TRANSFER LEARNING FOR BOOSTED RELATIONAL DEPENDENCY NETWORKS THROUGH GENETIC ALGORITHMS

Leticia Freire de Figueiredo

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Rio de Janeiro
Dezembro de 2021

TRANSFER LEARNING FOR BOOSTED RELATIONAL DEPENDENCY NETWORKS THROUGH GENETIC ALGORITHMS

Leticia Freire de Figueiredo

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Aprovada por: Prof. Gerson Zaverucha
Prof. Aline Marins Paes Carvalho
Prof. Valmir Carneiro Barbosa
Prof. Aurora Trinidad Ramirez Pozo

RIO DE JANEIRO, RJ – BRASIL
DEZEMBRO DE 2021

Figueiredo, Leticia Freire de

Transfer learning for boosted relational dependency networks through genetic algorithms/Leticia Freire de Figueiredo. – Rio de Janeiro: UFRJ/COPPE, 2021.

XIV, 52 p.: il.; 29, 7cm.

Orientadores: Gerson Zaverucha

Aline Marins Paes Carvalho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 47 – 52.

1. transfer learning. 2. statistical relational learning. 3. genetic algorithm. I. Zaverucha, Gerson *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedico este trabalho à minha família, por todo apoio incondicional.

Agradecimentos

Agradeço a minha família, em especial aos meus pais, por todo o apoio e incentivo durante o meu mestrado.
Agradeço aos meus orientadores, Aline e Gerson, por terem aceitado me orientar e por toda a dedicação durante todo este período.
Agradeço aos meus amigos, por todos os momentos de alívio entre tantas provas e trabalhos. Em especial, os que me acompanham desde a graduação, no curso de Ciência da Computação. Serão inesquecíveis os momentos que tivemos antes das aulas.
Um agradecimento especial à Globo e ao time de Recomendação, do qual faço parte, que me auxiliaram durante esta reta final do mestrado.
Agradeço também ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo suporte financeiro, viabilizando esta pequisa.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

TRANFERÊNCIA DE APRENDIZADO PARA REDES DE DEPENDÊNCIA RELACIONAL COM BOOSTING ATRAVÉS DE ALGORITMOS GENÉTICOS

Leticia Freire de Figueiredo

Dezembro/2021

Orientadores: Gerson Zaverucha
Aline Marins Paes Carvalho

Programa: Engenharia de Sistemas e Computação

Algoritmos de aprendizado de máquina aprendem modelos a partir de um conjunto de observações, visando encontrar regularidades. Entretanto, métodos tradicionais de aprendizado de máquina falham ao encontrar padrões em objetos que possuem relações entre si. Aprendizado estatístico relacional constrói padrões a partir de domínios relacionais e lida com dados com incerteza. A maioria dos métodos de aprendizado de máquina assume que os dados de treinamento e teste possuem a mesma distribuição e o mesmo espaço de features. Porém, em alguns cenários, essa suposição não é válida. Transferência de aprendizado é uma ténica que aproveita o conhecimento aprendido em uma tarefa fonte para aprender um modelo para uma tarefa alvo. A transferência entre domínios relacionais encontra um desafio adicional, sendo necessário fazer o mapeamento entre os vocabulários fonte e alvo. Esta dissertação propõe GROOT, um framework que aplica algoritmo genético e suas variações para encontrar o melhor mapeamento entre tarefas fonte e alvo e adaptar o modelo transferido. GROOT recebe um conjunto de árvores de regressão relacionais construídas com os dados fonte como ponto de partida para construir o modelo para a tarefa alvo. Ao longo das gerações, cada indivíduo carrega um mapeamento. Eles são submetidos aos operadores genéticos que recombinam sub-árvores e revisam a estrutura inicial, permitindo a poda ou expansão dos ramos. Nós também propomos um novo algoritmo, chamado modified Biased Random-Key genetic algorithm (mBRKGA), um método baseado em BRKGA e mostramos o cálculo de complexidade de espaço do mapeamento proposto no framework. Resultados experimentais conduzidos em conjuntos de dados do mundo real mostram que GROOT alcança resultados, como AUC ROC, melhores que os baselines na maioria dos casos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TRANSFER LEARNING FOR BOOSTED RELATIONAL DEPENDENCY NETWORKS THROUGH GENETIC ALGORITHMS

Leticia Freire de Figueiredo

December/2021

Advisors: Gerson Zaverucha
Aline Marins Paes Carvalho

Department: Systems Engineering and Computer Science

Machine learning improves models with a set of observations, aiming to find regularities. 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 assume the training and test data come from the same distribution and feature space. Nonetheless, in several scenarios, this assumption does not hold. Transfer learning is a technique that leverages 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 vocabularies. This dissertation proposes GROOT, a framework that applies genetic algorithm and their variations to discover the best mapping between the source and target tasks and adapt the transferred model. GROOT relies on a set of relational regression trees built from the source data as a starting point to build the models for the target task. 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. We also propose a new algorithm called modified Biased Random-Key genetic algorithm (mBRKGA), a BRKGA-based method and show the space complexity calculation of the proposed mapping in the framework. Experimental results conducted in real-world datasets show that GROOT reaches results, as AUC ROC, better than the baselines in most cases.

Contents

List of Figures x

List of Tables xii

1 Introduction 1

1.1 Contributions . . . 3

1.2 Outline . . . 4

2 Main Concepts 5

2.1 Relational Learning . . . 5

2.1.1 Inductive Logic Programming . . . 6

2.1.2 Statistical Relation Learning . . . 8

2.1.3 RDN-Boost . . . 8

2.2 Transfer Learning . . . 12

2.3 Genetic Algorithm . . . 14

2.3.1 Population . . . 15

2.3.2 Genetic operators . . . 15

2.3.3 BRKGA . . . 17

2.4 Related work . . . 18

3 GROOT: Genetic algorithms to aid tRansfer Learning with bOOsT-srl 21

3.1 Population . . . 21

3.1.1 Space Complexity of the mapping . . . 24

3.2 Evaluation . . . 25

3.3 Simple Genetic Algorithm . . . 26

3.3.1 Selection . . . 26

3.3.2 Crossover . . . 26

3.3.3 Mutation . . . 27

3.4 BRKGA . . . 29

3.4.1 Selection . . . 29

3.4.2 Crossover . . . 29

3.5 mBRKGA . . . 29
3.5.1 Crossover . . . 30
3.5.2 Mutation . . . 31
4 Experimental Results **32**
4.1 Research questions . . . 32
4.2 Datasets . . . 32
4.3 Methodology . . . 34
4.4 Results . . . 35
5 Conclusion **45**
5.1 Future work . . . 46
References **47**

List of Figures

2.1 Example of RPT. The goal is to predict if a web page is from a student, based on the number of outgoing links. The numbers at a leaf report the number and percentage of positive and negative examples reaching that leaf. 10

2.2 Example of a relational regression tree. 11

2.3 Example of traditional machine learning process versus transfer learning process. 12

2.4 Schema showing how the population is defined in the example above. Each individual is defined by a vector, where each position represents the variable in the equation. The gene represents each position in the vector and the content in each position corresponds to an allele. . . . 15

2.5 Example of fitness score for each individual. The Equation 2.4 was used as fitness function. 16

2.6 Example of crossover operator. 16

2.7 Example of mutation operator. The selected alleles, in red, are mutated to new values according the problem constraints. 17

2.8 Example of crossover. The tossed coin is unbiased. So the probability to get head or tail is the same. 18

2.9 Schema showing how the crossover occurs in BRKGA. 18

3.1 Schema showing how the individuals are defined in GROOT. Each chromosome represents a tree and each allele, a node. The individuals carry the mapping between the source and target vocabulary. 23

3.2 Example of transfer learning using GROOT. 24

3.3 Example of crossover between two random individuals. The subtrees are exchanged since the red nodes. 27

3.4 Example of prune in mutation. The red node is a revision point and will be erased from the tree. 28

3.5 Example of expansion in mutation. The red node is a revision point and will receive a new node in one of the leaves. 28

3.6 Example of crossover in the BRKGA method. The red nodes are chosen to compose the individual. 30
3.7 Example of how the population is divided in the mBRKGA. 30
3.8 Operators in mBRKGA. The Figure shows the TOP and MIDDLE buckets being used in the crossover and the BOTTOM bucket in the mutation. 31

4.1 Learning curves for AUC ROC and AUC PR obtained from IMDB → UW-CSE using one fold to train. 41
4.2 Learning curves for AUC ROC and AUC PR obtained from WebKB (pageclass) → Twitter using one fold to train. 41
4.3 Learning curves for AUC ROC and AUC PR obtained from Cora → IMDB using one fold to train. 42
4.4 Learning curves for AUC ROC and AUC PR obtained from IMDB → Cora using one fold to train. 42
4.5 Learning curves for AUC ROC and AUC PR obtained from WebKB (departmentof) → IMDB using one fold to train. 43
4.6 Learning curves for AUC ROC and AUC PR obtained from WebKB (departmentof) → UW-CSE using one fold to train. 43
4.7 Learning curves for AUC ROC and AUC PR obtained from DDI → IMDB using one fold to train. 44

List of Tables

2.1 Information about the labeled data in the categories. 14

4.1 Statistics about the datasets. 34
4.2 Hyperparameters used in the optimization functon. 35
4.3 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. 36
4.4 Results for the experiment with WebKB and Twitter 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. 36
4.5 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. . 37
4.6 The mean of the probabilities to predict the label equals the example. The last column has the probability of a negative example being classified as positive. 38
4.7 Results for the experiment with Cora and IMDB 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. . 38
4.8 Results for the experiment with WebKB and IMDB 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. . 39

4.9 Results for the experiment with WebKB 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. 39
4.10 Results for the experiment with DDI and IMDB 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. . 39

List of Algorithms

1 Top-Level genetic algorithms 22

2 Building the population 25

3 Evaluating the population 25

4 Selecting the best individual in the population. 26

Chapter 1

Introduction

Machine learning algorithms induce models from a set of observations, finding patterns, regularities or classification rules. However, most of these algorithms are limited by the way they represent knowledge, not being able to represent domains in which multiple entities are connected by relations [1]. In this way, methods that take advantage of the structure of the relations between multiple entities and generate models are necessary so that they can show correlations among objects and theirs relationships [2]. *Relational Learning* is a machine learning branch with methods that generate models dealing with relational data. *Statistical Relational Learning* combines relational learning and statistical learning with uncertainty in data [2]. It differs from relational learning in the way it represents if an example is covered or not by the model. In relational learning, if an example is covered by the model, the answer is true; otherwise, the answer is false. Already in statistical relational learning, a probability is attributed to the answer. Instead of giving true or false as an answer, the example will be assigned with a value between 0 and 1, corresponding to the probability of being covered by the model [1]. Many SRL systems were developed, as Markov logic networks [3] and Boosted Relational Dependency networks [4].

Usually, machine learning and statistical relational learning algorithms assume training and testing data come from the same distribution and same feature space. When new data arrives, and it do not comply with the old data, it would be necessary to collect more data to build the model from scratch. Nevertheless, collecting more data can be difficult because this effort is expensive to do, time-consuming or unrealistic in some scenarios. One solution for the lack of data problem is *transfer learning* [5–7]. In general terms, *transfer learning* comprises techniques that leverage the knowledge learned in one or more source task to improve the learning in a related target task [8].

Transferring knowledge between relational domains constitutes an additional challenge since the vocabulary of source and target domains are commonly different.

The vocabulary in relational domains is composed of predicates. A predicate corresponds to a relation between entities, defined as arguments. For example, a predicate called sister/2 can be defined, indicating a relation between two arguments. Two constants - for example, Anna and Mary - when following the predicate, build an atom. In this case, if we add the constants in the predicate sister/2, we have the atom sister(Anna, Mary), indicating that Anna is the sister of Mary. Also, the arguments could be typed, according to the system we are using. In the sister relation, the arguments can be typed as person. The types are defined into the language bias, which here follows the Aleph and Progol definitions. Aleph and Progol are Inductive Logic Programming systems, written in Prolog programming language, that define how to declare the predicates [9, 10].

One way to make the transfer is to establish a mapping from both vocabularies. After that, the target task can benefit from the model built from the source domain. In [11] and [12], giving the model in the form of a relational regression tree, their frameworks find a mapping where the source predicates are replaced by the target predicates, through a constrained search space built upon the source and target predicates.

Finding the best mapping can be challenging if the source domain, target domain or both have a large amount of predicates and types. A way to tackle this problem is to use metaheuristics to build a set of viable solutions to the mapping problem. During te search, it will follow, along many executions, the path indicated by the best solution, generating best mappings with operations which could enable to change the solutions.

This work proposes GROOT[1], a framework powered with Python, where relational regression trees, generated by the relational source domain, are used as a starting point to learn a model in the target task. GROOT leverages a genetic algorithm [13] and their variations to find the best vocabulary mapping between the relational domains and revising the structure tree, making possible to add or prune nodes. Genetic algorithms operate by changing individuals with specific operators, as mutation and crossover. Here, each individual carries a mapping and each mapping is evaluated by the RDN-Boost [4] method, creating a set of trees that approximate the joint probability distribution over the variables as a product of conditional distributions. RDN-Boost needs less time to train and it outperforms other SRL methods. A solution is considered the best if: 1) it is a feasible solution, respecting all the constraints defined by the problem and 2) it minimizes the negative area under the Precision-Recall curve (AUC PR) metric. If GROOT gives more than one solution that fits these restrictions, we choose the solution with the minimum negative conditional log-likelihood (CLL).

[1] https://github.com/MeLLL-UFF/groot

We present three genetic algorithms versions: simple genetic algorithm, Biased-Random Key Genetic Algorithm (BRKGA) [14], where we implemented our own BRKGA program, and a novel proposed modified Biased-Random Key Genetic Algorithm (mBRKGA). The BRKGA algorithm applies a bias in the generation of the individuals, using the best individuals for recombination and allowing to find better or equal solutions in comparison to the previous population. Different from BRKGA, the mBRKGA algorithm does not allow the worst population individuals to make part of the individuals recombination, applying transformation in the worst individuals only in mutation. This choice will contribute to generating better individuals as in the BRKGA algorithm.

In the *simple genetic algorithm*, the crossover operator exchanges random subtrees between two random individuals. The mutation operator can expand or prune a branch, applying a revision in the tree, modifying the structure. The *BRKGA* [14] splits the population into the best elite individuals, and the remaining population. The crossover operator generates one offspring by choosing at random a node from the elite group and another individual from the remaining part of the population. The mutation operator generates new individuals as in the first generation: assigning a random mapping to the new mutate individuals.

Moreover, we propose a new algorithm, called *mBRKGA*, based on BRKGA, where the population is divided into top, with the best individuals, bottom, with the worst individuals and middle, with the remaining individuals. In this case, the crossover operator generates a new offspring, as in the BRKGA, but choosing a random individual from the top and another from the middle. The mutation operator revises the structure, as in the simple genetic algorithm, but only in the bottom individuals. We also present a space complexity calculation according to our proposed mapping solution.

