_1$, we can group sets of C components by their common C_1 part, that is, for each equivalence class $[c] \in C_1$ we form the pair of $[c]$ and $[c]$'s successors in C. And with $\equiv \subseteq C_2 \times C_2$, we can define the grouping of common C_2 parts by pairing $[c]$ with its predecessors.

$$C\rangle^1_{\equiv} = \{([c], C^+_c) \mid [c] \in C_1/{\equiv}\}$$
$$C\rangle^2_{\equiv} = \{(C^-_c, [c]) \mid [c] \in C_2/{\equiv}\}$$

We can now extend the component grouping to value decompositions by aggregating the values of all successors or predecessors for an equivalence class.

For defining the aggregation we need a definition of an operation $\dot{\oplus}$, similar to that of $\dot{\ominus}$, that sums solution and foil values, compares them, and returns an aggregated value together with an indication whether it is advantageous for the solution or the foil. (Recall that $X^\bullet$ extracts all the foil elements from a mapping, and that $\hat{X}$ computes the sum of values in the range of X using $\oplus$.)

$$\dot{\oplus}S = \hat{S^\diamond} \,\dot{\ominus}\, \hat{S^\bullet} \qquad \textbf{where } S^\diamond = S - S^\bullet$$

With $\dot{\oplus}$ we can define the following two versions of *aggregated value decompositions* for a given value decomposition δ whose component type can be deconstructed into two parts $C = C_1 \times C_2$.

$$\delta\rangle^1_{\equiv} = \{(c, S) \mapsto \dot{\oplus}\{(c, c') \mapsto v \in \delta \mid c' \in S\} \mid (c, S) \in C\rangle^1_{\equiv}\}$$
$$\delta\rangle^2_{\equiv} = \{(S, c) \mapsto \dot{\oplus}\{(c', c) \mapsto v \in \delta \mid c' \in S\} \mid (S, c) \in C\rangle^2_{\equiv}\}$$

We can illustrate the idea of aggregated value decomposition with our example. Applying the definition of aggregated value decomposition (based on the equality of edges' target nodes) to the joint value decomposition $\delta_{OF'}$ yields the following.

$$\delta_{OF'}\rangle^2_{=} = \{BG\text{-}L \mapsto \dot{2}0, BG\text{-}V \mapsto 25, QR\text{-}S \mapsto 2, QR\text{-}D \mapsto \dot{2}\}$$

The aggregated explanation for F' says that the profits from the sale of the Vancouver and Seattle teams exceed the losses incurred from the sale of Las Vegas and the Dayton teams. The decomposition $\delta_{OF'}\rangle^2_{=}$ is visualized as an explanation in Figure 3c.

Since the MDS concept is generally applicable to any value decomposition, we can apply the idea also to aggregated value decompositions, and for F' this leads to a slightly smaller value decomposition and thus simplified explanation.

$$\delta^{\text{MDS}}_{OF'}\rangle^2_{=} = \{BG\text{-}L \mapsto \dot{2}0, BG\text{-}V \mapsto 25, QR\text{-}D \mapsto \dot{2}\}$$

The aggregated MDS explanation for F' says that the profit from the sale of the Vancouver team alone is enough to cover the entire loss. The decomposition $\delta^{\text{MDS}}_{OF'}\rangle^2_=$ is visualized as an explanation in Figure 3d.

Finally, we point out the relationship between component-based and plain combinatorial optimization problems. The goal of using itemizing measure functions is to retain details about the results that can be exploited for explanations. Thus, a component-based optimization problem should refine a plain optimization problem but otherwise have the same overall results. This requirement is captured in the following definition. We say that a component-based problem $(I, f, \vec{m}, \mu, C, \oplus)$ *refines* a problem (I, f, m', μ) if $\hat{m} = m'$. In what follows, we only consider component-based problems that refine the corresponding plain combinatorial optimization problems.

In addition to being correct and providing relevant information, an explanation should also be small enough so that it can be helpful and not overwhelm its client. A simple measure for the conciseness of an explanation is the proportion of its size compared to the size of a complete explanation, which is given by the number of components of the corresponding joint value decomposition. For value decompositions, we count pairs of different values as 2 and pairs of identical values as 1. Therefore, we define the conciseness of an explanation δX (for $X \in \{^\circ, ^{\text{MDS}}, \rangle^n_\equiv\}$) as follows.

$$C(\delta X) = 1 - \frac{|\delta X|}{|\delta|} \qquad |\delta| = \sum_{c \mapsto v \in \delta} |v| \qquad |v| = \begin{cases} 2 & \text{if } v = (x, y) \wedge x \neq y \\ 1 & \text{otherwise} \end{cases}$$

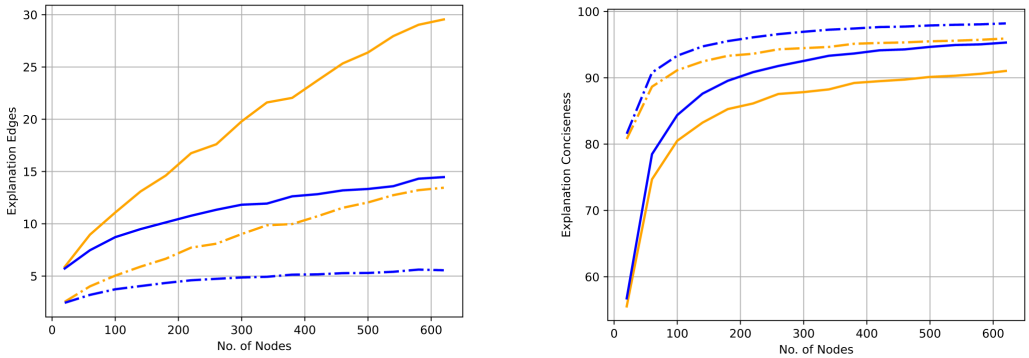
With this definition, higher values correspond to higher conciseness, and a perfectly concise explanation of 100% remains an ideal that can never be reached.

In our example, the size of the complete explanation is $|\delta_{OF}| = 6$. Thus, the focused explanation with size $|\delta^\circ_{OF}| = 4$ has a conciseness of $1 - 4/6 = 33\%$, which is not great, but is still an improvement over the complete explanation, which is with 0% not concise at all. For the alternative foil F', we get the conciseness values of $C(\delta^{\text{MDS}}_{OF'}) = C(\delta^\circ_{OF'}) = 1 - 7/8 = 12.5\%$, $C(\delta_{OF'}\rangle^2_=) = 1 - 4/8 = 50\%$, and $C(\delta^{\text{MDS}}_{OF'}\rangle^2_=) = 1 - 3/8 = 62.5\%$. The conciseness varies quite significantly in this example, but this can be due to the small graph size. To get a better sense of the explanation conciseness in larger, more realistic examples, we describe a corresponding analysis in the next section.

5. Explanation Conciseness for Weighted Bipartite Matching

To measure the effectiveness of the various explanation techniques for the maximum weighted bipartite matching problem, we perform a series of experiments. Due to a lack of test benchmarks, we generate bipartite graphs with varying number of nodes and edges and with different distributions of edge labels. We make the following assumptions in generating data for the experiments:

- We assume that the number of left nodes in the bipartite graph is the same as the number of right nodes. (We consider the size of the graph the total number of its nodes.)



(a) Size of explanations

(b) Explanation Conciseness

Figure 4: Effectiveness of various explanation techniques for maximum weighted bipartite matching problems. Explanation method: Focused, Aggregated / MDS (solid lines: density 1.0, dash-dot lines: average degree 6)

- We perform our experiments on both dense and sparse graphs. In the dense case each of the left nodes is connected to each of the right nodes. For the sparse case we use graphs with average degree of 6 for each node.
- The edge weights are randomly generated numbers in the range 30 - 2,000. The rationale for this range is as follows. A small and narrow range reduces the variability in the optimal and foil assignment, which could artificially inflate the efficacy of explanations. A range of 3.5 times the number of edges from a node provides enough variability in edge labels, while a larger range wouldn't add anything.

