Taylor & Francis
Taylor & Francis Group

Applied Artificial Intelligence
An International Journal

ISSN: 0883-9514 (Print) 1087-6545 (Online) Journal homepage: https://www.tandfonline.com/loi/uaai20

Neural Tensor Network Training Using Meta-Heuristic Algorithms for RDF Knowledge Bases Completion

Farhad Abedini, Mohammad Reza Keyvanpour & Mohammad Bagher Menhaj

To cite this article: Farhad Abedini, Mohammad Reza Keyvanpour & Mohammad Bagher Menhaj (2019) Neural Tensor Network Training Using Meta-Heuristic Algorithms for RDF Knowledge Bases Completion, Applied Artificial Intelligence, 33:7, 656-667, DOI: 10.1080/08839514.2019.1602317

To link to this article: https://doi.org/10.1080/08839514.2019.1602317

Published online: 16 Apr 2019.

APPLIED ARTIFICIAL INTELLIGENCE
2019, VOL. 33, NO. 7, 656–667
https://doi.org/10.1080/08839514.2019.1602317

Taylor & Francis
Taylor & Francis Group

Neural Tensor Network Training Using Meta-Heuristic Algorithms for RDF Knowledge Bases Completion

Farhad Abedini[a], Mohammad Reza Keyvanpour[b], and Mohammad Bagher Menhaj [c]

[a]Faculty of Computer and Information Technology Engineering, Qazvin Branch, Islamic Azad University, Qazvin, Iran.; [b]Department of Computer Engineering, Alzahra University, Vanak, Iran; [c]Center of Excellence in Control and Robotics, Electrical Engineering Department, Amirkabir University of Technology, Tehran, Iran

ABSTRACT

Neural tensor network (NTN) has been recently introduced to complete Resource Description Framework (RDF) knowledge bases, which has been the state-of-the-art in the field so far. An RDF knowledge base includes some facts from the real world shown as RDF "triples." In the previous methods, an objective function has been used for training this type of network, and the network parameters should have been set in a way to minimize the function. For this purpose, a classic nonlinear optimization method has been used. Since many replications are needed in this method to get the minimum amount of the function, in this paper, we suggest to combine meta-heuristic optimization methods to minimize the replications and increase the speed of training consequently. So, this problem will be improved using some meta-heuristic algorithms in this new approach to specify which algorithm will get the best results on NTN and its results will be compared with the results of the former methods finally.

Introduction

Neural tensor network (NTN) is a generalization of standard neural network, which has been introduced by Socher, for the specific purpose of knowledge bases completion (Socher et al. 2013). An RDF knowledge base includes some real-world facts shown as RDF triples in which each triple is called a fact. Each fact contains two entities and a relation between them. The first entity is called subject, the second as object, and the relation between them as predicate. For example, the fact "Einstein was born in Germany" could be shown as the triple <Einstein, born in, Germany>, in which Einstein is the first and Germany the second entity, and "born in" is the relation between these two entities. YAGO ontologies (Suchanek, Kasneci, and Weikum 2007), DBpedia (Bizer et al. 2009), Freebase (Bizer et al. 2009), and WorldNet (Miller 1995) are some examples of RDF knowledge bases that are also called knowledge graphs (Kim 2017) which could be used as very helpful resources to undertake issues like "Information Retrieval," "Natural

CONTACT Mohammad Reza Keyvanpour Keyvanpour@alzahra.ac.ir Department of Computer Engineering, Alzahra University, Vanak, Iran

Language Processing," "Query Expansion," "Question Answering," "Text Categorization" (Imani, Keyvanpour, and Azmi 2013), etc.

