A Different Approach for Clique and Household Analysis in Synthetic Telecom Data Using Propositional Logic

Sandro Skansi*, Kristina Šekrst ** and Marko Kardum***
* Faculty of Croatian Studies, Zagreb, Croatia
** Faculty of Humanities and Social Sciences, Zagreb, Croatia
*** Faculty of Croatian Studies, Zagreb, Croatia
sskansi@hrstud.hr

Summary – In this paper we propose an non-machine learning artificial intelligence (AI) based approach for telecom data analysis, with a special focus on clique detection. Clique detection can be used to identify households, which is a major challenge in telecom data analysis and predictive analytics. Our approach does not use any form of machine learning, but another type of algorithm: satisfiability for propositional logic. This is a neglected approach in modern AI, and we aim to demonstrate that for certain tasks, it may be a good alternative to machine learning-based approaches. We have used a simple DPLL satisfiability solver over an artificially generated telecom dataset (due to GDPR regulations), but our approach can be implemented on any telecom data by following the SAT encoding we have developed, and the DPLL solver can be substituted by a more advanced alternative such as CDCL. This paper extends the method presented in [1] for banking logs to data containing caller information, and proposes a more efficient encoding.

Keywords – Clique Detection; SAT Solving; DPLL; Household Identification; SAT encodings; Telecom Data Analysis

I. INTRODUCTION

The problem of clique detection (or informally, in the case of telecom data, reciprocal callers) is considered a rare event, unsuitable for machine-learning algorithms. Rare-event problems usually deal with unbalanced datasets, while most of machine-learning algorithms presuppose balanced classes in predictive classification. An example of using a non-machine-learning algorithm for rare events is [2], which relies on genetic algorithms, based on population-based optimization. Even though modern deep-learning methods hardly ever consider such occurrences, rare events are often interesting events pointing to unusual or peculiar behavior, or are able to give us insight into previously unknown relations. For example, detecting reciprocal payers in [1] may lead to fraud detection or uncovering money-laundering schemes.

The clique problem is one of well-known computational and optimization problems of finding complete subgraphs or cliques: subsets of graph vertices such that every two distinct vertices in it are adjacent. Clique detection – the task of finding such a clique – is NP-complete (NP stands for nondeterministic polynomial time), i.e. its validity can be quickly tested in polynomial time, but there is no known way to find a solution quickly (see [3] for more insight). If posed as a non-decision problem, clique detection is NP-hard, and due to it, exact solutions also exhibit exponential complexity. In order to solve NP-complete problems, we are left with brute-force search methods that check if every possible candidate in the list satisfies the given statement. Since such methods are unusable in large datasets with an exponential rise of computational complexity, various heuristic and approximation methods are often used for coarse-grained results. Because finding a solution to one NP-complete problem guarantees a solution to another of the same class, we aim to use a non-machine-learning AI algorithm based on Boolean satisfiability that performs better in changeable dynamic datasets.

In real-world settings, social networks are commonly represented as graphs, where vertices stand for people, the edges represent their acquaintances, and a clique is a subset of mutual friends, i.e. people that know each other. Such networks may be used to analyze business relationships or family relations, in order to improve sales and marketing techniques. In this paper we are focusing on cliques as household members, and our analysis can show that a detected clique may be really connected or just formally connected. We differentiate between *functional* and *official households*. Functional households comprise household members that actually live together, while official households are seen as households only on paper, for a variety of purposes, such as studying or working abroad, or even tax evasion.

Recently, the concept of a household has been of great importance in business and finance. For example, Guiso and Sodini [4] consider household economy as a new field since financial services and products used by households constitute a substantial part of financial industries in first-world countries. Households are today even more directly involved in financial decisions, such as pension funds, streaming accounts and services, or loan markets, which may be of interest to businesses in sales, marketing or banking. In order to detect households, we presuppose that household members communicate with each other more

often than with others. Even though it may seem counterintuitive that family members communicate more using phones since they live together and therefore their communication over the phone would seemingly decrease in quantity, recent research [5], [6] has addressed this issue, stating that mobile phones are regularly used, especially between parents and young people. Hence, we equate cliques to functional households using telecom data and presupposing their mutual communication.

By grouping people into households, such analysis may be used for marketing and sales, especially for telecom providers offering different plans and packages. For example, the services may be offered to one household leader to call rather than spend time on contacting three, four, five or even more household members. In order to do that, we require households to be functional, and not official, and a clique is a distinguishing feature of such households. Cliques outside of a household indicated a closed social network, and various standard social network analyses can be used to identify a household leader [7]. By identifying a household leader, a probability of successful product placement is thus increased.

