Computing Shortest Hyperpaths for Pathway Inference in Cellular Reaction Networks

Spencer Krieger[1(✉)] and John Kececioglu[2]

[1] Computational Biology Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
skrieger@andrew.cmu.edu

[2] Department of Computer Science, The University of Arizona, Tucson, AZ 85721, USA
kece@cs.arizona.edu

Abstract. Signaling and metabolic pathways, which consist of a series of reactions producing target molecules from source compounds, are cornerstones of cellular biology. The cellular reaction networks containing such pathways can be precisely modeled by *directed hypergraphs*, where each reaction corresponds to a hyperedge, directed from its set of reactants to its set of products. Given such a network represented by a directed hypergraph, inferring the most likely set of reactions that produce a given target from a given set of sources corresponds to finding a *shortest hyperpath*, which is NP-complete. The best methods currently available for shortest hyperpaths either offer no guarantee of optimality, or exclude hyperpaths containing cycles even though cycles are abundant in real biological pathways.

We derive a novel *graph-theoretic characterization* of hyperpaths, leveraged in a new formulation of the general shortest hyperpath problem as an integer linear program that for the first time handles hyperpaths containing cycles, and present a novel *cutting-plane algorithm* that can solve this integer program to optimality in practice. This represents a major advance over the best prior exact algorithm, which was limited to acyclic hyperpaths (and hence fails to find a solution for the many biological instances where all hyperpaths are in fact cyclic). In comprehensive experiments over thousands of instances from the standard `NCI-PID` and `Reactome` databases, we demonstrate that our cutting-plane algorithm quickly finds an *optimal hyperpath*, with a median running-time of under ten seconds and a maximum time of around thirty minutes, even on large instances with many thousands of reactions.

Source code implementing our cutting-plane algorithm for shortest hyperpaths in a new tool called `Mmunin` is available free for research use at http://mmunin.cs.arizona.edu.

S. Krieger—Research performed at the Department of Computer Science of the University of Arizona.

H. Tang (Ed.): RECOMB 2023, LNBI 13976, pp. 155–173, 2023.
https://doi.org/10.1007/978-3-031-29119-7_10

1 Introduction

Signaling and metabolic pathways are cornerstones of systems biology. They underly cellular communication, govern environmental response, and their perturbation has been implicated in the cause of disease [21]. Networks comprised of these pathways are traditionally represented as ordinary graphs [30,31], modeling each protein or molecule as a vertex, and each reaction by a collection of edges, directed from each reactant to each product. However, this does not faithfully model multiway reactions—ubiquitous in cellular processes—and shortest paths from these models are often not biologically meaningful [14,27].

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 [7,10,11,14,23,24,26,27,33]. In particular, a biochemical reaction with multiple reactants—all of which must be present for the reaction to proceed—and multiple 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 reaction networks, the lack of practical algorithms has hindered their potential for properly representing and reasoning about molecular reactions.

Biologically, a typical metabolic or signaling pathway consists of a series of reactions synthesizing a set of target molecules—a key metabolite or transcription factor—from a set of source compounds—molecules available to the cell or activated membrane-bound receptors [2]. Computationally, finding the most efficient way to produce a set of target molecules from a set of available source compounds maps to the shortest hyperpath problem we consider here: Given a network of cellular reactions whose reactants and reactions 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 hyperpath from the sources to the targets of minimum total weight. We briefly summarize prior work on related problems.

Related Work

The two fundamental hypergraph models that have emerged for pathway inference are hyperpaths and factories (see [15] for a survey).

Factories are informally a set of reactions that produce the targets from the sources, while ensuring intermediate metabolites are not depleted. Unlike hyperpaths, a factory's hyperedges are unordered, essentially running simultaneously. Current approaches find a factory producing the targets using either the fewest reactions (*min-edge*) or fewest sources (*min-source*). Cottret et al. [8] introduced the min-source factory problem and showed it is NP-hard, and Zarecki et al. [32] extended the problem to consider molecular weights of the sources. Methods from Acuña et al. [1] and Andrade et al. [3] enumerate all min-source factories either excluding or including stoichiometry. Krieger and Kececioglu [17] introduced the min-edge factory problem, showed it was NP-complete, incorpo-

rated negative regulation into pathway inference for the first time, and solved the problem with a mixed-integer linear program that is fast in practice.