We evaluated GROOT in real-world datasets and simulated scenarios where only a few data are available to train, selecting one fold to make part of the training set and the remaining for the testing set. The experimental results contribute to answering the research questions that measure how the framework performs when compared with another transfer learning framework and the model learned from scratch using the target domain. Our results demonstrate that GROOT gives comparable results, concerning the baselines, and outperforms the value of the metrics in some experiments. However, to achieve these results, we needed a huge execution time to get the best mapping between the tasks.

1.1 Contributions

In summary, this dissertation makes the following contributions:

- Propose a novel framework to transfer learning between relational domains, mapping the predicates between source and target tasks using genetic algorithm and their variations.
- A proposal of a new genetic algorithm variation, called mBRKGA, that bias the recombination between the individuals to guarantee better solutions in the future generations.
- The space complexity calculation for the proposed mapping in the framework. As will be explained in Section 3, GROOT allows mapping one source predicate to many target predicates, enabling a larger search space.
- Several experiments to evaluate the proposed framework. We conducted the experiments with different amounts of training data, using real-world datasets and comparing with another transfer learning framework and the model learned from scratch using the target domain.
- A paper accepted in the 30th International Conference on Inductive Logic Programming[2], showing the framework, with experiments using the simple genetic algorithm [15].

1.2 Outline

The remainder of the text is organized as follows. In chapter 2, we will present some important concepts to understand this work, as the related works already made with transfer learning and relational domains. In chapter 3, we will present the framework, giving some implementation details. Next, in chapter 4, we will show how we did the experiments with the framework and discuss the results. Finally, we present our conclusions in chapter 5 and point out future works.

[2] http://lr2020.iit.demokritos.gr/ilp/index.html

Chapter 2

Main Concepts

In this chapter, we introduce important concepts to the reader to understand the foundations of our contributions. First, we explain some important concepts about statistical relational learning. Next, we describe the RDN-Boost method, explain basic notions on transfer learning, and finalize the section with genetic search and optimization. We also review the related work regarding our proposed work.

2.1 Relational Learning

Logical and relational learning is an artificial intelligence subfield that combines principles and ideas from machine learning and knowledge representation [1]. Machine learning studies systems that can improve their performance with experience. Nevertheless, most of the machine learning techniques are limited from a knowledge representation perspective because they cannot represent domains with multiple entities and their relationships. A *relational representation* is able to represent entities with relationships amongst them. In this way, logical and relational learning is the study of machine learning and data mining within expressive knowledge representation reaching relational or first-order logic [1].

In logic, a *predicate* defines a relation and can be followed by some arguments. The representation *p/n* denotes a predicate *p* with *n* arguments. The number of arguments explicitly listed is defined as *arity*. The arguments of the predicates are *terms* and the terms can be a *constant* or a *variable* [1, 16]. In the code 2.1, *parent/2*, *grandparent/2* are predicates with arity equals to 2. Also, *je* , *paul* and *ann* are constants and *X* is a variable. The arguments could be associated with a type. For example, *je* can be associated to the *person* type. Before the type, some ILP systems define a mode type, indicating if the argument should be an "input" variable of the type, using the + symbol (*+person*), if the argument is an “output” variable of type (*-person*), or specifying that it should be a constant of type (*#person*) [10].

Listing 2.1: Example of logic program.

```
parent(jeff ,paul). parent(paul,ann).
parent(charles ,mary). parent(mary,john).
grandparent(X,Y) ← parent(X,Z), parent(Z,Y).
```

Atoms are formed by predicates followed by the number of terms according to the arity of the predicate. For example, *parent(charles,mary)* and *parent(mary,john)* are atoms. *Literals* are atoms as they are. *parent(paul, ann)* is a positive literal and their negations *not parent(paul, ann)* is a negative literal [16].

With these concepts, we can defined a logic program. A *logic program* comprehends a set of clauses. A *clause* is an expression of the form

$$A \leftarrow B_1, B_2, \ldots, B_m \tag{2.1}$$

where A and $B_i, i \in [1, m]$ are logical atoms. The symbol , stands for conjunction (and) and $\leftarrow$, for implication (if). Furthermore, all variables are universally quantified [1, 16]. In the example of the code 2.1, the logic program $grandparent(X, Y) \leftarrow parent(X, Z), parent(Z, Y)$ can be read as X is grandparent of Y if X is parent of Z and Z is parent of Y.

2.1.1 Inductive Logic Programming

Inductive logic programming focuses on finding a logic program H, called *hypothesis*, from a set of negative and positive examples and a background (prior) knowledge [16]. Suppose we want to learn the definition for $daughter/2$. The following facts are available as examples:

```
Pos daughter(dorothy, ann).
    daughter(dorothy, brian).
Neg daughter(rex, ann).
    daughter(rex, brian).
```

and the following background knowledge B

```
mother(ann, dorothy).
female(dorothy).
female(ann).
mother(ann, rex).
father(brian, dorothy).
father(brian, rex).
```

With this informations, we can infer the hypothesis H:

```
daughter(C, P) ← female(C), mother(P, C).
daughter(C, P) ← female(C), father(P, C).
```

where, the following conditions are satisfied:

- **Completeness:** the hypothesis covers all positive examples.
- **Consistency:** the hypothesis covers none of the negative examples.

When one or both of those conditions are not completely satisfied, it is possible to revise the hypothesis to generate a more accurate theory. Theory revision systems, as FORTE [17], search for revision points, that are clauses or literals responsible for misclassified examples. The theory revision systems assume that the initial theory is approximately correct and making revision in some points is more efficient than learning again from the scratch [18, 19].

The revision points can be splitted in two types:

1. **Generalization** revision points, which are clauses or literals contributing to failing at classifying positive examples, indicating the theory is too specific.
2. **Specialization** revision points, which are clauses or literals contributing to prove negative examples, indicating the theory is too general.

It is important to know the type of the revision point to apply a consistent revision, according to the dataset [18, 19].

The revision operators can add new rules or delete from the theory. When dealing with generalization points, the following operators can be used:

- **Delete-antecedents**: removes literals marked as revision points from clauses that cannot prove positive examples.
- **Add-rule**: generates new clauses from revision points, adding literals to existent clauses or learning a clause from scratch.

Otherwise, when dealing with specialization points, it is possible to apply the following operators:

- **Delete-rule**: erases a clause that proves negative examples.
- **Add-antecedent**: adds new literals in the inconsistent clause, highlighted as revision point.

2.1.2 Statistical Relation Learning

We can extend ILP to deal with uncertainty and noise with probabilistic formalisms, where two changes occur in the representation:

- clauses in the hypothesis and in the background knowledge are annotated with probabilistic information;
- the coverage relation becomes probabilistic.

Probabilistic ILP defines the probability of an example given the hypothesis and the background theory. Given as arguments an example e, the hypothesis H and the background knowledge B, the logic program returns the probability value $P(e|H,B)$, between 0 and 1. Probabilistic ILP objectifies a formal framework for statistical Relational Learning [16].

SRL learns in domains with complex, relational, and rich probabilistic structures. The probabilistic semantic are mostly based on graphical models or stochastic grammars. Initially, SRL approaches were mostly defined as directed graphical models, representing complex generative models. Later, there has been a growing interest in undirected models, as Markov networks. Different from directed models, the undirected ones can represent non-causal dependencies. Other alternatives are dependency networks as Relational dependency network [20], the formalism we follow in this dissertation.

2.1.3 RDN-Boost

This section will introduce the statistical relational learning method used in this work, namely RDN-Boost. Whereas current relational dependency network approaches learn only a single relational probability tree per predicate, RDN-Boost learns a set of relational regression trees using gradient-based boosting. In this section, we will review the concept of dependency networks and relational dependency networks. Finally, we will present the RDN-Boost.

Dependency Networks

Dependency networks (DNs) are an alternative form of graphical models that approximate the full joint distribution of a set of random variables with a set of conditional probability distributions (CPDs), learned independently. DNs combine characteristics of both undirected and directed graphical models, encoding the probabilistic relationships between a set of variables X. Dependencies are represented with a bidirected graph defined as $G = (V, E)$ and a set of conditional probability

distribution. Each node $v_i \in V$ corresponds to a variable in X and it is associated with a conditional probability distribution on the parents pa_i of the node v_i, calculated as:

$$P(v_i) = P(x_i|pa_i) = P(x_i|X \setminus \{x_i\}) \tag{2.2}$$

where the parents of the node i are the set of variables that turn $X_i \in X$ conditionally independent of the other variables and G contains an edge from parent node v_j to the node v_i if $X_j \in pa_i$ [21].

The structure and parameters of the model can be determined through learning the local CPDs. Each variable x_i has its distribution learned separately, conditioned on the other variables. Even the approach being simple and efficient to learn, learning DNs can result in an inconsistent network when learning the CPDs from finite samples. DNs are nearly consistent if learning from large datasets. However, despite the inconsistency, approximate inference techniques, as Gibbs sampling, can still be used to estimate the full joint distribution, regardless of the consistency of the local CPDs [20].

Relational Dependency Networks

Relational dependency networks (RDNs) extend DNs to the relational setting, upgrading them to a logical perspective. As in DNs, RDNs use a bidirected graph G_M with a set of conditional probability distributions P. RDNs comprehend a set of predicates Y that could be grounded given the instantiation of the variables. Each predicate $y_i \in Y$ is associated with a conditional probability $P(y_i|pa_i)$. The conditional probability is defined over the values of y_i given its parents. The algorithm to learn RDNs are similar to the DNs learning algorithm, except it uses a relational probability estimation algorithm to learn a set of conditional models, to maximize the pseudo-likelihood for each variable independently [4, 20].

Since RDNs can be represented as a set of conditional distributions, learning RDNs correspond to learning these distributions [4]. To capture these distributions, Neville et al. [21] use relational probability trees (RPTs). RPTs, as seen in Figure 2.1, are selective models that extend classification trees to a relational setting. The RPT algorithm utilizes aggregation functions to map a values set into a single feature value. For example, considering the publication dates on references of a paper, the RPT could build a feature that will test if the average publication date was after 1995 [21].

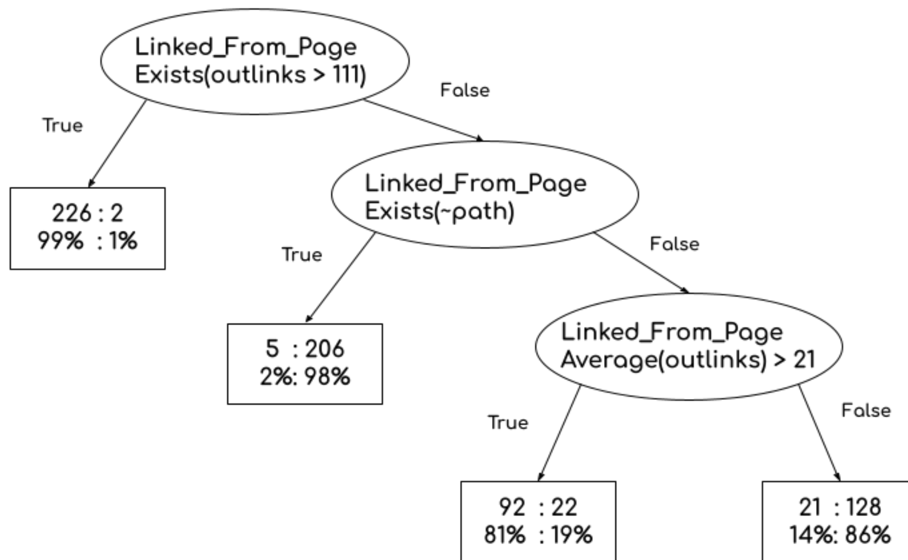

Figure 2.1: Example of RPT. The goal is to predict if a web page is from a student, based on the number of outgoing links. The numbers at a leaf report the number and percentage of positive and negative examples reaching that leaf.

Learning RDNs

Instead of representing the conditional probability distributions as RPTs, Natarajan et al. [4] replaced to relational regression trees (RRTs), considering the CPD of each predicate as a set of RRT. The Figure 2.2 has an example of RRT where the goal is to predict if A is advisedBy B. Following the tree, if B is professor, A is not professor, A has more than 1 publication and more than 1 publication with B, then the regression value is 0.09. Negative values indicate lower probabilities, i.e., the probability of the target predicate given that a particular path is true is less than 0.5 [22]. In this example, when following path B is professor and A is professor, the regression value of the path is -0.06, indicating that A is not advisedBy B.

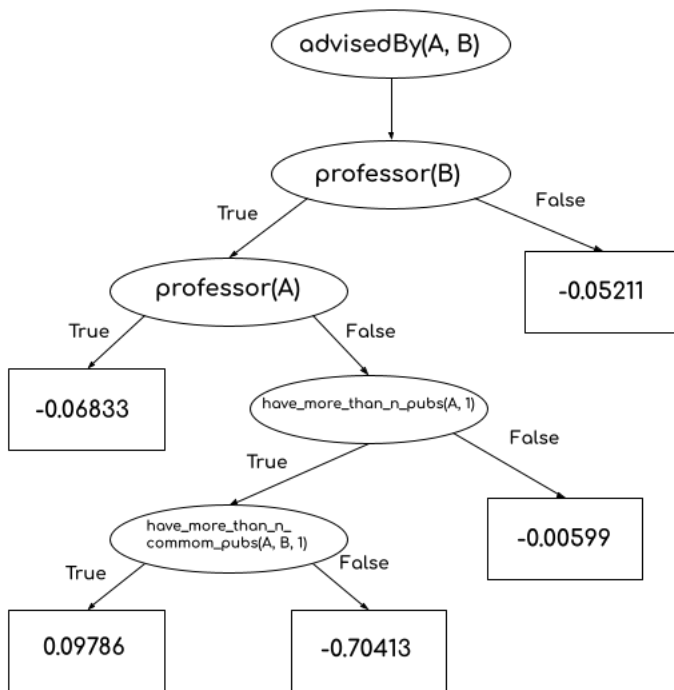

Figure 2.2: Example of a relational regression tree.

The idea is to learn a set of RRT, using gradient-based boosting. These trees are learned in the way to maximize the likelihood of the distributions with respect to the potential function, in each iteration. The potential function is defined using an initial potential ψ_0 and, iteratively, adding the gradients Δ, which is the gradient of the likelihood function. In this way, the potential function is given by

$$\psi_m = \psi_0 + \Delta_1 + ... + \Delta_m \tag{2.3}$$

where Δ_i is the gradient at iteration *i*. This approach resulted in the RDN-Boost algorithm.

In RDN-Boost, the learner TILDE [23] was used to learn the regression trees. For each predicate (y_k), the examples are generated to the learner TILDE to update the model with new regression trees. This routine is repeated until a pre-defined number of iterations M. After m steps, the model will have m regression trees, where each tree approximates the corresponding gradient for the predicate y_k, defined as Δ_m^k.

