JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 30, Number 11, 2023

Pp. 1–28
DOI: 10.1089/cmb.2023.0242

Recomb 2023

Shortest Hyperpaths in Directed Hypergraphs for Reaction Pathway Inference

SPENCER KRIEGER[1] and JOHN KECECIOGLU[2]

ABSTRACT

Signaling and metabolic pathways, which consist of chains of reactions that produce target molecules from source compounds, are cornerstones of cellular biology. Properly modeling the reaction networks that represent such pathways requires directed *hypergraphs*, where each molecule or compound maps to a vertex, and each reaction maps to a *hyperedge* directed from its set of input reactants to its set of output products. Inferring the most likely series of reactions that produces a given set of targets from a given set of sources, where for each reaction its reactants are produced by prior reactions in the series, corresponds to finding a *shortest hyperpath* in a directed hypergraph, which is NP-complete.

We give the first *exact algorithm* for general shortest hyperpaths that can find provably optimal solutions for large, real-world, reaction networks. In particular, we derive a novel *graph-theoretic characterization* of hyperpaths, which we leverage in a new integer linear programming formulation of shortest hyperpaths that for the first time handles cycles, and develop a *cutting-plane algorithm* that can solve this integer linear program to optimality in practice. Through comprehensive experiments over all of the thousands of instances from the standard Reactome and NCI-PID reaction databases, we demonstrate that our cutting-plane algorithm quickly finds an optimal hyperpath—inferring the most likely pathway—with a median running time of under 10 seconds, and a maximum time of less than 30 minutes, even on instances with thousands of reactions. We also explore for the first time how well hyperpaths infer true pathways, and show that shortest hyperpaths *accurately recover* known pathways, typically with very high precision and recall.

Source code implementing our cutting-plane algorithm for shortest hyperpaths is available free for research use in a new tool called `Mmunin`.

Keywords: cell signaling networks, combinatorial optimization, cutting plane algorithms, directed hypergraphs, inferring pathways, integer linear programming, metabolic networks, shortest hyperpaths, systems biology.

[1]Computational Biology Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA.
[2]Department of Computer Science, The University of Arizona, Tucson, Arizona, USA.

1. INTRODUCTION

SIGNALING AND METABOLIC PATHWAYS are cornerstones of systems biology. They underlie cellular communication, govern environmental response, and their perturbation has been implicated in the causes of disease (Li et al., 2020). Networks comprised of these pathways are traditionally represented as ordinary graphs (Sharan and Ideker, 2006; Vidal et al., 2011), modeling each molecule as a vertex, and each reaction by a collection of edges directed from each of its reactants to each of its products. Such representations in terms of ordinary graphs, however, do not faithfully model multiway reactions—which are ubiquitous in cellular processes—and shortest paths from these models are often not biologically meaningful (Klamt et al., 2009; Ritz et al., 2014).

Directed *hypergraphs* generalize ordinary graphs, where an edge, now called a *hyperedge*, is directed from one set of vertices, called its *tail*, to another set of vertices, called its *head*. Hypergraphs have been used to model many cellular processes (Christensen et al., 2009; Heath and Sioson, 2009; Hu et al., 2007; Klamt et al., 2009; Ramadan et al., 2010; Ramadan et al., 2004; Ritz and Murali, 2014; Ritz et al., 2014; Zhou and Nakhleh, 2011). In particular, a reaction with multiple *input reactants* (all of which must be present for the reaction to proceed) and multiple *output products* (all of which are produced upon its completion) is correctly captured by a single hyperedge, directed from its set of reactants to its set of products. Despite hypergraphs affording more faithful models of cellular processes, the lack of practical hypergraph algorithms has hindered their potential for properly representing and reasoning about such reaction networks.

Biologically, a typical metabolic or signaling pathway consists of a collection of reactions that synthesize a set of *target* molecules—key metabolites or transcription factors—from a set of *source* compounds—molecules available to the cell or activated membrane-bound receptors (Alberts et al., 2007). Computationally, inferring the most likely pathway maps to the *shortest hyperpath* problem (a generalization of shortest paths to directed hypergraphs) that we consider in this paper: given a cellular reaction network whose reactions, reactants, and products are modeled by the vertices and weighted hyperedges of a directed hypergraph, together with a set of sources and a set of targets, find a series of hyperedges of minimum total weight that yields a pathway producing the targets from the sources.

Next we briefly survey related work, then give our contributions, and the plan of the paper.

1.1. Related work

The two fundamental hypergraph models that have emerged for pathway inference in cellular reaction networks are hyperpaths and factories (see Krieger, 2022). We first contrast them, and then summarize computational results.

Informally, a *hyperpath* in a directed hypergraph is a series of hyperedges that starts from a set of source vertices and ends at a set of target vertices, where for each hyperedge in the series, the vertices it comes out of are covered by the vertices that prior hyperedges go into. In a network whose reactions are modeled by hyperedges, this means for each reaction its input reactants are created as the output products of prior reactions, so the reactions in the hyperpath may proceed without interruption from sources to targets.

On the other hand, a *factory* in a directed hypergraph for a network whose reactions now also have stoichiometries, is a set of hyperedges that again starts from a set of sources and ends at a set of targets, together with a positive real-valued *flux* on each hyperedge, where for all vertices other than sources or targets, the total incoming flux from all hyperedges that enter the vertex (multiplied by the stoichiometries for the reaction product represented by the vertex) is at least the total outgoing flux from all hyperedges that leave it (again multiplied by the stoichiometries now for the reactant represented by the same vertex). In a metabolic network, this flux condition means that by supplying the source compounds to the factory, its reactions will produce the target molecules without depleting intermediate metabolites.

While factories contain *unordered* hyperedges, and all their reactions effectively proceed simultaneously, hyperpaths contain *ordered* hyperedges, and their reactions essentially proceed in an ordered cascade. For many of the reactions in current pathway databases, stoichiometries are not given (and most reaction rates are unknown). While factories require stoichiometries (or an appropriate default choice for them), hyperpaths do not.

Factories. In systems biology and synthetic biology, factories have mainly been applied to metabolic networks. Current computational approaches find a factory that produces all targets while either minimizing the number of sources it uses, called a min-*source* factory, or minimizing the number of reactions it uses (equivalently, minimizing its number of hyperedges with nonzero flux), called a min-*edge* factory.

Cottret et al. (2008) introduced the *min-source factory* problem and showed it is NP-complete, while Zarecki et al. (2014) extended the problem to consider molecular weights of sources.

Acuña et al. (2012) and Andrade et al. (2016) give methods for *enumerating* all min-source factories, either excluding or including stoichiometry.

Krieger and Kececioglu (2022b) introduced the *min-edge factory* problem, showed it is NP-complete, incorporated higher-order models of *negative regulation* into pathway inference for the first time, and developed approaches based on mixed-integer linear programming (MILP) that are fast in practice to find optimal min-edge factories without interference from negative regulation.

Hyperpaths. Initially studied within the field of algorithms, hyperpaths have been applied in systems biology to pathway inference in cell-signaling and metabolic networks.

Italiano and Nanni (1989) proved that finding a shortest hyperpath is NP-complete, even for *singleton-head hypergraphs* (where all hyperedges have only one head vertex) that are *acyclic* and have *unit-weight* edges.

In a seminal paper that is the source for much of the subsequent work on directed hypergraphs, Gallo et al. (1993) explore many variants of hypergraphs and associated notions of *connectivity*, including what they call a *B*-path (though see the correction of Nielsen and Pretolani, 2001), which is essentially equivalent to our definition of hyperpath in Section 2. They give an algorithm for *reachability* in hypergraphs that finds all vertices reachable from a source vertex in time linear in the total cardinality of the tail- and head-sets of all hyperedges, an efficient algorithm for variants of shortest hyperpaths under *additive cost functions* (our objective in this paper of minimizing the total weight of the hyperedges in a hyperpath is non-additive), and study source-sink cuts in hypergraphs, proving that finding a *minimum cut* in a hypergraph is NP-complete.

Carbonell et al. (2012) gave an efficient algorithm to find a source-sink hyperpath if one exists (irrespective of its length), and proved that finding any hyperpath that must contain a fixed set of hyperedges is NP-complete.

Ritz and Murali (2014) and Ritz et al. (2017) were the first to give a practical *exact algorithm* (guaranteed to find optimal solutions) for *shortest acyclic hyperpaths* (the special case of cycle-free hyperpaths, which remains NP-complete), by formulating it as an MILP, and demonstrated that optimal acyclic hyperpaths can be found in practice even for large cell-signaling networks. Their acyclic MILP formulation—the *state-of-the-art* for shortest hyperpaths—does not extend to general hyperpaths with cycles, which occur naturally in cellular reaction networks, for instance due to feedback loops and the processes of assembly and disassembly of protein complexes.

Schwob et al. (2021) later extended the acyclic MILP formulation to include reaction time-dependencies, while Franzese et al. (2019) study parameterized notions of hypergraph connectivity that interpolate between ordinary-path- and hyperpath-connectivity.

Ausiello and Laura (2017) survey results on *singleton-head hypergraphs*, and note the *inapproximability* of shortest hyperpaths: unless P = NP, no polynomial-time approximation algorithm exists (that is guaranteed to find near-optimal solutions) for shortest hyperpaths with approximation ratio $(1 - o(1)) \ln n$ on hypergraphs with n vertices—which implies there is likely no efficient constant-factor approximation algorithm.

Krieger and Kececioglu (2021, 2022a) gave the first efficient *heuristic* for shortest hyperpaths that allows *cycles*, proved that their heuristic finds optimal hyperpaths for *singleton-tail hypergraphs* (where all hyperedges have only one tail vertex), and developed a tractable hyperpath *enumeration algorithm*. They demonstrated their heuristic is close to optimal in cellular reaction networks through experiments over all source-sink instances from the standard reaction pathway databases, leveraging their enumeration algorithm to verify near-optimality.

1.2. Our contributions

In contrast to prior work, we give the first practical exact algorithm for shortest hyperpaths with cycles, which solves a new integer linear programming (ILP) formulation by a cutting-plane approach to find provably optimal hyperpaths even in large cellular reaction networks. More specifically, we make the following contributions.

- We derive a new *graph-theoretic characterization* of hyperpaths in terms of source-sink cuts, that captures fully-general hyperpaths with cycles.
- We leverage this characterization to obtain the first *integer linear programming formulation* of the general shortest hyperpath problem with nonnegative edge weights. This ILP formulation has an exponential number of constraints, however, and cannot be solved directly.
- We show we can nevertheless solve this ILP indirectly by a practical *cutting-plane approach*, that computes over a small subset of the constraints, while guaranteeing an optimal solution to the full ILP.
- Our cutting-plane approach is typically *fast in practice* on real biological instances, finding optimal hyperpaths with a median running time of under 10 seconds, and a maximum running time of around 30 minutes, as measured through comprehensive experiments over all of the many thousands of source-sink instances from the two standard reaction databases in the literature.
- We also evaluate for the first time how well shortest hyperpaths infer true pathways, and show they *accurately recover* known pathways, typically with very high precision and recall.

A preliminary implementation of our cutting-plane algorithm in a new tool called `Mmunin` (short for ''integer-linear-programming-based cutting-plane algorithm for shortest source-sink hyperpaths'') is available free for noncommercial use at `http://mmunin.cs.arizona.edu`.

1.3. Plan of the paper

The next section defines the computational problem of shortest hyperpaths. Section 3 then gives a new graph-theoretic characterization of hyperpaths that leads to the first formulation of shortest hyperpaths as an *integer linear program* for fully general hyperpaths with cycles, as well as a *cutting-plane algorithm* for solving this integer program to optimality in practice. Section 4 compares our algorithm to alternate hyperpath methods through comprehensive experiments over the two standard reaction pathway databases, and shows we can accurately infer true pathways. Finally Section 5 concludes, and offers directions for further research.

2. SHORTEST HYPERPATHS IN DIRECTED HYPERGRAPHS

We arrive at our computational problem by successively defining hypergraphs, superpaths, hyperpaths, their cycles, and finally the Shortest Hyperpaths problem.

2.1. Directed hypergraphs

Directed hypergraphs are a generalization of directed graphs where an edge, instead of touching two vertices, now connects two subsets of vertices. Formally, a directed *hypergraph* is a pair (V, E), where V is a set of vertices and E is a set of directed *hyperedges*.[†] Each hyperedge $e \in E$ is an ordered pair (X, Y), where both $X, Y \subseteq V$ are nonempty vertex subsets.

Hyperedge $e = (X, Y)$ is directed from set X to set Y, where X is called the *tail* of e, and Y the *head* of e. We refer to these sets for hyperedge e by the functions $\text{tail}(e) = X$ and $\text{head}(e) = Y$. We also refer to the set of *in* and *out* hyperedges of a vertex v by the functions $\text{in}(v) = \{e \in E : v \in \text{head}(e)\}$ and $\text{out}(v) = \{e \in E : v \in \text{tail}(e)\}$.

Figure 1 shows how we draw a general directed hyperedge.

In the rest of the paper, the terms *hypergraph* and *hyperedge* always refer to a directed hypergraph and a directed hyperedge. We use the term *ordinary graph* to refer to a standard directed graph (the special case where all hyperedges e have $|\text{tail}(e)| = |\text{head}(e)| = 1$). We also occasionally use *edge* for hyperedge when the context is clear.

[†]Some of the literature uses the term *hyperarc* for edges in directed hypergraphs, and *hyperedge* only for undirected hypergraphs. We prefer to use hyperedge in both contexts, just as *edge* is commonly used for both directed and undirected graphs.

