Evolving Dependencies: From Graphs to Hypergraphs and SuperHypergraphs

Takaaki Fujita [1] *

[1] Independent Researcher, Shinjuku, Shinjuku-ku, Tokyo, Japan.
Email: Takaaki.fujita060@gmail.com

Abstract

Graph theory studies the mathematical structure of vertices and edges to model relationships and connectivity [1, 2]. Hypergraphs extend this framework by allowing hyperedges to connect arbitrarily many vertices at once [3], and superhypergraphs further generalize hypergraphs via iterated powerset constructions to capture hierarchical linkages among edges [4, 5]. A Dependency graph is a finite directed acyclic graph whose vertices denote tasks and whose edges encode prerequisite relationships. In this paper, we extend this notion by introducing *Dependency hypergraphs* and *Dependency superhypergraphs*, and we investigate their fundamental properties. A Dependency hypergraph relates each nonempty set of prerequisite vertices to exactly one dependent vertex, while a Dependency superhypergraph is an n-level acyclic structure whose supervertices lie in iterated powersets and whose superedges encode multi-layer dependencies. We establish how these models generalize classical Dependency graphs and demonstrate their potential for representing higher-order Dependency systems.

Keywords: Superhypergraph, Hypergraph, Dependency Graph, Dependency Hypergraph, Dependency Superhypergraph

Mathematics Subject Classification (2010): 05C65 — Hypergraphs

1 Introduction

1.1 From Graphs to SuperHyperGraphs

Classical graphs represent pairwise relationships via vertices and edges [6]. Hypergraphs extend this by permitting each *hyperedge* to join any nonempty subset of vertices, capturing higher-order interactions [7, 8]. SuperHyperGraphs take this further by iteratively applying the powerset operation: edges at one level become vertices at the next, creating nested, multi-tier networks that embed hierarchical dependencies within a unified framework [9–12]. This layered approach supports the flexible representation and analysis of complex systems with recursive or nested connections [13].

1.2 Dependency Graph (Dependence Graph)

A dependency graph (dependence graph) is a finite directed acyclic graph whose vertices denote tasks and whose edges encode prerequisite relationships [14–21]. Using dependency graphs provides clear visualization of task relationships, enables acyclic scheduling via topological sorting, supports parallel execution, optimizes resource allocation, and aids modular analysis [14–16].

1.3 Our Contributions: Dependency Hypergraphs and Dependency SuperHyperGraphs

We generalize this model in two stages. First, we define *Dependency Hypergraphs*, in which each hyperedge relates a nonempty set of prerequisite tasks to a single dependent task, enabling representation of multi-input dependencies. Second, we introduce *Dependency SuperHyperGraphs*, which extend dependency hypergraphs to n levels by iterating the hypergraph construction, thus capturing layered dependency structures. We present formal definitions, prove that these models strictly generalize classical dependency graphs, and explore their fundamental properties. Dependency SuperHypergraphs capture multi-layered prerequisites, model nested dependency hierarchies, facilitate advanced scheduling, improve analysis, enhance abstraction, and optimize resource allocation.

2 Preliminaries

Throughout this paper, we work with finite, simple graphs unless stated otherwise. This section reviews the key definitions and notation that will be used in later sections; for more in-depth discussions, please refer to the cited sources.

2.1 Hypergraphs and SuperHypergraphs

Hypergraphs extend ordinary graphs by allowing hyperedges to join any number of vertices simultaneously [3,7,22,23]. SuperHyperGraphs build on this by applying iterated powerset constructions to model hierarchical connections among edges [5, 10, 12, 24–26]. These frameworks are employed in molecular modeling, network analysis, and signal processing [27–29].

Definition 2.1 (Base Set)**.** Let S be a nonempty finite set, called the *base set*. All further constructions are drawn from S.

Definition 2.2 (Powerset)**.** The *powerset* of S, denoted $\mathcal{P}(S)$, is the collection of all subsets of S, including the empty set:

$$\mathcal{P}(S) \;=\; \{\, A \mid A \subseteq S \}.$$

Definition 2.3 (Hypergraph)**.** [3, 30] A *hypergraph* $H = (V, E)$ consists of

- a finite vertex set V,
- a finite family $E \subseteq \mathcal{P}(V) \setminus \{\emptyset\}$ of nonempty subsets of V, called *hyperedges*.

Hypergraphs generalize ordinary graphs by permitting edges to join any number of vertices simultaneously.

Example 2.4 (Co-authorship Hypergraph)**.** Consider a set of researchers

$$V = \{\text{Alice, Bob, Carol, Dave}\}.$$

Suppose there are three joint publications:

- Paper 1 co-authored by Alice and Bob,
- Paper 2 co-authored by Bob, Carol, and Dave,
- Paper 3 co-authored by Alice and Dave.