The completion of these knowledge bases is one of the most important issues nowadays and many activities are involved in this field (Weikum, Hoffart, and Suchanek 2016; Zhao et al. 2017). Using NTN for this issue, without utilizing external resources, has been recognized as the state-of-the-art in the area (Abedini et al., 2017; Abedini, Menhaj, and Keyvanpour 2017). Socher compared the outcomes of implementing this model on a set of standard data to other related models (Bordes et al. 2012, 2011; Jenatton et al. 2012; Sutskever, Salakhutdinov, and Tenenbaum 2009; Turian, Ratinov, and Bengio 2010) which resulted in the advantage and supremacy of the model (Socher et al. 2013). The mechanism of this method is first to train the network using available facts in the knowledge base, then to set the network parameters, and finally to check accuracy of the parameters through providing new triples to the trained network (Abedini et al., 2017). A cost function has been used to train this network, and the parameters should be set in a way that this cost function is minimized. There are many approaches such as optimization-based classifier (Tavoli and Keyvanpour 2017; Zhao et al. 2017, Pand Keyvanpour et al. 2012) but for this purpose, an "Unconstrained Nonlinear Optimization Method" had been used in NTN model, which needed many replications. So, we suggest in this paper to combine this old method with "meta-heuristic optimization algorithms" in order to minimize the replications and increase the implementation speed; for this purpose, we have used three meta-heuristic methods of simulated annealing (SA), genetics algorithm (GA), and particle swarm optimization (PSO) in this paper. So, the contributions of this paper are as follows:

- Minimizing the replications of training NTN;
- Increasing the NTN training speed;
- Training NTN using three new algorithms;
- Combining classic and meta-heuristic optimization methods to train NTN.

At the beginning of the paper, we introduce NTN, and the suggested approach to implement meta-heuristic algorithms on the network is presented in the next section. Then, the results of implementing the algorithms are evaluated, and finally we will conclude on the issue and suggest some further researches for future.

NTN

NTN was introduced by Socher for knowledge base completion using available facts in the knowledge base that is state-of-the-art. In this section, this network model is presented that reasons over knowledge bases by learning vector representations over them. Each relation triple is described by a neural network and

pairs of knowledge base entities which are given as input to that relation's model. If the entities are in that relationship, then the model returns a high score otherwise a low one. Due to this, any fact can be scored with some certainty mentioned implicitly or explicitly in the database (Socher et al. 2013).

The aim of this network is to learn models which have the ability to realize the additional facts that hold purely due to the existing relations in the same knowledgebase. The goal of the model is to state whether two entities ($e1$; $e2$) are in certain relationship R. For instance, whether the relationship $<e1, R, e2\geq$ <Einstein, born-in, Germany> is true and with what score of certainty. It is supposed *that* $e1, e2 \in R^d$ be the vector representations (or features) of the two entities. For now, it can be assumed that each value of this vector is randomly initialized to a small uniformly random number.

The NTN replaces a standard linear neural network layer with a bilinear layer that directly relates the two entity vectors across multiple dimensions. By this model, a score of how probable it is that two entities are in certain relationship can be computed. Let $e1, e2 \in Rd$ be the vector representation of two entities then, the NTN-based function that predicts the relationship of two entities can be described as shown in the following equation.

$$g<e1, R, e2\geq U^T f\left(e1^T W_R^{[1:k]} e2 + V_R [e1\ e2] + b_R\right) \quad (1)$$

where $f = \tanh$ is a standard nonlinearity applied element-wise. $W_R^{[1:k]} \in R^{d\times d\times k}$ is a tensor that is the generalization of vector and matrix to more than 2 indices (Liu 2017; Liu, Li, and Vong 2017), and the bilinear tensor product $e1^T W_R^{[1:k]} e2$ results in a vector $h \in R^k$, where each entry is computed by one slice $i = 1\ldots k$ of the tensor: $h_i = e1^T W_R^{[1:k]} e2$. The other parameters for relation R are the standard form of a neural network: $V_R \in R^{k\times 2d}$, $U_R \in R^d$ and $b_R \in R^d$.

Equation (2) shows a visualization of this model for two slices. The main advantage is that it can relate the two inputs multiplicatively instead of only implicitly through the nonlinearity as with standard neural networks where the entity vectors are simply concatenated (Abedini, Menhaj, and Keyvanpour 2017).