Hyperpaths were first studied in the field of algorithms (see the survey from Ausiello and Laura [4]). Italiano and Nanni [12] proved that finding a shortest source-sink hyperpath is NP-complete, even when hyperedges have a single head vertex. Gallo et al. [9] defined and explored special cases of hypergraphs and hyperpaths, including what they call a B-path (though see the correction of Nielsen and Pretolani [22]), which is essentially equivalent to our definition of hyperpath in Sect. 2. They showed the vertices reachable from a source vertex in a hypergraph can be found in time linear in the total size of the tail and head sets of all hyperedges, gave an efficient algorithm for a variant of shortest hyperpaths with a so-called additive cost function, and proved that finding a minimum cut in a hypergraph is NP-complete. Carbonell et al. [6] 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 specified set of hyperedges is NP-complete. Ritz et al. [25,26] were the first to solve the shortest *acyclic* hyperpath problem by formulating it as a mixed-integer linear program (MILP)—later extended by Schwob et al. [29] to include time-dependence among reactions—and showed that in the context of signaling networks, optimal acyclic hyperpaths can be found even for large cell-signaling hypergraphs. Their formulation does not extend to allow cycles, which are common in pathway databases (see experimental results in Sect. 4), and are often caused by feedback loops or by the components of protein complexes during assembly and disassembly. Krieger and Kececioglu [16,18] gave the first heuristic for the general shortest hyperpath problem allowing cycles, proved it finds optimal hyperpaths for the special case of singleton-tail hypergraphs, gave a tractable hyperpath enumeration algorithm, and verified that the heuristic is close to optimal on all instances from the standard pathway databases by leveraging the enumeration algorithm.

Our Contributions

In contrast to prior work, we provide an exact algorithm for the general shortest hyperpath problem, allowing cycles. Note that cycles appear in pathway databases—often as feedback loops—so an algorithm that handles cycles is vital for the adoption of hypergraph models of cellular reaction networks. We formulate this problem as an integer linear program (ILP) and develop a cutting-plane algorithm that can quickly solve this ILP to optimality in practice. 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. This cut-based ILP formulation has an exponential number of constraints, however, and cannot be solved directly.

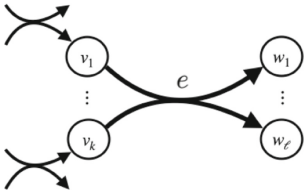

Fig. 1. A hyperedge e with $\text{tail}(e) = \{v_1, \ldots, v_k\}$ and $\text{head}(e) = \{w_1, \ldots, w_\ell\}$. To use e in a hyperpath P, every vertex $v_i \in \text{tail}(e)$ must have a preceding hyperedge f in P with $v_i \in \text{head}(f)$.

- Nevertheless, we show we can solve this ILP on real biological instances with a practical *cutting-plane algorithm* that computes over a small subset of the constraints, while guaranteeing an optimal solution to the full ILP.
- Our cutting-plane algorithm is typically *fast in practice*, finding optimal hyperpaths with a median running time under 10 s, and a maximum running time around 30 min, as measured through comprehensive experiments on thousands of instances from standard reaction databases.
- We demonstrate the strength of hyperpath models for pathway inference by showing the cutting-plane algorithm accurately recovers annotated pathways, with a median overlap score of over 95%, while recovered hyperedges outside the annotated pathway provide evidence for possible cross-talk.

A preliminary implementation of the 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 non-commercial use at `http://mmunin.cs.arizona.edu`.

Plan of the Paper

The next section defines the computational problem of shortest hyperpaths. Section 3 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 it to optimality in practice. Section 4 compares our algorithm to alternate hyperpath methods through comprehensive experiments over the two standard pathway databases, and demonstrates on concrete biological examples that we can accurately recover known annotated pathways. Finally, Sect. 5 concludes.

2 Shortest Hyperpaths in Directed Hypergraphs

A hypergraph is a generalization of an ordinary graph, 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 non-empty vertex subsets. Edge e is directed from set X to

set Y. We call X the *tail* of e, and Y the *head* of e, and refer to them by the functions $\text{tail}(e) = X$ and $\text{head}(e) = Y$. We also refer to the in- and out-edges of a vertex v by $\text{in}(v) = \{e \in E : v \in \text{head}(e)\}$ and $\text{out}(v) = \{e \in E : v \in \text{tail}(e)\}$. Figure 1 shows a directed hyperedge.

In ordinary directed graphs, a path from a vertex s to a vertex t is a sequence of edges starting from s ending at t, where for consecutive edges e and f in the sequence, 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 a path from s to t.

In generalizing these notions to directed 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 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.* □

This definition of superpath is equivalent to the notion of B-connectivity from the literature, but is more explicit, and more amenable to formulation as an integer linear program in Sect. 3.

We can now define hyperpaths in terms of superpaths. Recall that 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 edge $e \in P$ leaves a subset $P - \{e\}$ that is no longer a superpath.

We say a 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 an earlier hyperedge e in the ordering. While in ordinary graphs a minimal s,t-path can never contain a cycle, in hypergraphs an s,t-hyperpath, which by definition is minimal, can in fact contain cycles.

