Transfer learning for boosted relational dependency networks through genetic algorithm*

Leticia Freire de Figueiredo[1][0000−0001−7613−7423], Aline Paes[2][0000−0002−9089−7303], and Gerson Zaverucha[1][0000−0002−3641−6839]

[1] Department of Systems Engineering and Computer Science, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brazil
{lfreire,gerson}@cos.ufrj.br
[2] Institute of Computing, Universidade Federal Fluminense, Niteroi, RJ, Brazil
alinepaes@ic.uff.br

Abstract. Machine learning aims at generalizing from observations to induce models that aid decisions when new observations arrive. However, traditional machine learning methods fail at finding patterns from several objects and their relationships. Statistical relational learning goes a step further to discover patterns from relational domains and deal with data under uncertainty. Most machine learning methods, SRL included, assume the training and test data come from the same distribution. Nonetheless, in several scenarios, this assumption does not hold. Transfer learning aims at acting on scenarios like that, leveraging learned knowledge from a source task to improve the performance in a target task when data is scarce. A costly challenge associated with transfer learning in relational domains is mapping from the source and target background knowledge language. This paper proposes GROOT, a framework that applies genetic algorithm-based solutions to discover the best mapping between the source and target tasks and adapt the transferred model. GROOT relies on a set of relational dependency trees built from the source data as a starting point to build the models for the target data. Over generations, individuals carry a possible mapping. They are submitted to genetic operators that recombine subtrees and revise the initial structure tree, enabling a prune or expansion of the branches. Experimental results conducted on Cora, IMDB, UW-CSE, and NELL datasets show that GROOT reaches results better than the baselines in most cases.

Keywords: transfer learning · statistical relacional learning · genetic algorithm.

1 Introduction

Machine learning algorithms aim at generalizing observations from data to find patterns that aid decision making about future observations. However, when the

* Partially funded by CNPq and FAPERJ

patterns involve multiple objects and their relationships, traditional machine learning methods could not be enough to represent the relational interactions among data [7]. In this way, it is necessary to rely on methods that take advantage of the information that relational data could give through such interactions to generate models that show the correlations between the multiple objects [11]. *Relational learning* is a machine learning branch that englobes methods capable of dealing with relational data. Statistical relational learning combines relational learning and statistical learning with data under uncertainty [11].

Still, most relational and non-relational machine learning methods assume that the training and testing data come from the same feature space and distribution. When this assumption does not hold, it is necessary to collect more data and learn a new model from scratch, which can be expensive and, sometimes, even impossible. To tackle those situations, transfer learning between related tasks shows as a viable solution. [23] Transfer learning is a technique that leverages knowledge previously learned from a source task to boost the induction of a model to a target task, mainly when data is scarce [22]. Transfer learning has drawn the attention of relational learning for some time to improve and accelerate learning in the target domain [13], [15].

However, relational learning poses an additional challenge when transferring knowledge from a source domain to a target domain since their vocabulary will probably differ. Thus, it is necessary to establish a mapping from both vocabularies so that the learned hypothesis can be used to represent the target data. In [2], giving the hypothesis in the form of a relational regression tree, the framework called TreeBoostler first finds a mapping, defining the replacement between the source and target predicates, through a constrained search space built upon their predicates. In relational setting, the predicate corresponds to a relation between arguments that correspond to the attributes of the relation. For example, if the dataset indicates that Mary is daughter of John, we can define a predicate called daughter that has the arguments Mary and John [1]. Also, the attributes are typed. In the daughter relation, the arguments can be typed as *person.* [3] For each possibility of mapping predicates and their types, the procedure verifies if it can improve an evaluation function and revise the structure, making pruning or expanding the leaves to new branches. However, finding the best mapping could be challenging when background knowledge, from the source, target, or both domains, has many predicates and types. A possible way to solve this problem is to use metaheuristics to find the best solution instead of searching almost entirely the mapping space, constrained only to a few restrictions.

This work proposes GROOT[4], a framework written in Python that receives an initial set of trees from the source domain and applies genetic algorithm-based solutions to transfer a learned hypothesis from a source domain to a target

[3] The types are defined into the language bias, which here follows the Aleph and Progol definitions.

[4] https://github.com/MeLL-UFF/groot: the online repository contains the GROOT code, an experiment example, besides the datasets used for the experiments contained in this paper and the results.

domain. The solutions are measured by RDN-Boost, creating a set of trees that approximate the joint probability distribution over the variables as a product of conditional distributions [19]. One solution is considered the best if it has the best area under the Precision-Recall curve (AUC PR) metric. Each individual in the genetic algorithm carries a possible mapping. The crossover between them will combine random subtrees in two different individuals and the mutation could expand or prune a branch. We experiment GROOT with four domains and find out GROOT gives a better result in some metrics but needs a long time to return the answers.

