RESEARCH INSTITUTE FOR SYMBOLIC COMPUTATION

JOHANNES KEPLER UNIVERSITY LINZ

9th International Symposium on Symbolic Computation in Software Science (SCSS 2021), short and work-in-progress papers

Temur Kutsia (editor)

August 2021

RISC Report Series No. 21-16

ISSN: 2791-4267 (online)

Available at `https://doi.org/10.35011/risc.21-16`

Ninth International Symposium on

SYMBOLIC COMPUTATION IN SOFTWARE SCIENCE

SCSS 2021

SHORT AND WORK-IN-PROGRESS PAPERS

Temur Kutsia (editor)

September 8–10, 2021
Research Institute for Symbolic Computation
Johannes Kepler University Linz, Austria

Q-learning and MCTS techniques for improving an algorithm to compute discrete vector fields on finite topological spaces[*]

Julián Cuevas-Rozo
National University of Colombia
University of La Rioja, Spain
jucuevas@unirioja.es

Jose Divasón
University of La Rioja, Spain
jose.divason@unirioja.es

Laureano Lambán
University of La Rioja, Spain
lalamban@unirioja.es

Ana Romero
University of La Rioja, Spain
ana.romero@unirioja.es

Abstract. In this work we present an ongoing project on the improving of a previous symbolic computation algorithm computing discrete vector fields on finite topological spaces. To this aim, we consider different strategies to choose each one of the possible vectors at each step of the algorithm and we apply some reinforcement learning techniques.

1 Introduction

A *finite topological space* (or simply a *finite space*) is a topological space with finitely many points. Finite spaces and finite preordered sets are basically the same objects considered from different perspectives [1]. The study of homotopical invariants can be restricted to the class of finite T_0-spaces, since any finite topological space is known to be homotopy equivalent to a finite T_0-space [17]. Finite T_0-spaces correspond to finite partially ordered sets or posets. Given a finite T_0-space X, the incidence matrix corresponding to the order relation of its associated poset and the adjacency matrix of the Hasse diagram $\mathscr{H}(X)$ are called *topogenous matrix* and *Stong matrix* of the space, respectively.

In [6], some algorithms and programs to compute topological invariants of finite spaces have been presented, which are based on new developed constructive versions of theoretical results [4, 11]. In particular, a new module for the Kenzo symbolic computation system [7] was developed allowing the computation of homology groups of h-regular spaces without constructing the order complex associated with the poset, by defining a chain complex directly from the finite space. Moreover, our new Kenzo module includes an algorithm to determine homologically admissible Morse matchings on finite spaces, in order to use the Minian's version of discrete Morse theory. A *Morse matching* M on a finite space X is a set of edges of its Hasse diagram where no edges share a common vertex and such that the directed graph $\mathscr{H}(X)$, modified by reversing the orientation of the edges in M, is acyclic. If a Morse matching satisfies the property of being *homologically admissible* (see [11] for details), the homology of X can be determined by means of a chain complex smaller than the canonical one associated to the finite space generated by the *critical* points of the matching, which are the elements of X that are not incident to any edge in M. A Morse matching on a finite space can also be seen as a *discrete vector field* [14] on the associated chain complex. Then, the longer the Morse matching is, the smaller the number of generators we need to describe a chain complex to compute homology by means of the critical complex is.

[*]Partially supported by the Spanish Ministry of Science, Innovation and Universities, projects MTM2017-88804-P and PID2020-116641GB-I00.

In this work, we try to improve the algorithms presented in [6] by studying different strategies and methods to maximize the size of a discrete vector field. To this aim, we first consider 25 different strategies to choose which vector should be added in each step of the Morse matching algorithm. We also try some reinforcement learning techniques such as Q-learning and MCTS. The code and the experiments of our work can be found at:

`https://github.com/jodivaso/Q_Learning_MCTS_DVF`

2 Strategies for computing discrete vector fields on finite spaces

In our algorithm for computing Morse matchings on finite topological spaces presented in [6], we search for homologically admissible edges such that the set of such edges does not contain cycles in the modified Hasse diagram of the poset associated to the finite space. In this searching, we go through the columns and the rows of the Stong matrix in ascending order (an edge in the Morse matching, or a vector in the associated chain complex, corresponds to an element equal to 1 in the Stong matrix). The algorithm finishes when it is not possible to add a new edge to the matching satisfying the desired property, so that the corresponding discrete vector field is maximal. However, a different vector field could exist with a bigger size (the problem of finding a vector field of maximal length is difficult for big spaces).