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

For an edge weight function $\omega(e)$, we extend ω to edge subsets $F \subseteq E$ by $\omega(F) := \sum_{e \in F} \omega(e)$.

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

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

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

For uniform weights $\omega(e) = 1$, this finds a series of the *fewest possible reactions* that produce t from s, where for each reaction in the series, all its input reactants are produced as output products of earlier reactions in the series.

We note that Shortest Hyperpaths with a single source and sink can also capture more general versions of the problem with *multiple sources* and *multiple sinks*, 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 versions of shortest hyperpaths with multiple sources and sinks can be reduced to the problem above with a single source and sink.

Shortest Hyperpaths is NP-complete [25] (even for acyclic hypergraphs), so it is unlikely we can efficiently compute shortest hyperpaths in the worst-case. The next section presents a new formulation of Shortest Hyperpaths as an integer linear programming problem, which allows us to leverage techniques for solving integer linear programs to quickly find shortest hyperpaths in practice, even for large reaction networks.

3 Computing Hyperpaths by Integer Programming

We now formulate Shortest Hyperpaths as a discrete optimization problem known as an integer linear program, using a characterization of superpaths in terms of cuts. This integer linear program is the first formulation that handles fully general hyperpaths that may contain cycles.

3.1 Characterizing Superpaths via Cuts

We can give a clean characterization of superpaths—that captures fully general superpaths containing cycles—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 iff $\text{tail}(e) \subseteq C$ and $\text{head}(e) \cap \overline{C} \neq \emptyset$. 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 iff at least one hyperedge $e \in F$ crosses the cut.

The following characterization theorem for superpaths is the key that will enable us to find shortest hyperpaths via integer linear programming.

Theorem 1 (Characterizing Superpaths). *F 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 $\text{head}(e) \cap \overline{C}$ is nonempty. (Such an edge e must exist, as F reaches $t \in \overline{C}$.) We claim that $\text{tail}(e) \subseteq C$, which can be shown by proving $\bigcup_{f \in F\,:\, f \text{ precedes } e} \text{head}(f) \subseteq C$, using induction over the ordering of F. Thus edge $e \in F$ crosses cut C, so F crosses C as well.

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 $t \notin R$ (since otherwise F reaches t from s, contradicting that F is not an s,t-superpath). Thus $(R,\overline{R})$ is an s,t-cut. F does not cross cut R (since $e \in F$ crossing R would contradict that R holds all vertices reachable from s in F). □

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. 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 $x_e = 1$ iff $e \in F$. 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 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

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

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 iff 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 Eq. (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 edge weights ω, 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 edge weights.

Finally, our *integer linear program* is to compute $\operatorname{argmin}_{x \in \mathcal{D}} \{\omega \circ x\}$. Since domain $\mathcal{D}$ is all s,t-superpaths, this is equivalent to the definition of the Shortest Hyperpaths problem, so a solution to this ILP is a shortest s,t-hyperpath.

3.3 Solving the Integer Program by a Cutting Plane Algorithm

For a hypergraph of n vertices and m hyperedges, the integer linear program given 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.

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 adds inequalities to this set as it finds cuts that solutions to prior ILPs do not cross.

The above cutting-plane algorithm outputs an optimal solution to the full ILP, though it works with only a subset of its constraints. Note that when the algorithm outputs its final solution x^* at Step (5), which crosses every cut, x^* encodes an s,t-superpath (by Theorem 1). Furthermore, as x^* is an optimal solution for a less-constrained ILP that is a minimization problem, its objective function value is a lower bound on the weight of an optimal solution to the full ILP. Hence x^* is a minimum-weight s,t-superpath, or equivalently, a shortest s,t-hyperpath.

The next section specifies the initial inequalities $\mathcal{I}$ used in practice, which are crucial to its success. Then Sect. 3.5 presents our efficient separation algorithm, which finds multiple violated inequalities that are all added to the current ILP. As demonstrated in Sect. 4, this cutting-plane algorithm can find optimal hyperpaths even for large instances from real cellular reaction networks.

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 $\mathcal{I}$ containing both the structure-based and distance-based inequalities that we describe next.

Structure-Based Inequalities. We define three classes of inequalities, based on structural properties of hyperpaths.

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

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, $\left(\sum_{f \neq e \,:\, \text{head}(e) \cap \text{tail}(f) \neq \emptyset} x_f\right) \geq x_e$.

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). More formally, $\sum_{f \in \text{in}(t)} x_f \geq 1$.

Together, these 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). Adding these inequalities to the initial ILP dramatically reduces the number of iterations of the cutting-plane algorithm, greatly improving its running time.

