Southeast Asian Journal of Sciences, Vol. 11, No 2 (2025), pp. 67-73

INITIAL STUDY ON TRUST COMPUTATION BASED ON TENSOR AND LIE ALGEBRA IN COMPLEX NETWORKS

Dinh Que Tran

Department of Information Technology
Posts and Telecommunications Institute of Technology (PTIT)
Hanoi, Vietnam
E-mail: quetd@ptit.edu.vn

Abstract

Trust computation in complex networks is a crucial topic for systems involving interactions among agents, such as recommendation platforms, multi-agent systems, and social networks. This paper presents a novel framework combining tensor-based feature modeling and Lie algebraic dynamics to formally define and compute trust. We model measures such as dispatch, familiarity, responsibility, and influence in tensor form and propose a trust function based on weighted features. We then use Lie algebra to encode the structural evolution of trust over time and derive key mathematical properties. This approach enables robust and dynamic trust inference in large-scale and evolving networks.

1 Introduction

In complex networks, trust is a dynamic and relational concept that reflects the confidence of one node (agent) in the behavior of another. Traditional models often rely on static features or heuristics, limiting their adaptability and scalability [1, 3, 4, 5, 6, 7, 8, 9]. Recent advances suggest that multi-dimensional

Key words: complex networks, trust model, belief, trusworthiness.
2010 AMS Mathematics classification: 911D30, 91D10, 68U115, 68U35, 68M14, 68M115, 68T99.

representations and geometric approaches can significantly enhance trust modeling. This work leverages tensors to encode trust-relevant features and Lie algebra [12, 10, 11, 13, 14] to model structural and temporal evolution. This paper is an extension of our research work [2]. Our contributions include:

- A multi-dimensional tensor representation for trust features in networks.
- A formal trust definition based on weighted aggregation and conditions.
- An application of Lie algebra to model trust propagation and network evolution.
- Theoretical results including definitions, propositions, and brief proofs.

2 Formal Trust Definition and Verified Function Classes

2.1 General Definition in Tensor Form

Let $\mathcal{T} \in [0,1]^{n\times n\times d}$ be a 3-way trust tensor, where $\mathcal{T}_{i,j,:} = \boldsymbol{x}(i,j) \in [0,1]^d$ denotes the normalized feature vector characterizing the trust-related interaction from node i (trustor) to node j (trustee). Each slice $\boldsymbol{x}(i,j)$ includes d normalized factors such as familiarity, responsibility, dispatching efficiency, influence, etc.

Definition 1 (Trust Function in Tensor Space)**.** *A* trust function *is a mapping*

$$\mathcal{F}_{\text{trust}} : [0,1]^d \to [0,1], \quad \textit{with} \;\; \mathcal{F}_{\text{trust}}(\mathcal{T}_{i,j,:}) = \text{trust}(i,j),$$

that satisfies the following conditions:

(C1) Boundedness: *For all $\boldsymbol{x} \in [0,1]^d$, we have $\mathcal{F}_{\text{trust}}(\boldsymbol{x}) \in [0,1]$.*

(C2) Continuity: *The function $\mathcal{F}_{\text{trust}}$ is continuous on $[0,1]^d$.*

(C3) Differentiability: *$\mathcal{F}_{\text{trust}}$ is continuously differentiable on the open unit cube $(0,1)^d$, i.e., $\mathcal{F}_{\text{trust}} \in \mathcal{C}^1((0,1)^d)$.*

(C4) Monotonicity (Optional): *If a feature x_k positively contributes to trust, then*

$$\frac{\partial \mathcal{F}_{\text{trust}}}{\partial x_k}(\boldsymbol{x}) \geq 0 \quad \textit{for all } \boldsymbol{x} \in (0,1)^d.$$

This tensor-based formulation enables generalized trust modeling in multi-relational networks. It is particularly suitable for integration with deep learning models (e.g., tensorized neural architectures), differentiable optimization, and advanced trust propagation mechanisms across heterogeneous graph data.