In addition to generating the input graphs, we must also generate the foils for comparison with the solution. A general strategy for creating foils is to generate k-optimal solutions for the problem at hand and pick the foils from that set ($k = 2$ will probably often suffice). The reason for considering only foils that are close to the optimal is that only such foils will be considered as serious alternatives and as such require an explanation or be good (that is, non-straw man) examples in support of the solution. We are using the k-optimal solution approach for the foil generation for the weighted bipartite matching (using the Lawler-Murty Procedure [33, 37], which runs in $O(kn^4)$ time) and for the shortest path problems (using Yen's algorithm [54], which runs in $O(kn^3)$ time).

Unfortunately, for some problems, algorithms to generate k-optimal solutions are unknown or ineffective in producing reasonable counter examples. In those cases, we employ an alternative approach that systematically varies parts of the solution to produce non-optimal solution candidates. A variation is adopted as a foil as long as it is close to the optimal solution. In this paper, we only consider an alternative as a foil if it deviates by less than 2% from the optimal solution.

To evaluate the conciseness of explanations for bipartite matching, we considered graphs with sizes ranging from 20 to 620 nodes (generated in increments of 40 nodes). For each graph size, we generated 900 pairs of optimal and close foil assignments, and calculated the size of complete explanations and the conciseness of each explanation. The overall value of these metrics for a graph size is the average of the 900 values computed for that size.

Figure 4a displays the number of explanation edges on the Y axis and the size of the bipartite graph on the X axis.

First, we note that in sparse graphs we need fewer explanation edges than in dense graphs. This is probably due to the fact that sparse graphs, having fewer edges, offer less variability between optimal and the foil matchings, which leads to an increase in the number of common edges and consequently a decrease in the number of edges required for an explanation. Next, we observe that the size of explanations grows almost linearly with the size of the graph for all explanation techniques. But it does so rather slowly. For example, even for a large bipartite graphs of size 620 nodes, we need on average only 30 edges for dense graphs and 14 edges for sparse graphs to explain a decision with a focused explanation. Aggregated explanations required on average just half the number of edges of focused explanations. This is expected because aggregated explanations have the effect on fusing multiple incident edges (from optimal and foil assignments). Since the number of incident edges on a node in focused explanation is 2, the reduction in the number of explanation edges by half is expected. The fact that we need on average just 14 edges for dense graphs and 5 edges for sparse graphs of size 620 nodes attests to their effectiveness. Aggregated MDS shows a small but inconsequential improvement over aggregated explanations.

The effort saved in explaining the results is illustrated in Figure 4b, which show how the conciseness of explanation steadily increases with the size of graphs. We observe that for a given graph size, the explanations for sparse graphs are more concise than that of dense graphs. The conciseness of all explanations increases in the shape of a hyperbola with the size of the bipartite graph.

It may seem unexpected that the conciseness of explanations increases with the size of the graph. Based solely on the formula for explanation conciseness, we might expect it to decrease. However, even though $|\delta X|$ increases with the size of the graph, the ratio $\frac{|\delta X|}{|\delta|}$ still decreases due to a relatively greater increase in the total number of edges, leading to a more concise explanation.

The fact that conciseness increases with problem size means that our explanation mechanisms scales effectively for weighted bipartite matching problems.

6. Explaining Shortest Paths

A shortest path between two points in a labeled graph is a path for which the sum of edge labels is minimal. Suppose $G = (V, E, W)$ is a labeled graph where V, E, and W are sets of nodes, edges, and edge labels, respectively. We assume that edge labels are positive. With $\Pi_G^{v,w}$ denoting the set of paths between nodes v and w in graph G, the formalization of the shortest path problem as a component-based problem is as follows.

- $I = \{(G, v, w) \mid G = (V, E, W) \text{ is a labeled graph and } \{v, w\} \subseteq V\}$
- $f(G, v, w) = \Pi_G^{v,w}$
- $C = E$
- $\vec{m}(G, P) = \{e \mapsto W(e) \mid e \in P\}$

- $\oplus = +$
- $\mu = \min$

Note that components and the decomposed measure function have the same structure as in the matching example, namely sets of edges mapped to values that are summed. Based on this formalization, we can explain solutions to the shortest path problems with contrastive explanations using value decompositions. As an example, consider the graph shown in Figure 5a,[2] and assume that we want to find the shortest path between the nodes D and H, which can be computed, for example, with Bellman-Ford's [10] or Dijkstra's [14] algorithm. The resulting shortest path is $O = D\text{-}A\text{-}B\text{-}E\text{-}H$ with a length of 29, as shown in Figure 5b. However, a user may recognize the alternative path $F = D\text{-}A\text{-}B\text{-}H$, shown in Figure 5c, as a more direct path and may wonder whether it may not be shorter. This situation can occur in real life, especially when a route with fewer distinct streets may falsely give the impression of being shorter than the optimal path.

To generate an explanation, we first compute the join of the two value decompositions for the two paths O and F.

$$\delta_{OF} = \{D\text{-}A \mapsto (8, \dot{8}), A\text{-}B \mapsto (5, \dot{5}), B\text{-}E \mapsto 9, E\text{-}H \mapsto 7, B\text{-}H \mapsto \dot{18}\}$$

This joint decomposition is the basis for the complete explanation shown in Figure 5d. As before, the edges that are common to solution and foil are shown in black, edges that are only in the shortest path are shown in green, and edges that are only in the foil path are shown in red.

We can obtain a focused explanation by omitting the edges that are common to both solution and foil. To this end we compute the focused value decomposition.

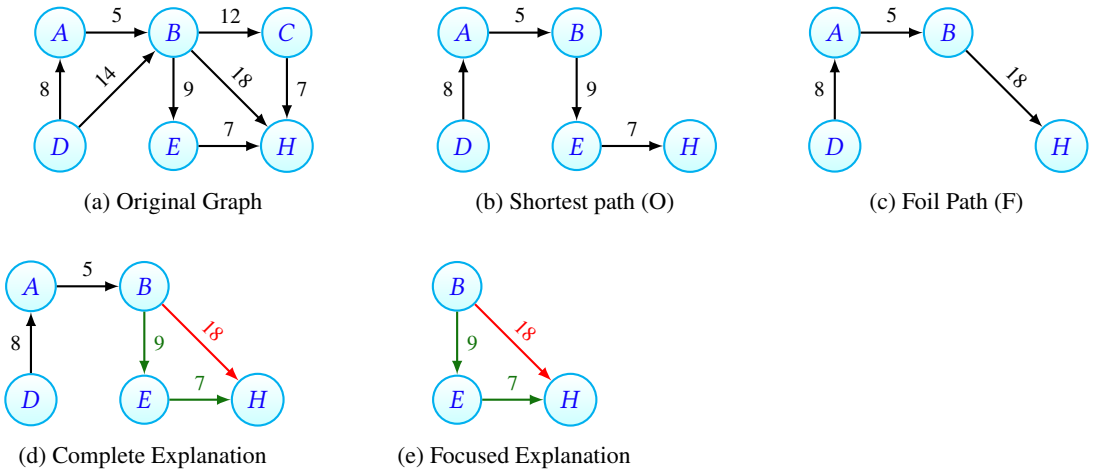
$$\delta^{\circ}_{OF} = \delta^{\text{MDS}}_{OF} = \{B\text{-}E \mapsto 9, E\text{-}H \mapsto 7, B\text{-}H \mapsto \dot{18}\}$$

The corresponding focused explanation is shown in Figure 5e. This concisely shows that the green (optimal) edges in the visualization have a total length of $9 + 7 = 16$, which is less than the length of the red (foil) edge of the foil, thereby making the optimal path shorter. In this (rather small) example, the MDS explanation produces the same result as the focused explanation.