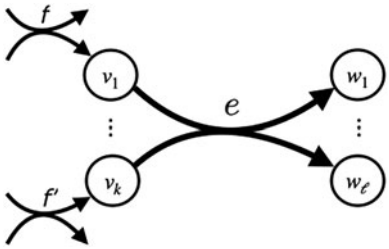

FIG. 1. A generic hyperedge e with $\text{tail}(e) = \{v_1, \ldots, v_k\}$ and $\text{head}(e) = \{w_1, \ldots, w_\ell\}$. In general, to use e in a hyperpath P, for every vertex $v \in \text{tail}(e)$ there must be a hyperedge f that precedes e in P with $v \in \text{head}(f)$.

2.2. Superpaths and hyperpaths

Hyperpaths are a generalization of directed paths to hypergraphs. In ordinary graphs, a path from a vertex s to a vertex t is a sequence of edges starting from s and ending at t, where for consecutive edges e and f, the preceding edge e must enter the vertex that the following edge f leaves. We say t is reachable from s when there is such an s, t-path from s to t.

In generalizing these notions to hypergraphs, the conditions both for when a hyperedge can follow another in a hyperpath and when a vertex is reachable from another become more involved. A hyperpath is again a sequence of hyperedges, but now for hyperedge f in a hyperpath, for every vertex $v \in \text{tail}(f)$, there must be some hyperedge e that precedes f in the hyperpath for which $v \in \text{head}(e)$. Reachability is captured by the following notion of superpath.

Definition 1 (Superpath). In a directed hypergraph (V, E), an *s, t-superpath*, for distinct vertices $s, t \in V$, is an edge subset $F \subseteq E$ such that the hyperedges of F can be ordered $e_1, e_2, \ldots, e_k$, where

(i) $\text{tail}(e_1) = \{s\}$,
(ii) for each $1 < i \leq k$,

$$\text{tail}(e_i) \subseteq \{s\} \cup \bigcup_{1 \leq j < i} \text{head}(e_j),$$

(iii) and $t \in \text{head}(e_k)$.

For an s, t-superpath, we call s its *source* vertex and t its *sink* vertex, and we say t is *reachable* from s. ■

The term *superpath* was introduced in Krieger and Kececioglu (2021), and is intended to convey that it is a *superset* of a hyperpath (which we define next). The concept of superpath is worth distinguishing, since as we will see in Section 3.2, superpaths are directly amenable to representation by the inequalites of an integer linear program, whereas hyperpaths are not.

We now define hyperpaths in terms of superpaths. A set S is *minimal* with respect to some property X if S satisfies X, but no proper subset of S satisfies X.

Definition 2 (Hyperpath). An *s, t-hyperpath* is a minimal s, t-superpath. ■

In other words, a hyperpath P is a superpath for which removing any hyperedge $e \in P$ leaves a subset $P - \{e\}$ that is no longer a superpath.

Figures 2–4 from Section 4.4 illustrate hyperpaths from cellular reaction networks.

The above definitions for superpath and hyperpath are essentially equivalent to the notions of B-connectivity and B-path in the original paper of Gallo et al. (1993) (though see Nielsen and Pretolani, 2001), while being more explicit. A similar definition of hyperpath is also given in Carbonell et al. (2012).

2.3. Cycles in hyperpaths

Hyperpath P contains a *cycle* if, for every ordering $e_1, \ldots, e_k$ of its hyperedges satisfying properties (i)–(iii) in the definition of superpath, P contains some hyperedge f with a vertex in $\text{head}(f)$ that also occurs in $\text{tail}(e)$ for a hyperedge e preceding f in the ordering. For the particular ordering $e_1, \ldots, e_k$ of P, this edge f creates a cycle by touching with its head the tail of an earlier edge e. In a sense this portion of $\text{head}(f)$ that touches $\text{tail}(e)$ is not strictly necessary (for reachability within the hyperpath) as $\text{tail}(e)$ must already be covered by prior hyperedges in the ordering, but it induces a cyclic dependency.

Figures 3 and 4 from Section 4.4 show two examples of hyperpaths with cycles in real reaction networks. In practice, cyclic hyperpaths are unavoidable: there are source-sink instances in cellular reaction networks where every s, t-hyperpath is cyclic, as discussed later in Section 4.2.

We mention that whether a hyperpath contains a cycle can be efficiently determined through the following construction. Given a hyperpath consisting of hyperedge set P that touches vertex set V, construct an ordinary directed graph $G = (P \cup V, E)$, where the vertices of G correspond to every hyperedge in P and every vertex in V, and where the edge set E of G has directed edges (v, e) and (e, w) for every hyperedge $e \in P$, every vertex $v \in \text{tail}(e)$, and every vertex $w \in \text{head}(e)$. Then hyperpath P contains a cycle if and only if graph G contains a cycle. Since acyclicity of an ordinary directed graph can be tested in linear time (see Cormen et al., 2009, pp. 612–614), this yields an algorithm for detecting whether a hyperpath is cyclic (and if so, identifying one of its cycles) that runs in time linear in the *size* of the hyperpath, as measured by the sum of the cardinalities of the tail- and head-sets of its hyperedges.

2.4. Shortest hyperpaths

We now come to our computational problem. When the hyperedges of a hypergraph all have real-valued weights that are strictly positive, an s, t-superpath of minimum total weight must be minimal. (If it is not, deleting hyperedges will yield another s, t-superpath of strictly smaller weight, contradicting its optimality.) Hence for strictly positive edge weights, a minimum-weight superpath is a minimum-weight hyperpath. This leads to the following definition of the shortest hyperpath problem.

For an edge weight function $\omega(e)$, we extend ω to edge subsets $F \subseteq E$ by

$$\omega(F) \;:=\; \sum_{e \in F} \omega(e)\,.$$

Below, *positive* edge weights means $\omega(e) > 0$ for all hyperedges e.

Definition 3 (Shortest Hyperpaths). The *Shortest Hyperpaths* problem is the following. Given a directed hypergraph (V, E), positive edge weight function $\omega : E \to \mathcal{R}$, source $s \in V$, and sink $t \in V - \{s\}$, find

$$\underset{F \subseteq E}{\text{argmin}} \left\{ \omega(F) \;:\; F \;\text{ is an } s, t\text{-superpath} \right\}. \tag{1}$$

This s, t-superpath of minimum weight is a shortest s, t-hyperpath. ■

We mention that Shortest Hyperpaths can be extended to *nonnegative* edge weights $\omega(e) \geq 0$ through postprocessing, as follows. First find an s, t-superpath S of minimum total weight under nonnegative ω, and then repeatedly remove any hyperedge $e \in S$ with $\omega(e) = 0$ whose deletion preserves reachability of t from s in $S - \{e\}$. This yields a final minimal superpath $P \subseteq S$, where P is now a shortest s, t-hyperpath. Testing a zero-weight hyperedge for removal can be done in linear time (Krieger and Kececioglu, 2021, 2022a). We define Shortest Hyperpaths using *positive* edge weights as it leads to a cleaner problem statement—and a more elegant integer linear programming formulation. (For a discussion of *negative* edge weights, see Section 5, under further research.)

We note that Shortest Hyperpaths with *multiple sources* and *multiple sinks* can be reduced to the above version with a single source and sink, as follows. To find a hyperpath that starts from a *set* of sources $S \subseteq V$, simply add a new source vertex s to the hypergraph together with a single hyperedge $(\{s\}, S)$ of zero weight, and equivalently find a hyperpath from the single source s. To find a hyperpath that reaches *all* vertices in a *set* of sinks $T \subseteq V$, add a new sink vertex t, a zero-weight hyperedge $(T, \{t\})$, and equivalently find a hyperpath to the single sink t. To find a hyperpath that reaches *some* vertex in a set of sinks $T \subseteq V$, add new sink vertex t, zero-weight hyperedges $(\{v\}, \{t\})$ from all $v \in T$, and again equivalently find a hyperpath to the single sink t. Thus all these versions are captured by the single-source and -sink case above.

For a cellular reaction network, on choosing *uniform edge weights* $\omega(e) = 1$, Shortest Hyperpaths (with an appropriate s and t and their incident hyperedges as described above) yields a series of reactions that produces all the targets starting from the sources (where for each reaction in the series, all its input reactants are formed as output products of prior reactions), using the fewest possible reactions. This is the *most*

parsimonious set of reactions producing all targets from the sources. With *non-uniform edge weights* under a simple likelihood model where $\omega(e) = -\log P(e)$ for probability $P(e)$ of reaction e occurring given that all its reactants are present, and assuming independence of reactions, Shortest Hyperpaths finds the *most likely* set of reactions that produces the targets from the sources.

Shortest Hyperpaths is NP-complete (Italiano and Nanni, 1989), so there is likely no algorithm that finds optimal solutions and is efficient in the worst-case. Shortest Hyperpaths remains NP-complete even when the underlying hypergraph is acyclic, has unit edge weights $\omega(e) = 1$ for all hyperedges e, all hyperedges e have singleton head-sets with $|\text{head}(e)| = 1$, and all tail-sets have $|\text{tail}(e)| \leq 2$. (This last restriction follows from the reduction in Krieger and Kececioglu (2022b) for Minimum Hyperedge Factory.) On the other hand, Shortest Hyperpaths is polynomial-time solvable for the class of hypergraphs where all hyperedges e have $|\text{tail}(e)| = 1$, as the efficient hyperpath heuristic of Krieger and Kececiogu (2021, 2022a) finds optimal hyperpaths for singleton-tail hypergraphs.

We also mention that by restricting to the following *doubly-reachable subgraph*, we can often significantly reduce the size of an instance of Shortest Hyperpaths in practice, while not altering its set of solutions. A hyperedge e is *doubly-reachable* from source s and sink t if all vertices in tail(e) are reachable in the forward direction from s through superpaths, and some vertex in head(e) is reachable in a backward sense from t, where we consider t to be backward-reachable from itself, and recursively, for a hyperedge f, if some vertex in head(f) is backward-reachable from t then so are all vertices in tail(f). The doubly-reachable subgraph of the input hypergraph consists of all doubly-reachable hyperedges and the vertices they touch. This subgraph safely contains all s, t-hyperpaths, is typically much smaller than the input hypergraph, and can be found in linear time (Krieger and Kececiogu, 2021, 2022a).

The next section casts Shortest Hyperpaths as a new integer linear programming problem, allowing us to leverage powerful techniques for solving such problems in practice, and compute optimal hyperpaths even for large real-world reaction networks.

3. COMPUTING SHORTEST HYPERPATHS BY INTEGER LINEAR PROGRAMMING

We now formulate Shortest Hyperpaths as a discrete optimization problem known as an integer linear program, using a new characterization of superpaths in terms of cuts. This integer linear programming formulation is the first in the literature that captures fully-general hyperpaths with cycles.

3.1. Characterizing superpaths via cuts in hypergraphs

We can give a surprisingly clean characterization of superpaths in terms of cuts. An s, t-*cut* of a hypergraph is a bipartition $(C, \overline{C})$ of its vertices V, for nonempty subsets $C \subseteq V$ and $\overline{C} := V - C$, where source $s \in C$ and sink $t \in \overline{C}$. We call C the *source side* and $\overline{C}$ the *sink side* of the cut, and often refer to a cut by just specifying its source side C.

A hyperedge e *crosses* s, t-cut C if $\text{tail}(e) \subseteq C$ and $\text{head}(e) \not\subseteq C$. In other words, for a hyperedge to cross a cut, all its tail vertices must be on the source side, while at least one head vertex must be on the sink side. We say an edge subset $F \subseteq E$ crosses an s, t-cut if at least one hyperedge $e \in F$ crosses the cut.

A remarkable aspect of the following characterization theorem is that it gives an equivalent representation of superpaths without requiring the *ordering* of hyperedges inherent to the definition of superpaths. This characterization in terms of cuts is the key that will enable us to find shortest hyperpaths via integer linear programming.

Theorem 1 (Characterizing Superpaths by Cuts). A set F of hyperedges is an s, t-superpath if and only if F crosses every s, t-cut.

Proof. To prove the forward implication, take an ordering of the hyperedges of s, t-superpath F that satisfies the definition of superpath, and an arbitrary s, t-cut C. In the ordering of F, consider the first hyperedge e such that $\text{head}(e) \not\subseteq C$, which must exist since F reaches $t \notin C$.

We claim $\text{tail}(e) \subseteq C$. To see this, notice every hyperedge $f \in F$ that precedes e in the ordering of F must have $\text{head}(f) \subseteq C$ (otherwise e is not the first with $\text{head}(e) \not\subseteq C$). Then by the definition of superpath,

$$\text{tail}(e) \;\subseteq\; \{s\} \;\cup \bigcup_{f \in F \,:\, f \text{ precedes } e} \text{head}(f) \;\subseteq\; C\,.$$

So tail$(e) \subseteq C$, while head$(e) \not\subseteq C$. Thus hyperedge e crosses cut C, which implies F crosses C as well, proving the forward implication.

For the reverse implication, we prove the contrapositive. Suppose F is not an s, t-superpath. Collect the set R of all vertices reachable from s in F. While $s \in R$, notice $t \notin R$ (otherwise F reaches t from s, contradicting that F is not an s, t-superpath). Thus $(R, \overline{R})$ is an s, t-cut. Furthermore F does not cross cut R (since $e \in F$ crossing R would contradict that R holds all vertices reachable in F), which proves the reverse implication. ■

Interestingly, Gallo et al. (1993) in their seminal paper use a different definition of edge crossing in a characterization theorem for hyperedge sets that cross all cuts, that led not to a characterization of superpaths, but to a structure they call a B-reduction.

3.2. Representing superpaths by linear inequalities

We now formulate Shortest Hyperpaths as an integer linear programming problem. In general, an integer linear program (ILP) is a mathematical optimization problem over integer-valued variables that maximizes a linear function of these variables, subject to constraints that are linear inequalities in the variables. The key to the formulation is to represent the set of all s, t-superpaths in a hypergraph (V, E) by linear inequalities.