Distance-Based Inequalities. We first show that for ordinary graphs, there is a small subset of constraints from our original ILP, given by what we call distance-based cuts, that we can efficiently find and that guarantee that the ILP solved on just these distance-based inequalities has objective function value equal to the shortest path length. We then generalize these inequalities 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

$$C_i \;:=\; \{s\} \cup \Big\{ v \in V \,:\, D(v) < d_i \Big\}. \tag{3}$$

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

The following theorem implies that for ordinary graphs, if we solve our original ILP just over inequalities corresponding to these distance-based cuts—which for a graph with n vertices is an ILP with less than n inequalities—then the objective function value of the optimal solution to this small ILP will in fact be the length of a shortest s,t-path.

Theorem 2 (Distance-Based Cuts Suffice for Ordinary Graphs). *In an ordinary graph, let F be an edge set that crosses every cut in the family $\mathcal{C}$ of distance-based s,t-cuts. Then F has total weight $\omega(F)$ that is at least the length $D(t)$ of a shortest s,t-path.*

Proof. We show by induction that the weight of an edge set crossing cuts $C_1, \ldots, C_i$ is at least d_i, for all $1 \leq i \leq k$. Since $d_k = D(t)$, this proves the theorem.

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. Since we have that $\omega(F) \geq \omega(f) \geq \omega(e) = D(v) = d_1$, 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 $f = (x,y)$ that crosses cut C_i. Notice that $D(x) < d_i$, which implies $D(x) = d_j$ for some $j < i$. Let $F' \subseteq F - \{(x,y)\}$ be the subset of F that crosses cuts $C_1, \ldots, C_j$. We have

$$\begin{aligned}
\omega(F) &\geq \omega(F') + \omega(f) \\
&\geq D(x) + \omega(f) && (4) \\
&\geq D(y) && (5) \\
&\geq d_i\,, && (6)
\end{aligned}$$

where inequality (4) follows from the inductive hypothesis on $j < i$, inequality (5) follows from the fact that $D(x) + \omega(f) = D(x) + \omega(x,y) \geq D(y)$, and inequality (6) follows from the definitions of edge (x,y) and cut C_i. Thus the induction holds. □

Notice that a shortest s,t-path crosses every cut in family $\mathcal{C}$. Hence a consequence of Theorem 2 is that, for the optimal solution to the ILP whose inequalities are just the cut constraints given by the distance-based cuts $\mathcal{C}$, its objective function value is the length of a shortest s,t-path.

The number of cuts in family $\mathcal{C}$ is less than the number of vertices. Moreover, Dijkstra's single-source shortest-path algorithm computes distance $D(v)$ for every vertex v reachable from s with distance at most $D(t)$. Thus for an ordinary 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 $\text{tail}(e)$, 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, $D(v) := \min_{e \in \text{in}(v)} \{D(\text{tail}(e)) + \omega(e)\}$.

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 Kececioglu [16,18], which 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 (3), to hypergraphs. This yields the *distance-based inequalities* that our cutting-plane algorithm starts from in its initial set $\mathcal{I}$.

3.5 A Separation Algorithm Leveraging Distance-Based Cuts

The cutting-plane algorithm starts with inequalities $\mathcal{I}$, where $\mathcal{I}$ contains the structure-based inequalities, as well as the distance-based inequalities from the family of cuts $\mathcal{C}$. 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 $\mathcal{C}$. To find new cuts not crossed by x^*, the separation algorithm considers every s,t-cut $C \in \mathcal{C}$, and enlarges its source-side $C \supseteq \{s\}$, called *source-augmentation*, or enlarges its sink-side $\overline{C} = V - C \supseteq \{t\}$, called *sink-augmentation*, to obtain a new cut $\widehat{C}$ not crossed by x^*.

Source-augmentation of $C \in \mathcal{C}$, to obtain $\widehat{C} \supset C$ not crossed by x^*, proceeds 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 crosses. (Here $\widehat{C}$ could grow until it includes sink t, which makes it no longer an s,t-cut, in which case there is no source-augmentation of C that x^* does not cross.) This process actually finds the cut $\widehat{C} \supset C$ of minimum size $\left|\widehat{C}\right|$ that x^* does not cross.

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

Sink-augmentation of $C \in \mathcal{C}$, to obtain $\widehat{C} \subset C$ not crossed by x^*, proceeds as follows. For a given active hyperedge e that crosses C, we enlarge $\overline{C}$ by moving one vertex $v \in \text{tail}(e) - \{s\}$ from C to its sink-side $\overline{C}$, yielding 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 edges 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 $\widehat{C}$. (Once an active hyperedge e crossing $\widehat{C}$ has $\text{tail}(e) = \{s\}$, this fails to find a sink-augmentation of C not crossed by x^*.) 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.