To sum up, our contributions are as follows: develop a method based on genetic algorithm to map predicates between a source task to a target task and use genetic operators to help the revision of the structure trees. The experiment are made with relational datasets to evaluate how GROOT improves the results compared with the baseline. The remainder of the paper is organized as follows: Section 2 introduces the necessary background to understand fundamental concepts addressed in this paper. In this section, we clarify about RDN-Boost, transfer learning, and genetic algorithm. Next, Section 3 brings related work showing transfer learning between relational tasks. Section 4 explains the GROOT framework, including their algorithms and inner components. Section 5 presents the results from GROOT, compared with the RDN-Boost and TreeBoostler results. The final section ends with the conclusion about the work and next steps.

2 Background Knowledge

In this section, we briefly introduce important concepts to the reader to understand the foundations of our contributions. First, we describe the RDN-Boost method. Next, we explain basic notions on transfer learning and finalize the section with genetic search and optimization.

2.1 RDN-Boost

Dependency networks (DNs) are graphical models for probabilistic interactions. A DN is a directed graph $G = (V, E)$, usually cyclic, where each node $v_i \in V$ corresponds to a random variable $X_i \in X$, where X is a set of variables enconding probabilistic relationships. Each node is also associated with a positive joint distribution approximated with a set P of conditional probability distributions that are calculated as:

$$p_i(X_i|\boldsymbol{parents(X_i)}) = p(X_i|X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n) = p(X_i|X \setminus X_i) \quad (1)$$

where $parents(X_i)$ denotes the set of nodes with incoming edges to X_i. Unlike Bayesian networks, the DN graph can be cyclic [10].

Relational dependency networks (RDNs) are an extension of DNs to the relational setting. A RDN models the dependencies in a directed graph G_M, like in DNs, but each node corresponds to an object in the data and an edge corresponds to a relationship among the objects. To each node in the graph, it is associated a conditional probability, given by equation 1 [19, 21].

Finally, *RDN-Boost* [19] considers the conditional probability distribution as a regression trees set to each predicate and, at each iteration, the idea is to maximize the likelihood of the distributions with respect to a potential function. The potential function is defined using an initial potential ψ_0 and, iteratively, adds the gradients Δ, which is the gradient of the likelihood function. In this way, the potential function is given by

$$\psi_m = \psi_0 + \Delta_1 + ... + \Delta_m \tag{2}$$

where Δ_i is the gradient at iteration i. When computing the equation 1, each branch is applied to determine the branches that are satisfied and their regression values are sumed up to the potential function [19].

2.2 Transfer Learning

Traditionally, machine learning methods assume the train and test data come from the same distribution and feature space. However, in some scenarios, training data is scarce due to difficulties and the cost to collect them. The lack of training data may degrade the machine learning method performance [26].

A possible way to tackle the lack of training data is to transfer a learned model from a source task to the target task, allowing the use of data from different domains, distributions, and tasks. The traditional learning process of machine learning methods is to use, from scratch, the data from a target domain to train and test a task. *Transfer learning* method aims to extract knowledge from a source task and apply it to a target task, alleviating the need to train the model from scratch in the target task [23].

Transfer learning is formalized as follows. Let $\mathcal{X}$ be the feature space, $P(X)$ the marginal probability distribution over $\mathcal{X}$, and $\mathcal{D} = \{\mathcal{X}, P(X)\}$ a domain. The source domain $\mathcal{D}_s$ is defined as $\mathcal{D}_s = \{\mathcal{X}_s, P(X_s)\}$ and the target domain, $\mathcal{D}_t = \{\mathcal{X}_t, P(X_t)\}$. A task $\mathcal{T}$ is defined by a label space $\mathcal{Y}$ and an objective function $f(\cdot)$ that can be learned from the training data (x_i, y_i), where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. A source task is defined as $\mathcal{T}_s = (\mathcal{Y}_s, f_s(\cdot))$ and a target domain, $\mathcal{T}_t = (\mathcal{Y}_t, f_t(\cdot))$. With a source domain $\mathcal{D}_s$ and a source task $\mathcal{T}_s$, a target domain $\mathcal{D}_t$ and a target task $\mathcal{T}_t$, where $\mathcal{D}_s \neq \mathcal{D}_t$ or $\mathcal{T}_s \neq \mathcal{T}_t$, the goal is to learn the target function $f(\cdot)$ leveraging the knowledge from $\mathcal{D}_s$ and $\mathcal{T}_s$ [23, 26, 27].