All models are trained with contrastive max-margin objective functions. The main idea is that each triplet in the training set $T^{(i)} = <e1^{(i)}, R^{(i)}, e2^{(i)}>$ should receive a higher score than a triplet in which one of the entities is replaced with a random entity. There are N_R many relations, indexed by $R^{(i)}$ for each triplet. Each relation has its associated neural tensor net parameters. We call the triplet with a random entity corrupted and denote the corrupted triplet as $T^{(i)} = <e1^{(i)}, R^{(i)}, e_c>$ where we sampled entity e_c randomly from the set of all entities that can appear at that position in that relation. Let the set of all relationships' NTN parameters be $\Omega = u, W, V, b, E$. We minimize Equation (3) objective function (Socher et al. 2013).

where N is the number of training triplets and we score the correct relation triplet higher than its corrupted one up to a margin of 1. For each correct

triplet, we sample C random corrupted triplets. Standard L_2 regularization of all the parameters, weighted by the hyper parameter λ, has been used.

Taking derivatives from this objective function is not possible. For this reason, an optimization method has been used for its minimization. This method is minFunc (Schmidt 2005). Then the model is trained by taking derivatives with respect to the five groups of parameters. An abstract view of NTN training algorithm can be shown in Algorithm 1.

Algorithm 1. An abstraction view of neural tensor network training algorithm

Algorithm 1. NTN train

Input: F: Facts with structure <*e*1,*r*,*e*2>, F_c: Corrupted facts with structure <*e*1,*r*,*e*2>, theta: Array of NTN Parameters
Output: Array of trained parameters, cost
1: **for** iteration = 1: MaxItration
2: theta = minFunc(*F*, F_c, theta, *J*)
3: cost = costFunc(theta)
4: **end for**
5: **Return** theta, cost

Input of this algorithm is a set of true facts from knowledge base called "*F*," a set of false facts as corrupted facts called "F_c," an array of different entries of all network parameters called "theta." Here for easiness of training, total entries of all parameters have been placed in single file and have been considered as a unique array (theta). But after training step, all of the parameters will be separated. The output is final values of parameters after training and cost of the training.

The function of *minFunc* is a Matlab function for unconstrained optimization of differentiable real-valued multivariate functions using line-search methods. It uses an interface very similar to the Matlab Optimization Toolbox function *fminunc* and can be called as a replacement for this function. On many problems, *minFunc* requires fewer function evaluations to converge than *fminunc*. Further, it can optimize problems with a much larger number of variables (*fminunc* is restricted to several thousand variables) and uses a line search that is robust to several common function pathologies (Schmidt 2005).

The default parameters of *minFunc* call a quasi-Newton strategy, where limited-memory Broyden-Fletcher-Goldfarb-Shanno updates with Shanno–Phua scaling are used in computing the step direction, and a bracketing line-search for a point satisfying the strong Wolfe conditions is used to compute the step direction. In the line search, (safeguarded) cubic interpolation is used to generate trial values, and the method switches to an Armijo back-tracking line search on iterations where the objective function enters a region where the parameters do not produce a real valued output (Schmidt 2005).

As respects the algorithm 1 needed many iterations in continue it is suggested to perform this method with "meta-heuristic optimization algorithms" in order to minimize the iterations and increase the implementation speed.

Suggested Algorithm

Since "minFunc" function in Algorithm 1 should be replicated many times to get the minimum cost, we suggested in this paper to use meta-heuristic algorithms to minimize the replications. So, three meta-heuristic methods of SA, GA and PSO have been combined with the above mentioned algorithm, which we will discuss about in the next sections. The reason why we have used these three methods is to evaluate different type of meta-heuristic methods to specify the best based on the results.

Training NTN Using SA

Algorithm 2 has been suggested for training NTN using SA algorithm (Rutenbar 1989). SA algorithm is inspired by physical solidification of metals, in which the metal is first exposed to heat for a while and then chills down gradually to get a strong and solid crystal structure. This algorithm is included in single-answer methods and does the query more locally.