To generate the example for the learner, the algorithm takes as input the current predicate index (k), the data and the actual model. For each example, the probability and gradient are computed. For each tree, the regression values are computed based on the groundings of the current example. The gradient is then set as the weight of the example. The gradient boosting allows to learn the structure

and parameters of the RDN at same time, since the set of regression trees for each predicate builds the structure of the conditional distribution and the sets of leaves form the parameters of the conditional distribution [4].

2.2 Transfer Learning

Machine learning algorithms build models using a set of labeled training instances. Traditionally, these methods assume training data and testing data have the same feature space and same data distribution. When this scenario does not occur, it is necessary to collect more data to rebuild the models from scratch using the new training data. However, collecting more data is expensive, time-consuming, and difficult. To reduce the effort in collecting more data, a technique called transfer learning can be used [5–7]. Transfer learning aims to improve a target model using knowledge from one or more related source tasks. In a traditional machine learning setting, each task is learned from scratch, independently. In transfer learning, the knowledge from previous related tasks is transferred to a target task when the target data has a few quality [5]. In the real world, many transfer learning situations can be observed. For example, if a person who knows how to play violin can learn faster how to play piano because both musical instruments share common knowledge [7]. Transfer learning is applied in many tasks as image classification [24–26], sentiment classification [27–29] and software defect prediction [30, 31].

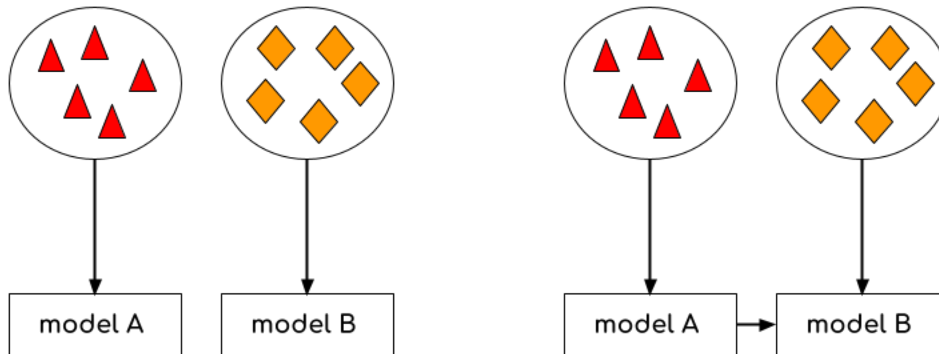

Figure 2.3: Example of traditional machine learning process versus transfer learning process.

Formally, a domain $\mathcal{D}$ can be defined as $\mathcal{D} = \{\mathcal{X}, P(X)\}$, where $\mathcal{X}$ is the feature space and $P(X)$ is the marginal probability distribution, where $X = (x_1, x_2, \ldots, x_n) \in \mathcal{X}$. With a specific domain, a task $\mathcal{T}$ is defined as $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. The $\mathcal{Y}$ is the label space and $f(\cdot)$ is an objective predictive function, used to predict from unlabeled x instances, corresponding to $f(x)$. In a probabilistic point of view,

$f(x)$ can be defined as $P(y|x)$ [5]. Suppose a document classification task, where each term is viewed as a binary feature and the task is to predict a label “True” or “False”. In this scenario, the feature space $\mathcal{X}$ is the space of all term vectors and x_i correponds the i_{th} document term vector. The $\mathcal{Y}$ space contains all labels “True” or “False” correspoding to each x_i [5]. Using the above definitions, *transfer learning* is defined as below.

Definition 2.2.1 (Transfer Learning) *Given a source domain $\mathcal{D}_S$, a source task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, transfer learning aims to improve the target predictive function $f(\cdot)_T$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$ and $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$ [5].*

The definition above implies in two cases:

1. the condition $\mathcal{D}_S \neq \mathcal{D}_T$ is true, so $\mathcal{X}_S \neq \mathcal{X}_T$ or $P_S(X) \neq P_T(X)$;
2. the condition $\mathcal{T}_S \neq \mathcal{T}_T$ is true, so $\mathcal{Y}_S \neq \mathcal{Y}_T$ or $P(Y_S|X_S) \neq P(Y_T|X_T)$.

If $\mathcal{D}_S = \mathcal{D}_T$ and $\mathcal{T}_S = \mathcal{T}_T$, the problem becomes a traditional machine learning problem [5]. The case where transferring the knowledge from the source task has a detrimental effect on the target task is called *negative transfer*. Supposing a source domain $\mathcal{D}_S$, a source task $\mathcal{T}_S$, a target domain $\mathcal{D}_T$, a target task $\mathcal{T}_T$, a predictive function $f_{T1}(\cdot)$, learning only with the domain $\mathcal{D}_T$ and a predictive function $f_{T2}(\cdot)$, learned from the transfer using predictive function $\mathcal{D}_S$. If the performance of predictive function $f_{T2}(\cdot)$ is worst than predictive function $f_{T1}(\cdot)$, a negative transfer occurred [6].

Transfer learning techniques can be split into threes categories: inductive transfer learning, transductive transfer learning and unsupervised transfer learning, mainly based on situation between domains ans tasks.

In *inductive transfer learning*, the target and source tasks are different, no matter about the domains. In this case, it is necessary some labeled data in the target domain to induce the target predictive function $f_T(\cdot)$. In *transductive transfer learning*, the source task is the same as the target task but the domains are different. Nevertheless, in this case, the labeled data comes only from the source domain, while the target domain has no labeled data. This scenario occurs when 1) $\mathcal{X}_S \neq \mathcal{X}_T$ or 2) $\mathcal{X}_S = \mathcal{X}_T$ and $P(X_S) \neq P(X_T)$. Finally, the *unsupervised transfer learning* occurs when the target task is different but related to the source task and no labeled data is available for both source and target domains. This category focus on unsupervised tasks in the target domain [5, 7]. This work fits in the inductive transfer learning category since the source domain is available and a few data in the target domain is necessary.

Table 2.1: Information about the labeled data in the categories.

	Labeled data	
	Source domain	Target domain
Inductive transfer learning	Available/Unavailable	Available
Transductive transfer learning	Available	Unavailable
Unsupervised transfer learning	Unavailable	Unavailable

The above three settings can be summarized into four cases. The fist case, *instance-based transfer learning*, assumes that part of the data from the source domain can be used in the target domain by re-weighting. The next case, *feature-representation transfer*, encodes the knowledge used to transfer across domains into a feature representation, to improve the performance of the target task. The *parameter-transfer* case assumes the tasks share some parameters and the last case, *relational- knowledge transfer*, focus on transfer learning within relational domains [5].

2.3 Genetic Algorithm

A problem can have many feasible solutions, but just a few or only one optimum solution. When dealing with a large space of solutions, the problem complexity can make the process of optimum solution search unfeasible [13]. Metaheuristic algorithms allow for the space to be searched at random so that the method can find a good local solution, nearly optimal, in an acceptable time.

Metaheuristic algorithms have two main components:

1. diversification: generating diverse solutions to explore the search space;

2. intensification: focusing the search in a local space, exploiting the information that a good solution can be found in the region.

Also, these algorithms can be defined in many ways. One definition of them is population-based. An example of a population-based algorithm is the genetic algorithm [32]. *Genetic algorithm* is based loosely on simulated evolution and provides a collection of candidate hypotheses, called population, where the algorithm searches for the best one by mutating and recombining the candidates, replacing a fraction of the population. The best candidate is defined according to an objective function, called fitness, that defines criteria to rank the potential hypotheses [32], [33].

2.3.1 Population

Genetic algorithm relies on a population composed of individuals. Each individual represents a candidate solution for the problem. The number of individuals in the population is an important factor that affects the scalability and performance of the algorithm. Small populations tend to premature convergence and produce substandard solutions. Otherwise, large populations lead to unnecessary waste of computational time [34].

The individual is composed by a chromosome, where the solution is coded. The chromosome has elements called genes, with varying values at specific positions, called alleles [35]. For example, if we want to minimize the function

$$F(x) = -x^2 + y^3 - z^3, \quad x, y, z \in \{-100, \dots, 100\} \tag{2.4}$$

we can represent each individual as a chromosome, composed with a vector where each position is a gene representing the variables x, y, z, respectively and varying between -100 and 100, as showed in Figure 2.4.

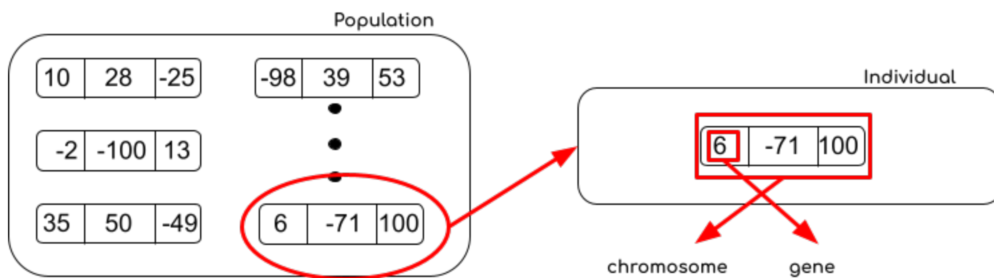

Figure 2.4: Schema showing how the population is defined in the example above. Each individual is defined by a vector, where each position represents the variable in the equation. The gene represents each position in the vector and the content in each position corresponds to an allele.

Each individual in the population is evaluated according a defined fitness function and receives a fitness score [33]. In our example, the Equation 2.4 is the fitness function that will measure the candidate solutions of the problem.

2.3.2 Genetic operators

The next steps in the genetic algorithm is to apply genetic operators in the individuals. The operators can recombine, mutate the individuals and select the fittest for the next generation.

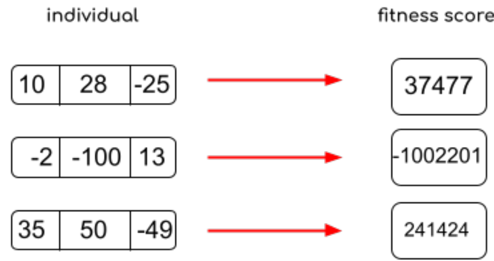

Figure 2.5: Example of fitness score for each individual. The Equation 2.4 was used as fitness function.

Selection

The selection operator selects the individuals with the best evaluation in the actual generation, to allocate more copies in the next generation with better fitness values. A way to make the selection is using the tournament selection, where n individuals are randomly selected and go into a tournament against each other selected individuals. The fittest individual wins the tournament and is selected. It is necessary m tournaments to select m individuals [34].

Crossover

The next step recombines parts of two random individuals to create two new offsprings, with a crossover probability p_c. The choice of which individual part will compose which offspring part is determined by the crossover methods. In this method, we can define, at random, one or two points in the individual to cut and exchange to generate the new individuals [34]. As illustrated in Figure 2.6, two random individuals are selected and a one-point crossover is applied to generate two new offsprings.

Figure 2.6: Example of crossover operator.

Mutation

Unlike the crossover operator, the mutation operator will generate a new individual, modifying the alleles. This operator is important to add diversity to the population and allows to explore the entire search space, since the operator can modify

the alleles, generating a new solution. A random individual is chosen and with a mutation probability p_m, each allele can be mutated to a new value according the constraints [34].

Figure 2.7: Example of mutation operator. The selected alleles, in red, are mutated to new values according the problem constraints.

At the end of all steps, we have a new generation of the genetic algorithm. We can also apply a elitist selection, where the n fittest individuals are guaranteed to be selected for the next generation.

2.3.3 BRKGA

Random-key genetic algorithms (RKGA) are a class of genetic algorithms to solve combinatorial optimization problems involving sequencing. In this class, the population is made with p vectors of random-keys, where each allele is generated independently, in a random way, in the real interval between (0, 1]. After computing the fitness of each individual, the population is divided into two groups of individuals: a group with p_e elite individuals, with the best individuals, and another group with $p - p_e$ individuals, with $p_e < p - p_e$. To produce a new generation, an elitist strategy is used. The elite individuals are copied from the generation G to generation $G + 1$. In the mutation, mutants are introduced into the population, produced in the same way that an individual in the initial population is generated. The crossover operator generates new offspring selecting two individuals at random in the entire population. After selecting the individuals, a biased coin is tossed to choose which individual, with their genes, will contribute to the offspring. Suppose two individuals - individual A and individual B - every time the coin toss head, we select genes from the individual A and, otherwise, when the coin toss tail, we select from the individual B [36].

In [14], the authors propose Biased random-key genetic algorithm (BRKGA). This algorithm is based on RKGA and differs on how to select the individuals for the crossover. In BRKGA, one individual is selected from the elite individuals and the other is chosen from the remaining population, as showed in Figure 2.9.

BRKGA was already used in many optimization applications [37], [38], [39]. For example, in [40], the authors solve the job-shop scheduling problem with BRKGA. Given n jobs, with a set of operations, and m machines, the problem consists in

Figure 2.8: Example of crossover. The tossed coin is unbiased. So the probability to get head or tail is the same.

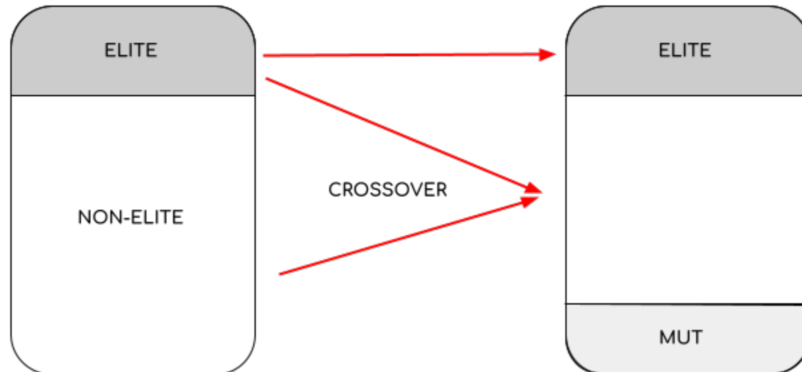

Figure 2.9: Schema showing how the crossover occurs in BRKGA.

finding a schedule that minimizes the finish time of the last operation in the schedule, according the precedence constraints. In the experiments, the BRKGA heuristic found the best-know solution for 31 in 43 instances of the problem.

2.4 Related work

Previous works have already proposed novel methods for leveraging transfer learning to relational domains. In [11], the authors propose a framework called TreeBoostler to transfer an 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 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 employs a final revision step. Such a component selects the revision points as the places in the trees with low probabilities for the true class. Next, the revision operators either expand leaves to generate new branches or prune nodes from the set of selected revision points. 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 because 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 quite an expensive process. Nevertheless, TreeBoostler was successfully compared to TAMAR [41] and TODTLER [42]. Another work that leverages transfer knowledge between RDN-Boost models is presented in [12]. The proposed framework TransBoostler takes advantage of a structure learned from the source task using RDN-Boost, searching for the best mapping between the predicates in the source and target domains that will improve the model in the target task. The mapping is found using the similarity between the predicates in the source and target domains. Each predicate is represented in the same feature space, using pre-trained word embeddings on Wikipedia. The predicates are transformed into vectors, where each component corresponds to the terms in the predicates. For example, the predicate *companyhaso ce* can be split into three terms (*company has o ce*) and translated into a vector with three components **(x, y, z)**. The similarity between pairs of predicates is calculated - one from the source domain and the other from the target domain. The source predicate is mapped to the most similar target predicate. Finally, a revision step is applied. As in TreeBoostler, one predicate from the source domain can only be mapped to one predicate in the target domain.