2.3 Genetic Algorithm

Genetic algorithm (GA) is a population-based metaheuristic based on natural selection and the principles of genetic. The method walks in a space of candidate hypotheses to find the best candidate, defined by the objective function, called fitness [8], [17]. GA relies on a population composed of individuals. Each individual contains a feasible solution to the problem, encoded in their chromosomes with genes inside. The evaluation of the population is made by the fitness function that informs if an individual is a good or a bad candidate. Along the generations, the GA evolves the solutions using genetic operators:

1. **Selection**: selects the best individuals according to their fitness value and reallocates it to the population. One way to make the selection is the tournament selection, which selects randomly k individuals from the population and, with a probability p, selects the best.
2. **Recombination**: two parent individuals form new offsprings, recombining their genes at different points.
3. **Mutation**: an individual has one of their genes randomly modified, generating a new solution, feasible or not.

At the end of all these steps, the old population is replaced by the new population generated by the genetic operators. Also, it is possible to apply an elitist technique, where the best individual of the last generation is introduced on the current population without modifications [4, 17].

3 Related Work

Previous works have already proposed novel methods for leveraging transfer learning to relational domains. In [2], the authors propose a framework called TreeBoostler to transfer a RDN-Boost model learned from a source task to a target task, searching for the best mapping between source predicates and types and target predicates and types. After finding the mapping between the source and target vocabulary, the performance of the transferred RDN-Boost is evaluated over the target data. In order to accommodate target predicates that were not mapped and adjust the learned trees to the target data, TreeBoostler employ a final revision step. Such a component selects the revision points as the places in the trees that have low probabilities for the true class. Next, from the revision points, the revision operators either expand leaves to generate new branches or prune nodes. TreeBoostler has a limitation when searching the best mapping: the source predicate can only be substituted by one and only one target predicate and the same is valid for the types. Furthermore, the search is almost complete in the sense that are only some restrictions to avoid experimenting with the whole search space of possible mappings. Depending on the size of the vocabulaty, this can be a quite expensive process.Nevertheless, TreeBoostler was successfully compared to TAMAR [15] and TODTLER [9].

Our framework is similar to the introduced solution but makes use of a genetic algorithm instead of an exhaustive search. Also, GROOT allows making a map between many source predicates to many target predicates, just adding an identifier to the source predicate to turn it into a unique on. The source types can be mapped to many target types, however, the target types can only be mapped to one source type.

Metaheuristics have already been used in Inductive Logic Programming. In QG/GA [18], the authors investigate a method called Quick Generalization (QG), which implements a random-restart stochastic bottom-up search to build a consistent clause without the needing to explore the graph in detail. The genetic algorithm (GA) evolves and recombines clauses generated by the QG, that forms the population.

In [25], the authors present EDA-ILP and REDA-ILP, an ILP system based on Estimation of Distribution Algorithms (EDA), a genetic algorithm variation. Instead of using genetic operators, EDA builds a probabilistic model of solutions and sample the model to generate new solutions. Across generations, EDA-ILP evaluates the population using Bayesian networks and returns the fittest individual. In REDA-ILP, the method also applies Reduce algorithm in bottom clauses to reduce the search space. In [14], the authors propose GENSYNTH, a tool that synthesizes invented predicates, free from bias, to Datalog language problems. According to a fitness function, the tool uses steps inspired in the genetic algorithm to find interpretable programs without language bias quickly.

4 GROOT: Genetic algorithms to aid tRansfer Learning with bOOsTsrl

The proposed framework transfers relational dependency trees learned from a source domain to a target domain. The idea is to take advantage of the built structure trees using the source domain as a starting point to learn how to solve the target task. Instead of conducting a complete search into the space of solutions, GROOT employs a genetic algorithm to find the best mapping between the predicates that optimizes the area under the Precision-Recall curve (AUC PR) value. In addition, GROOT includes genetic operators to revise the source structure to better accommodate the target examples.

4.1 Population

Along the generations, the genetic algorithm evaluates individuals in the population, where which one carries a possible solution to the problem. In GROOT, each individual is composed of chromosomes, a feasible mapping corresponding to each node in the trees, and the fitness function value. Each chromosome is a structure corresponding to one tree and each tree has alleles, corresponding to each node. In our framework, each individual has 10 chromosomes. Figure 1 depicts how the individuals are encoded in GROOT.