Algorithm 2. Training NTN using SA algorithm

Algorithm 2. SA NTN train

Input: F: Facts with structure <$e1$,r,$e2$>, F_c: Corrupted Facts with structure <$e1$,r,$e2$>, theta: Array of NTN parameters, cooling schedule
Output: Array of trained parameters, cost
1: theta = minFunc(F, F_c, theta, J)
2: cost = costFunc(theta)
3: $T = T_{max}$
4: **Repeat**
5: **Repeat**
6: theta′ = minFunc(F, F_c, theta, J)
7: cost′ = costFunc(theta)
8: ΔE = cost′-cost
9: **if** $\Delta E \leq 0$ **then**
10: theta = theta′
11: cost = cost′
12: **else**
13: accept cost′ and theta′ with a probability $\exp(-\Delta E/T)$
14: **end if**
15: **Until** Equilibrium condition
16: T = g(T)
17: **Until** Stopping criteria satisfied
18: **Return** theta, cost

Input of this algorithm are a set of true facts from knowledge base called "F," a set of false facts called "F_c," an element of different entries of all network parameters called "theta," and solidification temperature. Output of this algorithm is same as Algorithm 1. In lines 1 and 2, values of network parameters are set on F and F_c and its cost is saved in "cost" through running the function "minFunc" for one time. The sufficient temperature is also defined in line 3. Other algorithms are replicated enough times to reduce the temperature to the adequate level. There is another loop to provide balance circumstances. In each run of this loop, through

running first the function "minFunc," the value of network parameters' element is updated and its cost is calculated. Then, the amount of energy change is obtained according to the cost change. If the new cost is lower than the previously calculated one, it is accepted, and if not, the changed values are set with a higher cost based on idealistic measure with the probability exp $(-\Delta E/T)$.

The difference between this method and the previous one is that all answers are not accepted. In Algorithm 1, all the answers were accepted, but in this new one, just those answers with lower costs than the previous answer are accepted, though the worse answers are also accepted with a small probability in order to escape the local optimum. In fact, acceptance of the best answers will speed up algorithm convergence to the optimum point. However, this algorithm checks just one answer in each phase. So, we will suggest some methods in the following sections to cover a set of answers in each phase.

Training NTN Using GA

Algorithm 3 has been introduced for training NTN using GA. GA (Li et al. 2017) mechanism has been used in this algorithm. GA is the best known and also the oldest evolutionary algorithm based on Darwin`s theory. According to this theory, those creatures with more abilities to access to food resources and to do reproduction will survive and others will die. This algorithm is based on population and checks several answers in each phase.

Algorithm 3. Training NTN using GA algorithm

Algorithm 3. GA NTN train

Input: F: Facts with structure <$e1$,r,$e2$>, F_c: Corrupted Facts with structure <$e1$,r,$e2$>, theta: Array of NTN parameters
Output: Array of trained parameters, cost
1: theta(0) = minFunc(F, F_c, theta, J)
2: **For** i = 1: #chromosome **Do**
3: theta(i) = minFunc(F, F_c, theta(i − 1))
4: cost(i) = costFunc(theta(i))
5: **End for**
6: **While** (stopping criterion not met)
7: Select best chromosomes bestTheta1 and bestTheta2 as parents
8: theta1 = minFunc(F, F_c, bestTheta1, J)
9: theta2 = minFunc(F, F_c, bestTheta2, J)
10: Do Crossover on theta1 and theta2
11: Do Mutation
12: Select Children
13: **For** i = 1: #Children **Do**
14: theta(i) = children(i)
15: theta(i) = costFunc(F, F_c, theta(i))
16: **End for**
17: **Set** iteration number++
18: **End while**
19: **For** i = 1: #chromosome **Do**
20: cost(i) = costFunc(theta(i))
21: **End for**
22: **Select** best cost, theta
23: **Return** theta, cost

Input and output of Algorithm 3 is same as Algorithm 1. In lines 1–5, the primary population is put in chromosomes theta (*i*). Next phases are done in the main loop of the algorithm then. In this loop, two best chromosomes are first selected as the parents. Then, the process of updating these two chromosomes is done, and the intersecting procedure is implemented on these changed chromosomes in the next phase. The next phase is to select the best children based on the related policy, and the cost of each child is calculated then. Again, in the next phase, the best parents are selected out of these children. These phases are repeated enough times to get the appropriate result.