In our example the size of the complete explanation is $|\delta_{OF}| = 5$, which means the focused explanation with size $|\delta^{\circ}_{OF}| = 3$ has a conciseness of $C(\delta^{\circ}_{OF}) = 1 - 3/5 = 40\%$, which is good, but not a phenomenal gain. However, as the evaluation shows, the effect of focusing can lead to significant increases in conciseness for larger graphs and paths.

Fortunately, for evaluating the shortest path and other traffic network problems real-world data sets are available. We use the road network data for the cities of Rome, Italy (with about 3,300 nodes and 4,800 edges), New York City (264,000 nodes and 365,000 edges), the San Francisco Bay Area (321,000 nodes and 397,000 edges), and the state of

[2] We use the node name H instead of F to avoid confusion with F, which is used as a name for foils. (Moreover, we don't use G, since it denotes the graph.)



(a) Original Graph (b) Shortest path (O) (c) Foil Path (F)

(d) Complete Explanation (e) Focused Explanation

Figure 5: Explanations of Shortest Path Results

Colorado (432,0000 nodes and 517,000 edges) where the nodes are important points in the road network and the edge lengths are the distances between points. These graphs are a part of the evaluation data set provided by the 9th DIMACS Implementation Challenge [15], held in 2005-2006, aimed at creating a reproducible picture of the state-of-the art in the area of shortest path algorithms.

For the evaluation of explanation conciseness, we randomly sample two nodes from the graph, a start node s and a target node t, with a total of approximately 225,000 samples in each case. To keep the evaluation realistic, we make some additional assumptions. For example, we only consider paths with at least 20 edges to ensure that we consider sufficiently complex paths that might prompt requests for explanations from users. We also limit the upper number of edges in considered paths to 80, since those are less frequently computed. In addition, we assume that the foil is the second-shortest path between the nodes s and t.

The table in Figure 6a summarizes the graph data, displaying the average number of edges in optimal and foil paths for different limits on the path length. It also shows the number of edges shared between the solution and foil, as well as the number of unique edges. We can observe that the solution and foil paths often share a significant number of edges (around 80-90%), which should result in a high level of explanation conciseness.

Figure 6b shows how the size of focused explanations seems to grow asymptotically to a small constant that depends on the size of the graph. Interestingly, while both graph size and average path length are larger for the Colorado than for the Bay Area graph, the size of explanations tend to be actually smaller. This may be due to the irregular shape of the Bay Area graph. As the size of MDS explanations for shortest paths is always around 85-90% of the size of focused explanations, the corresponding plots have been omitted to avoid overcrowding of figures.

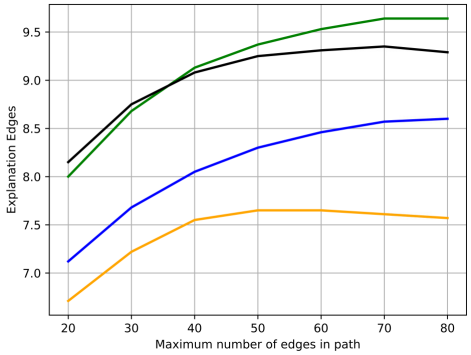
Figure 6c shows how the conciseness of focused explanations grows steadily in each graph from 55-60% to 75-80%, a trend to be expected from the explanation and graph sizes.

An important insight gained from this experiment is that focused explanations seem to scale quite well with the problem size, although not as dramatically as in the case for graph matching. Moreover, the fact that we can achieve

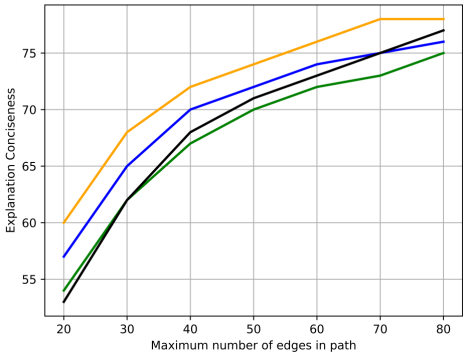
Network	# Nodes	# Edges
Rome	3.3K	4.8K
New York	264K	365K
Bay Area	321K	397K
Colorado	432K	517K

Network	Limit	Avg. Edges	Shared	Different
Rome	80	37.35	33.56	3.79
New York	80	37.28	32.98	4.30
Bay Area	80	38.58	33.76	4.83
Colorado	80	41.83	37.19	4.64
Rome	60	32.46	28.64	3.82
New York	60	31.93	27.70	4.23
Bay Area	60	32.49	27.73	4.76
Colorado	60	34.12	29.46	4.65
Rome	40	24.55	20.77	3.78
New York	40	24.10	20.07	4.03
Bay Area	40	24.46	19.90	4.56
Colorado	40	24.83	20.29	4.54
Rome	20	13.75	10.40	3.35
New York	20	13.29	9.73	3.56
Bay Area	20	13.57	9.57	4.00
Colorado	20	13.47	9.40	4.07

(a) Statistics of shortest paths of different lengths in different networks



(b) Size of focused explanations



(c) Explanation Conciseness

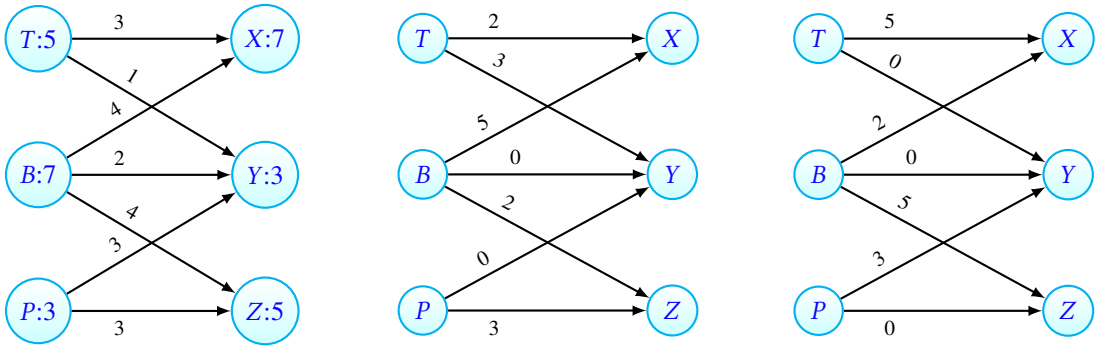
Figure 6: Conciseness of different explanation approaches for shortest paths of different maximum lengths.
Network: Rome, New York City, SF Bay Area, Colorado

good explanation conciseness for various large real-world graphs suggests that our explanation approach is effective in practice.

7. Transportation Problems

Consider the bipartite graph in Figure 7a. The set of nodes on the left side represent locations of factories for a certain product (here: Toronto (T), Boston (B), and Philadelphia (P)). These nodes are called *source nodes*. The positive number associated with a node label represents the capacity of the factories in that city. The nodes on the right side represent warehouses to which the products of the factories can be delivered. These nodes are called *demand nodes*. The numbers associated with demand nodes represents their product capacities. The edge labels represent the cost of transporting goods from a source to a demand node. (There is no limit on the capacity of an edge to transport products.) The transportation problem is to compute the amount of products to be transported from cities to warehouses along the various edges such that the overall cost of transportation is minimal.

Formally the transportation problem is represented by a weighted bipartite graph $G = (U \cup V, E, W, C)$ where the



(a) Original graph (b) Optimal assignment (O) (c) Foil assignment (F)

Figure 7: Transportation problem example with solution and foil assignment

first three components of the tuple have the same meaning as they had in the weighted bipartite matching problem from Section 2. In addition, the function $C : U \cup V \to \mathbb{N}$ assigns to each node a supply or a demand capacity. A *flow assignment* is a mapping $\Phi : E \to \mathbb{N}$ that respects the capacity of nodes and describes how much product to ship across each edge. The capacity constraint can be expressed with the help of the following function for computing the sum of the flow values of all edges incident to a particular node.

$$\sigma_\Phi(x) = \sum\{n \mid (v, w) \mapsto n \in \Phi \wedge (x = v \vee x = w)\}$$

The formalization of the transportation problem as a component-based problem is now as follows.