GROOT allows mapping many source predicates to many target predicates. To distinguish which source predicate has been replaced, a unique identifier is appended to each one of them. This is made because if the source predicate is the same always, the transfer will be made with the first mapped target predicate. The map between the predicates is made at random but restricted to predicates with the same arity and language bias. The target predicates are selected according to the arity and language bias of the source predicate and

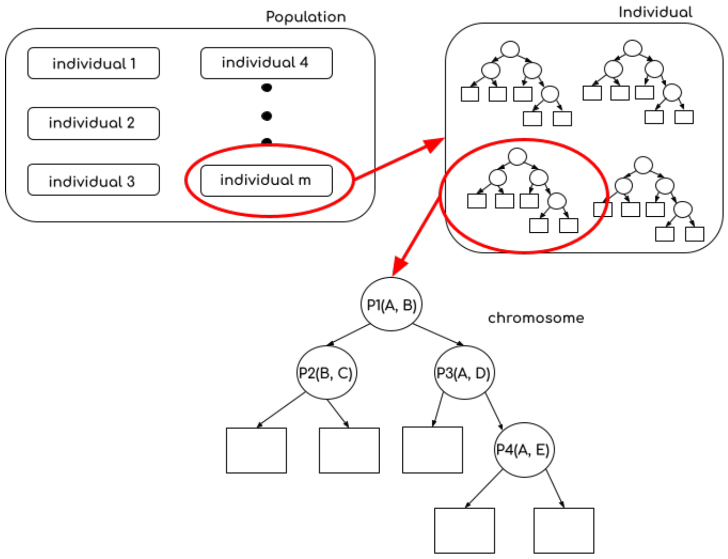

Fig. 1: Schema showing how the individuals are defined in GROOT. Each chromosome represents a tree and each allele, a predicate. The alleles carry the mapping between the source and target vocabulary.

are part of the feasible substitution set. The predicates are mapped sequentially, respecting the order they appear in the trees. If there are no feasible target predicates to replace the source predicate, the source predicate is mapped to null. For example, this case can occur when a source predicate has arity one but none of the predicates in the target background knowledge has this arity.

We assume that GROOT input includes a set of trees from the source dataset created with RDN-Boost using predicates from the source vocabulary. Each individual is a copy of this set, and each tree corresponds to a chromosome in the individual. For each predicate appearing in the nodes, GROOT verifies which predicate from the target vocabulary can be mapped for the source predicate. Using the IMDB dataset as an example, consider that a node has an atom originated from the *director(+person)* predicate template. We need to find a predicate in the target dataset with the same definition (arity equals one and person as the type of the argument). Supposing the transference IMDB $\rightarrow$ UW-CSE, we can map it either to *professor(+person)* or *student(+person)* as they have the same arity and same type. If there is no target predicate with the same arity and types, the source predicate is mapped to *null(null)*. If mapping between types is also possible, in this case, is *person* $\rightarrow$ *person*, we randomly choose the target predicate that can uniformly replace the source predicate. The main node in the trees already has a determined predicate, which is the predicate RDN-Boost is learning. For example, for the IMDB dataset, the main predicate is *workedun-*

der(+person, +person), which will be mapped to *advisedby(+person, +person)*, in the UW-CSE dataset.

4.2 Genetic Operators

The population can recombine and mutate a selected number of individuals. The crossover operator exchanges information by combining parts of two individuals. These two individuals are called parents and generate two new offsprings [17]. According to a crossover rate, two individuals are randomly chosen and, in each one, the following steps are done:

1. a chromosome (a tree) is selected uniformly
2. in the selected chromosome, an allele (a node) is elected to be the exchange point
3. the subtree starting at the selected alleles, including all the following nodes, are swapped between the individuals
4. the unselect chromosomes are passed to the new individuals as they were in the original individuals

After the crossover, two new individuals are created to replace the parents. The Figure 2 shows an example of a crossover operation. The red nodes indicate the alleles which will be exchanged between the individuals. On the right side of the figure, it is possible to see the final result.

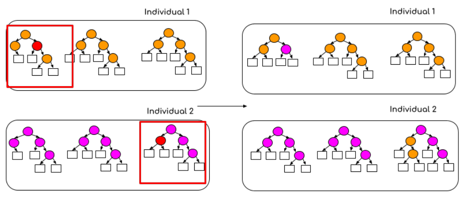

Fig. 2: Example of crossover between two individuals with 3 chromosomes. The red nodes are selected to be exchanged.