TAMAR [41] was one of the first proposals of transfer learning to statistical relational learning. It maps a source Markov Logic Network [3] to a target domain and makes revisions in the structure to improve the performance. The mapping between the source and target predicates is made locally, finding a single mapping at a time instead of finding one mapping and translating all the source at once. The system does an exhaustive search through the space of all possible mappings. Mapping is considered legal if the source predicate is mapped to a compatible target predicate or an empty predicate. For two predicates to be compatible, they need to have the same arity, and the constraints of the type of the arguments need to be compatible too. After the transfer, the structure is revised, shortening a clause or lengthening it. The mapping process in TAMAR could be a problem when the domains have a large number of predicates; searching through all the possible mappings would be unfeasible. TODTLER [42] uses Markov Logic Networks to deep transfer learning. The first step is to generate a second-order clauses model which contains predicate variables and restricts clause length. Then, a first-order model is built from the second-order model with the set of relevant predicates to the source data. At the final step, using the target domain, TODLER determines a Markov Logic Network that maximizes the joint probability of the data.

Our framework is similar to the introduced solutions but makes use of a genetic algorithm instead of an exhaustive search. Also, GROOT allows mapping between many source predicates to many target predicates, just adding an identifier to the

source predicate to turn it into a unique one. The source types can be mapped to many target types.

Metaheuristics have already been used in Inductive Logic Programming. In QG/GA [43], 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 to form the population.

In [44], 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 samples 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 the Reduce algorithm in bottom clauses to reduce the search space.

Chapter 3

GROOT: Genetic algorithms to aid tRansfer Learning with bOOsTsrl

In this work, we propose GROOT framework to transfer learning between similar tasks when addressing relational domains. The framework leverages the built structures generated by RDN-Boost, using the source dataset as a starting point to learn how to solve the target task and find the best mapping between the source and target predicates. Instead of doing a complete search into the solutions space, GROOT applies genetic algorithms in three versions: a simple genetic algorithm, BRKGA, and mBRKGA, to find the mapping that optimizes the area under the Precision-Recall curve (AUC PR) value. Also, in GROOT, the genetic operators are employed to revise the source structure better to accommodate the target examples. The Algorithm 1 shows how GROOT works. The functions used in the algorithm are explained further ahead.

Definition 3.0.1 (GROOT) *Given a set of relational regression trees, generated by the source domain, a target domain and a target task, GROOT searches the best mapping between the vocabularies using genetic algorithm and their variations.*

3.1 Population

Along with the generations of a genetic algorithm, each individual in the population carries a solution to the problem. In GROOT, we assume that the input includes a set of trees from the source dataset created with RDN-Boost, using predicates from the source vocabulary. Each individual has a copy of this set, a feasible mapping corresponding to each node in the trees, and the fitness function value. Each chromosome is a structure corresponding to set of trees, and the gene corresponds to a tree. Each tree has alleles corresponding to the value in each node. The genes

Algorithm 1 Top-Level genetic algorithms

Require: *theory*, set of relational regression trees
Ensure: *transferred model*, the regression trees generated with the target dataset
  population = POPULATION($n, theory$)
  EVALUATE($population$)
  num_generations ← 0
  while $num_generations \neq$ NUM_GENERATIONS **do**
    best_individual ← ELITISM($population$)
    apply selection in the population using the tournament selection, with 3 individuals in each tournament
    apply crossover in the population, with crossover rate equals to p_c
    MUTATION($population, MUTATION_RATE$)
    EVALUATE($population$)
    worst_individual ← GET_WORST_INDIVIDUAL($population$)
    population ← population \ worst_individual
    population ← population + best_individual
    $num_generations \leftarrow num_generations + 1$
  end while

in the same individual could have different sizes. Each individual has N genes. In GROOT, we defined N = 10. Figure 3.1 depicts how the individuals are encoded in GROOT.

GROOT allows mapping many source predicates to many target predicates. So, each predicate in the source dataset can be mapped to one or more predicates present in the target dataset. To distinguish which source predicate has been replaced, a unique identifier is appended to each one of them, allowing for more solutions. Supposing the framework only allows to map the source predicate to one target predicate; if the source predicate is always the same, the transfer will also be the same. Mapping between the predicates is made randomly but following some constraints:

1. the target predicate needs to have the same arity as the source predicate.
2. The target predicate needs to have the same language bias as the source predicate.

If the target dataset does not have any predicates that comply with the constraints, the source predicate is mapped to a predicate defined as $null/0$. For example, this can occur when the target dataset only has predicates with arity greater or equal to 2 but the source dataset has predicates with arity equal to one. Also, the predicates are mapped sequentially, respecting the appearance order in the trees.

It is also necessary to map the predicate types as defined in the language bias. A type from the source dataset can be mapped to one type from the target dataset, but a target type can replaced by many source types. When choosing the target predicates that can be mapped to the source predicate, we need to take into account

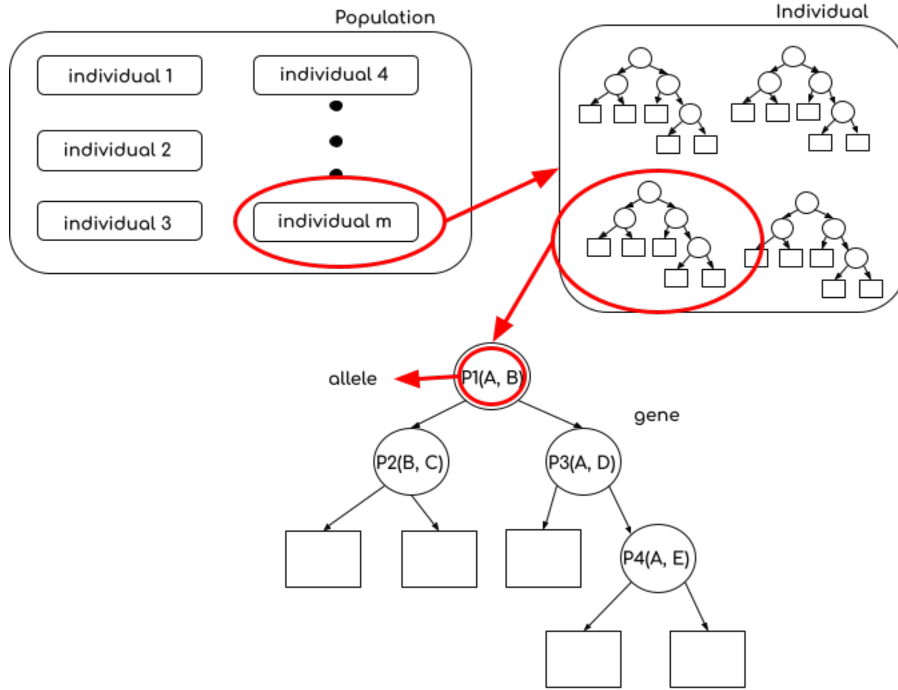

Figure 3.1: Schema showing how the individuals are defined in GROOT. Each chromosome represents a tree and each allele, a node. The individuals carry the mapping between the source and target vocabulary.

the existent mapped types. First, we find a mapping between the types that appear in the predicates, following the sequence in trees. The first predicate does not have its types mapped. In this scenario, we can select the predicates only according to the constraints already presented above. The next predicates have already mapped types, so we only consider the target predicates that respect the types mapped. After selecting target types, we select the predicates that generate a viable mapping. The Algorithm 2 presents the routine to build the population. In Figure 3.2, we show an example of the replacement betweeen the source and target predicates. Note that different predicates (*director* and *actor*) are mapped to the same predicate (*student*).

We can also permutate the types in the predicates. Given the selected predicate, the framework generates all the possible permutations between the types. If a permutation satisfies some predicate in the target domain, we can replace the selected predicate with the new found predicate. For example, given the predicate *companyeconomicsector(company,sector)*, we can generate the following types permuted: *(company,sector)* and *(sector, company)*. The target vocabulary has a predicate with the types corresponding to *(sector, company)*: *economicsectorcompany(sector,company)*. We choose uniformly if we want to maintain the original select predicate (companyeconomicsector) or change for the permutated predicate

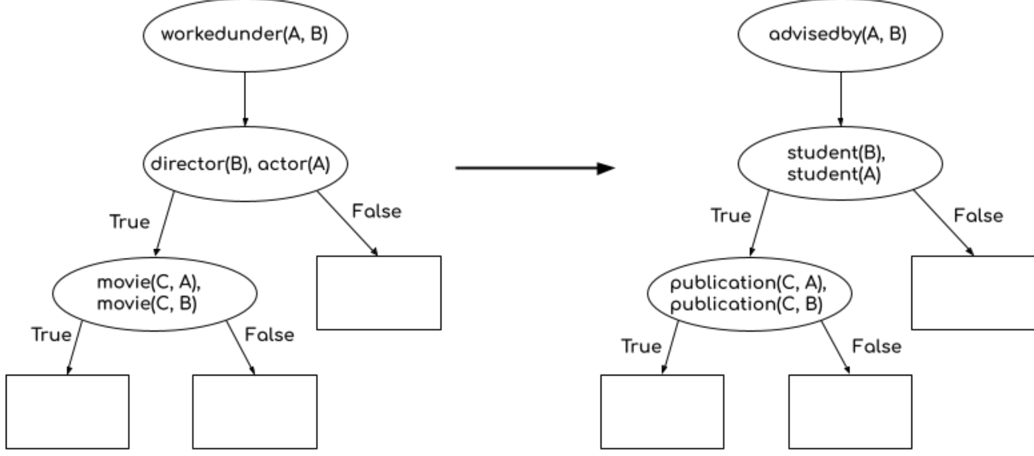

Figure 3.2: Example of transfer learning using GROOT.

(economicsectorcompany).

3.1.1 Space Complexity of the mapping

Assume that $\mathcal{S}$ is the source predicates set, with L_S size while $\mathcal{T}$ is the target predicates set, with size equals to L_T. A predicate $P_S^1 \in \mathcal{S}$ can be mapped to the predicate $P_T^1 \in \mathcal{T}$ and $P_S^2 \in \mathcal{S}$ can also be mapped to P_T^1. Therefore, the worst case for mapping predicates have complexity $O(L_S * L_T)$.

Assume that T_t is the set of types that come together with the arguments' modes of a predicate. Each type $t_t \in T_t$ could be mapped to many source types in the corresponding set T_s of the source BK. However, a source type can only be mapped to one target type to make replacements consistent. Assuming that the size of T_s is N and that T_t has size equals to M, the worst case complexity for the possible mapping are

$$\binom{N}{j} = \frac{N!}{j!(N-j)!} \tag{3.1}$$

where the N source types can choose j target types. The amount of target types that can be mapped does not decrease, given the target type could be mapped to one or more different source types. Therefore, we could map the N source types to one target type, or to two target types and so on, up to N target types, supposing $M \geq N$. In the worst case, we have all the possible combinations of choice for the amount of the target types.

$$\sum_{j=0}^{N} \binom{N}{j} = 2^N \tag{3.2}$$

Algorithm 2 Building the population

```
function POPULATION(n, theory)
    population ← []
    num_individuals ← 0
    while num_individuals ≠ n do
        mapping ← []
        for each predicate ∈ theory do
            select a list with the predicates in target domain that respect the constraints
            if lenght of list = 0 then
                insert null/0 predicate in mapping list
            else
                select one predicate from the list
                insert the chosen target predicate in mapping list
            end if
        end for
        individual ← mapping
        population ← individual
        num_individuals ← num_individuals + 1
    end while
    return population
end function
```

3.2 Evaluation

GROOT evaluates the population using the target dataset, as showed in Algorithm 3. It uses genetic algorithms to minimize the negative value of the AUC PR. Each individual has a mapping between the source and target vocabulary and the structure to make the replacements and revision. The mapping and structure trees are given to RDN-Boost, which trains the model with the training dataset. The test dataset evaluates how the model is performing and returns metrics as AUC PR, AUC ROC, and CLL. The returned AUC PR is given to the individual as a fitness score.

Algorithm 3 Evaluating the population

```
function EVALUATE(population)
    for each individual ∈ population do
        train RDN-Boost with the training set, transferring the mapping from the individual and the structure
        test the model with the test set
        return the AUC PR, AUC ROC and CLL values
        individual.fitness ← -AUC PR
    end for
end function
```

The training set, as mentioned above, is composed of the target dataset. The algorithm used to transfer the predicates using a structure, provided by the source domain, was built in TreeBoostler [11]. The algorithm replaces the source predicates with the mapped target predicate in each node, starting with the first node. After replacing all the predicates, the new model is trained with the training set, generating the regression value in each leaf.

Building and evaluating the population follow the same procedure at each version of the algorithms. They differ on how the genetic operators are developed. Next, we describe the genetic algorithms with their genetic operators versions.

3.3 Simple Genetic Algorithm

When using the simple genetic version, GROOT includes three operators, namely selection, crossover and mutation.

3.3.1 Selection

We apply an elitist strategy, selecting the best individual in the population and copying it to the next generation. We can see how to do the elitist strategy in Algorithm 4.

Algorithm 4 Selecting the best individual in the population.

```
function ELITISM(population)
    select the individual with the smallest negative AUC PR
    if more than one individual has the same value, select the individual with the best CLL
    return best_individual
end function
```

After, we apply the same selection method introduced in Section 2.3.2. The number of individuals participating in each tournament is equal to three and the tournaments are executed with the number of times equal to the population size.

3.3.2 Crossover

After selecting the individuals to form the population, the crossover operator selects randomly two of them to exchange their information. If a random value, between 0 and 1, is smaller or equal the crossover probability p_c, the following steps are taken:

1. A gene is chosen uniformly in each individual.

2. A node is chosen uniformly in the selected gene.
3. The subtree starting at the selected node, including all the following nodes, are swapped between the individuals.
4. The non-selected genes are copied as they are for the new individuals.

Figure 3.3 exemplifies how the crossover method works on the individuals.