This separation algorithm can find up to $2k$ inequalities that are violated by x^*, where $k = |\mathcal{C}|$ is the number of distance-based cuts, all of which are added to the current set $\mathcal{S}$ by the cutting-plane algorithm. For a hypergraph of size $\ell = \sum_{e \in E} \big(|\text{tail}(e)| + |\text{head}(e)|\big)$, this separation algorithm can be implemented to run in $O(k^2\,\ell)$ time.

4 Experimental Results

We present experimental results with our implementation of the cutting-plane algorithm, named `Mmunin` [20], that show it can find optimal hyperpaths in large real-world cellular reaction networks quickly, often in less than 10 s. We first give details of our experimental setup, describing our datasets and implementation. We then show, through comprehensive experiments over all source-sink instances from the two standard reaction databases in the literature, how `Mmunin` surpasses both the state-of-the-art heuristic for general shortest hyperpaths [16,18], and the state-of-the-art exact algorithm for shortest acyclic hyperpaths [25]. Finally, we discuss an illustrative biological example, and analyze how well `Mmunin` recovers known biological pathways.

4.1 Experimental Setup

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

Table 1. Dataset summaries

	NCI-PID		Reactome	
Pathways	213		2,516	
Vertices	9,009		20,458	
Hyperedges	8,456		11,802	
Sources	3,200		8,296	
Targets	2,636		5,066	
Reachable targets	2,220		2,432	
	mean	max	mean	max
Tail size	1.9	10	2.4	26
Head size	1.1	5	1.6	28
In-degree	1.0	323	0.9	1,056
Out-degree	1.7	326	1.4	1,167
Doubly-reachable set	756	1,836	929	1,725

Datasets and Preparation. We prepared instances from two benchmark datasets, called `NCI-PID` and `Reactome`, following the hypergraph construction protocol of Ritz et al. [25] and Krieger and Kececioglu [18]. The `NCI-PID` dataset aggregates all pathways from NCI-PID [28], while the `Reactome` dataset aggregates all pathways from Reactome [13]. To build a hypergraph from each dataset, we map each protein or small molecule to a vertex, and each reaction to a hyperedge, with reactants and positive regulators in the tail, and products in the head. All hyperedges are given unit weight, even though the cutting-plane algorithm handles general weights, as NCI-PID is missing reaction rates for many reactions. Table 1 gives summaries of these hypergraphs. The `NCI-PID` hypergraph has 9,009 vertices and 8,456 hyperedges, while the `Reactome` hypergraph has 20,458 vertices and 11,802 hyperedges. For all instances, we create a super-source s and add a single hyperedge e, where $\text{tail}(e) = \{s\}$ and $\text{head}(e)$ contains all vertices with no in-edge, which we call *sources*. For each individual instance, we create a sink t and connect it by an ordinary graph edge (v, t) to one vertex v with no out-edges, which we call a *target*. Considering all possible targets v, generates 2,636 instances from `NCI-PID`, and 5,066 instances from `Reactome`.

Implementation. The cutting-plane algorithm and its separation algorithm are all implemented in Python 3.8, comprising around 2,000 lines of code. All procedures are implemented as described earlier, with a few exceptions. First, we use a procedure from `Hhugin` to compute the doubly-reachable subgraph H, which contains only those hyperedges from the input hypergraph G 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 begin execution of `Hhugin` and the cutting plane algorithm in

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 path-length difference-from-optimal	(∞)	(∞)	(∞)	(∞)
Hhugin path-length difference-from-optimal	1	6	0	0

Table 3. Performance of Mmunin

Instances	NCI-PID 2,220		Reactome 2,432	
	median	max	median	max
Number of iterations	3	1,598	3	874
Time per iteration (sec)	2	12	3	12
Total time (sec)	7	1,788	9	776

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

We also made one modification to Hhugin. Previously, when hyperedges were removed from the heap, their in-edge lists were frozen, so that new hyperedges were never added. We changed this behavior so that these in-edge lists continue to grow as new hyperedges are extracted from the heap.

Source code for the cutting-plane algorithm, including all datasets, is available at http://mmunin.cs.arizona.edu [20].

4.2 Comparing Alternate Hyperpath Methods

Mmunin outperforms state-of-the-art hypergraph methods for pathway inference: the hyperpath heuristic Hhugin [19], and the MILP for shortest acyclic hyperpaths [25] , which we call AcycMILP. We compare these methods over all instances from Reactome and NCI-PID. Table 2 gives statistics for these instances.