The *variables* of our ILP encode the hyperedges in a superpath $F \subseteq E$. For every hyperedge $e \in E$, there is a variable x_e, where $x_e \in \{0, 1\}$. An assignment of values to these variables encodes a superpath F by $x_e = 1$ if and only if $e \in F$. We represent the collection of all variables in the ILP by a vector $x = (x_e)_{e \in E}$, where $x \in \{0, 1\}^{|E|}$.

The *constraints* of the ILP ensure that an assignment of values to the variables actually encodes an s, t-superpath. The domain $\mathcal{D}$ of the ILP is all assignments of values to variables x that satisfy the constraints. For our ILP, the domain is

$$\mathcal{D} := \left\{ x \in \{0, 1\}^{|E|} \ : \ \forall s,t\text{-cuts } C \sum_{e \in E \,:\, e \text{ crosses } C} x_e \ \geq \ 1 \right\}. \tag{2}$$

This has a constraint for every s, t-cut of the hypergraph, which is a linear inequality in the variables x_e. Notice that this inequality for a cut C is satisfied if and only if at least one hyperedge e crossing C has $x_e = 1$. Equivalently, the set $F \subseteq E$ encoded by x must contain at least one hyperedge $e \in F$ crossing C. Thus, assignments $x \in \mathcal{D}$ encode edge subsets F that cross every s, t-cut.

Consequently, by Theorem 1, the domain $\mathcal{D}$ of the ILP in equation (2) is exactly the set of all s, t-superpaths in the hypergraph.

The *objective function* of the ILP is to minimize $\sum_{e \in E} \omega(e)\, x_e$, for fixed positive edge weights $\omega(e) > 0$, which is a linear function of the variables x. For $x \in \mathcal{D}$, the value of this objective function is the total weight of the hyperedges in superpath F encoded by x. We can write this objective function as a dot product $\omega \circ x$, where ω is now a vector of positive edge weights.

Finally, our *integer linear program* is to compute,

$$\operatorname{argmin} \left\{ \omega \circ x \ : \ x \in \mathcal{D} \right\}. \tag{3}$$

Since domain $\mathcal{D}$ is all s, t-superpaths, this is equivalent to relation (1), so a solution to this integer linear program is a shortest s, t-hyperpath.

3.3. Solving the integer linear program by a cutting plane algorithm

For a hypergraph of n vertices and m hyperedges, the integer linear program given by relation (3) above has m variables and $\Theta(2^n)$ constraints (corresponding to the number of s, t-cuts). Thus for a large hypergraph, we cannot even feasibly write down the corresponding ILP, due to its exponentially-many constraints. Nevertheless, we can actually compute optimal solutions to this full ILP in practice, even for large hypergraphs, using an approach known as a cutting-plane algorithm.[†]

[†]In combinatorial optimization, the term *cutting-plane algorithm* traditionally refers to an algorithm that tackles an integer linear program by solving a series of *linear programs* (see Cook et al., 1998, pp. 234–235). Here we use the term to refer to an algorithm that instead solves a series of *integer programs*. Rather than referring to our approach as ''cutting-plane-like,'' we simply call it a cutting-plane algorithm, while recognizing this distinction.

A *cutting-plane algorithm* computes over a subset of the constraints of the full integer linear program, and solves a series of less-constrained problems, stopping once it detects it has an optimal solution to the full ILP. The key to a cutting-plane algorithm is an efficient *separation algorithm*, which for a given solution x to the current ILP, reports whether x satisfies all constraints of the full ILP, and if x does not, returns a constraint violated by x. (This violated constraint is a hyperplane, called a cutting plane, that separates x from the domain of the full ILP.) For our above ILP for Shortest Hyperpaths, this proceeds as follows.

(1) Let $\mathcal{I}$ be an initial set of inequalities, containing a subset of the inequalities from the full ILP, and let $\mathcal{S} := \mathcal{I}$ be the current set of inequalities.
(2) Solve the ILP restricted to the inequalities in $\mathcal{S}$, and let x^* be the optimal solution to this current ILP.
(3) Run the separation algorithm to efficiently find an s, t-cut C that is not crossed by the hyperedges e with $x^*_e = 1$, if such a cut exists.
(4) If the separation algorithm found a cut C not crossed by x^*, add the new cut-inequality given by C to set $\mathcal{S}$, and go back to Step (2).
(5) Otherwise, the hyperedges in x^* cross every s, t-cut. Halt and output the current solution x^*.

This starts with an initial set of inequalities $\mathcal{I}$, and iteratively adds inequalities to this set as it finds cuts that are not crossed by solutions to prior ILPs.

While this algorithm does not directly solve the full ILP for Shortest Hyperpaths, but instead works on a subset of its constraints, it is nevertheless guaranteed to find an optimal hyperpath, as we show next.

Theorem 2 (Cutting Plane Algorithm Solves Shortest Hyperpaths). The cutting-plane algorithm always halts, and outputs a shortest s, t-hyperpath.

Proof. The cutting-plane algorithm halts on every input, as each iteration adds an inequality to the ILP corresponding to a new s, t-cut, which is not crossed by the current solution x^* and hence is distinct from the cuts added by prior iterations, which x^* does cross, while there are a finite number of such cuts.

When the algorithm halts and outputs its final solution x^* at Step (5), let F be the corresponding set of hyperedges e with $x^*_e = 1$. Set F crosses every s, t-cut, so by Theorem 1 this F is an s, t-superpath. Furthermore, this final solution x^* is optimal for an ILP that is a less-constrained minimization problem than the full ILP. Hence the objective function value of x^* is a lower bound on the value of an optimal solution to the full ILP. Thus F is a minimum-weight s, t-superpath, or equivalently, a shortest s, t-hyperpath. ■

As indicated by the proof, the cutting-plane algorithm at most adds an inequality for every s, t-cut, so in the worst-case it solves the full ILP after 2^n iterations, given a hypergraph with n vertices. In practice, however, it finds a shortest hyperpath on solving a vastly smaller ILP, as shown later in Section 4.3.

An important observation that follows from the above proof is that the objective function values of successive solutions x^* found by the cutting-plane algorithm are monotonically increasing, and if we stop the algorithm at any point, the value of its current solution x^* yields a lower bound on the length of a shortest s, t-hyperpath. Consequently, given a candidate hyperpath P obtained by a fast heuristic (such as in Krieger and Kececiogu, 2021, 2022a), the gap between this lower bound and the length of P yields an upper bound on how far P is from optimal. If the gap is fortuitously zero, by the lower bound matching the length of P, this verifies P is optimal. Such an approach that leverages a heuristic can enable us to find a provably-optimal hyperpath even before the full cutting-plane algorithm terminates.

The simplest possible realization of the above cutting-plane algorithm starts with $\mathcal{I} = \emptyset$, and uses the following *trivial separation algorithm*. Simply collect the set R of vertices reachable from s by current solution x^* along hyperedges e with $x^*_e = 1$. This reachable set R can be computed in linear time (Krieger and Kececiogu, 2021, 2022a). If $t \in R$, then x^* encodes an s, t-superpath, which crosses every s, t-cut. Otherwise $(R, V - R)$ is an s, t-cut that x^* does not cross, which yields a cut-inequality violated by x^*. In practice, this elementary version of the cutting-plane algorithm requires a huge number of iterations before it finds an optimal solution, making it intolerably slow. Instead, we develop a more sophisticated version that both identifies a stronger set $\mathcal{I}$ and uses a more powerful separation algorithm.

The next section specifies the initial inequalities $\mathcal{I}$ we use in practice, which are crucial to its success. Then Section 3.5 presents an efficient separation algorithm that finds multiple violated inequalities, which

are all added to the ILP at the current iteration. As demonstrated in Section 4, together these enhancements enable the cutting-plane algorithm to solve large real-world instances from actual cellular reaction networks to optimality.

3.4. Strengthening the initial integer program

We can markedly reduce the number of iterations of the cutting-plane algorithm by seeding it with a strong set of initial inequalites $\mathcal{I}$. We start with $\mathcal{I}$ consisting of both the structure-based and distance-based inequalities that we describe next. This choice yields a small initial set $\mathcal{I}$, whose number of inequalities grows at most linearly with the size of the input hypergraph.

3.4.1. Structure-based inequalities. We now define a concise class of inequalities based on *structural* properties of hyperpaths, which has the following three subclasses.

The *tail-covering inequalities* ensure that for any hyperedge chosen by a solution to the ILP, its tail set is covered by the head sets of other chosen hyperedges (corresponding to condition (ii) in Definition 1 for a superpath). More formally, for every hyperedge $e \in E$, and every vertex $v \in \text{tail}(e) - \{s\}$, we have the inequality,

$$\sum_{f \in \text{in}(v)} x_f \;\geq\; x_e \,. \tag{4}$$

The *head-hitting inequalities* ensure that for a hyperedge e chosen by a solution, its head intersects (or ''hits'') the tail of another chosen hyperedge (following from the minimality condition in Definition 2 for a hyperpath, since otherwise e can be safely trimmed while maintaining reachability). More formally, for every hyperedge $e \in E$ with $t \notin \text{head}(e)$, we have the inequality,

$$\sum_{f \neq e \,:\, \left|\text{head}(e) \,\cap\, \text{tail}(f)\right| \,\geq\, 1} x_f \;\geq\; x_e \,. \tag{5}$$

The *target-production inequality* ensures that target t is reached by hyperedges chosen by the solution (corresponding to condition (iii) in Definition 1 for an s,t-superpath). Formally this is the inequality,

$$\sum_{f \in \text{in}(t)} x_f \;\geq\; 1 \,. \tag{6}$$

Together the above *structure-based inequalities* drive the solution found by the cutting-plane algorithm toward having the connected structure of a hyperpath (whereas without them, the solutions over many iterations tend to be disconnected hyperedges that cross the current set of cuts). As shown later in Section 4.6, adding this class of inequalities to the initial ILP dramatically reduces the number of iterations of the cutting-plane algorithm, greatly improving its running time.

For a hypergraph with total tail-set cardinality $t := \sum_{e \in E} \left|\text{tail}(e)\right|$, these three structure-based subclasses together have $\Theta(t)$ inequalities.

3.4.2. Distance-based inequalities. Next we define a concise class of cut-inequalities that are based on *distances* of vertices from the source, which are efficiently computable. We first define this distance-based class for ordinary graphs, and then generalize it to hypergraphs.

Ordinary graphs. For an ordinary directed graph with source s and sink t reachable from s, let $D(v)$ be the length of a shortest s,v-path for every vertex v reachable from s. Over these reachable vertices v other than s with distance $D(v) \leq D(t)$, let $d_1 < \cdots < d_k$ be the sorted set of their unique distances $D(v)$. Define a sequence of s,t-cuts $C_1 \subset \cdots \subset C_k$ associated with these unique distances, for $1 \leq i \leq k$, by

$$C_i \;:=\; \{s\} \cup \left\{v \in V \;:\; D(v) \;<\; d_i\right\}. \tag{7}$$

Finally, denote the family of these s,t-cuts by $\mathcal{C} := \{C_1, \ldots, C_k\}$, which we call the *distance-based cuts*.

In Section 3.4.3, we prove in Theorem 3 that this class of distance-based cuts is sufficiently strong for ordinary graphs to make the small integer program with cut-inequalities just from this class have an optimal

solution value equal to the length of a shortest s, t-path. (In other words, solving this restricted ILP yields a tight lower bound on the shortest-path length whose gap is zero for ordinary graphs.) Later in Section 4.6, we show experimentally that the generalization of this class to hypergraphs, which we give next, is strong as well.

The number of cuts in family $\mathcal{C}$ above is less than the number of vertices. Moreover, for ordinary graphs with nonnegative edge weights, Dijkstra's single-source shortest-path algorithm (see Cormen et al., 2009, pp. 658–659) in the course of its computation computes distance $D(v)$ for every vertex v reachable from s with distance at most $D(t)$. Thus for a graph with n vertices and m edges, we can find the *distance-based inequalities* given by cuts $\mathcal{C}$, which constitute less than n inequalities, in the same time as running Dijkstra's algorithm, namely $O(m + n \log n)$ time.

Generalizing to hypergraphs. To generalize the distance-based cuts $\mathcal{C}$ to hypergraphs, we need both a measure of distance to a vertex in a hypergraph, and a way to efficiently compute this measure. While there does not appear to be any natural distance measure on vertices for shortest hyperpaths that corresponds to $D(v)$ in ordinary graphs, we can define a generalized vertex distance as follows.

For a hyperedge e, let an s, e-superpath be a superpath from source s that reaches all vertices in tail(e), and define an s, e-*hyperpath* to be a minimal s, e-superpath. For a hyperpedge e, let its *tail-distance* $D(\text{tail}(e))$ be the length of a shortest s, e-hyperpath. Finally, for a vertex v in a hypergraph, we can define its *vertex-distance* from source s to be,

$$D(v) \quad := \quad \min_{e \in \text{in}(v)} \left\{ D(\text{tail}(e)) + \omega(e) \right\}. \tag{8}$$

Note that computing tail-distances $D(\text{tail}(e))$ for hyperedges is at least as hard as Shortest Hyperpaths, so computing the above vertex-distances $D(v)$ is unfortunately NP-complete as well.

To make this practical, we run the efficient hyperpath heuristic of Krieger and Kececiogu (2021, 2022a), which in polynomial time computes estimated tail-distances $\widetilde{D}(\text{tail}(e))$ for all hyperedges e that are reachable from source s. We then apply these tail-distance estimates in the above definition of vertex distance, to obtain efficiently-computable estimated vertex-distances $\widetilde{D}(v)$.