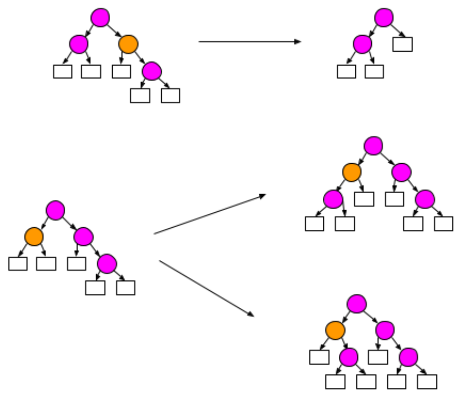

Fig. 3: A mutation example. Top: pruning starts from the orange node and erases all the nodes below it. Bottom: the expansion example shows the possibility to include a new node in one leaf.

The mutation operator selects an individual according to a mutation rate and applies small changes in their alleles. In GROOT, instead of changing the mapping, a revision structure is applied. When evaluating the individuals, a weighted variance of each node is returned as a result of the covered examples. An example is covered when a path in the hypothesis covers it. If a node has a variance higher than 2.5×10^{-3}, it is considered a revision point and could be pruned or expanded. This value is the same value used as default in TreeBoostler [2]. The decision to prune or expand is made uniformly. An individual can have many revision points but just one is chosen to be revised, at a generation. The expansion is made if a revision point has, at least, one leaf. The leaf is expanded with one predicate from the target domain. When pruning a node, all the nodes under the revision point are also erased and, at the place of the node, a leaf is set. In this case, it is not necessary to have leaves. The Figure 3 exemplifies how the mutation occurs in the individual.

4.3 Selection and Evaluation

Selection copies individuals with better fitness values to the population of individuals. A method called tournament selection is used in GROOT. The tournament selection chooses, at random, some individuals to enter into a tournament. The individual in the group with the best fitness value wins and is selected to be included in the population [4]. GROOT also applies elitism selection, where the fittest individual of the generation is guaranteed to be in the next generation. An individual is considered the fittest when it has the lowest negative AUC PR. If more individuals have the same fitness value, the one with the lowest negative conditional log-likelihood (CLL)[5] score is chosen.

After applying the genetic operators, the population is evaluated to set the fitness value for each individual. GROOT uses genetic algorithm to minimize the negative value of the AUC PR value. The individuals carry the mapping between the source and target predicates and the structure to make the transfer. The mapping and structure trees are given to RDN-Boost, which trains the model with the training dataset. The test dataset is used to evaluate how the model performs and gives as result the area under the ROC curve (AUC ROC), AUC PR, conditional log-likelihood values and others. The individual receives the AUC PR value as the fitness value.

5 Experimental Results

In this section we present the experiments devised to answer the following research questions:

- **Question 1:** Does GROOT perform better than learning from scratch?

[5] Conditional log-likelihood gives the probability log of a given example be true, given the other random variables.

- **Question 2:** Does GROOT perform better than another transfer learning framework?
- **Question 3:** Does GROOT reach good results in a viable time?

The first question focus on verifying the benefits of transfer learning compared to learning from scratch. The second question is relative to how our framework can perform comparing with another transfer learning framework when using the same data. The final question addresses how long GROOT takes to run the entire transfer process.

Next, we present the datasets used in the experiments and, finally, the methodology used to find the parameters of the genetic algorithm and the results.

5.1 Datasets

We used the following four datasets for the experiments.

- Cora dataset [3] contains citations to Computer Science research papers. The dataset has 1295 distinct citations to 122 papers. The goal is to predict *samevenue*, which shows the relation between two venues. The dataset has 10 predicates and five types, with the number of all ground literals equals to 152100.
- IMDB dataset [16] contains five mega examples, describing four movies, their directors and the first-billed actors who appear in them. A mega example contains a connected group of facts and each one is independent of the other [16]. The main predicate is *workedunder*, indicating if two persons worked together. The dataset has six predicates, three types and 71824 ground literals.
- NELL [5] is a system that extracts information from web texts, learning to read each day better than the day before. Our experiments ran with two datasets created by the system: Finances, which predict if a company belongs to an economic sector and Sports, predicting which sports a team plays. NELL Sports dataset has eight predicates, four types and 4323, while NELL Finances has 10 predicates, five types and 51578 ground literals.
- UW-CSE dataset [16] is a dataset with mega-examples based on five Computer Science areas and lists facts about people in the academic department and their relationships. The relation *advisedby* predicts if one person is advised by another person. The dataset has 14 predicates, nine types and 16900 ground literals.