II. SATISFIABILITY

SAT or the Boolean satisfiability problem is the issue of determining a satisfying assignment of a given Boolean formula. Usually, the formula needs to be in the so-called conjunctive normal form (CNF), in which we deal with a conjunction of one more clauses, where a clause is defined as a disjunction of possibly negated literals. That is, we are dealing with a conjunction of different disjunctions, often used in automated theorem proving. We need to note that the conversion into CNF may lead to exponentially large formulas, thus slowing the automation process. A clause $A_1 \vee A_2 \vee \ldots \vee A_n$ is valid iff there exist i, j such that $A_i = \neg A_j$. A CNF formula $c_1 \wedge c_2 \wedge \ldots c_n$ is valid if each of its clauses is valid.

The Boolean satifiability problem (SAT) was the first known NP-complete problem, proved independently by Cook [8] and Levin [9] in the early 1970's, when the notion of NP-complete problems still had not been established. Cook and Levin had shown that NP problems can be reduced to SAT problems for CNF formulas, which also includes clique-detection reductions [10]. The first SAT solvers were propositional truth tables developed by Charles Sanders Peirce [11] and Ludwig Wittgenstein [12]. The first real SAT algorithm was the Davis-Putnam algorithm [13], and a complete backtracking-based search extension was the Davis–Putnam–Logemann–Loveland (DPLL) algorithm from 1962 [14]. After more than 50 years, DPLL forms the basis of most modern SAT solvers today. Since decision problems inside the NP class have worst-case exponential complexity, efficient SAT solvers are constantly being improved, especially from the early 2000s [15] up to recent years [16].

DPLL algorithm chooses a literal, assigns a truth value to it, and thus by simplifying the formula, it recursively checks if it is satisfiable. If that turns out to be the case, the original formula is satisfiable, otherwise the same procedure is performed with the opposite truth value. If there is a unit clause, we have a unary constraint, and one promising assignment to the corresponding variable. While trying different assignments, the algorithm is building partial solutions, which might prove successful or not. The basic process of DPLL is:

DPLL Algorithm:

1. **Guess a variable.**
2. **Find all unit clauses from the last assignment and assign the needed value.**
3. **Loop step 2 until there is no change.**
4. **If the current assignment does not yield true for every clause, break from the recursion and try a different assignment.**
5. **If it is possible, guess another variable. The algorithm terminates if there is a solution, i.e. all clauses are satisfied, or step 5 is unable to be performed since there is no solution.**

Standard DPLL implementations use CNF files in the widely accepted DIMACS (named after the Center for Discrete Mathematics and Theoretical Computer Science) file format, which defines Boolean expressions stated in conjunctive normal forms. Such input file is a textual file containing comments, problem type and clauses of the parsed formula:

1. The file may begin with comment lines denoted with a lowercase *c*, which can occur throughout the file, but are usually placed in the beginning.
2. The problem line denoted with lowercase *p* consists of the problem type (*cnf* for CNF files) and the number of variables and clauses.
3. The list of clauses is given line by line, where each line is a clause, and clauses are separated by spaces, tabs or newline/carriage-return characters. A variable is defined as *i*, while its negated version is represented as *-i*. The variables are assumed to be numbered from 1 to *n*. The definition of a clause may extend to multiple lines; however, the end of a clause is determined with a final value of 0.

III. IMPLEMENTATION

In order to efficiently analyze reciprocal callers, we want to use hypergraphs in order to better streamline the analysis of functional and official households. A hypergraph is a generalization of a graph, where an edge can connect any number of vertices, that is a hypergraph H is a pair (V, E), where V is a set of vertices/nodes, and E is a family of subsets of V, known as hyperedges. Hypergraphs are usually represented using multi-dimensional arrays, and are often used in improving SAT solvers, with various optimization-methods such as decomposition [17] or [18].

As noted in the introduction, our approach is based on converting a dummy Telco dataset with standard parameters to a hypergraph, and then encoding it to a CNF formula written in the DIMACS file format, and then finally using a satisfiability-solving algorithm to find a

satisfiable assignment of the formula which corresponds to a clique of size *k*, where *k* is a parametrization argument. This in turn means that all *j* up to *k* will be checked in the final algorithm, resulting in *O(nk)*, where *n* is the complexity of the satisfiability algorithm, i.e. polynomially increasing the overall complexity.