- $I = \{G = (U \cup V, E, W, C) \mid (U \cup V, E, W)$ is a labeled graph and $C : U \cup V \to \mathbb{N}\}$
- $f(G) = \{\ \Phi \mid \Phi : E \to \mathbb{N} \wedge \forall (u, v) \in E.\sigma_\Phi(u) = C(u) \wedge \sigma_\Phi(v) = C(v)\}$
- $C = E$
- $\vec{m}(G, \Phi) = \{e \mapsto n \cdot W(e) \mid e \mapsto n \in \Phi\}$
- $\oplus = +$
- $\mu = \min$

Note that in the decomposed measure function, each value is obtained by multiplying the assigned flow with the cost of transportation across that edge.

The transportation problem is similar to the weighted bipartite matching problem in problem structure as well as the structure of the explanations which are generated. A key difference is that in the matching problem edges are either selected or not whereas in the transportation problem edges are weighted. Since we can express binary selection decisions with weights of 0 and 1, we can actually view and represent matching problems as special instances of transportation problems. (We also have to set the capacity of all nodes to 1.)

An optimal flow assignment for our example transportation problem is shown in Figure 7b. The cost along each edge can be obtained by multiplying the edge labels from the assignment with the edge labels of the original graph. The total cost is the sum of all these values, which is in this case $2{\cdot}3 + 3{\cdot}1 + 5{\cdot}4 + 0{\cdot}2 + 2{\cdot}4 + 0{\cdot}3 + 3{\cdot}3 = 46$.

In the solution, we can observe that warehouse X with a demand of 7 is receiving 5 units of products from B through the high-cost edge B-X. This may raise the question, *Would it be cheaper to move more products to X from T?*. The foil F generated by this consideration is shown in Figure 7c. Surprisingly, F has a higher total cost of 52.

To generate an explanation as to why the solution is better than the foil, we first compute the joint value decompositions for the assignments O and F.

$$\begin{aligned}\delta_{OF} = \{&T\text{-}X \mapsto (6, 1\dot{5}), T\text{-}Y \mapsto (3, \dot{0}),\\ &B\text{-}X \mapsto (20, \dot{8}), B\text{-}Y \mapsto (0, \dot{0}), B\text{-}Z \mapsto (8, 2\dot{0}),\\ &P\text{-}Y \mapsto (0, \dot{9}), P\text{-}Z \mapsto (9, \dot{0})\}\end{aligned}$$

Note that the values in the value decompositions represent the total cost of transportation along an edge and not the quantity to be transported across an edge. This means that a user may want to look at an explanation together with the solution. In an implemented system one could have features to select between different edge labels (assignments, computations, computed values) in order to customize an explanation to the specific needs of a user.

The complete explanation is shown in Figure 8a where the green and red labels represent the optimal and foil costs, respectively. The focused explanation computes the difference between solution and foil values for common components and otherwise omits edges with the same values in both solution and foil, which in this case is the edge B-Y. And again, in this small example, an MDS explanation doesn't yield a smaller explanation than the focused one.

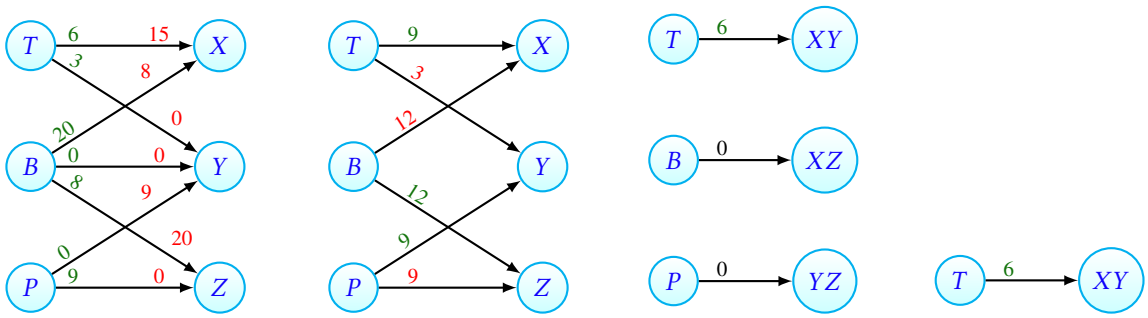
$$\delta^{\circ}_{OF} = \delta^{\text{MDS}}_{OF} = \{T\text{-}X \mapsto 9, T\text{-}Y \mapsto \dot{3}, B\text{-}X \mapsto 1\dot{2}, B\text{-}Z \mapsto 12, P\text{-}Y \mapsto 9, P\text{-}Z \mapsto \dot{9}\}$$

The focused explanation is shown in Figure 8b. Green labels are used for edges that favor the solution, while red labels are used for edges that favor the foil.

The explanation can be interpreted in the following way: The foil can achieve indeed significant savings (of 12) along the B-X edge. However, this gain is completely lost by the advantage that the solution has over the foil along edge B-Z. Similarly, the foil's advantage for edge P-Z is compensated by the solution's advantage for edge P-Y. This leaves to compare edges T-X and T-Y, which shows the overall advantage of the solution.

The above strategy of "balancing" or "canceling" advantages can actually be simulated by an aggregated explanation. Specifically, we can group edges by their source nodes to obtain aggregated costs for all the connected warehouses. The aggregated value decomposition reflects the canceling by having a total of 0 for both B and P.

$$\delta^{\circ}_{OF}\rangle^{1}_{=} = \{T\text{-}XY \mapsto 6, B\text{-}XZ \mapsto 0, P\text{-}YZ \mapsto 0\}$$



(a) Complete (b) Focused = MDS (c) Aggregated (d) Aggregated MDS

Figure 8: Explanations for the transportation problem

The corresponding aggregated explanation is shown in Figure 8c. For this aggregated representation, minimal dominating sets can effectively shrink the value decomposition as follows.

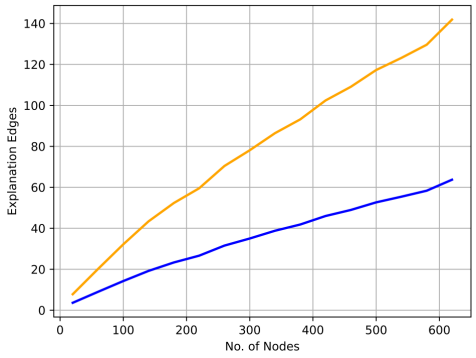
$$\delta_{OF}^{\text{MDS}}\rangle_{=}^{1} = \{T\text{-}XY \mapsto 6\}$$

The corresponding aggregated MDS explanation is shown in Figure 8d, which tells us that a benefit of 6 from T, which is a result of aggregating the gain of 9 along T-X and a loss of 3 along T-Y, is the reason why the optimal solution is the better than the foil.

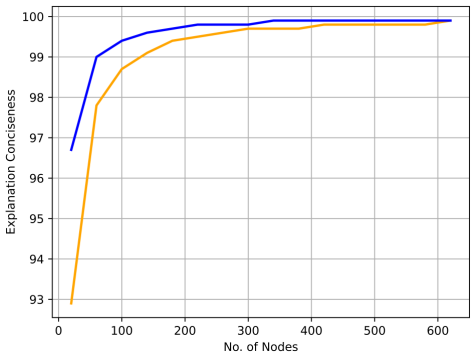
In our example the size of the complete explanation is $|\delta_{OF}| = 13$ and that of both focused and MDS explanation is $|\delta_{OF}^{\circ}| = |\delta_{OF}^{\text{MDS}}| = 6$, which means a conciseness of $C(\delta_{OF}^{\circ}) = C(\delta_{OF}^{\text{MDS}}) = 1 - 6/13 = 54\%$. The size of aggregated and aggregated MDS explanations are $|\delta_{OF}^{\circ}\rangle_{=}^{1}| = 3$ and $|\delta_{OF}^{\text{MDS}}\rangle_{=}^{1}| = 1$, respectively, giving us a explanation conciseness of $C(\delta_{OF}^{\circ}\rangle_{=}^{1}) = 1 - 3/13 = 77\%$ and $C(\delta_{OF}^{\text{MDS}}\rangle_{=}^{1}) = 1 - 1/13 = 92\%$.