Methodology We compared the results generated by GROOT with the performance of TreeBoostler, another framework to transfer learning between relational domains that also uses RDN-Boost to evaluate their results. We also compared with the results from models learned from scratch, produced by RDN-Boost and RDN-B-1. RDN-B-1 differs from RDN-Boost because the model learns just one tree, instead of a set of trees. GROOT starts from the trees created by

	Parameters
Mutation rate	[0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4]
Crossover rate	[0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95]
Number of individuals	[10, 30, 50]

Table 1: Hyperparameters used in the optimization functon.

RDN-Boost when learning from the source domain for each experiment. The datasets are, initially, divided in folds. For our experiments, we joined all the folds and split it into three new folds, training the model with one fold and evaluating with the remaining data, employing the same methodology used in literature [6, 12, 2]. Cross-validation is applied: for each of the three folds, we train five times to accommodate randomness with the current training set and test with the other folds, completing each experiment with 15 trainings. The final results are then averaged over the runs. Each training fold is divided into three sub-folds to find the hyperparameters for the genetic algorithm. On these three sub-folds, an internal cross-validation procedure is applied: one fold is selected for training and two folds for validation. Finding the best combination of parameters for the genetic algorithm is a hard task. Thus, this search uses `gp_minize` from Scikit-Optimize package [24] built with Scikit-Learn. The `gp_minize` function uses Bayesian optimization, approximating the desired function to optimize by a Gaussian process. We used 10 evaluations to get the parameters in each validation set. The RDN-Boost also has hyperparameters but we used the same as defined in TreeBoostler [2]. The best validation set gives the parameters to be used in the training fold. When training, we sample the amount of negative as been the ratio of two negatives for one positive; when testing, we used all the examples from the test dataset [20]. At the final generation, the best individual of the population is chosen to give the mapping to generate the results.

IMDB → UW-CSE

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.239 ± 0.000	0.796 ± 0.000	0.085 ± 0.000	2.595 ± 0.124 s
RDN-B	**-0.814 ± 0.003**	0.801 ± 0.005	0.094 ± 0.011	7.582 ± 0.457 s
TreeBoostler	-0.368 ± 0.004	0.905 ± 0.004	0.168 ± 0.014	11.152 ± 0.700 s
GROOT	-0.262 ± 0.033	**0.939 ± 0.010**	**0.336 ± 0.018**	18.3 ± 6.5 min

Table 2: Results for the experiment with IMDB and UW-CSE datasets, comparing with TreeBoostler, RDN-B-1 and RDN-Boost. The table shows the values for AUC ROC, AUC PR, CLL, and runtime. The first two rows show the results when learning the target dataset from scratch.

Results The results are presented in Tables 2, 3, 4, and 5, with the experiments realized between the IDMB, UW-CSE, Cora and NELL datasets, reporting the mean and standard deviation results. When GROOT finds the best results for the metric, the values are shown in bold. GROOT obtains an improvement for at least one metric in most of the experiments and achieves comparable results when confronting with methods that learn from scratch, answering positively the **Question 1** and **Question 2**.

IMDB → Cora

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.213 ± 0.004	0.534 ± 0.008	0.012 ± 0.000	5.9 ± 5.1 min
RDN-B	-0.500 ± 0.01	0.542 ± 0.006	0.013 ± 0.001	42.8 ± 7.6 min
TreeBoostler	-0.325 ± 0.008	0.729 ± 0.006	0.261 ± 0.022	194.0 ± 50.1 min
GROOT	-0.326 ± 0.006	0.582 ± 0.005	0.183 ± 0.010	41.0 ± 0.6 min

Table 3: Results for the experiment with IMDB and Cora datasets, comparing with TreeBoostler, RDN-B-1 and RDN-Boost. The table shows the values for AUC ROC, AUC PR, CLL, and runtime. The first two rows show the results when learning the target dataset from scratch.

Cora → IMDB

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.224 ± 0.000	0.843 ± 0.000	0.487 ± 0.000	2.249 ± 0.067 s
RDN-B	-0.697 ± 0.000	0.843 ± 0.000	0.487 ± 0.000	4.100 ± 0.137 s
TreeBoostler	-0.236 ± 0.000	0.958 ± 0.001	0.541 ± 0.055	9.564 ± 0.140 s
GROOT	-0.208 ± 0.015	**0.965 ± 0.012**	0.326 ± 0.176	86.4 ± 30.8 min

Table 4: Results for the experiment with IMDB and Cora datasets, comparing with TreeBoostler, RDN-B-1 and RDN-Boost. The table shows the values for AUC ROC, AUC PR, CLL, and runtime. The first two rows show the results when learning the target dataset from scratch.