Using these vertex-distance estimates $\widetilde{D}(v)$, and their unique estimated vertex-distances $\widetilde{d}_1 < \cdots < \widetilde{d}_k$, we can directly generalize our prior distance-based cuts $C_1, \ldots, C_k$, defined by equation (7), to hypergraphs. This yields the *distance-based inequalities* that our cutting-plane algorithm starts with in its initial set $\mathcal{I}$.

For a hypergraph with n vertices, this distance-based class comprises less than n inequalities.

To summarize, the *structure-* and *distance-based classes*, for an input hypergraph with n vertices and hyperedges of total tail-set cardinality t, together yield an initial constraint set $\mathcal{I}$ of $O(n + t)$ inequalities.

3.4.3. Sufficiency for ordinary graphs. We now prove that the structure- and distance-based classes together suffice for *ordinary graphs*, in the sense that solving the ILP with just these two classes of inequalities finds a shortest s, t-path. We first prove that the distance-based class by itself captures shortest path *lengths*, as the value of an optimal solution to the ILP with this class alone is the length of a shortest s, t-path. We then build on this to prove that further adding the structure-based class captures shortest *paths*, as it makes optimal solutions to the ILP now be shortest paths themselves.

Below, s, t-*path* means the standard graph-theoretic notion of a simple path from s to t in an ordinary directed graph.

Capturing path lengths. The class of distance-based inequalities is sufficient to capture *lengths* of shortest s, t-paths, as we now show. The following theorem only requires nonnegative edge weights $\omega(e) \geq 0$.

Theorem 3 (Distance-Based Cuts Capture Shortest Path Lengths). For an ordinary directed graph G with source s, sink t, and nonnegative edge weights ω, let F^* be an edge set of minimum total weight that crosses every cut in the family $\mathcal{C}$ of distance-based s, t-cuts. Then its weight $\omega(F^*)$ is the length of a shortest s, t-path in G.

Proof. We first show by induction that the weight of an edge set crossing cuts $C_1, \ldots, C_i$ is at least d_i, for all $1 \leq i \leq k$, where k is $|\mathcal{C}|$. As d_k is $D(t)$, this proves the weight of any edge set crossing all cuts in $\mathcal{C}$ is at least $D(t)$.

For the basis with $i = 1$, consider cut $C_1 = \{s\}$, and let $e = (s, v)$ be a minimum-weight edge leaving s. Consider any set F crossing C_1, which must contain an edge f leaving s. We have,

$$\omega(F) \geq \omega(f) \geq \omega(e) = D(v) = d_1\,,$$

so the basis holds.

For the inductive step with $i > 1$, let F be any edge set that crosses cuts $C_1, \ldots, C_i$. Set F must contain an edge $f = (u, v)$ that crosses cut C_i. Notice that $D(u) < d_i$, which implies $D(u) = d_j$ for some $j < i$. Let $F' \subseteq F - \{f\}$ be the subset of F that crosses cuts $C_1, \ldots, C_j$. We have,

$$\omega(F) \geq \omega(F') + \omega(f) \tag{9}$$

$$\geq D(u) + \omega(u, v) \tag{10}$$

$$\geq D(v) \tag{11}$$

$$\geq d_i\,. \tag{12}$$

In the above, inequality (9) is by nonnegativity of edge weights. Inequality (10) follows from the inductive hypothesis on $j < i$, noting $f = (u, v)$. Inequality (11) is due to a shortest s, u-path followed by edge (u, v) yielding an s, v-path. Inequality (12) follows from the definitions of edge (u, v) and cut C_i. So the induction holds.

Thus any edge set F that crosses all cuts in family $\mathcal{C}$ has weight $\omega(F) \geq D(t)$. Now consider an edge set F^* of minimum weight $\omega(F^*)$ crossing family $\mathcal{C}$, and a shortest s, t-path P^*. Notice that P^* crosses all cuts in $\mathcal{C}$, since for any s, t-cut the first edge in P^* entering a vertex on the sink-side (which must exist as P^* reaches t) crosses the cut. Hence $\omega(F^*) \leq \omega(P^*) = D(t)$. Together these two inequalities imply $\omega(F^*) = D(t)$, which proves the theorem. ■

So by Theorem 3, solving an integer linear program containing just the distance-based inequalities gives the length of a shortest s, t-path. An optimal solution, however, can have disconnected edges, and is not necessarily itself a path. We next show that adding the structure-based inequalities makes optimal solutions be shortest paths.

Capturing shortest paths. We now add the class of structure-based inequalities. Our next theorem states that for ordinary graphs, the structure- and distance-based classes together suffice in the following sense to capture shortest *paths*. Solving the small integer program containing just these two constraint classes—which for an ordinary graph with n vertices and m edges has m variables, less than n distance-based inequalities, and at most $2m$ structure-based inequalities—yields a shortest s, t-path.

As an aside, we note that the following result gives a concise ILP formulation of shortest paths in ordinary graphs that is different from the standard linear-programming-based formulations in the literature (see Wolsey, 1998, pp. 42–43). Of course, simply writing down the ILP requires us to have already solved the shortest path problem, just in determining the distance-based cuts. Nevertheless, this ILP formulation could provide a convenient way to tackle NP-complete variants of shortest path problems, by casting them as ILPs with additional side constraints in order to solve them to optimality in practice.

Formally, the integer linear program $\mathcal{P}$ referred to in Theorem 4 below has for its *constraints* both the structure-based inequalities (4)–(6) and the distance-based inequalities that correspond to cut-inequalities (2) for the distance-based cuts (7), specialized for an ordinary graph whose edges e all have singleton tail- and head-sets and positive weights $\omega(e) > 0$. The *variables* and *objective function* for $\mathcal{P}$ are the same as for integer program (3).

Theorem 4 (Structure- and Distance-Based Inequalities Capture Shortest Paths). For an ordinary directed graph G with positive edge weights, consider the corresponding integer linear program $\mathcal{P}$ whose constraints are just the structure- and distance-based inequalities. Then optimal solutions to $\mathcal{P}$ are shortest s, t-paths in G.

Proof. Let x^* be an optimal solution to the integer linear program whose constraints are just the structure- and distance-based inequalities, for an ordinary graph with positive edge weights, and let P^* be the corresponding set of edges e with $x^*_e = 1$.

We first show every s, t-path P is a minimal feasible solution to this integer linear program. To see this, notice any such P crosses every s, t-cut C, as the first edge of P entering the sink side of C (which must exist as P reaches t) crosses C. Hence every such P satisfies the distance-based inequalities. Such paths P also clearly satisfy the structure-based inequalities. Thus they are feasible solutions. Furthermore they are minimal, as any proper subset of an s, t-path violates at least one of the classes of structure-based inequalities.

Consequently, *optimality* of P^* implies the total weight of any s, t-path P satisfies $\omega(P^*) \leq \omega(P)$. Hence $\omega(P^*) \leq D(t)$.

Moreover, *minimality* holds for P^* as well (meaning P^* is a minimal feasible solution to the integer program), since with positive edge weights, removing edges from P^* while preserving feasibility yields a strictly better solution, contradicting optimality of P^*.

We next argue sink t must be reachable from source s along edges in P^*. Suppose not. By the target production inequality, there must be an edge $e \in P^*$ that enters t. Then by the tail-covering inequalities, tracing in-edges in P^* backward from e must lead to a cycle $C \subseteq P^*$, since t is not reachable from s.

The distance-based inequalities ensure P^* contains a minimal subset $Q \subseteq P^*$ that crosses all distance-based cuts. The proof of Theorem 3 then shows $\omega(Q) \geq D(t)$. Furthermore Q is acyclic, as minimality of Q implies every edge $(v, w) \in Q$ crosses a distance-based cut, and consequently has $D(v) < D(w)$. Nevertheless P^* contains cycle C. Thus P^* must properly contain Q. Positive edge weights then force $\omega(P^*) > \omega(Q)$. However, this implies $\omega(P^*) > D(t)$, a contradiction.

Hence t must be reachable from s through P^*. This implies P^* contains an s, t-path. Minimality of P^* requires P^* be an s, t-path. Optimality of P^* then ensures P^* is a shortest s, t-path. ■

While a result analogous to Theorem 4 does not hold for *hypergraphs* (since the very first ILP solved by the cutting-plane algorithm, whose constraints $\mathcal{I}$ are exactly the structure- and distance-based classes, does not in general return an s, t-hyperpath, otherwise the algorithm would immediately halt after one iteration), the above theorem for ordinary graphs does suggest these classes are strong, and indeed our experimental results in Section 4.3 show that with these two classes—and the separation algorithm described next—we typically need only three iterations of the cutting-plane algorithm to solve current cellular reaction instances to optimality in practice.

3.5. A separation algorithm leveraging distance-based cuts

Recall that the cutting-plane algorithm starts with inequalities $\mathcal{I}$, where $\mathcal{I}$ contains both the structure-based class and the distance-based class for family $\mathcal{C}$ of distance-based cuts. Since the solution x^* to the current ILP crosses all cut-inequalities in $\mathcal{S} \supseteq \mathcal{I}$ with its *active hyperedges* e where $x^*_e = 1$, solution x^* already crosses every cut in family $\mathcal{C}$. To find new cuts not crossed by x^*, the separation algorithm examines every s, t-cut $C \in \mathcal{C}$, and considers enlarging its source-side $C \supseteq \{s\}$, called *source-augmentation*, or enlarging its sink-side $\overline{C} = V - C \supseteq \{t\}$, called *sink-augmentation*, to obtain a new s, t-cut $\widehat{C}$ not crossed by x^*. Such a cut $\widehat{C}$ specifies a cut-inequality from the full ILP that is violated by current solution x^*.

We mention that the trivial separation algorithm described near the end of Section 3.3 can be viewed as a special case of this more general separation algorithm, which only performs source-augmentation of the single cut $C = \{s\} \in \mathcal{C}$, and just finds one inequality violated by x^*.

The general *separation algorithm* outlined above that augments distance-based cuts can find up to $2k$ inequalities from the full ILP that are violated by x^*, where $k = |\mathcal{C}|$ is the number of distance-based cuts, since source- and sink-augmentation can together yield two new violated inequalities for each cut in family $\mathcal{C}$. All the violated inequalities that are found for x^* are added to the current set $\mathcal{S}$ for the next iteration of the cutting-plane algorithm.

This general separation algorithm is *correct*, as argued in the next subsection on source-augmentation, in the following sense. Given current solution x^*, it either (a) returns cut-inequalities from the full ILP that are *violated* by x^*, or (b) detects and reports that x^* satisfies *all* inequalities in the full ILP.

Moreover, the general separation algorithm is also *efficient* (even though the full ILP contains exponentially-many inequalities), as shown by the time analysis at the end of this section.

We next describe source-augmentation, sink-augmentation, and then bound the running time of the resulting separation algorithm.

3.5.1. Source-side augmentation. Given current solution x^* from the cutting-plane algorithm, and cut C from the family $\mathcal{C}$ of distance-based cuts (which implies x^* crosses C), we perform source-augmentation of C to obtain a *new* s,t-cut $\widehat{C} \supset C$ not crossed by x^*, as follows. Recall that hyperedge e crosses C if $\text{tail}(e) \subseteq C$ but $\text{head}(e) \not\subseteq C$.

For a given active hyperedge e that crosses C, we enlarge C to form a new cut $\widehat{C} = C \cup \text{head}(e)$, which is the minimal enlargement of C that e no longer crosses. We then repeat this process on $\widehat{C}$ for every active hyperedge that crosses it, until we obtain a final cut $\widehat{C}$ that *no* active hyperedge of x^* crosses.

We note that this process can fail to find a new cut $\widehat{C} \supset C$ not crossed by x^*, as $\widehat{C}$ could potentially grow to encompass sink t, in which case $\widehat{C}$ is no longer an s,t-cut. When this occurs, however, there is *no* s,t-cut that properly contains the source side of the original cut C that x^* does not cross.

We also note that the above process actually finds the unique s,t-cut $\widehat{C} \supset C$ of *minimum cardinality* $\left|\widehat{C}\right|$ that x^* does not cross.

Importantly, the trivial cut $C = \{s\}$ is one of the distance-based cuts in family $\mathcal{C}$, and source-augmentation of this trivial cut by the process described above yields set $\widehat{C} \supset \{s\}$ consisting of *all* vertices reachable from s along active hyperedges. So by the reasoning in the proof of Theorem 1, this separation algorithm is *guaranteed* to find a cut not crossed by x^* whenever one exists.

This last observation verifies that our separation algorithm is *correct*: given current solution x^*, it either (a) finds an s,t-cut not crossed by x^*, or (b) detects that x^* crosses every s,t-cut, by discovering that t is reachable from s along active hyperedges.

3.5.2. Sink-side augmentation. We perform sink-augmentation of distance-based cut $C \in \mathcal{C}$, to obtain a new cut $\widehat{C} \subset C$ not crossed by x^*, as follows.

For a given active hyperedge e that crosses C, we enlarge $\overline{C} = V - C$ by moving one vertex $v \in \text{tail}(e) - \{s\}$ from C to its sink-side $\overline{C}$, yielding the new cut $\widehat{C} = C - \{v\}$. This is a minimal enlargement of $\overline{C}$ that e no longer crosses. Of course, this may cause new active hyperedges to now cross $\widehat{C}$ that did not before. To reduce the number of new hyperedges crossing $\widehat{C}$, we exploit the freedom in picking v by greedily choosing the $v \in \text{tail}(e) - \{s\}$ that causes the fewest active hyperedges to newly cross $\widehat{C}$. We repeat this process on $\widehat{C}$ for every active hyperedge crossing it, until we obtain a final cut $\widehat{C} \subset C$ that x^* does not cross.