We represent this as a hypergraph $H = (V, E)$ with hyperedges

$$E = \{\, e_1, e_2, e_3 \},$$

where

$$e_1 = \{\text{Alice}, \text{Bob}\}, \quad e_2 = \{\text{Bob}, \text{Carol}, \text{Dave}\}, \quad e_3 = \{\text{Alice}, \text{Dave}\}.$$

Each hyperedge e_i corresponds to the set of co-authors on Paper i. Since each e_i is a nonempty subset of V, H satisfies Definition 2.3 and models the co-authorship relationships as a hypergraph.

Definition 2.5 (Iterated Powerset)**.** [31, 32] For $k \geq 0$, define

$$\mathcal{P}^0(S) = S, \quad \mathcal{P}^{k+1}(S) = \mathcal{P}\big(\mathcal{P}^k(S)\big).$$

Thus $\mathcal{P}^k(S)$ is the k-fold application of the powerset operator.

Definition 2.6 (n-SuperHyperGraph)**.** [11, 33, 34] Let $n \geq 1$ and S be a base set. An *n-superhypergraph* is a pair

$$\mathrm{SHG}^{(n)} = (V, E), \quad V, E \subseteq \mathcal{P}^n(S),$$

where each element of V is called an *n-supervertex* and each element of E an *n-superedge*. Informally, superhypergraphs capture hierarchical relationships by iterating the hypergraph construction.

Example 2.7 (Cross–Department Committees as a 2-SuperHyperGraph)**.** Let

$$S = \{\text{Alice}, \text{Bob}, \text{Carol}, \text{Dave}\}$$

be the set of all employees. At the first level, we form *teams*:

$$T_1 = \{\text{Alice}, \text{Bob}\}, \quad T_2 = \{\text{Bob}, \text{Carol}\}, \quad T_3 = \{\text{Carol}, \text{Dave}\},$$

and set

$$V_1 = \{T_1, T_2, T_3\} \subseteq \mathcal{P}(S).$$

At the second level, we group teams into *departments*:

$$D_A = \{T_1, T_2\}, \quad D_B = \{T_2, T_3\}, \quad V_2 = \{D_A, D_B\} \subseteq \mathcal{P}^2(S).$$

Finally, we describe *cross–department committees* as hyperedges in $\mathcal{P}(V_2) = \mathcal{P}^3(S)$:

$$C_1 = \{D_A, D_B\}, \quad C_2 = \{D_A\}, \quad E_2 = \{C_1, C_2\}.$$

Here C_1 represents a committee that spans both departments, and C_2 a committee internal to department D_A. Since

$$V_2, E_2 \subseteq \mathcal{P}^2(S),$$

the pair

$$\text{SHG}^{(2)} = (V_2, E_2)$$

meets the requirements of Definition 2.6 and is a concrete instance of a 2-SuperHyperGraph.

2.2 Dependency Graph

A Dependency graph is a finite directed acyclic graph whose vertices denote tasks and whose edges indicate each task's prerequisite(cf. [35–37]).

Definition 2.8 (Dependency Graph)**.** (cf. [35–37]) A *Dependency graph* is a directed graph

$$G = (V, E), \quad E \subseteq V \times V,$$

together with the following interpretation:

- Each vertex $v \in V$ represents a *task* (or module, component, etc.).
- A directed edge $(u, v) \in E$ indicates that v *depends* on u, i.e. task v cannot begin until task u has completed.

We require G to be *acyclic*, i.e. it contains no directed cycle, so that dependencies do not form contradictions.

Example 2.9 (Cake Baking Process)**.** Consider the following Dependency graph $G = (V, E)$ modeling the steps to bake a simple cake:

$$V = \{v_1, \ldots, v_8\}, \quad E = \{(v_1, v_3), (v_2, v_5), (v_3, v_4), (v_4, v_5), (v_5, v_6), (v_6, v_7), (v_7, v_8)\}.$$

Here the vertices represent tasks:

- v_1: Gather ingredients,
- v_2: Preheat oven,
- v_3: Mix batter,
- v_4: Pour batter into pan,
- v_5: Bake cake,

- v_6: Cool cake,
- v_7: Apply frosting,
- v_8: Serve cake.

The edge $(v_i, v_j) \in E$ indicates that task v_j depends on completion of task v_i. This directed graph is acyclic and captures exactly the prerequisite order of the baking process.

Definition 2.10 (Partial Order and Topological Order)**.** The *reachability relation* in a Dependency graph $G = (V, E)$ defines a partial order $\leq_G$ on V, where $u \leq_G v$ if and only if there is a (directed) path from u to v. A *topological ordering* is a bijection

$$\pi : V \longrightarrow \{1, 2, \ldots, |V|\}$$

such that whenever $(u, v) \in E$, we have $\pi(u) < \pi(v)$. Existence of a topological ordering is equivalent to G being acyclic.

Definition 2.11 (Path and Ancestors/Descendants)**.** A *path* in $G = (V, E)$ is a sequence of distinct vertices

$$u = v_0 \to v_1 \to \cdots \to v_k = v$$

with each $(v_{i-1}, v_i) \in E$. We say u is an *ancestor* of v (and v a *descendant* of u) whenever $u \leq_G v$.