For all these instances, Mmunin computes an optimal shortest hyperpath in less than 30 min, while allowing cycles for the first time. Mmunin surpasses AcycMILP on 22 Reactome instances and 38 NCI-PID instances where all s,t-hyperpaths are cyclic, hence AcycMILP fails to return any hyperpath. Mmunin outperforms Hhugin on the 23 NCI-PID instances where Hhugin is suboptimal.

`Mmunin` not only returns an optimal hyperpath for these instances, but also finishes its computation before `Hhugin`, showing that `Mmunin` (even without the distance-based constraints given by `Hhugin`) is faster for these instances. In fact, we found that `Mmunin` is faster than `Hhugin` on over 20% of all instances.

4.3 Speed of Computing Optimal Hyperpaths

`Mmunin` is typically fast in practice, with a median running time under 10 s. Table 3 gives statistics on the running time, number of iterations, and time per iteration over the instances from each dataset. The maximum running time over all these instances is just under 30 min, demonstrating it is now feasible to find optimal shortest hyperpaths even for large instances with over 10,000 hyperedges. We note that inherent randomness in the CPLEX solver may cause variable running times, even for the same instance.

The number of iterations needed for the cutting-plane algorithm to return an optimal solution tends to be low, around 2 or 3 iterations, but can be as high as 1,598 iterations. Even though the cutting-plane algorithm may require many iterations, the instance with the highest average time per iteration takes only 12 s per iteration. Note that at each iteration, the cutting-plane algorithm solves an ILP containing thousands of variables and inequalities within this time.

4.4 A Concrete Biological Example

As an illustration of the hyperpaths found by `Mmunin`, we show one concrete biological example from `NCI-PID`. This instance was chosen from the 23 instances where `Mmunin` outperforms the hyperpath heuristic `Hhugin`, because the size of the hyperpaths makes them reasonable to draw. Note that `AcycMILP` is also optimal on this instance, since the optimal hyperpath is acyclic. Figure 2 shows the hyperpaths returned by `Mmunin` and `Hhugin` for this instance, which represents the deactivation of "nuclear factor of activated T cells" (NFATC2) by "cAMP response element modulator" (CREM) in `NCI-PID`. Hyperedges drawn in a red dash are unique to `Mmunin`'s hyperpath, hyperedges in a green dotted line are unique to `Hhugin`'s hyperpath, and hyperedges in a solid black line are common to both hyperpaths. Eight of the nine hyperedges that are shared by both pathways have been omitted for simplicity, and have been replaced with ellipses. The hyperedges from MAPK8 to JUN and from MAPK3 to JUN denote transcription of JUN, where all other hyperedges denote biochemical reactions. Vertices with gray fill denote the activated form of a given protein. Note that stoichiometry of the reactions are not considered in our hyperpath formulation, so two copies of NFATC2 are needed in the final reaction.

The hyperpaths contain hyperedges from four different 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

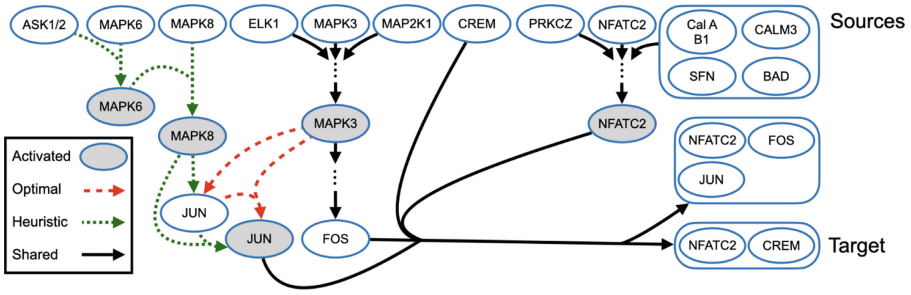

Fig. 2. Comparing hyperpaths whose target is the NFATC2/CREM complex.

and its DNA binding sites, resulting in the attenuation of transcription of T helper-1-specific cytokine genes in human medullary thymocytes [5].

`Hhugin`'s hyperpath contains 13 hyperedges and `Mmunin`'s hyperpath contains 11 hyperedges, which is optimal. Notably, hyperedges unique to `Mmunin` use vertices that are shared between both hyperpaths, and is therefore much simpler, making it a more likely pathway.