The first question is about the dataset format. We will be needing a simple comma-separated values file (CSV) of the form:

numberFrom, IDcallerFrom, numberTo, IDcallerTo

The synthetic dataset we have used was made by randomly generating 50000 entries consisting of 8-digit strings in the numberFrom column, one number at the time with a 0.03 probability of resampling a number which has occured previously. The *numberTo* column entries where generated by randomly sampling from the *numberFrom* entries. The *IDcallerFrom* column was made by assigning random IDs to the numbers, with a 0.08 probability of repeating the same ID for different numbers. The *IDcallerTo* was inferred from the *IDcallerFrom* column.

Additional parameters such as time or date can be included, but we will not be using them in our approach. Whether filtering by additional parameters can improve the overall effectiveness of finding households we leave open for future research.

The conversion of this dataset to a hypergraph is quite simple. A first step is to notice that a conversion to a graph based on *IDcallerFrom* and *IDcallerTo* is trivial. A hypergraph is then obtained by adding all numbers (*numberFrom*) belonging to a *IDcallerFrom* and making a hyperedge between all of them and the *IDcallerTo*. In the same manner the *IDcallerTo* is expanded. The final hyperedge exists encompassing all *numberFrom* and *numberTo* iff there is a call between any pair of them.

We use a hypergraph since we want to be able to equate e.g. 12340001 calls 00001111 and 56782222 calls 00001111 when 12340001 and 56782222 belong to the same *IDcallerFrom*. We are not interested in the direction of the call, but we keep this information since it is the usual way Telco data is presented.

The next step is to encode this hypergraph in a CNF formula. We will use an encoding of the ID-graph (a graph using only *IDcallerFrom* and *IDcallerTo* where two nodes are connected iff there is call between the respective IDs) and then a simple expansion of clauses will create the needed hypergraph encoding. We follow a slight modification of the approach taken in [19] to encode the ID-graph to SAT, while the subsequent expansion to a hypergraph is a simple extension of the original encoding which is, to the best of our knowledge, our small contribution. The encoding of hypergraphs that we formulate is then just a simple generalization of the graph encoding. The idea behind the encoding is that we want a propositional variable $x_{j,m}$ to be true if and only if *m* is the *j*-th node in the clique. As we said earlier, the transformation is parameterized by clique size, and we denote it by *k*. If the final CNF is satisfiable, there is a clique of size *k*, and its nodes are exactly the components that are true. Get such an output, a proper encoding is needed, and two types of constraints are needed:

1. For each *j*, there is a *j*-th node in the clique, i.e. $x_{j,n1} \vee x_{j,n2} \vee \ldots \vee x_{j,nM}$, where nL, $1 =< L =< N$ are all nodes in the ID-graph

2. It is impossible that two nodes *n* and *m* not connected in the ID-graph both belong to the clique: $\neg x_{i,n} \vee \neg x_{j,m}$, for every node *n*, *m*, and every *j*, *n*.

The expansion to the hypergraph is then a simple matter of adding to every clause a copy of these constraints for every *numberFrom* and *numberTo* present in nodes that are connected in the hypergraph.

IV. ANALYSIS

To our knowledge, this is the first application of a SAT solver to telecom data in order to inspect and evaluate reciprocal callers. One possible objection might be that a simple lookup using a dictionary or a database query may seem faster on smaller datasets. However, such telecom data is not suitable for different methods since 1) datasets are big enough so that brute-force method complexity may rise exponentially 2) datasets are extremely dynamic and large, and may not be suitable for relational models. For example, relations per database limits are easily reached, along with maximal relation sizes and columns/rows per table. In case of simple scripting lookups, memory and disk space can be a huge issue, along with building possible dictionaries, while machine-learning methods are generally problematic for "needles in the haystack" data classification. Advances have been made in the cases of credit-card fraud detection, where fradulent activities are rare events, but in such cases there is a strong need of labeled datasets, and contextual information is needed to build accurate profile-based systems, which is often against privacy regulations [20]. Since different datasets and different providers may have different data quality, unbalanced datasets remain an issue in data mining and machine-learning procedures. Therefore, we used a propositional-logic algorithm to efficiently solve NP-complete consequences, in order to acquire a solution suitable for everyday practices and business purposes, hence eliminating unnecessary cost and resources.