3 Review and Results: Dependency HyperGraph

A Dependency hypergraph is a hypergraph where each hyperedge relates a nonempty set of prerequisite vertices to exactly one dependent vertex.

Definition 3.1 (Directed Hypergraph)**.** A *directed hypergraph* is a pair

$$H = (V, E),$$

where

- V is a finite set of *vertices*,
- E is a finite set of *hyperedges*, and each hyperedge $e \in E$ is an ordered pair

$$e = \big(t(e),\ h(e)\big) \quad \text{with} \quad t(e) \subseteq V,\ h(e) \subseteq V, \quad t(e) \neq \emptyset,\ h(e) \neq \emptyset.$$

The set $t(e)$ is called the *tail* of e and $h(e)$ the *head* of e.

Definition 3.2 (Dependency Hypergraph)**.** A *Dependency hypergraph* is a directed hypergraph

$$D = (V, E)$$

satisfying the following two conditions:

(i) Single–head. For every $e \in E$, the head is a singleton:

$$\big|h(e)\big| = 1.$$

We write $h(e) = \{v\}$ and think of v as depending on all tasks in $t(e)$.

(ii) Acyclicity. Define a relation $\leq_D$ on V by declaring $u \leq_D v$ if there is a finite sequence of hyperedges

$$e_1, \ldots, e_k \in E$$

and vertices $v_0, \ldots, v_k$ with $v_0 = u$, $v_k = v$, and

$$v_{i-1} \in t(e_i), \quad h(e_i) = \{v_i\}, \quad i = 1, \ldots, k.$$

Then $\leq_D$ is required to be a *partial order* (i.e. if $u \leq_D v$ and $v \leq_D u$ then $u = v$).

In this setting, each $t(e)$ is the set of *prerequisites* for the single dependent task in $h(e)$.

Example 3.3 (Software Build Process)**.** Consider a small software project with the following tasks:

$$V = \{ v_1, v_2, v_3, v_4, v_5\},$$

where

- v_1: generate parser code from grammar,
- v_2: compile parser,
- v_3: generate lexer code from specifications,
- v_4: compile lexer,
- v_5: link parser and lexer into executable.

We model this as a Dependency hypergraph $D = (V, E)$ with hyperedges

$$E = \{ e_1,\ e_2,\ e_3\},$$

defined by

$$e_1 : \big(t(e_1) = \{v_1\},\ h(e_1) = \{v_2\}\big), \quad e_2 : \big(t(e_2) = \{v_3\},\ h(e_2) = \{v_4\}\big),$$
$$e_3 : \big(t(e_3) = \{v_2, v_4\},\ h(e_3) = \{v_5\}\big).$$

Here:

- e_1 and e_2 encode the fact that compilation of parser or lexer requires first generating their code,
- e_3 encodes that linking v_5 depends on both compiled parser (v_2) and compiled lexer (v_4).

One checks immediately that each $h(e_i)$ is a singleton and that there is no directed cycle, so D is indeed a Dependency hypergraph.

Example 3.4 (PCR Experiment Workflow)**.** Consider the following Dependency hypergraph $D = (V, E)$ modeling a polymerase chain reaction (PCR) workflow:

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}, \quad E = \{e_1, e_2, e_3\}.$$

Interpretation of vertices:

- v_1: Collect biological samples,
- v_2: Extract genomic DNA,
- v_3: Design and synthesize primers,
- v_4: Prepare PCR master mix (polymerase, dNTPs, buffer),
- v_5: Run thermocycler reaction,
- v_6: Analyze PCR products by gel electrophoresis.

Define hyperedges by their tail and head sets:

$$e_1 : \big(t(e_1) = \{v_1\},\ h(e_1) = \{v_2\}\big), \quad e_2 : \big(t(e_2) = \{v_2, v_3, v_4\},\ h(e_2) = \{v_5\}\big), \quad e_3 : \big(t(e_3) = \{v_5\},\ h(e_3) = \{v_6\}\big).$$

Here:

- e_1 encodes that DNA extraction (v_2) depends on collecting samples (v_1).
- e_2 encodes that running the PCR (v_5) requires DNA (v_2), primers (v_3), and master mix (v_4).
- e_3 encodes that gel analysis (v_6) depends on completing the PCR run (v_5).

Since each head $h(e_i)$ is a singleton and there are no directed cycles among vertices, D satisfies the conditions of Definition 3.2 and is a valid Dependency hypergraph.