4.5 Analysis of Recovering Known Pathways

`Mmunin` accurately recovers annotated pathways from `Reactome`. We define new problem instances for a small number of annotated pathways P^* from `Reactome`: ten pathways from the 22 instances with only cyclic hyperpaths (so `AcycMILP` fails to return any hyperpath), where the target appears in only one annotated `Reactome` pathway. (These ten benchmark pathways are: "Regulation of Complement Cascade", "Sphingolipid Metabolism", "Triglyceride Catabolism", "Hydrocarboxylic Acid-Binding Receptors", "Transport of Small Molecules", "Proton/Oligopeptide Cotransporters", "Interleukin-37 Signaling", "Uncoating of the Influenza Virion", "Glycogen Synthesis", and "Tolerance by Mtb to Nitric Oxide Produced by Macrophages".) For each instance, hypergraph G includes all vertices and hyperedges from `Reactome`, the sources consist of all vertices in G with no in-edges and all vertices in P^* with no in-edges from P^*, and the targets are all vertices in P^* with no out-edges from P^*. For these ten instances, the number of hyperedges in P^* ranges from 2 to 98, with a median of [14,17]. Note that these instances now contain multiple targets. For five of these instances, the sink is not reachable from the source, so to restore reachability, we added to the source set vertices that are unreachable due to an unreachable cycle. (A simple example of this is when vertex a is in the tail of all in-edges to vertex b and vice versa.) This resulted in a doubly-reachable set containing 3,600 hyperedges for some instances, which is twice the size of the doubly-reachable set for any single-target instance (shown in Table 1). Due to this increase, the running time of `Mmunin` on these instances was significantly longer, taking at most 25 h to compute an optimal hyperpath.

We compared `Mmunin`'s hyperpath P with the known pathway P^* from `Reactome` for each instance. For five of the instances, $P = P^*$, meaning `Mmunin` perfectly recovered the annotated pathway. For the other five instances, P contained fewer hyperedges than P^*, either due to redundant branches in P^* (so $P \subset P^*$), or hyperedges outside P^* more efficiently reaching vertices within P^*, which is evidence of potential crosstalk between biological pathways. We measured the similarity of P and P^* for each instance by the so-called Sorensen coefficient $2\,|P \cap P^*| \,/\, (|P| + |P^*|)$. Over these ten instances, this similarity measure ranged from 0.62 to 1, with a median of $[0.95, 1]$. Overall, this experiment shows that `Mmunin` is able to accurately recover known pathways, or possibly discover more efficient ones.

5 Conclusion

We have presented a new formulation of the general shortest hyperpath problem as an integer linear program, and a practical cutting-plane algorithm that for the first time can find shortest hyperpaths with cycles. Comprehensive experiments on large real-world cellular reaction networks show we can quickly compute optimal hyperpaths, and accurately recover annotated biological pathways.

Acknowledgments. We thank Anna Ritz and T.M. Murali for helpful discussions and for providing the `BioPax` parser, and the anonymous referees for their useful comments. This research was supported by the US National Science Foundation, through grants CCF-1617192 and IIS-2041613 to JK.

References

1. Acuña, V., Milreu, P.V., Cottret, L., Marchetti-Spaccamela, A., Stougie, L., Sagot, M.F.: Algorithms and complexity of enumerating minimal precursor sets in genome-wide metabolic networks. Bioinformatics **28**(19), 2474–2483 (2012)
2. Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P.: Molecular Biology of the Cell. Garland Science, New York (2007)
3. Andrade, R., Wannagat, M., Klein, C.C., Acuña, V., Marchetti-Spaccamela, A., Milreu, P.V., Stougie, L., Sagot, M.F.: Enumeration of minimal stoichiometric precursor sets in metabolic networks. Alg. Mol. Bio. **11**(1), 25 (2016)
4. Ausiello, G., Laura, L.: Directed hypergraphs: introduction and fundamental algorithms–a survey. Theoret. Comput. Sci. **658**, 293–306 (2017)
5. Bodor, J., Habener, J.F.: Role of transcriptional repressor ICER in cyclic amp-mediated attenuation of cytokine gene expression in human thymocytes. J. Biol. Chem. **273**(16), 9544–9551 (1998)
6. Carbonell, P., Fichera, D., Pandit, S.B., Faulon, J.L.: Enumerating metabolic pathways for the production of heterologous target chemicals in chassis organisms. BMC Syst. Biol. **6**(1), 10 (2012)
7. Christensen, T.S., Oliveira, A.P., Nielsen, J.: Reconstruction and logical modeling of glucose repression signaling pathways in Saccharomyces cerevisiae. BMC Syst. Biol. **3**(1), 7 (2009)