In Table 3, GROOT did not provide a good result. This is because the source structure trees learned from the IMDB dataset have nodes with predicates containing arity equals to one. The predicates from the Cora dataset have only predicates with arity greater or equal to two. GROOT could not deal with this problem and just replaced the source predicate with a null predicate. The null predicate is a predicate that does not exist in any of the vocabularies and indicates an absence of mapping. Instead of the previous commented table, in Table 2, the framework reaches the best result in AUC PR and AUC ROC metrics. This occurs because GROOT can map the same source predicate to different target predicates, instead of TreeBoostler that attributes only one target predicate to one source predicate.

The statistical significance was measured by a paired t-test with $p <= 0.05$. The values are considered statistically significant in the experiment IMDB $\rightarrow$ Cora in all metrics, except by the pair GROOT and TreeBoostler. In the experiment IMDB $\rightarrow$ UW-CSE, the AUC PR value is statistically significant when considering the pair GROOT with both RDN-B-1 and RDN-Boost. Although most of the experiments needed a long time to get the results, the experiment IMDB $\rightarrow$ Cora reached the worst runtime in TreeBoostler results. This occurs because the mapping between the predicates is poor, requiring a revision that almost recreates all the trees. We wanted to show this case as en example of a bad transfer scenario. Taking into account the experiment IMDB $\rightarrow$ Cora, we attribute a partial negative answer to the **Question 3**. We conclude that, besides the number of predicates and types, the quantity of ground literals also impacts the time, mainly in the revision step. However in GROOT the revision is not made in all revision points, so this is not an issue, enabling the framework to have a lower runtime.

Nell Sports $\rightarrow$ Nell Finances

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.178 ± 0.005	0.601 ± 0.069	0.045 ± 0.025	10.948 ± 2.003 s
RDN-B	**-0.284 ± 0.019**	0.796 ± 0.032	0.114 ± 0.027	56.401 ± 11.519 s
TreeBoostler	-0.167 ± 0.006	**0.979 ± 0.003**	0.083 ± 0.026	2.6 ± 0.8 min
GROOT	-0.197 ± 0.028	0.976 ± 0.012	**0.167 ± 0.080**	534.8 ± 219.3 min

Table 5: Results for the experiment with Nell datasets, comparing with TreeBoostler, RDN-B-1 and RDN-Boost. The table shows the values for AUC ROC, AUC PR, CLL, and runtime. The first two rows show the results when learning the target dataset from scratch.

6 Conclusion

This work proposes a framework called GROOT aiming at transfer learning between relational domains. GROOT uses a genetic algorithm to find a mapping between predicates from source and target datasets. Along with generations, each individual carries a mapping for the transfer, exchange information with other individuals, and revise their structure trees to get better performance.

The results presented an improvement, when compared with the baselines, in the value of the metrics. In most of the experiments, GROOT improves the AUC ROC values, even with the genetic algorithm optimizing the AUC PR metric. We showed from experiments that GROOT reaches better or competitive results when comparing with another transfer learning framework. This is possible because our framework provides a larger search space to make the mappings between source and target domains, allowing more combinations for the predicates replacement. Unlike TreeBoostler, GROOT allows mapping many source predicates to many target predicates and one source type to many target types.

However, those enhancements are generally followed with a longer runtime than the baselines.

In future work, we intend to improve the evaluation component since the genetic algorithm evaluates, in every generation, many individuals to train and test the model. A possible solution for this problem is to rely on clever and faster inference procedures. Another improvement concerns the tree revision. In GROOT, the revision is made, at most, once in each tree, per generation. After finding the best mapping and generate the model, the resulting structure could be revised using a metaheuristic search, making prunings and expansions in the branches.