Theorem 3.5 (Generalization of Dependency Graphs)**.** *Every ordinary (directed, acyclic) Dependency graph*

$$G = (V,\, E_G), \quad E_G \subseteq V \times V,$$

embeds naturally as a Dependency hypergraph $D = (V, E_D)$ by setting

$$E_D \;=\; \big\{\big(\{u\},\, \{v\}\big) \;\big|\; (u, v) \in E_G\big\}.$$

Proof. Let $G = (V, E_G)$ be an acyclic Dependency graph. Define

$$E_D = \{\, e_{u,v} \mid (u, v) \in E_G\}, \quad e_{u,v} = \big(\{u\}, \{v\}\big).$$

Then each $e_{u,v}$ has a nonempty tail $\{u\}$ and a singleton head $\{v\}$, so $D = (V, E_D)$ satisfies the single-head condition of Definition 3.2. Moreover, any directed cycle in D would give rise to a sequence of edges

$$e_{v_0,v_1},\; e_{v_1,v_2},\; \ldots,\; e_{v_{k-1},v_k},\; e_{v_k,v_0},$$

which in turn corresponds exactly to the directed cycle

$$(v_0, v_1),\, (v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, v_0)$$

in G. Since G is acyclic, no such cycle can exist, so D is acyclic as well. Hence D is a Dependency hypergraph that extends G verbatim. □

Theorem 3.6 (Underlying Hypergraph)**.** *Let $D = (V, E)$ be any Dependency hypergraph (Definition 3.2). Then the pair*

$$H = \big(V,\, E_H\big), \quad E_H = \{\, t(e) \,\cup\, h(e) \;\big|\; e \in E\}$$

is an (undirected) hypergraph in the sense that each element of E_H is a nonempty subset of V.

Proof. By Definition 3.1, for every hyperedge $e \in E$ we have $t(e) \neq \emptyset$ and $h(e) \neq \emptyset$. Therefore

$$t(e) \,\cup\, h(e) \;\neq\; \emptyset,$$

and $t(e) \cup h(e) \subseteq V$. Hence E_H is a family of nonempty subsets of V. This exactly matches the standard definition of a hypergraph (see Definition 2.3), so H is an (undirected) hypergraph. □

4 Results: Dependency SuperHyperGraph

A Dependency superhypergraph is an n-level acyclic hypergraph whose vertices lie in iterated powersets and whose superedges encode multi-layer dependencies.

Definition 4.1 (Dependency n-SuperHyperGraph)**.** Let V_0 be a finite base set and define iteratively

$$\mathcal{P}^0(V_0) = V_0, \quad \mathcal{P}^{k+1}(V_0) = \mathcal{P}\big(\mathcal{P}^k(V_0)\big).$$

A *Dependency n-SuperHyperGraph* is a tuple

$$D^{(n)} \;=\; \big(V_n,\; E_n,\; t,\; h\big)$$

where

- $V_n \subseteq \mathcal{P}^n(V_0)$ is the set of *n-supervertices*,
- $E_n \subseteq \mathcal{P}^n(V_0)$ is the set of *n-superedges*,
- $t\colon E_n \to \mathcal{P}(V_n)$ assigns to each $e \in E_n$ its *tail* $t(e) \subseteq V_n$,
- $h\colon E_n \to \mathcal{P}(V_n)$ assigns its *head* $h(e) \subseteq V_n$,

subject to:

(i) *Single-head:* for every $e \in E_n$, the head is a singleton,

$$\big|\,h(e)\big| = 1.$$

(ii) *Acyclicity:* define a relation $\leq_{D^{(n)}}$ on V_n by

$$u \leq_{D^{(n)}} v \quad \Longleftrightarrow \quad \exists\, e_1, \ldots, e_k \in E_n,\ \exists\, v_0, \ldots, v_k \in V_n :$$
$$v_0 = u,\ v_k = v, \quad v_{i-1} \in t(e_i),\ h(e_i) = \{v_i\},\ i = 1, \ldots, k.$$

Then $\leq_{D^{(n)}}$ must be a partial order (i.e. $u \leq v \leq u \implies u = v$).

Example 4.2 (Software Project Phase Dependencies)**.** Consider a software project whose atomic tasks are

$$V_0 = \{\, v_1, v_2, v_3\},$$

where

$$v_1 : \text{Write code}, \quad v_2 : \text{Write documentation}, \quad v_3 : \text{Run tests}.$$

At the first super-level, define the *phases* as

$$S_{\text{dev}} = \{v_1, v_2\}, \quad S_{\text{test}} = \{v_3\}, \quad V_1 = \{\, S_{\text{dev}}, S_{\text{test}}\,\} \subseteq \mathcal{P}(V_0).$$

At the second super-level, define the *milestone* and wrap singleton phase-sets:

$$M = \{\, S_{\text{dev}}, S_{\text{test}}\,\}, \quad V_2 = \big\{\, \{S_{\text{dev}}\},\ \{S_{\text{test}}\},\ M\,\big\} \subseteq \mathcal{P}^2(V_0).$$

We now introduce superedges in $E_2 \subseteq \mathcal{P}^2(V_0)$ via their tails and heads:

$$e_1 :\ \big(t(e_1) = \{\{S_{\text{dev}}\}\},\ h(e_1) = \{\{S_{\text{test}}\}\}\big),$$
$$e_2 :\ \big(t(e_2) = \{\{S_{\text{dev}}\}, \{S_{\text{test}}\}\},\ h(e_2) = \{M\}\big).$$

Interpretation:

- e_1 encodes that the testing phase S_{test} depends on completion of the development phase S_{dev}.
- e_2 encodes that the project milestone M depends on both development and testing phases.