Next we examine the efficacy of our explanation techniques for the transportation problem through various experiments. To keep the data similar to our matching example and allow for a better comparison, we assume that supply and demand as well as the transportation costs are numbers between 30 and 2,000. Also, like in the weighted bipartite matching example, we assume that the number of left nodes and right nodes in the graph is the same. We have evaluated the size of explanations for graphs of size 20 to 620 (generated in increments of 40 nodes). Like with the other evaluations in this paper, to keep the simulation realistic, we consider a foil only when the foil cost is within 2% of the solution. For each graph size we have generated 5,000 pairs of optimal and close foil assignments, and we have computed the size of complete explanations and explanation conciseness for each pair. The overall value of these metrics for a graph size is the average of the 5,000 values computed for that size.

Figure 9a displays the number of explanation edges on the Y axis and the size of the bipartite graph on the X axis. For focused and aggregated explanations, the number of explanation edges increases linearly with the number of nodes such that the number of edges is almost half the size of the graph. The size of focused explanations is about $1/4^{\text{th}}$ and that of aggregated explanations is about $1/8^{\text{th}}$ of the number of nodes in the graph. MDS explanations result in a slight but insignificant improvement over focused explanations and hence are not shown in evaluations.



(a) Size of explanations



(b) Conciseness of explanations

Figure 9: Explanation size and conciseness for the transportation problem. Techniques: Focused, Aggregated and Aggregated MDS

Figure 9b shows that the conciseness increases in the shape of a hyperbola with the size of graphs for the various explanations. Focused and aggregated MDS explanations start with a very high conciseness of 93% and 97%, respectively, which flattens out at around 99.5%. The low number of explanation edges and the correspondingly high explanation conciseness suggest the efficacy and scalability of our approach.

8. Explaining Maximum Flows

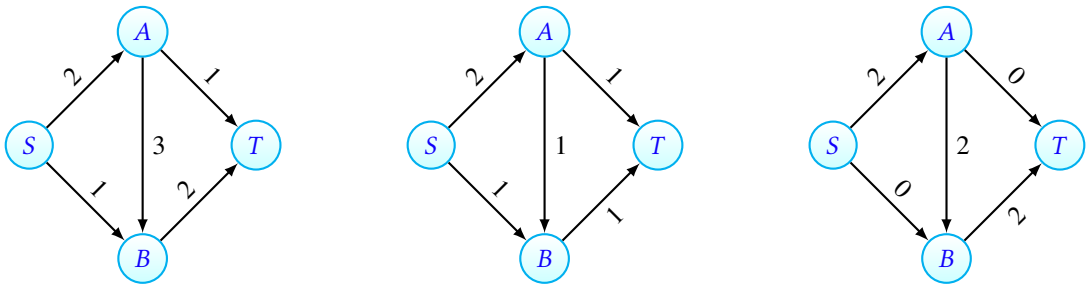
In maximum-flow problems, the edge labels of a directed graph represent the capacity of an edge, that is, the maximum amount of *flow* possible along that edge. Given a source node s and target node t, a *valid flow* between s and t is defined as the sum of the flow of the outgoing edges of s (or equivalently, the sum of the flow of t's incoming edges) such that the following two conditions are satisfied.

- *Capacity Constraint*: The flow along any edge is no more than its capacity.
- *Flow Conservation*: Except for the source and the target nodes, the amount of flow entering a node must be equal to the amount of flow exiting the node.

The maximum-flow problem is to compute the maximum valid flow between s and t. For example, for the graph in Figure 10a, the maximum flow between S and T is 3. The flow along various edges leading to the maximum flow is shown in Figure 10b.

Formally, the maximum-flow problem is represented by a directed labeled graph $G = (V, E, C)$ where V and E are sets of vertices and edges, respectively, and the function $C : E \to \mathbb{N}$ assigns a flow capacity to each edge. As in transportation problems, a flow assignment is a mapping $\Phi : E \to \mathbb{N}$ that respects the capacity and the flow conservation constraints, and for each node $v \in V$, $\sigma^-_\Phi(v)$ and $\sigma^+_\Phi(v)$ represent the total inflow and outflow of node v, respectively.

The formalization of the maximum-flow problem as a component-based problem is now as follows.



(a) Original graph (b) Optimal assignment (O) (c) Foil assignment (F)

Figure 10: Maximum-flow problem with solution and foil assignments

- $I = \{(G, s, t) \mid G = (V, E, C)$ is a labeled graph, $\{s, t\} \subseteq V$, and $C : E \to \mathbb{N}\}$
- $f(G, s, t) = \{\Phi \mid \Phi : E \to \mathbb{N} \wedge \forall (u, v) \in E.(v \neq t \implies \sigma^-_\Phi(v) = \sigma^+_\Phi(v)) \wedge (\Phi(u, v) \leq C(u, v))\}$
- $C = E$
- $\vec{m}(G, \Phi) = \{e \mapsto n \mid e \mapsto n \in \Phi\}$
- $\oplus = +$
- $\mu = \max$

A surprising aspect of the optimal flow shown in Figure 10b is that the flow along the edge *A*-*B* is only 1, even though its capacity is 3. A user may wonder whether the overall maximum flow could be increased by increasing the flow along *A*-*B*. A corresponding foil flow is shown in Figure 10c. Note that the flow value for *A*-*B* can't be 3, because the total capacity for the outgoing flow for *B* is only 2. Thus, the flow along AB can be at most 2. However, if the flow along AB is increased to 2, the total maximum flow in the network decreases from 3 to 2, which is surprising.

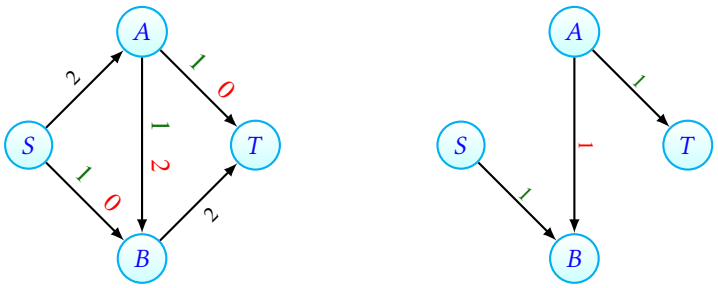
As with the previous examples, to explain the solution, we start by computing the join of the two value decompositions for the assignments *O* and *F*.

$$\delta_{OF} = \{S\text{-}A \mapsto (2, \dot{2}), A\text{-}B \mapsto (1, \dot{2}), A\text{-}T \mapsto (1, \dot{0}), S\text{-}B \mapsto (1, \dot{0}), B\text{-}T \mapsto (2, \dot{2})\}$$

The corresponding complete explanation is shown in Figure 11a with green numbers representing the flow in the solution and red numbers the flow in the foil. The focused value decomposition omits edges that have the same flow for solution and foil.

$$\delta^\circ_{OF} = \delta^{\text{MDS}}_{OF} = \{A\text{-}B \mapsto \dot{1}, A\text{-}T \mapsto 1, S\text{-}B \mapsto 1\}$$

The corresponding focused explanation is shown in Figure 11b, which again is equal to the MDS explanation. The focused explanation shows that increasing the flow for edge *A*-*B* leads indeed to an advantage of 1 for that edge for the foil, but it is superseded by the advantages for the edges *S*-*B* and *A*-*T* in the solution.