2.2 Verified Classes of Trust Functions

Proposition 1 (Linear Weighted Sum Trust Function)**.** *Let* $\boldsymbol{w} \in \mathbb{R}^d$ *with* $w_k \geq 0$ *and* $\sum_{k=1}^{d} w_k = 1$*, and define:*

$$\mathcal{F}_{\text{trust}}(\boldsymbol{x}) = \sigma\left(\sum_{k=1}^{d} w_k x_k\right), \quad \text{where } \sigma(z) = \frac{1}{1+e^{-z}}.$$

Then $\mathcal{F}_{\text{trust}}$ *satisfies conditions (C1)–(C4).*

Proof. The sigmoid function maps $\mathbb{R}$ into $(0,1)$, ensuring (C1). The composition of a linear map and sigmoid is continuous (C2) and continuously differentiable (C3). Since $\sigma'(z) > 0$ and $w_k \geq 0$, we have

$$\frac{\partial \mathcal{F}_{\text{trust}}}{\partial x_k} = \sigma'\left(\sum w_k x_k\right) w_k \geq 0,$$

verifying (C4). □

Proposition 2 (Nonlinear Squashed Function)**.** *Let* $f : [0,1]^d \to \mathbb{R}$ *be a differentiable nonlinear function (e.g., a polynomial), and let* $\phi : \mathbb{R} \to [0,1]$ *be a differentiable squashing function (e.g., sigmoid or scaled* tanh*). Define:*

$$\mathcal{F}_{\text{trust}}(\boldsymbol{x}) = \phi(f(\boldsymbol{x})).$$

Then $\mathcal{F}_{\text{trust}}$ *satisfies (C1)–(C3). Monotonicity (C4) holds if* f *is monotonic in each* x_k*.*

Proof. Since both f and ϕ are differentiable, their composition is continuously differentiable (C3). The output of ϕ lies in $[0,1]$ (C1), and both functions are continuous (C2). The derivative is given by:

$$\frac{\partial \mathcal{F}_{\text{trust}}}{\partial x_k} = \phi'(f(\boldsymbol{x})) \cdot \frac{\partial f}{\partial x_k},$$

which is non-negative if ϕ' and $\frac{\partial f}{\partial x_k}$ are non-negative, satisfying (C4). □

Proposition 3 (Probabilistic Trust Model)**.** *Let* $\theta_{i,j} \in \{0,1\}$ *be a latent binary trustworthiness variable, and define:*

$$\mathcal{F}_{\text{trust}}(\boldsymbol{x}) = \mathbb{P}(\theta_{i,j} = 1 \mid \boldsymbol{x}),$$

modeled using differentiable logistic regression:

$$\mathbb{P}(\theta_{i,j} = 1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b).$$

Then $\mathcal{F}_{\text{trust}}$ *satisfies (C1)–(C4).*

Proof. The sigmoid output lies in $(0, 1)$ (C1). The function is smooth and differentiable (C2–C3). The partial derivative is:

$$\frac{\partial \mathcal{F}_{\text{trust}}}{\partial x_k} = \sigma'(\boldsymbol{w}^\top \boldsymbol{x} + b) w_k \geq 0,$$

assuming $w_k \geq 0$, so monotonicity (C4) is satisfied. □

Proposition 4 (Neural Network Trust Function)**.** *Let* $\mathcal{F}_{\text{trust}}(\boldsymbol{x}) = NN_\theta(\boldsymbol{x})$[1] *be a neural network with differentiable activation functions The neural network has learned to map patterns in the input data to predicted trust scores. (e.g., ReLU, sigmoid, tanh), and a final sigmoid output layer. Then* $\mathcal{F}_{\text{trust}}$ *satisfies (C1)–(C3), and optionally (C4).*

Proof. The final sigmoid ensures bounded output in $(0, 1)$ (C1). The composition of differentiable functions ensures (C2)–(C3). Monotonicity (C4) can be achieved via constraints, monotonic architectures, or post-hoc analysis of $\nabla \mathcal{F}_{\text{trust}}$. □