In this algorithm, a set of answers are checked in each phase along with the selection of the appropriate results. However, jumping action is also done in line 11 to escape the local optimum. In fact, combining the appropriate answers speeds up algorithm convergence to the optimum point. In order to show the power of this algorithm, we use another population-based method in the next section, called "PSO," and all the results will be compared in the end.

Training NTN Using PSO

PSO method is used to train NTN in Algorithm 3. PSO (Li et al. 2017) mechanism has been used in this algorithm. This algorithm is in fact an organized set of creatures cooperating to find foods, which is inspired by the behaviors of birds and fishes which work based on collective conscience. Each parts and particles of the population could be considered as a candidate answer for the question, having a specific location and speed. The next location of the particle is defined based on the best observed location by itself (Pbest) and the best observed location by the whole population (Gbest). It is also assumed that each particle has a unique moving direction, shown by velocity vector "V."

Algorithm 4. Training NTN using PSO algorithm

Algorithm 4. PSO NTN train

Input: F: Facts with structure <*e*1,*r*,*e*2>, F_c: Corrupted facts with structure <*e*1,*r*,*e*2>, theta: Array of NTN parameters
Output: Array of trained parameters, cost
1: theta(0) = minFunc(*F*, F_c, theta, *J*)
2: **For** *i* = 1: #Particle Do
3: theta(*i*) = minFunc(*F*, F_c, theta(*i* − 1), *J*)
4: cost(*i*) = costFunc(theta(*i*))
5: Pbest(*i*).theta = theta(*i*)
6: Pbest(*i*).cost = cost(*i*)
7: **End for**
8: Gbest = Best Pbest
9: **For** it = 1: MaxIteration **Do**
10: **For** *i* = 1: #Particle **Do**
11: $V(i) = V(i) + \rho_1 c_1(\text{Pbest}(i).\text{theta-theta}(i)) + \rho_2 c_2(\text{Gbest.theta-theta}(i))$
12: theta(*i*) = theta(*i*) + *V*(*i*)

```
13:      theta(i) = minFunc(F, F_c, theta(i), J)
14:      cost(i) = costFunc(theta(i))
15:      if cost(i)<Pbest(i).cost then
16:         Pbest(i).theta = theta(i)
17:         Pbest(i).cost = cost(i)
18:      End if
19:      if Pbest(i).cost<Gbest.cost then
20:         Gbest = Pbest(i)
21:      End if
22:   End for
23: End for
24: Return Gbest.theta, Gbest.cost
```

Input and output of Algorithm 4 are the same as one discussed before. In lines 1–7, the primary population is set in particles "theta (*i*)," and the best location of each particle and the best location of all particles are set in "Pbest" and "Gbest," respectively. Speed, location, and the best found answer are updated in each replication of the main loop, and the best answer is presented finally. Results of implementing these three algorithms are evaluated in the next section to show their function and performance.

Experimental Results

In this section, we will evaluate the abovementioned algorithms. For this purpose, a set of standard data were used, which were evaluated in the previous works (Socher et al. 2013). Statistical specifications of this data set are shown in Table 1. "WordNet" has been used in this data set as the RDF-based knowledge base. This set has 112,581 standard "WordNet" triples as samples, in which there are 38,696 unique entities in 11 different relations.

To train NTN using Algorithm 1, inputs are obtained from training triples of data set and given to data network. Then, network parameters are set through running the training rule of NTN. For setting the parameters, optimization phase of the purpose function should be repeated enough times to get the minimum cost. These phases have been repeated 50 times for this network, and the results are shown in Figure 1.