References

1. Inductive Logic Programming in a Nutshell. In: Introduction to Statistical Relational Learning. The MIT Press (08 2007)
2. Azevedo Santos, R., Paes, A., Zaverucha, G.: Transfer learning by mapping and revising boosted relational dependency networks. Machine Learning **109**(7), 1435–1463 (2020)
3. Bilenko, M., Mooney, R.J.: Adaptive duplicate detection using learnable string similarity measures. In: Getoor, L., Senator, T.E., Domingos, P.M., Faloutsos, C. (eds.) Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003. pp. 39–48. ACM (2003)
4. Burke, E.K., Kendall, G.: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer Publishing Company, Incorporated, 2nd edn. (2013)
5. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E.R.H., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: Fox, M., Poole, D. (eds.) Proc. of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010. AAAI Press (2010)
6. Davis, J., Domingos, P.M.: Deep transfer via second-order markov logic. In: Danyluk, A.P., Bottou, L., Littman, M.L. (eds.) Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009. ACM International Conference Proceeding Series, vol. 382, pp. 217–224. ACM (2009)
7. De Raedt, L.: Logical and Relational Learning. Springer Publishing Company, Incorporated, 1st edn. (2008)
8. Gandomi, A., Yang, X.S., Talatahari, S., Alavi, A.: Metaheuristic Algorithms in Modeling and Optimization, pp. 1–24 (12 2013)
9. Haaren, J.V., Kolobov, A., Davis, J.: Todtler: Two-order-deep transfer learning. In: Proc. of the 29th AAAI Conference on Artificial Intelligence. p. 3007–3015. AAAI'15, AAAI Press (2015)
10. Heckerman, D., Chickering, D.M., Meek, C., Rounthwaite, R., Kadie, C.: Dependency networks for inference, collaborative filtering, and data visualization. J. Mach. Learn. Res. **1**, 49–75 (Sep 2001)
11. Khosravi, H., Bina, B.: A survey on statistical relational learning. In: Proc. of the 23rd Canadian Conference on Advances in Artificial Intelligence. p. 256–268. Springer-Verlag, Berlin, Heidelberg (2010)

12. Kumaraswamy, R., Odom, P., Kersting, K., Leake, D., Natarajan, S.: Transfer learning via relational type matching. In: Aggarwal, C.C., Zhou, Z., Tuzhilin, A., Xiong, H., Wu, X. (eds.) 2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015. pp. 811–816. IEEE (2015)
13. Kumaraswamy, R., Ramanan, N., Odom, P., Natarajan, S.: Interactive Transfer Learning in Relational Domains. KI - Künstliche Intelligenz **34**(2), 181–192 (2020)
14. Mendelson, J., Naik, A., Raghothaman, M., Naik, M.: Gensynth: Synthesizing datalog programs without language bias. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 6444–6453 (2021)
15. Mihalkova, L., Huynh, T., Mooney, R.J.: Mapping and revising markov logic networks for transfer learning. In: Proc. of the 22nd National Conference on Artificial Intelligence - Volume 1. p. 608–614. AAAI Press (2007)
16. Mihalkova, L., Mooney, R.J.: Bottom-up learning of markov logic network structure. In: Ghahramani, Z. (ed.) Machine Learning, Proc. of the 24th International Conference (ICML 2007). ACM International Conference Proceeding Series, vol. 227, pp. 625–632. ACM (2007)
17. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
18. Muggleton, S., Tamaddoni-Nezhad, A.: QG/GA: a stochastic search for progol. Mach. Learn. **70**(2-3), 121–133 (2008)
19. Natarajan, S., Khot, T., Kersting, K., Gutmann, B., Shavlik, J.: Boosting relational dependency networks. In: Frasconi, P., Lisi, F.A. (eds.) Online Proc. of the International Conference on Inductive Logic Programming 2010. pp. 1–8 (Jun 2010), `https://lirias.kuleuven.be/handle/123456789/283041`
20. Natarajan, S., Khot, T., Kersting, K., Gutmann, B., Shavlik, J.W.: Gradient-based boosting for statistical relational learning: The relational dependency network case. Mach. Learn. **86**(1), 25–56 (2012)
21. Neville, J., Jensen, D.: Relational dependency networks. J. Mach. Learn. Res. **8**, 653–692 (May 2007)
22. Olivas, E.S., Guerrero, J.D.M., Sober, M.M., Benedito, J.R.M., Lopez, A.J.S.: Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA (2009)
23. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. on Knowl. and Data Eng. **22**(10), 1345–1359 (Oct 2010)
24. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
25. Pitangui, C.G., Zaverucha, G.: Learning theories using estimation distribution algorithms and (reduced) bottom clauses. In: Muggleton, S., Tamaddoni-Nezhad, A., Lisi, F.A. (eds.) Inductive Logic Programming - 21st International Conference, ILP 2011,Revised Selected Papers. Lecture Notes in Computer Science, vol. 7207, pp. 286–301. Springer (2011)
26. Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. Journal of Big Data **3**(1), 9 (2016). https://doi.org/10.1186/s40537-016-0043-6, `https://doi.org/10.1186/s40537-016-0043-6`
27. Yang, Q., Zhang, Y., Dai, W., Pan, S.J.: Transfer Learning. Cambridge University Press (2020). https://doi.org/10.1017/9781139061773