In order to improve our algorithm and try to find Morse matchings on a finite space as big as possible, we have decided to consider different *strategies* to sort rows and columns of the Stong matrix, and compare them analyzing if there exist remarkable differences between the size of the discrete vector fields obtained in each case. We have considered five different strategies to sort a list of elements in a finite T_0-space: *:standard* (in ascending order, the one used in our initial implementation of the algorithm), *:indegree* (sorted by the number of “head ends” adjacent to each vertex), *:reverse-indegree* (reverse order to the previous one), *:outdegree* (sorted by the number of “tail ends” adjacent to a vertex), and *:reverse-outdegree* (reverse order to the previous one).

The strategies considered above have been tested in random finite h-regular spaces of different sizes between 10 and 100 elements. On each one of these spaces, we have computed 25 discrete vector fields in Kenzo by using the five strategies in order to sort the column and row indexes of the Stong matrices. In these experiments we have observed that the strategy *:outdegree - :reverse-outdegree* (that is, considering outdegree order for column indexes and reverse-outdegree for row indexes) predominates over the others, obtaining the maximum of all strategies in 81.5% of all spaces. The second best strategy is *:outdegree - :indegree* (maximum in 68% of all cases) and the third one is *:outdegree - :standard* (65%). We can also observe that there is not a unique strategy which performs well in all cases and, moreover, in most of the finite spaces we have considered, the different strategies have produced discrete vector fields with low difference between their lengths.

3 Application of Q-learning algorithm

After trying the different strategies to compute discrete vector fields on finite topological spaces introduced in the previous section and seeing that there is not a unique strategy which performs the best in all spaces, we have tried to improve our symbolic computation algorithm by means of *machine learning* methods. Due to the type of problem we are studying, we have focused on the so-called *reinforcement learning*. Machine learning methods have already been applied in other computer algebra problems [3, 13, 16].

Reinforcement learning is a type of machine learning technique where the key point of is to establish a suitable reward system. That is, when the process has reached a concrete state and it has to select a possible action in order to continue, a reward is applied, which depends on the profit obtained after the corresponding change of state. In particular, we use the *Q-learning* algorithm [18], which is a particular case of reinforcement learning. This algorithm requires the definition of *states* (the possible situations which can be encountered), *actions* (transitions from one state to another state) and (positive or negative) *rewards* received after each transition. More concretely, the Q-learning algorithm is based on the construction of a *Q-table*, which is a matrix where we have a row for every state and a column for every action. It is first initialized to 0, and then values are updated after training, taking into account the rewards obtained. The process of *training* is done by iterating a sequence of actions starting from the initial state, combining *exploration* (choosing a random action) and *exploitation* (choosing actions based on already learned Q-values). For each state (each row in the table), the best action is supposed to be the one with the highest value in the corresponding column of the Q-table. See [18] for more details.

In our problem, we have chosen to use the Python library for reinforcement learning *Gym* [12]. We construct a new class called `DVFMatrixEnv` which implements the interface `gym.Env`, which represents a learning `environment`. This interface requires the definition of a set of `states` and a set of `actions`, and the implementation of four methods: `init`, `reset`, `render` and `step`. The most important method is `step`, which, for an input pair of state and action, returns a new state, a reward and a boolean variable called `done` which determines if the problem ends or not after applying the action.

The input data to construct an environment of type `DVFMatrixEnv` is the Stong matrix of the finite T_0-space where we want to compute a discrete vector field. The elements of the set of actions are the possible vectors we can select in the space. Then, a state represents a set of possible vectors (some of the sets are not valid, but this will be specified in the algorithm by means of a negative reward). The number of possible states is $2^{\text{numberOfActions}}$. Both actions and states are represented as numbers. The initial state is 0. Once we have defined our environment, we perform the training of the Q-learning algorithm with a given number of repetitions. After a high number of iterations, the Q-table allows one to define a good process to obtain a Morse matching as big as possible.