As shown in the figure, a lot of replications are needed to reach to the optimum algorithm (28 times). So, we used three meta-heuristic methods in this paper to minimize replication times and speed up the algorithm. Result of the implementation of these three methods in first 10 replications, compared with the previous method, is shown in Figure 2. As shown in this figure, replication times have been minimized using these meta-heuristic

Table 1. Properties of dataset (Socher et al. 2013).

Dataset	#R	#Ent.	#Train	#Dev	#Test
WordNet	11	38,696	112,581	2609	10,544

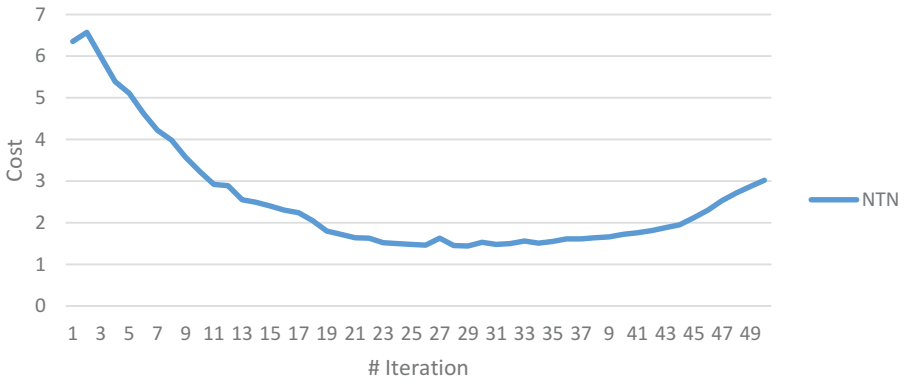
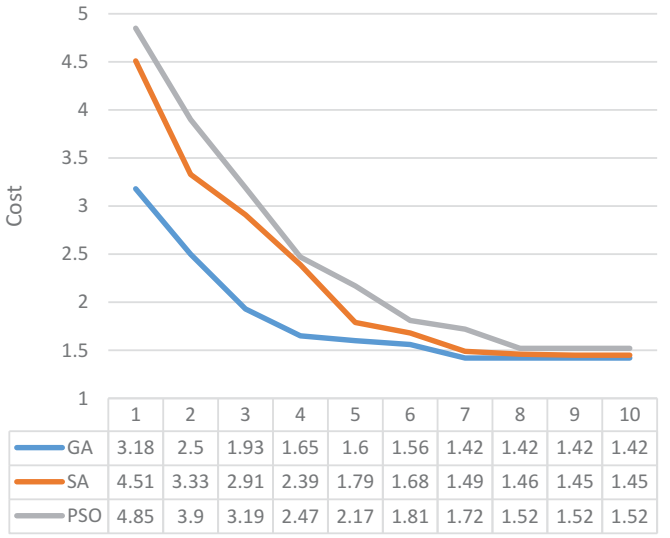


Figure 1. Diagram of training cost of NTN with Algorithm 1.



	1	2	3	4	5	6	7	8	9	10
GA	3.18	2.5	1.93	1.65	1.6	1.56	1.42	1.42	1.42	1.42
SA	4.51	3.33	2.91	2.39	1.79	1.68	1.49	1.46	1.45	1.45
PSO	4.85	3.9	3.19	2.47	2.17	1.81	1.72	1.52	1.52	1.52

Figure 2. Comparison diagram of three suggested methods.

methods. It is also shown that GA reaches to the optimum point faster than others. This issue is more obvious in Figures 3 and 4.

These figures show GA with 8 replication times and minimum cost of 1.42 has the best performance. However, the important issue here is that the time of one replication of each algorithm is different. So, it is not easy to choose the best method based on this parameter. To solve the problem, we suggest calculating the whole time of each implementation. The whole time of each algorithm is calculated through multiplying replication times by the time of each replication. This procedure is shown in Table 2.