V. CONCLUSION

Our goal was to show that artificial-intelligence approach not based on machine-learning may be used for telecom data analysis. An application of a Boolean satisfiability solver to telecom data may yield interesting relations for detecting different kinds of households and their properties: 1) do members of each household actually communicate with each other 2) if so, one can use this finding for business advantage, especially for marketing and sales departments by identifying functional households and their leaders. Our basis was the DPLL algorithm, which can be substituted by more advanced

counterparts, such as CDCL (conflict-driven clause learning) [21], in which backtracking to appropriate decision level is non-chronological. As a final note, this method and analysis may be used for other datasets and clique analysis of different social connections and statuses, in order to gain more insight for business and research.

BIBLIOGRAPHY

[1] S. Skansi, and B. Dropuljić. Identifying Reciprocal Payers in Banking Logs using SAT Solvers. IEEE Conference Publications: 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pp. 1500–1503. 2016.

[2] G. Weiss, and H. Hirsh. Learning to Predict Extremely Rare Events. AAAI Technical Report WS-00-05. 2000.

[3] A. Douik, H. Dahrouj, T. Y. Al-Naffouri, and M.-S. Alouini. A Tutorial on Clique Problems in Communications and Signal Processing. arXiv:1808.07102 [cs.IT]. 2018.

[4] L. Guiso, and P. Sodini. Household Finance: An Emerging Field. Handbook of the Economics of Finance. Elsevier B. V., pp. 1397–1532. 2013.

[5] K. Devitt, and D. Roker. The Role of Mobile Phones in Family Communication. Children & Society, vol. 23, pp. 189–202. 2009.

[6] M. Castells, M. Fernández-Ardèvol, J. Linchuan Quiu, and A. Sey. Mobile Communication and Society: A Global Perspective. The MIT Press. 2007.

[7] M. Zubair Shafiq, M. U. Ilyas, A. X. Liu, and H. Radha. Identifying Leaders and Followers in Online Social Networks. IEEE Journal on Selected Areas in Communications 31(9), pp. 618–628. 2013.

[8] S. A. Cook. The Complexity of Theorem-Proving Procedures. Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, pp. 151–158. 1971.

[9] L Levin. Универсальные задачи перебора. Проблемы передачи информации. 9 (3): pp. 115–116. 1973. [Universal'nye perebornye zadachi/Problemy peredachi informatsii]

[10] N. Dahale, N. S. Chaudhari, and Maya Ingle. CLISAT – Clique Encodings in Implementation of SAT. Mathematical Sciences International Research Journal, vol. 3 (2), pp. 565. 2014.

[11] A. Biere, M. Heule, H. van Maaren, and T. Walsh (eds.). Handbook of Satisfiability. IOS Press. 2009.

[12] Wittgenstein, L. Tractatus Logico-Philosophicus. London: Routledge. 1974 [Original edition 1921].

[13] M. Davis, and H. Putnam. A Computing Procedure for Quantification Theory. Journal of the ACM, vol. 7, no. 3, pp. 201–215. 1960.

[14] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. Communications of the ACM, vol. 5, no. 7, pp. 394–397. 1962.

[15] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, Chaff: Engineering an Efficient SAT Solver. 39th Design Automation Conference. 2001.

[16] T. Toda, and T. Soh. Implementing Efficient All Solutions SAT Solvers. Journal of Experimental Algorithmics 1.12. 2016.

[17] We Li, and P. van Beek, Guiding Real-world SAT Solving with Dynamic Hypergraph Separator Decomposition. 16th IEEE International Conference on Tools with Artificial Intelligence, Boca Raton, FL, USA, 2004, pp. 542–548. 2004.

[18] F. Capelli, A. Durand, and S. Mengel. Hypergraph Acyclicity and Propositional Model Counting. International Conference on Theory and Applications of Satisfiability Testing 2014, pp. 399–414. 2014.

[19] M. Sipser. Introduction to the Theory of Computation. Boston: Thompson Course Technology. 2006.

[20] N. Yousefi, M. Alaghband and I. Garibay. A Comprehensive Survey on Machine Learning Techniques and User Authentication Approaches for Credit Card Fraud Detection. arXiv:1912.02629

[21] J. P. Marques-Silva; K. A. Sakallah. GRASP-A New Search Algorithm for Satisfiability. Digest of IEEE International Conference on Computer-Aided Design (ICCAD). pp. 220–227. 1996.