We have tested the method on some of the random examples studied in the previous section. In all the cases where the algorithm works, the Q-learning technique has provided us satisfactory results. That is, for each space, the vector field we obtained by using this approach has the same number of vectors as the best one obtained by applying the strategies given in Section 2. Nevertheless, the size of the Q-table results to be an evident problem to be treated. In fact, we have only tested satisfactorily the Q-learning technique in 13 spaces; for other spaces, the Q-table cannot be built. It is a common drawback of this type of machine learning and there exist some well-studied alternatives to avoid it, such as the one studied in the next section.

4 Monte Carlo tree search

Monte Carlo tree search (MCTS) is a heuristic search algorithm for some kinds of decision processes. It is commonly used in the field of game theory as a method to solve a game tree by analysing the most promising movements. In fact, MCTS has proved to be very competitive in deterministic games with large branching factors for many years. One of the most famous applications is the AlphaGo program, in which a neural network combined with MCTS beat a professional human Go player in 2015.

In our problem, we can see the different possibilities for selecting the vectors as a search tree: each node is a 1 of the Stong matrix that can be chosen in a concrete step; branches correspond to the sequence of vectors that have been selected. The final goal is to get a branch as large as possible.

Essentially, MCTS is a smart search, since in each step it accumulates value estimates to guide towards highly rewarding trajectories in the search tree. In other words, MCTS focuses on nodes that are more promising, avoiding applying brute force algorithms (like minimax), which are unfeasible over big state spaces. More concretely, each MCTS iteration consists of four steps: selection (choose a promising node p that can be expanded), expansion (create one or several child nodes of p, one for each possible action at that point), simulation (choose one of those child nodes and apply random decisions until finish) and backpropagation (use the result of the simulation to update the information of the tree).

We have implemented a MCTS version based on *Gym*. To balance exploration *vs.* explotation, in the selection phase we employed the UCB formula (Upper Confidence Bound, see [2, 9]): $v_i + c\sqrt{\log(N)/n_i}$, where v_i is the value estimate of the node, N the number of visits of the parent node, n_i the number of visits of the node and c is the exploration hyperparameter, which theoretically is equal to $\sqrt{2}$ but can be empirically chosen depending on the problem to strengthen exploration or explotation. That is, the UCB-value of a node decreases in the number of visits (exploration) and increases in node's value (exploit). Thus, UCB permits to *sort* the nodes: nodes with a high UCB-value are more promising or need to be explored. In order to avoid prioritizing nodes that have never been visited (otherwise, a non-visited node has ∞ as UCB-value and will always have higher priority over an already visited one), we decide to give parent's UCB-value to a non-visited node. The simulation step consists of randomly choosing vectors until reaching a state in which no more admissible vectors can be added. The reward is the number of vectors that we reached (the depth of the branch). That information is backpropagated to the tree and the UCB-values of each node are updated. MCTS permits to handle big spaces, since there is no need to create a Q-table. It can give a result immediately and indeed, MCTS improves its result with more time. It is a decision process which applies random actions over the most promising nodes, and the information obtained from the simulation is used to update the list of the most promising nodes (based on the UCB-value). At the end, the best decision is the one with higher reward, in our case, the largest branch.

The algorithm essentially requires as input the Stong matrix of the finite space and the number of MCTS iterations (also named as playouts) that one wants to perform. Our first experiments show that MCTS obtains the same or more vectors than applying any of the 25 strategies and the Q-learning algorithm in 80% of all spaces, with no need to explore the whole tree and performing better than a pure iterated random strategy. Concretely, by using MCTS we obtain the same number of vectors in 42% of cases and more vectors in 37.5% of spaces.