(a) Complete explanation (b) Focused & MDS explanation

Figure 11: Explanations of the maximum-flow solution

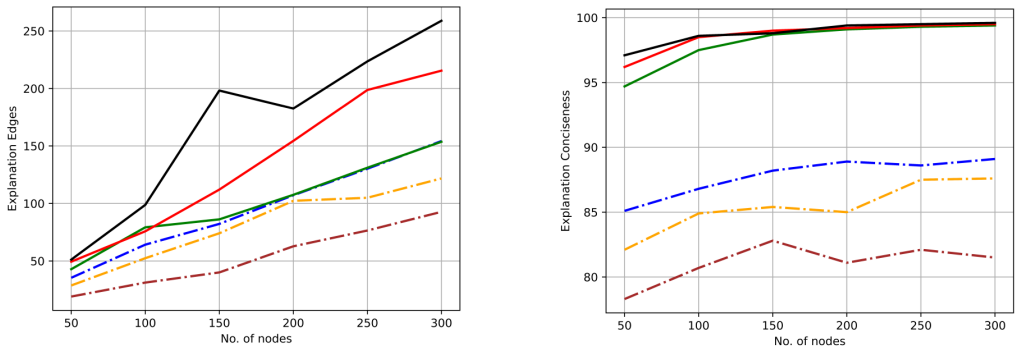
In our example the size of the complete explanation is $|\delta_{OF}| = 8$ and that of both focused and MDS explanation is $|\delta^{\circ}_{OF}| = |\delta^{\text{MDS}}_{OF}| = 3$, which means a conciseness of $C(\delta^{\circ}_{OF}) = C(\delta^{\text{MDS}}_{OF}) = 1 - 3/8 = 62.5\%$.

In the experiments to investigate the efficacy of our explanation approaches for maximum-flow problems we have categorized flow networks low and high density (or connectivity). Examples of low-density networks are physical networks, such as road and train networks, electricity networks, sewer and water pipelines, and irrigation systems [29, 44, 52]. Networks with high density include telecommunication and wifi networks. The density sometimes depends on attributes such as performance and safety [46]. To ensure that our results are valid for networks of both types, we have performed experiments for sparse graphs in which nodes have an average (in+out) degree of 4, 6, and 8, and for dense graphs with densities 0.3, 0.5 and 0.7.

To keep the simulation realistic, we only consider a foil flow if its value is no less than 98% of the maximum flow. The flow capacities of various edges are assigned random numbers between 30 and 2,000. Like in the other experiments in the paper, we have selected this range to prevent artificially inflating the conciseness. We have evaluated the size of explanations for graphs of size 50 to 300 (generated in increments of 50 nodes). For each graph size, we have generated 10,000 pairs of optimal and close foil assignments and computed the size of focused explanations and explanation conciseness for each pair. The overall value of these metrics for a graph size is the average of the 10,000 values computed for that size.

The graphs in Figure 12a show the number of explanation edges and explanation conciseness on the Y axis for the size of the flow graphs on the X axis. For each graph size (in terms of number of nodes) we generate flow graphs for the different degrees and densities. As Figure 12a reveals, explanation size increases almost linearly with the number of nodes. Although density does affect the size of explanations, it does so only by a constant factor.

Figure 12b illustrates the explanation conciseness. Two trends can be observed. Firstly, conciseness increases hyperbolically with the number of nodes (regardless of graph density), and reaches stabilization between 97.5% and 99.5% for dense graphs. On the other hand, for sparse graphs, conciseness remains mostly in the 80-90% range. Secondly, we also observe that conciseness increases with the density of the graph. This may seem counter-intuitive at first. However, as the density of the graph increases, the fraction of edges with common flows also increases,



(a) Size of focused explanations.

(b) Effectiveness of focused explanations.

Figure 12: Explanation conciseness for the maximum-flow problem.
Sparse graphs, average (in+out) degree: 4, 6, and 8; Dense graph densities: 0.3, 0.5, 0.7

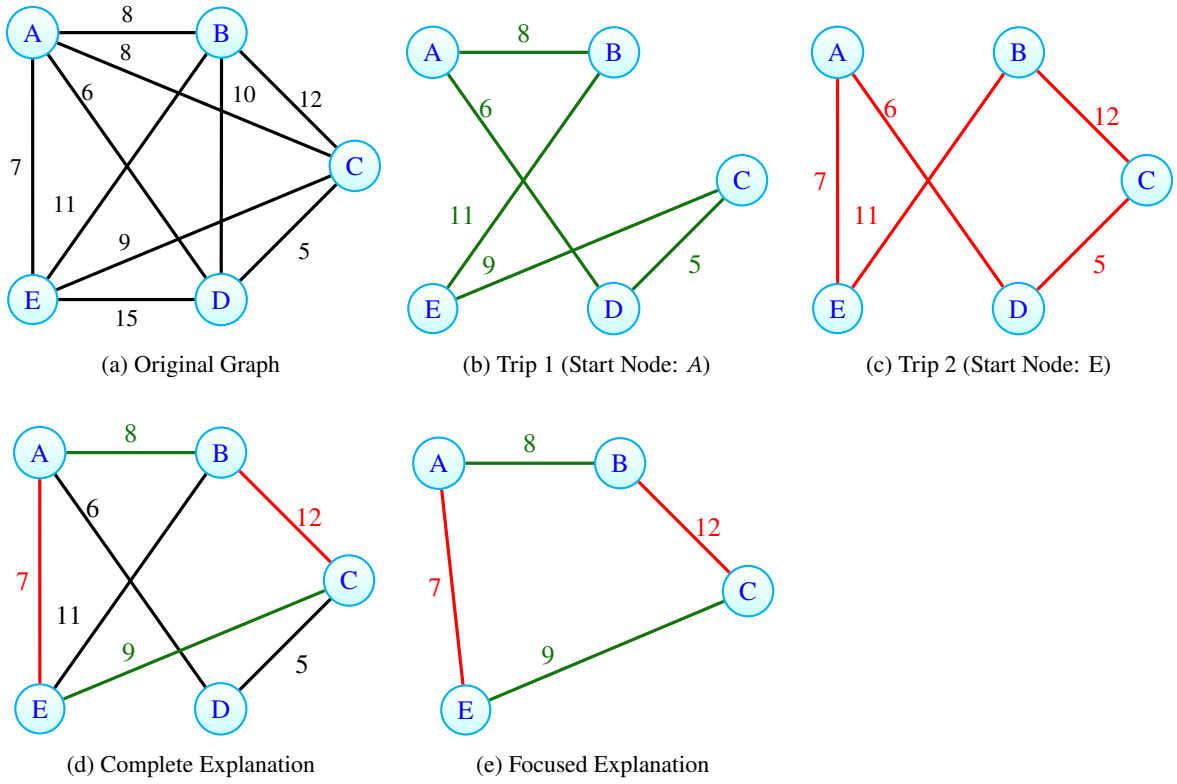
leading to an increase in the conciseness measure. Altogether, the evaluation indicates that our explanation approach scales for maximum-flow problems.

9. Explanations for NP-Hard Combinatorial Optimization Problems

In the preceding sections we saw the application of our explanation techniques for optimization problems that are efficiently solvable. A natural question then arises: Would our approach work for NP-hard problems as well? In this section, we demonstrate, using the Traveling Salesman Problem (TSP) [32], that our explanation mechanism indeed works effectively for such problems.

Consider the graph in Figure 13. The traveling salesman problem is to find an optimal round trip, that is, given a starting point, for instance, node A, we visit all the other nodes of the graph exactly once and return to A, seeking to minimize the total trip distance. Problems like this often arise in logistics and route planning. Since the TSP is an NP-hard problem and therefore finding exact, optimal solutions is often impossible, approximation algorithms that find "good enough" solutions are commonly employed. This adds a small twist to our explanation methods, but doesn't change the overall picture. Here we we will use for simplicity a simple greedy algorithm, known as the *nearest-neighbour algorithm* [11], but the discussion applies similarly to other, more sophisticated algorithms discussed in [42].