We note that the above process can fail to find a new cut $\widehat{C}$ not crossed by x^* whose sink-side properly contains the original cut, as once an active hyperedge e crossing $\widehat{C}$ has $\text{tail}(e) = \{s\}$, there is no further reduction of the source-side of $\widehat{C}$ that x^* does not cross. In this situation, due to the freedom in choosing vertices v to move to the sink-side of $\widehat{C}$, another sequence of vertex choices might have potentially led to finding a different successful sink-augmentation $\widehat{C}$. For efficiency, our implementation does not backtrack over its choices when this occurs, and simply produces no sink-augmentation of this particular distance-based cut C, although one could exist. (In contrast, source-augmentation always finds a cut $\widehat{C} \supset C$ not crossed by x^* whenever one exists.) This does not affect correctness of the separation algorithm as argued earlier, but it can reduce the number of violated inequalities that the separation algorithm finds to add to the current ILP per iteration of the cutting-plane algorithm.

3.5.3. Time complexity. We bound the time complexity of the separation algorithm in terms of the following key parameters:

- the number of distance-based cuts k,
- the number of active hyperedges h in the current solution x^*,
- the number of doubly-reachable hyperedges m (which are discussed at the end of Section 2; informally, these are all hyperedges that are both reachable in the forward direction from the source, and reachable in a backward sense from the sink),
- the number of vertices n in the doubly-reachable subgraph (namely, the cardinality of the union of the tail- and head-sets of all doubly-reachable hyperedges), and
- the standard measure of the *size* ℓ of a directed hypergraph (from Gallo et al., 1993), restricted to its doubly-reachable hyperedges $\widehat{E}$, where

$$\ell \;:=\; \sum_{e \in \widehat{E}} \left(\left|\text{tail}(e)\right| + \left|\text{head}(e)\right| \right).$$

To aid the analysis, we make the implementation of source- and sink-augmentation more concrete.

Source-augmentation of a distance-based cut C can be implemented straightforwardly as a repeated scan of the list of active hyperedges F in solution x^*. For the current augmentation $\widehat{C} \supseteq C$, the scan checks whether hyperedge $e \in F$ crosses $\widehat{C}$. If so, it updates the current augmentation to $\widehat{C} \cup \text{head}(e)$, and performs the next scan on this updated $\widehat{C}$. These scans are repeated until verifying that no hyperedge in F crosses $\widehat{C}$, upon which this final $\widehat{C}$ is returned as the source-augmentation of cut C (or until encountering an active hyperedge e crossing $\widehat{C}$ with $t \in \text{head}(e)$, which causes unsuccessful termination for C).

To easily test whether a hyperedge crosses the current augmentation, we maintain a boolean variable for each vertex v in the doubly-reachable subgraph that records whether v is on the source-side of $\widehat{C}$. Initializing these variables when processing a given distance-based cut takes $\Theta(n)$ time.

Since each scan until termination makes at least one hyperedge e of F no longer cross $\widehat{C}$ (where e remains noncrossing during subsequent scans), this does at most $h = |F|$ scans. Each scan takes $O(\ell)$ time to test hyperedges for whether they cross the current cut $\widehat{C}$ (by checking the boolean variables for vertices in the tail- and head-sets of the hyperedge), and to update $\widehat{C}$ and its corresponding vertex variables. This spends $O(h\ell)$ time per cut C.

So for a family of k distance-based cuts, source-augmentation takes total time $\Theta(kn) + O(kh\ell)$. As $n \leq \ell$, this is $O(kh\ell)$ total time.

Sink-augmentation of a distance-based cut can be similarly implemented as a repeated scan of the active hyperedges, where for efficiency we now use the following variables:

- a boolean variable for each vertex v that indicates whether v is on the source side of the current augmentation,
- a boolean variable for each hyperedge e that records whether *all* of the tail- and head-vertices for e are on the source side of the current augmentation, and
- an integer variable for each vertex v that counts the number of active hyperedges $e \in \text{in}(v) - \text{out}(v)$ such that the boolean variable for e is true.

This integer count for v is precisely the number of active hyperedges that will newly cross the current augmentation on moving v from its source side to its sink side. Initializing these variables for vertices and hyperedges when processing a given distance-based cut takes $O(\ell)$ time.

When the scan detects that an active hyperedge e crosses the current augmentation $\widehat{C}$ (by checking the boolean vertex variables in the tail- and head-sets for e), we examine each vertex $v \in \text{tail}(e) - \{s\}$ as a candidate to move to the sink-side of $\widehat{C}$ (unless $\text{tail}(e) = \{s\}$ causes unsuccessful termination for this cut). The candidate v whose associated integer variable is minimum is then moved, updating the current augmentation to $\widehat{C} - \{v\}$. During this process, scanning all active hyperedges to test for crossing, examining their tail sets to identify the vertex to move whose count is smallest, and updating the current augmentation $\widehat{C}$ together with its associated variables for vertices and hyperedges, all take $O(\ell)$ time per scan.

As before, each scan until termination makes at least one active hyperedge no longer cross the current augmentation (and this hyperedge remains non-crossing during subsequent scans), so this performs at most h scans, spending $O(h\ell)$ time per distance-based cut. For k such cuts, this again takes $O(k\,h\ell)$ total time.

To summarize the *time complexity*, for k distance-based cuts, a solution with h active hyperedges, and a doubly-reachable subgraph of size ℓ, the separation algorithm runs in $O(k\,h\ell)$ time.

In terms of the input hypergraph, for a doubly-reachable subgraph of n vertices and m hyperedges, since $k \leq n$ and $h \leq m$ (as at worst all vertices have distinct distances and all hyperedges are active—though in practice k and h are typically several orders of magnitude smaller), this is $O(\ell\,mn)$ time worst-case.

4. EXPERIMENTAL RESULTS

We now present experimental results with a preliminary implementation of our cutting-plane algorithm, in a tool we call `Mmunin` (Krieger and Kececioglu, 2022d).[†] As the experiments show, `Mmunin` can find optimal hyperpaths in large real-world cellular reaction networks quickly, often in less than 10 seconds.

[†]The name `Mmunin` (short for ''integer-linear-programming-based cutting-plane algorithm for shortest source-sink hyperpaths'') was chosen to complement that of our tool `Hhugin` (Krieger and Kececioglu, 2022c), which implements an efficient shortest hyperpath heuristic. (Huginn and Muninn are the mythical ravens of the Norse god Odin.)

We first give details on our experimental setup, describing the datasets and our implementation. We then show, through comprehensive experiments across all of the thousands of source-sink instances from the two standard cellular reaction databases in the literature, how `Mmunin` surpasses both the state-of-the-art heuristic for *general* shortest hyperpaths (Krieger and Kececioglu, 2021, 2022a) and the state-of-the-art exact algorithm for shortest *acyclic* hyperpaths (Ritz et al., 2017). The last subsections discuss three biological examples, study how well `Mmunin` recovers known biological pathways, and investigate the effect of the various constraint classes on running time.

4.1. Experimental setup

We briefly describe the datasets and how we transform them into hypergraphs, and then give details on our implementation, including two modifications that further improve running time.

Datasets and their preparation. Our source-sink instances for Shortest Hyperpaths were formed from the two standard cellular reaction databases in the literature, following an established hypergraph construction protocol (Krieger and Kececioglu, 2022a; Ritz et al., 2017). Our `NCI-PID` dataset aggregates all 213 pathways from the US National Cancer Institute's Pathway Interaction Database (NCI-PID) (Schaefer et al., 2009), while our `Reactome` dataset aggregates all 2,516 pathways from the Reactome database (Joshi-Tope et al., 2005).

To build a hypergraph for each dataset, we map each protein or small molecule to a *vertex*, and each reaction to a *hyperedge*. In each reaction's hyperedge, its input reactants and positive regulators are in the *tail*, and its output products are in the *head*. The set S of all vertices that originally have no incoming hyperedges we call the *sources*. The set T of vertices that originally have no outgoing hyperedges we call the *targets*. For the hypergraph, we create a single *supersource* vertex s, with one outgoing hyperedge $(\{s\}, S)$ to the sources. For the hypergraph, we also create a single *sink* vertex t.

In our *single-target experiments*, each source-sink instance has a single target $v \in T$. In the instance for target v, we create the hyperedge $(\{v\}, \{t\})$ as the sole hyperedge in the hypergraph that is incoming to sink t for that instance. In this way, each s, t-instance (with supersource s and sink t) for a single-target experiment is implicitly associated with *one* target v.

Table 1 gives summaries of these hypergraphs and their source-sink instances. The `NCI-PID` hypergraph has more than 9,000 vertices and 8,000 hyperedges, while the `Reactome` hypergraph has more than 20,000 vertices and 11,000 hyperedges. In both hypergraphs, vertices have small average in-degree and out-degree. The head-sets and tail-sets of the hyperedges also tend to have small average size. Both hypergraphs contain many hyperedges that are self-loops, which cannot be trivially removed, and which would be excluded when finding shortest *acyclic* hyperpaths. Some of these self-loops are unreachable,

Table 1. Dataset Summaries

	NCI-PID		*Reactome*	
Pathways	213		2,516	
Vertices	9,009		20,458	
Hyperedges	8,456		11,802	
Self-loops	40		433	
Unreachable self-loops	14		32	
Sources	3,200		8,296	
Targets	2,636		5,066	
Reachable targets	2,220		2,432	
	mean	*max*	*mean*	*max*
Vertex in-degree	1.0	323	0.9	1,056
Vertex out-degree	1.7	326	1.4	1,167
Hyperedge tail-size	1.9	10	2.4	26
Hyperedge head-size	1.1	5	1.6	28
Forward-reachable set	6,169	6,169	4,645	4,645
Backward-reachable set	1,198	2,863	4,027	7,021
Doubly-reachable set	756	1,836	929	1,725

meaning at least one tail vertex has only this same hyperedge as an incoming hyperedge; in such a case, we resolve this cycle in the self-loop by adding such tail vertices to the source set.

Out of all possible targets $v \in T$ in these hypergraphs, only a smaller number are actually *reachable targets*, where an s, t-hyperpath exists from supersource s to sink t for that target v. (Each such hyperpath corresponds to a series of reactions that creates the target molecule v from the set of source compounds S.) As listed in Table 1, both `NCI-PID` and `Reactome` have more than 2,000 reachable targets, each corresponding to an s, t-instance that has an s, t-hyperpath.

For each s, t-instance, we first reduce the hypergraph to its doubly-reachable subgraph for that instance, as mentioned at the end of Section 2. (This subgraph is induced by the *doubly-reachable hyperedges*, which for an s, t-instance are those that are both forward-reachable from s and backward-reachable from t.) This reduction yields a significantly smaller hypergraph, without altering the set of solutions for that s, t-instance. As shown in Table 1, it reduces the average number of hyperedges for an instance to less than 800 for `NCI-PID` (down from over 8,000), and to less than 1,000 for `Reactome` (down from over 11,000).

All hyperedges are given *unit weight*, even though the cutting-plane algorithm handles positive edge weights. (In the reaction databases, reaction rates for many reactions are unknown.) A shortest hyperpath in these unit-weight hypergraphs corresponds to a pathway that produces the target from the sources using the fewest reactions—the most parsimonious pathway for that target.

Implementation. The cutting-plane algorithm and its separation algorithm are both implemented in Python 3.8, comprising around 2,000 lines of code. All procedures are implemented as described in Section 3, with a few exceptions.

First, for an s, t-instance, we use a procedure from `Hhugin` (Krieger and Kececioglu, 2022c) to compute the doubly-reachable subgraph, which contains only those hyperedges from the input hypergraph that can possibly be in any s, t-hyperpath. Next, the initial distance-based inequalities require approximate tail-distances from `Hhugin`, but to improve running time, the cutting-plane algorithm begins with only the distance-based inequality given by cut $C = \{s\}$. We start execution of `Hhugin` and the cutting-plane algorithm in parallel, and at each iteration, the cutting-plane algorithm checks if `Hhugin` has terminated, in which case it computes the full set of distance-based inequalities using the approximate tail-distances returned by `Hhugin`, and adds them to the current constraint set $\mathcal{S}$.

Lastly, at each iteration the cutting-plane algorithm compares the objective function value of its current solution to the length of `Hhugin`'s heuristic hyperpath $\widetilde{P}$, and if they are equal, returns $\widetilde{P}$ (since we then know $\widetilde{P}$ is optimal, as this objective function value is a lower bound on the length of a shortest hyperpath).

We also made one modification to `Hhugin` to improve its solution quality, while only slightly increasing its running time. In the original description of `Hhugin` (Krieger and Kececioglu, 2021, 2022a), once a hyperedge is extracted from the heap in the overall while-loop, its in-edge list becomes frozen, and further hyperedges are not added to it. We changed this rule, so that now the in-edge list for a hyperedge no longer on the heap can continue to grow as more hyperedges are subsequently extracted.

Source code for this cutting-plane algorithm implementation, including all datasets, is available free for research use at `http://mmunin.cs.arizona.edu` (Krieger and Kececioglu, 2022d).

4.2. Comparing suboptimality of alternate methods

`Mmunin` outperforms the state-of-the-art hypergraph methods for pathway inference via hyperpaths: both the efficient *heuristic* for general shortest hyperpaths with *cycles*, called `Hhugin` (Krieger and Kececioglu, 2021, 2022a); and the MILP-based *exact algorithm* for shortest *acyclic* hyperpaths (Ritz and Murali, 2014; Ritz et al., 2017), which we call `AcycMILP`. We compare these three methods across all instances from `Reactome` and `NCI-PID`.

Table 2 summarizes statistics from this comparison for *reachable* instances, where an s, t-hyperpath exists. The table reports for how many of these instances a tool found a suboptimal solution, and their *optimality gap* on these suboptimal instances, which is the difference between the number of hyperedges in their suboptimal hyperpath and the shortest hyperpath. (As `Mmunin` implements an exact algorithm for fully-general hyperpaths, it is never suboptimal, and its optimality gap is always zero.) When `AcycMILP` was suboptimal, it found no solution for a reachable instance, which is indicated by an infinite optimality gap.