Since GA and SA are of cumulative methods, several answers are checked in each phase, and so the time of each replication is different. SA method needs to check several answers in each phase to reach to balance and so it takes more

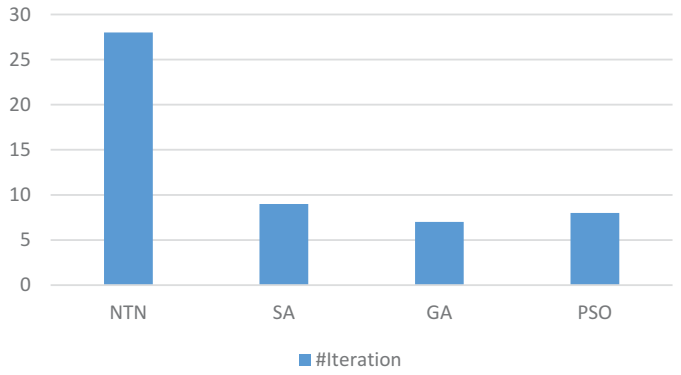


Figure 3. Diagram of number of iteration.

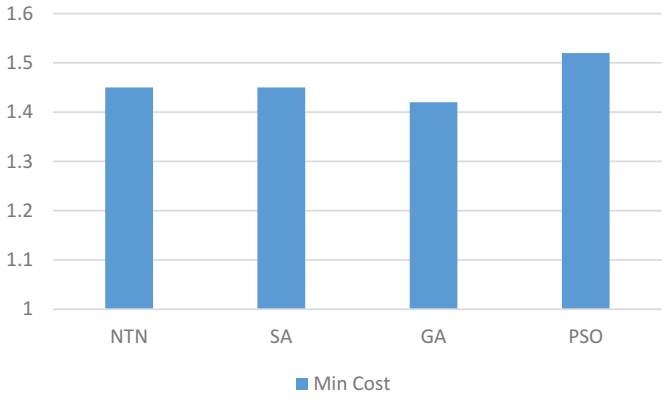


Figure 4. Diagram of optimum point.

Table 2. Execution times of algorithms.

	NTN	SA	GA	PSO
# Iterations	28	9	**7**	8
Time of each iteration (min)	**5**	15	11	15
Total time (min)	140	135	**77**	120

time. As shown in the figure, meta-heuristic methods improved the whole time, along with minimizing replication times, which lead to higher speed of algorithm, and GA has the best performance. After training phase, accuracy level of all methods showed the same results, using test data of Table 1.

Conclusion

In this paper, we showed that combination of meta-heuristic and classic optimization methods could improve the speed of NTN training to learn correct facts' pattern of a RDF-based knowledge base. For this purpose, three

new algorithms were introduced. Results showed that all these three methods improve the performance, but GA had the best function. The reason of this supremacy is that GA is based on population and checks a set of answers in each phase, instead of a single answer. This procedure will minimize the replication times. On the other hand, there is no need to run several answers in each phase, and new answers are obtained through jumping and intersection operators, which have lower levels of complexity. So, replication time of each phase in GA is smaller than SA and PSO. In fact, the results of this paper showed that GA is the best algorithm for the discussed subject.

ORCID

Mohammad Bagher Menhaj http://orcid.org/0000-0002-9470-5532

References

Abedini, F., M. B. Menhaj, and M. R. Keyvanpour. 2017. An MLP-based representation of neural tensor networks for the RDF data models. *Journal of Neural Computing and Applications*, Springer, 1–10. doi: 10.1007/s00521-017-3053-1.

Abedini, F., M. B. Menhaj, and M. R. Keyvanpour. 2017. Neuron mathematical model representation of neural tensor network for rdf knowledge base completion. *Journal of Computer & Robotics* 10 (1):1–10.

Bizer, C., J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. 2009. DBpedia-A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web* 7 (3):154–65. doi:10.1016/j.websem.2009.07.002.

Bordes, A., X. Glorot, J. Weston, and Y. Bengio. 2012. Joint learning of words and meaning representations for open-text semantic parsing. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, p. 127-135..

Bordes, A., J. Weston, R. Collobert, and Y. Bengio. 2011. Learning structured embeddings of knowledge bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artifitial Intelligence*. AAAI Press: San Francisco, California. p. 301-306.