Since each head $h(e_i)$ is a singleton and the induced relation on V_2 is acyclic, the tuple

$$D^{(2)} = \big(V_2, \{e_1, e_2\}, t, h\big)$$

satisfies Definition 4.1 and is thus a valid Dependency 2-SuperHyperGraph.

Example 4.3 (House Construction Phases)**.** We model a simplified house construction process as a Dependency 2-SuperHyperGraph $D^{(2)} = (V_2, E_2, t, h)$.

Base tasks (V_0)

$$V_0 = \{\, t_1 : \text{lay foundation},\ t_2 : \text{build walls},\ t_3 : \text{install roof},\ t_4 : \text{paint house}\}.$$

First-level supervertices ($V_1 \subseteq \mathcal{P}(V_0)$)

$$P_{\text{struct}} = \{t_1, t_2, t_3\}, \qquad P_{\text{finish}} = \{t_4\},$$
$$V_1 = \{\, P_{\text{struct}},\, P_{\text{finish}} \,\}.$$

Second-level supervertices ($V_2 \subseteq \mathcal{P}(V_1)$)

$$v_a = \{P_{\text{struct}}\}, \quad v_b = \{P_{\text{finish}}\}, \quad v_c = \{P_{\text{struct}}, P_{\text{finish}}\},$$
$$V_2 = \{\, v_a,\, v_b,\, v_c \,\}.$$

Second-level superedges ($E_2 \subseteq \mathcal{P}(V_1)$) Define two superedges $e_1, e_2 \in E_2$ by

$$e_1 :\ t(e_1) = \{v_a\},\ h(e_1) = \{v_b\}, \qquad e_2 :\ t(e_2) = \{v_a, v_b\},\ h(e_2) = \{v_c\}.$$

Interpretation

- e_1 encodes that the finishing phase (v_b) depends on completion of the structural phase (v_a).
- e_2 encodes that the overall completion (v_c) depends on both structural and finishing phases.

Verification Each head $h(e_i)$ is a singleton in V_2, and there is no directed cycle among $v_a \to v_b \to v_c$, so the induced relation on V_2 is a partial order. Hence $D^{(2)}$ satisfies Definition 4.1 and is a valid Dependency 2-SuperHyperGraph.

Theorem 4.4 (Generalization of Dependency Graphs)**.** *Every ordinary acyclic Dependency graph*

$$G = (V, E_G), \quad E_G \subseteq V \times V,$$

embeds as a Dependency 1*-SuperHyperGraph*

$$D^{(1)} = \big(V_1, E_1, t, h\big)$$

by taking

$$V_1 = \big\{\{v\} \mid v \in V\big\} \ \subseteq\ \mathcal{P}(V), \quad E_1 = \big\{\{u, v\} \mid (u, v) \in E_G\big\} \ \subseteq\ \mathcal{P}(V),$$

and defining for each $e = \{u, v\} \in E_1$,

$$t(e) = \{\{u\}\}, \quad h(e) = \{\{v\}\}.$$

Proof. Since G is acyclic, no directed cycle $(v_0 \to v_1 \to \cdots \to v_0)$ exists. In $D^{(1)}$, each $e = \{u, v\}$ has nonempty tail $\{\{u\}\} \subseteq V_1$ and singleton head $\{\{v\}\}$, so Definition 4.1(i) holds. Any cycle in $D^{(1)}$ would correspond exactly to a cycle in G by projecting $\{u\} \mapsto u$, so acyclicity (ii) is preserved. Thus $D^{(1)}$ is a Dependency 1-SuperHyperGraph extending G. □

Theorem 4.5 (Generalization of Dependency HyperGraphs)**.** *Every Dependency hypergraph*

$$H = (V, E_H, t_H, h_H)$$

(as in Definition 3.2) embeds as a Dependency 1*-SuperHyperGraph*

$$D^{(1)} = \big(V_1, E_1, t, h\big)$$

by setting

$$V_1 = \big\{\{v\} \mid v \in V\big\}, \quad E_1 = \big\{T \cup H \mid e \in E_H,\ T = t_H(e),\ H = h_H(e)\big\},$$

and, for each $e' = T \cup H \in E_1$,

$$t(e') = \big\{\{u\} \mid u \in T\big\}, \quad h(e') = \big\{\{v\} \mid v \in H\big\}.$$

Proof. Since H satisfies single-head and acyclicity on V, the same properties hold in $D^{(1)}$ after replacing each vertex v with the singleton $\{v\}$. Tails and heads remain nonempty and singletons respectively, and any cycle in $D^{(1)}$ projects to a cycle in H, which cannot exist. □