Table 2. Suboptimality of Alternate Hyperpath Methods

	NCI-PID		Reactome	
Reachable instances	2,220		2,432	
AcycMILP suboptimal	38		22	
Hhugin suboptimal	23		0	
Mmunin suboptimal	(none)		(none)	
	median	*max*	*median*	*max*
AcycMILP optimality gap	(∞)	(∞)	(∞)	(∞)
Hhugin optimality gap	1	6	0	0
Mmunin optimality gap	(0)	(0)	(0)	(0)

For all these instances, Mmunin computes an optimal hyperpath in less than 30 minutes, while allowing cycles for the first time. Mmunin surpasses AcycMILP on the 22 Reactome instances and 38 NCI-PID instances where all s, t-hyperpaths are cyclic, causing AcycMILP to not return *any* solution. Mmunin outperforms Hhugin on the 23 NCI-PID instances where Hhugin finds a *suboptimal* solution.

Notably, Mmunin not only finds an optimal hyperpath for these instances where Hhugin is suboptimal, it even finishes its computation *before* Hhugin—showing that Mmunin (without the distance-based constraints obtained using Hhugin) actually finds an *optimal* solution for these instances faster than the heuristic finds a *suboptimal* solution. In fact, overall, we found that Mmunin is faster than Hhugin on more than 20% of all instances.

4.3. Performance for computing optimal hyperpaths

Mmunin is typically fast in practice, with a median running time across all instances of under 10 seconds. Table 3 gives performance statistics for computing optimal hyperpaths by Mmunin over all reachable instances: specifically, the number of hyperedges in the shortest hyperpath, the number of inequalities in the final ILP, the number of iterations to find an optimal solution, the running time per iteration, and the total time over all iterations. (Due to inherent randomness in the CPLEX solver we use, there is some variation in running time even for the same instance.)

The maximum running time across all instances is just under 30 minutes, demonstrating it is now feasible to find shortest hyperpaths in cellular reaction networks even for large instances with more than 10,000 hyperedges.

The number of hyperedges in the shortest hyperpath tends to be very low, with a median of 3 hyperedges for both NCI-PID and Reactome, and a maximum of 16 and 17 hyperedges, respectively. This is likely because, in current practice, the pathways that have been annotated tend to have a small number of reactions, and they tend to exhibit low overlap with other annotated pathways. As a result, the shortest hyperpath to any target using only the reactions from one annotated pathway will tend to have a small number of hyperedges (and any hyperpath outside the annotated pathway would tend to be longer).

The number of iterations for Mmunin to find an optimal solution also tends to be very low—around 2 or 3 iterations—but it can require more than 1,500 iterations. Note that at each iteration, the cutting-plane

Table 3. Performance of Mmunin

	NCI-PID		Reactome	
Reachable instances	2,220		2,432	
	median	*max*	*median*	*max*
Hyperedges in shortest hyperpath	3	16	3	17
Final number of inequalities	2,507	11,767	158	7,660
Number of iterations	3	1,598	3	874
Time per iteration (seconds)	2.2	12.4	2.5	12.2
Total time (seconds)	7.4	1,788	8.5	776

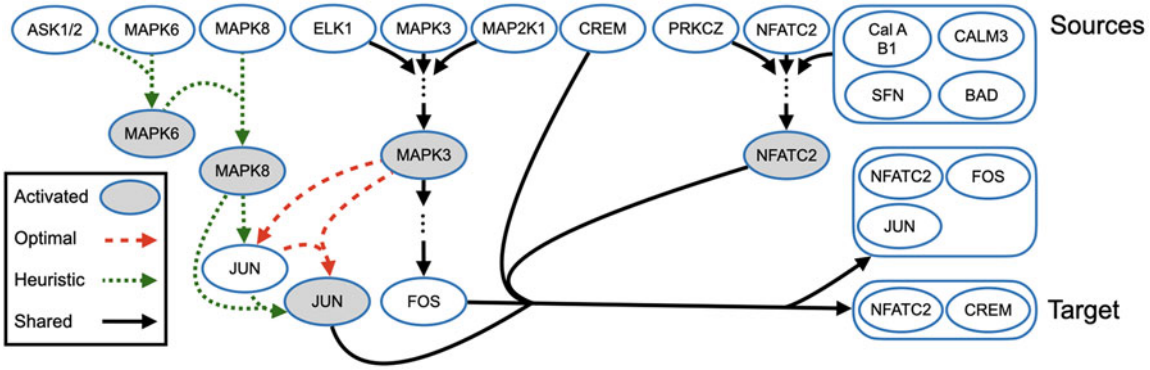

FIG. 2. Comparison of hyperpaths whose target is the NFATC2/CREM complex. Sources are shown across the top of the figure, and the target is in the bottom right corner. Hyperedges drawn in dashed red are unique to `Mmunin`'s hyperpath, hyperedges in dotted green are unique to `Hhugin`'s hyperpath, and hyperedges in solid black are common to both hyperpaths. Eight hyperedges that are shared by both hyperpaths have been replaced by ellipses. Hyperedges from MAPK8 to JUN and from MAPK3 to JUN denote transcription and translation of JUN. Vertices with gray fill denote the activated form of a protein.

algorithm must solve an ILP containing thousands of variables and inequalities. Even when an instance requires many iterations, on average each iteration is fast, with a maximum time per iteration around 12 seconds.

The number of inequalities in the final ILP varies widely between instances and between datasets, with median values of around 2,500 inequalities for `NCI-PID` and around 150 inequalities for `Reactome`. The number of inequalities for an instance is mainly a function of both the size of its doubly-reachable subgraph (which determines how many structure-based inequalities are in the initial ILP), and the number of iterations of the cutting-plane algorithm (which strongly affects how many cut-based inequalities are added to the ILP). The difference in median values between the datasets comes largely from their doubly-reachable hyperedges, since most instances require very few iterations of the cutting-plane algorithm. The maximum values of more than 11,000 inequalities for `NCI-PID` and 7,000 inequalities for `Reactome` are on instances where a large number of cutting-plane iterations are needed to find an optimal solution.

4.4. Biological examples

To provide some concrete examples of shortest hyperpaths found by `Mmunin`, we discuss three biological instances (of a size that can be readily shown): an instance from `NCI-PID`, where `Mmunin` finds a shorter hyperpath than the heuristic `Hhugin`; and two instances from `Reactome` where all s, t-hyperpaths are cyclic, and `AcycMILP` finds no solution.

The NFATC2/CREM complex. The first example is drawn from the 23 instances in `NCI-PID` where `Mmunin` outperforms `Hhugin`, as it finds a suboptimal hyperpath. (We chose this particular instance because the size of its hyperpaths makes them reasonable to draw.) We note that `AcycMILP` is also optimal on this instance, as there is a shortest hyperpath that is acyclic.

Figure 2 shows the hyperpaths returned by `Mmunin` and `Hhugin` for this example, which consist of reactions involved in the deactivation of the ''nuclear factor of activated T cells 2'' (NFATC2) by the ''cAMP response element modulator'' (CREM). Sources are shown across the top of the figure, and the target is in the bottom right corner. Hyperedges drawn in a red dash are unique to `Mmunin`'s hyperpath, and hyperedges in a green dotted line are unique to `Hhugin`'s hyperpath, while hyperedges in a solid black line are common to both hyperpaths. Eight hyperedges that are shared by both pathways have been replaced in the figure by ellipses, for simplicity. All hyperedges denote biochemical reactions except the hyperedges from MAPK8 to JUN and from MAPK3 to JUN, which denote the transcription and translation of JUN. Vertices with a gray fill denote the activated form of a protein. Note that stoichiometry of the reactions is not considered in our hyperpath formulation, so this hyperpath is valid, even though two copies of NFATC2 are needed in the final reaction.

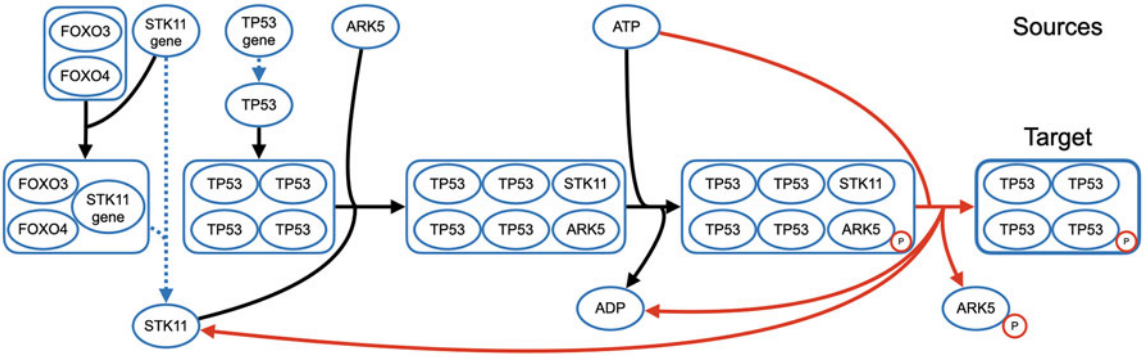

FIG. 3. Shortest hyperpath to the phosphorylated p53 tetramer. Sources are shown across the top of the figure, and the target is in the bottom right corner. The hyperedges in dotted blue represent transcription/translation events. The hyperedge in red creates a cycle in the hyperpath, as node STK11 in its head occurs in the tail of a prior hyperedge in the hyperpath.

Each hyperpath contains hyperedges from the following four NCI-PID pathways: ''Calcium signaling in the $CD4^+$ TCR pathway,'' ''RAS signaling in the $CD4^+$ TCR pathway,'' ''JNK signaling in the $CD4^+$ TCR pathway,'' and ''Nongenotropic androgen signaling.'' The hyperpaths show CREM repressing the activated form of NFATC2 before forming a ternary complex with NFATC2 and its DNA binding sites, resulting in the attenuation of transcription of T helper-1-specific cytokine genes in human medullary thymocytes (Bodor and Habener, 1998).

`Hhugin`'s hyperpath contains 13 hyperedges, while `Mmunin`'s hyperpath contains 11 hyperedges, which is optimal. Notably, hyperedges unique to `Mmunin`'s hyperpath only use vertices that are common to both hyperpaths, further simplifying it, and making it a more likely pathway.

The phosphorylated p53 tetramer. The second example was chosen from the 22 instances in `Reactome` with no acyclic hyperpath, on which `AcycMILP` finds no solution. We note that the heuristic `Hhugin` also finds an optimal hyperpath for this particular instance.

Figure 3 shows the optimal hyperpath returned by `Mmunin`, which captures the phosphorylation of ''cellular tumor antigen p53'' (TP53) by ''AMPK-related protein kinase 5'' (ARK5), causing its activation. Sources are shown across the top of the figure, and the target is in the bottom right corner. The hyperedges in dotted blue represent transcription/translation events, and the hyperedge in red shows the cycle in the hyperpath, where a node in the head of the red hyperedge is in the tail of an earlier hyperedge in the hyperpath.

This hyperpath shows the transcription and translation of key proteins involved in the formation of a complex with the TP53 homotetramer. Once this complex is formed, ARK5 is activated by phosphorylation, allowing it to directly phosphorylate p53 (Hou et al., 2011), activating p53 and allowing it to assist in DNA damage repair.

This example from `Reactome` has more than 1,600 hyperedges in the doubly-reachable subgraph, making it one of the largest instances. It is also one of the longer cycles for any `Reactome` instance, with three hyperedges involved in the cycle. Interestingly, even though this is a large instance, no alternate hyperpaths exist that reach the target for this instance.

The BMP7/BMPR2/BMPR1A/SMAD6/SMURF1 complex. The third example was also chosen from the 22 instances in `Reactome` where all source-sink hyperpaths are cyclic, hence `Mmunin` outperforms `AcycMILP`. Again `Hhugin` is optimal on these as well.

Figure 4 shows the optimal hyperpath found by both `Mmunin` and `Hhugin` for this example, which represents a signaling pathway resulting in the degradation of the ''bone morphogenetic protein receptors'' (BMPR2 and BMPR1A) in `Reactome` (Murakami et al., 2003). Sources are shown across the top of the figure, and the target is in the bottom right corner. The part of the final hyperedge shown with a dotted line represents PPM1A acting as a positive regulator of the reaction. The cycle in this hyperpath is shown in red, and represents the phosphorylation and subsequent dephosphorylation of ''Mothers against decapentaplegic homolog 1'' (SMAD1).

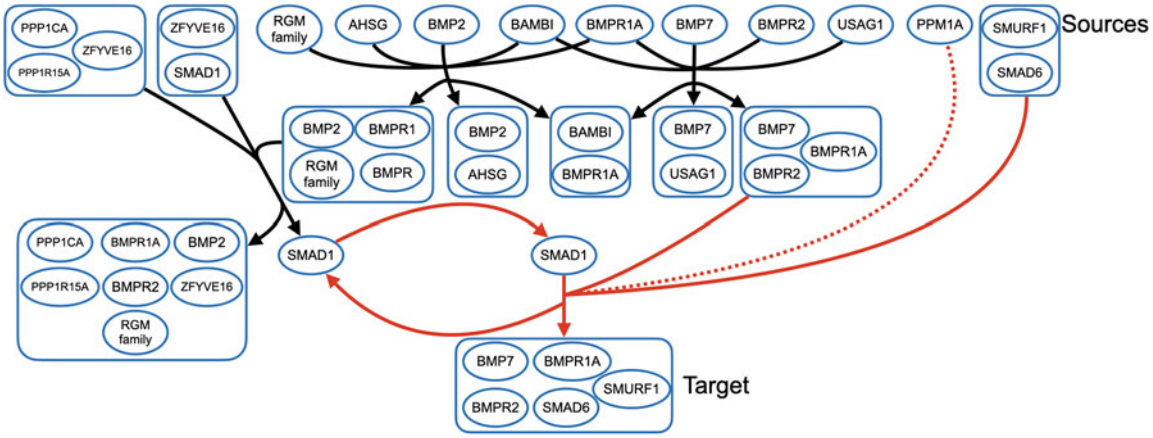

FIG. 4. Shortest hyperpath to the BMP7/BMPR2/BMPR1A/SMAD6/SMURF1 complex. Sources are shown across the top of the figure, and the target is shown at the bottom. The portion of the final hyperedge in dotted red represents PPM1A acting as a positive regulator of the reaction. The hyperedges in red form a cycle in the hyperpath.