Proposition 5 (Hybrid Rule-Based Trust Function)**.** *Let* $\mathcal{F}_{\text{trust}}(\boldsymbol{x}) = RuleSet(\boldsymbol{x}) \cdot NN_\theta(\boldsymbol{x})$*, where RuleSet is piecewise differentiable with values in* $[0, 1]$*, and* NN_θ *is a differentiable neural network as above. Then* $\mathcal{F}_{\text{trust}}$ *satisfies (C1)–(C3). (C4) depends on both components.*

Proof. Both components are bounded in $[0, 1]$, so the product is also bounded (C1). If RuleSet and NN_θ are differentiable almost everywhere, the product is differentiable almost everywhere (C3), and continuous (C2). Monotonicity (C4) depends on whether both components are non-decreasing in x_k. □

2.3 Discussion

By enforcing continuous differentiability, this framework guarantees that trust functions can be smoothly integrated into neural architectures, dynamic equations, and optimization problems. Depending on application needs, trust models may emphasize simplicity (linear), expressivity (neural), interpretability (rules), or uncertainty modeling (probabilistic).

3 Lie Algebra for Trust Computing

3.1 Lie Algebra Background

Definition 2 (Lie Algebra)**.** *Let* $\mathfrak{g}$ *be a vector space over* $\mathbb{R}$ *equipped with a bilinear map* $[\cdot, \cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$ *called the Lie bracket. Then* $(\mathfrak{g}, [\cdot, \cdot])$ *is a* Lie algebra *if the following axioms hold for all* $X, Y, Z \in \mathfrak{g}$*:*

[1] This refers to a neural network (NN) parameterized by weights θ which takes $\boldsymbol{x}$ as input and produces a trust value as output.

(L1) Bilinearity: $[aX + bY, Z] = a[X, Z] + b[Y, Z]$ *for all* $a, b \in \mathbb{R}$.

(L2) Antisymmetry: $[X, Y] = -[Y, X]$.

(L3) Jacobi Identity: $[X, [Y, Z]] + [Y, [Z, X]] + [Z, [X, Y]] = 0$.

In trust modeling, $\mathfrak{g}$ is a space of generators describing system dynamics or structural interactions between agents, and $G(t) \in \mathrm{Lie}(G)$ is a trust transition matrix at time t.

3.2 Trust Modeling via Lie Generator

Let $G(t) \in \mathbb{R}^{n\times n}$ represent the trust propagation matrix over time t, and define the generator of the trust evolution as:

$$\mathfrak{g}(t) = \frac{dG(t)}{dt} G(t)^{-1}.$$

This yields:

$$G(t) = \exp(t\mathfrak{g})G(0),$$

where $\mathfrak{g} \in \mathbb{R}^{n\times n}$ is the Lie algebra generator encoding structural and relational dynamics.

Define trust between node i and node j at time t as:

$$\mathcal{F}^{\mathrm{Lie}}_{\mathrm{trust}}(i, j, t) = \sigma\left([G(t)]_{ij}\right), \quad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

3.3 Propositions

Proposition 6 (Lie Algebra Trust Function Satisfies Trust Definition)**.** *Let* $G(t) = \exp(t\mathfrak{g})G(0)$ *with bounded initial trust matrix* $G(0) \in [0, 1]^{n\times n}$ *and Lie generator* $\mathfrak{g}$ *such that* $G(t)_{ij} \in \mathbb{R}$ *is continuous in* t*. Then the function*

$$\mathcal{F}^{\mathrm{Lie}}_{\mathrm{trust}}(i, j, t) = \sigma(G(t)_{ij})$$

satisfies (C1)–(C3) in Definition 1.

Proof. Since $G(t)$ is differentiable in t by the matrix exponential of a fixed generator, $G(t)_{ij}$ is smooth and thus continuous (C2) and differentiable (C3). The sigmoid function maps $\mathbb{R} \to (0, 1)$, hence the composed function is bounded (C1). □

Proposition 7 (Feature-Dependent Lie Generator Trust Model)**.** *Let* $\boldsymbol{x}(i, j) \in [0, 1]^d$ *be a feature vector, and define the Lie generator as:*

$$\mathfrak{g}_{i,j} = \sum_{k=1}^{d} w_k x_k(i, j) A_k, \quad \textit{with } A_k \in \mathbb{R}^{n\times n},\ w_k \geq 0,$$

where A_k are structural basis matrices (e.g., dispatch, response, influence). Then the trust function:

$$\mathcal{F}^{\text{Lie}}_{\text{trust}}(i,j) = \sigma\left([\exp(\mathfrak{g}_{i,j})]_{ij}\right)$$

satisfies conditions (C1)–(C4).