The nearest-neighbour algorithm begins by choosing a start node and then repeatedly selects the neighbor connected by the shortest edge. In our example, we choose node D as the nearest neighbour of A. Then, we find the nearest neighbour of D from the remaining nodes, and so on, until we have visited all nodes, as shown in the trip Figure 13b. Since we are computing a round trip, we return to node A from node B. The total cost of this trip is 39 units. The outcome of the nearest-neighbour algorithm generally depends on the initial node that is selected. That is, selecting a different starting node might result in a different trip with a different cost. Figure 13c shows the trip computed with node E as the starting node, which has a cost of 41 units.



(a) Original Graph (b) Trip 1 (Start Node: A) (c) Trip 2 (Start Node: E)

(d) Complete Explanation (e) Focused Explanation

Figure 13: Explanations of Travelling Salesman Problem Results

A user might ask why the second trip is costlier than the first. We can use our explanation strategy to address this query. The first trip, for the purpose of generating explanation, acts as the optimal output, and the second, costlier trip, is the foil. The complete and focused explanations are shown in Figures 13d and 13e, respectively. We observe that the sum of foil only edges $(7 + 12 = 19)$ AE and BC is larger than the optimal only edges AB and EC $(8 + 9 = 17)$. This makes the foil path more expensive.

Since an approximation algorithm doesn't guarantee the computation of an optimal solution, a foil contemplated by a user might actually result in a better solution. In that case, the same reasoning and presentation applies, just that in this case the foil takes on the role as the optimal solution and the previously thought best solution is now the foil; everything else stays the same. For non-approximation algorithm this twist cannot occur, but as discussed, the reasoning about differences of closely related solutions is not affected anyhow.

The efficacy of our explanation techniques for NP-hard problems is not really surprising, since our explanation techniques work on the representation of the output and are not directly concerned with the complexity of the algorithms generating it.

10. Related Work

The topic of explanations has been explored in a number of different areas. While the origins of research into the nature of explanations can be traced back to philosophy [2, 30, 43], the need for explaining computation has recently

received a lot of attention, specifically in the area of AI [3, 36].

Explanations for individual program executions have been investigated, especially in the area of debugging where an explanation is based on some representation of the program execution that is built alongside the computation, *program traces* being the most popular example.

10.1. Explanations from Traces

Since traces can grow very large quickly, even for small examples, one has to find some means to reduce the amount of information presented to users for traces to be of use. [39] as well as [41] describe how the use of dynamic slicing for traces can help to generate explanations for the executions of functional programs. Their approach is based on eliminating those parts from a trace that do not contribute to user-selected parts of outputs. Specifically, their technique takes a section of the computation result that is selected by the user and uses *backward slicing* [53] to generate a path to the input focusing on the computations that were responsible for that section. Although the approach is effective, the resulting traces can still be quite large and include many details that, while technically relevant for the computation, don't contribute to the explanation.

Similar to traces, value decompositions gather details about program executions, but they are generally much smaller and contain fewer details. Moreover, since value decompositions are directly linked to the information on which the decisions of the optimization are based, they are more likely to contain information that is relevant for explanations. Another difference is that explanations in our approach are based on joint value decompositions, which require at least two program executions to obtain a foil in addition to the solution. A limitation of value decomposition is that they only work for a specific class of programs whereas generic program traces are a universally applicable structure.

In this regard, value decompositions are similar to provenance traces, which are a representation explicitly created for the purpose of explanation [1]. A provenance trace consists of meta-information about the origin, history, or derivation of an object which is used in establishing trust and providing security in computer systems, particularly on the web. Like value decompositions, provenance traces are a domain-specific explanation structure that works only in certain situations. This approach is very similar to backward slicing presented by [39] and consequently suffers similar limitations. Another domain-specific approach to keep traces small has been explored in the realm of concurrent programs where a heuristic algorithm is used to simplifying traces with the goal of identifying concurrency bugs [31].

A different approach to manage trace complexity was pursued in [8] by defining a query language for the systematic transformation of (tree-based) program traces. The query language offers operations for hiding and propagating information plus a rich set of selectors to specify the set of nodes operations should be applied to. The query language has been used to define a rich set of filters that users can apply selectively to their program traces to focus on interesting and relevant parts.

A mechanism to stepwise explain how to solve constraint satisfaction problems is described in [12]. While our approach uses two outputs, optimal and foil, to generate the explanation, the constraint satisfaction work explains just the output of the constraint problem. Additionally, we have completed, focused, MDS, and aggregated explanations which represent explanations at different levels of detail. The constraint satisfaction work has the notion of nested explanations that allow users to focus on a step of the problem that they find hard to understand. In this respect, their explanation is more fine-grained than ours. There is also a notion of minimalism in the components that are selected for the explanation in both the works. Given that we have a numeric domain, for example, we have values associated with explanation components like edge value in the shortest path problem, we directly use this value in selecting the components for the explanation. Since the constraint satisfaction domain is generally non-numeric, they develop a cost function to remove unnecessary components from their explanation. Perhaps the most important difference between the two approaches is that while our approach is algorithm agnostic and just makes use of the representation of the output, the constraint satisfaction work makes use of the trace of the algorithm for explanation.

10.2. Domain-Specific Structures for Explanations

We already mentioned the approaches of [1] and [31] that exploited domain-specific information to generate smaller program traces. Our concept of deriving explanations from joint value decompositions is more general and applies to any program that implements a combinatorial optimization problems.

The notion of a value decomposition was introduced in [17] as a structure for explaining the results of dynamic programs and is thus also a domain-specific structure, which relies on the fact that dynamic programming algorithms can be viewed as instances of a mathematical semiring structure. That approach also generates contrastive explanations and thus also requires two program results. In addition to the different application domains, another important difference to our approach is that their technique requires an "explanation designer" to specifically decompose the values being used in the algorithm and associate a semantics to the decomposed values. In contrast, our explanation mechanism doesn't require any additional annotation and works with the original specification of the problem making it easier to adapt.

Finally, we have adopted the concepts of value decompositions and minimal dominating sets (MDSs) from [17], but we apply them in a subtly different way. Specifically, in our representation the set of components is constructed individually for each solution to a specific problem, whereas the set of categories in [17] has to be fixed before any computation takes place. This difference can be characterized as *dynamic* vs. *static* value decompositions.

11. Conclusions

Algorithms for solving combinatorial optimization problems are widely used, but techniques to explain their outputs are scarce. This lack of explainability presents a significant challenge, as explanations are crucial for building users' trust and acceptance. This is especially true as the "right to explanation" is increasingly recognized as a fundamental right by countries around the world.

We have introduced a representation that facilitates the automatic generation of explanations for any algorithm solving combinatorial optimization problems, requiring only the identification of an appropriate component structure, which is typically provided as part of the problem description. At the core of our representation is a simple structure, the joint value decomposition, which allows the generation of complete, focused, MDS, and aggregated versions of explanations that can answer questions on different levels of granularity about why a particular solution is better than putative alternatives. Our experiments show that the explanations produced by our approach are highly concise and scale well across a range of different applications.

Ultimately, the understandability of computational results and their explanations depends on individual users, their backgrounds, and their expectations. Thus, to assess which explanations are helpful for which users, user studies are required. Such studies require a lot of effort and are beyond the scope of the current paper, which is concerned with establishing the formal foundation of generic, wide-ranging explanation structures. In future work, we plan to conduct those studies when we have the opportunity to test specific applications of optimization problems. A related aspect is the idea of adapting explanations to the specific needs of different users, for example, by interactively constructing explanations based on user input. We have explored this question previously in the context of generating user-specific program traces [7, 8, 9]. In future work, we will explore specifically how to construct most relevant foils through user interaction.