Imani, M. B., M. R. Keyvanpour, and R. Azmi. 2013. A NOVEL EMBEDDED FEATURE SELECTION METHOD: A COMPARATIVE STUDY IN THE APPLICATION OF TEXT CATEGORIZATION. *Applied Artificial Intelligence* 27 (5):408–27. doi:10.1080/08839514.2013.774211.

Jenatton, R., N. Le Roux, A. Bordes, and G. Obozinski. 2012. A latent factor model for highly multi-relational data. In Proceedings *of the 25th International conference on Neural Information Processing System - Volume 2*. Curran Associates Inc.: Lake Tahoe, Nevada. p. 3167-3175.

Keyvanpour, M. R., M. R. Ebrahimi, and M. Javideh. 2012. DESIGNING EFFICIENT ANN CLASSIFIERS FOR MATCHING BURGLARIES FROM DWELLING HOUSES. *Applied Artificial Intelligence* 26 (8):787–807. doi:10.1080/08839514.2012.718227.

Kim, H. 2017. Building a K-Pop knowledge graph using an entertainment ontology. *Knowledge Management Research & Practice* 15 (2):305–15. doi:10.1057/s41275-017-0056-8.

Li, T., G. Shao, W. Zuo, and S. Huang 2017, February. Genetic Algorithm for Building Optimization: State-of-the-Art Survey. In Proceedings of the 9th International Conference on Machine Learning and Computing (pp. 205–10). ACM: Singapore, Singapore.

Liu, D., W. Li, and S.-W. Vong. 2018. Tensor complementarity problems: The GUS-property and an algorithm. In *Linear and Multilinear Algebra*, Taylor and Francis, 66(9): p. 1726-1749.

Liu, X. 2018. The eigenvalues and eigenvectors of nonsingular tensors, similar tensors and tensor products. In *Linear and Multilinear Algebra*, Taylor and Francis, 66(9): p. 1717-1725.

Miller, G. A. 1995. WordNet: A Lexical Database for English. *Communications of the ACM* 38:39–41. doi:10.1145/219717.219748.

Rutenbar, R. A. 1989. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices Magazine* 5 (1):19–26. doi:10.1109/101.17235.

Schmidt, M., "minfuc http://people.cs.ubc.ca/schmidtm/software/minfunc.html," 2005.

Socher, R, D. Chen, and C. D. Manning. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the 26thional Conference on Neural Information Processing Systems - Volume 1*. Curran AssociateInc.: Lake Tahoe, Nevada. p. 926-934.

Suchanek, F. M., G. Kasneci, and G. Weikum. 2007. Yago: A core of semantic knowledge. Proceedings of the 16th international conference on World Wide Web. ACM: Banff, Alberta, Canada. doi:. 10.1094/PDIS-91-4-0467B

Sutskever, I., R. Salakhutdinov, and J. B. Tenenbaum. 2009. Modelling relational data using Bayesian clustered tensor factorization. In *Proceedings of the 22nd International Conference on NInformation Processing System*. Curran Associates Inc. : Vancouver, British Columbia, Canada. p. 1821-1828.

Tavoli, R., and M. Keyvanpour. 2017. A Novel Word-Spotting Method for Handwritten Documents Using an Optimization-Based Classifier. *Applied Artificial Intelligence* 31 (4):346–75. doi:10.1080/08839514.2017.1346964.

Turian, J., L. Ratinov, and Y. Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. *Proceedings of ACL* : Uppsala, Sweden. p. 384–394.

Weikum, G., J. Hoffart, and F. Suchanek. 2016. Ten years of knowledge harvesting: Lessons and challenges. *Data Engineering*. 39(3): 41-50.

Zhao, Y., Y.-J. Wu, E. Levina, and J. Zhu. 2017. Link prediction for partially observed networks. *Journal of Computational and Graphical Statistics* 26 (3):725–33. doi:10.1080/10618600.2017.1286243.