8. Cottret, L., et al.: Enumerating precursor sets of target metabolites in a metabolic network. In: Proceedings of the 8th Workshop on Algorithms in Bioinformatics, pp. 233–244 (2008)
9. Gallo, G., Longo, G., Pallottino, S., Nguyen, S.: Directed hypergraphs and applications. Discret. Appl. Math. **42**(2–3), 177–201 (1993)
10. Heath, L.S., Sioson, A.A.: Semantics of multimodal network models. IEEE/ACM Trans. Comput. Biol. Bioinf. **6**(2), 271–280 (2009)
11. Hu, Z., Mellor, J., Wu, J., Kanehisa, M., Stuart, J.M., DeLisi, C.: Towards zoomable multidimensional maps of the cell. Nat. Biotech. **25**(5), 547–554 (2007)
12. Italiano, G.F., Nanni, U.: Online maintenance of minimal directed hypergraphs. Department of Computer Science, Columbia University, Tech. Rep. (1989)
13. Joshi-Tope, G., et al.: Reactome: a knowledgebase of biological pathways. Nucleic Acids Res. **33**, D428-432 (2005)
14. Klamt, S., Haus, U.U., Theis, F.: Hypergraphs and cellular networks. PLoS Comput. Biol. **5**(5), e1000385 (2009)
15. Krieger, S.: Algorithmic Inference of Cellular Reaction Pathways and Protein Secondary Structure. Ph.D. dissertation, Department of Computer Science, The University of Arizona (July 2022)
16. Krieger, S., Kececioglu, J.: Fast approximate shortest hyperpaths for inferring pathways in cell signaling hypergraphs. In: Proceedings of the 21st ISCB Workshop on Algorithms in Bioinformatics (WABI). Leibniz International Proceedings in Informatics, vol. 201, pp. 1–20 (2021)
17. Krieger, S., Kececioglu, J.: Computing optimal factories in metabolic networks with negative regulation. Bioinformatics, In: Proceedings of the 30th ISCB Conference on Intelligent Systems for Molecular Biology (ISMB) **38**(Suppl_1), i369–i377 (2022)
18. Krieger, S., Kececioglu, J.: Heuristic shortest hyperpaths in cell signaling hypergraphs. Algorithms Mol. Biol. **17**(1), 12 (2022)
19. Krieger, S., Kececioglu, J.: `Hhugin`: hypergraph heuristic for general shortest source-sink hyperpaths, version 1.0 (2022). http://hhugin.cs.arizona.edu
20. Krieger, S., Kececioglu, J.: `Mmunin`: integer-linear-programming-based cutting-plane algorithm for shortest source-sink hyperpaths, version 1.0 (2022). http://mmunin.cs.arizona.edu
21. Li, Y., McGrail, D.J., Latysheva, N., Yi, S., Babu, M.M., Sahni, N.: Pathway perturbations in signaling networks: linking genotype to phenotype. Semin. Cell Dev. Biol. **99**, 3–11 (2020)
22. Nielsen, L.R., Pretolani, D.: A remark on the definition of a B-hyperpath. Department of Operations Research, University of Aarhus, Tech. Rep. (2001)
23. Ramadan, E., Perincheri, S., Tuck, D.: A hyper-graph approach for analyzing transcriptional networks in breast cancer. In: Proceedings of the 1st ACM Conference on Bioinformatics and Computational Biology, pp. 556–562 (2010)
24. Ramadan, E., Tarafdar, A., Pothen, A.: A hypergraph model for the yeast protein complex network. In: Proceedings of the 18th Parallel and Distributed Processing Symposium, pp. 189–196 (2004)
25. Ritz, A., Avent, B., Murali, T.: Pathway analysis with signaling hypergraphs. IEEE/ACM Transactions on Comp. Bio. and Bioinf. **14**(5), 1042–1055 (2017)
26. Ritz, A., Murali, T.: Pathway analysis with signaling hypergraphs. In: Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM-BCB), pp. 249–258 (2014)
27. Ritz, A., Tegge, A.N., Kim, H., Poirel, C.L., Murali, T.: Signaling hypergraphs. Trends Biotechnol. **32**(7), 356–362 (2014)

28. Schaefer, C.F., et al.: PID: the pathway interaction database. Nucl. Acids Res. **37**, 674–679 (2009)
29. Schwob, M.R., Zhan, J., Dempsey, A.: Modeling cell communication with time-dependent signaling hypergraphs. IEEE/ACM Trans. Comput. Biol. Bioinform. **18**, 1151–1163 (2019)
30. Sharan, R., Ideker, T.: Modeling cellular machinery through biological network comparison. Nat. Biotechnol. **24**(4), 427–433 (2006)
31. Vidal, M., Cusick, M.E., Barabási, A.L.: Interactome networks and human disease. Cell **144**(6), 986–998 (2011)
32. Zarecki, R., Oberhardt, M.A., Reshef, L., Gophna, U., Ruppin, E.: A novel nutritional predictor links microbial fastidiousness with lowered ubiquity, growth rate, and cooperativeness. PLoS Comput. Biol. **10**(7), 1–12 (2014)
33. Zhou, W., Nakhleh, L.: Properties of metabolic graphs: biological organization or representation artifacts? BMC Bioinform. **12**(1), 132 (2011)