References

[1] Acar, U.A., Ahmed, A., Cheney, J., Perera, R.: A Core Calculus for Provenance. In: Int. Conf. on Principles of Security and Trust. pp. 410–429 (2012)

[2] Achinstein, P.: The Nature of Explanation. Oxford University Press, New York, NY (1983)

[3] Adadi, A., Berrada, M.: Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI). IEEE Access **6**, 52138–52160 (2018)

[4] Amazon and MIT: Amazon Last Mile Routing Research Challenge. `https://routingchallenge.mit.edu/` (2021)

[5] Augen, P.: The Anthem dictionary of literary terms and theory. Anthem Press (2010)

[6] Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., Kann, V.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer-Verlag, Berlin, Heidelberg, 1st edn. (1999)

[7] Bajaj, D., Erwig, M., Fedorin, D., Gay, K.: A Visual Notation for Succinct Program Traces. In: IEEE Int. Symp. on Visual Languages and Human-Centric Computing. pp. 1–9 (2021)

[8] Bajaj, D., Erwig, M., Fedorin, D., Gay, K.: Adaptable Traces for Program Explanations. In: Asian Symp. on Programming Languages and Systems. pp. 202–221. LNCS 13008 (2021)

[9] Bajaj, D., Erwig, M., Fedorin, D., Gay, K.: A Visual Notation for Succinct Program Traces. Journal of Computer Languages **75** (2023)

[10] Bellman, R.: On a routing problem. Quarterly of Applied Mathematics **16**(1), 87–90 (1958)

[11] Bellmore, M., Nemhauser, G.L.: The traveling salesman problem: A survey. Operations Research **16**(3), 538–558 (1968)

[12] Bogaerts, B., Gamba, E., Guns, T.: A framework for step-wise explaining how to solve constraint satisfaction problems. Artificial Intelligence **300**, 103550 (2021)

[13] Cunha, J., Dan, M., Erwig, M., Fedorin, D., Grejuc, A.: Explaining Spreadsheets with Spreadsheets. In: ACM SIGPLAN Conf. on Generative Programming: Concepts & Experiences. pp. 161–167 (2018). https://doi.org/https://doi.org/10.1145/3278122.3278136

[14] Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959)

[15] DIMACS: 9th DIMACS Implementation Challenge - Shortest Paths. `http://www.diag.uniroma1.it//challenge9/download.shtml` (2006)

[16] Erwig, M., Gopinath, R.: Explanations for Regular Expressions. In: Int. Conf. on Fundamental Approaches to Software Engineering. pp. 394–408. LNCS 7212 (2012)

[17] Erwig, M., Kumar, P.: Explainable Dynamic Programming. Journal of Functional Programming **31**(e10) (2021)

[18] Erwig, M., Kumar, P.: MADMAX: A DSL for Explanatory Decision Making. In: ACM SIGPLAN Conf. on Generative Programming: Concepts & Experiences. pp. 144–155 (2021)

[19] Erwig, M., Kumar, P., Fern, A.: Explanations for Dynamic Programming. In: Int. Symp. on Practical Aspects of Declarative Languages. pp. 179–195. LNCS 12007 (2020)

[20] Erwig, M., Walkingshaw, E.: A DSL for Explaining Probabilistic Reasoning. In: IFIP Working Conference on Domain-Specific Languages. pp. 335–359. LNCS 5658 (2009)

[21] Erwig, M., Walkingshaw, E.: Visual Explanations of Probabilistic Reasoning. In: IEEE Int. Symp. on Visual Languages and Human-Centric Computing. pp. 23–27 (2009)

[22] Erwig, M., Walkingshaw, E.: Causal Reasoning with Neuron Diagrams. In: IEEE Int. Symp. on Visual Languages and Human-Centric Computing. pp. 101–108 (2010)

[23] Erwig, M., Walkingshaw, E.: A Visual Language for Explaining Probabilistic Reasoning. Journal of Visual Languages and Computing **24**(2), 88–109 (2013)

[24] Faulhaber, A.K., Ni, I., Schmidt, L.: The effect of explanations on trust in an assistance system for public transport users and the role of the propensity to trust. In: Mensch Und Computer 2021. p. 303–310. MuC '21, ACM, New York, NY, USA (2021)

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

[26] Forney, G.D.: The viterbi algorithm. Proceedings of the IEEE **61**(3), 268–278 (1973)

[27] Garfinkel, P.: Forms of Explanation. Yale University Press, New Haven, CT, USA (1981)

[28] Goodman, B., Flaxman, S.: European union regulations on algorithmic decision-making and a "right to explanation". AI Magazine **38**, 50–57 (2017)

[29] Hadaj, P., Strzałka, D.: Modelling selected parameters of power grid network in the south-eastern part of poland: The case study. Energies **13**(1) (2020)

[30] Hempel, C.: Aspects of Scientific Explanation and Other Essays in the Philosophy of Science. Free Press, New York, NY (1965)

[31] Jalbert, N., Sen, K.: A trace simplification technique for effective debugging of concurrent programs. In: ACM Int. Symposium on Foundations of Software Engineering. pp. 57–66 (2010)

[32] Lawler, E., Lenstra, J., Rinnooy Kan, A., Shmoys, D.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Wiley (1985)

[33] Lawler, E.L.: A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. Management Science **18**(7), 401–405 (1972)

[34] Lipton, P.: Contrastive Explanation. Royal Institute of Philosophy Supplement **27**, 247–266 (1990)

[35] Lipton, P.: Inference to the Best Explanation. Routledge, New York, NY, USA (2004)

[36] Miller, T.: Explanation in Artificial Intelligence: Insights from the Social Sciences. Artificial Intelligence **267**, 1–38 (2019)

[37] Murty, K.G.: Letter to the editor - an algorithm for ranking all the assignments in order of increasing cost. Oper. Res. **16**, 682–687 (1968)

[38] Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity (1981)

[39] Perera, R., Acar, U.A., Cheney, J., Levy, P.B.: Functional Programs That Explain Their Work. In: ACM Int. Conf. on Functional Programming. pp. 365–376 (2012)

[40] Read, S.J., Marcus-Newhall, A.: Explanatory coherence in social explanations : a parallel distributed processing account. Journal of Personality and Social Psychology **65**, 429–447 (1993)

[41] Ricciotti, W., Stolarek, J., Perera, R., Cheney, J.: Imperative Functional Programs that Explain their Work. Proceedings of the ACM on Programming Languages **1**(ICFP) (2017)

[42] Rosenkrantz, D., Stearns, R., Lewis, P.: An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing **6**(3), 563–581 (1977)

[43] Ruben, D.H.: Explaining Explanation. Routledge, London, UK (1990)

[44] Schrijver, A.: On the history of the transportation and maximum flow problems. Mathematical Programming **91**, 437–445 (2002)

[45] Selbst, A.D., Powles, J.: Meaningful information and the right to explanation. International Data Privacy Law **7**, 233–242 (2017)

[46] Teng, J., Gu, W., Xuan, D.: Chapter 10 - defending against physical attacks in wireless sensor networks. In: Das, S.K., Kant, K., Zhang, N. (eds.) Handbook on Securing Cyber-Physical Critical Infrastructure, pp. 251–279. Morgan Kaufmann, Boston (2012)

[47] Thagard, P.: Explanatory coherence. Behavioral and Brain Sciences **12**(3), 435–467 (1989)

[48] The French Government: Digital Republic Act of France. `https://www.republique-numerique.fr/pages/digital-republ` (2016)

[49] Toth, P., Vigo, D. (eds.): The Vehicle Routing Problem. Society for Industrial and Applied Mathematics (SIAM) (2002)

[50] Trevisan, L.: Combinatorial Optimization: Exact and Approximate Algorithms. Standford University (2011)

[51] Walkingshaw, E., Erwig, M.: A DSEL for Studying and Explaining Causation. In: IFIP Working Conference on Domain-Specific Languages. pp. 143–167 (2011)

[52] Wegner, F.: Network flow models for power grids. Karlsruhe Institute of Technology (2014)

[53] Weiser, M.: Program Slicing. IEEE Transactions on Software Engineering (4), 352–357 (1984)

[54] Yen, J.Y.: Finding the k shortest loopless paths in a network. Management Science **17**(11), 712–716 (1971)