Theorem 4.6 (Underlying SuperHyperGraph)**.** *Let $D^{(n)} = (V_n, E_n, t, h)$ be a Dependency n-SuperHyperGraph (Definition 4.1). Then the undirected hypergraph*

$$H^{(n+1)} = \big(V_n,\ E'\big), \quad E' = \big\{\, t(e) \cup h(e) \mid e \in E_n \big\}$$

is an $(n+1)$-SuperHyperGraph, since

$$V_n \subseteq \mathcal{P}^n(V_0), \quad t(e) \cup h(e) \ \subseteq\ V_n \ \subseteq\ \mathcal{P}^n(V_0) \implies E' \subseteq \mathcal{P}(V_n) = \mathcal{P}^{n+1}(V_0).$$

Proof. By Definition 4.1, each $t(e), h(e) \subseteq V_n \subseteq \mathcal{P}^n(V_0)$ and both are nonempty. Hence $t(e) \cup h(e) \neq \emptyset$ and lies in $\mathcal{P}^n(V_0)$. Therefore $E' \subseteq \mathcal{P}(V_n) = \mathcal{P}^{n+1}(V_0)$, showing (V_n, E') is an $(n+1)$-SuperHyperGraph. □

Theorem 4.7 (Induced Sub–SuperHyperGraph)**.** *Let $D^{(n)} = (V_n, E_n, t, h)$ be a Dependency n-SuperHyperGraph (Definition 4.1), and let $U \subseteq V_n$ be any nonempty subset of supervertices. Define*

$$E' \ =\ \big\{\, e \in E_n \ \big|\ t(e) \cup h(e) \ \subseteq\ U \big\},$$

and restrict the tail and head maps:

$$t' \ =\ t\big|_{E'}, \quad h' \ =\ h\big|_{E'}$$

Then $D'^{(n)} = (U, E', t', h')$ is itself a Dependency n-SuperHyperGraph.

Proof. Since each $e \in E'$ satisfies $t(e) \cup h(e) \subseteq U \subseteq V_n$, the images of t' and h' lie in $\mathcal{P}(U)$. The single–head condition is inherited verbatim from $D^{(n)}$, and any directed cycle in $D'^{(n)}$ would already be a cycle in $D^{(n)}$, contradicting its acyclicity. Hence $D'^{(n)}$ meets both requirements of Definition 4.1. □

Theorem 4.8 (Existence of Minimal Supervertices)**.** *In every finite Dependency n-SuperHyperGraph $D^{(n)} = (V_n, E_n, t, h)$ with $V_n \neq \emptyset$, there exists at least one* minimal *supervertex*

$$u \in V_n \quad \textit{such that} \quad \nexists\, e \in E_n :\ h(e) = \{u\}.$$

Equivalently, there is an element $u \in V_n$ minimal with respect to the partial order $\leq_{D^{(n)}}$.

Proof. The relation $\leq_{D^{(n)}}$ defined in Definition 4.1(ii) is a partial order on the finite set V_n. A standard result in order theory states that every nonempty finite poset contains at least one minimal element. Concretely, if no minimal supervertex existed, one could choose $u_0 \in V_n$ and find $u_1 < u_0$, then $u_2 < u_1$, and so on, producing an infinite strictly descending chain, which is impossible in a finite set. Hence a minimal supervertex exists. □

Theorem 4.9 (Linear Extension / Topological Sort)**.** *Every finite Dependency n-SuperHyperGraph $D^{(n)} = (V_n, E_n, t, h)$ admits a* topological ordering*: there is a bijection*

$$\pi : V_n \ \longrightarrow\ \{1, 2, \ldots, |V_n|\}$$

such that whenever $u <_{D^{(n)}} v$, one has $\pi(u) < \pi(v)$.

Proof. Since $\leq_{D^{(n)}}$ is a partial order on the finite set V_n, by the classical theorem on linear extensions (see, e.g., any standard text on posets) there exists a total order on V_n that extends $\leq_{D^{(n)}}$. Label the elements of V_n in increasing order to obtain the desired bijection π. This ordering is often constructed by repeatedly selecting and removing minimal elements (Theorem 4.8) and assigning them the next available label. □

Theorem 4.10 (Immediate–Dependency Relation)**.** *Define the* covering relation $\prec$ *on V_n by*

$$u \prec v \quad \Longleftrightarrow \quad \exists\, e \in E_n :\ u \in t(e),\ h(e) = \{v\}.$$

Then the partial order $\leq_{D^{(n)}}$ is precisely the reflexive transitive closure of $\prec$.