5 Conclusions and further work

This work presents some improvements that we carried out over a symbolic computation algorithm for computing Morse matchings on finite topological spaces, concretely on how to select vectors to maximize the size of a discrete vector field. We first studied 25 different strategies to choose which vector should be added. We also tried techniques that are usually employed in reinforcement learning contexts, such as Q-learning and MCTS. The initial experiments show promising results. We plan to extend this work by employing deep Q-learning, which uses neural networks to approximate the Q-table, and also combine it with MCTS. In addition, further work is necessary to optimize the implementation of the algorithms (for instance, to avoid recalculations and to exploit parallelism) and tune appropriately the parameters, hyperparameters and rewards to get a fast and good result.

References

[1] P. Alexandroff (1937): *Diskrete Rume*. Mat. Sb. (N.S.) 2, pp. 501–518.

[2] Peter Auer, Nicolo Cesa-Bianchi & Paul Fischer (2002): *Finite-time analysis of the multiarmed bandit problem*. Machine learning 47(2), pp. 235–256, doi:10.1023/A:1013689704352.

[3] Y. Bengio, A. Lodi & A. Prouvost (2020): *Machine Learning for Combinatorial Optimization: a Methodological Tour dHorizon*. `https://arxiv.org/pdf/1811.06128.pdf`.

[4] N. Cianci & M. Ottina (2017): *A new spectral sequence for homology of posets*. Topology and its Applications 217, pp. 1–19, doi:10.1016/j.topol.2016.12.001.

[5] J. Cuevas-Rozo (2020): *Finite topological spaces in Kenzo*. `https://github.com/jcuevas-rozo/finite-topological-spaces`.

[6] J. Cuevas-Rozo, L. Lambn, A. Romero & H. Sarria (2020): *Effective homological computations on finite topological spaces*. Applicable Algebra in Engineering, Communication and Computing, doi:10.1007/s00200-020-00462-8. In press.

[7] X. Dousson, J. Rubio, F. Sergeraert & Y. Siret (1999): *The Kenzo program*. Institut Fourier, Grenoble. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.

[8] A. I. Khan & S. Al-Habsi (2020): *Machine Learning in Computer Vision*. Procedia Computer Science 167, pp. 1444–1451, doi:10.1016/j.procs.2020.03.355.

[9] Levente Kocsis & Csaba Szepesvári (2006): *Bandit Based Monte-Carlo Planning*. In Johannes Fürnkranz, Tobias Scheffer & Myra Spiliopoulou, editors: Machine Learning: ECML 2006, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 282–293, doi:10.1007/11871842_29.

[10] A. Maier, C. Syben, T. Lasser & C. Riess (2019): *A gentle introduction to deep learning in medical image processing*. Zeitschrift fr Medizinische Physik 29(2), pp. 86–101, doi:10.1016/j.zemedi.2018.12.003.

[11] G. Minian (2012): *Some remarks on Morse theory for posets, homological Morse theory and finite manifolds*. Topology and its Applications 159(12), pp. 2860–2869, doi:10.1016/j.topol.2012.05.027.

[12] OpenAI (2016): *Gym*. `https://gym.openai.com/`.

[13] D. Peifer, M. Stillman & D. Halpern-Leistner (2020): *Learning Selection Strategies in Buchberger's Algorithm*. `https://arxiv.org/pdf/2005.01917.pdf`.

[14] A. Romero & F. Sergeraert (2010): *Discrete Vector Fields and Fundamental Algebraic Topology*. `https://arxiv.org/pdf/1005.5685.pdf`.

[15] D. Silver, A. Huang & C. Maddison (2016): *Mastering the game of Go with deep neural networks and tree search*. Nature 529, pp. 484–489, doi:10.1038/nature16961.

[16] L. R. Silverstein (2019): *Probability and Machine Learning in Combinatorial Commutative Algebra*. Ph.D. thesis, University of California Davis.

[17] R. E. Stong (1966): *Finite topological spaces*. Trans. Amer. Math. Soc. 123(2), pp. 325–340, doi:10.1090/S0002-9947-1966-0195042-2.

[18] C. J. C. H. Watkins (1989): *Learning from Delayed Rewards*. Ph.D. thesis, Cambridge University.