The hyperpath for this example shows the formation of many different complexes that include ''bone morphogenetic proteins'' (BMP2, BMP7) and their receptors. These receptor complexes are then marked for ubiquitination and degradation both by SMAD1 and the complex of SMAD6 and ''SMAD-specific E3 ubiquitin protein ligase 1'' (SMURF1).

This particular example shows how even a short hyperpath can be fairly involved, by sharing many vertices among its hyperedges in various ways. While this hyperpath has only five hyperedges, most have tails and heads with multiple vertices, and hence would likely not be captured correctly by ordinary graph-based methods.

4.5. *Accuracy of pathway inference*

`Mmunin` recovers known annotated pathways from `Reactome` typically with very high accuracy. For the following *pathway inference* experiments, we formed new problem instances for a small number of known pathways in `Reactome`: namely, all ten pathways from the 22 instances with only cyclic hyperpaths where the target appears in only *one* annotated `Reactome` pathway[†] (and hence there is a unique *true* pathway that produces the target). This gave ten instances for the inference experiment.

For each instance, given by a particular `Reactome` pathway P^*, the input hypergraph G contains *all* vertices and hyperedges from `Reactome`; the *sources* consist of all vertices in G with no incoming hyperedges in G, and all vertices in pathway P^* with no incoming hyperedges from within P^*; and the *targets* are all vertices in P^* with no outgoing hyperedges from within P^*. Across these ten instances, the number of hyperedges in pathway P^* has range $[2, 98]$, with median $[14, 17]$. We note that all these instances now contain *multiple targets*.

For five of these instances, the sink was not reachable from the supersource, due to an unreachable cycle. (A simple example of such a cycle is when vertex v is in the tail of all in-edges to vertex w and vice versa.) To reestablish reachability, we manually inspected each unreachable cycle and added vertices to the source set[‡] from the cycle that would not normally qualify as sources (since these vertices have incoming hyperedges). We tried to choose for these new source vertices those that would minimize the number of reactions that could now be bypassed, since these new sources could now be in the tail of a hyperedge in a hyperpath before being reached in the head of a prior hyperedge.

[†]These ten benchmark pathways are identified by the following names: ''Glycogen Synthesis,'' ''Hydrocarboxylic Acid-Binding Receptors,'' ''Interleukin-37 Signaling,'' ''Proton/Oligopeptide Cotransporters,'' ''Regulation of Complement Cascade,'' ''Sphingolipid Metabolism,'' ''Tolerance by Mtb to Nitric Oxide Produced by Macrophages,'' ''Transport of Small Molecules,'' ''Triglyceride Catabolism,'' and ''Uncoating of the Influenza Virion.''

[‡]For all ten instances, the specific vertices that were added to the source set are listed in file `supplementary_sources` on `Mmunin`'s website, identified by their Pathway Commons ID. This includes both vertices that contain no incoming hyperedges from within the known pathway P^*, and vertices added to establish reachability of an otherwise unreachable cycle.

This resulted in a doubly-reachable subgraph containing up to 3,600 hyperedges for some pathway inference instances, which is more than double the size of this subgraph for any of our previous single-target instances. Due to the expansion of this subgraph, the running time of Mmunin on some instances was significantly longer, taking up to 25 hours to compute an optimal hyperpath (which we investigate further below).

To evaluate the accuracy of pathway inference on these instances, we compared Mmunin's *inferred* hyperpath $\widetilde{P}$ with the *true* annotated pathway P^* from Reactome for each instance. On five of the instances $\widetilde{P} = P^*$, meaning Mmunin perfectly recovered the annotated pathway. For the other five instances, $\widetilde{P}$ contained fewer hyperedges than P^*, either due to redundant branches in P^* (causing $\widetilde{P} \subset P^*$) or the existence of hyperedges outside P^* that more efficiently reach vertices within P^* (which is evidence of potential crosstalk between cellular pathways).

We evaluated the accuracy of recovering the true pathway P^* by inferred pathway $\widetilde{P}$ using three measures: *precision*, *recall*, and *overlap*, which are respectively,

$$\frac{|\widetilde{P} \cap P^*|}{|\widetilde{P}|}, \quad \frac{|\widetilde{P} \cap P^*|}{|P^*|}, \quad \frac{|\widetilde{P} \cap P^*|}{\frac{1}{2}\left(|\widetilde{P}| + |P^*|\right)}.$$

Across the ten instances,

- precision has range [0.56, 1], with median [0.9, 1];
- recall has range [0.70, 1], with median 1; and
- overlap has range [0.62, 1], with median [0.95, 1].

This experiment shows Mmunin is able to accurately recover known pathways, or possibly discover more efficient ones.

Factors affecting inference time. The markedly longer running time for some instances of pathway inference is likely due to a much larger doubly-reachable subgraph. For these instances, we compared Mmunin's running time with three features of the instance with which we thought it might share a correlation: (a) the number of hyperedges in the doubly-reachable subgraph, (b) the number of hyperedges in the shortest hyperpath, and (c) the number of targets for the instance (since all these instances now have more than one target, in contrast to our earlier single-target experiments).

Figure 5a–c shows scatterplots comparing running time with these three features. The ten instances can be naturally separated into two groups by their running time, so we identified the same four instances with high running time by red discs across the plots, and the same six instances with low running time by blue triangles.

Interestingly, of these three features, the only one having any correlation with running time is the number of *doubly-reachable hyperedges*. The number of targets for an instance, and the length of its shortest hyperpath, surprisingly seem to have no relation with running time.

4.6. Effect of constraint classes on running time

Our last experiments study the effect of the structure- and distance-based constraint classes—and their subclasses—on the running time of Mmunin. We first compare running times when *excluding* separately each single constraint-subclass, and then compare times when *including* subclasses together one-by-one within the cutting-plane algorithm.

4.6.1. Excluding constraint classes. To understand which constraints best contribute to Mmunin's running time, we conducted the following exclusion experiment. We separately considered these classes: the distance-based cuts, the head-hitting constraints, the tail-covering constraints, the target-production constraint, and the class of cut-inequalities added at each iteration of the cutting-plane algorithm. For each of these classes, we determined their contribution to Mmunin's running time by removing that single class from the initial ILP or from subsequent iterations, and then comparing the resulting running time with the original time for Mmunin with all classes present. We performed this experiment on the 22 instances from Reactome with no acyclic hyperpath. We chose this small set of instances because they have diverse shortest hyperpath lengths and (as shown later) Mmunin's running time when excluding constraint classes can be very high. The results of this exclusion experiment are summarized in Table 4.

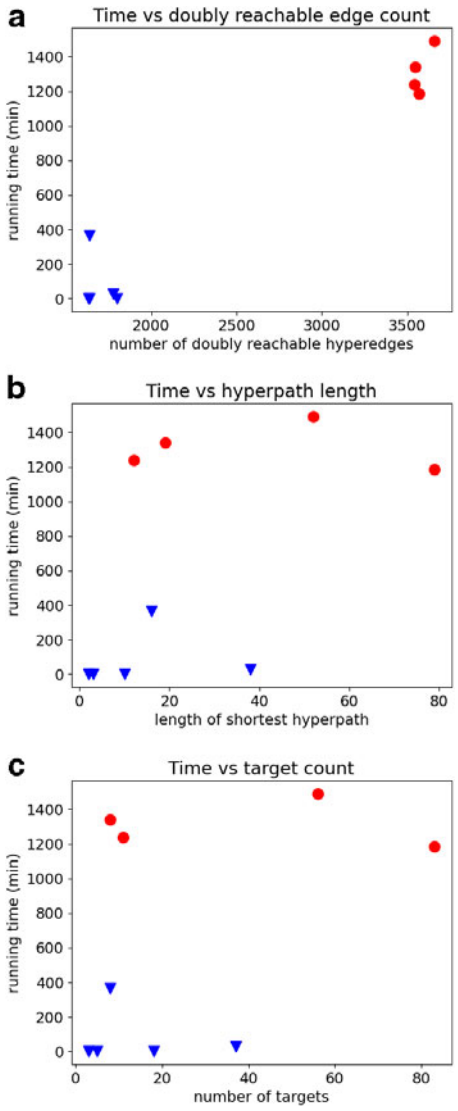

FIG. 5. Comparison of `Mmunin`'s running time to three features of the instances for the pathway inference experiment. The three features are: (a) size of the doubly-reachable edge set, (b) shortest hyperpath length, and (c) number of targets. Red discs mark the four instances with high running time, while blue triangles mark the six instances with low running time.

TABLE 4. COMPARING EXCLUDED CONSTRAINT CLASSES

Excluded class	*none*		*distance-based*		*head-hitting*		*tail-covering*		*target-prod*		*cutting-plane*	
Not optimal in 24 hours	0/22		3/22		3/22		3/22		0/22		0/22	
	median	*max*	*median*	*max*	*median*	*max*	*median*	*max*	*median*	*max*	*median*	*max*
Number of iterations	2	802	2	(∞)	112	(∞)	11	(∞)	2	718	2	893
Time per iteration (s)	4.2	4.8	4.2	5.6	1.0	4.9	3.0	12.0	4.2	4.7	4.0	4.5
Total time (s)	9.0	1,454	9.1	(∞)	56.8	(∞)	27.8	(∞)	8.7	1,488	8.7	1,208

We found that the distance-based cuts, head-hitting constraints, and tail-covering constraints contribute the most to a reasonable running time, as their absence causes a huge slowdown. In reality, rather than literally excluding *all* distance-based cuts, to ensure that the cutting-plane algorithm terminates (given enough time) we must replace the distance-based cuts with the single cut $C = \{s\}$. Our implementation of `Mmunin` in fact starts with this single cut, but once `Hhugin` terminates in parallel, all of the distance-based cuts are then computed and supplied to the cutting-plane algorithm. For many of the easier instances, `Hhugin` does not actually terminate before `Mmunin`, in which case the distance-based cuts never get added, even when they are not deliberately excluded. For the harder instances, however, the distance-based cuts, head-hitting constraints, and tail-covering constraints are all necessary for `Mmunin` to finish within 24 hours. For the easier instances, exclusion of the head-hitting constraints has the most impact, as shown by the largest increase in median running time and iteration count.

To determine the effect on running time of the cutting-plane inequalities that are source- and sink-augmented distance-based cuts, we replaced `Mmunin`'s general separation algorithm with the *trivial* one that at each iteration just finds the set R of vertices reachable from s along the active hyperedges of the current solution x^*, and only adds the single cut $C = R$ to the ILP. We found that using this trivial separation algorithm and excluding the target-production constraint had little effect on running time for these 22 instances. On the full set of instances from `NCI-PID` and `Reactome`, however, the general separation algorithm and the target-production constraint do improve the running time for the hardest instances. Unsurprisingly, using the trivial separation algorithm increases the number of iterations needed to find an optimal hyperpath, likely because fewer cut-based inequalities are added at each iteration.

4.6.2. Including constraint classes. For the second inclusion experiment, we first ranked the constraint classes by their effect on running time (from greatest effect to least) based on the outcome of the previous exclusion experiment, which gave the following order: (a) head-hitting constraints, (b) tail-covering constraints, (c) distance-based constraints, (d) target-production constraint, and (e) augmented cutting-planes. Then for each of the 22 cyclic `Reactome` instances, we initially ran `Mmunin` without any constraint classes, followed by including classes together one-by-one in this order, rerunning `Mmunin` on including each successive class. For this inclusion experiment, we set a time limit of 48 hours for each run.

The simplest possible version of the cutting-plane algorithm with *none* of the initial constraint classes (and the trivial separation algorithm) did not finish within this time limit on *any* of these instances. When starting with no initial constraints in the first ILP, the earliest iterations of this simplest cutting-plane algorithm tend to find current solutions x^* with only one active hyperedge, whose tail is a subset of sources; then current solutions whose active hyperedges are pairs of hyperedges, whose tails are sources; then current solutions that are triples of such hyperedges; and so on, for successively larger cardinalities, until exceeding the time limit.

Including just the *head-hitting constraints* allows `Mmunin` to now find optimal hyperpaths on eight of the 22 instances within 48 hours. For these instances, the cutting-plane algorithm still requires a very large number of iterations to find an optimal solution, with a median of more than 5,000 iterations, and a minimum time of more than 24 hours. Even though this restricted version of `Mmunin` can find optimal solutions for some instances within the limit, further constraint classes are necessary to make it in fact practical.

Its performance when in addition including the *tail-covering constraints* is similar to its performance when only the distance-based constraints are excluded. `Mmunin` finds an optimal hyperpath within 48 hours now for 19 of the 22 instances, with a median of 2 iterations, a median time per iteration of 4 seconds, and a median total time of 10 seconds. This is close to when excluding the distance-based class likely because excluding the other classes (namely the target-production constraint and the augmented cutting-planes) does not have a major impact on performance on these instances (as seen earlier in Table 4).