Proof. By Definition 4.1(ii), $u \leq_{D^{(n)}} v$ means exactly that there is a sequence of hyperedges $e_1, \ldots, e_k$ producing a chain of vertices $u = v_0, v_1, \ldots, v_k = v$ with each $v_{i-1} \prec v_i$. Reflexivity is clear by taking $k = 0$, and transitivity follows by concatenating such chains. Conversely, any chain of coverings yields a sequence of hyperedges witnessing $u \leq_{D^{(n)}} v$. Thus $\leq_{D^{(n)}}$ is the reflexive transitive closure of $\prec$. □

5 Conclusion and Future Works

In this paper, we extended the classical concept of dependency graphs by introducing *Dependency Hypergraphs* and *Dependency SuperHypergraphs*, and we explored their structural and theoretical properties in detail.

As future work, we aim to further generalize these models by incorporating uncertainty-based frameworks such as Fuzzy Sets [38], HyperFuzzy Sets [39], Intuitionistic Fuzzy Sets [40], Hesitant Fuzzy Sets [41], Neutrosophic Sets [42], and Plithogenic Sets [43]. Such extensions would enhance the expressiveness of dependency modeling in uncertain or multi-valued environments.

Funding

The authors received no grants, contracts, or in-kind contributions for this study.

Acknowledgements

We are indebted to the colleagues, reviewers, and mentors whose suggestions sharpened our arguments. We also recognise the authors cited throughout the paper for providing the intellectual groundwork on which this study builds. Finally, we thank the institutions and individuals who supplied the computing resources and working environment that made the writing of this manuscript possible.

Data Availability

The work is conceptual and does not rely on empirical data; consequently, no datasets were generated or analysed. Researchers interested in empirical validation are encouraged to design their own studies using the theoretical framework presented here.

Ethical Approval

Because the research involves no human or animal subjects, formal ethical approval is unnecessary.

Conflicts of Interest

The authors declare that they have no competing financial or non-financial interests.

Disclaimer

The results and views expressed are those of the authors alone. Although care has been taken to ensure accuracy and proper citation, inadvertent errors may remain. Readers should verify any critical information independently, and practitioners should test the proposed ideas in real-world contexts before implementation.

References

[1] Reinhard Diestel. *Graph theory*. Springer (print edition); Reinhard Diestel (eBooks), 2024.

[2] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018.

[3] Claude Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.

[4] Florentin Smarandache. *Introduction to the n-SuperHyperGraph-the most general form of graph today*. Infinite Study, 2022.

[5] Mohammad Hamidi, Florentin Smarandache, and Mohadeseh Taghinezhad. *Decision Making Based on Valued Fuzzy Superhypergraphs*. Infinite Study, 2023.

[6] Reinhard Diestel. Graph theory 3rd ed. *Graduate texts in mathematics*, 173(33):12, 2005.

[7] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3558–3565, 2019.

[8] Derun Cai, Moxian Song, Chenxi Sun, Baofeng Zhang, Shenda Hong, and Hongyan Li. Hypergraph structure learning for hypergraph neural networks. In *IJCAI*, pages 1923–1929, 2022.

[9] Takaaki Fujita and Florentin Smarandache. Fundamental computational problems and algorithms for superhypergraphs. *HyperSoft Set Methods in Engineering*, 3:32–61, 2025.

[10] N. B. Nalawade, M. S. Bapat, S. G. Jakkewad, G. A. Dhanorkar, and D. J. Bhosale. Structural properties of zero-divisor hypergraph and superhypergraph over $\mathbb{Z}_n$: Girth and helly property. *Panamerican Mathematical Journal*, 35(4S):485, 2025.

[11] Florentin Smarandache. *Introduction to the n-SuperHyperGraph-the most general form of graph today*. Infinite Study, 2022.

[12] Takaaki Fujita and Florentin Smarandache. A concise study of some superhypergraph classes. *Neutrosophic Sets and Systems*, 77:548–593, 2024.

[13] Mohammad Hamidi and Mohadeseh Taghinezhad. *Application of Superhypergraphs-Based Domination Number in Real World*. Infinite Study, 2023.

[14] Jaime Simão Sichman and Rosaria Conte. Multi-agent dependence by dependence graphs. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 483–490, 2002.

[15] Fang Deng and James A Jones. Weighted system dependence graph. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 380–389. IEEE, 2012.

[16] Apostolos Gerasoulis and Tao Yang. On the granularity and clustering of directed acyclic task graphs. *IEEE Transactions on Parallel and distributed Systems*, 4(6):686–701, 2002.

[17] Sebastian Rötzer, Sebastian Schweigert-Recksiek, Dominik Thoma, and Markus Zimmermann. Attribute dependency graphs: modelling cause and effect in systems design. *Design Science*, 8:e27, 2022.

[18] Željko Agić, Alexander Koller, and Stephan Oepen. Semantic dependency graph parsing using tree approximations. In *Proceedings of the 11th International Conference on Computational Semantics*, pages 217–227, 2015.

[19] Nadathur R Satish, Kaushik Ravindran, and Kurt Keutzer. Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors. In *Proceedings of the 8th ACM international conference on Embedded software*, pages 149–158, 2008.