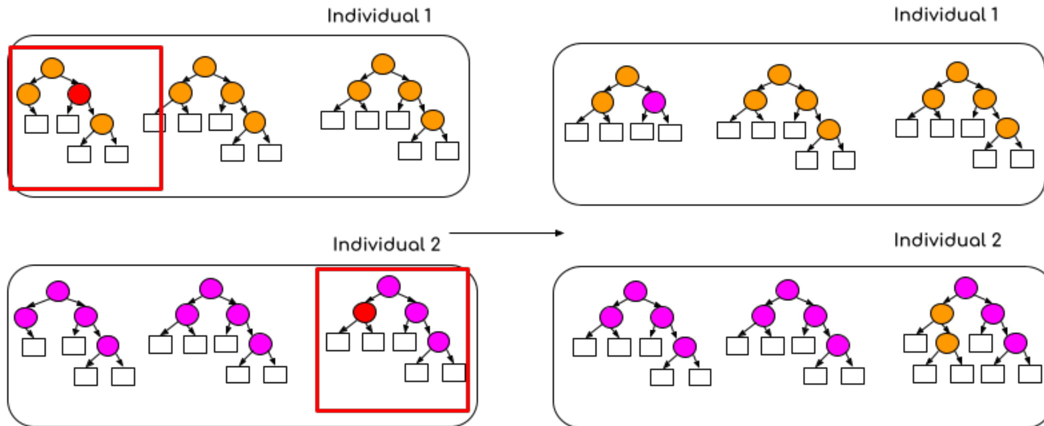

Figure 3.3: Example of crossover between two random individuals. The subtrees are exchanged since the red nodes.

The crossover operator exchanges subtrees instead of trees. This is made to attribute more diversity in the solutions space when creating new structures in the genes and not maintaining the tree's structure always like the same.

In the mutation operator, we also change the structure but adding or erasing a subtree, instead of exchange between individuals.

3.3.3 Mutation

The mutation operator selects an individual at random, similar to the crossover operator. The mutation operator modifies the genes according to two revision operators, pruning or expansion. If a random value is smaller or equals the mutation probability p_m, we choose randomly and uniformly one revision operator to modify the genes. The revision is applied in the nodes level. To verify if a node can be revised, a weighted variance is assigned to the leaf nodes as a result of the covered examples, in the training moment. An example is covered when a path in the hypothesis covers it. If a node has a weighted variance higher than a certain threshold δ, it is considered a revision point and can be pruned or expanded.

If the pruning operator is selected, the following steps are taken:

1. the nodes in the gene are evaluated if they can be a revision point: if the node has two leaves as their children and both variances higher than δ, the node is considered a revision point and composes the set of revision points;

2. one node is selected from the set of revision points;

3. the selected node is erased from the gene.

Figure 3.4: Example of prune in mutation. The red node is a revision point and will be erased from the tree.

Otherwise, if the expansion operator is selected, the following steps are taken:

1. the alleles in the chromosome are evaluated if they can be a revision point: if the node has, at least, one leaf and this leaf has variance higher than δ, the node is considered a revision point;

2. one allele is selected in the set of revision points;

3. a new node is appended in the selected allele, in the branch with a leaf higher than δ. The newly added node has one predicate from the target domain.

As can be seen above, a gene can have many revision points but just one is selected to be revised.

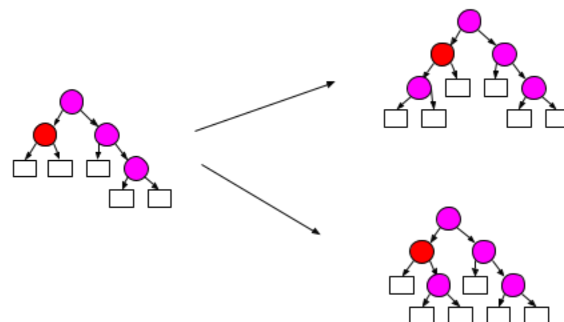

Figure 3.5: Example of expansion in mutation. The red node is a revision point and will receive a new node in one of the leaves.

3.4 BRKGA

The BRKGA method is different from the simple genetic algorithm on how to choose the individuals to make the crossover. The mutation operator will generate new individuals as defined in Section 3.1. In our framework, we built our own implementation of the BRKGA, instead of using the brkgaAPI [45], allowing us to set the variables as tree nodes in place of real values between (0, 1]. Our implementation uses all the remaining components that are presented in the algorithm [14]: the population division, generating the elite individuals; the crossover between one elite individual and the other from the remaining population, and the mutation generating new individuals as in the first generation.

3.4.1 Selection

In BRKGA, the *selection* operator takes the best n_e individuals to compose the elite. The individuals with the best fitness score are chosen. If two individuals have the same fitness value, we use the CLL value as a tie-breaking criterion to rank the individuals.

3.4.2 Crossover

The crossover operator selects two individuals: one from the elite group and another from the remaining population, both selected randomly. The final result is one offspring. Every individual has the same number of genes. We will call the genes from the elite individual as elite gene. Thus, to each gene, the following steps are taken:

1. a random value between 0 and 1 is generated; if this value is smaller or equal to a crossover probability p_c, the node from the elite gene is chosen, otherwise, the node comes from the other chromosome individual is chosen;
2. the chosen nodes compose a new chromosome in the new offspring;
3. if we have an individual with a gene containing more nodes than the other individual, the remaining nodes are appended to the gene in the offspring.

3.5 mBRKGA

In this work, we propose *mBRKGA, a modified Biased-Random Key Genetic Algorithm*. The main idea of the method is to improve the worst individuals, instead of removing them off. In mBRKGA, the population in divided into three buckets:

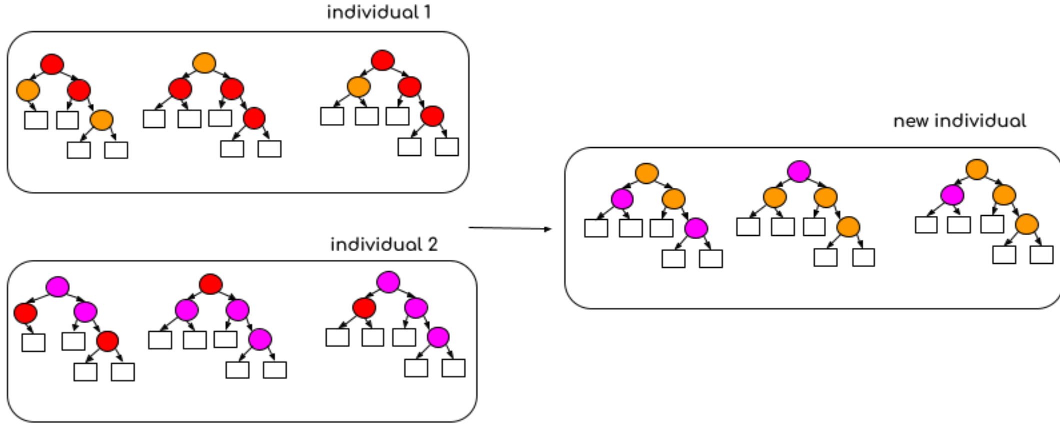

Figure 3.6: Example of crossover in the BRKGA method. The red nodes are chosen to compose the individual.

TOP, **MIDDLE** and **BOTTOM**. The TOP bucket has the individuals with the best fitness values. We selected n_t individuals of the population to be part of the TOP. After, we select the n_b individuals of the remaining population to compose the BOTTOM bucket. In this case, we are selecting the individual with the worst fitness values. The MIDDLE is composed of individuals that are not part of the TOP, neither of the BOTTOM.

Figure 3.7: Example of how the population is divided in the mBRKGA.

3.5.1 Crossover

In the crossover operator, mBRKGA chooses one individual from the TOP and another from the MIDDLE, generating n_c new individuals. This approach aims to generate better individuals, excluding the worst individuals. We believe that when mixing the top and middle individuals we can create a new solution, closer to the best solution.

The method to make the recombination between the individuals is the same used in BRKGA. A randomly selected individual from the top bucket will provide their nodes with a probability p_c. The other individual, chosen from the middle bucket, will contribute with probability $1 - p_c$.

3.5.2 Mutation

In the mutation, instead of creating new individuals from the scratch, we apply the same mutation method used in the Simple Genetic Algorithm in the BOTTOM individuals. As the individual at the bottom did not provide a good solution, the idea is to revise the structure, adding or removing nodes, to improve it.

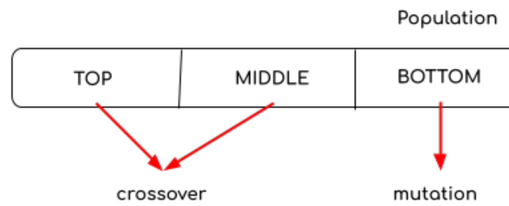

Figure 3.8: Operators in mBRKGA. The Figure shows the TOP and MIDDLE buckets being used in the crossover and the BOTTOM bucket in the mutation.

Chapter 4

Experimental Results

This chapter presents the questions we want to answer with the proposed framework, the applied methodology, and the realized experiments.

4.1 Research questions

The experiments are devised to answer the following research questions:

- **Question 1:** Does GROOT perform better than learning from scratch?
- **Question 2:** Does GROOT perform better than another transfer learning framework?
- **Question 3:** Does GROOT perform better than the baselines with increasing amount of examples in the training target data?
- **Question 4:** Does GROOT reach good results in a viable time?

The **Question 1** focuses on verifying if transferring the structures from the source domain to the target domain achieves better results than learning the model from scratch using the target dataset. To answer **Question 2**, we compare our framework with another transfer learning framework when using the same relational data. To answer the **Question 3**, we vary the amount of available training data to check how GROOT will perform. At last, the **Question 4** addresses how much time GROOT needs to finalize the entire transfer learning process.

4.2 Datasets

We evaluate our framework using the following datasets:

- Cora dataset [46] contains citations to Computer Science research papers, segmented into fields as author, title and venue, for example. The dataset has 1295 distinct citations to 122 papers and the goal is to predict *samevenue*, which shows the relation between two venues.

- Drug-Drug Interaction(DDI) dataset [47] has information about the drugs, with predicates like Enzyme and Transporter. The goal is to predict the *interacts* predicate with the relation between two drugs.

- IMDB dataset [48] contains five mega examples, describing four movies, their directors and the first-billed actors who appear in them. Each director is being ascribed to genres based on the genres of their directed movies, while the gender predicate is used to determine the gender of the actors. A mega example contains a connected group of facts and each one is independent of the other [48]. The main predicate is *workedunder*, indicating if two persons worked together.

- Twitter dataset [42] contains information about the Belgian soccer matches, including the follower accounts, tweeted words and account types relations. The main predicate is *accounttype* defining which type is the account.

- UW-CSE dataset [48] 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.

- WebKB dataset [49, 50] consists of pages and hyperlinks from websites of four Computer Science departments, with categories for each webpage and the words within it. We consider two tasks in this dataset: to predict the *departmentof*, that identifies a person as belonging to a department and the *pageclass* predicate, defining which class a page belongs.

Table 4.1 shows information about the datasets including the amount of types and predicates.

Table 4.1: Statistics about the datasets.

Datasets	Number of types	Number of predicates	Main predicate
Cora	5	10	samevenue
DDI	4	15	interacts
IMDB	3	6	workedunder
Twitter	3	4	accounttype
UW-CSE	9	14	advisedby
WebKB	4	13	departmentof pageclass

4.3 Methodology

We compared our experimental results against the following baselines: RDN-Boost and TreeBoostler. TreeBoostler is another transfer learning framework that also evaluates their results with RDN-Boost. RDN-Boost, or RDN-B, gives models learned from scratch, using the target domain. We also compare our results with RDN-B-1, which differs from RDN-B in trees number. We generate models with ten trees in RDN-B, while in RDN-B-1, we generate a model with a single tree. The RDN-B hyperparameters are the same as defined in TreeBoostler [11].

In each experiment, GROOT receives the structure trees generated from the source data using RDN-B and uses the target domain to evaluate the improvements made with the genetic algorithm. We respect the original division of the datasets' folds. The number of positive and negative examples in each fold is not necessarily the same.

Cross-validation is applied in the experiments: each training set is executed ten times to accommodate randomness. The final results are averaged over the runs. For example, if a dataset has five folds, we ran the experiment 50 times. We compose the training set with one fold and the testing set comprises the remaining folds, following the same methodology used in the transfer learning literature to simulate a limited set of examples [11, 50, 51].

An internal cross-validation procedure is applied in the training set to find the best hyperparameters for the genetic algorithms. The training set is split into three folds, and one fold are selected for validation. To find the best combination of parameters for the genetic algorithms, we rely on the `gp_minize` from Scikit-Optimize package [52] built with Scikit-Learn. The `gp_minize` function uses Bayesian optimization, approximating the desired function to optimize by a Gaussian process. We are evaluating the validation set ten times to get the best parameters. Table 4.2

shows the hyperparameters we provide to the optimization function.

Table 4.2: Hyperparameters used in the optimization functon.

	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]
% of elite individuals = % of top individuals	[0.1, 0.15, 0.2, 0.25]
% of mutations = % of bottom individuals	[0.1, 0.15, 0.2, 0.25, 0.3]

The crossover rate values are chosen based on the literature [35, 53]. The mutation rate values are defined as smaller as possible [35, 53]. However, as we desire to revise the structure, we decided to allow higher values. The number of elite individuals/top individuals and mutations/bottom individuals is expressed into percentages concerning the number of individuals [14]. Thus, if we chose 30 as being the number of individuals and 0.1 as the number of elite individuals, the algorithm will compose the elite set with three individuals. The elite individuals and mutation parameters cover the BRKGA hyperparameters, while the top individuals and bottom individuals correspond to mBRKGA hyperparameters. Also, we defined the threshold to determine if a node is a revision point equal to $\delta = 2.5 \times 10^{-3}$.

When training with the baselines and with GROOT, we subsampled the number of negative examples to be the ratio of two negatives for two positives. Otherwise, when testing, we use all the examples from the testing set [22]. At the last generation, we choose the individual with the best fitness value that will provide the best mapping to generate the final results. The number of generations was defined as 15 for all the experiments. If the best fitness value repeats half of the generation's number plus one, the algorithm stops. We are using the following metrics to compare the performance between GROOT with the baselines: conditional log-likelihood (CLL), the area under the ROC curve (AUC ROC), the area under the PR curve (AUC PR) and training time.

4.4 Results

The experiments are made using different amounts of data from the training set. We vary the positive examples between 10% and 100%. If the percentage of the data offers a set with less than three positive examples, we select three examples. The number of negative examples follows the previous rule in the training set: two negative examples to one positive example. In the testing set, all the examples,

positives or negatives, are used.

In Tables 4.3, 4.5, 4.7, 4.4, 4.8, 4.9 and 4.10 we present the results using 100% of positive examples in the training set, with the results for negative conditional log-likelihood (CLL), area under the Precision-Recall curve (AUC PR), area under the ROC curve (AUC ROC) and the time spent to train, in seconds (s) or minutes (min). The best achieved results are marked in bold. The statistical significance was measured by a paired t-test with $p <= 0.05$. When GROOT has values statistically significant when comparing with TreeBoostler, the $\star$ symbol are beside the value. The $\lozenge$ symbol indicates when GROOT is statistically significant when comparing with RDN-B and $\circ$, when comparing with RDN-B-1.

Table 4.3: 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.