Proof. Each A_k is fixed, and $\mathfrak{g}_{i,j}$ is a linear combination of $\boldsymbol{x}(i,j)$ with non-negative weights, hence differentiable in $\boldsymbol{x}$. The matrix exponential is differentiable with respect to $\mathfrak{g}_{i,j}$, and thus the map $\boldsymbol{x} \mapsto [\exp(\mathfrak{g}_{i,j})]_{ij}$ is differentiable. Composition with sigmoid yields bounded output (C1), continuity (C2), and differentiability (C3). Since $w_k \geq 0$ and exp preserves monotonicity in this construction, partial derivatives are non-negative if all A_k influence trust positively, satisfying (C4). □

4 Conclusion

This paper proposes a model of computing trust in complex networks based on tensors of features collected from nodes (agents) interaction and influence. A class of functions of trust computation on tensors are constructed . The Lie algebra framework introduces a principled way to integrate continuous-time structural dynamics into trust modeling. The matrix exponential guarantees smooth propagation and compatibility with neural network integration. When the generator is defined in terms of input features, it naturally aligns with the formal trust function definition and remains fully differentiable for gradient-based optimization.

References

[1] Wanita Sherchan, Surya Nepal, and Cecile Paris, A Survey of Trust in Social Networks, *ACM Computing Surveys*, vol. 45(4), Article 47, August 2013.

[2] Dinh Que Tran, Phuong Thanh Pham, TreeXTrust: A Tree-Based Explainable Trust Model Using Social Semantics and Influence, *Journal of Computer Science*, AGH University of Science and Technology, 26(2), pp. 1–25, 2025.

[3] Matheus R. F. Mendonca, Andre M. S. Barreto, and Artur Ziviani, Approximating Network Centrality Measures Using Node Embedding and Machine Learning, *arXiv preprint arXiv:2006.16392*, 2020.

[4] J. Tang et al., Social Influence Analysis in Large-scale Networks, *KDD'09*, June 28, 2009.

[5] Bingoi et al., Topic-Based Influence Computation in Social Networks under Resource Constraints, *IEEE Transactions on Services Computing*, vol. PP, no. 99, 2018.

[6] Kan Li et al., Social Influence Analysis: Models, Methods, and Evaluation, *Engineering*, vol. 4, 2018, pp. 40–46.

[7] Dinh Que Tran and Phuong Thanh Pham, Integrating Interaction and Similarity Threshold of User's Interests for Topic Trust Computation, *Southeast-Asian J. of Sciences*, vol. 7(01), 2019, pp. 28–35.

[8] Vedran Podobnik et al., How to Calculate Trust Between Social Network Users?, *SoftCOM 2012 – 20th International Conference on Software, Telecommunications and Computer Networks*, IEEE, pp. 1–6.

[9] Chung-Wei Hang et al., Operators for Propagating Trust and Their Evaluation in Social Networks, *AAMAS*, 2009.

[10] Zheng Du, Min-Hung Chen, and Yung-Hsiang Lu, Tensor-based Trust Evaluation with Lie Algebra for Dynamic Networks, *Journal of Computer Science and Systems*, vol. 19(3), 2023.

[11] Taco Cohen and Max Welling, Group Equivariant Convolutional Networks, *International Conference on Machine Learning (ICML)*, 2016.

[12] Alexander Kirillov, Introduction to Lie Groups and Lie Algebras, Cambridge University Press, 2008.

[13] Mei Lu and Fanzhang Li, Survey on Lie Group Machine Learning, *https://www.sciopen.com/article/10.26599/BDMA.2020.9020011*, 2020.

[14] Sheng Zhang, Hang Su, and Jun Zhu, Learning Graphical Lie Algebra with Deep Structured Embedding, *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.