The further addition of the *distance-based constraints* allows us to finally find optimal solutions on *all* 22 instances, yet its typical performance is still roughly the same as the preceding version: a median of 2 iterations, a median time per iteration of 4 seconds, and now a median total time of 9 seconds.

The results for adding the final two constraint classes—the *target-production constraint* and the *augmented cutting-planes*—are already given in Table 4 under the respective columns ''cutting-plane'' and ''none.'' The effect of including these last two classes for these 22 instances is inconclusive: their inclusion or exclusion does not have a consistent effect on our median and maximum performance criteria.

In short, the results of this experiment confirm the importance of the head-hitting, tail-covering, and distance-based inequalities, and indicate that their mutual inclusion allows us to typically find optimal hyperpaths quickly.

5. CONCLUSION

We have presented the first practical exact algorithm for Shortest Hyperpaths—a generalization of shortest paths to directed hypergraphs—which employs a cutting-plane-like approach to solve an integer linear programming formulation, derived from a new graph-theoretic characterization of superpaths, that for the first time captures fully-general hyperpaths with cycles. Comprehensive experiments on large real-world cellular reaction networks show we can compute optimal hyperpaths quickly, and that shortest hyperpaths infer true pathways with high accuracy.

Further research

This opens many directions for further research. We offer some avenues for investigation in combinatorial optimization, algorithm design, computational biology, and algorithm evaluation.

Studying the superpath polytope. The largest advance in solving Shortest Hyperpaths to optimality is likely to come from studying an object in combinatorial optimization called the *superpath polytope*: a representation of the convex hull of the domain $\mathcal{D}$ for our integer linear program—consisting of all binary vectors that encode the s, t-superpaths of a hypergraph—in terms of linear inequalities over real-valued variables. (The description of domain $\mathcal{D}$ given earlier in equation (2) uses linear inequalities over integer-valued variables.) Even if a full description of the superpath polytope cannot be completely determined, identifying classes of *facet-defining inequalities* that lie on the faces of this polytope, and that have an efficient separation algorithm or a good separation heuristic, should further improve the cutting-plane approach to Shortest Hyperpaths in practice, by enabling it to solve a series of more computationally-efficient *linear programs*.

On a related note, it is surprising how quickly modern solvers such as `CPLEX` can find optimal solutions to the series of integer linear programs in our current cutting-plane approach—even on instances with thousands of hyperedge variables—which suggests the linear programming relaxation of our integer program may already tend to be integer-valued in practice.

Accommodating negative edge weights. Our formulation of Shortest Hyperpaths relies on positive edge weights (but can be extended to nonnegative edge weights through postprocessing as mentioned toward the end of Section 2), and breaks down for *negative edge weights*.

This parallels the behavior of shortest paths in ordinary directed graphs, where algorithms like Dijkstra's for the special case of nonnegative edge weights are much simpler and more efficient than those for the general case of arbitrary weights (and in particular negative weights), whose known algorithms differ markedly from Dijkstra's. Moreover, shortest paths with negative edge weights can be efficiently solved in ordinary graphs only because the problem statement permits non-simple paths that can repeatedly use a given cycle, allowing an algorithm to report there is no best solution once it discovers a reachable negative-weight cycle, which can be efficiently detected. When shortest paths are required to be simple paths, which effectively corresponds to requiring solutions be *minimal*, the problem under negative edge weights becomes NP-complete even for ordinary graphs, as an algorithm can then be forced to find a longest path (or equivalently, detect a Hamiltonian path). In the context of Shortest Hyperpaths, solutions already must be minimal—as hyperpaths are minimal superpaths—so for negative edge weights the recourse of reporting there is no best solution is never available (since a shortest hyperpath trivially exists whenever the sink is reachable from the source), hence an algorithm has to also deal with finding longest hyperpaths.

While our integer program finds a minimal superpath for positive weights, it fails for arbitrary edge weights. Furthermore, as indicated above, any algorithm for Shortest Hyperpaths that handles negative weights can be forced to solve a generalization of Longest Paths to hypergraphs. Whether there is a more

general algorithm that can in practice find optimal hyperpaths while accommodating negative edge weights is open, and will require a much different design than our current one.

Incorporating negative regulation. More directly relevant to computational biology is incorporating negative regulation when finding shortest hyperpaths. A *negative regulator* is a molecule that inhibits a particular reaction from occurring. Consequently, a hyperpath whose reactions create (as output products) negative regulators of its own reactions is not a suitable solution for producing targets from sources (as it inhibits itself from proceeding).

Krieger and Kececioglu (2022b) consider negative regulators in the context of finding optimal factories in metabolic networks, and introduce the notion of non-interference under *first-order* negative regulation—where a factory does not directly produce negative regulators of its own reactions—and *second-order* negative regulation—where in addition negative regulators of a factory's reactions are not indirectly produced downstream by reactions outside the factory.

Their approach to finding optimal factories under first- and second-order negative regulation provides a general technique that applies to integer-programming-based methods (through adding side-constraints and generating next-best solutions). Exploring whether this technique also enables finding optimal *hyperpaths* under first- and second-order negative regulation seems promising.

Collecting benchmark hypergraphs. Our experiments evaluating the cutting-plane approach implemented by `Mmunin` in Section 4.5 suggest its hardest instances arise from growth in the size of the doubly-reachable subgraph. While comprehensive evaluation of `Mmunin` over all single-target instances from the two available cellular reaction databases shows it can quickly find optimal hyperpaths in *current* reaction networks, its performance will inevitably degrade for even larger *future* networks, due to the inherent difficulty of solving Shortest Hyperpaths via integer linear programming.

Are there other classes of *real-world directed hypergraphs* (in contrast to simulated instances from random hypergraph generators which can lack the challenging structure of real instances) besides cellular reaction networks, for evaluating shortest hyperpath solvers, that contain harder structure (such as larger doubly-reachable subgraphs), and naturally arise from applications needing solution? Collecting such challenging benchmark hypergraphs would be a valuable contribution to the evaluation and development of improved hyperpath solvers, and likely other hypergraph algorithms as well.

Clearly there are many avenues for future investigation.

ACKNOWLEDGMENTS

We thank Anna Ritz and T.M. Murali for helpful discussions, and for providing the `BioPax` parser.

This is an extended journal version of a RECOMB conference paper (Krieger and Kececioglu, 2023).

AUTHORS' CONTRIBUTIONS

J.K.: Conceptualization, funding, and supervision. S.K.: Data curation, investigation, software, and visualization. J.K. and S.K. both contributed to method development, method analysis, writing, and editing.

AUTHOR DISCLOSURE STATEMENT

The authors declare they have no conflicting financial interests.

FUNDING INFORMATION

This research was supported by the U.S. National Science Foundation through grants CCF-1617192 and IIS-2041613 to John Kececioglu.

REFERENCES

Acuña V, Milreu PV, Cottret L, et al. Algorithms and complexity of enumerating minimal precursor sets in genome-wide metabolic networks. Bioinformatics 2012;28(19):2474–2483.

Alberts B, Johnson A, Lewis J, et al. *Molecular Biology of the Cell*. Garland Science: New York, New York; 2007.

Andrade R, Wannagat M, Klein CC, et al. Enumeration of minimal stoichiometric precursor sets in metabolic networks. Algorithms Mol Biol 2016;11(1):25.

Ausiello G, Laura L. Directed hypergraphs: Introduction and fundamental algorithms—A survey. Theor Comput Sci 2017;658:293–306.

Bodor J, Habener JF. Role of transcriptional repressor ICER in cyclic amp-mediated attenuation of cytokine gene expression in human thymocytes. J Biol Chem 1998;273(16):9544–9551.

Carbonell P, Fichera D, Pandit SB, et al. Enumerating metabolic pathways for the production of heterologous target chemicals in chassis organisms. BMC Syst Biol 2012;6(1):10.

Christensen TS, Oliveira AP, Nielsen J. Reconstruction and logical modeling of glucose repression signaling pathways in *Saccharomyces cerevisiae*. BMC Syst Biol 2009;3(1):7.

Cook WJ, Cunningham WH, Pulleyblank WR, et al. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons: 1998.

Cormen TH, Leiserson CE, Rivest RL, et al. *Introduction to Algorithms*, 3rd ed. MIT Press, Cambridge, MA; 2009.

Cottret L, Vieira Milreu P, Acuña V, et al. Enumerating precursor sets of target metabolites in a metabolic network. Proceedings of the 8th Workshop on Algorithms in Bioinformatics (WABI). 2008; pp. 233–244.

Franzese N, Groce A, Murali T, et al. Hypergraph-based connectivity measures for signaling pathway topologies. PLoS Comput Biol 2019;15(10):e1007384.

Gallo G, Longo G, Pallottino S, et al. Directed hypergraphs and applications. Discret Appl Math 1993;42(2–3):177–201.

Heath LS, Sioson AA. Semantics of multimodal network models. IEEE/ACM Trans Comput Biol Bioinform 2009;6(2):271–280.

Hou X, Liu J-E, Liu W, et al. A new role of NUAK1: directly phosphorylating p53 and regulating cell proliferation. Oncogene 2011;30(26):2933–2942.

Hu Z, Mellor J, Wu J, et al. Towards zoomable multidimensional maps of the cell. Nat Biotechnol 2007;25(5):547–554.

Italiano GF, Nanni U. Online maintenance of minimal directed hypergraphs. Technical Report. Department of Computer Science, Columbia University; 1989.

Joshi-Tope G, Gillespie M, Vastrik I, et al. Reactome: A knowledgebase of biological pathways. Nucleic Acids Res 2005;33:D428–D432.

Klamt S, Haus U-U, Theis F. Hypergraphs and cellular networks. PLoS Comput Biol 2009;5(5):e1000385.

Krieger S. *Algorithmic Inference of Cellular Reaction Pathways and Protein Secondary Structure*. PhD Dissertation, Department of Computer Science, The University of Arizona; 2022.

Krieger S, Kececioglu J. Fast approximate shortest hyperpaths for inferring pathways in cell signaling hypergraphs. Proceedings of the 21st Workshop on Algorithms in Bioinformatics (WABI), Volume 201 of Leibniz International Proceedings in Informatics. 2021; pp. 1–20.

Krieger S, Kececioglu J. Heuristic shortest hyperpaths in cell signaling hypergraphs. Algorithms Mol Biol 2022a; 17(1):12.

Krieger S, Kececioglu J. Computing optimal factories in metabolic networks with negative regulation. Proceedings of the 30th ISCB Conference on Intelligent Systems for Molecular Biology (ISMB). Bioinformatics 2022b;38(Suppl 1): i369–i377.

Krieger S, Kececioglu J. `Hhugin`: Hypergraph heuristic for general shortest source-sink hyperpaths, Version 1.0; 2022c. Available from: `http://hhugin.cs.arizona.edu`

Krieger S, Kececioglu J. `Mmunin`: Integer-linear-programming-based cutting-plane algorithm for shortest source-sink hyperpaths, Version 1.0; 2022d. Available from: `http://mmunin.cs.arizona.edu`

Krieger S, Kececioglu J. Computing shortest hyperpaths for pathway inference in cellular reaction networks. Proceedings of the 27th Conference on Research in Computational Molecular Biology (RECOMB), Volume 13976 of Springer Lecture Notes in Bioinformatics, 2023; pp. 155–173.

Li Y, McGrail DJ, Latysheva N, et al. Pathway perturbations in signaling networks: Linking genotype to phenotype. Semin Cell Dev Biol 2020;99:3–11.

Murakami G, Watabe T, Takaoka K, et al. Cooperative inhibition of bone morphogenetic protein signaling by SMURF1 and inhibitory SMADs. Mol Biol Cell 2003;14(7):2809–2817.

Nielsen LR, Pretolani D. A remark on the definition of a *B*-hyperpath. Technical Report. Department of Operations Research, University of Aarhus; 2001.

Ramadan E, Perincheri S, Tuck D. A hyper-graph approach for analyzing transcriptional networks in breast cancer. Proceedings of the 1st ACM Conference on Bioinformatics and Computational Biology (ACM-BCB). 2010; pp. 556–562.

Ramadan E, Tarafdar A, Pothen A. A hypergraph model for the yeast protein complex network. Proceedings of the 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS). 2004; pp. 189–196.

Ritz A, Avent B, Murali T. Pathway analysis with signaling hypergraphs. IEEE/ACM Trans Comput Biol Bioinform 2017;14(5):1042–1055.

Ritz A, Murali T. Pathway analysis with signaling hypergraphs. Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM-BCB). 2014; pp. 249–258.

Ritz A, Tegge AN, Kim H, et al. Signaling hypergraphs. Trends Biotechnol 2014;32(7):356–362.

Schaefer CF, Anthony K, Krupa S, et al. PID: The Pathway Interaction Database. Nucleic Acids Res 2009;37:674–679.

Schwob MR, Zhan J, Dempsey A. Modeling cell communication with time-dependent signaling hypergraphs. IEEE/ACM Trans Comput Biol Bioinform 2021;18:1151–1163.

Sharan R, Ideker T. Modeling cellular machinery through biological network comparison. Nat Biotechnol 2006; 24(4):427–433.

Vidal M, Cusick ME, Barabási A-L. Interactome networks and human disease. Cell 2011;144(6):986–998.

Wolsey LA. *Integer Programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons: 1998.

Zarecki R, Oberhardt MA, Reshef L, et al. A novel nutritional predictor links microbial fastidiousness with lowered ubiquity, growth rate, and cooperativeness. PLoS Comput Biol 2014;10(7):e1003726.

Zhou W, Nakhleh L. Properties of metabolic graphs: Biological organization or representation artifacts? BMC Bioinformatics 2011;12(1):132.

Address correspondence to:
Dr. Spencer Krieger
Computational Biology Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
USA

E-mail: skrieger@andrew.cmu.edu