IMDB → UW-CSE

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.194 ± 0.002	0.919 ± 0.009	0.224 ± 0.025	2.061 ± 0.186 s
RDN-B	**-0.299 ± 0.026**	0.928 ± 0.006	0.238 ± 0.017	6.056 ± 0.636 s
TreeBooostler	-0.283 ± 0.014	0.931 ± 0.003	0.237 ± 0.016	6.607 ± 0.515 s
GROOT - Genetic	-0.278 ± 0.035 $\circ$	**0.940 ± 0.008**	**0.337 ± 0.049** $\star\lozenge\circ$	12.492 ± 3.348 min
GROOT - BRKGA	-0.280 ± 0.035 $\circ$	0.935 ± 0.005	0.332 ± 0.040 $\star\lozenge\circ$	5.370 ± 0.404 min
GROOT - mBRKGA	-0.279 ± 0.030 $\circ$	0.935 ± 0.006	0.327 ± 0.050 $\star\lozenge\circ$	5.997 ± 1.725 min

Table 4.4: Results for the experiment with WebKB and Twitter 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.

WebKB (pageclass) → Twitter

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.196 ± 0.001	0.500 ± 0.000	0.007 ± 0.000	17.491 ± 12.836 s
RDN-B	**-0.396 ± 0.010**	0.500 ± 0.000	0.007 ± 0.000	132.693 ± 55.403 s
TreeBooostler	-0.253 ± 0.011	0.933 ± 0.000	0.062 ± 0.000	27.567 ± 4.986 s
GROOT - Genetic	-0.171 ± 0.058	**0.992 ± 0.006** $\lozenge\circ$	**0.411 ± 0.259** $\star\lozenge\circ$	32.236 ± 9.362 min
GROOT - BRKGA	-0.248 ± 0.100	0.933 ± 0.028 $\lozenge\circ$	0.062 ± 0.029	14.518 ± 0.932 min
GROOT - mBRKGA	-0.207 ± 0.083	0.975 ± 0.028 $\lozenge\circ$	0.304 ± 0.208 $\lozenge\circ$	31.667 ± 7.170 min

We achieved better results for AUC ROC and AUC PR metrics in Tables 4.3 and 4.4 and got competitive results for the CLL value, when comparing with RDN-B-1, RDN-B and TreeBoostler. This occurs because GROOT can map the same source predicate to many different target predicates, different from TreeBoostler, which

maps one source predicate to only one target predicate. Nevertheless, in Table 4.5, GROOT did not provide a good result in any of the metrics. This is because the source structure trees learned from IMDB have predicates with an arity equal to one, but the Cora domain has only predicates with arity equal to two. So, when mapping between the predicates, GROOT could not deal with this problem and replaced the source predicate with a null predicate, indicating an absence of mapping.

Table 4.5: 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.

IMDB → Cora

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	**-0.765 ± 0.012**	0.613 ± 0.042	0.479 ± 0.040	24.704 ± 7.612 s
RDN-B	-0.610 ± 0.026	**0.636 ± 0.031**	0.502 ± 0.027	196.219 ± 60.208 s
TreeBooostler	-0.624 ± 0.001	0.612 ± 0.005	**0.551 ± 0.007**	71.174 ± 8.027 s
GROOT - Genetic	-0.323 ± 0.014 ⋆◊∘	0.582 ± 0.004 ⋆	0.183 ± 0.008 ⋆◊∘	26.588 ± 1.110 min
GROOT - BRKGA	-0.323 ± 0.014 ⋆◊∘	0.582 ± 0.004 ⋆	0.183 ± 0.008 ⋆◊∘	21.656 ± 1.236 min
GROOT - mBRKGA	-0.324 ± 0.014 ⋆◊∘	0.582 ± 0.004 ⋆	0.183 ± 0.008 ⋆◊∘	18.333 ± 0.889 min

In Tables 4.7, 4.8, 4.9 and 4.10, GROOT improved the CLL metric only. The conditional log-likelihood is calculated as the average of the conditional log-likelihood to each example. The conditional likelihood is defined as the probability of a label being positive given the example[1]. For example, the conditional likelihood of a positive example gives the probability of the label being defined as positive given the positive example. For negative examples, we use the complementary probability. The conditional likelihood of negative examples gives the probability of the label being negative given the negative example. Here, we want the likelihood of the label being positive unconcerned with the example. Taking out the log of the likelihood, we got negative values because the probabilities are less than one and the log of values between zero and one are negatives. Furthermore, closer to zero the value, the smaller the returned log. If a probability is equal to 0.5, their log will be equal to -0.6931; on the other hand, if a probability has a value equal to 0.35, the log will be -1.0498. In Table 4.6, we show the average of the probabilities of all examples in the experiments.

[1] https://github.com/starling-lab/BoostSRL/blob/development/src/edu/wisc/cs/will/DataSetUtils/ComputeAUC.java

Table 4.6: The mean of the probabilities to predict the label equals the example. The last column has the probability of a negative example being classified as positive.

Experiments		Mean of positive examples probability (P_p)	Mean of negative examples probability (P_n)	Mean of negative examples probability being positive($1 - P_n$)
IMDB → UW-CSE	Genetic	0.602	0.785	0.215
	BRKGA	0.589	0.783	0.217
	mBRKGA	0.599	0.785	0.215
WebKB (pageclass) → Twitter	Genetic	0.782	0.866	0.134
	BRKGA	0.740	0.837	0.163
	mBRKGA	0.763	0.851	0.149
IMDB → Cora	Genetic	0.369	0.731	0.269
	BRKGA	0.368	0.732	0.268
	mBRKGA	0.369	0.731	0.269
Cora → IMDB	Genetic	0.682	0.857	0.143
	BRKGA	0.698	0.866	0.134
	mBRKGA	0.701	0.866	0.134
WebKB (departmentof) → IMDB	Genetic	0.768	0.857	0.143
	BRKGA	0.760	0.855	0.145
	mBRKGA	0.794	0.868	0.132
WebKB (departmentof) → UW-CSE	Genetic	0.642	0.778	0.222
	BRKGA	0.627	0.775	0.225
	mBRKGA	0.636	0.775	0.225
DDI → IMDB	Genetic	0.521	0.785	0.215
	BRKGA	0.430	0.751	0.249
	mBRKGA	0.438	0.757	0.243

To have a negative CLL, we need to have big likelihoods in defining the labels of the examples. The AUC and ROC curves use the correctly classified positive examples to compute their values and ignore the correctly classified negative examples [54]. As we can see in the tables, GROOT did not achieve good results for the curves but for the CLL metric, so the framework can identify negative examples with more confidence, contributing to the smaller CLL.

Table 4.7: Results for the experiment with Cora and IMDB 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	-0.167 ± 0.003	0.991 ± 0.010	0.794 ± 0.121	1.820 ± 0.019 s
RDN-B	-0.095 ± 0.032	0.992 ± 0.010	0.808 ± 0.118	3.458 ± 0.098 s
TreeBooostler	-0.077 ± 0.003	**0.997 ± 0.001**	**0.861 ± 0.033**	7.181 ± 0.101 s
GROOT - Genetic	**-0.204 ± 0.022** ⋆◊∘	0.960 ± 0.017 ⋆◊∘	0.281 ± 0.021 ⋆◊∘	25.214 ± 6.272 min
GROOT - BRKGA	-0.204 ± 0.023 ⋆◊∘	0.900 ± 0.029 ⋆◊∘	0.246 ± 0.025 ⋆◊∘	10.500 ± 4.189 min
GROOT - mBRKGA	-0.204 ± 0.025 ⋆◊∘	0.901 ± 0.029 ⋆◊∘	0.258 ± 0.086 ⋆◊∘	11.155 ± 4.264 min

Table 4.8: Results for the experiment with WebKB and IMDB 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.

WebKB (departmentof) → IMDB

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.167 ± 0.002	0.992 ± 0.005	0.774 ± 0.082	1.780 ± 0.025 s
RDN-B	-0.093 ± 0.018	0.994 ± 0.004	0.803 ± 0.062	3.356 ± 0.077 s
TreeBooostler	-0.093 ± 0.017	**0.995 ± 0.004**	**0.817 ± 0.076**	4.925 ± 0.095 s
GROOT - Genetic	-0.217 ± 0.046 ⋆♢	0.962 ± 0.011 ⋆♢∘	0.268 ± 0.034 ⋆♢∘	19.038 ± 6.357 min
GROOT - BRKGA	**-0.222 ± 0.049** ⋆♢	0.950 ± 0.013 ⋆♢∘	0.218 ± 0.032 ⋆♢∘	8.147 ± 0.389 min
GROOT - mBRKGA	-0.189 ± 0.052 ⋆♢	0.980 ± 0.024	0.646 ± 0.345	13.808 ± 5.367 min

Table 4.9: Results for the experiment with WebKB 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.

WebKB (departmentof) → UW-CSE

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.195 ± 0.001	0.916 ± 0.007	0.216 ± 0.039	1.952 ± 0.089 s
RDN-B	-0.309 ± 0.023	**0.925 ± 0.004**	0.228 ± 0.014	5.766 ± 0.477 s
TreeBooostler	-0.316 ± 0.023	0.922 ± 0.005	**0.243 ± 0.030**	6.286 ± 0.220 s
GROOT - Genetic	-0.328 ± 0.055 ∘	0.883 ± 0.023 ⋆♢∘	0.113 ± 0.025 ⋆♢∘	11.620 ± 3.877 min
GROOT - BRKGA	**-0.335 ± 0.059** ∘	0.858 ± 0.002 ⋆♢∘	0.091 ± 0.011 ⋆♢∘	5.748 ± 0.081 min
GROOT - mBRKGA	-0.333 ± 0.063 ∘	0.872 ± 0.024 ⋆♢∘	0.112 ± 0.055 ⋆♢∘	9.546 ± 8.147 min

Table 4.10: Results for the experiment with DDI and IMDB 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.

DDI → IMDB

	CLL	AUC ROC	AUC PR	Time
RDN-B-1	-0.168 ± 0.004	0.988 ± 0.010	0.756 ± 0.123	1.802 ± 0.028 s
RDN-B	-0.110 ± 0.037	0.988 ± 0.010	0.756 ± 0.123	3.378 ± 0.061 s
TreeBooostler	-0.082 ± 0.007	**0.998 ± 0.001**	**0.909 ± 0.027**	6.796 ± 0.116 s
GROOT - Genetic	-0.262 ± 0.031 ⋆♢∘	0.969 ± 0.051 ⋆	0.715 ± 0.363	13.595 ± 4.506 min
GROOT - BRKGA	**-0.313 ± 0.008** ⋆♢∘	0.868 ± 0.003 ⋆♢∘	0.092 ± 0.014 ⋆♢∘	10.146 ± 2.802 min
GROOT - mBRKGA	-0.305 ± 0.022 ⋆♢∘	0.882 ± 0.039 ⋆♢∘	0.179 ± 0.259 ⋆♢∘	14.042 ± 10.354 min

Looking at the tables, we can conclude that GROOT obtains an improvement for at least one metric in most of the experiments and achieves comparable results

when confronting all baselines, answering positively the **Question 1** and **Question 2**. The size of the population can also interfere with the results. As we can have many feasible solutions, as seen in the complexity calculation, small populations will premature convergence, returning substandard solutions [34]. In the parameters for the algorithms, we enable to have a small population size, which could contribute to have improvement in just one metric and not in all.

In all experiments, GROOT needed a long time to finish the execution. This occurs because, every time an individual makes changes in their mapping, it is necessary to evaluate again the model. The training and testing procedures take seconds but how the framework needs to evaluate many individuals, the final time in each generation is high. So, the **Question 4** is answered negatively.

In order to compare the performance of our method with increasing amounts of positive target examples in the training set, we performed an experiment transferring the same pairs of datasets. Figures 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7 demonstrate the experiments. The figures show the AUC PR (upper left), AUC ROC (upper right) and CLL (center) values. For the PR and ROC curves, the higher the value, the better. For the CLL metric, the lower the value, the better. As we can see, GROOT outperforms the baselines in every amount of data, when comparing the AUC PR and AUC ROC, in the experiments with IMDB $\rightarrow$ UW-CSE (Figure 4.1) and WebKB $\rightarrow$ Twitter (Figure 4.2). In the others experiments, except for IMDB $\rightarrow$ Cora, we got good results in CLL, as seen in the tables above. With the results in the figures, we can also answer positively the **Question 3**, because GROOT improves, at least, one metric in most of the experiments.

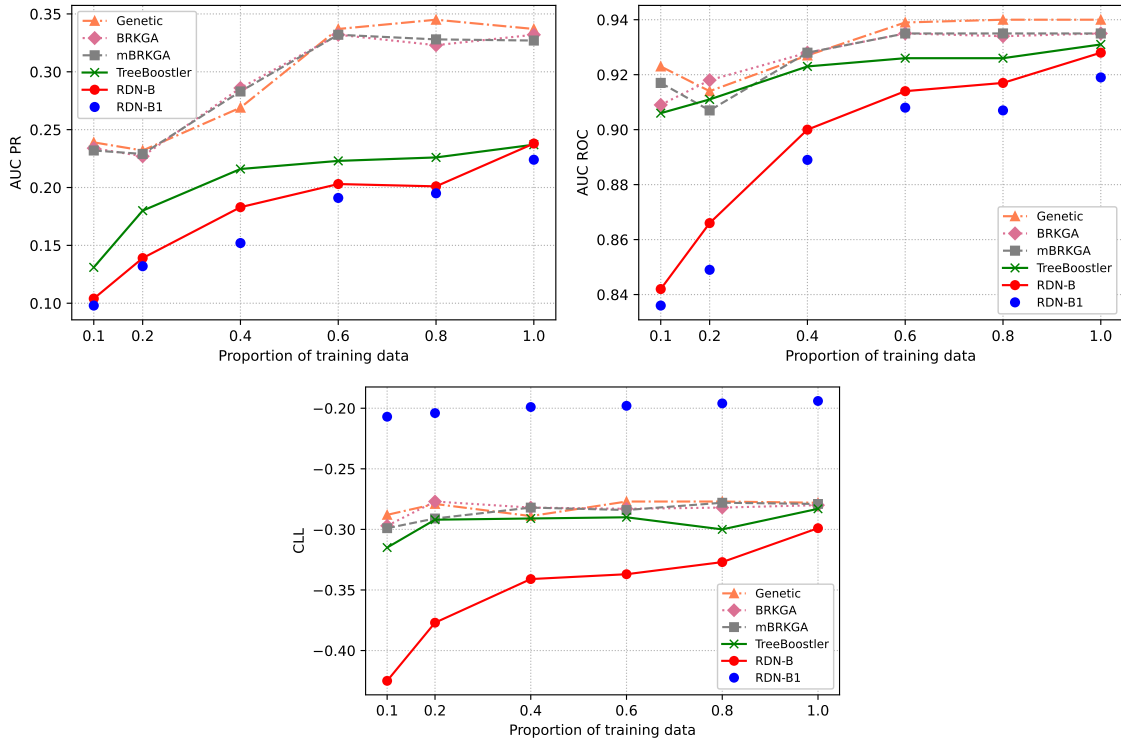

Figure 4.1: Learning curves for AUC ROC and AUC PR obtained from IMDB → UW-CSE using one fold to train.