[20] Muhammad Amber Hassaan, Donald D Nguyen, and Keshav K Pingali. Kinetic dependence graphs. *ACM SIGPLAN Notices*, 50(4):457–471, 2015.

[21] George K Baah, Andy Podgurski, and Mary Jean Harrold. The probabilistic program dependence graph and its application to fault diagnosis. In *Proceedings of the 2008 international symposium on Software testing and analysis*, pages 189–200, 2008.

[22] Takaaki Fujita and Prem Kumar Singh. Hyperfuzzy graph and hyperfuzzy hypergraph. *Journal of Neutrosophic and Fuzzy Systems (JNFS)*, 10(01):01–13, 2025.

[23] Yue Gao, Zizhao Zhang, Haojie Lin, Xibin Zhao, Shaoyi Du, and Changqing Zou. Hypergraph learning: Methods and practices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(5):2548–2566, 2020.

[24] Eduardo Martín Campoverde Valencia, Jessica Paola Chuisaca Vásquez, and Francisco Ángel Becerra Lois. Multineutrosophic analysis of the relationship between survival and business growth in the manufacturing sector of azuay province, 2020–2023, using plithogenic n-superhypergraphs. *Neutrosophic Sets and Systems*, 84:341–355, 2025.

[25] Julio Cesar Méndez Bravo, Claudia Jeaneth Bolanos Piedrahita, Manuel Alberto Méndez Bravo, and Luis Manuel Pilacuan-Bonete. Integrating smed and industry 4.0 to optimize processes with plithogenic n-superhypergraphs. *Neutrosophic Sets and Systems*, 84:328–340, 2025.

[26] Mohammed Alqahtani. Intuitionistic fuzzy quasi-supergraph integration for social network decision making. *International Journal of Analysis and Applications*, 23:137–137, 2025.

[27] Berrocal Villegas Salomón Marcos, Montalvo Fritas Willner, Berrocal Villegas Carmen Rosa, Flores Fuentes Rivera María Yissel, Espejo Rivera Roberto, Laura Daysi Bautista Puma, and Dante Manuel Macazana Fernández. Using plithogenic n-superhypergraphs to assess the degree of relationship between information skills and digital competencies. *Neutrosophic Sets and Systems*, 84:513–524, 2025.

[28] Takaaki Fujita. Hypergraph and superhypergraph approaches in electronics: A hierarchical framework for modeling power-grid hypernetworks and superhypernetworks. *Journal of Energy Research and Reviews*, 17(6):102–136, 2025.

[29] Shouxian Zhu. Neutrosophic n-superhypernetwork: A new approach for evaluating short video communication effectiveness in media convergence. *Neutrosophic Sets and Systems*, 85:1004–1017, 2025.

[30] Alain Bretto. Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer*, 1, 2013.

[31] Florentin Smarandache. *SuperHyperFunction, SuperHyperStructure, Neutrosophic SuperHyperFunction and Neutrosophic SuperHyperStructure: Current understanding and future directions*. Infinite Study, 2023.

[32] Florentin Smarandache. Superhyperstructure & neutrosophic superhyperstructure, 2024. Accessed: 2024-12-01.

[33] Florentin Smarandache. *Extension of HyperGraph to n-SuperHyperGraph and to Plithogenic n-SuperHyperGraph, and Extension of HyperAlgebra to n-ary (Classical-/Neutro-/Anti-) HyperAlgebra*. Infinite Study, 2020.

[34] Takaaki Fujita. Review of probabilistic hypergraph and probabilistic superhypergraph. *Spectrum of Operational Research*, 3(1):319–338, 2025.

[35] Winston Ewert. The dependency graph of life. *BIO-Complexity*, 2018, 2018.

[36] Chadi Kari, Alexander Russell, and Narasimha Shashidhar. Work-competitive scheduling on task dependency graphs. *Parallel Processing Letters*, 25(02):1550001, 2015.

[37] Marco Kuhlmann and Peter Jonsson. Parsing to noncrossing dependency graphs. *Transactions of the Association for Computational Linguistics*, 3:559–570, 2015.

[38] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.

[39] Z Nazari and B Mosapour. The entropy of hyperfuzzy sets. *Journal of Dynamical Systems and Geometric Theories*, 16(2):173–185, 2018.

[40] Krassimir Atanassov and George Gargov. Elements of intuitionistic fuzzy logic. part i. *Fuzzy sets and systems*, 95(1):39–52, 1998.

[41] Vicenç Torra and Yasuo Narukawa. On hesitant fuzzy sets and decision. In *2009 IEEE international conference on fuzzy systems*, pages 1378–1382. IEEE, 2009.

[42] Florentin Smarandache. A unifying field in logics: Neutrosophic logic. In *Philosophy*, pages 1–141. American Research Press, 1999.

[43] Florentin Smarandache. Plithogeny, plithogenic set, logic, probability, and statistics. *arXiv preprint arXiv:1808.03948*, 2018.