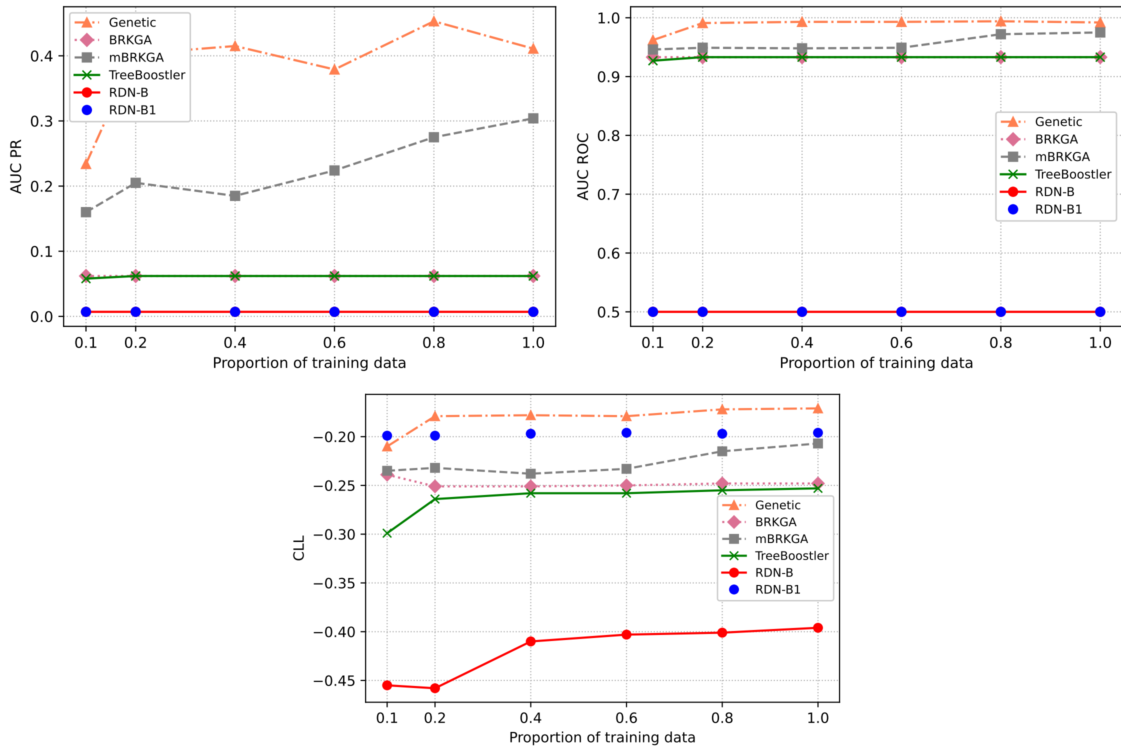

Figure 4.2: Learning curves for AUC ROC and AUC PR obtained from WebKB (pageclass) → Twitter using one fold to train.

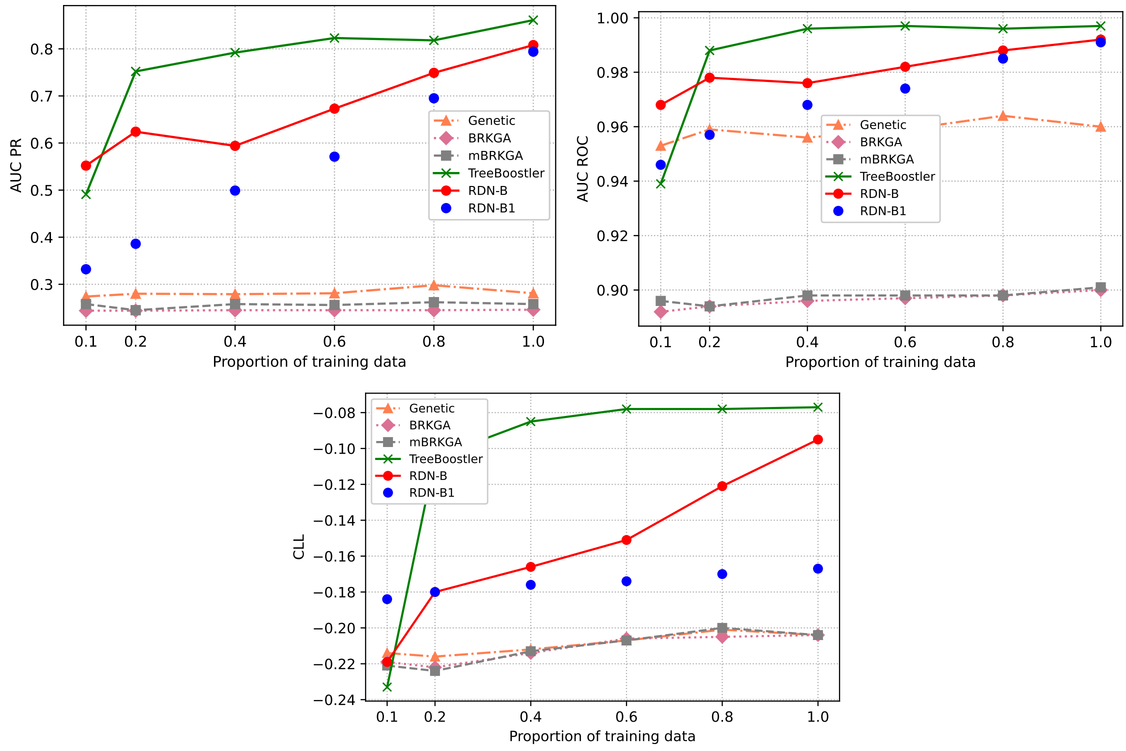

Figure 4.3: Learning curves for AUC ROC and AUC PR obtained from Cora → IMDB using one fold to train.

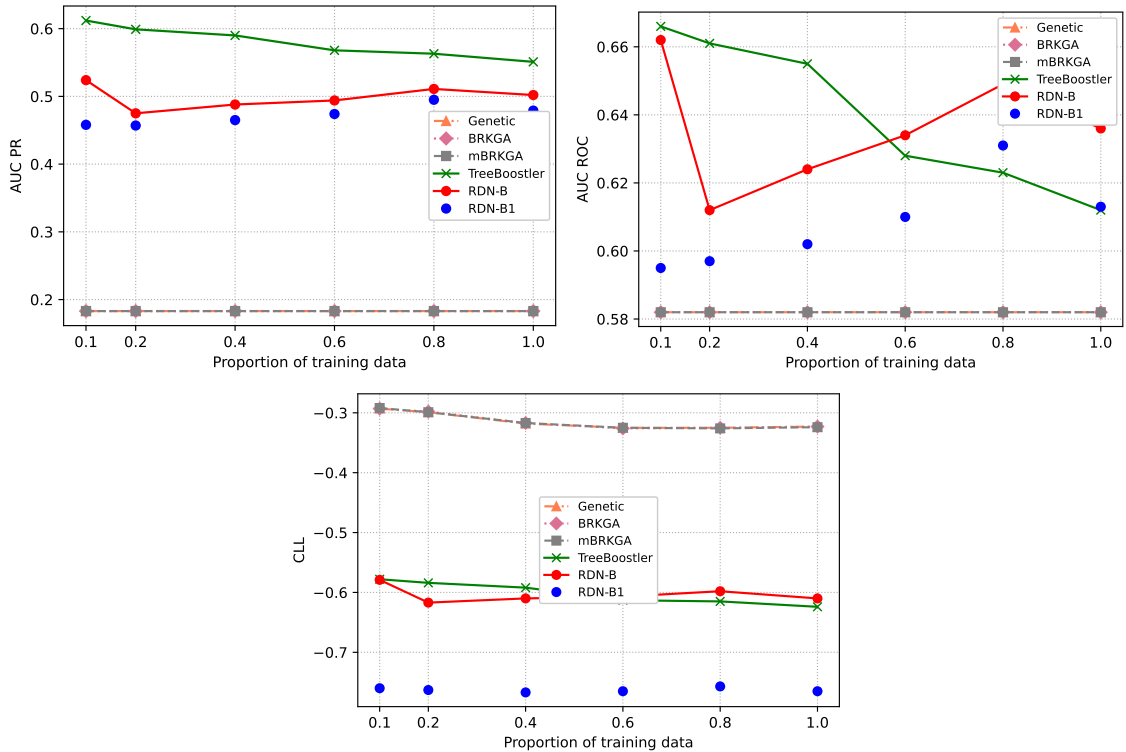

Figure 4.4: Learning curves for AUC ROC and AUC PR obtained from IMDB → Cora using one fold to train.

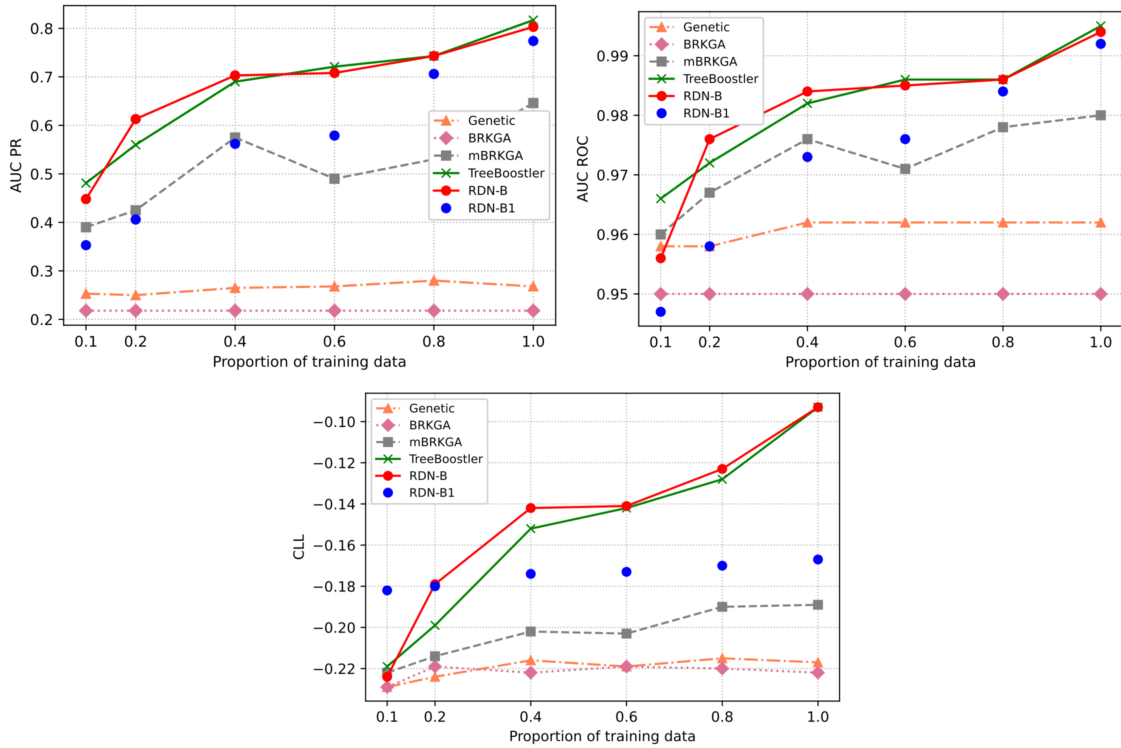

Figure 4.5: Learning curves for AUC ROC and AUC PR obtained from WebKB (departmentof) → IMDB using one fold to train.

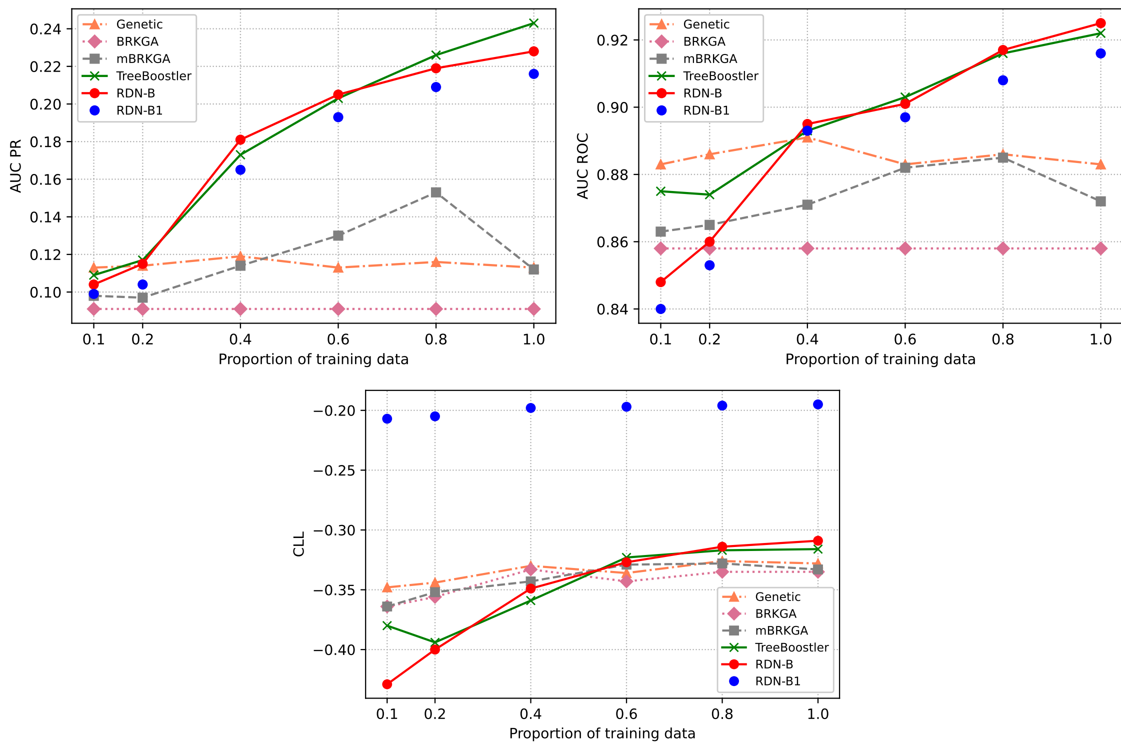

Figure 4.6: Learning curves for AUC ROC and AUC PR obtained from WebKB (departmentof) → UW-CSE using one fold to train.

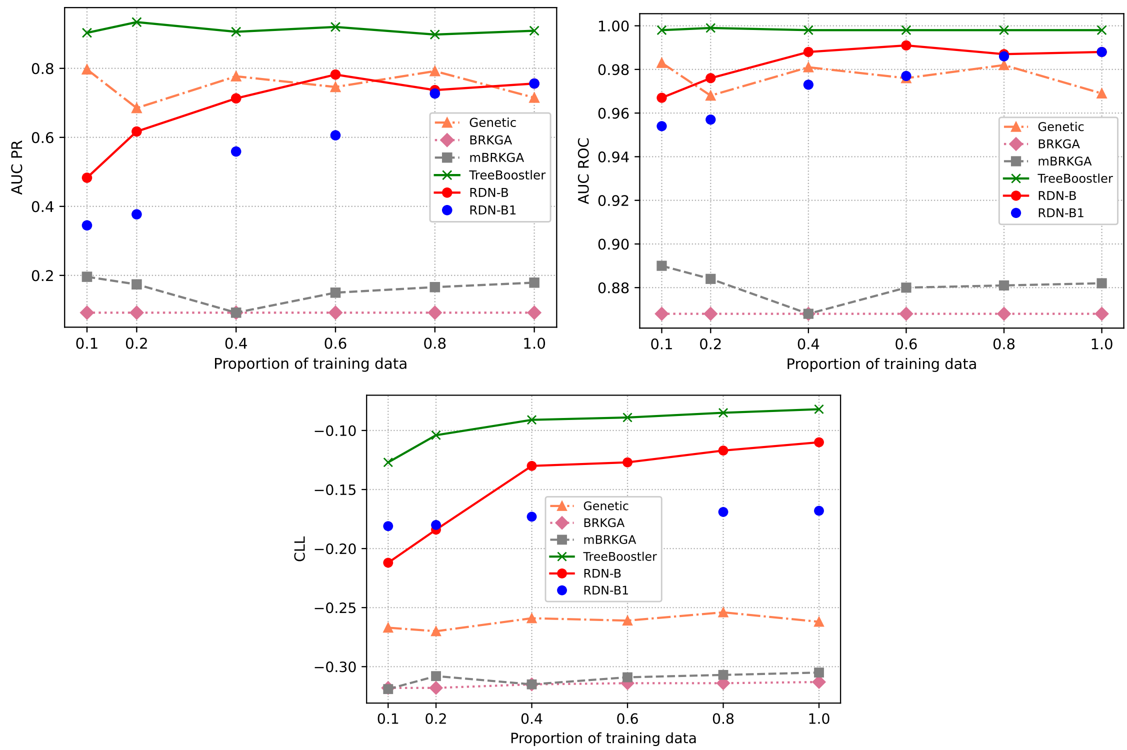

Figure 4.7: Learning curves for AUC ROC and AUC PR obtained from DDI → IMDB using one fold to train.

Chapter 5

Conclusion

To address the transfer learning problem with relational domains, this dissertation presented a framework called GROOT, aiming at the transfer using genetic algorithms to find a mapping between predicates from source and target datasets. The framework receives a structure generated by the RDN-Boost method using the source domain and uses it as a starting point to find the best mapping. Along with generations, each individual carries a mapping to transfer, exchange information with other individuals, and revise the structure tree to improve the performance. Further to the simple genetic algorithm, we employed a genetic algorithm variation in the framework. The Biased Random-Key genetic algorithm (BRKGA) was used in the experiments, with our own implementation of the algorithm. The difference between the two methods is BRKGA uses a biased crossover, selecting one individual from the best ones in the population and another from the remaining population.

We also propose a new method called modified Biased Random-Key genetic algorithm (mBRKGA), where we apply the mutation in the worst individuals and make the crossover with the best individuals and reasonable individuals. Furthermore, in this dissertation, we calculated the space complexity according to the proposed mapping between the predicates. As GROOT can map one source predicate to many target predicates, the search space increase if the number of predicates increases.

The experimental results presented an improvement, when compared with the baselines, in the value of the metrics. In most of the experiments, GROOT improves the CLL values, even with the methods optimizing the AUC PR metric. We showed that the framework reaches better or competitive results when compared with another transfer learning framework. This is possible because of the large search space. However, those enhancements are generally followed with a longer runtime than the baselines.

5.1 Future work

A possible future work is to improve the evaluation component. The main bottleneck in our work is the time to evaluate the individuals. In every generation, we have many individuals that change their mappings. So, the methods need to train and test again the new model. This demands a long time even evaluating in parallel. A solution for this problem is to find new ways to make the inference procedure, making it possible to use the old model to evaluate the test dataset with the new changes. An another solution to improve the long running time is to use the brkgaAPI [45], where is a well-consolided implementation of the BRKGA algorithm. In this solution, we can increase the population size, since this size should not interfere in the throughput of the algorithm in the most of the situations.

Another improvement concerns the tree revision. In GROOT, the revision is made in the genes, at most, once, per generation. After finding the best mapping and generating the model, a revision could be done in the resulting model, using a metaheuristic search from the revision points.

References

[1] DE RAEDT, L. *Logical and Relational Learning*. Springer Publishing Company, Incorporated, 2008. ISBN: 978-3-540-20040-6.

[2] 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, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN: 3642130585.

[3] RICHARDSON, M., DOMINGOS, P. M. "Markov logic networks", *Mach. Learn.*, v. 62, n. 1-2, pp. 107–136, 2006.

[4] NATARAJAN, S., KHOT, T., KERSTING, K., et al. "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. Disponível em: <https://lirias.kuleuven.be/handle/123456789/283041>.

[5] PAN, S. J., YANG, Q. "A Survey on Transfer Learning", *IEEE Trans. on Knowl. and Data Eng.*, v. 22, n. 10, pp. 1345–1359, out. 2010.

[6] WEISS, K., KHOSHGOFTAAR, T. M., WANG, D. "A survey of transfer learning", *Journal of Big Data*, v. 3, n. 1, pp. 9, 2016. ISSN: 2196-1115. doi: 10.1186/s40537-016-0043-6. Disponível em: <https://doi.org/10.1186/s40537-016-0043-6>.

[7] ZHUANG, F., QI, Z., DUAN, K., et al. "A comprehensive survey on transfer learning", *Proceedings of the IEEE*, v. 109, n. 1, pp. 43–76, 2020.

[8] OLIVAS, E. S., GUERRERO, J. D. M., SOBER, M. M., et al. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Hershey, PA, Information Science Reference - Imprint of: IGI Publishing, 2009.

[9] ROBERTS, S. "An introduction to Progol", *Department of Computer Science, University of York*, v. 244, 1997.

[10] SRINIVASAN, A. "The aleph manual". 2001.

[11] AZEVEDO SANTOS, R., PAES, A., ZAVERUCHA, G. "Transfer learning by mapping and revising boosted relational dependency networks", *Machine Learning*, v. 109, n. 7, pp. 1435–1463, 2020.

[12] LUCA, T., PAES, A., ZAVERUCHA, G. "Mapping Across Relational Domains for Transfer Learning with Word Embeddings-based Similarity". In: *International Conference on Inductive Logic Programming, International Joint Conference on Learning & Reasoning*. Springer, 2021.

[13] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st ed. USA, Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.

[14] GONÇALVES, J. F., RESENDE, M. G. C. "Biased random-key genetic algorithms for combinatorial optimization", *J. Heuristics*, v. 17, n. 5, pp. 487–525, 2011. doi: 10.1007/s10732-010-9143-1. Disponível em: <https://doi.org/10.1007/s10732-010-9143-1>.

[15] FIGUEIREDO, L., PAES, A., ZAVERUCHA, G. "Transfer Learning for Boosted Relational Dependency Networks Through Genetic Algorithm". In: *International Conference on Inductive Logic Programming, International Joint Conference on Learning & Reasoning*. Springer, 2021.

[16] KERSTING, K. *An Inductive Logic Programming Approach to Statistical Relational Learning*, v. 148. IOS Press, 2006.

[17] RICHARDS, B. L., MOONEY, R. J. "Automated Refinement of First-Order Horn-Clause Domain Theories", *Mach. Learn.*, v. 19, n. 2, pp. 95–131, 1995. doi: 10.1007/BF01007461. Disponível em: <https://doi.org/10.1007/BF01007461>.

[18] DUBOC, A. L., PAES, A., ZAVERUCHA, G. "Using the bottom clause and mode declarations in FOL theory revision from examples", *Mach. Learn.*, v. 76, n. 1, pp. 73–107, 2009. doi: 10.1007/s10994-009-5116-8. Disponível em: <https://doi.org/10.1007/s10994-009-5116-8>.

[19] PAES, A., ZAVERUCHA, G., COSTA, V. S. "On the use of stochastic local search techniques to revise first-order logic theories from examples", *Mach. Learn.*, v. 106, n. 2, pp. 197–241, 2017. doi: 10.1007/s10994-016-5595-3. Disponível em: <https://doi.org/10.1007/s10994-016-5595-3>.

[20] KOLLER, D., FRIEDMAN, N., DŽEROSKI, S., et al. *Introduction to statistical relational learning*. MIT press, 2007.

[21] NEVILLE, J., JENSEN, D. "Relational Dependency Networks", *J. Mach. Learn. Res.*, v. 8, pp. 653–692, maio 2007. ISSN: 1532-4435.

[22] NATARAJAN, S., KHOT, T., KERSTING, K., et al. "Gradient-based boosting for statistical relational learning: The relational dependency network case", *Mach. Learn.*, v. 86, n. 1, pp. 25–56, 2012.

[23] BLOCKEEL, H. "Top-Down Induction of First Order Logical Decision Trees", *AI Commun.*, v. 12, n. 1-2, pp. 119–120, 1999. Disponível em: <http://content.iospress.com/articles/ai-communications/aic178>.

[24] ZHU, Y., CHEN, Y., LU, Z., et al. "Heterogeneous transfer learning for image classification". In: *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[25] HUSSAIN, M., BIRD, J. J., FARIA, D. R. "A study on cnn transfer learning for image classification". In: *UK Workshop on computational Intelligence*, pp. 191–202. Springer, 2018.

[26] DAI, W., XUE, G.-R., YANG, Q., et al. "Co-Clustering Based Classification for out-of-Domain Documents". KDD '07, p. 210–219, New York, NY, USA, 2007. ACM. ISBN: 9781595936097.

[27] GUPTA, R., SAHU, S., ESPY-WILSON, C., et al. "Semi-supervised and transfer learning approaches for low resource sentiment classification". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5109–5113. IEEE, 2018.

[28] KAYA, M., FIDAN, G., TOROSLU, I. H. "Transfer learning using Twitter data for improving sentiment classification of Turkish political news". In: *Information sciences and systems 2013*, Springer, pp. 139–148, 2013.

[29] BATAA, E., WU, J. "An investigation of transfer learning-based sentiment analysis in japanese", *arXiv preprint arXiv:1905.09642*, 2019.

[30] MA, Y., LUO, G., ZENG, X., et al. "Transfer learning for cross-company software defect prediction", *Information and Software Technology*, v. 54, n. 3, pp. 248–256, 2012.

[31] CHEN, J., YANG, Y., HU, K., et al. "Multiview transfer learning for software defect prediction", *IEEE Access*, v. 7, pp. 8901–8916, 2019.

[32] GANDOMI, A., YANG, X.-S., TALATAHARI, S., et al. "Metaheuristic Algorithms in Modeling and Optimization". pp. 1–24, 12 2013. ISBN: 9780123983640.

[33] MITCHELL, T. M. *Machine Learning*. New York, McGraw-Hill, 1997.

[34] BURKE, E. K., KENDALL, G. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer Publishing Company, Incorporated, 2013. ISBN: 1461469392.

[35] FOGEL, D. B. "An introduction to simulated evolutionary optimization", *IEEE Trans. Neural Networks*, v. 5, n. 1, pp. 3–14, 1994.

[36] BEAN, J. C. "Genetic algorithms and random keys for sequencing and optimization", *ORSA journal on computing*, v. 6, n. 2, pp. 154–160, 1994.

[37] GONÇALVES, J. F., RESENDE, M. G. "A biased random-key genetic algorithm for the unequal area facility layout problem", *European Journal of Operational Research*, v. 246, n. 1, pp. 86–107, 2015.

[38] GONÇALVES, J. F., RESENDE, M. G. "A biased random key genetic algorithm for 2D and 3D bin packing problems", *International Journal of Production Economics*, v. 145, n. 2, pp. 500–510, 2013.

[39] NORONHA, T. F., RESENDE, M. G., RIBEIRO, C. C. "A biased random-key genetic algorithm for routing and wavelength assignment", *Journal of Global Optimization*, v. 50, n. 3, pp. 503–518, 2011.

[40] GONÇALVES, J. F., DE MAGALHÃES MENDES, J. J., RESENDE, M. G. "A hybrid genetic algorithm for the job shop scheduling problem", *European journal of operational research*, v. 167, n. 1, pp. 77–95, 2005.

[41] 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. ISBN: 9781577353232.

[42] HAAREN, J. V., KOLOBOV, A., DAVIS, J. "TODTLER: Two-Order-Deep Transfer Learning". In: *Proc. of the 29th AAAI Conference on Artificial Intelligence*, AAAI'15, p. 3007–3015. AAAI Press, 2015.

[43] MUGGLETON, S., TAMADDONI-NEZHAD, A. "QG/GA: a stochastic search for Progol", *Mach. Learn.*, v. 70, n. 2-3, pp. 121–133, 2008.

[44] 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*, v. 7207, *Lecture Notes in Computer Science*, pp. 286–301. Springer, 2011.

[45] TOSO, R. F., RESENDE, M. G. C. "A C++application programming interface for biased random-key genetic algorithms", *Optim. Methods Softw.*, v. 30, n. 1, pp. 81–93, 2015. doi: 10.1080/10556788.2014.890197. Disponível em: <https://doi.org/10.1080/10556788.2014.890197>.

[46] BILENKO, M., MOONEY, R. J. "Adaptive duplicate detection using learnable string similarity measures". In: Getoor, L., Senator, T. E., Domingos, P. M., et al. (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.

[47] KAKADIYA, A., NATARAJAN, S., RAVINDRAN, B. "Relational Boosted Bandits", *CoRR*, v. abs/2012.09220, 2020. Disponível em: <https://arxiv.org/abs/2012.09220>.

[48] 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)*, v. 227, *ACM International Conference Proceeding Series*, pp. 625–632. ACM, 2007.

[49] CRAVEN, M., SLATTERY, S. "Relational learning with statistical predicate invention: Better models for hypertext", *Machine Learning*, v. 43, n. 1, pp. 97–119, 2001.

[50] KUMARASWAMY, R., ODOM, P., KERSTING, K., et al. "Transfer Learning via Relational Type Matching". In: Aggarwal, C. C., Zhou, Z., Tuzhilin, A., et al. (Eds.), *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pp. 811–816. IEEE, 2015.

[51] 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*, v. 382, *ACM International Conference Proceeding Series*, pp. 217–224. ACM, 2009.

[52] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., et al. "Scikit-learn: Machine Learning in Python", *J. Mach. Learn. Res.*, v. 12, pp. 2825–2830, 2011.

[53] BÄCK, T., HAMMEL, U., SCHWEFEL, H. "Evolutionary computation: comments on the history and current state", *IEEE Trans. Evol. Comput.*, v. 1, n. 1, pp. 3–17, 1997.

[54] DAVIS, J., GOADRICH, M. "The relationship between Precision-Recall and ROC curves". In: Cohen, W. W., Moore, A. W. (Eds.), *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, v. 148, *ACM International Conference Proceeding Series*, pp. 233–240. ACM, 2006. doi: 10.1145/1143844.1143874. Disponível em: <https://doi.org/10.1145/1143844.1143874>.